Cheminformatics OLCC (2019)

The LibreTexts libraries are Powered by NICE CXOne and are supported by the Department of Education Open Textbook Pilot Project, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

This text was compiled on 03/17/2025

TABLE OF CONTENTS

Licensing

*A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.*

CHAPTER OVERVIEW

1: Introduction

Hypothes.is Tag= f19OLCCc1
Note: Any annotation tagged **f19OLCCc1** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

Greetings, welcome to the homepage of the Fall 2019 Cheminformatics OLCC. This course is sponsored by the ACS Division of Chemical Education's Committee on Computers in Chemical Education. This course is designed for either graduate students, or upper division undergraduate students. In this course students will lean how molecules are represented on computers, and use PubChem to learn some basic coding to access information through a variety of APIs.

We have a draft syllabus and please contact Bob Belford (rebelford@ualr.edu) if you are interested in learning more

1.1: Introduction

What is cheminformatics?

Modern cheminformatics evolved out of the "drug discovery" needs of the pharmaceutical industry. The term was originally coined as "chemoinformatics" by Frank Brown of the R.W. Johnson Pharmaceutical Research Institute in his 1998 manuscript, "Chemoinformatics: What is it and How does it Impact Drug Discovery". The term "chemoinformatics" still tends to be preferred in Europe today, we will use "cheminformatics" in this class, as that terminology aligns with the premier open-access journal of the field, the Journal of Cheminformatics.

Although much of the material in this class will be of value to students of pharmacology, we are taking a much broader perspective of the field and treating cheminformatic's skills as essential to one of the primary paradigms of science. The informatics, data representational and analytics skills learned in this class would be of value to a wide variety of tasks in the pursuit of knowledge in the chemical sciences, and this course is of value to students beyond those of the pharmaceutical sciences. In fact, we can go a step further and state that cheminformatics is changing the fundamental cognitive artifacts used to represent, manipulate and communicate chemical information, and in a world of instant access to interconnected digital data, a fundamental understanding of cheminformatics is an essential skill for tomorrow's practicing chemist.

Paradigms of Science

Cheminformatics can be considered to be a "fourth paradigm science" in the context of the 2009 Microsoft Research book published in honor of the late Jim Gray, "The Fourth Paradigm: Data-Intensive Scientific Discovery." In the forward of this book is a transcript based on Jim's last talk; "eScience: A Transformed Scientific Method", where he describes the four paradigms of science. The following paradigms of science are based on figure 1* of this transcript:

> **First Paradigm: Empirical Science** (thousands of years old)
> - the experimental chemist (scientist) making measurements and observations of the physical universe and generating empirical data.
>
> **Second Paradigm: Theoretical Science** (centuries old)
> - the theoretical chemist (scientist) defining complex mathematical relationships that underpin natural observations.
>
> **Third Paradigm: Computational Science** (decades old)
> -the computational chemist (scientist) using computing machines to perform complex calculations to predict behavior and generate computational data.
>
> **Fourth Paradigm: Data Exploration** (emerging)
> -the eScientist using computing machines to discover complex relationships across datasets of both empirical and computational data.

The fourth paradigm actually depends on the other paradigms and requires the ability to acquire, manipulate and understand data. This class will introduce you to a variety of open source software programs and public compound databases, with a teaching focus on PubChem. But the skills will allow you to pursue data exploration with other data sets and resources. Since we are bringing in resources across the web, we will use a web annotation service, Hypothes.is, to connect those resources to the chapter discussions.

Hypothes.is Web Annotations

This is a collaboratively taught intercollegiate course and will use the Hypothes.is Web Annotation Service (https://web.hypothes.is/) to discuss the content in a webpage overlay. Students need to create an account at Hypothes.is, and then join the class discussion group through a link that their instructor will send them. Please review your syllabus before creating your account, as your instructor may include naming protocols for students in your class. You probably should also install the browser plugin when you create your account, as that will allow you to annotate pages external to the LibreText HyperLibrary.

There are two types of annotations faculty and students in this class will will make, those intrinsic to a page being discusses, and those extrinsic to the page.

Two types of Annotations

1. **Intrinsic Annotations:** You simply highlight content intrinsic to a page within the LibreText hyperlibrary, choose your class discussion group and annotate. After you save your annotation it will automatically appear in the overlay of that page. This is

possible because hypothes.is is integrated into LibreText, and note through the WYSIWYG editor you can format your annotations, add images, videos and the like.

2. **Extrinsic Annotations:** For content external to a page of the LibreText HyperLibrary you need to tag the annotation so it can be displayed on a page within the HyperLibrary. The table of contents of each chapter will be used for this purpose, and have a specific tag that you use for that chapter. When an annotation is made on any open access webpage on the web, and tagged with that chapter's tag, the annotation will appear at the bottom of that chapter's table of contents, and include a contextual link to the annotation. If the page being annotated is not part of the LibreText HyperLibrary you may need to install a browser plugin to be able to annotate it (which will be neccessary if hypothes.is is not integrated into the webpage). To install the plugin go to https://web.hypothes.is/start/.

Annotation Features

1. **Annotation Overlay:** Unlike web 2.0 comment features where people discuss an article at the bottom of a webpage, Hypothes.is uses an "overlay" on a web page that can be activated by clicking the arrow on the upper right corner of the webpage. If you click on an annotation in the overlay, the page scrolls down to the actual highlighted text. This is sort of like commenting on a piece of note-paper attached to a webpage, instead of commenting on the webpage itself, and you have to activate the overlay to see the annotation.
2. **Contextual Links:** A contextual link is a link to target text within a page. When you click a contextual link, it opens the page in a new browser, activates the overlay, and scrolls down to the targeted text. Technically, these are called direct links that combine a webpages URL with a selector that refers to specific text within the page, the target text of the contextual link.
3. **Groups:** Hypothes.is allows you to make comments and tags that can have either public or private group access. Only members of a group can see group annotations, and we will use a group that includes faculty and students from multiple campuses. Your instructor will provide you a link in your class syllabus that will allow you to join the group. Your instructor may also provide you with a unique username that "hides" your identity from everyone except your instructor, and provide instructions on how to create a Google email account for this class. This will not only allow your instructor to quickly identify students in your class, as compared to other classes that are involved with the course, but also allow you to create your own hypothes.is account that you can use outside of the class.
4. **Tags:** Learning how to tag annotations is an important ability, as it not only allows you to search and sort your tags, but also to connect tags to different web objects. So if you find 5 passages of text dealing with molecular fingerprints, you cold tag them, and then search them from your homepage (which automatically lists your tags), and have instant access to them. Your username is also a tag, and so if someone else had used the tag "fingerprint", you could find their target text, of by combining the tag with your username tag, filter the query to just the items you had tagged.
5. **Replies:** Within the overlay are discussions. As a student you will get an email if someone replies to a question you have, but also, through the overlay you can look at all the other questions dealing with chapter, and the discussions that evolved out of them. Our vision is that the overlay becomes a layer to the textbook where students across multiple campuses can learn from the questions and answer discussions of others.

Note on Annotations

It is interesting to note that annotations were part of Tim Berners-Lee's original 1989 proposal for the World Wide Web, and were integrated into the prerelease version of the 1994 NCSA graphical Mosaic Web Browser, and yet today, 30 years after Tim Bererns-Lee's original proposal, few faculty and students use them. In this class you will be expected to use web annotations to connect open-access cheminformatics resources across the web to the discussions of your class topics.

The W3C Web Annotation Working Group has an excellent interactive image describing Web Annotation Architecture that you might enjoy walking through.

What is PubChem?

A brief description of PubChem is warranted here, no more than a few small paragraphs. We should also state that there will be a chapter on other resources, but this class will focus on PubChem with respect to training students how to access data.

What is Programmatic Access?

A brief comparison of a GUI/webpage and an API. The goal here is to put the foundations down for training in programmatic access through PubChem, but the skills can be used with any database.

Non-Open Access Resources

This course is not attempting to provide a comprehensive coverage of contemporary cheminformatics resources, but to train students in the skill sets associated with data exploration in the chemical sciences. This is an intercollegiate course open to all schools, and although some will have access to content like ACS's SciFinder, STN and Elsevier's Reaxys, others, especially Primary Undergraduate Institutions, which often do not have the graduate level research needs to support those technologies, will not. None-the-less, the skills students learn in this class should assist them in utilizing those resources in their future endeavors.

In a similar vein, each chapter will provide a bibliography including suggested reading material, that will be delineated into open access and restricted access publications of the primary literature. But required reading assignment will be limited to open access publications. This is for for several reasons

1. Only open access content can be connected to the textbook discussions through Hypothes.is.
2. We can not expect our students to pay the exorbitant fees that publishers charge for access to single articles
3. This is an Open Education Resource (OER), which we expect others to use once the course is over, and to be of value they must have access to the content.

We regrettably recognize that in making the decision to limit this course to open access content that there will be a substantial amount of high-quality cheminformatics material that will not be available to our students. We believe that access to a quality education, which is one of the 17 United Nation's Sustainable Development Goals for 2030, is a fundamental human right, and that the content of this course needs to be available to all.

Contributors

Robert E. Belford, UA Little Rock

*This contextual link uses the https://web.hypothes.is/ annotation service to take you to the part of a PDF on the Fourth Paradigm that Microsoft Uploaded to the web. This is a public link and should be viewable to anyone on the web, but if you are in the class you will need to create an hypothes.is account and join the class group as outlined in your syllabus, as otherwise you will not be able to see or participate in the class discussions.

1.2: Brief History of Cheminformatics

when are you going to finish the database sections?

Introduction

There are multiple surveys on the history of cheminformatics. As these are owned by publishing companies we can not share them online. See you instructor if you do not have access, as we do have the ability to share them within our class rooms.

Restricted Access Sources

1. Cheminformatics: a history, by Peter Willet. *WIREs Comput Mol Sci* 2011, 1: 46-56. doi: 10.1002/wcms.1
    - 11 pages with 91 references published in 2011.
        - Abstract: This paper gives a brief history of the development of chemoinformatics since the first studies in the late 1950s and early 1960s of methods for searching databases of chemical molecules and for predicting their biological and chemical properties. Topics, and associated key papers, that are discussed include: structure, substructure, and similarity searching; the processing of generic chemical structures and of chemical reactions; chemical expert systems; the identification of qualitative and quantitative structure–activity relationships in both two and three dimensions; pharmacophore analysis; ligand–protein docking; molecular diversity analysis; and drug-likeness studies. Brief mention is also made of other important areas such as computer-assisted synthesis design and computer-assisted structure elucidation.
2. Chemoinformatics - An Introdution for Computer Scientists, by Nathan Brown,ACM Comput. Surv.41, 2, Article 8 (February 2009), DOI: 10.1145/1459352.1459353
    - 28 pages, 75 references, published in
        - Excellent introdctory article, sections 1.4 (Origins of Chemoinformatics) and 2 (Chemistry and Graph Theory) are interesting reads.
3. Chemoinformatics: Past, Present, and Future, by William Lingran Chen. *Journal of Chemical Information and Modeling* **2006** *46* (6), 2230-2255. DOI: 10.1021/ci060016u
    - 26 pages with 332 references, published in 2006.
        - Abstract: The history of chemoinformatics is reviewed in a decade-by-decade manner from the 1940s to the present. The focus is placed on four traditional research areas: chemical database systems, computer-assisted structure elucidation systems, computer-assisted synthesis design systems, and 3D structure builders. Considering the fact that computer technology has been one of the major driving forces of the development of chemoinformatics, each section will start from a brief description of the new advances in computer technology of each decade. The summary and future prospects are given in the last section.

1.3: Introduction to Data and Databases

Introduction

What is data?

To the chemist data are the measured or counted values that can be collected or produced to understand relationships of observable or computed phenomena that are germane to the practice of science (both empirical and computational). To the chemist there are different types of data that are defined by how the data is generated, like the mass or temperature of a sample, or the spectra of a compound. This data is often stored on a computer in a file or database, and can be subsequently processed through various software programs.

To the computer scientist or software program data has a different meaning in that there are different data types that represent how the computer stores information. That is, a computer does not store a measured phenomena like the temperature of a sample, but a digital data type, a representation of the temperature that a software agent can interact with. For example, a letter of the alphabet would be a different type of data than a number, because you can not do arithmetic calculations on letters like you do on numbers.

Cheminformaticians need to understand both meanings of the concept of data, and in this section we will introduce how computers store data, and the different types of data from the perspective of programming and software agents. Then we will move onto data in the chemistry sense of the word.

What is a database?

Databases are a way computers store information in a manner that can be retrieved. You use databases all the time. Do you realize that as you read this web page you are using a database? Yes, this web page is not a digital file like a MS Word document that saves the information like a sheet of paper, but instead the web browser is displaying information that was pulled from a database as the page is loaded. That is, LibreText is a Wiki that is hosted on the MindTouch knowledge management platform and the information you see is drawn from a database when the page is loaded. Webpages that are pulled from databases are often called dynamic web content, and those that are files are called static web content. Of course databases can store different types of information, and this class will be using databases that store information related to chemical compounds. But it is important to realize that the use of databases in the twentieth century are pervasive, and you are actually using a database right now, as you read this webpage.

How do databases store information?

Databases store data, which is the representation of information through a binary code that computing machines can read. A bit is the smallest binary value with two possibilities, 0 or 1. This data needs to be stored on a physical medium so the machine can read it. In the old days data was stored on punch cards (figure 1.3.1), which allowed for a binary representation of each position, which could be either punched or not punched (bitten or not bitten). If each location of memory is allowed a certain number of bits, then you can generate different combinations, and give those different combinations different meanings.

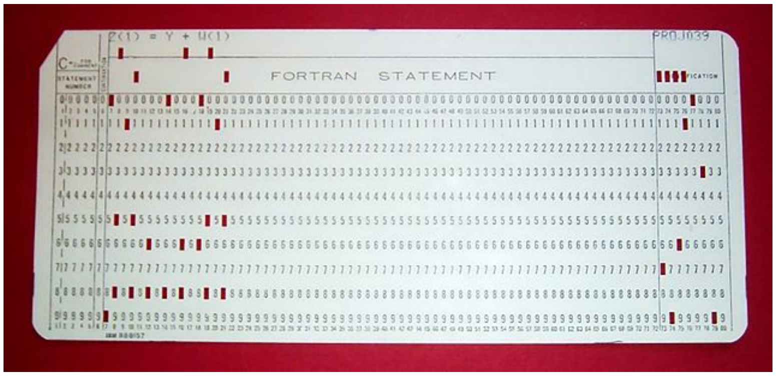


Figure 1.3.1: Old Fortran punch card, one of the earliest computer based means for storing data (image credit: By Arnold Reinhold - CC BY-SA 2.5, https://commons.wikimedia.org/w/inde...p?curid=775153)

A quick look at these possibilities shows that n bits gives 2^n possible combinations.

- 1 bit has two (2^1)possibilities : 0 or 1, and so can represent two different things
- 2 bits has four (2^2) possibilities: 00, 01, 10, or 11, and so can represent four different things

- 3 bits has 8 (2^3) possibilities: 001, 010, 100, 011, 101, 101, 110, 111, and so can represent 8 different things.
- 8 bits has 8 (2^8) possibilities, which is 256, ranging from 00000000 to 11111111, and we won't write them all down here.
- n bits has (2^n)

A byte of data is defined as 8 bits and so has (2^8) or 128 values (which run for 0 to 127). In the early days of computers 8 bit chip of memory was common and the American Standard Code for Information Interchange (ASCII) was developed and based on the 8 bit byte, which shows one set of code that allows computers to interact with a keyboard to store information. Note the first 32 ASCII characters are unprintable codes used to control devices, and the remaining 156 characters are used to store symbols like numbers and the letters of the alphabet. The full list can be found at https://www.ascii-code.com/ and here are some examples. It should be noted that there are other codes besides the ASCII codes.

binary byte	meaning		binary byte	meaning
00100001	!		00000000	null (not printable)
00100101	%		00000010	start of text
00110001	1		00000011	end of text
01000001	A		00001101	carriage return
01100001	a		11111111	ÿ

The take home message here is that everything is stored on the computer in the form of a binary bit, be it a text document, picture, molecular structure data or a spectral file. Each of these represent a different data type and so when you interact with the database, you need to know what type of data is stored, and then use software that can "read" that type of data. Likewise, if you write some simple script to interact with data, you need to recognize the data type you are interacting with, for example, you can do math with numbers, but not letters, and so a number needs to be a different data type than a letter.

Today we do not use punch cards but still store data as a binary representation on a physical device that can be electronically read, like magnetic tape, hard drives, flash drives, SSD (Solid State Disk) and the like. The way magnetic based storage devices work is through the North-South alignment of the magnetic field, where one of these (N-S) would be given the value of 1, and the other (S-N) would be the 0. If you are interested in learning how a hard drive works there is a real good 6 minute video on Nick Parlante's computer science page from Stanford. Flash drives and SSDs have no moving parts and are not based on magnetism, but represent ones and zeros by the ability of tiny channels (gates) within a transistor to be able to conduct (1) or not conduct (0) electricity. It should be noted that after 10-20 years flash drives can lose their memory. In fact, surprisingly magnetic tape is the longest lasting digital storage, although it is the slowest to use.

Data Types

There are multiple data types and when a programming language communicates with a computer it must specify the data type so that it can be properly handled. In fact different programming languages may handle data types differently. When you input data this is often done through a data field, and the data field will specify the type of data. We will look at a some of the very basics here. We are introducing these here because if you define a variable, you need to define what type of data it is. For example, a word is different than a number, and so if you define a variable, the software needs to know if it is a number or a letter. In this class your school will use either R or Python, which may handle data type definitions slightly differently, but in the end, they are doing the same things. We will now look at several different types of data. Note, in both R and Python you can change a variable data type after you first define it, while in many other programs you can not.

Numeric Types

These can be used in calculations. There are two basic types of numbers that we need to be aware of, integers and floating numbers. This is important because the way computers store these data types is different. If you think back to your general chemistry, there were exact numbers and measured numbers. Exact numbers were integers and measured numbers had precision, and used a decimal point.

Integers

Integers are exact numbers and do not have a decimal point. Integers are stored directly as binary values, but the first bit is used to indicate the sign of the number (plus or minus). So a 32 bit (4 bytes) chip could represent 2^{31} different positive or negative integers,

Floating point

Floating point numbers are stored like scientific notation, where part of the bit represents the mantissa, which is the number being multiplied by 10 to a power (and represents the precision), and another part of the bit represents the exponent.

Character Types

These are alphabetical characters. They are often called a string literal and when defined in code need to be placed into double quotations.

Other Types

There are sort of two other types of data. The first are data types used in programming, like time and Boolean logic data types. The second are files that are used to store data, and not used in programming.

Date & Time

There are a variety of date and time formats and these are actual data types.

Boolean (Logic)

Boolean data are data types that have two possible values, true or false. These can be used with Boolean logic operators like AND, OR and NOT, along with comparative operators, like equals, not equal, less than, less than or equal, greater than and greater than or equal. These are very important in programming because they allow computers to make logical decisions.

Specific file types

You can also store files in a database. These can be image files, pdf files, chemical structural data files, spectral files, etc.

Databases

In the old days of punch cards one would physically store the card deck in a filing cabinet and if you wanted to perform a calculation on the data you would physically have to retrieve the cards and load them into submit it to the computing machine.

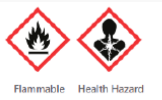
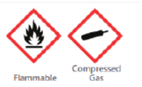
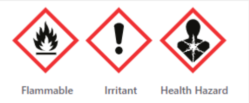
Figure 1.3.2: March 21, 1957 image of people working on an IBM type 704 electronic data processing machine at Langley Research Center (Image credit: NASA)

Types of Databases

Flat-file Database

A flat file database is the simplest type of database and consists of a data table where the columns are fields and the rows are records. So in the following table, each row has the record of a chemical, which has the data fields that describe attributes of that chemical like its name, number of atoms, melting point, GHS pictogram and molecular formula. A flat file database is essentially of the same structure as a data table in a book like the CRC Handbook on Chemistry and Physics, except that you can search it like a webpage

It is important to note that each field is of a specific data type. The name and molecular formula are string literals (a string of characters), the number of atoms are integers, the melting points are floating point numbers and the pictograms are image files. When you create a field in a database you must identify the type of data stored in it.

Name	Number of atoms	melting point (^{o}C)	GHS pictogram	molecular formula
n-butane	14	-138.2		C_4H_{10}
Isobutane	14	-138.3		C_4H_{10}
benzene	12	5.5		C_6H_6

Table$PageIndex2$: The structure of a flat-file database. What is important to see here is that each record has different fields associated with it, and those fields need to identify the type of data they contain (you can not upload an image file to an integer field).

There are some shortcomings to the flat-file database, in that values may not be unique, that is, isomers like n-butane and isobutane would have the same number of atoms and molecular formulas, or that a name may have synonyms, and you may have searched that above file for 2-methypropane and missed it. Of course you could have a new record for each synonym, but that would be very inefficient.

Relational-Database

These are the most common types of databases used online. A relational database is like a table with an index number for each record, and you correlate the index number instead of the field value.

n	Name		na	Number of atoms		mp	melting point (^{o}C)		gp	GHS pictogram		mf	molecular formula
n1	n-butane		na1	14		mp1	-138.2		gp1			mf1	C_4H_{10}
n2	Isobutane		na2	14		mp2	-138.3		gp2			mf2	C_4H_{10}
n3	benzene		na3	12		mp3	5.5		gp3			mf3	C_6H_6

Table$PageIndex3$: Making 5 relational tables of the data in table 1.3.2, each with its own unique index number.

We now treat the flat-file data table as 5 different tables, each with a different index number (n,na,mp,gp & mf), and set up a relationship so the record of the first chemical is not defined by the value within the field, but by its index value. So the first relationship is identified by the index values of n1, na1, mp1, gp1 & mf1, which relate to their respective field values (the record of row 1 in table 1.3.2). So for example, the number of atoms in n-butane and isobutane are defined by na1 and na2, not 14 and 14, ie., we are relating different things. Also, if we wanted to include all the synonyms, we only have to include their index value, and when we show the record, all of them show.

1.4: Installing Python

What is Python?

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed." from Python.org executive summary.

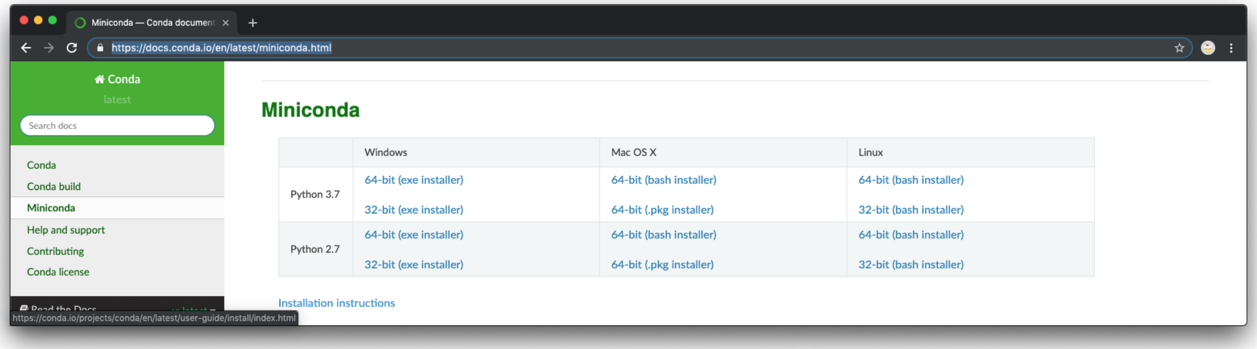
There are two versions of Python that are available: 2 and 3. Python 2 will no longer be maintained after January 1, 2020, so we will be focusing on Python 3.

Installing Python

Python does not come preinstalled on all computers (Python is native in Raspberry Pi, and Mac OS does come with Python 2.7) There a few ways to install python on your computer, but will use the miniconda respository management system. This is subset of the anaconda distribution system for Python and R data science programming languages. There are some differences between the anaconda and minconda installs, but the main advantage of miniconda is that we will have more control of what we add, and doesn't take up as much disk space. The conda system allows for creating multiple environments so that you can test out different packages or try new packages that may conflict with working installations.

There will be a few differences for installing on Windows, Mac OS X and Linux. Most students in this course will be using Windows or Macs, so we will focus explanations here.

1. Go to https://docs.conda.io/en/latest/miniconda.html and download the latest Python 3 installer. As of the time of this writing, 3.7 was the release. Windows 10 is 64-bit so you should choose that. If your computer is still running Windows 7, you may have issues with some of the packages. Set up for this course has not been fully tested on Windows 7, but there were some early issues that could not be fully overcome. Running the 32-bit installer on Windows 7 seemed to work better at the time.



Miniconda — Conda document
https://docs.conda.io/en/latest/miniconda.html
Conda
latest
Search docs
Conda
Conda build
Miniconda
Help and support
Contributing
Conda license
Read the Docs
https://conda.io/projects/conda/en/latest/user-guide/install/index.html

Miniconda

	Windows	Mac OS X	Linux
Python 3.7	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer) 64-bit (.pkg installer)	64-bit (bash installer) 32-bit (bash installer)
Python 2.7	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer) 64-bit (.pkg installer)	64-bit (bash installer) 32-bit (bash installer)

Installation instructions

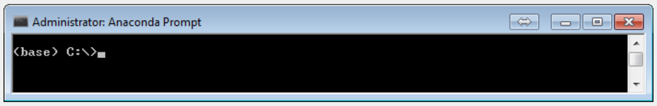
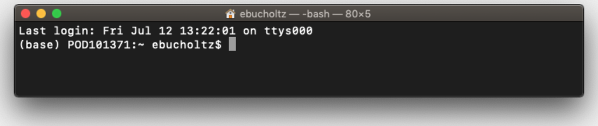
2. After downloading either the Windows exe or Mac OS X installer, double click and follow instructions for installation.

To check for successful installation:

- Windows: Open the Anaconda Prompt (Click Start, select Anaconda Prompt)
- macOS: Open Launchpad, then open terminal or iTerm.
  - After opening Anaconda prompt (terminal on Linux or macOS), choose any of the following methods:
    - Enter a command such as `conda list` . If Anaconda is installed and working, this will display a list of installed packages and their versions.
    - You should probably update your conda with `conda update -n base conda` .
    - Enter the command `python` . This command runs the Python shell. If Anaconda is installed and working, the version information it displays when it starts up will include “Anaconda”. To exit the Python shell, enter the command `quit()` .

Creation of our Cheminformatics Environment

When you open your Ananconda prompt you will have something that looks like either:

Windows	Mac
 `(base) C:\>`	 `Last login: Fri Jul 12 13:22:01 on ttys000` `(base) POD101371:~ ebucholtz$`

the (base) at the prompt indicates that you are in the base environment. We will be setting up our own environment for this course.

1. Regardless of operating system use the package management system conda to create an enviornment called OLCC2019 using the version of python you just downloaded. To do so at the prompt type:

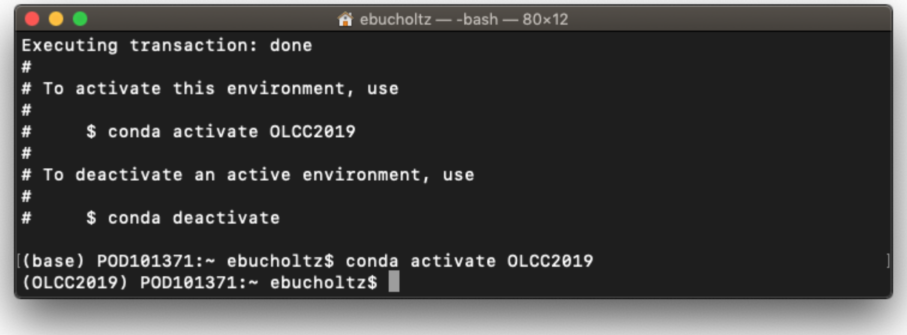
```
conda create -n OLCC2019 python=3.7
```

   and follow any prompts to proceed with y. This will install install necessary packages.

2. Regardless of operating system activate the environment. To do so at the prompt type:

```
conda activate OLCC2019
```

   you should notice that the prompt no longer says base, but OLCC2019. To go back to the base, type conda deactivate. This will allow us to have multiple environments for this course if we need later.



```
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate OLCC2019
#
# To deactivate an active environment, use
#
#     $ conda deactivate

[(base) POD101371:~ ebucholtz$ conda activate OLCC2019
(OLCC2019) POD101371:~ ebucholtz$
```

3. Regardless of operating system, install the following packages via conda. Each line below represents a command to type at the prompt.

Command	What it does
`conda install -c conda-forge rdkit`	Installs RDKit. The RDKit is an open source collection of cheminformatics and machine-learning software.
`conda install jupyter -y`	Installs Jupyter notebooks. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

`conda install -c conda-forge scikit-learr`	Installs Scikit-learn, which is a free machine learning software library for the python programming language.
`conda install -c conda-forge seaborn` the -y extension may not work with this command, so you will have answer y to proceed.	Installs Seaborn, which is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
`conda install -c mordred-descriptor mordr`	Installs mordred, which is a Python library for a developed descriptor-calculation software application that can calculate more than 1800 two- and three-dimensional descriptors.
`conda install -c anaconda pip -y`	Installs pip, which is the de facto standard package management system for python.
`conda install -c conda-forge pmw -y`	Installs PyMol terminal window pop up within Jupyter Notebooks.
`pip install biopandas`	Installs Biopandas which allows you to visualizemolecular structures of biological macromolecules (from PDB and MOL2 files) in pandas DataFrames.
`pip install pypdb`	Installs PyPDB which is a python programming interface for the RCSB Protein Data Bank (PDB).
Finally, lets do one more command to make sure all the conda installs have the proper updates and play well together: `conda update --all` You will have to click y to proceed.	This should be done as each package that was installed may be calling specifically for attributes that are in a previous version of a different package. Some packages may be updated, some new packages may be installed, and some packages may be downgraded. The goal is to create an environment where all packages play well together.

4. The final step is installation of pymol. Pymol is an open source molecular visualization system. Pymol has a paid version and a free version. The free version requires different installation steps dependent on the operating system you are using for this course.

Windows	Mac

1. Download pre-compiled Open-Source PyMOL from Christoph Gohlke of the Laboratory for Fluorescence Dynamics, University of California, Irvine. There are lots of pre-compiled distributions. The filename you are looking for is:

```
pymol-2.4.0a0-cp37-cp37m-win_amd64.whl
\ \ \
\ \ \__ for 64 bit Windows
\ \
\ \___________ for Python 3.7.x
\
\__________________ PyMOL version 2.4.0a0
```

2. Download the pre-compiled pymol lauchger as well: pymol_launcher-2.1-cp37-cp37m-win_amd64.whl
3. In the conda OLCC2019 environment, switch to the download directory of your computer (e.g. C:\Downloads, or C:\Users\yourusername\Downloads) <

```
<OLCC2019> C:\> cd C:\Downloads
```

```
<OLCC2019> C:\> cd C:\Users\yourusernam
```

4. Install the pymol launcher via pip (it also installs PyMol automatically)

```
pip install --no-index --find-links="%
```

5. Update Pymol with the following command:

```
pip install --upgrade --no-deps pymol-
```

Mac install directions will be added later. If you are familiar with macports or homebrew these are the easiest ways to add.

5. Link the conda environment to the Jupyter notebook.

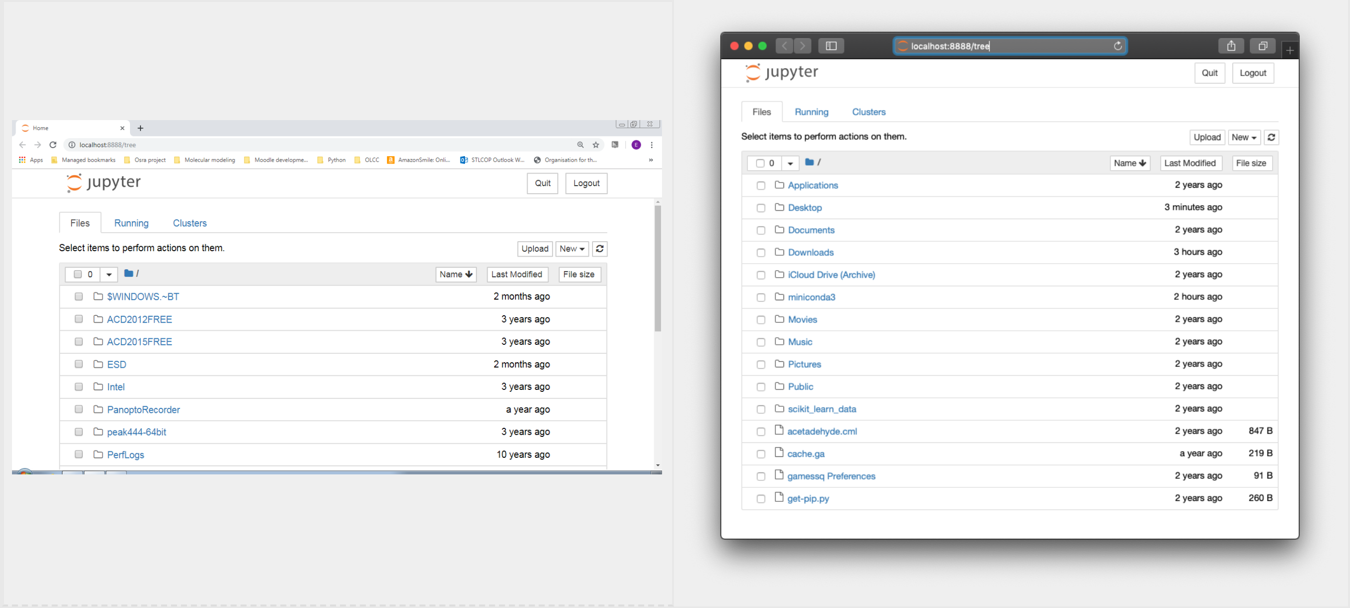
```
python -m ipykernel install --user --name OLCC2019
```

6. Start the jupyter notebook:

```
jupyter notebook
```

If you were successful, you should have a browser window that comes up like the following:

Windows	Mac



localhost:8888/tree

Select items to perform actions on them.

Name	Last Modified	File size
$WINDOWS.~BT	2 months ago	
ACD2012FREE	3 years ago	
ACD2015FREE	3 years ago	
ESD	2 months ago	
Intel	3 years ago	
PanoptoRecorder	a year ago	
peak444-64bit	3 years ago	
PerfLogs	10 years ago	

localhost:8888/tree

Select items to perform actions on them.

Name	Last Modified	File size
Applications	2 years ago	
Desktop	3 minutes ago	
Documents	2 years ago	
Downloads	3 hours ago	
iCloud Drive (Archive)	2 years ago	
miniconda3	2 hours ago	
Movies	2 years ago	
Music	2 years ago	
Pictures	2 years ago	
Public	2 years ago	
scikit_learn_data	2 years ago	
acetadehyde.cml	2 years ago	847 B
cache.ga	a year ago	219 B
gamessq Preferences	2 years ago	91 B
get-pip.py	2 years ago	260 B

Notice that the website is running locally on your machine as localhost:8888/tree

You can save files, make new folders, change directors all from this window.

1.5: Installing R

What is R?

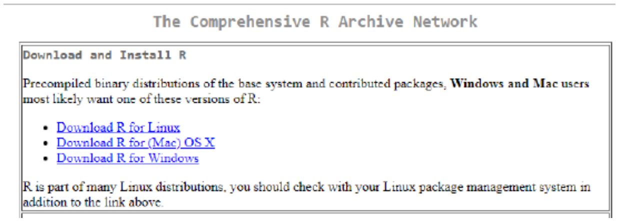
R is a free and open source programming language initially developed by Ross Ihaka and Robert Gentleman of the University of Auckland in 1993 and now maintained through the R-Project. It is a statistical computing program that supports graphical analysis, data mining and machine learning activities that we will be performing in this class. There is an extensive R developer community that have created over 15,000 additional packages that are available at the Comprehensive R Archive Network (CRAN).

Although we can run all our programs from within R, we are also going to install RStudio, which in an Interactive Developer Environment (IDE) for R. IDEs are code development environments that have a variety of useful features that make it easier to experiment around to both learn and test your code. Figure 1.4.2 shows the RStudio user interface.

Installations

Install R

1. Go to Project R (https://www.r-project.org/)
2. Choose CRAN download and this will take you to mirror sites across the planet (https://cran.r-project.org/mirrors.html)
3. Choose a mirror site near you
4. Choose your operating system, download and run the installer



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

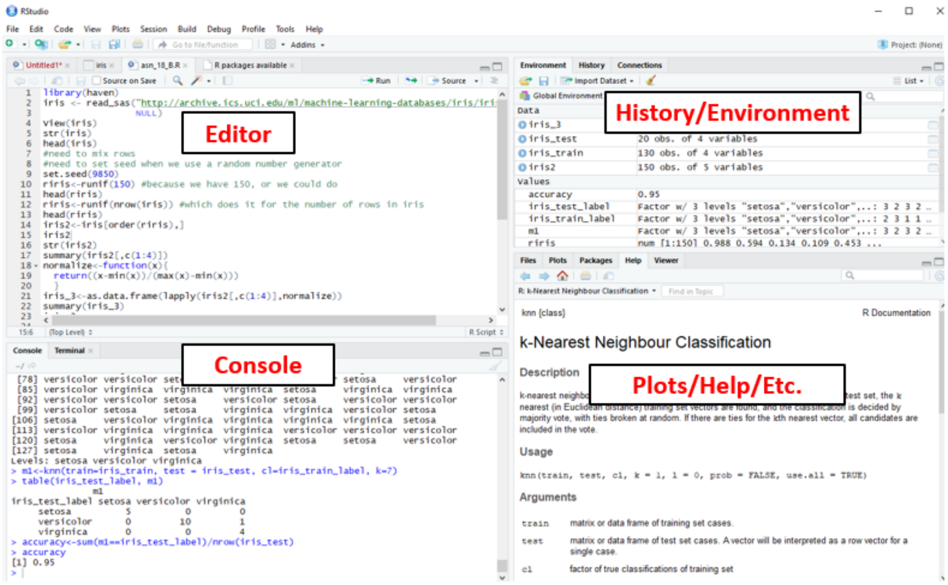
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Figure 1.5.1: Download options at CRAN (Comprehensive R Archive Network)

Install R Studio

In this class we will use R Studio, which is an IDE (Integrated Developer Environment), where

1. Got to RStudio (https://www.rstudio.com/)
2. Click on Download
3. Choose the free desktop "Open Source License" version
4. Follow Instructions



```
library(haven)
iris <- read_sas("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iri
                 NULL)
view(iris)
str(iris)
head(iris)
#need to mix rows
#need to set seed when we use a random number generator
set.seed(9850)
riris<-runif(150) #because we have 150, or we could do
head(riris)
riris<-runif(nrow(iris)) #which does it for the number of rows in iris
head(riris)
iris2<-iris[order(riris),]
iris2
str(iris2)
summary(iris2[,c(1:4)])
normalize<-function(x){
  return((x-min(x))/(max(x)-min(x)))
  }
iris_3<-as.data.frame(lapply(iris2[,c(1:4)],normalize))
summary(iris_3)
```

```
~/
 [78] versicolor versicolor set                              setosa     versicolor
 [85] versicolor virginica  virginica  virginica  setosa     virginica  virginica
 [92] versicolor versicolor setosa     versicolor setosa     versicolor versicolor
 [99] versicolor setosa     setosa     virginica  virginica  versicolor setosa
[106] setosa     versicolor virginica  virginica  virginica  virginica  setosa
[113] versicolor virginica  virginica  virginica  versicolor versicolor versicolor
[120] setosa     virginica  versicolor virginica  setosa     setosa     versicolor
[127] setosa     virginica  setosa     virginica
Levels: setosa versicolor virginica
> m1<-knn(train=iris_train, test = iris_test, cl=iris_train_label, k=7)
> table(iris_test_label, m1)
               m1
iris_test_label setosa versicolor virginica
     setosa          5          0         0
     versicolor      0         10         1
     virginica       0          0         4
> accuracy<-sum(m1==iris_test_label)/nrow(iris_test)
> accuracy
[1] 0.95
>
```

Data	
iris_3	
iris_test	20 obs. of 4 variables
iris_train	130 obs. of 4 variables
iris2	150 obs. of 5 variables
Values	
accuracy	0.95
iris_test_label	Factor w/ 3 levels "setosa","versicolor",..: 3 2 3 2 …
iris_train_label	Factor w/ 3 levels "setosa","versicolor",..: 2 3 1 1 …
m1	Factor w/ 3 levels "setosa","versicolor",..: 3 2 3 2 …
riris	num [1:150] 0.988 0.594 0.134 0.109 0.453 ...

knn {class} R Documentation

k-Nearest Neighbour Classification

Description

k-nearest neighbo test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

Usage

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Arguments

train	matrix or data frame of training set cases.
test	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
cl	factor of true classifications of training set

Figure 1.5.2: The RStudio IDE. Note, the top of the IDE has a toolbar that allows you a variety of functions and you can collapse the blocks and resize them to fit your needs.

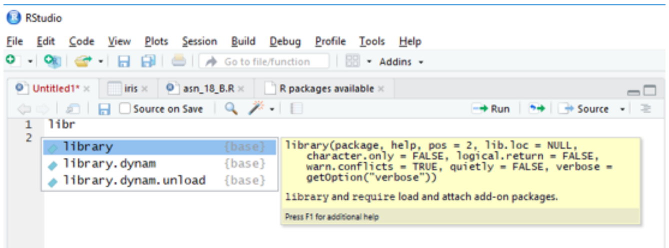
Install R Packages

Throughout this course you will need to install R Packages and so it is good to go over it now, at the very beginning. Some packages come with the default download and others you need to install from CRAN. But even the ones that come with the default installation of R need to be installed, as otherwise they would use up too much memory.

Identifying Downloaded Packages

library() identifies available packages

To see what Packages you have installed go to the editor and use the library() function. Note as you type, the IDE provides a list of options, and if you hover over "library", it places the syntax for that function into an overlay. If you click F1 while hoovering it places the library help commands into the bottom right block, and if you click on the word "library" it is inserted into the editor with parenthesis. Once you click library() you will get a list of all available packages



```
libr
```

```
library              {base}
library.dynam        {base}
library.dynam.unload {base}
```

```
library(package, help, pos = 2, lib.loc = NULL,
    character.only = FALSE, logical.return = FALSE,
    warn.conflicts = TRUE, quietly = FALSE, verbose =
    getOption("verbose"))
```

library and require load and attach add-on packages.

Press F1 for additional help

Downloading a New Package

install.packages("package name") - this command will download a package from CRAN to your computer. Note, this command uses parenthesis.

Activating a Package

library(package name) - this command will activate your package that has been downloaded so that you can run it in R. Note, this command does not use parenthesis.

Note on term "Library" is confusing: What you have is a library of packages on your own computer, and the command library(package name) is looking into the library for the package, and if it is there, it is activating the package. The package is not a library.

Optional Example: R Commander

If you are new to R you may find R Commander to help you learn commands. This is a package that gives an interface like Excel or Google sheets. Although it will not work for many of the things we are using in the class, it can save you time if you need to quickly learn how to make a scatter plot, or something like that. R Commander is not part of the R download, and so you will need to install and activate. So you may wish to perform the following tasks:

1. **library()** - scroll through the packages in your library, note Rcmdr is not there
2. **install.package("Rcmdr")** - this goes to CRAN and installs R Commander
3. **library()** - scroll through the packages in your library, note Rcmdr is there.
4. **library(Rcmdr)** - vola, you have an interface like Excel or Google Sheets. Note, if you perform a task, Rcmdr shows you the code it uses.

1.6: Installing Mathematica

What is Mathematica?

Mathematica is a technical computing environment sold by Wolfram Research. It can be run on both desktop computers and on the web. We generally recommend that you use the desktop version. The assignments presented here use Mathematica 12.1.

Obtaining Mathematica

Many universities have a site license that gives students and faculty free access to Mathematica. Your instructor can provide more information about accessing Mathematica at your school.

If your school does not have a site license, you will need to purchase a copy. A 15-day free trial is available, as are discounts for students and faculty.

Mathematica is free for the Raspberry Pi. Although the Raspberry Pi is somewhat slower than a typical desktop or laptop computer, the Raspberry Pi version of Mathematica has all of the functionality of the desktop computer version (except for some of the most compute-intensive machine learning-based computer vision functions). In general, this should not be a problem, as the rate limiting steps in these tutorials is usually remote database access. In general, it is recommended that you have a Raspberry Pi with 4 GB of RAM (or more) for best performance, but these tutorials should work even on Raspberry Pi's with 1 GB of RAM (however the graphical user interface may be quite slow).

Getting Started with Mathematica

Some resources for getting started in Mathematica are provided in the appendix.

1.7: Accessing PubChem through a Web Interface

PUG

PUG stands for Power User Gateway and is an Application Program Interface (API) service PubChem offers that allows users to access data programmatically. Access to this data is done through a REST or SOAP. REST is a web service type of architecture and uses web URIs (Uniform Resource identifiers). A URI is similar to the common web URL (Uniform Resource Locater) that browsers use to find web pages, but is associated with an object that may, or may not be a webpage (a URL is a type of URI). REST can provide data in many file formats, like text, html and jpeg. SOAP (Simple Object Access Protocol) is actually a protocol that works with XML files and is typically used for organizations that need higher levels of security. Although PUG works with both SOAP and REST, this course will focus on the use of REST interfaces.

REST Architecture

REST = Representational State Transfer is a way for computers to communicate over the web, where one computer may be a database server and the other is the client. One advantage to REST interfaces is that they are built upon the internet's Hypertext Transfer Protocol (http) that web browsers use, and which most people are familiar with. In essence, they are a special type of URL that interacts with specific objects with a database. A REST request is analogous to a sentence where the noun is the object and the verb is the action. Here are some typical REST verbs

- GET - retrieve a resource/object
- POST - upload a resource/object
- PUT - update a resource/object
- DELETE - remove a resource/object

In PubChem data is stored of essentially three types, each with its own identifier; compound (CID), substance (SID) and BioAssay (AID). The following figure shows the general process where you input a name, that gets converted to an identifier, and you then perform an operation to produce the type of object you are seeking and then returns an output of the file type that you are seeking.

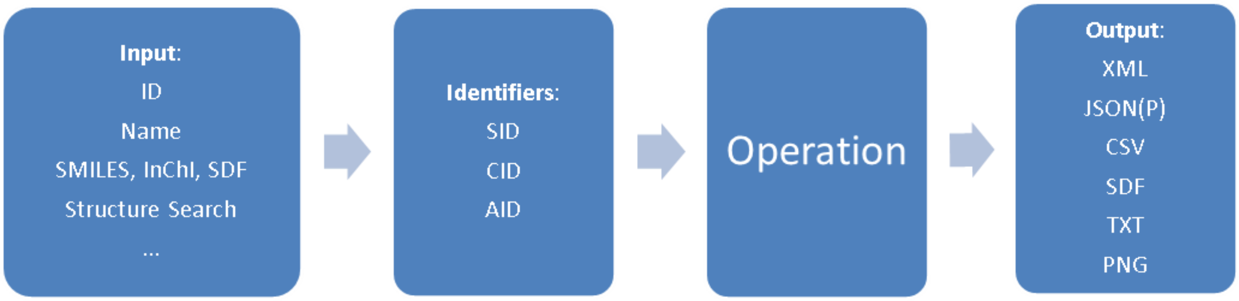

Input:
ID
Name
SMILES, InChI, SDF
Structure Search
...
Identifiers:
SID
CID
AID
Operation
Output:
XML
JSON(P)
CSV
SDF
TXT
PNG

Figure 1.7.1: Flow chart for a REST request in PubChem (Image Credit: PubChem)

The PUG REST request is based on http (or https) and we can consider the URL to consist of four parts; the prolog, input, operation and output

https://pubchem.ncbi.nlm.nih.gov/rest/pug	/compound/name/aspirin	/property/InChI	/TXT
prolog	*input*	*operation*	*output*

Prolog

The prolog essentially identifies the API service being used in the request.

Input

There are a variety of input methods supported

By Identifier

/substance/sid/[insert: substance ID]

/compound/cid/[insert: compound ID]

/assay/aid/[insert: Assay ID]

Examples

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/999/synonyms/txt

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/15/png

For a list of properties

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/1,2,3,4,5/property/MolecularFormula,MolecularWeight,CanonicalSMILES/CSV

For a summary of assay 999

https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/999/summary/JSON

By Name

/compound/name/[insert: name of chemical]

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/glucose/PNG

By Structure

If you have a structural drawing software you can convert you image to a SMILES string or InChI Key and search with that

/compound/smiles/[insert: smiles string here]/[output]/file type

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC(=O)C/property/IUPACName/txt

Operation

There is a variety of data available.

Images

Images are available for all types of structure input, just finish with png

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/THC/PNG

Synonyms

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/THC/synonyms/txt

Compound Properties

Note, these are computed properties. Actual experimental values are not available because there can be more than one value for the same property.

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/THC/property/MolecularWeight/txt

The following properties can be obtained through the REST architecture

Property	Notes
MolecularFormula	**Molecular formula**.
MolecularWeight	The molecular weight is the sum of all atomic weights of the constituent atoms in a compound, measured in g/mol. In the absence of explicit isotope labelling, averaged natural abundance is assumed. If an atom bears an explicit isotope label, 100% isotopic purity is assumed at this location.

Property	Notes
CanonicalSMILES	Canonical SMILES (Simplified Molecular Input Line Entry System) string. It is a unique SMILES string of a compound, generated by a “canonicalization” algorithm.
IsomericSMILES	Isomeric SMILES string. It is a SMILES string with stereochemical and isotopic specifications.
InChI	Standard IUPAC International Chemical Identifier (InChI). It does not allow for user selectable options in dealing with the stereochemistry and tautomer layers of the InChI string.
InChIKey	Hashed version of the full standard InChI, consisting of 27 characters.
IUPACName	Chemical name systematically determined according to the IUPAC nomenclatures.
XLogP	Computationally generated octanol-water partition coefficient or distribution coefficient. XLogP is used as a measure of hydrophilicity or hydrophobicity of a molecule.
ExactMass	The mass of the most likely isotopic composition for a single molecule, corresponding to the most intense ion/molecule peak in a mass spectrum.
MonoisotopicMass	The mass of a molecule, calculated using the mass of the most abundant isotope of each element.
TPSA	**Topological polar surface area**, computed by the algorithm described in **the paper by Ertl et al**.
Complexity	The **molecular complexity** rating of a compound, computed using the Bertz/Hendrickson/Ihlenfeldt formula.
Charge	The total (or net) charge of a molecule.
HBondDonorCount	Number of hydrogen-bond donors in the structure.
HBondAcceptorCount	Number of hydrogen-bond acceptors in the structure.
RotatableBondCount	Number of rotatable bonds.
HeavyAtomCount	Number of non-hydrogen atoms.
IsotopeAtomCount	Number of atoms with enriched isotope(s)
AtomStereoCount	Total number of atoms with tetrahedral (sp3) stereo [e.g., (R)- or (S)-configuration]
DefinedAtomStereoCount	Number of atoms with defined tetrahedral (sp3) stereo.
UndefinedAtomStereoCount	Number of atoms with undefined tetrahedral (sp3) stereo.
BondStereoCount	Total number of bonds with planar (sp2) stereo [e.g., (E)- or (Z)-configuration].
DefinedBondStereoCount	Number of atoms with defined planar (sp2) stereo.
UndefinedBondStereoCount	Number of atoms with undefined planar (sp2) stereo.
CovalentUnitCount	Number of covalently bound units.
Volume3D	Analytic volume of the first diverse conformer (default conformer) for a compound.

Property	Notes
XStericQuadrupole3D	The x component of the quadrupole moment (Qx) of the first diverse conformer (default conformer) for a compound.
YStericQuadrupole3D	The y component of the quadrupole moment (Qy) of the first diverse conformer (default conformer) for a compound.
ZStericQuadrupole3D	The z component of the quadrupole moment (Qz) of the first diverse conformer (default conformer) for a compound.
FeatureCount3D	Total number of 3D features (the sum of FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D and FeatureHydrophobeCount3D)
FeatureAcceptorCount3D	Number of hydrogen-bond acceptors of a conformer.
FeatureDonorCount3D	Number of hydrogen-bond donors of a conformer.
FeatureAnionCount3D	Number of anionic centers (at pH 7) of a conformer.
FeatureCationCount3D	Number of cationic centers (at pH 7) of a conformer.
FeatureRingCount3D	Number of rings of a conformer.
FeatureHydrophobeCount3D	Number of hydrophobes of a conformer.
ConformerModelRMSD3D	Conformer sampling RMSD in Å.
EffectiveRotorCount3D	Total number of 3D features (the sum of FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D and FeatureHydrophobeCount3D)
ConformerCount3D	The number of conformers in the conformer model for a compound.
Fingerprint2D	Base64-encoded PubChem Substructure Fingerprint of a molecule.

Output

The following output formats are supported

Output Format	Description
XML	standard XML, for which a schema is available
JSON	JSON, JavaScript Object Notation
JSONP	JSONP, like JSON but wrapped in a callback function
ASNB	standard binary ASN.1, NCBI’s native format in many cases
ASNT	NCBI’s human-readable text flavor of ASN.1
SDF	chemical structure data
CSV	comma-separated values, spreadsheet compatible
PNG	standard PNG image data
TXT	plain text

Sources

- PUG REST Tutorial https://pubchemdocs.ncbi.nlm.nih.gov/pug-rest-tutorial
-

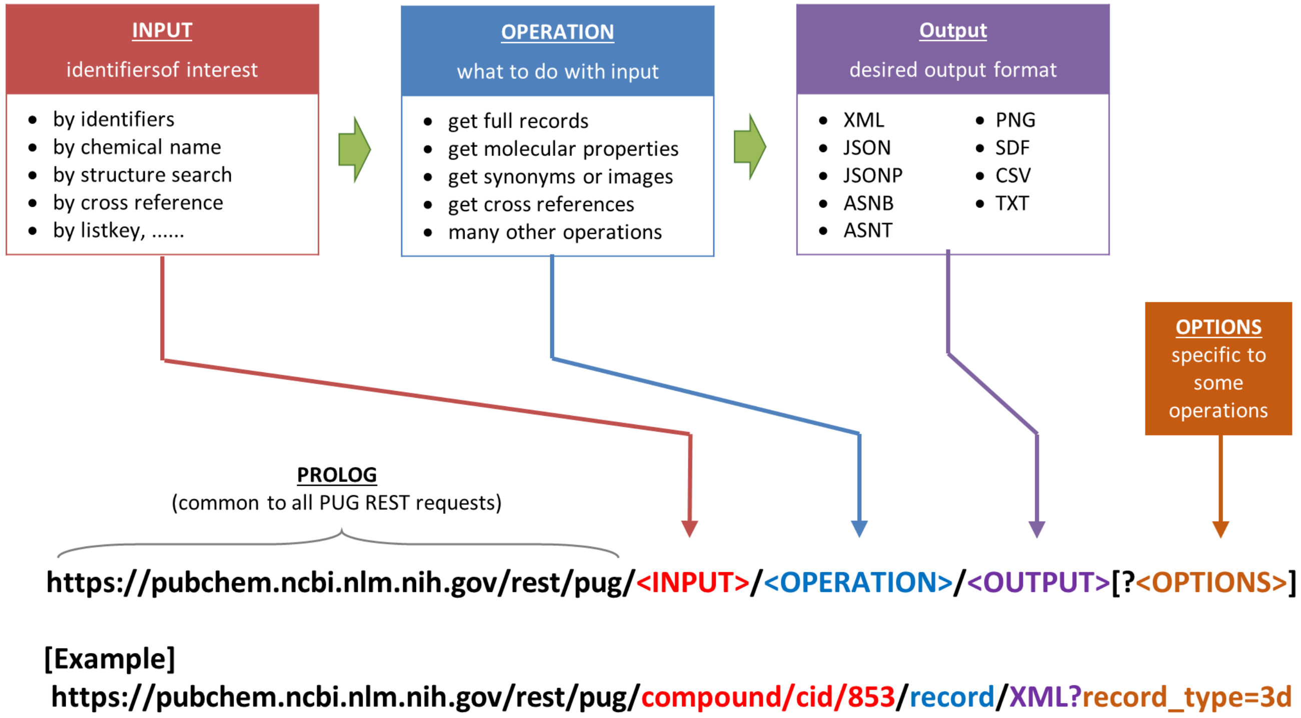
1.8: Programmatic Access to the PubChem Database

Concepts and Syntax of PUG-REST requests

PUG-REST is the simplest to use and learn among the existing programmatic access methods to PubChem. Importantly, because information necessary for a PUG-REST request can be encoded into a single Uniform Resource Locator (URL) that can be written by hand without programming expertise. Conceptually, a web service request from the user to PubChem requires three pieces of information:

- **input**: a list of PubChem identifiers of interest (e.g., CID, AID, SID).
- **operation**: what to do with the input identifiers.
- **output**: the format of the output from the operation.

In PUG-REST, these three pieces of information are encoded into an URL in the following format:



Some tasks require additional pieces of information that do not fit into the three-part PUG-REST URL. They should be provided as a list of '&'-separated option name and option value pairs, following the question mark ("?") appended at the end of the request URL. Some examples are presented in next section, but there are much more things that users can do through PUG-REST. To get more detailed information on PUG-REST, read the following four articles:

- **PUG-SOAP and PUG-REST: Web Services for Programmatic Access to Chemical Information in PubChem**
  Kim *et al.*, *Nucleic Acids Res.* **2015**, 43(W1), W605-W611.
  (http://dx.doi.org/10.1093/nar/gkv396).
- **An update on PUG-REST: RESTful interface for programmatic access to PubChem.**
  Kim *et al.*, *Nucleic Acids Res.* **2018**, 46(W1):W563-W570.
  (https://www.ncbi.nlm.nih.gov/pubmed/29718389).
- **PUG-REST Help** (http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html)
- **PUG-REST Tutorial** (http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html)

Example PUG-REST Request for Molecular Properties of a Compound

The request must include the PROLOG, the INPUT, the OPERATION, and the OUPUT. For any request, the prolog will have the format **https://pubchem.ncbi.nlm.nih.gov/rest/pug/**. The INPUT, OPERATION and OUTPUT will change depending on the context of the information you are requesting.

The **INPUT** is then added. In these three examples, the input is for acetone. the Example inputs can be based on:

- Name **compound/name/acetone/**
- Compound Identifier (CID) **compound/CID/180/**
- InChI Key **compound/inchikey/CSCPPACGZOOCGX-UHFFFAOYSA-N/**

The **OPERATION** is then added. In this case we will get the molecular weight, molecular formula, and its SMILES line notation string:**property/MolecularWeight,MolecularFormula,CanonicalSMILES/**

The **OUTPUT** can be obtained as text or comma separated values or eXtensible Markup Language data.

- Text= **TXT** *NOTE: This output type is limited to a single property value*
- comma separated values= **CSV**
- eXtensible Markup Language= **XML**

Putting these all together results in the following example requests:

1. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/acetone/property/MolecularWeight,MolecularFormula,CanonicalSMILES/XML
2. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/CID/180/property/MolecularWeight/TXT
3. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/inchikey/CSCPPACGZOOCGX-UHFFFAOYSA-N/property/MolecularWeight,MolecularFormula,CanonicalSMILES/CSV

Try it yourself!

Write a PUG-REST URL Request that returns an XML file for morphine that contains values for its compound identifier, IUPAC name, molecular formula, and hydrogen bond acceptor sites in the molecule. (*Hint: look at the output for example 1 above.*)

1.9: Cheminformatics Resources

Open Access Online Resources

Cheminformatics Books

The following list is in reverse chronological order (and alphabetical for publications of the same year). There is no endorsement of any book implied by this list, and when possible, a link to both Amazon and original publisher are provided.

1. Applied Chemoinformatics: Achievements and Future Opportunities, 2018, Engel & Gasteiger (Editors). Wiley, Amazon.
2. Chemoinformatics: Basic Concepts and Methods, 2018, Engel & Gasteiger (Editors). Wiley, Amazon .
3. In Silico Medicinal Chemistry, 2016, Nathan Brown. RSC Publishing, Amazon.
4. Introducing Cheminformatics, 2013, David Wild. LuLu, Amazon (Kindle)
5. Handbook of Chemoinformatics Algorithms, 2010, Faulon & Bender. CRC, Amazon
6. An Introduction to Chemoinformatics, 2003, Leach & Gillet. Springer, Amazon.
7. Chemoinformatics: A Textbook, 2003, Gasteiger & Engel, Wiley, Amazon.

Cheminformatics Journals

Open Access

Restricted Access

1.10: Python Assignment 1

Getting Molecular Properties through PUG-REST

Downloadable Files

Lecture01_Basics.ipynb

- Download and run the above file in your Jupyter notebook.
    - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
- This page is an html version of the above file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. If you are not on the discussion group you should contact your instructor for the link to join.

Objectives

- Learn the basic approach to getting data from PubChem through PUG-REST
- Retrieve a single property of a single compound.
- Retrieve a single property of multiple compounds
- Retrieve multiple properties of multiple compounds.
- Write a `for` loop to make the same kind of requests.
- Process a large amount of data by splitting them into smaller chunks

The Shortest Code to Get PubChem Data

Let's suppose that we want to get the molecular formula of water from PubChem through PUG-REST. You can get this data from your web browsers (Chrome, Safari, Internet Explorer, etc) via the following URL:
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/MolecularFormula/txt
Getting the same data using a computer program is not very difficult. This task can be with a three-line code.

Line 1: First, the "**requests**" **python library** (https://3.python-requests.org/) is imported. The "requests" library contains a set of pre-written codes that allows you to access information on the web.

In [1]:

```
import requests
```

Note: if you receive an error indicating that you do not have the requests library, you should go back to your anaconda prompt and type

```
pip install requests
```

Line 2: Get the desired information using **the function get( )** in the requests library. The PUG-REST request URL (enclosed within a pair of quotes(") is provided within the parentheses. The result will be stored in **a variable** called res .

In [2]:

```
res =
requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/prop
```

Line 3: The **res** variable (which means "result" or "response") contains not only the requested data but also some information about the request. To view the returned data, you need to get the data from **res** and **print** it out.

In [3]:

```
print(res.text)
```

As another example, the following code retrieves the number of heavy (non-hydrogen) atoms of butadiene.

In [4]:

```
res = requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/butadiene/
print(res.text)
```

Note that in this example, we did not import the **requests** library because it has already been imported (in the very fist example for getting the molecular formula of water).

LibreText Reading:

Review Section 1.6.2 Rest Architecture before doing this assignment, and reference back to the Compound Properties Table as needed.

Exercise 1a: Retrieve the molecular weight of ethanol in a "text" format.

In [7]:

```
# Write your code in this cell:
```

Exercise 1b: Retrieve the number of hydrogen-bond acceptors of aspirin in a "text" format.

In [8]:

```
# Write your code in this cell:
```

Formulating PUG-REST request URLs using variables

In the previous examples, the PUG-REST request URLs were directly provided to the **requests.get( )**, by explicitly typing the URL within the parentheses. However, it is also possible to provide the URL using a variable. The following example shows how to formulate the PUG-REST request URL using variables and pass it to **requests.get( )**.

In [9]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugin   = "compound/name/water"
pugoper = "property/MolecularFormula"
pugout  = "txt"

url     = pugrest + '/' + pugin + '/' + pugoper + '/' + pugout
print(url)
```

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/MolecularFormula/txt

A PUG-REST request URL encodes three pieces of information (input, operation, output), preceded by the prologue commont to all requests. In the above code cell, these pieces of information are stored in four different variables (**pugrest**, **pugin**, **pugoper**, **pugout**) and combined into a new variable **url**.

One can also generate the same URL using the **join( )** function, available for a string.

In [10]:

```
url = "/".join( [pugrest, pugin, pugoper, pugout] )
print(url)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/Molecular
```

Here, the strings stored in the four variables are joined by the "/" character as a separator. Note that the four variables are enclosed within the square bracket ([]), meaning that a list containing them as elements is provided to **join( )**.

Then, the url can be passed to **requests.get( )**.

In [11]:

```
res = requests.get(url)
print(res.text)
```

Warning: Avoid using **in** or **input** as a variable name in python. In python, in is a reserved keyword and input is the name of a built-in function. In the example above, the variables are prefixed with "pug" to avoid this naming conflict.

Making multiple requests using a for loop

The approach in the previous section (that use variables to construct a request URL) looks very inconvenient, compared to the three-line code shown at the beginning, where the request URL is directly provided to **requests.get( )**. If you are making only one request, it would be simpler to provide the URL directly to **requests.get( )**, rather than assign the pieces to variables, constructing the URL from them, and passing it to the function.
However, if you are making a large number of requests, it would be very time consuming to type the respective request URLs for all requests. In that case, you want to store common parts as variables and use them in a loop. For example, suppose that you want to retrieve the SMILES strings of 5 chemicals.

In [12]:

```
names = [ 'cytosine', 'benzene', 'motrin', 'aspirin', 'zolpidem' ]
```

Now the chemical names are stored in a list called **names** . Using a **for** loop, you can loop over each chemical name, formulating the request URL and retrieving the desired data, as shown below.

In [13]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper = "property/CanonicalSMILES"
pugout  = "txt"

for myname in names:    # loop over each element in the "names" list

    pugin = "compound/name/" + myname

    url = "/".join( [pugrest, pugin, pugoper, pugout] )
    res = requests.get(url)
    print(myname, ":", res.text)
```

Warning: When you make a lot of programmatic access requests using a loop, you should limit your request rate to or below **five requests per second**. Please read the following document to learn more about PubChem's usage

policies:https://pubchemdocs.ncbi.nlm.nih.gov/programmatic-access$_RequestVolumeLimitations
Violation of usage policies may result in the user being **temporarily blocked** from accessing PubChem (or NCBI) resources**

In the for-loop example above, we have only five input chemical names to process, so it is not likely to violate the five-requests-per-second limit. However, if you have thousands of names to process, the above code will exceed the limit (considering that this kind of requests usually finish very quickly). Therefore, the request rate should be adjusted by using the **sleep( )** function in the **time** module. For simplicity, let's suppose that you have 12 chemical names to process (in reality, you could have much more to process).

In [14]:

```
names = [ 'water', 'benzene', 'methanol', 'ethene', 'ethanol', \
          'propene','1-propanol', '2-propanol', 'butadiene', '1-butanol', \
          '2-butanol', 'tert-butanol']
```

LibreText Reading:

In analyzing the code of the following example you should reference Code Example 9.1.1 of Section 9.1.2 of Appendix 9.1

In [15]:

```
import time

pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper = "property/CanonicalSMILES"
pugout  = "txt"

for i in range(len(names)):    # loop over each index (position) in the "names" list

    pugin = "compound/name/" + names[i]    # names[i] = the ith element in the names list.

    url = "/".join( [pugrest, pugin, pugoper, pugout] )
    res = requests.get(url)
    print(names[i], ":", res.text)

    if ( i % 5 == 4 ) :  # the % is the modulo operator and returns the remainder of a calculation (if i = 4, 9, ...)
        time.sleep(1)
```

There are three things noteworthy in the above example (compared to the previous examples with the five chemical name queries).

- First, the for loop iterates from 0 to [**len(names)** − 1], that is, [0, 1, 2, 3, ...,11].
- The variable **i** is used (in **names(i)** to generate the input part (**pugin**) of the PUG-REST request URL.
- The variable **i** is used (in the **if** if sentence) to stop the program for one second for every five requests.

It should be noted that the request volumn limit can be lowered through the dynamic traffic control at times of excessive load (https://pubchemdocs.ncbi.nlm.nih.gov/dynamic-request-throttling). Throttling information is provided in the HTTP header response, indicating the system-load state and the per-user limits. Based on this throttling information, the user should moderate the speed at which requests are sent to PubChem. We will cover this topic later in this course.

Exercise 3a: Retrieve the XlogP values of linear alkanes with 1 ~ 12 carbons.

- Use the chemical names as inputs
- Use a for loop to retrieve the XlogP value for each alkane.
- Use the sleep() function to stop the program for one second for every **five** requests.

In [16]:

```
# Write your code in this cell: (The solution code below will be removed later)
```

Exercise 3b Retrieve the **isomeric** SMILES of the 20 common amino acids.

- Use the chemical names as inputs. Because the 20 common amino acids in living organisms predominantly exist as one chrial form (the L-form), the names should be prefixed with **"L-"** (e.g., "L-alanine", rather than "alanine"), except for "glycine" (which does not have a chiral center).
- Use a for loop to retrieve the isomeric SMILES for each alkane.
- Use the sleep() function to stop the program for one second for every **five** requests.

In [17]:

```
# Write your code in this cell (The solution code below will be removed later)
```

Getting multiple molecular properties

All the examples we have seen in this notebook retrieved a single molecular property for a single compound (although we were able to get a desired property for a group of compounds using a for loop). However, it is possible to get multiple properties for multiple compounds with a single request.

The following example retrieves the hydrogen-bond donor count, hydrogen-bond acceptor count, XLogP, TPSA for 5 compounds (represented by PubChem Compound IDs (CIDs) in a comma-separated values (CSV) format.

In [18]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugin   = "compound/cid/4485,4499,5026,5734,8082"
pugoper = "property/HBondDonorCount,HBondDonorCount,XLogP,TPSA"
pugout  = "csv"

url = "/".join( [pugrest, pugin, pugoper, pugout] )   # Construct the URL
print(url)
print("-" * 30)   # Print "-" 30 times (to print a line for readability)

res = requests.get(url)
print(res.text)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/4485,4499,5026,5734,8082/prope
------------------------------
"CID","HBondDonorCount","HBondDonorCount","XLogP","TPSA"
4485,1,1,2.200,110.0
4499,1,1,3.300,110.0
5026,1,1,4.300,123.0
5734,1,1,0.2,94.6
8082,1,1,0.800,12.0
```

In [19]:

```
res.text.rstrip()
```

Out[19]:

```
'"CID","HBondDonorCount","HBondDonorCount","XLogP","TPSA"\n4485,1,1,2.200,110.0\n4499
```

PubChem has a standard time limit of **30 seconds per request**. When you try to retrieve too many properties for too many compounds with a single request, it can take longer than the 30-second limit and a time-out error will be returned. Therefore, you may need to split the compound list into smaller chunks and process one chunk at a time.

In [20]:

```
cids = [ 443422,  72301,   8082,    4485,    5353740, 5282230, 5282138, 1547484, 9413
         5494,    5422,    5417,    5290,    5245,    5026,    4746,    4507,    4499
         4494,    4474,    4418,    4386,    4009,    4008,    3949,    3926,    3878
         3698,    3547,    3546,    3336,    3333,    3236,    3076,    2585,    2520
         2312,    2162,    1236,    1234,    292331,  275182,  235244,  108144,  1049
         5942250, 5311217, 4564402, 4715169, 5311501]
```

In [21]:

```
chunk_size = 10

if ( len(cids) % chunk_size == 0 ) : # check if total number of cids is divisible by
    num_chunks = len(cids) // chunk_size # sets number of chunks
else : # if divide by 10 results in remainder
    num_chunks = len(cids) // chunk_size + 1 # add one more chunk

print("# Number of CIDs:", len(cids) )
print("# Number of chunks:", num_chunks )
```

```
# Number of CIDs: 55
# Number of chunks: 6
```

In [22]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper = "property/HBondDonorCount,HBondAcceptorCount,XLogP,TPSA"
pugout  = "csv"

csv = ""   #sets a variable called csv to save the comma separated output

for i in range(num_chunks) : # sets number of requests to number of data chunks as de

    idx1 = chunk_size * i        # sets a variable for a moving window of cids to sta
    idx2 = chunk_size * (i + 1)  # sets a variable for a moving window of cids to end

    pugin = "compound/cid/" + ",".join([ str(x) for x in cids[idx1:idx2] ]) # build p
    url = "/".join( [pugrest, pugin, pugoper, pugout] )   # Construct the URL

    res = requests.get(url)
    if ( i == 0 ) : # if this is the first request, store result in empty csv variabl
        csv = res.text
    else :          # if this is a subsequent request, add the request to the csv var
        csv = csv + "\n".join(res.text.split()[1:]) + "\n"

    if (i % 5 == 4):
        time.sleep(1)

print(csv)
```

```
"CID","HBondDonorCount","HBondAcceptorCount","XLogP","TPSA"
443422,0,5,3.1,40.2
72301,0,5,3.2,40.2
8082,1,1,0.800,12.0
4485,1,7,2.200,110.0
5353740,2,5,3.5,76.0
5282230,2,5,3.2,84.9
5282138,1,8,4.400,120.0
1547484,0,2,5.800,6.5
941361,0,4,6.000,6.5
5734,1,5,0.2,94.6
5494,0,6,5.0,57.2
5422,0,8,6.4,61.9
5417,0,5,3.2,40.2
5290,2,5,2.6,62.2
5245,5,8,-3.1,148.0
5026,1,8,4.300,123.0
4746,1,1,6.8,12.0
4507,1,7,2.900,110.0
4499,1,7,3.300,110.0
4497,1,8,3.100,120.0
4494,1,8,2.900,134.0
4474,1,8,3.800,114.0
4418,1,5,4.100,45.2
4386,2,3,4.400,49.3
4009,2,5,3.5,76.0
4008,1,9,5.600,117.0
3949,0,7,4.9,34.2
3926,1,5,6.0,35.6
3878,2,5,1.4,90.7
3784,1,8,4.300,104.0
3698,2,3,-0.2,68.0
3547,1,5,1.0,70.7
3546,3,5,-0.5,132.0
3336,1,1,5.5,12.0
3333,1,5,3.900,64.6
```

```
3236,0,2,3.8,20.3
3076,0,6,3.1,84.4
2585,3,5,4.200,75.7
2520,0,6,3.800,64.0
2351,0,3,5.3,15.7
2312,0,2,4.6,12.5
2162,2,7,3.000,99.9
1236,1,8,6.800,114.0
1234,0,7,3.800,73.2
292331,2,3,3.900,49.3
275182,1,8,6.1,72.9
235244,1,8,6.7,72.9
108144,2,5,3.9,117.0
104972,1,6,3.300,72.7
77157,1,4,3.2,49.8
5942250,2,5,3.5,76.0
5311217,1,7,4.500,90.9
4564402,0,4,4.1,45.5
4715169,2,3,-1.6,63.3
5311501,0,4,4.4,43.7
```

Exercise 4a: Below is the list of CIDs of known antiinflmatory agents (obtained from PubChem via the URL: https://www.ncbi.nlm.nih.gov/pccompound?LinkName=mesh_pccompound&from_uid=68000893). Download the following properties of those compounds in a comma-separated format: Heavy atom count, rotatable bond count, molecular weight, XLogP, hydrogen bond donor count, hydrogen bond acceptor count, TPSA, and isomeric SMILES.

- Split the input CID list into small chunks (with a chunk size of 100 CIDs).
- Process one chunk at a time using a for loop.
- Do not forget to add sleep() to comply the usage policy.

In [23]:

```
cids = [ 471, 1981, 2005, 2097, 2151, 2198, 2206, 2214, 2244, 2307, 2308, 2313, 2355,
len(cids)
```

Out[23]:

```
708
```

In [24]:

```
# Write your code in this cell.
```

1.11: R Assignment 1

Getting Molecular Properties through PUG-REST

S. Kim, J. Cuadros

August 4th, 2019

Downloadable Files

L01_MolecularPropPUGREST.R
L01_MolecularPropPUGREST_wc.R
RL01_PugRest.ipynb

- You can use the R-studio you created in section 1.4 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
- This page is an html version of the above R file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. If you are not on the discussion group you should contact your instructor for the link to join.
- _wc is the R file with comments.
- .pynb is a Jupyter Notebook that opens with an R Kernal

Objectives

- Learn the basic approach to getting data from PubChem through PUG-REST
- Retrieve a single property of a single compound.
- Retrieve a single property of multiple compounds
- Retrieve multiple properties of multiple compounds.
- Write a `for` loop to make the same kind of requests.
- Process a large amount of data by splitting them into smaller chunks

The Shortest Code to Get PubChem Data

Let's suppose that we want to get the molecular formula of water from PubChem through PUG-REST. You can get this data from your web browsers (Chrome, Safari, Internet Explorer, etc) via the following URL:
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/MolecularFormula/txt

Getting the same data using a computer program is not very difficult. In R, this task can be with a single line of code.

```
readLines('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/Mo
FALSE)
```

```
## [1] "H2O"
```

The `readLines` function can read a text from a URL (or a file). The PUG-REST request URL, enclosed within a pair of single or double quotes('x' or "x"), is provided within the parentheses. The second argument, `warn=FALSE` avoids a warning that appears when the data stream doesn't end with an empty line.

There are many other ways to get information from the web into R, for example using the function `GET` from the `httr` package, but for the moment we will stick to the simplest option.

As another example, the following code retrieves the number of heavy (non-hydrogen) atoms of butadiene.

```
readLines('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/butadiene/propert
warn=TRUE)
```

```
## [1] "4"
```

Exercise 1a: Retrieve the molecular weight of ethanol in a "text" format.

```
# Write your code here
```

Exercise 1b: Retrieve the number of hydrogen-bond donors of aspirin in a "text" format.

```
# Write your code here
```

Formulating PUG-REST request URLs using variables

In the previous examples, the PUG-REST request URLs were directly provided to the `readLines` , by explicitly typing the URL within the parentheses. However, it is also possible to provide the URL using a variable. The following example shows how to formulate the PUG-REST request URL using variables and pass it to `readLines` .

```
pugrest <- 'https://pubchem.ncbi.nlm.nih.gov/rest/pug'
pugin <- 'compound/name/water'
pugoper <- 'property/MolecularFormula'
pugout <- 'txt'

url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
url
```

```
##     [1]     "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/
MolecularFormula/txt"
```

A PUG-REST request URL encodes three pieces of information (input, operation, output), preceded by the prologue common to all requests. In the above code, these pieces of information are stored in four different variables (`pugrest` , `pugin` , `pugoper` , `pugout`) and combined into a new variable `url` .

Here, the strings stored in the four variables are joined by the â€œ/â€ character as a separator.

Then, the url can be passed to `readLines` .

```
readLines(url)
```

```
## [1] "H2O"
```

Warning: Avoid using reserved or function names as a variable names. `in` , `c` , `t` , `names ...` are some examples of variable names to be avoided in `R` . In the example above, the variables are prefixed with "pug" to avoid naming conflicts.

Making multiple requests using a for loop

The approach in the previous section (that use variables to construct a request URL) looks very inconvenient, compared to the code shown at the beginning, where the request URL is directly provided to `readLines` . If you are making only one request, it would be simpler to provide the URL directly to `readLines` , rather than assigning the pieces to variables, constructing the URL from them, and passing it to the function. However, if you are making a large number of requests, it would be very time consuming to type the respective request URLs for all requests. In that case, you want to store common parts as variables and use them in a loop. For example, suppose that you want to retrieve the SMILES strings of 5 chemicals.

```
chemicals <- c('cytosine', 'benzene', 'motrin', 'aspirin', 'zolpidem')
```

Now the chemical names are stored in a character vector called `chemicals` . Using a `for` loop, you can loop over each chemical name, formulating the request URL and retrieving the desired data, as shown below.

```
pugrest <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper <- "property/CanonicalSMILES"
pugout  <- "txt"

# loop over each element in the 'chemicals' list
for(chemical in chemicals) {
    pugin <- paste('compound/name/',chemical,sep="")
    url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
    res <- readLines(url,warn=TRUE)
    print(paste(chemical, ": ", res,sep=""))
}
```

```
## [1] "cytosine: C1=C(NC(=O)N=C1)N"
## [1] "benzene: C1=CC=CC=C1"
## [1] "motrin: CC(C)CC1=CC=C(C=C1)C(C)C(=O)O"
## [1] "aspirin: CC(=O)OC1=CC=CC=C1C(=O)O"
## [1] "zolpidem: CC1=CC=C(C=C1)C2=C(N3C=C(C=CC3=N2)C)CC(=O)N(C)C"
```

Warning: When you make a lot of programmatic access requests using a loop, you should limit your request rate to or below **five requests per second**. Please read the following document to learn more about PubChem's usage policies: https://pubchemdocs.ncbi.nlm.nih.gov/programmatic-access$_RequestVolumeLimitations . **Violation of usage policies** may result in the user being **temporarily blocked** from accessing PubChem (or NCBI) resources**

In the for-loop example above, we have only five input chemical names to process, so it is not likely to violate the five-requests-per-second limit. However, if you have thousands of names to process, the above code will exceed the limit (considering that this kind of requests usually finish very quickly). Therefore, the request rate should be adjusted by using the `Sys.sleep` function. For simplcity, let's suppose that you have 12 chemical names to process (in reality, you could have much more to process).

```
chemicals <- c('water', 'benzene', 'methanol', 'ethene', 'ethanol',
               'propene','1-propanol', '2-propanol', 'butadiene',
               '1-butanol', '2-butanol', 'tert-butanol')
```

```
pugrest <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper <- "property/CanonicalSMILES"
pugout  <- "txt"

# loop over each element in the 'chemicals' list
for(chemical in chemicals) {
    pugin <- paste('compound/name/',chemical,sep="")
    url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
    res <- readLines(url,warn=TRUE)
    print(paste(chemical, ": ", res,sep=""))
    Sys.sleep(.3)
}
```

```
## [1] "water: O"
## [1] "benzene: C1=CC=CC=C1"
## [1] "methanol: CO"
## [1] "ethene: C=C"
## [1] "ethanol: CCO"
## [1] "propene: CC=C"
## [1] "1-propanol: CCCO"
## [1] "2-propanol: CC(C)O"
## [1] "butadiene: C=CC=C"
## [1] "1-butanol: CCCCO"
## [1] "2-butanol: CCC(C)O"
## [1] "tert-butanol: CC(C)(C)O"
```

It should be noted that the request volume limit can be lowered through the dynamic traffic control at times of excessive load (https://pubchemdocs.ncbi.nlm.nih.gov/dynamic-request-throttling). Throttling information is provided in the HTTP header response, indicating the system-load state and the per-user limits. Based on this throttling information, the user should moderate the speed at which requests are sent to PubChem. We will cover this topic later in this course.

Exercise 3a: Retrieve the XlogP values of linear alkanes with 1 ~ 12 carbons.
- Use the chemical names as inputs - Use a `for` loop to retrieve the XlogP value for each alkane. - Use the `Sys.sleep` function to stop the program for 10 seconds for every request.

```
# Write your code here
```

Exercise 3b: Retrieve the **isomeric** SMILES of the 20 common aminoacids. - Use the chemical names as inputs. Because the 20 common aminoacids in living organisms predominantly exist as one chrial form (the L-form), the names should be prefixed with **'˜L-'** (e.g., 'L-alanine', rather than 'alanine'), except for 'glycine' (which does not have a chiral center). - Use a `for` loop to retrieve the isomeric SMILES for each alkane. - Use the `Sys.sleep` function to stop the program for 10 seconds for every request.

```
# Write your code here
```

Getting multiple molecular properties

All the examples we have seen in this notebook retrieved a single molecular property for a single compound (although we were able to get a desired property for a group of compounds using a **for** loop). However, it is possible to get multiple properties for multiple compounds with a single request.

The following example retrieves the hydrogen-bond donor count, hydrogen-bond acceptor count, XLogP, TPSA for 5 compounds, represented by PubChem Compound IDs (CIDs) in a comma-separated values (CSV) format.

```
pugrest <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugin   <- "compound/cid/4485,4499,5026,5734,8082"
pugoper <- "property/HBondDonorCount,HBondDonorCount,XLogP,TPSA"
pugout  <- "csv"

url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
url
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/4485,4499,5026,5734,8082/property/HBondDonorCount,HBondDonorCount,XLogP,TPSA/csv"
```

```
# Print "-" 30 times (to print a line for readability)
cat(strrep("-",30))
```

```
## ------------------------------
```

```
read.table(url,sep=",",header=TRUE)
```

```
## CID HBondDonorCount HBondDonorCount.1 XLogP TPSA
## 1 4485 1 1 2.2 110.0
## 2 4499 1 1 3.3 110.0
## 3 5026 1 1 4.3 123.0
## 4 5734 1 1 0.2 94.6
## 5 8082 1 1 0.8 12.0
```

In `R` , the `read.table` function allows reading formatted-text data files (or streams), such as CSV files. It returns a `data.frame` , which is a convenient data structure for data tables.

PubChem has a standard time limit of **30 seconds per request**. When you try to retrieve too many properties for too many compounds with a single request, it can take longer than the 30-second limit and a time-out error will be returned. Therefore, you may need to split the compound list into smaller chunks and process one chunk at a time.

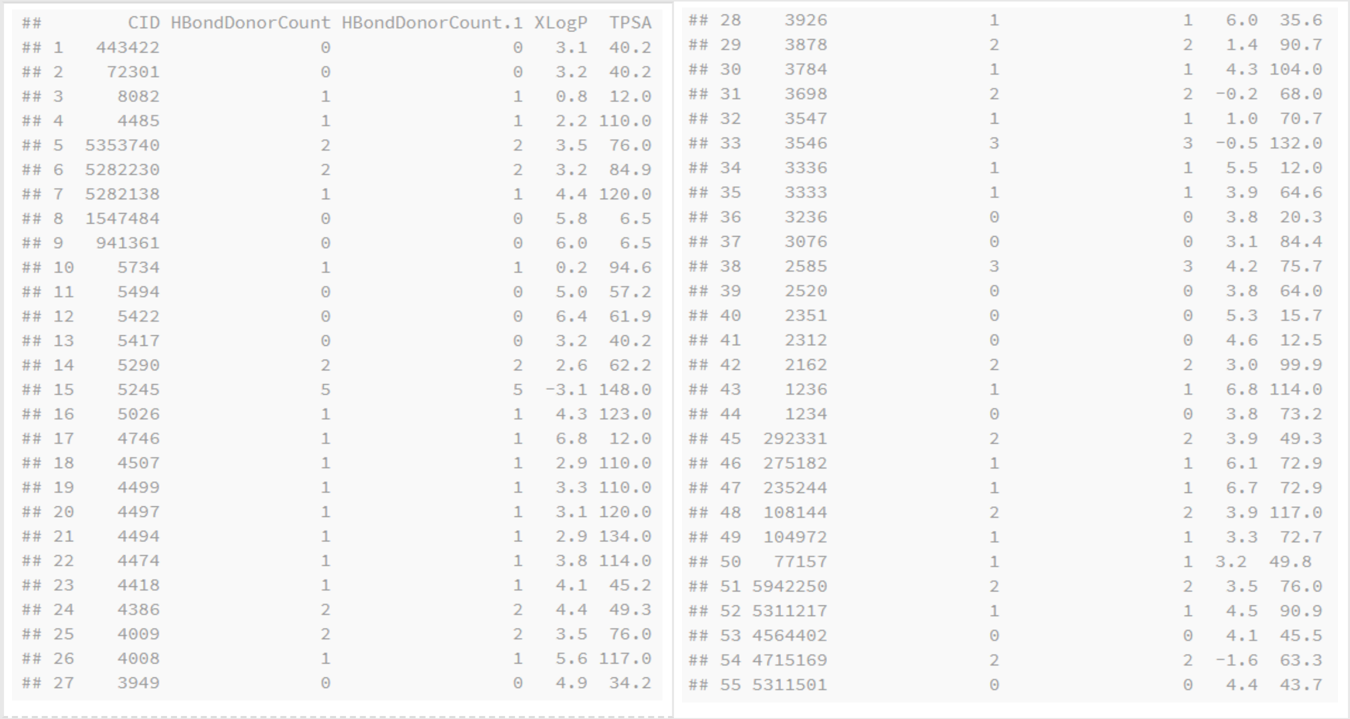
```
cids <- c(443422, 72301, 8082, 4485, 5353740, 5282230, 5282138, 1547484,
          941361, 5734, 5494, 5422, 5417, 5290, 5245, 5026, 4746, 4507,
          4499, 4497, 4494, 4474, 4418, 4386, 4009, 4008, 3949, 3926, 3878,
          3784, 3698, 3547, 3546, 3336, 3333, 3236, 3076, 2585, 2520, 2351,
          2312, 2162, 1236, 1234, 292331, 275182, 235244, 108144, 104972, 77157,
          5942250, 5311217, 4564402, 4715169, 5311501)
chunk_size <- 10
num_chunks <- ceiling(length(cids) / chunk_size)

paste("# Number of CIDs:", length(cids) )
```

```
## [1] "# Number of CIDs: 55"
```

```
paste("# Number of chunks:", num_chunks )
```

```
## [1] "# Number of chunks: 6"
```

```
pugrest <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper <- "property/HBondDonorCount,HBondDonorCount,XLogP,TPSA"
pugout  <- "csv"

pugin <- paste("compound/cid/",
               paste(cids[1:10],collapse=","),sep="")

url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
df <- read.table(url,sep=",",header=TRUE)


for (i in 2:num_chunks) {
    idx1 = chunk_size * (i - 1) + 1
    idx2 = chunk_size * i

    pugin <- paste("compound/cid/",
                   paste(cids[idx1:pmin(idx2,length(cids))],collapse=","),sep="")

    url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
    df <- rbind(df,read.table(url,sep=",",header=TRUE))
    Sys.sleep(10)
}
df
```



```
##        CID HBondDonorCount HBondDonorCount.1 XLogP  TPSA
## 1   443422               0                 0   3.1  40.2
## 2    72301               0                 0   3.2  40.2
## 3     8082               1                 1   0.8  12.0
## 4     4485               1                 1   2.2 110.0
## 5  5353740               2                 2   3.5  76.0
## 6  5282230               2                 2   3.2  84.9
## 7  5282138               1                 1   4.4 120.0
## 8  1547484               0                 0   5.8   6.5
## 9   941361               0                 0   6.0   6.5
## 10    5734               1                 1   0.2  94.6
## 11    5494               0                 0   5.0  57.2
## 12    5422               0                 0   6.4  61.9
## 13    5417               0                 0   3.2  40.2
## 14    5290               2                 2   2.6  62.2
## 15    5245               5                 5  -3.1 148.0
## 16    5026               1                 1   4.3 123.0
## 17    4746               1                 1   6.8  12.0
## 18    4507               1                 1   2.9 110.0
## 19    4499               1                 1   3.3 110.0
## 20    4497               1                 1   3.1 120.0
## 21    4494               1                 1   2.9 134.0
## 22    4474               1                 1   3.8 114.0
## 23    4418               1                 1   4.1  45.2
## 24    4386               2                 2   4.4  49.3
## 25    4009               2                 2   3.5  76.0
## 26    4008               1                 1   5.6 117.0
## 27    3949               0                 0   4.9  34.2
```

```
## 28    3926               1                 1   6.0  35.6
## 29    3878               2                 2   1.4  90.7
## 30    3784               1                 1   4.3 104.0
## 31    3698               2                 2  -0.2  68.0
## 32    3547               1                 1   1.0  70.7
## 33    3546               3                 3  -0.5 132.0
## 34    3336               1                 1   5.5  12.0
## 35    3333               1                 1   3.9  64.6
## 36    3236               0                 0   3.8  20.3
## 37    3076               0                 0   3.1  84.4
## 38    2585               3                 3   4.2  75.7
## 39    2520               0                 0   3.8  64.0
## 40    2351               0                 0   5.3  15.7
## 41    2312               0                 0   4.6  12.5
## 42    2162               2                 2   3.0  99.9
## 43    1236               1                 1   6.8 114.0
## 44    1234               0                 0   3.8  73.2
## 45  292331               2                 2   3.9  49.3
## 46  275182               1                 1   6.1  72.9
## 47  235244               1                 1   6.7  72.9
## 48  108144               2                 2   3.9 117.0
## 49  104972               1                 1   3.3  72.7
## 50   77157               1                 1  3.2  49.8
## 51 5942250               2                 2   3.5  76.0
## 52 5311217               1                 1   4.5  90.9
## 53 4564402               0                 0   4.1  45.5
## 54 4715169               2                 2  -1.6  63.3
## 55 5311501               0                 0   4.4  43.7
```

Figure \(\PageIndex{1}\): Screen Capture Image of output for above code

Exercise 4a: Below is the list of CIDs of known antiinflmatory agents (obtained from PubChem via the URL: https://www.ncbi.nlm.nih.gov/pccompound?LinkName=mesh_pccompound&from_uid=68000893). Download the following properties of those compounds in a comma-separated format: Heavy atom count, rotatable bond count, molecular weight, XLogP, hydrogen bond donor count, hydrogen bond acceptor count, TPSA, and isomeric SMILES.

- Split the input CID list into small chunks (with a chunk size of 100 CIDs).
- Process one chunk at a time using a for loop.
- Do not forget to add `Sys.sleep` to comply the usage policy.

```
cids <-  c(471, 1981, 2005, 2097, 2151, 2198, 2206, 2214, 2244, 2307, 2308, 2313,
2355, 2396, 2449, 2462, 2466, 2581, 2662, 2794, 2863, 3000, 3003, 3033, 3056,
3059, 3111, 3177, 3194, 3230, 3242, 3282, 3308, 3332, 3335, 3342, 3360, 3371,
3379, 3382, 3384, 3394, 3495, 3553, 3612, 3672, 3715, 3716, 3718, 3778, 3824,
3825, 3826, 3935, 3946, 3965, 4009, 4037, 4038, 4044, 4075, 4159, 4237, 4386,
4409, 4413, 4487, 4488, 4495, 4534, 4553, 4614, 4641, 4671, 4692, 4781, 4888,
4895, 4921, 5059, 5090, 5147, 5161, 5208, 5228, 5339, 5352, 5359, 5362, 5468,
5469, 5475, 5480, 5509, 5733, 5743, 5744, 5745, 5753, 5754, 5755, 5834, 5865,
5875, 5876, 5877, 6094, 6213, 6215, 6247, 6436, 6741, 7090, 7497, 8522, 9053,
9231, 9642, 9782, 9878, 10114, 10154, 10170, 10185, 10206, 12555, 12938, 13802,
14982, 15209, 16490, 16533, 16623, 16639, 16752, 16923, 17198, 19161, 20469,
21102, 21700, 21800, 21826, 21975, 22419, 23205, 26098, 26248, 26318, 28718,
28871, 30869, 30870, 30951, 31307, 31378, 31508, 31635, 31799, 31800, 32153,
32327, 32798, 33958, 35375, 35455, 35935, 36833, 37425, 38081, 38503, 39212,
39941, 40000, 40632, 41643, 43261, 44219, 47462, 47795, 50294, 50295, 51717,
54445, 54585, 57782, 59757, 60164, 60490, 60542, 60712, 60726, 60864, 61486,
62074, 62924, 63006, 63019, 64704, 64738, 64746, 64747, 64927, 64945, 64971,
64982, 65394, 65464, 65655, 65679, 65762, 66249, 67417, 68700, 68704, 68706,
68731, 68749, 68819, 68865, 68869, 68917, 71246, 71354, 71364, 71386, 71398,
71414, 71415, 71771, 72158, 72300, 73400, 82153, 84003, 84429, 90763, 91626,
91670, 100472, 102011, 104762, 104943, 107641, 107738, 107793, 108068, 108130,
114753, 114840, 114917, 114999, 115239, 119032, 119286, 119365, 119607, 119828,
119871, 121928, 121957, 122139, 122179, 122182, 123619, 123673, 123723, 124978,
128191, 128229, 128571, 133021, 134896, 146364, 151075, 151166, 152165, 155354,
155761, 156391, 158103, 159557, 162666, 164676, 167928, 168928, 174093, 174277,
176155, 177976, 180604, 183088, 189821, 192156, 196122, 196840, 196841, 200674,
201587, 219121, 222786, 229860, 235244, 236702, 259846, 263373, 275182, 292331,
425990, 439503, 439533, 441335, 441336, 442534, 442993, 443943, 443949, 443967,
444036, 445154, 445858, 446925, 479503, 485711, 490428, 501254, 522325, 546807,
578771, 584547, 610479, 633091, 633097, 636374, 636398, 656604, 656656, 656852,
657238, 667550, 927704, 969510, 969516, 1548887, 1548910, 2737488, 3033890,
3033980, 3045402, 3051696, 3055172, 4129359, 4306515, 4483645, 5018304, 5185849,
5280802, 5280914, 5280915, 5281004, 5281071, 5281515, 5281522, 5281792, 5282183,
5282193, 5282230, 5282387, 5282402, 5282492, 5283542, 5283734, 5284538, 5284539,
5311051, 5311052, 5311066, 5311067, 5311093, 5311101, 5311108, 5311169, 5311180,
5318517, 5320420, 5322111, 5352624, 5353725, 5353726, 5353740, 5353864, 5354499,
5377381, 5420804, 5420805, 5458396, 5472495, 5481958, 5701991, 5702036, 5702148,
5702212, 5702252, 5702287, 5745214, 5942250, 6420050, 6429274, 6437368, 6437387,
6438873, 6447131, 6453785, 6473881, 6509979, 6708733, 6710677, 6714002, 6917783,
6917852, 6917894, 6918172, 6918173, 6918332, 6918445, 6918452, 6918612, 6925666,
7060958, 7251185, 9554199, 9798098, 9799453, 9841438, 9843941, 9846332, 9865808,
9868219, 9869053, 9871508, 9875547, 9883509, 9897518, 9897771, 9907157, 9913795,
9919776, 9926694, 9934547, 10363606, 10918539, 11158972, 11513733, 11561674,
11616712, 11870423, 11949636, 11954221, 11954316, 11954353, 11954369, 11957468,
11961431, 11972243, 11972532, 12300053, 12313906, 12313911, 12606303, 12634263,
12714644, 12874922, 13018150, 13020033, 13041095, 14010989, 14515707, 14798494,
15895902, 16051947, 16132369, 16213022, 16213698, 16218996, 16219353, 16220118,
16759566, 16760658, 17750985, 17753757, 18526330, 18632363, 18647121, 18943026,
20054915, 21120116, 21637635, 21637642, 21893738, 21893804, 21982135, 22141508,
22811280, 23509770, 23631982, 23653552, 23657872, 23663407, 23663409, 23663418,
23663959, 23663989, 23665411, 23665999, 23667642, 23669636, 23674183, 23674255,
23674745, 23675763, 23680530, 23681059, 23684814, 23688663, 23693301, 23694214,
23702389, 24181458, 24721429, 24761485, 24799587, 24847961, 24847981, 24867460,
24867465, 24867475, 24883465, 24916955, 25077872, 25113755, 25796773, 40469526,
44119558, 44202892, 44260118, 44266812, 44386560, 45006151, 45006158, 45039955,
45356876, 45356931, 45357558, 45357932, 45358013, 45358120, 45358130, 45358140,
45358148, 45358149, 45488525, 46174093, 46397498, 46780650, 46780910, 46783539,
46783786, 46783814, 46863906, 46878350, 46882877, 50989825, 51026956, 51340230,
51398089, 53384387, 53394893, 53486221, 53486290, 53486322, 54194814, 54605501,
54675840, 54676228, 54677470, 54677971, 54677972, 54677977, 54682045, 54684589,
54690031, 54697648, 54708862, 54714524, 56841932, 56842111, 56845155, 57347755,
57486087, 67668959, 67804972, 67986221, 70470286, 70678885, 71306882, 71587162,
72774967, 72941490, 72941625, 73758129, 73759663, 73759808, 74787565, 77906397,
78577433, 90488794, 91711382, 91826463, 91873711, 91881846, 92131836, 92462493,
102004404, 102601886, 117072385, 117072403, 117072410, 118701141, 118701402,
118984459, 122130078, 122130111, 122130185, 122130213, 122130768, 122173054,
122173183, 122361610, 123134657, 124081055, 124463365, 126968472, 126968501,
126968801, 126969212, 126969455, 129009998, 129010022, 129010033, 129010043,
129316829, 129317859, 129317898, 129628207, 129628892, 129670532, 129735029,
131632430, 131635023, 131676243, 131750284, 131954647, 131954667, 132399051,
132399058, 133112890, 133126366, 133126370, 133562807, 133659920, 133687604,
134129698, 134159361, 134460917, 134612785, 134687786, 134688123, 134688323,
134688977, 134689786, 134693106, 134693125, 134693234, 134694728, 134694860,
135413496, 135413505, 135414247, 135484078, 135515521, 135565709, 136040192,
137177332, 137699687, 137705034, 137705717, 137705725, 137705994, 137706376,
137706400, 137795135, 138059757, 138107776, 138113311, 138113507, 138113581,
138114182, 138114743)

length(cids)
```

```
## [1] 708
```

```
# Write your code here
```

1.12: Mathematica Assignment 1

CHAPTER OVERVIEW

2: Representing Small Molecules on Computers

Hypothes.is Tag= f19OLCCc2

Topic hierarchy

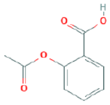
2.1: Introduction

Learning Objectives:

- Gain understanding of chemical structural data
- Introduce basic ways of communicating structural data

Representing Chemicals

In your studies of chemistry starting in the freshmen years you have encountered many ways of representing chemicals, and here we will list a few.

1. Trivial Names (Aspirin)
2. Systematic Names (2-acetyloxybenzoic acid)
3. Formula ($C_9H_8O_4$)
4. Images 

We will now look at several other ways of representing chemicals, most notably connection tables and line notation.

Chemical Representation for Cheminformatics

Most often, data and information about chemical compounds is either directly about molecular structure (for example, a 2D structural formula, or 3D atomic coordinates for a particular conformation of a compound), or is tied to a molecular structure (for example, physical properties of a compound, which you identify by its structural formula). The notion of indexing, sorting, searching and retrieving information using *molecular structures* originated within the domain of modern chemistry.

Almost all chemists engage in communication tasks to register, search, view, and publish molecular structures. Most forms of chemical representation were developed with these uses in mind. Cheminformatics involves storing, finding, and analyzing these structures using the data-processing power of computers to match chemical compounds with literature publications, measured properties, synthetic procedures, spectra, and computational studies. To do this work, computers need to use chemical representation to identify, exchange and validate information about chemical compounds.

In order for (human) chemists to rely on insights from cheminformatics, it is important to understand the way in which computers store and analyze chemical structure, the methods that computer programs employ, and the results that they produce. Therefore, cheminformatics depends upon the use of representations of molecular structures and related data that are understandable both to **human scientists** and to **machine algorithms**.

Formulating Chemical Structure Data

Interacting with a machine is a form of communication. How does communication between chemists differ from communication between a chemist and a machine? In cheminformatics, you are working within a system governed by strict rules that are explicitly defined. If you know the rules, then you can make the system work for you. If you don't know the rules for a given form of representation, sometimes features designed to satisfy the requirements in one context will appear as bugs in another context.

If one chemist was to recommend to another that a reaction should be performed using "chloroform" as a solvent for a reaction, this would generally be a successful exercise in communication. For all practical purposes, this word is understood by every chemist, and has no ambiguity. However, because "chloroform" is a so-called *trivial name*, there is no formula for converting it into the actual chemical structure that it represents, and a machine will not be able to participate in this exchange of information unless it has been explicitly instructed as to the chemical structure that this word represents, expressed in a format that the machine can work with.

A more descriptive way to communicate the composition that is chloroform is by chemical formula, in this case $CHCl_3$. A computer program could interpret basic molecular structure rules to determine that the substance being described has 5 atoms: 1 carbon, 1 hydrogen and 3 chlorine. Assembling this into a molecule with bonds can be based on valence rules, identifying 4 of the atoms as normally monovalent and one as normally tetravalent. It is quite simple to create a software algorithm that can join the atoms together in the most obvious way, which also happens to be correct.

Beyond such tiny simple molecules, difficulties soon arise. Some of these ambiguities affect human chemists in the same way that they affect machines. Consider the molecular formula of C_3H_6O, which is associated with multiple reasonable structures, including a ketone, an aldehyde, a cyclic alcohol, oxygenated alkenes and cyclic ethers, one of which exists as two enantiomers:

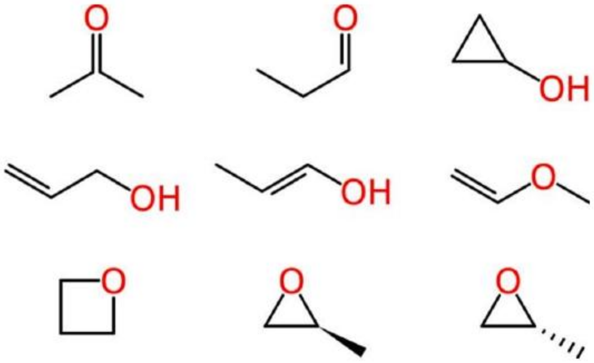


Figure 2.1.1: Different ways of drawing C_3H_6O (Image credit: Evan Hepler-Smith)

Ambiguous representations can refer to more than one chemical entity. This is true of most chemical names when used unsystematically, such as “octane,” when employed as a common term for all saturated hydrocarbons with eight carbon atoms, rather than systematically to indicate the straight-chain isomer only. Empirical and molecular formulas are also typically ambiguous.

In an **unambiguous** system of representation, each name or formula refers to exactly one chemical entity, typically in a way that allows you to draw a structural formula for it. However, each chemical entity might be represented by more than one name or formula. A **canonical** form is a completely unique representation within a system. For example, “diethyl ketone” and “3-pentanone” are both unambiguous names: each represents one and only one compound. However, since they represent the *same* compound, they are not unique names. Within the system of Preferred IUPAC Names (see below), “3-pentanone” is a canonical name – an unambiguous *and* unique representation of this compound.

Note that, since canonical names are necessarily canonical within a system, they might not function properly if you are interested in structural information that is not addressed within the system, or if you do not have structural information that is required by the system. For example, within a system that does not address stereochemistry, the different enantiomers of a chiral compound will have the same “canonical” representation. Within a system that requires the specification of stereochemistry, on the other hand, you will have to choose between stereospecific canonical representations. If you happen to be working with a racemic mixture or a compound of unknown stereo configuration, this may lead to misrepresentation and misunderstanding.

A chemical structure representation contains two kinds of information: **explicit** and **implicit**. **Explicit** information is what’s directly represented in a data structure and should at minimum contain what otherwise would not be known, such as the specific atom in a carbon skeleton to which a substituent is attached. **Implicit** information is what you (or a computer) can figure out from a data structure, given some knowledge of general principles and a little bit of work.

In general, data structures that contain less explicit information are more simple and compact, but they require more computation to draw chemical conclusions from them. Data structures that contain more explicit information take up more space and are at greater risk of containing inconsistencies, but they can be more quickly analyzed in a wider variety of ways.

To automate functions on chemical data, the data structure needs to be **systematically** defined and consistently applied. These definitions are part of what constitutes explicit information that an algorithm can readily identify and parse. Balancing the level of explicit information can also impact the ambiguity of a system, and the ability to accurately exchange chemical structures between systems. These are especially important considerations for operations that range across a significant portion of the corpus of reported chemical compounds (well over 100 million), beyond the scale at which human validation of results is possible.

Representing Chemical Structure

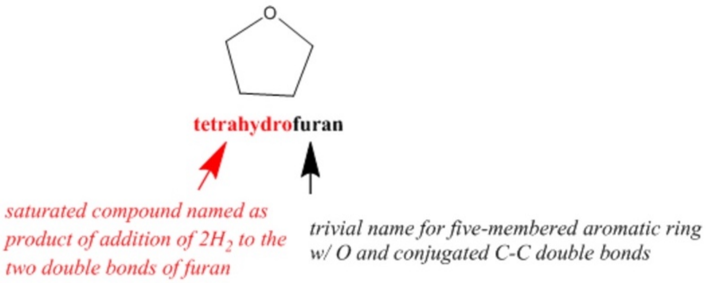
Structural Formula

Generally, the most effective way to communicate with another chemist about the structure of a compound is to draw its structural formula. A **structural formula** is any formula that indicates the connectivity of a compound – that is, which of its atoms are linked to each other by covalent bonds. Unfortunately, structural formula are most valuable for small molecules as they can get to complex as the size of the molecule increases. On the other hand, a computer does not "see" a formula like a human does, but "reads" it as a form of data, and we will look at two data structures that computers can "read", connection tables and line notations.

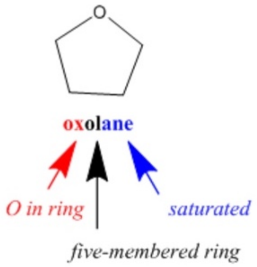
Systematic Names

Systematic names describe the structural formula of compounds. If you know the rules and vocabulary, you should be able to write a name based on a structural formula and vice-versa. Chemists have developed various ways of translating formulas into names, so it is nearly always possible to write more than one systematic name for a given compound.

IUPAC (International Union of Pure and Applied Chemistry) nomenclature is a well-known international system of chemical names that is generally systematic but flexible, allowing the use of certain well-established trivial names. Since systematic IUPAC names are made according to formalized rules, they could, in principle, be used by both humans and computers. However, IUPAC names are often quite difficult for chemists to read, let alone to write, and the rules are non-canonical, resulting in numerous different options for naming each compound. IUPAC has introduced even more rules for determining canonical Preferred IUPAC Names (PINs) that are oriented toward making systematic names more easily readable by machines.


O
tetrahydrofuran
saturated compound named as product of addition of 2H$_2$ to the two double bonds of furan
trivial name for five-membered aromatic ring w/ O and conjugated C-C double bonds

Most frequently-used systematic name, often abbreviated THF



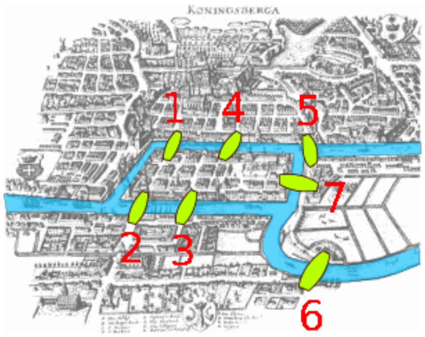
New Preferred IUPAC Name; less familiar, but eaiser for a computer to parse

Semantic technologies further enable systematic classification and organization of scientific terms, including descriptions of chemical structures, such as provided by ChEBI (Chemical Entities of Biological Interest). ChEBI describes small molecular entities based on nomenclature, symbolism and terminology endorsed by IUPAC and the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (NC-IUBMB). This dataset is highly curated by both human experts and machine processes, is openly searchable and programmatically accessible, and includes full references to original authoritative sources.

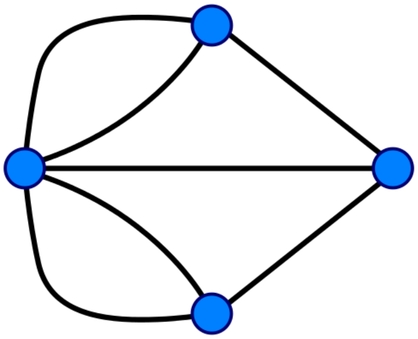
Copyright Statement

Graphic Visualizations

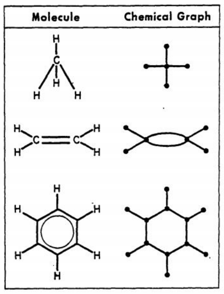
Cheminformatics takes advantage of the mathematical discipline of graph theory when representing and comparing chemical structures. A graph represents the relationship between two things and graph theory involves the pair-wise relationship between two objects, where the object is a node (vertice or point on the graph) and the connection between the nodes are the edges (links or lines) of the graph. In chemistry the atoms are the vertices and the bonds are the edges. In fact you use graph theory when you use Google Maps to choose a route between two cities, where the cities are the vertices and the roads connecting them are the edges.



Konigsberg Bridges on map (credit: Maksim, Wikimedia Commons)



Konigsberg Bridge Problem in terms of graph (credit: Riojajar, Wikimedia Commons)



(credit: Office of Naval Research, Technical Report No. 41, An introduction to Graph Theory, D.H. Rouvray)

Figure 2.1.2: On the left is a map of Konigsberg (left), a graph describing the map (middle) and some simple molecules and their graphs (right).

In 1736 Leonhard Euler formulated the foundations of graph theory when he tackled the Konigsberg bridge problem, which was to determine if you could walk across every bridge to this island in the city of Knogsberg just once and walk across all of the bridges, (and he proved that you could not). Mathematically, Euler treated the land masses as the nodes and the bridges as the edges that link the nodes. In 1878 the mathematician James Sylvester introduced the concept of the chemicograph in his Journal of Nature article "Chemistry and Algebra", the same year he published the chemicographs of figure 3 in Volume 1, No. 1 of his American Journal of Mathematics article "On an Application of the New Atomic Theory to the Graphical Representation of the Invariants and Covariants of Binary Quantics, with Three Appendices". In Sylvester's chemicograph the atoms became the nodes and the covalent bonds the edges, and note, a double or triple bond were treated like having two or three edges connecting the nodes (atoms). One can quickly see a relationship between these and the Lewis Dot structures chemistry students cover in high school and freshmen chemistry, but as we shall see, computers can handle structures much more complicated than we can draw on a paper.

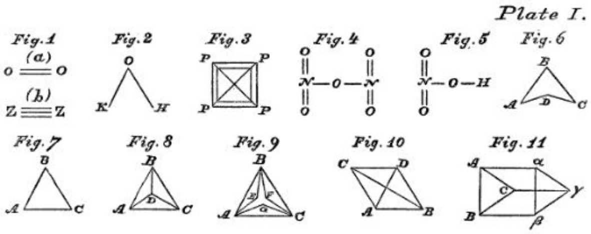


Figure 2.1.3: First eleven of forty five chemicographs from Sylvester's 1878 article on the Application of the New Atomic Theory to the Graphical Representation of the Invariants and Covariants of Binary Quantics, with Three Appendices".

One of the advantages to graph theory is it can be used to determine if two graphs have a one-to-one mapping of nodes and edges, that is if they are isomorphic (identical), and if a subgraph of one graph is isomorphic to a subgraph of another, those parts are identical. Although this elementary introductory course will not delve into graph theory, it is important that students understand the basic data structures that graph theory based algorithms use, and yes, we will be using those algorithms.

Chemical Graphs on Computers

Connection tables

A connection tables does for computers what systematic nomenclature does for human chemists: they organize structural information defined in a molecular graph in a form that is machine readable. The difference is that computers can read, sort, search, and group connection tables far faster than humans can work with systematic names or any other kind of formula or notation. Connection tables essentially provide information on the atoms in a molecule, where the bonds are, and what types of bonds there are. They are covered in more depth in section 2.2 and there are many types of structural data files that use connection tables (section 2.5). Besides connection tables, other common forms of machine-readable representations are graphic visualizations, line notations, and other descriptive forms such as nomenclature.

Chemists most frequently think about chemical structure in 2D, although molecules actually exist in 3D physical space. Most chemical data systems offer 2D and 3D visualizations that human chemists can use in searching and analysis. The 2D coordinates stored in a connection table can be used to infer and display chemical information, including the basic structural formula and additional information such as the E/Z geometry of alkene-like double bonds, the cis/trans isomerism of ligands in a square planar metal complex, or substituents on a cyclic alkane. 2D representations are designed to mimic the experience of drawing structural formulas on paper. Human often convert these electronic drawings two images files for use in publications and presentations, but these image files (jpeg, gif, ping,....) are no longer connected directly to chemical data and are thus not machine readable.

3D (x,y,z) coordinates can also be stored for each atom and used to display the *conformation* of a molecule. These coordinates may be determined experimentally (typically via x-ray crystallography), or calculated (using force-fields, quantum chemistry, molecular dynamics or composite models such as docking). Understanding a molecule's actual shape, whether it be in solution, in a vacuum, or in the binding site of a protein, opens up a whole new domain of computational chemistry. Most molecules have some flexibility, and even if a given conformation is the most stable, there are often a number of competing shapes to consider. Knowing how a particular set of coordinates was determined is crucial to making intelligent use of it for cheminformatics purposes.

Line Notations

Line notations represent chemical structures as a linear string of symbolic characters that can be interpreted by systematic rule sets and will be covered in section 2.3. Line notation could be considered as nomenclature for computers, as like a connection table a computer can "read" a line notation and develop a molecule the same way a human can read IUPAC nomenclature and generate the molecule. Many forms of line notation are both machine and human readable.

Line notation is widely used in Cheminformatics because:

a. many computational processes operate more effectively on data structured as linear strings than data structured as tables.
b. line notations can be reasonably legible to human chemists designing functions with these tools.

Linear representations are particularly well-suited to many identification and characterization functions, such as determining:

- whether molecules are the same;
- how similar they are, according to some metric;
- whether one molecular entity is a substructure of another;
- whether two molecules are related by a specific transformation;
- what happens when molecules are cut into pieces and grafted together at different positions.

In these and other applications of cheminformatics, linear line notation representations have key advantages for speed and automation, especially when you’d like to handle huge numbers of structures (e.g. searching a large database).

Examples of line notations include the Wiswesser Line-Formula Notation (WLN), Sybyl Line Notation (SLN) and Representation of structure diagram arranged linearly (ROSDAL). Currently, the most widely used linear notations are the Simplified Molecular-Input Line-Entry System (SMILES) and the IUPAC Chemical Identifier (InChI). In this class we will focus on SMILES and InChI line notation.

Contributors

Robert E. Belford (University of Arkansas Little Rock; Department of Chemistry). The breadth, depth and veracity of this work is the responsibility of Robert E. Belford, rebelford@ualr.edu. You should contact him if you have any concerns. This material has both original contributions, and content built upon prior contributions of the LibreTexts Community and other resources, including but not limited to:

- Evan Hepler-Smith
- Leah R. McEwen
- Acknowledgements: Alex Clark, Sunghwan Kim

(Material adapted from Spring 2017 Cheminformatics OLCC)

2.2: Connection Tables

Learning Objectives:

- Introduce concept of Connection Table
- Introduce shortcomings of Simplified Connection Table

Introduction

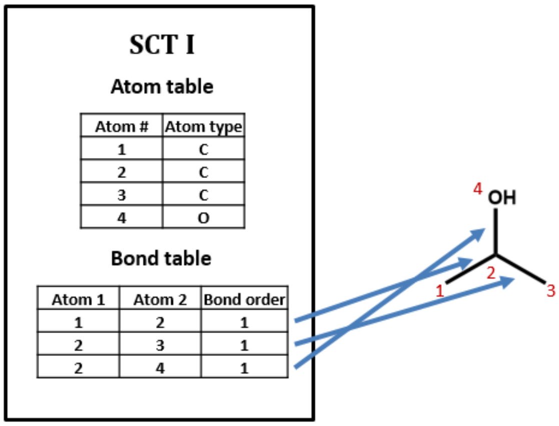
A connection table is a data table that provides information a computer needs to generate a molecular graph. Specifically, it needs to define what the atoms are, and how they are connected, that is, the edges and the nodes of the graph. This is typically done in a file, and in the file are a minimum of two tables, the atom table and the bond table. But the file can also have additional information, like what isotopes are present, or what are the 3-D coordinates for a particular conformation.

Thus, this section will be broken up into several parts. First, we will discuss a Simplified Connection Table (SCT) to get a feel for the basic logic behind a connection table. Then we will look at some real files that students will be required to download from databases and work with.

Simplified Connection Table (SCT)

The purpose of this section is to give the student a feel for the issues associated with creating a connection table that can generate a molecular graph. **The SCT is not a real file,** but a description of the data within a file that the computer needs to be able to "read". Real files using connection tables will be approached in section 2.5 Structural Data Files. In essence, there are two tables, the atom table and the bond table.

- **The atom table** provides an index number for each atom. It may provide index numbers for the hydrogen (explicit representations), or it may not (implicit representations).
- **The bond table** uses the index number of the atom table to define what atoms are bonded to each other, and the types of bonds. The bond order is defined by a number, where 1 is a single bond, 2 is a double bond and 3 is a triple bond.



SCT I

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	O

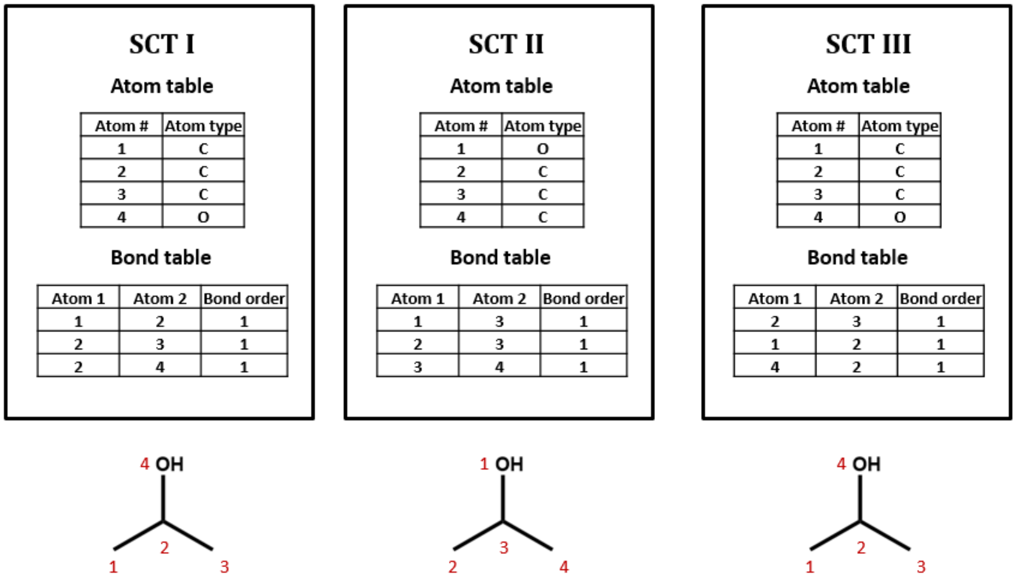
Bond table

Atom 1	Atom 2	Bond order
1	2	1
2	3	1
2	4	1

Figure 2.2.1: SCT for isopropanol.

Are connection tables unique?

No, there can be many ways of assigning index numbers to the atoms in a connection table, and one would need to use an algorithm for generating a canonical connection table. For n atoms there are n! (n factorial) ways of assigning index numbers. From figure 2.2.2, if any of the four atoms is assigned the value of 1, there are 6 ways you can assign the remaining three atoms, and since there are 4 atoms you can assign the value of 1 to, there are 24 ways to assign the above index numbers, which is 4!.



SCT I

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	O

Bond table

Atom 1	Atom 2	Bond order
1	2	1
2	3	1
2	4	1

SCT II

Atom table

Atom #	Atom type
1	O
2	C
3	C
4	C

Bond table

Atom 1	Atom 2	Bond order
1	3	1
2	3	1
3	4	1

SCT III

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	O

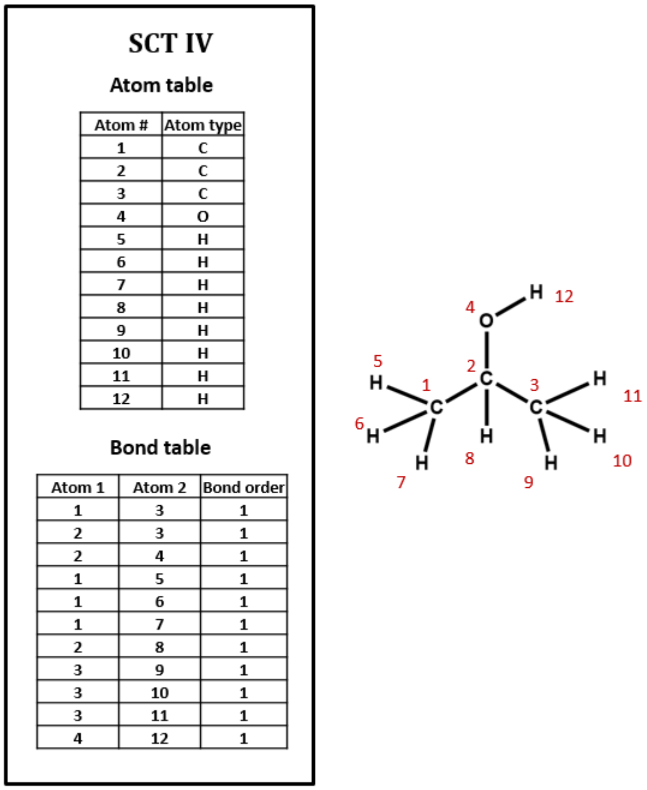
Bond table

Atom 1	Atom 2	Bond order
2	3	1
1	2	1
4	2	1

Figure $2.2.2$: Three possible ways of assigning index numbers to a connection table for isopropanol.

Was the above connection table explicit or implicit in the assignment of hydrogens?

The above connection table is implicit in the assignment of hydrogens, and algorithms based on a set of valency rules could determine the number of hydrogens. Figure 2.2.3 has explicit hydrogens for isopropanol.



SCT IV

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	O
5	H
6	H
7	H
8	H
9	H
10	H
11	H
12	H

Bond table

Atom 1	Atom 2	Bond order
1	3	1
2	3	1
2	4	1
1	5	1
1	6	1
1	7	1
2	8	1
3	9	1
3	10	1
3	11	1
4	12	1

Figure $2.2.3$: Isopropanol connection table with explicit hydrogens.

By comparing figures 2.2.2 and 2.2.3 you can see how the implicit table is simpler and would result in a smaller file.

Exercise 2.2.1

Can you see what is wrong with the Bond table in figure 2.2.3?

Answer

Note that atom 2 is only in three bonds, and atom three is in five bonds, and that somehow, atom 3 is attached to atom 1.

SCT Shortcomings

The Simplified Connection Table is at the core of graph theory representation of chemical entities in that it provides the critical information of what atoms are present and where the bonds are. In section 2.5 we will look at several structural data files that are built around connection tables, but before that it is prudent to identify many of the issues with molecular representation that a crude SCT does not cover, and these will be outlined in section 2.3. There are sort of three basic types of shortcomings to connection tables.

The first is that real molecules are dynamic 3 dimensional structures and so a real data file needs to define the relative coordinates of the atoms, which is typically done by adding a 3D coordinate layer to the atom table (section 2.5). The fact that molecules are dynamic means the bonds are both vibrating and rotating, with the later resulting in multiple conformations (different orientations) of the atoms over time. Atomic coordinates typically represent the most stable orientation as determined through computational calculations that minimize the energy of the system and take into account environmental factors. This means the coordinates of a molecule in free space may be completely different than in a protein environment, and so it is important that you understand how the coordinates in a molecular data file were generated, and if they are appropriate to your needs.

The second shortcoming to connection tables is actually structural in nature, and often a SCT does not provide enough information to uniquely describe a molecular species. In the case of isopropanol above, there were 24 connection tables that described the molecule, but they all described the same molecule, that is, they described isoproponal. Sterioisomerism is a case where two different molecules (say cis and trans dichlorethene) would have the same connection table, and so stereo isomers would require additional information to uniquely distinguish between the two isomers. Other areas where issues with SCT arise include resonance structures, aromaticity, tautomers, multcovalent units, coordination complexes, conjugate acid/base equilibrium and the like. To handle these issues additional data beyond that of the SCT. These will be discussed in section 2.3 Molecular Graph Issues.

The third potential issue with connection table based representation of chemicals is more functional in nature, in that they are bulky files and hard to read by a human. Line notation is a string of characters, like a word, which describes the molecule. One could consider line notation to be nomenclature for computers in the sense that a string of characters represents a :word". In reality, line notations are often converted to connection tables when software agents are doing calculations because many of the software algorithms are based on manipulating connection tables.

The shortcomings of connection tables will be picked up again when we look at real structural data files and how they handle these situations.

Contributors

Robert E. Belford (University of Arkansas Little Rock; Department of Chemistry). The breadth, depth and veracity of this work is the responsibility of Robert E. Belford, rebelford@ualr.edu. You should contact him if you have any concerns. This material has both original contributions, and content built upon prior contributions of the LibreTexts Community and other resources, including but not limited to:

- Evan Hepler-Smith
- Leah McEwen

(Material adopted from the Spring 2017 Cheminformatics OLCC)

2.3: Molecular Graph Issues

Learning Objectives:

- Identify issues with representing molecules as molecular graphs
- Review chemical principles that can provide issues with structural data

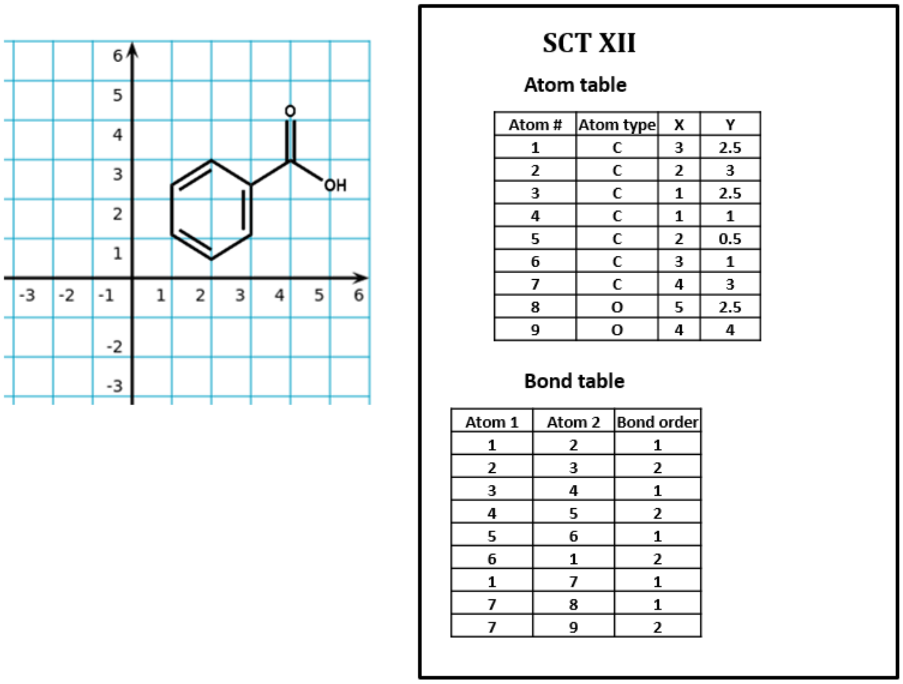
Introduction

This section is intended to be a review for chemistry students, while also assisting non-chemistry majors in identifying some of the complexities of chemistry that come into play when describing molecular structures. Even chemistry students are advised to quickly skim through this material, as we are parsing common topics from the perspective of a Simple Connection Table (SCT), which may give them a deeper understanding of the nuances of chemical structure representation.

As many of these issues require features beyond those of the simple connection table, many of these issues will be revisited when we look at real data files in section 2.5 Structural Data Files.

Atom Coordinates

Note that the SCT atom table does not tell you anything about the relative position of atoms. (As we have seen, you often have to go to the bond table just to figure out which atom is which.) Many connection table formats contain two- or three-dimensional spatial coordinates for each atom entry. These coordinates may simply record the relative position of atoms in a structural formula sketched in a chemical drawing program (*SCT XII*).



SCT XII

Atom table

Atom #	Atom type	X	Y
1	C	3	2.5
2	C	2	3
3	C	1	2.5
4	C	1	1
5	C	2	0.5
6	C	3	1
7	C	4	3
8	O	5	2.5
9	O	4	4

Bond table

Atom 1	Atom 2	Bond order
1	2	1
2	3	2
3	4	1
4	5	2
5	6	1
6	1	2
1	7	1
7	8	1
7	9	2

Figure $2.3.1$: The addition of two dimensional coordinates added to a connection table based on an external coordinate system, a 3D system would have an additional column for Z values.

We will address atom coordinates in section 2.5 when we look at real structural data files.

Stereochemistry

Isomers are different molecules with the same atomic constituency, that is they have the same number of atoms for each element and the atom tables are essentially identical, (the numbering of the atoms may be different, but the two atom tables are isomorphic). There are two basic types of isomers, constitutional isomers and stereoisomers. Constitutional isomers are also called structural isomers and have different bond connectivity for the same atoms. This means they have different (non-isomorphic) bond tables, and so the Simple Connection Table (SCT) has no problem distinguishing constitutional isomers. Stereoisomers have the same (isomorphic) SCTs, that is, both the atom and the bond table are essentially the same, (the atom numbering may differ, but this is reflected in the bond connections and so the SCTs are essentially the same (isomorphic). What distinguishes the atoms of steriochemical isomers is the atomic arrangement in space, not the connections.

You may ask why is this important? One example often used in textbooks is the biological significance of two sterioisomers of thalidomide, a chemical used as an antidepressant for pregnant mothers in the 1960s. In synthesizing the chemical the "drug" was actually a mixture of both isomers, one of which was an effective medication and the other of which caused horrific birth defects. This was clearly an "unintended consequence" and one of the most important functions of cheminformatics is to help scientists identify unintended effects of potential drugs by looking at a multiplicity of bioassays, including toxicological screening assays.

Figure 2.3.2: Birth defect caused by the mutagenic isotope of thalidomide, which was prescribed by the mother's doctor.

Isomer Review

First it is prudent to review isomers. Figure 2.3.3 gives an flowchart of the major type of isomers chemists deal with. This section is dealing with the issues of representing stereochemistry in simplified connection tables and so will focus on that. This section will also introduce conformational isomers, which are different pseudo-stable orientations around rotatable bonds and so are not really separate molecules but can be of interest to cheminformatics because the different conformations can be favored in different environments, and exhibit different chemical behaviors (the most probable conformation in a protein environment may be different than in free space).

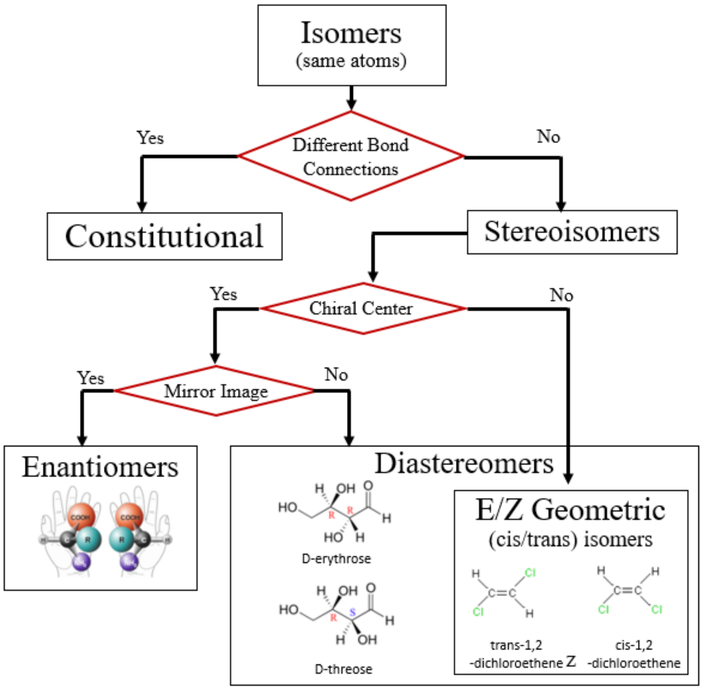
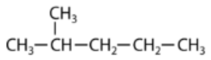
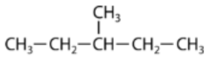


Figure 2.3.3: Flowchart of the common types of isomers that

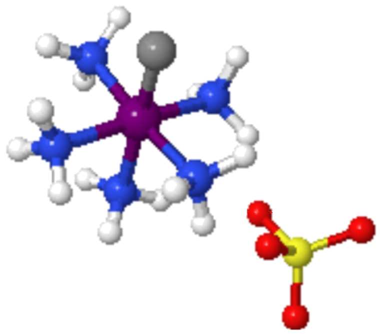
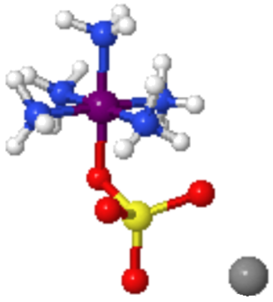
Constitutional Isomers

Constitutional or structural isomers are molecules composed of the same atoms but with different bonds (connectivity). There are two subsets to constitutional isomers, linkage isomers and ionization sphere isomers. Linkage isomers are simple ones within a molecule where the bond links are different, and so they can not have the same bond table.


2-methylpentane


3-methylpentane

Ionization sphere isomers are multi-covalent unit coordination complexes (salts) where the neutral salt that is composed of a positive cation and a negative anion have the same formula, but in the two isomers the anions exchange with a ligand. These types of species present challenges for connection tables and these issues will be approached in the multicovalent unit and coordination complex sections of this chapter.

	
$[Cr(NH_3)_5Br]SO_4$	$[Cr(NH_3)_5(SO_4)]$ Br

2.3.4: On the left the bromine ligand (grey) is bonded to the Chromium of the pentaamminebromoChromium(III) cation and the sulfate is the anion, while on the right the bromine is the anion and the "sulfate" is the ligand of the pentaamminebromosulfato(III) cation.

Conformers

These are often called conformational isomers but they do not represent true isomers, but are often of great importance in cheminformatic investigations. If you have a single bond a molecule can freely rotate, and as it rotates the groups attached to the rotating atom change their geometric positions and thus change their interaction energies both with respect to other atoms in the molecule, but also with respect to atoms in their environment. There are approaching an infinity of conformations a molecule posses and these can be understood by looking at a simple hydrocarbon, ethane (CH_3CH_3) where there is free rotation around the C-C bond. Ethane has two extreme conformations, the eclipsed and the staggered which can be visualized by the Newman projection (right of figure) where you are looking down the C-C bond axis.

These are not really isomers in that it is the same molecule and the following animated gif shows the potential energy changing for an isolated molecule of ethane as it goes through these rotations.

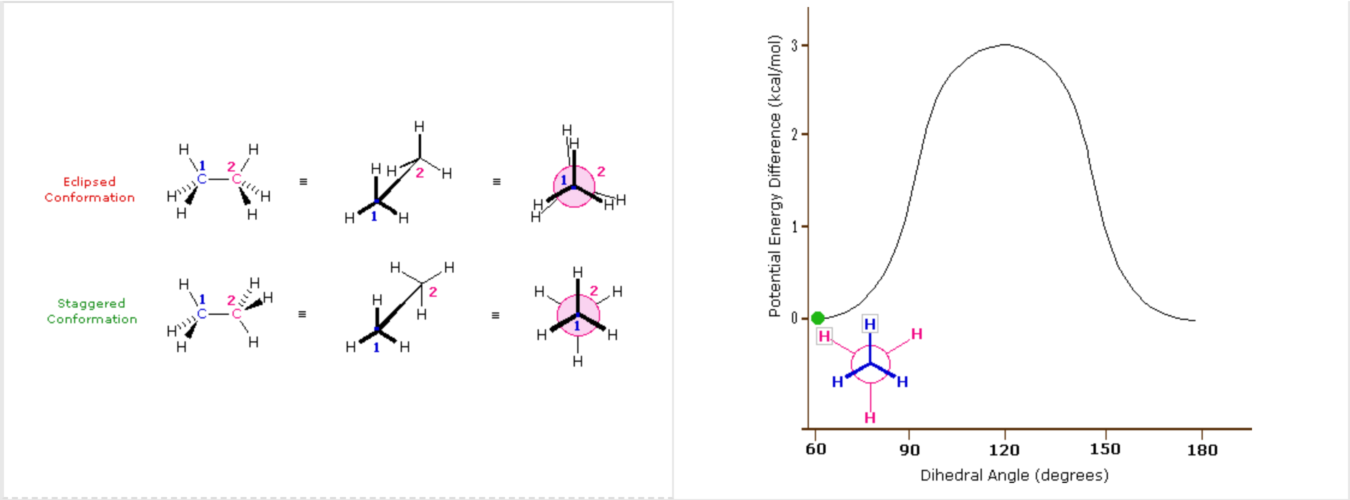


Figure 2.3.5: Perspective and Newman drawing of the two extreme conformations of ethane (left) and an animated gif showing the potential energy change as a result of rotation around the C-C bond (right).

What is important to recognize is that a connection table will often give the coordinates of the atoms in a molecule, and in reality the coordinates change as the molecule transitions from one conformer to another. But also, the above diagram represents a simple isolated molecule, and in real systems, molecules are often in protein environments that will define the most stable state, and also the reactivity of a molecule will depend on its conformational state. So to truly represent a molecule on computer one needs to take into account a multiplicity of conformational states, all of which can be represented in a connection table.

Stereoisomers

Stereoisomers have isomorphic simplified connection tables (they have the same connectivity) but differ in the arrangement in space, and so need additional information to distinguish them. There are two basic types of stereoisomers, enantiomers and diastereoisomers. If two nonsuperimposible stereoisomers are mirror images of each other they are enantiomers, and if two stereoisomers are not mirror images of each other they are diastereoisomers.

Enantiomers

Enantiomers are chiral molecules, that is if you invert the molecule through a mirror plan it is not superimposible on its mirror image. If it is superimposible, then it is the same molecule as its mirror image and they are not isomers. The term chirality comes from the Greek word for handedness, and your hand is a chiral structure. If a carbon atom in an organic molecule is attached to four different substituents it is a chiral atom. Bromochlorofluoromethane has a nonsuperimposible mirror image across the chiral carbon and so is a chiral molecule, while dichlorofluoromethane is superimposible on its mirror image, and therefore is not an isomer with its mirror image.

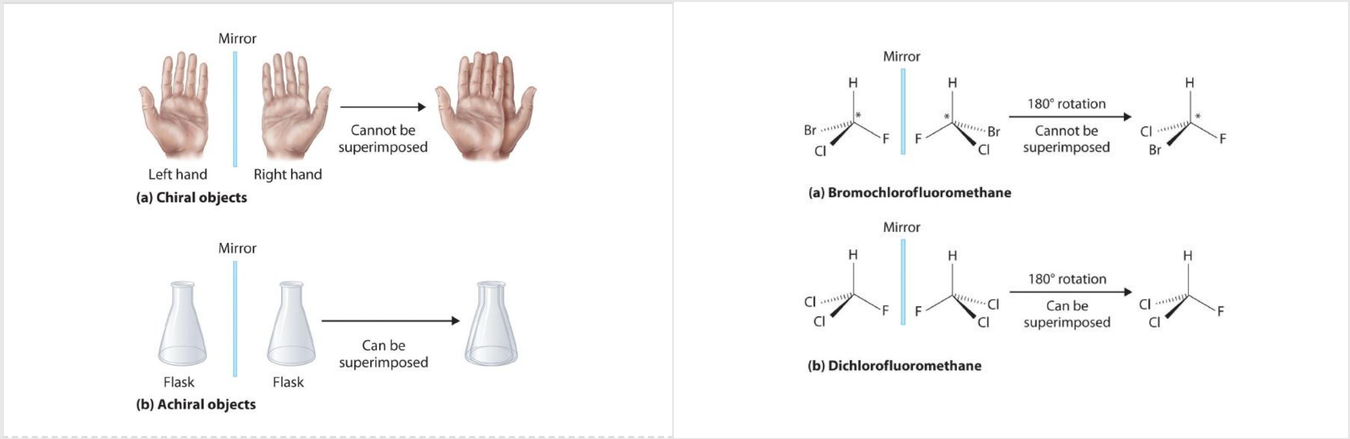


Figure 2.3.6: The top objects are chiral and the bottom are not. These means that there are two different types of bromochlorofluoromethane (the "left" and the "right" handed ones), while there is only one type of dichlorofluoromethane (because it is superimposable on its mirror image, and so it is the same molecule as its mirror image).

Chirality is a function of structure and so there are chiral centers (the plane going through the central carbon, hydrogen and fluorine in the above example was one such center of inversion). These can get very complicated very quickly, but one thing to note is that in an organic molecule a carbon that is bonded to four other atoms is chiral if none of the groups bonded to it are identical (note in the bottom part of figure 2.3.6 two of the groups are identical (Cl), and so it was not chiral.

Lets look at thalidomide, the molecule responsible for the birth defects in figure 2.3.2. Figure 2.3.7 shows four ways of drawing this structure.

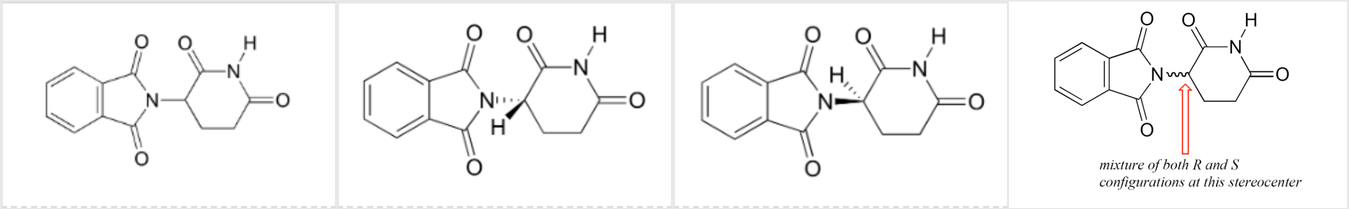


Figure 2.3.7: Four ways of drawing thalidomide, each means something different. Sequentially (left to right) these are; undefined, left-hand (S), right-hand(R) and a mixture. See ICP rules (below) to understand R and S notation.

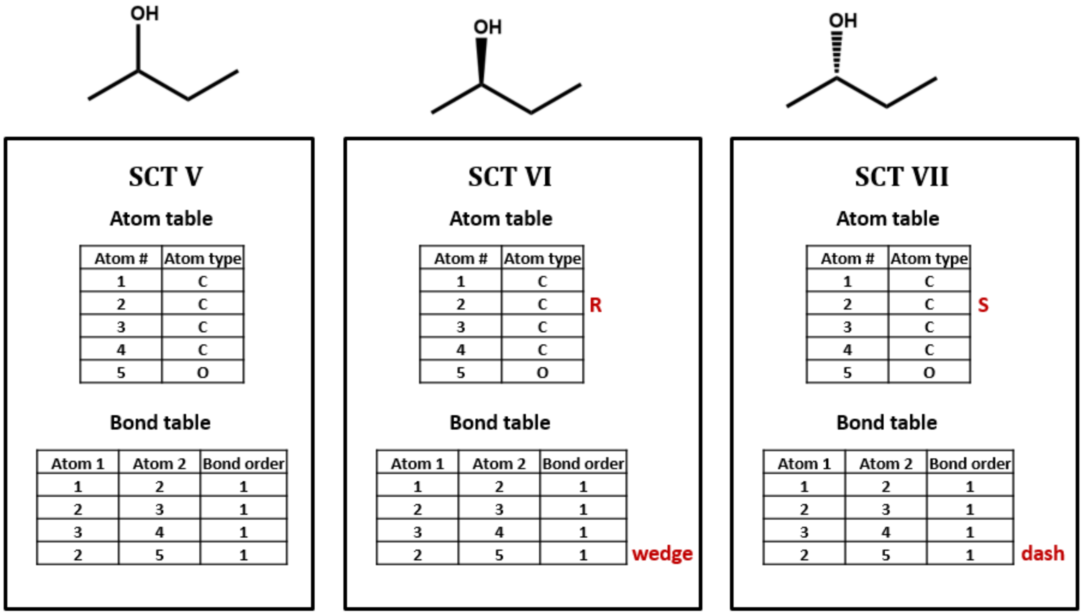
Figure 2.3.7 represents a major challenge in cheminformatics in that authors often do not define the stereochemistry of molecules when they submit structures in their publications (left most image) and this can lead to issues when data is abstracted from the literature.

Before going into how we identify chirality we should first take a look at the "Simplified Connection Table" (SCT) to understand the issue with respect to representing chiral molecules on computer.

Chirality and Connection Tables

VERY IMPORTANT: In this class we will be using chemical compound databases to retrieve and store chemical information and in the following two figures you see three connection tables for theses stereoisomers. As pointed out in figure 2.3.7, when someone measures and uploads data concerning a chiral compound there are actually four possible identities, the stereo chemistry is not defined (left most image), it is defined (middle and right images) or it is a mixture (right image).

Lets look at the connection table of the chiral compound 2-butanol (figure 2.3.8. If you look at the SCT, they are all the same. That is, we need to add additional information, and when we get to actual chemical structure files we will look deeper into this.



SCT V

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	C
5	O

Bond table

Atom 1	Atom 2	Bond order
1	2	1
2	3	1
3	4	1
2	5	1

SCT VI

Atom table

Atom #	Atom type	
1	C	
2	C	R
3	C	
4	C	
5	O	

Bond table

Atom 1	Atom 2	Bond order	
1	2	1	
2	3	1	
3	4	1	
2	5	1	wedge

SCT VII

Atom table

Atom #	Atom type	
1	C	
2	C	S
3	C	
4	C	
5	O	

Bond table

Atom 1	Atom 2	Bond order	
1	2	1	
2	3	1	
3	4	1	
2	5	1	dash

Figure 2.3.8: Three ways of representing 2-butanol. Beyond the information of the SCT, we need to add the material in red, identifying which atom is chiral, and which bond goes into the plane (dash) and out of the plane (wedge) of the drawing.

Cahn-Ingold-Prelog (CIP) Nomenclature

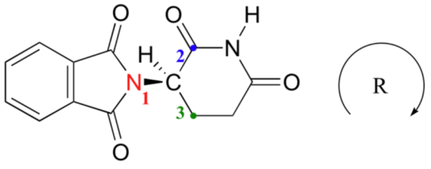
Chiral molecules are distinguished by the letters R- (rectus, Latin for Right handed) and S- (sinister, Latin for left handed). The rules used for identifying chiral centers are the Cahn-Ingold-Prelog (CIP) rules. A review of organic chemistry may be required as these can get a bit complicated, but here is the jist using the bromochlorofluoromethane molecule of figure 2.3.6 as an example.

Step 1: Identify chiral atom(s) and rank others by order of priority of atomic mass (number 1 is largest, 3 is smallest) (Br=1, Cl=2, F=3 H=4

Step 2: Place your thumb of either the right of left hand along the axis of the chiral carbon towards the atom of smallest priority (H here

Step 3, Starting with the atom of highest priority, the fingers of one of your hands will point in the direction of next highest (sequentially decreasing priority), and that hand tells you if it is R or S. So the image on the left of the mirror is R-Bromchlorofluoromethane and the one on the right is the S-Bromochlorofluoromethane.

Now applying this to the middle right image of figure 2.3.7 shows that image is the R isomer.



(R)-thalidomide

Figure 2.3.8: Applying CIP rules to the middle right thalidomide structure in figure 2.3.7.

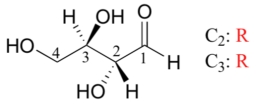
NOTE: If an atom has one chiral center it must be chiral, if it has more than one, it may or may not be chiral. Compounds with more than one stereocenter may be either an enantiomer or a diastereomer.

Diasteriomers

These are stereoisomers that are not mirror images of each other. There are two types, geometric and those with chiral centers.

Compounds with Multiple Chiral Centers

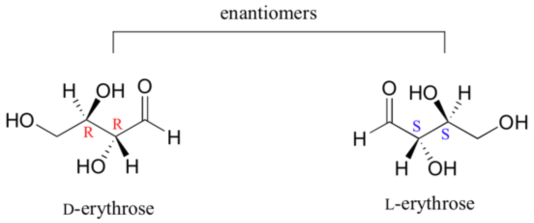
We turn our attention next to molecules which have more than one stereocenter. We will start with a common four-carbon sugar called D-erythrose.



D-erythrose

A note on sugar nomenclature: biochemists use a special system to refer to the stereochemistry of sugar molecules, employing names of historical origin in addition to the designators '*D*' and '*L*'. You will learn about this system if you take a biochemistry class. We will use the *D/L* designations here to refer to different sugars, but we won't worry about learning the system.

As you can see, *D*-erythrose is a chiral molecule: C_2 and C_3 are stereocenters, both of which have the *R* configuration. In addition, you should make a model to convince yourself that it is impossible to find a plane of symmetry through the molecule, regardless of the conformation. Does D-erythrose have an enantiomer? Of course it does – if it is a chiral molecule, it must. The enantiomer of erythrose is its mirror image, and is named L-erythrose (once again, you should use models to convince yourself that these mirror images of erythrose are not superimposable).



Notice that both chiral centers in L-erythrose both have the *S* configuration. *In a pair of enantiomers, **all** of the chiral centers are of the opposite configuration.*

What happens if we draw a stereoisomer of erythrose in which the configuration is *S* at C_2 and *R* at C_3? This stereoisomer, which is a sugar called D-threose, is *not* a mirror image of erythrose. D-threose is a **diastereomer** of both D-erythrose and L-erythrose.

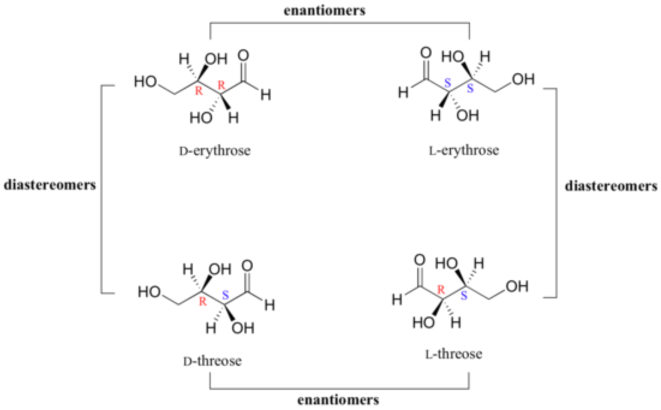


Figure 2.3.10: Looking at the diasteriomers of erythrose. Note that each of the threose is a diasteromer to both of the erythroses. That is, there is at least one steriocenter that is not of opposite configuration.

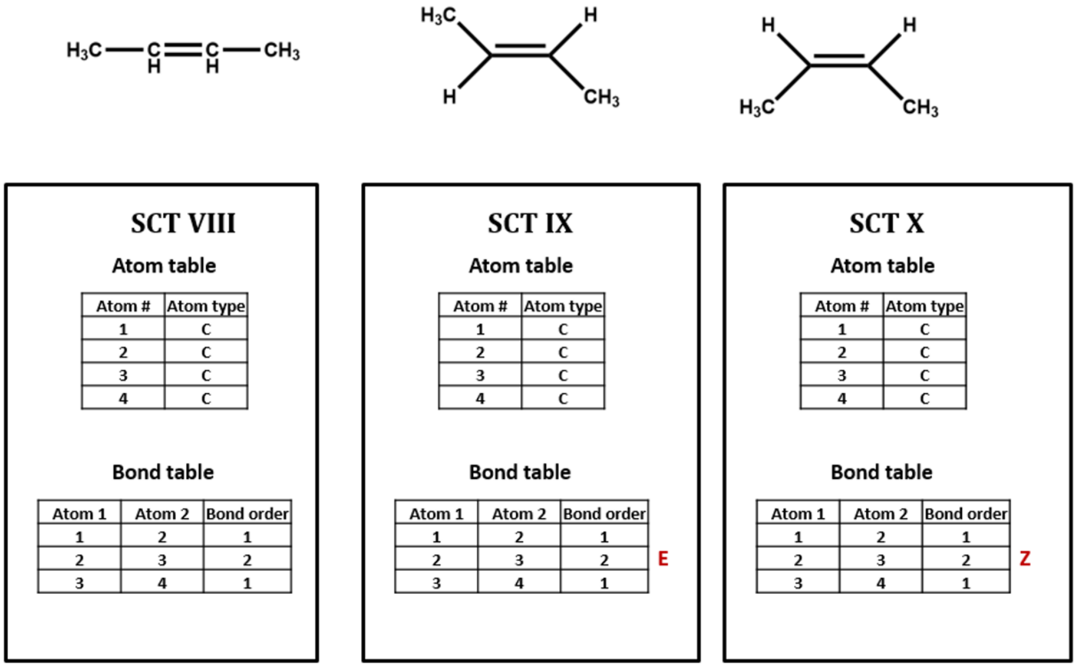
The definition of diastereomers is simple: if two molecules are stereoisomers (same molecular formula, same connectivity, different arrangement of atoms in space) but are *not* enantiomers, then they are diastereomers by default. *In practical terms, this means that at least one - but not all - of the chiral centers are opposite in a pair of diastereomers.* By definition, two molecules that are diastereomers are *not* mirror images of each other.

L-threose, the enantiomer of D-threose, has the *R* configuration at C_2 and the *S* configuration at C_3. L-threose is a diastereomer of both erythrose enantiomers.

In general, a structure with *n* stereocenters will have 2^n different stereoisomers. (We are not considering, for the time being, the stereochemistry of double bonds – that will come later).

Geometric Isomers

Geometric isomers are a type of non-chiral diasteriomers that have the same connectivity but differ in orientation. Figure 2.2.4 shows 2-butene, which is a planar molecule having a 120^{o} bond angle for the carbons attached to the double bond. Since the double bond can not rotate, the orientations are fixed, meaning the hydrogens are on the same side or opposite side. The image on the left does not define the stereoisomerism, while the two on the right do. The middle image has the hydrogens opposite each other, which is classified as the trans or "E" configuration, while the one on the right has them on the same side of the double bond, which is the cis or "Z" configuration.



SCT VIII

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	C

Bond table

Atom 1	Atom 2	Bond order
1	2	1
2	3	2
3	4	1

SCT IX

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	C

Bond table

Atom 1	Atom 2	Bond order	
1	2	1	
2	3	2	E
3	4	1	

SCT X

Atom table

Atom #	Atom type
1	C
2	C
3	C
4	C

Bond table

Atom 1	Atom 2	Bond order	
1	2	1	
2	3	2	Z
3	4	1	

Figure $2.3.11$: Three Lewis dot structures and their connection tables for 2-butene. Not, all three tables have the same SCT, and so a real file would have three options, to define the stereochemistry as E or Z, or not to define it at all.

In the case of geometric isomers physical data can be different, for example, the z isomer may have a slightly different boiling point, and so when someone reports a value and uploads it to the database, it needs to be determined which isomer they had, if they know, or if it is a mixture. So often times databases may have different properties for the same substance because the scientist who made the measurement did not know or report the structure correctly.

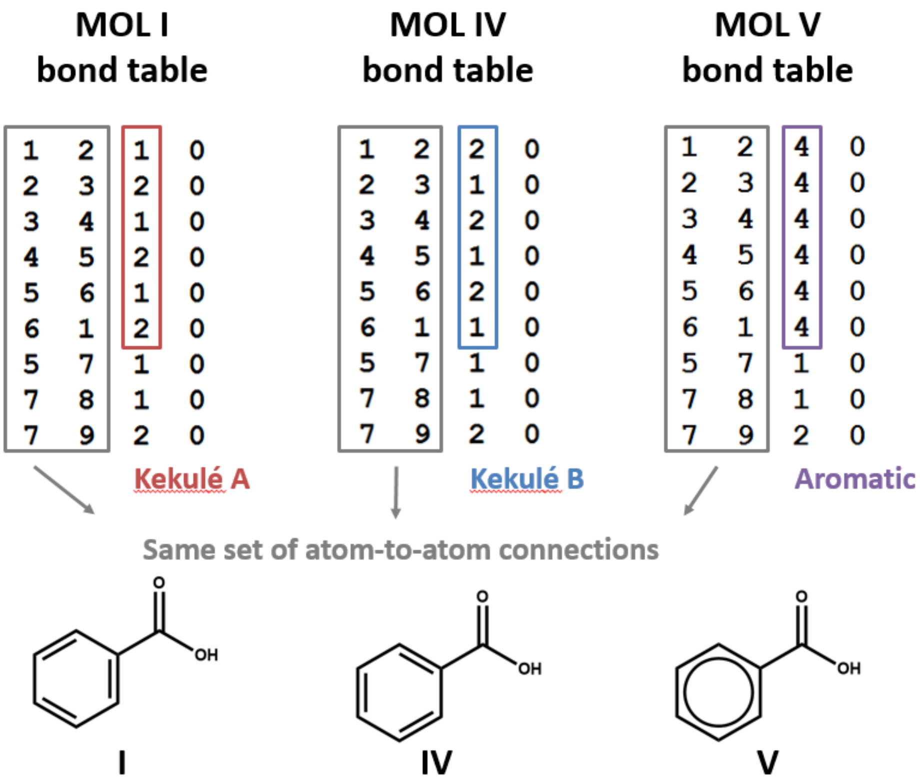
Once again, the SCT can not distinguish these isomers, and you need more information than what the atoms are, and what is bonded to what.

Resonance Structures

Lewis dot structures and connection tables consider a covalent bond to consist of two electrons shared between two nuclei, thus forming a bonding orbital. But many times pi bonds of adjacent atoms can overlap to produce an orbital that involves electrons being shared between three or more nuclei. In this case you can't draw one Lewis dot structure (or connection table), and have to draw two (or more) with each of these structures being a resonance structure, and the real molecule being a sort of average of all resonance structures. In the case of nonaromatic resonance structure, the current protocol is to draw each resonance structure as its own connection table. Connection tables have a special way of representing aromatic compounds that have rings of delocalized electrons.

Aromatic Structures

Aromatic structures are common in organic chemistry and involve conjugated ring systems where electrons in p orbitals combine into pi-orbital rings systems forming delocalized orbitals over multiple nuclei. Benzene is the simplest aromatic compound, and because these are so ubiquitous, they are typically given a bond order of 4.



MOL I bond table

1	2	1	0
2	3	2	0
3	4	1	0
4	5	2	0
5	6	1	0
6	1	2	0
5	7	1	0
7	8	1	0
7	9	2	0

MOL IV bond table

1	2	2	0
2	3	1	0
3	4	2	0
4	5	1	0
5	6	2	0
6	1	1	0
5	7	1	0
7	8	1	0
7	9	2	0

MOL V bond table

1	2	4	0
2	3	4	0
3	4	4	0
4	5	4	0
5	6	4	0
6	1	4	0
5	7	1	0
7	8	1	0
7	9	2	0

Figure 2.3.12: The two Kekule structures represent the resonance structures of the benzene ring, which are often "combined" to form the ring structure on the right, which is given a bond order of 4.

Tautomers

Hydrogens are often labile and can easily jump from one atom to another and this can occur very rapidly. So a molecule may be jumping back and forth between two Lewis dot structures/connection tables.

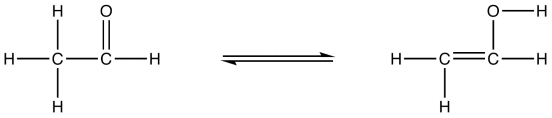


Figure 2.3.3: keto-enol tautomerism showing the two structures associated with acetaldehyde, where the hydrogen is jumping between the two carbons as the electron pair of the double bond switches between C=O and C=C.

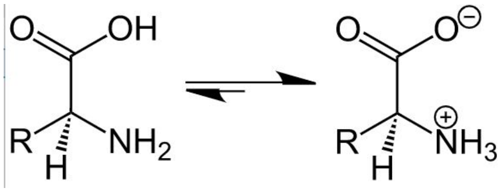


Figure 2.3.4: Amino acid undergoing tautomerism as it transforms between neutral and bi-charged (zwitter ionic) forms.

Zwitterions are typically dealt with as two files, and then connected by the database.

Multicovalent Units

A covalent unit is a single chemical entity held together by covalent bonds. Many chemical substances consits of multiple covalent units, like salts and mixtures. In the case of salts, each covalent unit has a charge being positive (cation) or negative (anion) and the sum of the charges must be neutral or the salt will not form.

A salt can be represented by a Simple Connection Table through the bond table where two groups of atoms in the atom table are simply not connected. We will take a closer look at this in section 2.5 when we look at actual chemical structural data files. But in essence, a file for a chemical multicovalent unit substance contains several disconnected bond groups within the bond table, and shows every atom of the salt. In the case of crystal structures these can include the 3D coordinates.

Mixtures are more complicated because neither the coordinates or the ratio of the substances are typically defined, and so they are typically not represented as structural data.

Contributors

Robert E. Belford (University of Arkansas Little Rock; Department of Chemistry). The breadth, depth and veracity of this work is the responsibility of Robert E. Belford, rebelford@ualr.edu. You should contact him if you have any concerns. This material has both original contributions, and content built upon prior contributions of the LibreTexts Community and other resources, including but not limited to:

- Evan Hepler-Smith
- Leah R. McEwen
- Material Adopted from 2017 Cheminformatics OLCC

2.4: Line Notation

Learning Objectives:

- Explain what SMILES, SMARTS and SMIRKS are.
- Explain what InChI and InChIKey are.
- Review SMILES specification rules.
- Compare and contrast SMILES and InChI.
- Demonstrate how to interpret SMILES, SMARTS, InChI strings into their corresponding chemical structures.

Introduction

Line notations represent structures as a linear string of characters. They are widely used in Cheminformatics because computers can easily process linear strings of data. Examples of line notations include the Wiswesser Line-Formula Notation (WLN)[1], Sybyl Line Notation (SLN)[2,3] and Representation of structure diagram arranged linearly (ROSDAL)[4,5]. Currently, the most widely used linear notations are the Simplified Molecular-Input Line-Entry System (SMILES)[6-9] and the IUPAC Chemical Identifier (InChI)[10-13], which are described below. In this class we will focus on SMILES and InChI line notation.

SMILES

The **Simplified Molecular-Input Line-Entry System** (SMILES)[6-9] is a line notation for describing chemical structures using short ASCII strings. SMILES is like a connection table in that it identifies the nodes and edges of a molecular graph. SMILES was developed in the late 1980s and implemented by Daylight Chemical Information Systems (Santa Fe, NM), but it is still widely used today. A detailed information on SMILES can be found in Chapter 3[14] of the Daylight Theory Manual as well as the SMILES tutorial[15].

SMILES Specification Rules

In SMILES, hydrogen are typically implicitly implied and atoms are represented by their atomic symbol enclosed in brackets unless they are elements of the "organic subset" (B, C, N, O, P, S, F, Cl, Br, and I), which do not require brackets unless they are charged. So gold would be [Au] but chlorine would be Cl. If hydrogens are explicitly implied brackets are used. A formal charge is represented by one of the symbols + or -. Single, double, triple, and aromatic bonds are represented by the symbols, -, =, #, and :, respectively. Single and aromatic bonds may be, and usually are, omitted. Here are some examples of SMILES strings.

Table 2.4.1: Common notations used in SMILES strings, note, *single and aromatic bonds are often omitted

Function	Symbol		Function	Symbol
single bond*	-		Positive charge	[C+]
double bond	=		Negative charge	[C-]
triple bond	#		aromatic carbon	c (lower case c)
aromatic bond*	:			

Table 2.4.2 shows some common SMILES strings. Note the following conventions

- **Branches** are specified by enclosures in parentheses and can be nested or stacked, as shown in these examples.
- **Rings** are represented by breaking one single or aromatic bond in each ring, and designating this ring-closure point with a digit immediately following the atoms connected through the broken bond. Atoms in aromatic rings are specified by lower cases

letters.

- **Aromatic Rings** use lower case c
- Although the carbon-carbon bonds in these two SMILES are omitted, it is possible to deduce that the omitted bonds are single bonds (for cyclohexane) and aromatic bonds (for benzene). One can also represent an aromatic compound as a non-aromatic, KeKulé structure. For example, the following is a valid SMILES string for benzene.
- C1=CC=CC=C1 Benzene (C6H6)

Table 2.4.2: Smiles Strings for some common molecules, note there are several ways to represent aromaticity

SMILES	Name (formula)		SMILES	Name (formula)		SMILES	Name(formula)
C	Methane (CH_4)		COC	Dimethyl ether (CH_3OCH_3)		CC(C)CO	Isobutyl alcohol (CH_3-$CH(CH_3)$-CH_2-OH)
CC	Ethane (CH_3CH_3)		CCO	Ethanol (CH_3CH_2OH)		CC(CCC(=O)N)CN	5-amino-4-methylpentanamide
C=C	Ethene (CH_2CH_2)		CC=O	Acetaldehyde (CH_3-CH=O)		C1CCCCC1	Cyclohexane (C_6H_{12})
C#C	Ethyne (CHCH)		CC(=O)[O-]	Acetate		c1ccccc1	Benzene (C_6H_6) (aromatic representation)
C#N	Hydrogen Cyanide (HCN)		[C-]#N	Cyanide anion		C1=CC=CC=C1	Benzene (C_6H_6) (KeKulé representation)

Note that aromaticity is not a measurable physical quantity, but a concept without a unanimous mathematical definition. As a result, different aromaticity detection algorithms often disagree with each other on whether a given molecule is aromatic or not, *making it difficult to interchange information between databases that use different aromaticity detection algorithms for SMILES generation.*

Also note that a ring structure can have multiple potential ring-closure points. For example, a six-membered ring has six bonds, each of which can be a ring-closure point. As a result, a ring compound may be represented by many different but equally valid SMILES strings. Actually, it is very common that there are a lot of SMILES strings that represent the same structure, whether it has a ring or not, because one can start with any atom in a molecule to derive a SMILES string. Therefore, it is necessary to select a "unique SMILES" for a molecule among many possibilities. Because this is done through a process called "canonicalization", this unique SMILES string is also called the "canonical SMILES".

Isomeric SMILES

Isomeric SMILES allow for the specification of the isotopism and stereochemistry of a molecule. Information on isotopism is indicated by the integral atomic mass preceding the atomic symbol. The atomic mass must be specified inside square brackets. For example, C-13 methane can be represented by "[13CH4]". Configuration around double bonds is specified by "directional bonds" (characters / and \). For example, E- and Z-1,2-difluoroethene can be represented by the following isomeric SMILES:

- F/C=C/F or F\C=C\F (E)-1,2-difluoroethene (trans isomer)
- F/C=C\F or F\C=C/F (Z)-1,2-difluoroethene (cis isomer)

Configuration around tetrahedral centers are indicated by the symbols "@" or "@@"

- C[C@@H](C(=O)O)N L-Alanine
- C[C@H](C(=O)O)N D-Alanine

More detailed information on chirality specification can be found in Chapter 3[14] of the Daylight Theory Manual.

Limitations of SMILES

SMILES is proprietary and it is not an open project. This has led different chemical software developers to use different SMILES generation algorithms, resulting in different SMILES versions for the same compound. Therefore, SMILES strings obtained from different databases or research groups are not interchangeable unless they used the same software to generate the SMILES strings. With an aim to address this interchangeability issue of SMILES, an open-source project has launched to develop an open, standard version of the SMILES language called OpenSMILES. However, the most noticeable community effort in this area is development of InChI, which is described in next section.

SMARTS

SMiles **AR**bitrary **T**arget **S**pecification (SMARTS) notation allows one to search in certain databases (like PubChem) for generic structures. It is a language used for describing molecular patterns. SMARTS is useful for substructure searching, which finds a particular pattern (subgraph) in a molecule. SMARTS are straightforward extensions of SMILES. All SMILES symbols and properties are legal in SMARTS. SMARTS includes logical operators and additional molecular descriptors. Detailed information on SMARTS is given in the SMARTS specification document in the Daylight theory manual and SMARTS tutorial.

SMIRKS

Another extension of SMILES is SMIRKS, which is a line notation for generic reactions. A generic reaction represents a group of reactions that undergo the same set of atom and bond changes. Note that SMILES and SMARTS can be used to represent reactions, using the ">" symbol between the reactants, products, and agents, as described in the SMILES and SMARTS specification documents. (Therefore, these SMILES and SMARTS that describe reactions are often called reaction SMILES and reaction SMARTS, respectively.) On the other hand, SMIRKS is used to represent *types* of reactions (e.g., S_N2 reaction). More detailed information on SMIRKS is given in the SMIRKS specification document and SMIRKS tutorial.

InChI

Since 1919 the International Union of Pure and Applied Chemistry (IUPAC) has been the international authority on chemical nomenclature and terminology. IUPAC currently consists of members from 57 national adhering organizations (NAOs) whose recommendations are made public through the IUPAC journal Pure and Applied Chemistry and the IUPAC Color books. As we entered the new millennial the leadership of IUPAC recognized the need to extend chemical nomenclature into the digital realm of computer databases and software agents, and in March of 2000 during a meeting at the U.S. Naval Academy started a project with the U.S. National Institute of Standards and Technology (NIST), to build a machine readable nomenclature standard, the InChI. It was originally called INChI for IUPAC/NIST Chemical Identifier, but was changed to InChI (International Chemical Identifier) as although it was built with efforts from NIST, it was not appropriate for NIST as a government agency to place its name as a recommendation for the identifier. In 2010 the InChI Trust was formed and development of the standard is continuing the purview of the InChI Trust and the IUPAC InChI subcommittee.

InChI is an open, freely available non-proprietary computer generated chemical identifier that is based on a hierarchical layered line notation (see below). The first three layers essentially deal with the information within the simplified connection table, and the additional layers are added as needed, and deal with complexities like isomers, isotopic distributions and the other types of issues brought up in section 2.3 of this chapter, and these layers are extensible. A standard InChI has a predefined number of layers, and these can be extended to non-standard InChI's that can have new layers relating to define additional information, that is what is meant by extensible layes. Unlike SMILES, InChI is a canonical line notation and so is a unique identifier that is built upon a set of nomenclature rules. That is, although there are canonical SMILES built through a canonicalization algorithm, there can be more than one canonicalization algorithm for SMILES, and so you can have more than one SMILES string for the same structure.

Students may be familiar with the American Chemical Society's Chemical Abstract Service (CAS) registry number, which is supposed to be a unique identifier based on the registry system, but issues can arise (see below, other identifiers, and problem). Also, a CAS registry number is associated with a compound that has been published in the primary literature or patents, and the CAS system bases its identifiers on the registry system, not the structure of the molecule. That is, InChI is not a registry system, it is a type of nomenclature that describes the structure of a molecule, and you can make an InChI for a molecule that does not exist, as long as you specify its structure.

The most recent version of InChI (and its documentation) can be obtained at the InChI Trust Download site.

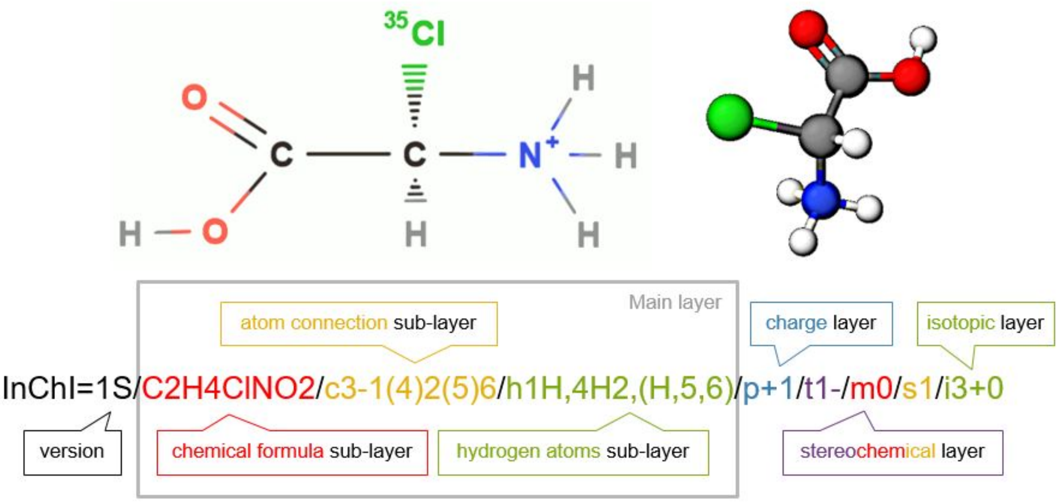
InChI: A Layered Notation

The power of a layered notation is that it gets to the essence of what is a molecule? For example, we think water is H_2O, but if you look at a real sample of water you will note that some of the hydrogens are protium (one proton) and others are deuterium (a proton and a neutron), and in fact the ratio of deuterium to protium in ground water samples can vary from one region of the US to another, and thus the molar mass of samples of water can vary. In fact IUPAC has now adopted an "interval atomic weight notation system" for some elements whose atomic mass varies across samples, and this can affect physical properties of a sample, like the vapor pressure of water. So water is water, but not all water is the same, and the question becomes, do you care? If you are uploading data to a database and know the isotopic distribution you care, but if you do not know it, you do not care, but in both cases, your data deals with water. Through a layered notation system you can have an isotopic layer to describe your water if you care, but you don't need it if you don't care. This leads to one of the issues that comes up with the layered notation, in that you can have different InChI's for a compound, depending on the kind of information you want in the name, that is how many and what kind of layers you use. This leads to the standard InChI, which is an InChI that has defined layers and is thus canonical, and starts with a number followed by the letter "s" to indicate the version of the standard.

Standard InChI

Standard InChI version 1.05 was released in January 2017 and has 6 core layers (and several sublayers within the core layers) and starts with InChI=1S/... . Each layer in the InChI string is separated by a "/" and the "main layer" is essentially the connection table. The InChI software generates both standard and nonstandard InChI, with the standard InChI having "fixed options" that ensures interoperability between databases and software agents. The standard InChI (version 1.05) has the following layers:

1. Main Layer
    1. Chemical Formula Layer (based on Hill Notation)
    2. Connections- bonds between atoms and may have sublayers, with the last one dealing with mobile hydrogens.
2. Charge Layer
    1. Component Charge
    2. Protons
3. Stereochemical Layer
    1. Double Bond sp^2 (Z/E) Sterochemistry
    2. Tetrahedral Sterochemistry
4. Isotopic Layer
5. Fixed Hydrogen Layer (binds mobile hydrogens)
6. Polymer Layer (actually a new experimental layer) and does not affect the content of the earlier layers.



^{35}Cl
O
C
C
N$^+$
H
H
H
H
H—O
Main layer
atom connection sub-layer
charge layer
isotopic layer
InChI=1S/C2H4ClNO2/c3-1(4)2(5)6/h1H,4H2,(H,5,6)/p+1/t1-/m0/s1/i3+0
version
chemical formula sub-layer
hydrogen atoms sub-layer
stereochemical layer

Figure 2.4.1: The main layers for a standard InChI of [(R)-carboxy(chloro)methyl]azanium, the protonated form of 2-(^{35}Cl)chloro-R-glycine . Note each layer or sublayer is separated by a forward slash [/].

NonStandard InChI

Note a nonstandard InChI does not start with InChI=1S/... but with a InChI=1/.... and has additional layers that approach different facets of a molecule's structure or features. In fact a company could create their own lawyer for a nonstandard information and encode into it proprietary information that they wished to keep private. The nonstandard InChI may not be canonical, but can handle facets of information information that a standard can not, in fact for a standard to be canonical different tautomers must have the same InChI, or you have two InChIs for the same molecule

So defining specific tautomers is one use of a nonstandard InChI as can be seen in the case of 4,5-Dihydro-1,3-Oxazol-3-lum. Figure 2 shows the two tautomeric forms of this molecule which must have the same standard InChI or it would not be canonical (you would have two InChIs for the same molecules). If you want to define just one of the tautomers, you need to use a nonstandard InChI and add a fixed hydrogen layer (in red). Although these are two ways of drawing the same molecule, one form may be favored over the other in certain environments and so there may be data indicative of the behavior of one of these form and not the other, and thus there may be a need to distinguish between the tautomers.

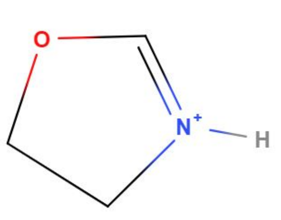
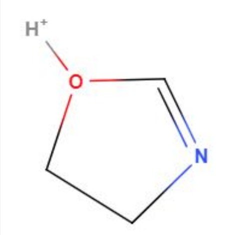
InChI=1S/C3H5NO/c1-2-5-3-4-1/h3H,1-2H2/p+1	
	
InChI=1/C3H5NO/c1-2-5-3-4-1/h3H,1-2H2/p+1/fC3H6NO/h4H/q+1	InChI=1/C3H5NO/c1-2-5-3-4-1/h3H,1-2H2/p+1/fC3H6NO/h5H/q+1

Figure 2.4.2: The above two structures are tautomeric drawings of the same molecule and thus have the same standard InChI. If you were interested in just one of the structures you could use a nonstandard InChI with a fixed hydrogen layer (in red). Borrowed from section 6.2 of InChI Trust FAQ.

Drawbacks of InChI

InChIs are not meant to be human readable but to contain molecular information that computers can read within the layers, so unlike SMILES you can't really read even a simple InChI (see Figure 3), never mind a complex one (figure 4).

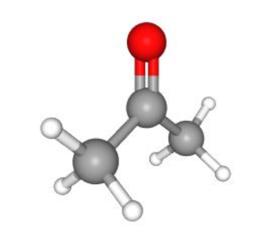
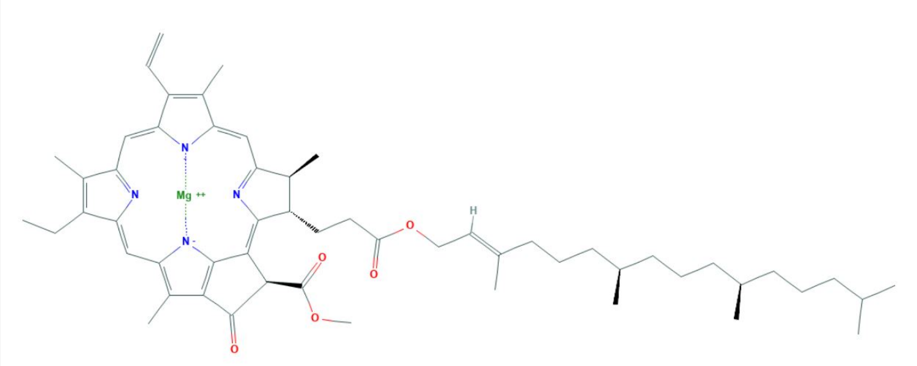
SMILES	InChI
CC(=O)C	InChI=1S/C3H6O/c1-3(2)4/h1-2H3
 Acetone	

Figure 2.4.3: Canonical SMILES and InChI for Acetone (source: PubChem)

Another drawback of InChI is just like an IUPAC systematic name, they are of variable length and become real long (figure 4). The problem with the variable length is it makes InChI impractical as a database registry number, and the length is often too long for internet search engines to handle.

IUPAC Systematic Name	InChI
magnesium;methyl (3*R*,21*S*,22*S*)-16-ethenyl-11-ethyl-12,17,21,26-tetramethyl-4-oxo-22-[3-oxo-3-[(*E*,7*R*,11*R*)-3,7,11,15-tetramethylhexadec-2-enoxy]propyl]-23,25-diaza-7,24-diazanidahexacyclo$[18.2.1.1^{5,8}.1^{10,13}.1^{15,18}.0^{2,6}]$hexacosa-1,5,8(26),9,11,13(25),14,16,18,20(23)-decaene-3-carboxylate	InChI=1S/C55H73N4O5.Mg/c1-13-39-35(8)42-28-44-37(10)41(24-25-48(60)64-27-26-34(7)23-17-22-33(6)21-16-20-32(5)19-15-18-31(3)4)52(58-44)50-51(55(62)63-12)54(61)49-38(11)45(59-53(49)50)30-47-40(14-2)36(9)43(57-47)29-46(39)56-42;/h13,26,28-33,37,41,51H,1,14-25,27H2,2-12H3,(H-,56,57,58,59,61);/q-1;+2/p-1/b34-26+;/t32-,33-,37+,41+,51-;/m1./s1



Chlorophyll A

Figure $2.4.4$: On the top left is the IUPAC systematic name for chlorophyll A and on the right is it's InChI (source: PubChem).

There is an additional issue with the InChI in that some of the characters interfere with web search queries and thus the InChI itself is not appropriate for web searches. To solve these problems a hashed InChI Key has been developed which is of constant length and enables web searches. The hashed key is also of constant length, making it better suited for databases.

InChI Keys

The InChI suite will generate a hashed version of the InChI, the InChI Key. The hash function generates a standard key of 27 characters that stores information in four parts (see figure 5). The InChIKey may be a standard or nonstandard key as indicated by the version, but all keys are of the same length and format.

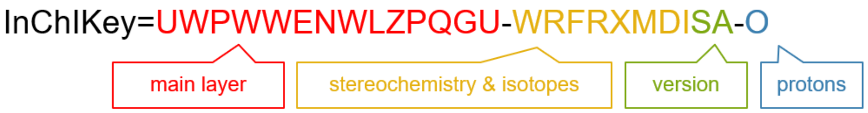

InChIKey=UWPWWENWLZPQGU-WRFRXMDISA-O
main layer
stereochemistry & isotopes
version
protons

Figure $2.4.5$: InChI key for 2-(^{35}Cl)chloro-R-glycine (molecule in figure 1).

The hash function is a one-way conversion (figure 6), that is, if you have an InChI you can generate the key, but if you have the key you can not generate the InChI. The key can function as an identifier is you made it registry number where you would need a look up table to know the molecule it is associated with.

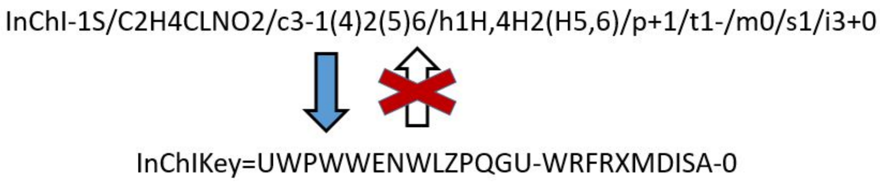

InChI-1S/C2H4CLNO2/c3-1(4)2(5)6/h1H,4H2(H5,6)/p+1/t1-/m0/s1/i3+0
InChIKey=UWPWWENWLZPQGU-WRFRXMDISA-O

Figure $2.4.6$: The one way InChIKey generation function.

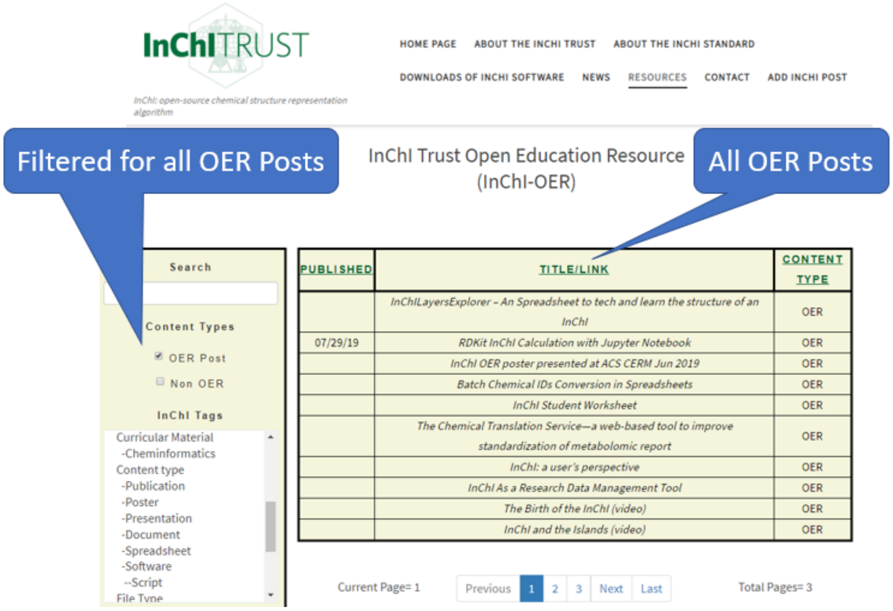
If two different chemical compound databases have the same chemical (InChI)they will generate the same standard InChIKeys and thus it is customary for databases and other information sources like Wikipedia chemboxes to generate standard InChIKeys, and they effectively function as a standard "registry number", that is, if two chemicals in different databases have the same standard InChIKey, they are the same chemical. On the other hand if you had a non-standard InChI the non-standard layers would induce variability of the key and so you could not compare across databases.

InChIKeys and Web Searching

The molecule (R)-2-(^{35}Cl)chloroglycine probably does not exist and was created to demonstrate the layers of an InChI and the correlating key. If you do a web search of the entire key (UWPWWENWLZPQGU-WRFRXMDISA-0) you do not get any hits, but if you search just the main layer you get several hits. What you are doing is essentially looking for any molecule with the same simplified connection table, that is, all stereoisomers, or isotopic labels. One of the hits is for [(*S*)-carboxy(chloro)methyl]azanium which is the other isomer. If you go to properties they are all computed and none were deposited to PubChem by vendors or contributors, and so this molecule has probably never been synthesized. Under 5.2 Related Compounds/Exact Same Parent you also get the non-protonated form (2-chloro-L-glycine) , of which there is published information. It is also of interest that the search of first part of the InChIKey also turned up an article in Russian on the L isomer, (you may need to download the pdf to see the actual bonds).

InChI OER

The InChI Trust runs an Open Education Resource (OER) where you can find material on InChI https://www.inchi-trust.org/oer/. The InChI OER is a repository where anyone can upload and tag material on InChI, or link to and tag existing material on the use of InChI. Once material is posted within the OER it can be searched through a filter system.



InChI Trust Open Education Resource (InChI-OER)

PUBLISHED	TITLE/LINK	CONTENT TYPE
	InChILayersExplorer – An Spreadsheet to tech and learn the structure of an InChI	OER
07/29/19	RDKit InChI Calculation with Jupyter Notebook	OER
	InChI OER poster presented at ACS CERM Jun 2019	OER
	Batch Chemical IDs Conversion in Spreadsheets	OER
	InChI Student Worksheet	OER
	The Chemical Translation Service—a web-based tool to improve standardization of metabolomic report	OER
	InChI: a user's perspective	OER
	InChI As a Research Data Management Tool	OER
	The Birth of the InChI (video)	OER
	InChI and the Islands (video)	OER

Figure 2.4.7: InChI OER tag filter and associated content. The default setting is to show all OER site material, clicking non-OER will extend the filter to include off site material like publications which have records that have been submitted to the OER.

InChI Layers Explorer

In this activity we will use the InChI OER to obtain an Excel spreadsheet that breaks an InChI into layers, and start to analyze how cheminformatics functionality can be integrated into common tools like spreadsheets. Go to the InChI OER and in the filter click "Spreadsheet" (middle of figure 2.4.8). This filters the content to items that are tagged "spreadsheet" and also removes any tag that is not associated with one of those content items. Now move down to tag category "File Type" and while holding the <ctrl> key, click Excel (right figure 2.4.8). You now get a list of excel spreadsheets (figure 2.4.9).

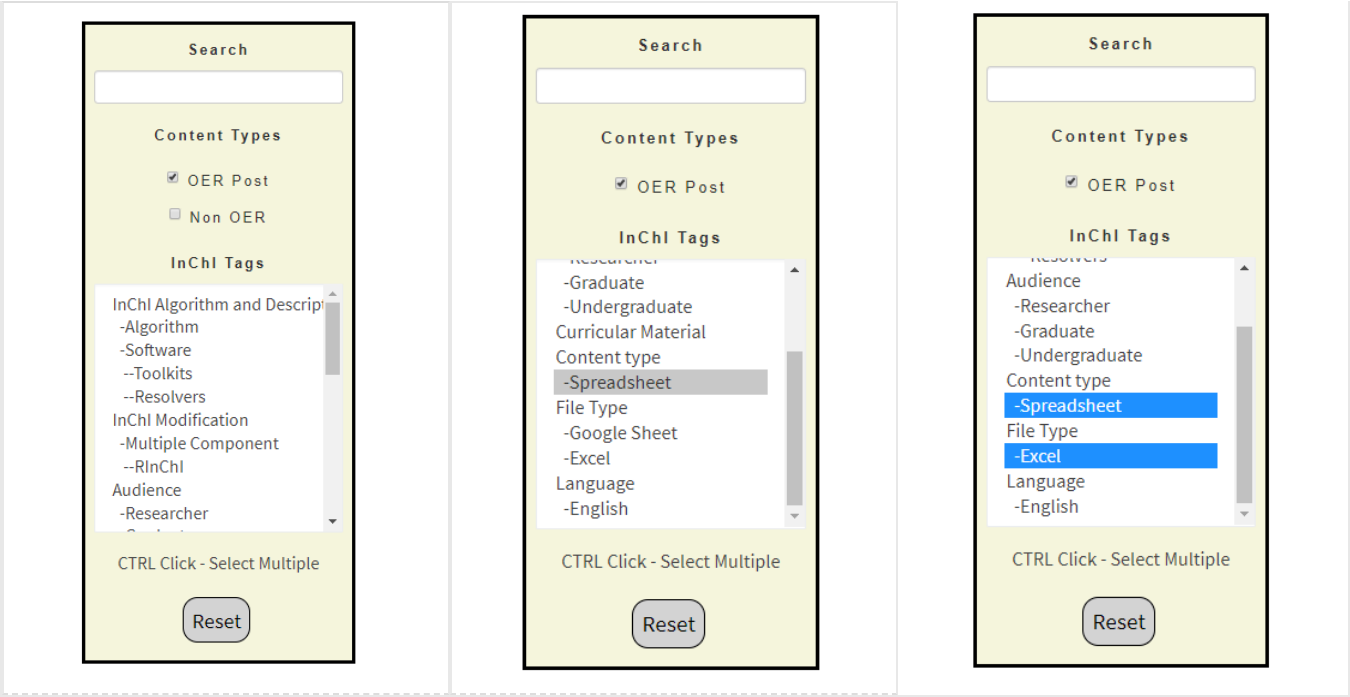
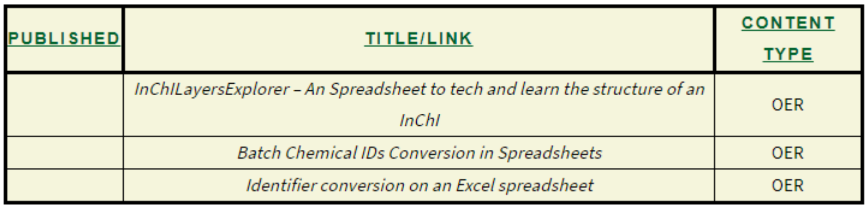


Figure 2.4.8: InChI OER Tag Filter.

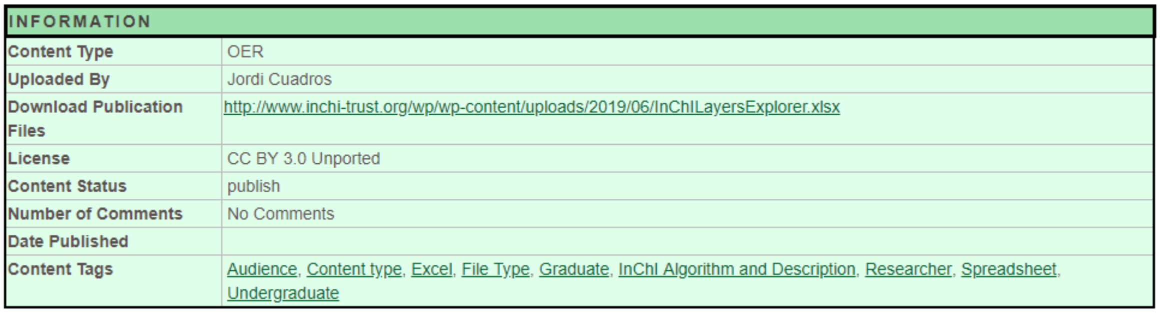
On the left is the default setting and all content loaded to the site is displayed in the window (right side of figure 2.4.7. In the middle the filter for spreadsheets is activated, and you can see there are two types that have been uploaded, Google Sheets and Excel sheets. On the right both Spreadsheet and Excel have been activated, and so only spreadsheets in Excel are displayed and the content view is reduced to those items that are tagged both "Spreadsheet" and "Excel" (Figure 2.4.9)



PUBLISHED	TITLE/LINK	CONTENT TYPE
	InChILayersExplorer – An Spreadsheet to tech and learn the structure of an InChI	OER
	Batch Chemical IDs Conversion in Spreadsheets	OER
	Identifier conversion on an Excel spreadsheet	OER

Figure 2.4.9: At the time this page was created there were three items uploaded to the InChI OER that were tagged as Excel Spreadsheets.

Click on the InChILayersExplorer and you go to it's content page. This page will have a description of the content and a green information box (Figure 2.4.10), and in the information box is a "Download Publication Files", that allows you to obtain the spreadsheet.



INFORMATION	
Content Type	OER
Uploaded By	Jordi Cuadros
Download Publication Files	http://www.inchi-trust.org/wp/wp-content/uploads/2019/06/InChILayersExplorer.xlsx
License	CC BY 3.0 Unported
Content Status	publish
Number of Comments	No Comments
Date Published	
Content Tags	Audience, Content type, Excel, File Type, Graduate, InChI Algorithm and Description, Researcher, Spreadsheet, Undergraduate

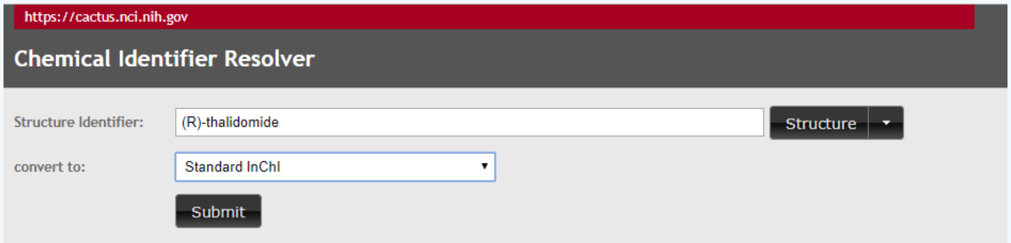
Figure 2.4.10: *Green Information box for the InChILayersExplorer*

Now click on the link in the "Download Publications File" field and you will have a copy of the InChI Layers Explorer, which you should open and enable editing.

✓ Activity 2.4.1

Using the InChILayersExplorer show the difference between the InChI for (R)-thalidomide and (S)-thalidomide. Note, the goal of this activity is not to answer the question, but to gain an understanding on how the InChILayersExplorer works, which is in effect a "smart spreadsheet" that communicates with database web APIs via webservices functions. One of the skills we hope you can gain from this class is enough familiarity with how code works so if you see new code, you can hack in and figure how it works. Be sure to enable the spreadsheet after you download it. This spreadsheet communicates with the NCI Chemical Resolver (section 2.7.)

1. Type (R)-thalidomide in the yellow region (type over CoA), OK, it fails, now try the (S) isomer, and it still fails, so now try thalidomide without specifying an isomer. OK, so you have the InChI for thalidomide, but there is nothing in the stereochemical layer, as you have not specified the stereochemistry. These spreadsheet uses the Chemical Identifier Resolver of the NIH which will be covered in section 2.6.2.1.1), which can be accessed directly at https://cactus.nci.nih.gov/chemical/structure and is shwon in figure 2.4.11. Now lets start by searching for (R)-thalidomide directly in the resolver (figure 2.4.11).



Structure Identifier: (R)-thalidomide

convert to: Standard InChI

Figure 2.4.11: NCI/CADD Chemical Resolver set up to find standard InChI for (R)-thalidomid

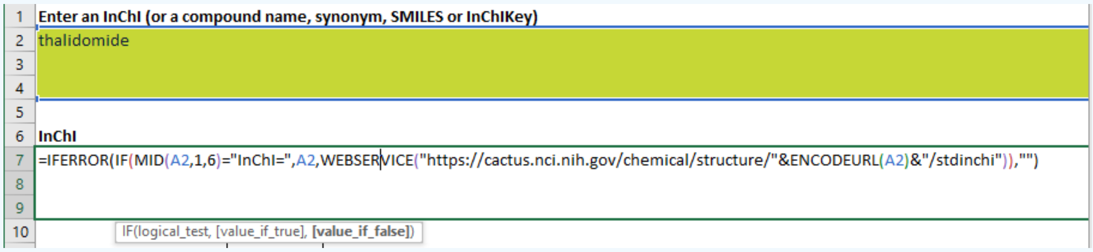
As you may have guessed, neither (R) or (S) works, but "thalidomide" does (incidentally, you have to hit submit, not Structure), and so this resolver will not provide information on the isomers of thalidomide. So now do a web search of (R)-thalidomide, and paste in its key (UEJJHQNACJXSKW-SECBINFHSA-N), and note the stereochemical layer [/t9-/m1/s1] is the only part that is different. Now repeating for (S)-thalidomide.

You should get the following results

Table 2.4.3

Compound	InChI Key	Stereochemical layer
thalidomide	UEJJHQNACJXSKW-UHFFFAOYSA-N	none
(R)-thalidomide	UEJJHQNACJXSKW-SECBINFHSA-N	/t9-/m1/s1
(S)-thalidomide	UEJJHQNACJXSKW-VIFPVBQESA-N	/t9-/m0/s1

Note, if you click on the merged cells that generates the InChI (Rows 7-8) you see the following code.



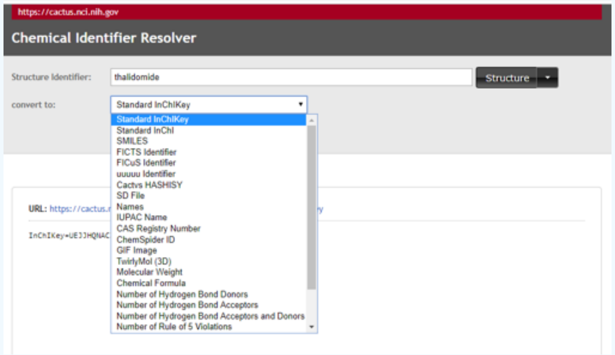
```
1   Enter an InChI (or a compound name, synonym, SMILES or InChIKey)
2   thalidomide
3
4
5
6   InChI
7   =IFERROR(IF(MID(A2,1,6)="InChI=",A2,WEBSERVICE("https://cactus.nci.nih.gov/chemical/structure/"&ENCODEURL(A2)&"/stdinchi")),"")
8
9
10          IF(logical_test, [value_if_true], [value_if_false])
```

Figure 2.4.11: Code in spreadsheet that uses WEBSERVICE function to get InChI from NCI/CADD chemical resolver

Now open up a browser tab and paste in the following URL:

https://cactus.nci.nih.gov/chemical/structure/thalidomide/stdinchi

Now go back to the NCI Chemical Resolver and click the dropdown box of the "convert to" field (figure$2.4.12$and try another option, say "TwirlyMol(3D).



https://cactus.nci.nih.gov

Chemical Identifier Resolver

Structure Identifier: thalidomide

convert to: Standard InChIKey

URL: https://cactus.n

InChIKey=UEJJHQNAC

$2.4.12$: Dropdown menu of NCI Chemical resolver showing some of the options.

Can you figure out the URL that uses the NCI Chemical Resolver to give the 3D molecule in a webpage? Once you have done this, can you identify a problem that has resulted from these molecular representations. Hint, think of adding two more columns to table $2.4.3$, one for 2D and one for 3D images. What is the issue when you draw the 3D image that does not arise when you draw the 2D?

References and Further Reading

1. (1) Wiswesser, W. J. *J. Chem. Inf. Comput. Sci.* **1982**, *22*, 88.
2. (2) Ash, S.; Cline, M. A.; Homer, R. W.; Hurst, T.; Smith, G. B. *J. Chem. Inf. Comput. Sci.***1997**, *37*, 71.
3. (3) Homer, R. W.; Swanson, J.; Jilek, R. J.; Hurst, T.; Clark, R. D. *J. Chem Inf. Model.***2008**, *48*, 2294.
4. (4) Barnard, J. M.; Jochum, C. J.; Welford, S. M. *Acs Symposium Series* **1989**, *400*, 76.
5. (5) Rohbeck, H. G. In *Software Development in Chemistry 5*; Gmehling, J., Ed.; Springer Berlin Heidelberg: 1991, p 49.
6. (6) Weininger, D. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31.
7. (7) Weininger, D.; Weininger, A.; Weininger, J. L. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 97.
8. (8) Weininger, D. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 237.
9. (9) SMILES: Simplified Molecular Input Line Entry System (http://www.daylight.com/smiles/) (Accessed on 6/30/2015).
10. (10) Heller, S.; McNaught, A.; Stein, S.; Tchekhovskoi, D.; Pletnev, I. *J. Cheminform.* **2013**, *5*, 7.
11. (11) Heller, S.; McNaught, A.; Pletnev, I.; Stein, S.; Tchekhovskoi, D. *J. Cheminform.* **2015**, *7*, 23.
12. (12) The IUPAC International Chemical Identifier (InChI) (http://www.iupac.org/home/publications/e-resources/inchi.html) (Accessed on 6/29/2015).
13. (13) InChI Trust (http://www.inchi-trust.org/) (Accessed on 6/29/2015).
14. (14) Daylight Theory Manual, Chapter 3: SMILES - A Simplified Chemical Language (http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html) (Accessed on 6/23/2015).
15. (15) Daylight SMILES Tutorial (http://www.daylight.com/dayhtml_tutorials/languages/smiles/index.html) (Accessed on 6/23/2015).

Contributors

Robert E. Belford (University of Arkansas Little Rock; Department of Chemistry). The breadth, depth and veracity of this work is the responsibility of Robert E. Belford, rebelford@ualr.edu. You should contact him if you have any concerns. This material has both original contributions, and content built upon prior contributions of the LibreTexts Community and other resources, including but not limited to:

- Sunghwan Kim
- Material Adapted from 2017 Cheminformatics OLCC

2.5: Structural Data Files

Learning Objectives:

- Gain understanding of chemical structural data files
- Survey data formats
- Survey molecular visualization and manipulation software and web services

Introduction

Structural data files are the files software agents typically use when processing chemical structural information, but can also contain additional information like molecular spectra. In principle you could say that there are two major components any structural data file, the simplified connection table and additional information. In effect the InChI line notation sort of models them, in that the main layer is the simplified connection table and the other layers are the additional information, except that in a structural data files hydrogen can be implicit or explicit (in the InChI they are explicit). So when you look at the different types of structural data files you will see they all have an atom table and a bond table. Information about individual atoms like isotopic definitions are associated with the atom table. That atom table may also indicate the 3d coordinates associated with a specific environment, and if that information is missing software agents will use an energy minimization calculation to determine 3D structure of an isolated atom.

In this section we will give a brief of the different types of structural data files and a survey of software programs and web services that can be used to display and manipulate structural data files, with a focus on open source options. There will be some overlap with these software programs and next section on chemical resolvers, which allow you to convert between file types.

Common Types of Structural Data Files

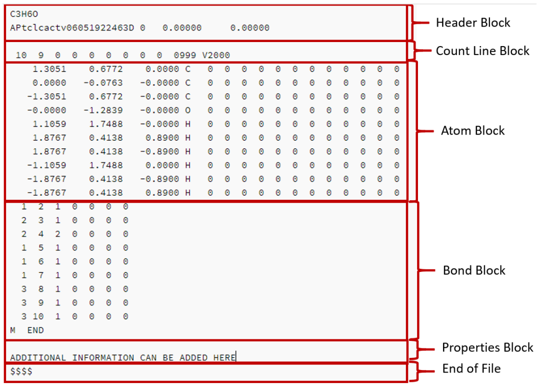
There are a variety of file formats and the most common are based on the MDL Molfile, of which V2000 is the most common, although V3000 is also commonly used. The SDF (Structure Data File) is based on the Molfile and figure represent an SDF file for acetone obtained through the NCI/CADD chemical identifier resolver.

Molfile

The following is a molfile for acetone obtained from the NCI chemical resolver. All molfiles have a header and a connection table (CTAB) that has two blocks, the Atom Block and the Bond Block.

The Header block has two lines, the first gives the name/formula of the molecule (if known) and is of variable format, the second gives the program that made file, the date and time it was made, and if 2D or 3D coordintates are given (figure 2.5.1 was created June 5, 2019 at 22:46 and has 3D coordinates).

The Count line block tells us acetone has 10 atoms and 9 bonds, it also provides the version number of the molfile. N

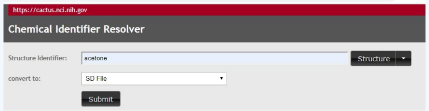


```
C3H6O
APtclcactv06051922463D 0   0.00000     0.00000

 10  9  0  0  0  0  0  0  0  0999 V2000
    1.3051    0.6772    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000   -0.0763   -0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.3051    0.6772   -0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.0000   -1.2839   -0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.1059    1.7488   -0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.8767    0.4138    0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.8767    0.4138   -0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.1059    1.7488    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.8767    0.4138   -0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.8767    0.4138    0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  2  0  0  0  0
  1  5  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  3  8  1  0  0  0  0
  3  9  1  0  0  0  0
  3 10  1  0  0  0  0
M  END

ADDITIONAL INFORMATION CAN BE ADDED HERE
$$$$
```

Figure 2.5.1: Molefile for acetone

Activity 2.5.1

Go to the NCI Chemical Identifier Resolver (https://cactus.nci.nih.gov/chemical/structure); in Structure Identifier type "acetone", choose convert to SD File and submit.



https://cactus.nci.nih.gov

Chemical Identifier Resolver

Structure Identifier: acetone

convert to: SD File

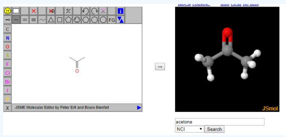
Compare your file to figure 2.5.1, and hopefully the only difference you will see is the date the file was generated. We will discuss chemical resolvers in the next section.

Professor Bob Hanson at Saint Olaf College created a program for an earlier offering of this class called Hack-a-Mol that we will use to explore data files.

https://chemapps.stolaf.edu/jmol/jsmol/hackamol.htm

Activity 2.5.2

Open Hack-a-mol in a new window and search NCI for "acetone".



Now compare the molfile to the molfile from activity 2.5.1. What is the difference, and can you explain what is going on?

Activity 2.5.3

Open a new browser window, load Hack-a-Mol, search for Acetone, but use Pubchem instead of NCI.

Note the atom numbers in the atom block are implicit starting with 1 and going down to 10. We can also see that the oxygen is atom 4. From the bond block we see that atom 4 is attached to atom 2 (carbon) and it is a double bond. We also see that atom two is involved with two additional bonds, one to atom 1 and the other to atom 3, and both of those atoms are carbon. The connection table defines the molecules connectivity, and when coupled with 3D coordinates, gives its geometric shape. In this particular table we have included the hydrogens explicitly, but they could have been ommited. Also note the file ends with the four dollar signs,

$$\tag{2.5.1}$$

.

Figure 2.5.2 is the same data

```
C3H6O
APtclcactv06051922463D 0   0.00000     0.00000

 10  9  0  0  0  0  0  0  0  0999 V2000
    1.3051    0.6772    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000   -0.0763   -0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.3051    0.6772   -0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.0000   -1.2839   -0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.1059    1.7488   -0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.8767    0.4138    0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.8767    0.4138   -0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.1059    1.7488    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.8767    0.4138   -0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.8767    0.4138    0.8900 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  2  0  0  0  0
  1  5  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  3  8  1  0  0  0  0
  3  9  1  0  0  0  0
  3 10  1  0  0  0  0
M  END

ADDITIONAL INFORMATION CAN BE ADDED HERE
$$$$
```

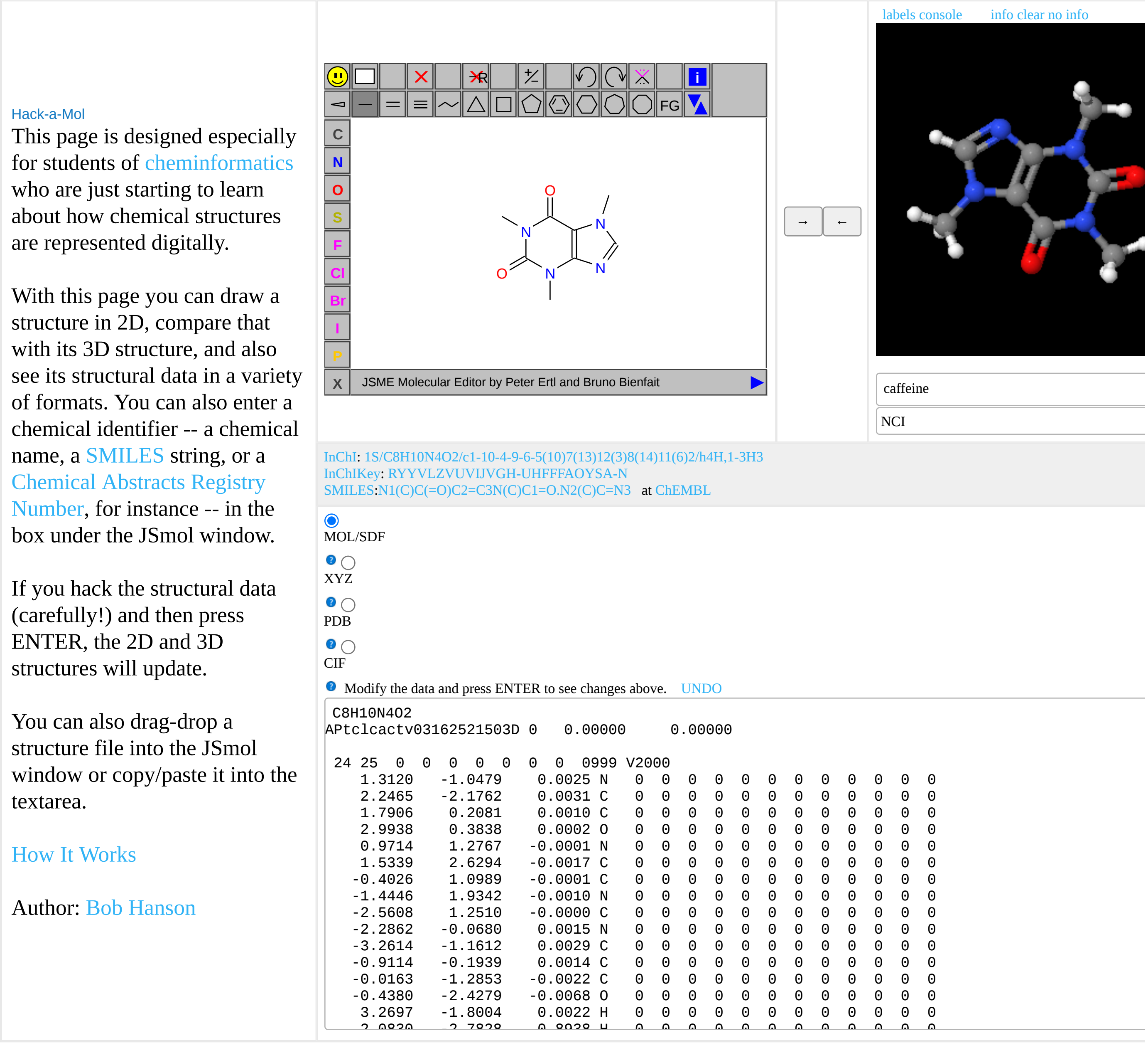
```
180
  -OEChem-06051922532D

 10  9  0     0  0  0  0  0  0999 V2000
    3.7320    0.7500    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    0.2500    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000    0.7500    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660   -0.7500    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.3100    1.2869    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.4631    1.0600    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.6900    0.2131    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.2460   -0.7500    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660   -1.3700    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.4860   -0.7500    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  2  0  0  0  0
  2  3  1  0  0  0  0
  2  4  1  0  0  0  0
  3  5  1  0  0  0  0
  3  6  1  0  0  0  0
  3  7  1  0  0  0  0
  4  8  1  0  0  0  0
  4  9  1  0  0  0  0
  4 10  1  0  0  0  0
M  END
> <PUBCHEM_COMPOUND_CID>
180

> <PUBCHEM_COMPOUND_CANONICALIZED>
1

> <PUBCHEM_CACTVS_COMPLEXITY>
26.3

> <PUBCHEM_CACTVS_HBOND_ACCEPTOR>
```

```
1

> <PUBCHEM_CACTVS_HBOND_DONOR>
0

> <PUBCHEM_CACTVS_ROTATABLE_BOND>
0

> <PUBCHEM_CACTVS_SUBSKEYS>
AAADcYBAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGgAAAAAACASAgAACAAAAAAAIAIAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

> <PUBCHEM_IUPAC_OPENEYE_NAME>
acetone

> <PUBCHEM_IUPAC_CAS_NAME>
2-propanone

> <PUBCHEM_IUPAC_NAME_MARKUP>
propan-2-one

> <PUBCHEM_IUPAC_NAME>
propan-2-one

> <PUBCHEM_IUPAC_SYSTEMATIC_NAME>
propan-2-one

> <PUBCHEM_IUPAC_TRADITIONAL_NAME>
acetone

> <PUBCHEM_IUPAC_INCHI>
InChI=1S/C3H6O/c1-3(2)4/h1-2H3

> <PUBCHEM_IUPAC_INCHIKEY>
CSCPPACGZOOCGX-UHFFFAOYSA-N

> <PUBCHEM_XLOGP3_AA>
-0.1

> <PUBCHEM_EXACT_MASS>
58.042

> <PUBCHEM_MOLECULAR_FORMULA>
C3H6O

> <PUBCHEM_MOLECULAR_WEIGHT>
58.08

> <PUBCHEM_OPENEYE_CAN_SMILES>
CC(=O)C

> <PUBCHEM_OPENEYE_ISO_SMILES>
CC(=O)C

> <PUBCHEM_CACTVS_TPSA>
17.1

> <PUBCHEM_MONOISOTOPIC_WEIGHT>
58.042

> <PUBCHEM_TOTAL_CHARGE>
```

Hack-a-Mol

Here's a website at St. Olaf College where you can play with the relationship between 2D structures, 3D renderings, identifiers, and connection tables, courtesy of the cheminformatician Bob Hanson. There's a link on the page to a document explaining "How it Works" (also linked here). As this course proceeds you will learn how we communicate with the NCI resolver and PubChem, and many of the fundamental features behind this application.

We have also embedded Hack-a-Mol below, and when doing your assignments you may want to open in a new window.



Hack-a-Mol

This page is designed especially for students of cheminformatics who are just starting to learn about how chemical structures are represented digitally.

With this page you can draw a structure in 2D, compare that with its 3D structure, and also see its structural data in a variety of formats. You can also enter a chemical identifier -- a chemical name, a SMILES string, or a Chemical Abstracts Registry Number, for instance -- in the box under the JSmol window.

If you hack the structural data (carefully!) and then press ENTER, the 2D and 3D structures will update.

You can also drag-drop a structure file into the JSmol window or copy/paste it into the textarea.

How It Works

Author: Bob Hanson

caffeine

NCI

InChI: 1S/C8H10N4O2/c1-10-4-9-6-5(10)7(13)12(3)8(14)11(6)2/h4H,1-3H3
InChIKey: RYYVLZVUVIJVGH-UHFFFAOYSA-N
SMILES:N1(C)C(=O)C2=C3N(C)C1=O.N2(C)C=N3 at ChEMBL

● MOL/SDF

○ XYZ

○ PDB

○ CIF

Modify the data and press ENTER to see changes above. UNDO

```
 C8H10N4O2
APtclcactv03162521503D 0   0.00000     0.00000

 24 25  0  0  0  0  0  0  0  0999 V2000
    1.3120   -1.0479    0.0025 N   0  0  0  0  0  0  0  0  0  0  0  0
    2.2465   -2.1762    0.0031 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.7906    0.2081    0.0010 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.9938    0.3838    0.0002 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.9714    1.2767   -0.0001 N   0  0  0  0  0  0  0  0  0  0  0  0
    1.5339    2.6294   -0.0017 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4026    1.0989   -0.0001 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.4446    1.9342   -0.0010 N   0  0  0  0  0  0  0  0  0  0  0  0
   -2.5608    1.2510   -0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -2.2862   -0.0680    0.0015 N   0  0  0  0  0  0  0  0  0  0  0  0
   -3.2614   -1.1612    0.0029 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.9114   -0.1939    0.0014 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.0163   -1.2853   -0.0022 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4380   -2.4279   -0.0068 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.2697   -1.8004    0.0022 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.0830   -2.7828    0.8938 H   0  0  0  0  0  0  0  0  0  0  0  0
```

Let’s take another look at benzoic acid. Clear the 2D sketch window using the white box button at the top, second from the left, and then draw benzoic acid. Click the right arrow button. That should render a 3D structure in the window to the right and generate a MOL file in the text window below. (For details on how where this data comes from, see “2D to 3D” and “3D to structure data” sections in “How it Works.”)

Now, take a look at the MOL file in the text window. You will note that, as a default, Hack-a-Mol includes explicit H in the MOL files it generates. (See discussion of explicit and implicit H earlier in this module for more information.)

Identify the atoms and bonds that make up the ring. (These will vary depending on the way that you drew the molecule – the 2D sketch application numbers atoms and bonds in the order that they are drawn.) Remember, the first two columns in each bond table entry refer to rows in the atom table, and the third column gives the bond type (1=single, 2=double, etc.) connecting these two atoms. (You can check yourself by hovering over atoms in the 3D window or clicking the “labels” link above this window.)

Once you have identified the six ring bonds in the MOL file, manually adjust them to generate the other Kekulé structure of the ring. (That is, switch the 1’s for 2’s and the 2’s for 1’s in the bond type fields (third column) of the bond table entries for the six ring bonds.) With the cursor still in the text window, press enter. This should generate the other Kekulé structure for benzoic acid in both the 3D and 2D windows.

Just for kicks, let’s generate a nonsense structure. Change all of the ring bonds to double bonds, and press enter. You should now have a chemically-offensive structure involving a cyclohexahexene ring with six positively charged carbon atoms violating valence rules. There’s a lesson here – software won’t tell you that your structure data is chemically nonsensical unless it is programmed to do so.

Revert to benzoic acid, either by changing the bonds back manually or just by clearing the 2D sketch window, re-drawing, and clicking the right arrow button again.

Now, let’s stick a chlorine atom onto the benzene ring. Using the atom and bond tables, locate the atom table entry for a ring hydrogen ortho, meta, or para to the carboxyl group (your pick!). Change the atom symbol in this atom table entry from H to Cl, and press enter. You should now have the chlorobenzoic acid isomer of your choice in both 3D and 2D windows.

One more exercise: let’s make our benzoic acid into pyridine-3-carboxylic acid – that is, benzoic acid with N in place of one of the ring carbons meta to the carboxylic group. This is the compound better known as niacin (vitamin B3).

(Tangential fun fact: niacin, discovered as an acidic reaction product of nicotine, was originally named nicotinic acid. In the 1930s, it was found to be the essential nutrient that prevented pellagra, a devastating disorder widely prevalent in the American South in the early twentieth century. Public health officials promoted enriching flour with nicotinic acid, and the epidemic of pellagra began to disappear. However, physicians and scientists worried that the name "nicotinic acid" gave the impression that they were curing mass disease by putting tobacco into bread. A National Research Council committee decided to change the name of the substance to niacin, short for nicotinic acid vitamin.)

Anyway: locate the entry for a ring carbon meta to the carboxyl group. (Hint: 1) use the atom and bond tables to identify the carbon atom bonded to the two oxygen atoms; 2) find the ring carbon bonded to that carboxyl carbon; 3) find a ring carbon two bonds away from that carboxyl-substituted ring carbon.) Change that carbon to N, and press enter.

Now we have the N atom in our ring, but you will notice that it's positively charged. We didn't change any of the explicit hydrogens, so the N atom remains protonated, like the C atom that it replaced. Let's get rid of that hydrogen atom. Locate the entry for the N-H bond in the bond table and the entry for the corresponding H atom in the atom table, and delete both of them. Press enter.

Unless you were very lucky, you should now have a monstrous mess in the 3D window and nothing at all in the 2D window. Uh-oh. Go back to the MOL file window, press ctrl-Z twice to undo the deletion of those rows, and press enter. That will take you back to N-protonated niacin.

By deleting a row of the atom table, we renumbered all of the subsequent atom table entries. Since we didn't change the atom references in the bond table, this broke all of the bonds to these renumbered atoms.

Once again, delete that N-H bond from the bond table and the entry for that H atom in the atom table. However, now fix the bond table references by **decreasing the atom number by 1** for all atoms below the row that you deleted. (That is, if the hydrogen that you deleted was the 13th atom table entry, change each 14 in the first two columns of the bond table to a 13, and change each 15 in the first two columns of the bond table to a 14.)

Hit enter. Ugh – your structure is probably screwed up **again**, even if you did all of this renumbering correctly. You may even have lost your ring, for some reason.

Take a look at the counts line of the MOL file – the row above the atom table, just below the file headers. The first two numbers in this line refer to the number of atoms and bonds in the molecule. Since we deleted an atom and a bond, we need to decrease each of these from 15 to 14. Do so, and then press enter again. You should now have niacin.

Whew. **Thank goodness that connection table handling is so amenable to automation!**

Play around some more with Hack-a-Mol. Take a look at the "How it Works" page – a lot of the notations, apps, and processes referred to on this page will be covered in subsequent weeks. You may find it useful to continue to come back to this page and play around with it as you move on in this course.

EXERCISE

1. Does Hack-A-Mol handle the number 4 for an aromatic bond? How can you tell? Can you create a chemically sound but non-aromatic structure using 4s in the bond field?

2. Perfluorinated octanoic acid (PFOA) is a surfactant that played a key role for a long time in the manufacture of fluorinated polymers including Teflon. Over the past decade, it has been the subject of significant public health concern and a whole bunch of litigation.

Pull PFOA into Hack-a-Mol by typing it into the text search box below the 3D window and clicking "search."

2a. Edit the mole file to defluorinate PFOA, converting it into octanoic acid.

2b. Now make it into acetic acid. (It is possible to do this in a way that yields correct-looking 2D and 3D renderings without changing any XYZ coordinately, but you have to be ***very*** careful about how you delete and relabel atoms and bonds.)

FURTHER READING

- https://en.wikipedia.org/wiki/Chemical_table_file
- CTFile Formats, June 2005, Elsevier/MDL, https://web.archive.org/web/20070630061308/http://www.mdl.com/downloads/public/ctfile/ctfile.pdf (Documentation for v2000 MOL file and related chemical table file formats.)
- Hack-a-Mol: https://chemapps.stolaf.edu/jmol/jsmol/hackamol.htm
- (Documentation: https://chemapps.stolaf.edu/jmol/docs/misc/hackamolworkings.pdf)

2.6: Chemical Resolvers, Molecular Editors and Visualization

Introduction

Chemical resolvers can take one form of molecular representation and convert it to another. That is, they can resolve what the compound is from its represenation. They can be web based services or software applications. A molecular editor is in essence a type of resolver, that has a graphical editor interface where human's can draw molecules. But under the hood, it is using cheminformatics representations like connection tables. Database services like PubChem and ChemSpider also have integrated editors and resolvers and so the distinction across these is a bit fuzzy.

Chemical Resolvers

We will define chemical resolvers as programs that can resolve a chemical structure from a representation, and then use that to transform it to another representation or provide information on the chemical.

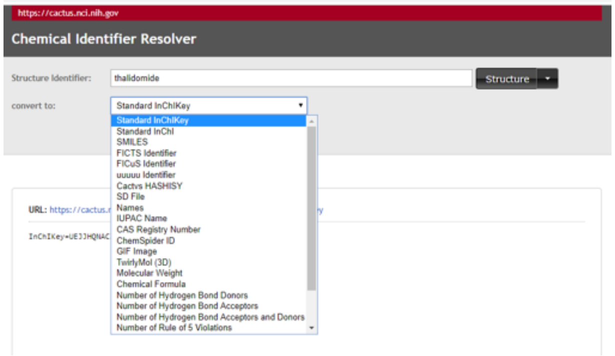
Web-Based Resolvers

These are services that typically offer both a GUI (for humans) and an API (for machines). This list is not comprehensive, and will grow as time allows.

CIR

The **Chemical Identifier Resolver** (CIR) is a service of the Computer-Aided Drug Design (CADD) group of the Chemical Biology Laboratory (CBL) of the National Cancer Institute (NCI) in Maryland US. The direct link to CIR is here:

https://cactus.nci.nih.gov/chemical/structure



https://cactus.nci.nih.gov

Chemical Identifier Resolver

Structure Identifier: thalidomide

convert to: Standard InChIKey

URL: https://cactus.n

InChIKey=UEJJHQNAC

Figure $2.6.1$: The GUI interface for the NCI/CADD CIR (https://cactus.nci.nih.gov/chemical/structure).

The CIR service also offers a variety of API interfaces and we already explored one of them with the InChILayersExplorer of activity 2.4.1 (section 2.4.3.3.1), where an Excel spreadsheet used the CIR service to convert a name to an InChI.

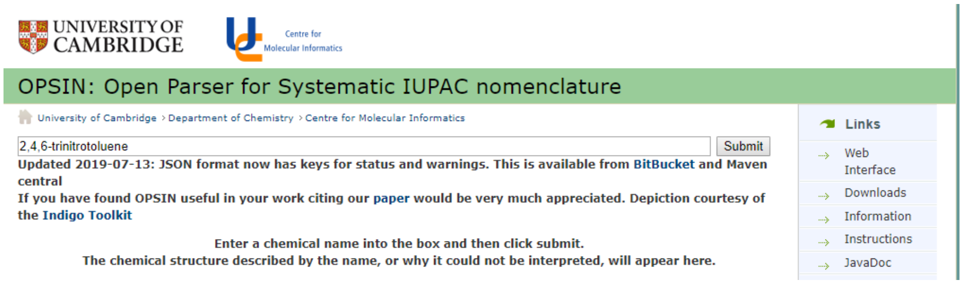
There are also a variety of other resources available through the **CA**DD Group **C**hemiformatics **T**ools and **U**ser **S**ervices (CACTUS) that students are encouraged to explore.

https://cactus.nci.nih.gov/

OPSIN

The **Open Parser for Systematic IUPAC Nomenclature** (OPSIN) is run by the Centre for Molecular Informatics and the University of Cambridge in England, the URL is:

https://opsin.ch.cam.ac.uk/



OPSIN: Open Parser for Systematic IUPAC nomenclature

University of Cambridge › Department of Chemistry › Centre for Molecular Informatics

2,4,6-trinitrotoluene

Updated 2019-07-13: JSON format now has keys for status and warnings. This is available from BitBucket and Maven central

If you have found OPSIN useful in your work citing our paper would be very much appreciated. Depiction courtesy of the Indigo Toolkit

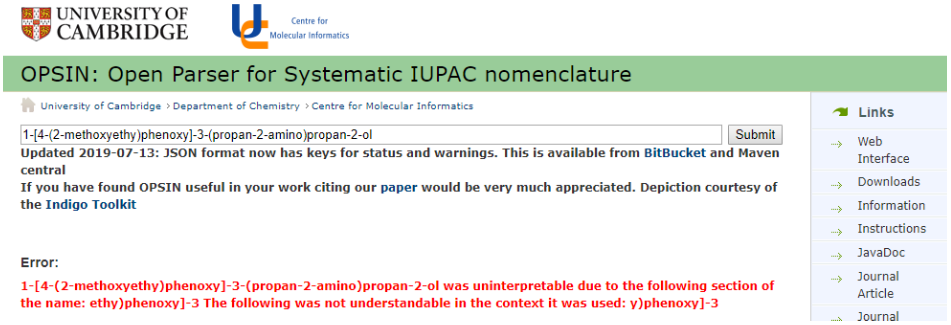
Enter a chemical name into the box and then click submit.
The chemical structure described by the name, or why it could not be interpreted, will appear here.

Figure 2.6.2: OPSIN resolver, (https://opsin.ch.cam.ac.uk/)

One of the nice things of the OPSIN resolver is that if you have an incorrect IUPAC name, it stops where it can't parse the name and tell's you where the problem is. For example, patents will often use IUPAC names and believe it or not, they are often misspelled and wrong! So lets look at this name here,

1-[4-(2-methoxyethy)phenoxy]-3-(propan-2-amino)propan-2-ol.

Do you see what is wrong with it? Well pasting it into OPSIN gives you an error and an idea where to look!



OPSIN: Open Parser for Systematic IUPAC nomenclature

University of Cambridge › Department of Chemistry › Centre for Molecular Informatics

1-[4-(2-methoxyethy)phenoxy]-3-(propan-2-amino)propan-2-ol

Updated 2019-07-13: JSON format now has keys for status and warnings. This is available from BitBucket and Maven central

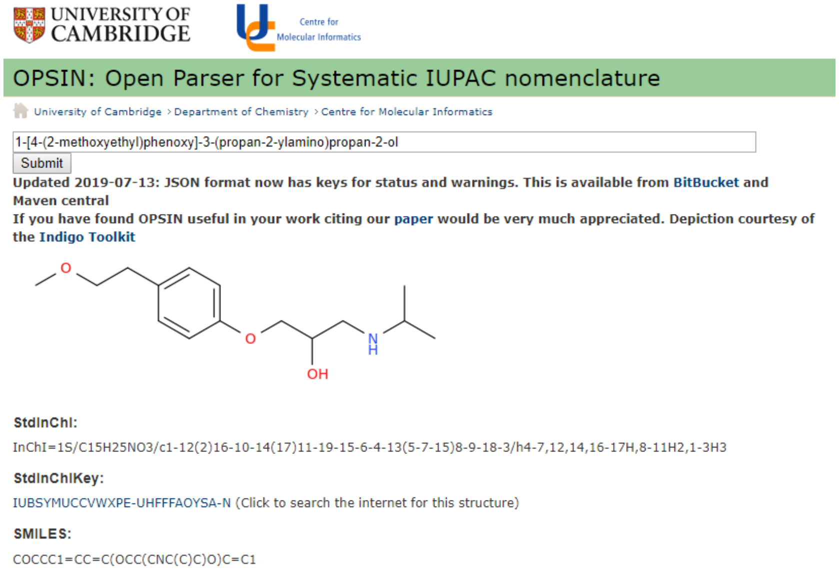
If you have found OPSIN useful in your work citing our paper would be very much appreciated. Depiction courtesy of the Indigo Toolkit

Error:

1-[4-(2-methoxyethy)phenoxy]-3-(propan-2-amino)propan-2-ol was uninterpretable due to the following section of the name: ethy)phenoxy]-3 The following was not understandable in the context it was used: y)phenoxy]-3

Figure 2.6.3: OPSIN resolver showing an error in resolving the IUPAC name.

The beauty is the error message occurs when the resolver could no longer parse the word and it got stumped at the y), which should have been yl), as in 1-[4-(2-methoxyethyl)phenoxy]-3-(propan-2-ylamino)propan-2-ol.



OPSIN: Open Parser for Systematic IUPAC nomenclature

University of Cambridge › Department of Chemistry › Centre for Molecular Informatics

1-[4-(2-methoxyethyl)phenoxy]-3-(propan-2-ylamino)propan-2-ol

Updated 2019-07-13: JSON format now has keys for status and warnings. This is available from BitBucket and Maven central

If you have found OPSIN useful in your work citing our paper would be very much appreciated. Depiction courtesy of the Indigo Toolkit

StdInChI:

InChI=1S/C15H25NO3/c1-12(2)16-10-14(17)11-19-15-6-4-13(5-7-15)8-9-18-3/h4-7,12,14,16-17H,8-11H2,1-3H3

StdInChIKey:

IUBSYMUCCVWXPE-UHFFFAOYSA-N (Click to search the internet for this structure)

SMILES:

COCCC1=CC=C(OCC(CNC(C)C)O)C=C1

Figure 2.6.4: Discovering the type in figure 3 allowed us to correct the IUPAC name and we now have identifiers for this compound.

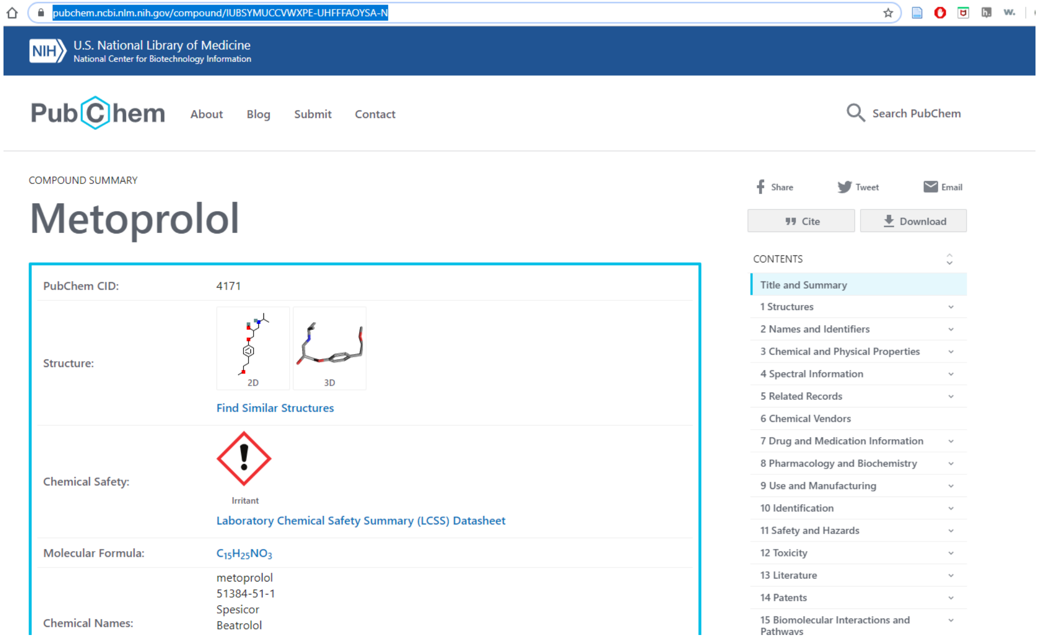
The above figures show how the OPSIN GUI can be used. In activity X we will look at a Google Spreadsheet that uses the OPSIN resolver to convert a list of IUPAC names to InChIKeys and then use those keys to link directly to PubChem on those chemicals, and thus instantly get access to information on a list of compounds. Lets manually do that now. That is, we append the InChI key to following URL stem

https://pubchem.ncbi.nlm.nih.gov/compound/INCHIKEY

and so the following link gets us to information on this compound

https://pubchem.ncbi.nlm.nih.gov/compound/IUBSYMUCCVWXPE-UHFFFAOYSA-N

bringing us to the following information on PubChem



pubchem.ncbi.nlm.nih.gov/compound/IUBSYMUCCVWXPE-UHFFFAOYSA-N

NIH U.S. National Library of Medicine
National Center for Biotechnology Information

PubChem About Blog Submit Contact Search PubChem

COMPOUND SUMMARY

Metoprolol

Share Tweet Email
Cite Download

PubChem CID:	4171
Structure:	2D 3D Find Similar Structures
Chemical Safety:	Irritant Laboratory Chemical Safety Summary (LCSS) Datasheet
Molecular Formula:	$C_{15}H_{25}NO_3$
Chemical Names:	metoprolol 51384-51-1 Spesicor Beatrolol

CONTENTS
- Title and Summary
- 1 Structures
- 2 Names and Identifiers
- 3 Chemical and Physical Properties
- 4 Spectral Information
- 5 Related Records
- 6 Chemical Vendors
- 7 Drug and Medication Information
- 8 Pharmacology and Biochemistry
- 9 Use and Manufacturing
- 10 Identification
- 11 Safety and Hazards
- 12 Toxicity
- 13 Literature
- 14 Patents
- 15 Biomolecular Interactions and Pathways

Figure 2.6.5: After using OPSIN to correct the incorrect IUPAC name we can deduce the compound is Metoprolol and obtain information on it.

IUPAC Name2PubChem

Go to the InChI OER (https://www.inchi-trust.org/oer/) and filter Content Type/Spreadsheet with File Type/Google Sheet, and click on the hit for the "IUPAC Name2PubChem". From here you will find a Google sheet that automates the above process for a list of chemicals. Video 2.6.1 shows how you can go to the OPSIN chemical resolver and figure out how to create a Google sheet that performs these functions.

Video 2.6.1: 6:47 min YouTube video describing how to connect an column of IUPAC names to information in PubChem (https://youtu.be/oDxMUJ0dNWw)

If you download the module from the InChI OER you will see the code described in the video which you can cut and paste into a spreadsheet. This module uses two webservices functions, the IMPORTDATA function and the HYPERLINK function, along with the concatenate function for a string of text, &. It is very important that you develop the skill of looking at code you have never seen and then finding out what it does through web searches.

Resolver Programs

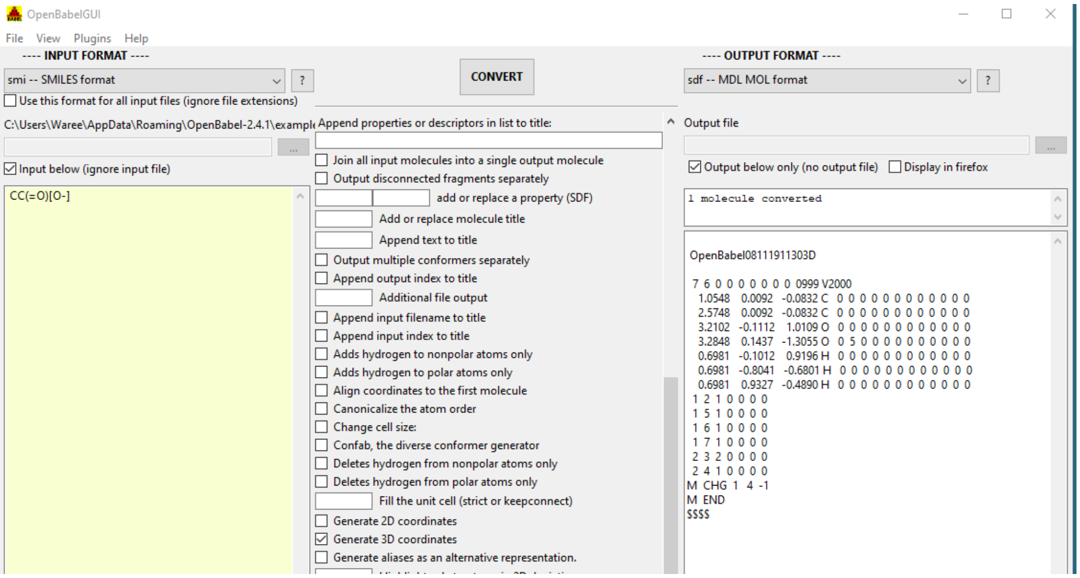
OK, Resolver Programs may not be the best title for this section. These are software packages that convert molecular file formats, and are often called tool kits. These have many functionalities that can be used offline, in contrast to web services, which need internet access.

Open Babel

Open Babel is an open source Chemistry Toolbox and does much more than just convert structural formats and can be downloaded from the following URL.

http://openbabel.org

In figure $2.6.6$ we see using Open Babel to convert a SMILES string to a mol file with 3D coordinates. The Open Babel Documentation can give you a feel for some of the things you can do with Open Babel, http://openbabel.org/docs/current/OpenBabel.pdf .



---- INPUT FORMAT ----

smi -- SMILES format

☐ Use this format for all input files (ignore file extensions)

C:\Users\Waree\AppData\Roaming\OpenBabel-2.4.1\example

☑ Input below (ignore input file)

```
CC(=O)[O-]
```

Append properties or descriptors in list to title:

☐ Join all input molecules into a single output molecule
☐ Output disconnected fragments separately
add or replace a property (SDF)
Add or replace molecule title
Append text to title
☐ Output multiple conformers separately
☐ Append output index to title
Additional file output
☐ Append input filename to title
☐ Append input index to title
☐ Adds hydrogen to nonpolar atoms only
☐ Adds hydrogen to polar atoms only
☐ Align coordinates to the first molecule
☐ Canonicalize the atom order
☐ Change cell size:
☐ Confab, the diverse conformer generator
☐ Deletes hydrogen from nonpolar atoms only
☐ Deletes hydrogen from polar atoms only
Fill the unit cell (strict or keepconnect)
☐ Generate 2D coordinates
☑ Generate 3D coordinates
☐ Generate aliases as an alternative representation.

---- OUTPUT FORMAT ----

sdf -- MDL MOL format

Output file

☑ Output below only (no output file) ☐ Display in firefox

```
1 molecule converted
```

```

OpenBabel08111911303D

 7  6  0  0  0  0  0  0  0  0999 V2000
    1.0548    0.0092   -0.0832 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.5748    0.0092   -0.0832 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.2102   -0.1112    1.0109 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.2848    0.1437   -1.3055 O   0  5  0  0  0  0  0  0  0  0  0  0
    0.6981   -0.1012    0.9196 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.6981   -0.8041   -0.6801 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.6981    0.9327   -0.4890 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  5  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  2  3  2  0  0  0  0
  2  4  1  0  0  0  0
M  CHG  1   4  -1
M  END
$$$$
```

Figure $2.6.6$: Using Open Babel to convert between file types, here converting a SMILES string to a 3D mol file with coordinates.

Molecular Editors

Web Services

1. MolView - site maintained by Herman Bergwerf
   - http://molview.org/
2. PubChem Sketcher
   - https://pubchem.ncbi.nlm.nih.gov/#draw=true
3. ChemSpider Structure Search
   - https://www.chemspider.com/StructureSearch.aspx
4. Chemagic
   - https://chemagic.org/molecules/amini.html
5. Wikipedia Chemical Structure Explorer
   - http://www.cheminfo.org/Wikipedia/
   - see JChem Inf. article (https://jcheminf.biomedcentral.com/articles/10.1186/s13321-015-0061-y)

Software Packages

Dr. Tamas Gunda has posted a decent resource: Chemical Drawing Programs - The Comparison of Accelrys (Biovia) Draw, ChemBioDraw (ChemDraw), DrawIt, ChemDoodle and Chemistry 4-D Draw (http://www.gunda.hu/dprogs/index.html)

Open Source

1. JChemPaint - open source (LGLP license):
   - https://jchempaint.github.io/
2. BIOVIA Draw – free for academic use
   - http://accelrys.com/products/collaborative-science/biovia-draw/draw-no-fee.php
3. ChemSketch Freeware - free for academic and personal use, requires registration and download:
   - http://www.acdlabs.com/resources/freeware/chemsketch/
4. BIO-RAD - Chemical Structure drawing, spectral analysis & more
   - http://www.knowitall.com/academic/
5. JSME - open source (BSD license):
   - http://peter-ertl.com/jsme/

Fee-Based

1. ChemDraw – requires subscription and download
   - Basic drawing package: http://www.cambridgesoft.com/Ensemble_for_Chemistry/ChemDraw/ChemDrawPrime/Default.aspx
2. ChemDoodle – requires purchase and download
   - https://www.chemdoodle.com/
   - Free Trial: https://www.chemdoodle.com/free-trial/
3. ChemSketch – requires purchase and download
   - http://www.acdlabs.com/products/draw_nom/draw/chemsketch/
   - Free Trial at above link
4. Chemistry 4-D Draw
   - http://www.cheminnovation.com/products/chem4d.asp

Molecular Visualization

1. Jmol and Jsmol
   - http://jmol.sourceforge.net/
2. Avogadro
   - https://avogadro.cc/

2.7: Python Assignment

There are two assignments to this chapter

Compound vs Substance

Downloadable Files

 lecture04_Standardization

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

- Requests
- RDKit
- time
- PIL (?)
- IPython.Display

- **Python Assignment 2A**
  - Lecture02_structure_input.ipynb
  - (Assignment has additional sdf files)
- Python Assignment 2B
  - Lecture03-list-conversion.ipynb

Chemical Structure Inputs for PUG-REST

Downloadable Files

▯ **Lecture02_structure_input.ipynb**

▯ lecture02_exb_compound1.sdf

▯ lecture02_exb_compound2.sdf

▯ lecture02_exb_compound3.sdf

▯ lecture02_exb_compound4.sdf

▯ lecture02_exb_compound5.sdf

▯ Structure2D_CID_5288826.sdf

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor for the link to join the discussion group.
- The SDF files will be needed to complete the assignment

Objectives

- Use SMILES and InChI strings to specify the input compound for a PUG-REST request.
- Use a structure-data (SD) file to specify the input compound for a PUG-REST request.
- Learn to submit a PUG-REST request using the HTTP-POST method.

You can use a chemical structure as an input for a PUG-REST request. PUG-REST accepts some popular chemical structure line notations such as SMILES and InChI strings. It is also possible to use an Structure-Data File (SDF) as a structure input.

To learn how to specify the structure input in a PUG-REST request, one needs to know that there are two methods by which data are transferred from clients (users) and servers (PubChem) through PUG-REST. Discussing what these methods are in detail is beyond the scope of this material, and it is enough to know three things:

- When you make a PUG-REST request by typing the request URL in the address bar of your web browser (such as Google Chrome, MS Internet Explorer), the HTTP GET method is used
- The HTTP GET method transfers information encoded in a single-line URL.
- Some chemical structure inputs are not appropriate to encode in a single-line URL (because they may contain special characters not compatible with the URL syntax, span over multiple lines, or too long), and the HTTP POST needs to be used for such cases.

For more information on HTTP GET and POST, read the following documents.

- HTTP request methods (https://www.w3schools.com/tags/ref_httpmethods.asp)
- Get vs. POST (https://www.diffen.com/difference/GET-vs-POST-HTTP-Requests)

Here, import the Requests library, necessary to make web service requests to PubChem.

In [1]:

```
import requests
```

Using the HTTP GET method.

Structure encoded in the URL path.

In some cases, you can encode a chemical structure in the PUG-REST request URL path as in the following example.

In [2]:

```
prolog = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug'
```

In [3]:

```
smiles1 = "CC(C)CC1=CC=C(C=C1)C(C)C(=O)O"
url = prolog + "/compound/smiles/" + smiles1 + "/cids/txt"
print(url)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC(C)CC1=CC=C(C=C1)C(C)C(=O
```

This request URL returns ibuprofen (CID 3672).

In [4]:

```
res = requests.get(url)
print(res.text)
```

```
3672
```

Try to run the following

In [5]:

```
smiles2 = "CC1=C([C@@](SC1=O)(C)/C=C(\C)/C=C)O"

url = prolog + "/compound/smiles/" + smiles2 + "/cids/txt"
res = requests.get(url)
print(res.text)
```

```
Status: 400
Code: PUGREST.BadRequest
Message: Unable to standardize the given structure - perhaps some special characters
Detail: error:
Detail: status: 400
Detail: output: Caught ncbi::CException: Standardization failed
Detail: Output Log:
Detail: Record 1: Warning: Cactvs Ensemble cannot be created from input string
Detail: Record 1: Error: Unable to convert input into a compound object
Detail:
Detail:
```

Note in the above example that the SMILES string contains special characters. In this case a forward slash ("/"), which is also used in the URL path. These special characters conflict with the PUG-REST request URL syntax, causing an error when used in the PUG-REST request URL.

Structure encoded as a URL argument

To circumvent the issue mentioned above, the SMILES string may be encoded as the URL argments (as an optional parameter followed by the "?" character).

In [6]:

```
url = prolog + "/compound/smiles/cids/txt?" + "smiles=" + smiles2
print(url)
res2 = requests.get(url)
print(res2.text)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1=C([C@@]
135403829
```

Structure passed in a dictionary

It is also possible to pass the structure query as a key-value pair in a dictionary.

Tutorial on Python Dictionaries

The following tutorial goes over python dictionaries: https://www.w3schools.com/python/python_dictionaries.asp

The following example does the same task as the previous example does.

In [7]:

```
url = prolog + "/compound/smiles/cids/txt"
struct = { 'smiles': smiles2 }
res3 = requests.get(url, params = struct)
print(res3.text)
```

```
135403829
```

The object returned from a web service request (res, res2, and res3 in our examples) contains information on the request URL through which the data have been retrieved. This information can be accessed using the ".url" attribute of the object, as shown in this example:

In [8]:

```
print(smiles2)     # the orginal smiles string unencoded
print(res2.url)    # from (request 2) structure encoded as a URL argument
print()
print(struct)      # to show the smiles string in the dictionary is unencoded for URL
print(res3.url)    # from (request 3) structure passed in a dictionary
```

```
CC1=C([C@@](SC1=O)(C)/C=C(\C)/C=C)O
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1=C(%5BC@(
{'smiles': 'CC1=C([C@@](SC1=O)(C)/C=C(\\C)/C=C)O'}
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1%3DC%28%
```

From these two URLs, we can see two important things:

- When the structure is passed using a key-value pair in a dictionary (i.e., "res3"), the structure is automatically encoded as a URL argument (after the "?" mark).
- When the structure is passed in a dictionary, the special characters in the SMILES string are converted according to the URL encoding rules: https://www.w3schools.com/tags/ref_urlencode.asp. [for example, the equal sign "=" changes into "%3D", and "(" into "%28", "/" into "%2F", etc]

It illustrates that the last two approaches using HTTP GET are essentially the same.

Exercise 1a Retrieve (in the CSV (comma-separated values) format) the Hydrogen bond donor and acceptor counts, TPSA, and XLogP of the chemical represented by the SMILES string: "C1=CC(=C(C=C1Cl)O)OC2=C(C=C(C=C2)Cl)Cl". When you construct a PUG-REST url for this request, encode the structure in the URL path.

In [9]:

```
# Write your code in this cell.
```

Exercise 1b Get the CID corresponding to the following InChI string, using the HTTP GET method. Pay attention to the case sensitivity of the URL parameter part after the "?" mark.

In [10]:

```
inchi = "InChI=1S/C17H14O4S/c1-22(19,20)14-9-7-12(8-10-14)15-11-21-17(18)16(15)13-5-3-2-4-6-13/h2-10H,11H2,1H3"
```

In [11]:

```
# Write your code in this cell
```

Using the HTTP POST method

Comparison of HTTP POST and GET

All the three examples above use the HTTP GET method, as implied in the use of "requests.get()". Alternatively, one can use the HTTP POST method. For example, the following example returns the identical result as the last two HTTP GET examples.

In [12]:

```
url = prolog + "/compound/smiles/cids/txt"
struct = { 'smiles': smiles2 }
res = requests.post(url, params = struct)    # the SMILES as a URL parameter
print(res.url)
print(res.text)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1%3DC%28%
135403829
```

In [13]:

```
url = prolog + "/compound/smiles/cids/txt"
```

```
struct = { 'smiles': smiles2 }
res = requests.post(url, data = struct)     # the SMILES as data
print(res.url)
print(res.text)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt
135403829
```

Note that the first one passes the input SMILES string as a parameter, while the second one passes it as data. Because of this, the URL stored in the "res.url" variable in the second code does not contain structure informaion.

HTTP POST for multi-line structure input

The HTTP POST method should be used if the input molecular structure for PUG-REST request span over multiple lines (e.g., stored in a structure-data file (SDF) format). The SDF file contains structure information of a molecule in a multi-line format, along with other data.

In [14]:

```
mysdf = '''1983
  -OEChem-07241917072D

 20 20  0     0  0  0  0  0  0999 V2000
    2.8660   -2.5950    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    4.5981    1.4050    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    1.4050    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    0.4050    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320   -0.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000   -0.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320   -1.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000   -1.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660   -1.5950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320    1.9050    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320    2.9050    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    4.2690    0.2150    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.4631    0.2150    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.3291    1.7150    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    4.2690   -1.4050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.4631   -1.4050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    4.3520    2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320    3.5250    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.1120    2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.3291   -2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  9  1  0  0  0  0
  1 20  1  0  0  0  0
  2 10  2  0  0  0  0
  3  4  1  0  0  0  0
  3 10  1  0  0  0  0
  3 14  1  0  0  0  0
  4  5  2  0  0  0  0
  4  6  1  0  0  0  0
  5  7  1  0  0  0  0
  5 12  1  0  0  0  0
  6  8  2  0  0  0  0
  6 13  1  0  0  0  0
  7  9  2  0  0  0  0
  7 15  1  0  0  0  0
  8  9  1  0  0  0  0
  8 16  1  0  0  0  0
 10 11  1  0  0  0  0
 11 17  1  0  0  0  0
 11 18  1  0  0  0  0
 11 19  1  0  0  0  0
M  END
> <PUBCHEM_COMPOUND_CID>
1983

> <PUBCHEM_COMPOUND_CANONICALIZED>
1

> <PUBCHEM_CACTVS_COMPLEXITY>
139

> <PUBCHEM_CACTVS_HBOND_ACCEPTOR>
2

> <PUBCHEM_CACTVS_HBOND_DONOR>
2

> <PUBCHEM_CACTVS_ROTATABLE_BOND>
1
$$$$
'''
```

In this example, the triple quotes (''') are used to enclose a multi-line string. This multi-line sdf data is used as an input for a PUG-REST request through the HTTP POST.

In [19]:

```
url = prolog + "/compound/sdf/cids/txt"
mydata = { 'sdf': mysdf }
res = requests.post(url, data=mydata) # the multiline sdf as URL data
print(res.url)
print(res.text)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/sdf/cids/txt
1983
```

HTTP POST for SDF file input

One may want to use the structure stored in a file as the input for a PUG-REST request. The following code shows how to read an SDF file into a variable. This code cell assumes that the 'Structure2D_CID_5288826.sdf' file is in the current directory. The file can be downloaded from the **Chapter 2 Assignments** page.

In [20]:

```
with open('Structure2D_CID_5288826.sdf', 'r') as file:
    mysdf = file.read()

print(mysdf)
```

```
5288826
  -OEChem-08171913162D

 40 44  0     1  0  0  0  0  0999 V2000
    2.2314    0.0528    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000   -2.4021    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000    2.4021    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    6.1607   -0.9511    0.0000 N   0  0  3  0  0  0  0  0  0  0  0  0
    3.6897   -0.4755    0.0000 C   0  0  1  0  0  0  0  0  0  0  0  0
    4.5133   -0.9511    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
    5.3370   -0.4755    0.0000 C   0  0  1  0  0  0  0  0  0  0  0  0
    2.8660   -0.9511    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
    4.2392    0.2219    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.6897    0.4755    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    5.3370    0.4755    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    5.5918    0.2219    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    4.5133    0.9511    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660   -1.9022    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
    4.5133   -1.9022    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    0.9511    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.6897   -2.3777    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    6.8418   -1.6832    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    4.5133    1.9022    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    1.9022    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.6897    2.3777    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    5.0597   -1.6022    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    5.6284   -1.2740    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.0496   -1.1875    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    4.3760    0.8266    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.6795    0.4887    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    5.9476    0.3679    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    5.5490    1.0581    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    6.1840    0.4057    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    5.4989    0.8349    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660   -2.5222    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    5.0503   -2.2122    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.6897   -2.9977    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    6.3879   -2.1055    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    7.2641   -2.1371    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    7.2957   -1.2609    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    5.0503    2.2122    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.6897    2.9977    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000   -3.0222    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000    3.0222    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  8  1  0  0  0  0
  1 16  1  0  0  0  0
 14  2  1  6  0  0  0
  2 39  1  0  0  0  0
  3 20  1  0  0  0  0
  3 40  1  0  0  0  0
  4  7  1  0  0  0  0
  4 12  1  0  0  0  0
  4 18  1  0  0  0  0
  5  6  1  0  0  0  0
  5  8  1  0  0  0  0
  5  9  1  1  0  0  0
  5 10  1  0  0  0  0
  6  7  1  0  0  0  0
  6 15  1  0  0  0  0
  6 22  1  1  0  0  0
  7 11  1  0  0  0  0
  7 23  1  6  0  0  0
  8 14  1  0  0  0  0
  8 24  1  1  0  0  0
  9 12  1  0  0  0  0
  9 25  1  0  0  0  0
  9 26  1  0  0  0  0
 10 13  2  0  0  0  0
 10 16  1  0  0  0  0
 11 13  1  0  0  0  0
 11 27  1  0  0  0  0
 11 28  1  0  0  0  0
 12 29  1  0  0  0  0
 12 30  1  0  0  0  0
 13 19  1  0  0  0  0
 14 17  1  0  0  0  0
 14 31  1  0  0  0  0
 15 17  2  0  0  0  0
 15 32  1  0  0  0  0
 16 20  2  0  0  0  0
 17 33  1  0  0  0  0
 18 34  1  0  0  0  0
 18 35  1  0  0  0  0
 18 36  1  0  0  0  0
 19 21  2  0  0  0  0
 19 37  1  0  0  0  0
 20 21  1  0  0  0  0
 21 38  1  0  0  0  0
M  END
> <PUBCHEM_COMPOUND_CID>
5288826

> <PUBCHEM_COMPOUND_CANONICALIZED>
1

> <PUBCHEM_CACTVS_COMPLEXITY>
494

> <PUBCHEM_CACTVS_HBOND_ACCEPTOR>
4

> <PUBCHEM_CACTVS_HBOND_DONOR>
2

> <PUBCHEM_CACTVS_ROTATABLE_BOND>
0

> <PUBCHEM_CACTVS_SUBSKEYS>
AAADceB6MAAAAAAAAAAAAAAAAAAAASAAAAA8YIEAAAAWAEjBAAAAHgAACAAADzzhmAYyBoMABgCAAiBCAAACC

> <PUBCHEM_IUPAC_OPENEYE_NAME>
(4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,2-e

> <PUBCHEM_IUPAC_CAS_NAME>
(4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,2-e

> <PUBCHEM_IUPAC_NAME_MARKUP>
(4<I>R</I>,4<I>a</I><I>R</I>,7<I>S</I>,7<I>a</I><I>R</I>,12<I>b</I><I>S</I>)-3-methyl

> <PUBCHEM_IUPAC_NAME>
(4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,2-e

> <PUBCHEM_IUPAC_SYSTEMATIC_NAME>
(4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,2-e

> <PUBCHEM_IUPAC_TRADITIONAL_NAME>
(4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,2-e

> <PUBCHEM_IUPAC_INCHI>
```

```
InChI=1S/C17H19NO3/c1-18-7-6-17-10-3-5-13(20)16(17)21-15-12(19)4-2-9(14(15)17)8-11(10

> <PUBCHEM_IUPAC_INCHIKEY>
BQJCRHHNABKAKU-KBQPJGBKSA-N

> <PUBCHEM_XLOGP3>
0.8

> <PUBCHEM_EXACT_MASS>
285.136493

> <PUBCHEM_MOLECULAR_FORMULA>
C17H19NO3

> <PUBCHEM_MOLECULAR_WEIGHT>
285.34

> <PUBCHEM_OPENEYE_CAN_SMILES>
CN1CCC23C4C1CC5=C2C(=C(C=C5)O)OC3C(C=C4)O

> <PUBCHEM_OPENEYE_ISO_SMILES>
CN1CC[C@]23[C@@H]4[C@H]1CC5=C2C(=C(C=C5)O)O[C@H]3[C@H](C=C4)O

> <PUBCHEM_CACTVS_TPSA>
52.9

> <PUBCHEM_MONOISOTOPIC_WEIGHT>
285.136493

> <PUBCHEM_TOTAL_CHARGE>
0

> <PUBCHEM_HEAVY_ATOM_COUNT>
21

> <PUBCHEM_ATOM_DEF_STEREO_COUNT>
5

> <PUBCHEM_ATOM_UDEF_STEREO_COUNT>
0

> <PUBCHEM_BOND_DEF_STEREO_COUNT>
0

> <PUBCHEM_BOND_UDEF_STEREO_COUNT>
0
```

```
> <PUBCHEM_ISOTOPIC_ATOM_COUNT>
0

> <PUBCHEM_COMPONENT_COUNT>
1

> <PUBCHEM_CACTVS_TAUTO_COUNT>
-1

> <PUBCHEM_COORDINATE_TYPE>
1
5
255

> <PUBCHEM_BONDANNOTATIONS>
10  13  8
10  16  8
13  19  8
16  20  8
19  21  8
14  2  6
20  21  8
5  9  5
6  22  5
7  23  6
8  24  5

$$$$
```

Now the structure stored in the "mysdf" can be used in a PUG-REST request through HTTP-POST. For example, the code cell below shows how to retrieve various names (also called "synonyms") of the input structure.

In [17]:

```
url = prolog + "/compound/sdf/synonyms/txt"
mydata = { 'sdf': mysdf }
res = requests.post(url, data=mydata)
print(res.text)
```

```
morphine
Morphia
Morphinum
Morphium
Morphina
Morphin
(-)-Morphine
Duromorph
```

```
MS Contin
DepoDur
Meconium
Morphinism
Moscontin
Ospalivina
Morfina
l-Morphine
Dulcontin
Nepenthe
Roxanol
Kadian
57-27-2
MORPHINE SULFATE
Infumorph
Dreamer
Morpho
Avinza
Hocus
Unkie
Cube juice
Hard stuff
Oramorph SR
Statex SR
M-Eslon
Ms Emma
Morphin [German]
Morfina [Italian]
Duramorph
Morphina [Italian]
Morphine [BAN]
Astramorph PF
Duramorph PF
CCRIS 5762
Dolcontin
HSDB 2134
(5R,6S,9R,13S,14R)-4,5-Epoxy-N-methyl-7-morphinen-3,6-diol
UNII-76I7G6D29C
D-(-)-Morphine
CHEBI:17303
CHEMBL70
EINECS 200-320-2
4,5alpha-Epoxy-17-methyl-7-morphinen-3,6alpha-diol
7,8-Didehydro-4,5-epoxy-17-methyl-morphinan-3,6-diol
(7R,7AS,12BS)-3-METHYL-2,3,4,4A,7,7A-HEXAHYDRO-1H-4,12-METHANO[1]BENZOFURO[3,2-E]ISOQU
DEA No. 9300
Morphine Anhydrate
```

```
76I7G6D29C
(5alpha,6alpha)-17-methyl-7,8-didehydro-4,5-epoxymorphinan-3,6-diol
Morphine (BAN)
Morphine Forte
RMS
Morphine H.P
(5alpha,6alpha)-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
Morphinan-3,6-alpha-diol, 7,8-didehydro-4,5-alpha-epoxy-17-methyl-
Morphine Extra-Forte
Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl-, (5alpha,6alpha)-
9H-9,9c-Iminoethanophenanthro(4,5-bcd)furan-3,5-diol, 4a,5,7a,8-tetrahydro-12-methyl-
methyl[?]diol
Aguettant
Dinamorf
Sevredol
Dimorf
MOI
Epimorph
Morphitec
Oramorph
Rescudose
Statex Drops
OMS Concentrate
RMS Uniserts
Roxanol UD
(Morphine)
Substitol (TN)
Mscontin, Oramorph
(4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,2-e
(-)-(etorphine)
MSIR
Roxanol 100
(-)Morphine sulfate
Morfina Dosa (TN)
SDZ202-250
NSC11441
SDZ 202-250
MS/L
MS/S
Epitope ID:116646
Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl- (5alpha,6alpha)-
SCHEMBL2997
M.O.S
BIDD:GT0147
GTPL1627
DTXSID9023336
Morphine 0.1 mg/ml in Methanol
Morphine 1.0 mg/ml in Methanol
BQJCRHHNABKAKU-KBQPJGBKSA-N
ZINC3812983
BDBM50000092
AKOS015966554
DB00295
AN-23579
AN-23737
LS-91748
C01516
D08233
7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
UNII-1M5VY6ITRT component BQJCRHHNABKAKU-KBQPJGBKSA-N
17-methyl-7,8-didehydro-4,5alpha-epoxymorphinan-3,6alpha-diol
7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol(morphine)
(5A,6A)-7,8-DIDEHYDRO-4,5-EPOXY-17-METHYLMORPHINIAN-3,6-DIOL
(5alpha,6alpha)-7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
(5alpha,6beta)-17-methyl-7,8-didehydro-4,5-epoxymorphinan-3,6-diol
3-(4-Hydroxy-phenyl)-1-propyl-piperidine-3-carboxylic acid ethyl ester
6-tert-Butyl-3-methyl-1,2,3,4,5,6-hexahydro-2,6-methano-benzo[d]azocine
(-)(5.alpha.,6.alpha.)-7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl- (5..alpha.,6.alpha.)-
Morphine solution, 1.0 mg/mL in methanol, ampule of 1 mL, certified reference materia
(1S,5R,13R,14S)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15
(1S,5R,13R,14S,17R)-4-methyl-12-oxa-4-azapentacyclo[9.6.1.0;{1,13}.0;{5,17}.0;{7,18}]
(1S,5R,13R,14S,17R)-4-methyl-12-oxa-4-azapentacyclo[9.6.1.0^{1,13}.0^{5,17}.0^{7,18}]
(morphine) 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,1
2-{4-[2,4-diamino-6-pteridinylmethyl(methyl)amino]phenylcarboxamido}pentanedioic acid
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-
4-methyl-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15-tetrae
4-methyl-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15-tetrae
4-methyl-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15-tetrae
6,11-Dimethyl-3-(3-methyl-but-2-enyl)-1,2,3,4,5,6-hexahydro-2,6-methano-benzo[d]azoci
9H-9,9c-Iminoethanophenanthro(4,5-bcd)furan-3,5-diol, 4alpha,5,7alpha,8-tetrahydro-12
MORPHINE, (5A,6A)-7,8-DIDEHYDRO-4,5-EPOXY-17-METHYLMORPHINIAN-3,6-DIOL, MORPHIUM, MOR
Morphine;4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]
```

Exercise 2a Retrieve (in the **CSV** format) the XlogP, molecular weight, hydrogen bond donor count, hydrogen bond aceptor count, and TPSA of the compounds contained in the five sdf files below, which can be downloaded from the **Chapter 2 Assignments** page.

- Use a for loop to retrieve the data for each compound.
- Import the time package and add "time.sleep(0.2)" to sleep 0.2 seconds after retrieving the data for each compound.
- Refer to the "lecture 1" notebook to see how to merge the multiple CSV outputs into a single CSV output.

In [18]:

```
files = ['lecture02_ex2b_compound1.sdf','lecture02_ex2b_compound2.sdf','lecture02_ex2
         'lecture02_ex2b_compound4.sdf','lecture02_ex2b_compound5.sdf']
```

In [ ]:

```
# Write your code in this cell.
```

2.7.2: Python Assignment 2B

Interconversion between PubChem records

Downloadable Files

Lecture03_list_conversion.ipynb

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

PUG-REST can be used to retrieve PubChem records related to another PubChem records. Basically, PUG-REST takes an input list of records in one of the three PubChem databases (Compound, Substance, and BioAssay) and returns a list of the related records in the same or different database. Here, the meaning of the relationship between the input and output records may be specified using an optional parameter. This allows one to do various tasks, including (but not limited to):

- Depositor-provided records (i.e., substances) that are standardized to a given compound.
- Mixture compounds that contain a given component compound.
- Stereoisomers/isotopomers of a given compound.
- Compounds that are tested to be active in a given assay.
- Compounds that have similar structures to a given compound.

Getting depositor-provided records for a given compound

First let's import the requests package necessary to make a web service request.

In [1]:

```python
import requests
```

The code snippet below retrieves the substance record associated with a given CID (CID 129825914).

In [2]:

```python
prolog    = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

pr_input  = "compound/cid/129825914"
pr_oper   = "sids"
pr_output = "txt"
url       = prolog + '/' + pr_input + '/' + pr_oper + '/' + pr_output

res = requests.get(url)
print(res.text)
```

```
341669951
```

It is also possible to provide a comma separated list of CIDs as input identifiers.

In [3]:

```python
pugin   = "compound/cid/129825914,129742624,129783988"
pugoper = "sids"
pugout  = "txt"
url     = prolog + '/' + pugin + '/' + pugoper + '/' + pugout

res = requests.get(url)
print(res.text)
```

```
341669951
341492923
341577059
345261280
368769438
```

In the example above, the input list has three CIDs, but the PUG-REST request returned five SIDs. It means that some CID(s) must be associated with multiple SIDs, but it is hard to see which CID it is. Therefore, we want the SIDs grouped by the corresponding CIDs. This can be done using the optional parameter "**list_return=grouped**" and changing the output format to **json**.

In [4]:

```python
pugin   = "compound/cid/129825914,129742624,129783988"
pugoper = "sids"
pugout  = "json"
pugopt  = "list_return=grouped"
url     = prolog + '/' + pugin + '/' + pugoper + '/' + pugout + "?" + pugopt

res = requests.get(url)
print(res.text)
```

```
{
  "InformationList": {
    "Information": [
      {
        "CID": 129825914,
        "SID": [
          341669951
        ]
      },
      {
        "CID": 129742624,
        "SID": [
          341492923
        ]
      },
      {
        "CID": 129783988,
        "SID": [
          341577059,
          345261280,
          368769438
        ]
      }
    ]
  }
}
```

Note that the **json** output format is used in the above request. The "**txt**" output format in PUG-REST returns data into a single column but the result from the above request cannot fit well into a single column.

If you want output records to be "flattened", rather than being grouped by the input identifiers, use "**list_return=flat**".

In [5]:

```
pugopt  = "list_return=flat"
url     = prolog + '/' + pugin + '/' + pugoper + '/' + pugout + "?" + pugopt

res = requests.get(url)
print(res.text)
```

```
{
  "IdentifierList": {
    "SID": [
      341492923,
      341577059,
      341669951,
      345261280,
      368769438
    ]
  }
}
```

The default value for the "list_return" parameter is:

- "flat" when the output format is TXT
- "grouped" when the output format is JSON and XML

It is also possible to specify the input list **implicitly**, rather than providing the input identifiers explicitly. For example, the following example uses a chemical name to specify the input list.

In [6]:

```
# Input CIDs are provided using a chemical name
url = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/lactose/cids/txt'
res = requests.get(url)
cids = res.text.split()
print("# CIDs returned:", len(cids))
print(",".join(cids))

# Input CIDs are provided using the name, then coverted to SIDs.
url = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/lactose/sids/txt'
```

```
res = requests.get(url)
sids = res.text.split()
print("# SIDs returned (method 1):", len(sids))
#print(",".join(sids))

# Input *SIDs* are provided using the name, and returned the input SIDs.
url = 'https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/name/lactose/sids/txt'
res = requests.get(url)
sids = res.text.split()
print("# SIDs returned (method 2):", len(sids))
#print(",".join(sids))
```

```
# CIDs returned: 7
6134,440995,84571,294,439186,49837892,69301022
# SIDs returned (method 1): 419
# SIDs returned (method 2): 125
```

The above example illustrates how the list conversion works.

- In the first request, the name "lactose" is searched for against the Compound database and the resulting 7 CIDs are returned.
- If you change the operation part from "cids" to "sids" (as in the second request), the same name search is done first against the **Compound** database, followed by the list conversion from the resulting 7 CIDs to associted 415 SIDs.
- In the third request, the name search is performed against the **Substance** database, and the resulting 125 SIDs are returned.

Exercise 1a Statins are a class of drugs that lower cholesterol levels in the blood. Retrieve in **JSON** the substance records associated with the compounds whose names contain the string "statin".

- Make only one PUG-REST request.
- For partial name matching, set the *name_type* parameter to "word" (See the PUG-REST document for an example).
- Group the substances by the corresponding compound records.
- Print the json output using print()

In [7]:

```
# Write your code in this cell.
```

Getting mixture/component molecules for a given molecule.

The list interconversion may be used to retrieve mixtures that contain a given molecule as a component. To do this, the input molecule should be a single-component compound (that is, with only one covalently-bound unit), and the optional parameter "**cids_type=component**" should be provided.

In [8]:

```
prolog    = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

url = prolog + "/compound/name/tylenol/cids/txt?cids_type=component"
res = requests.get(url)
cids = res.text.split()
print(len(cids))
print( cids )
```

```
349
['137528085', '137524012', '136090259', '134821662', '134821661', '134539182', '13223
```

It should be noted that, if the input molecule is a multi-component compound, the option "**cids_type=component**" returns the components of that compound. For example, the following example shows how to get all components of the first molecule in the "cids" list generated in the previous example.

In [9]:

```
url = prolog + "/compound/cid/" + cids[0] + "/cids/txt?cids_type=component"
res = requests.get(url)
component_cids = res.text.split()
print( "CID:", cids[0])
print( "Number of Components", len(component_cids))
print( component_cids )
```

```
CID: 137528085
Number of Components 3
['446155', '4891', '1983']
```

Exercise 2a: Many over-the-counter drugs contain more than one active ingredients. In this exercise, we want to find component molecules that occur with three common pain killers (aspirin, tylenol, advil) as a mixture.

Step 1. Define a list that contains three drug names (aspirin, tylenol, advil).

In [10]:

```
# Write your code in this cell.
```

Step 2. Using a for loop, retrieve PubChem CIDs corresponding to the three drugs and store them in a new list. In order not to overload the PubChem servers, stop the program for 0.2 second for each iteration in the for loop (using sleep()).

In [11]:

```
# Write your code in this cell.
```

Step 3. Using another for loop, do the following things for each drug:

- Get the PubChem CIDs of the mixture compounds that contain each drug and store them in a list.
- Get the PubChem CIDs of the components that occur in any of the returned mixtures, by setting the "list_return" parameter to "flat". This can be done with a single request.
- Print all the components.
- Stop the code for 0.2 second using sleep() each time a PUG-REST request is made.

In [12]:

```
# Write your code in this cell.
```

Getting compounds tested in a given assay

PUG-REST may be used to retrieve compounds tested in a given assay. For example, the following code cell shows how to get all compounds tested in AID 1207599.

In [13]:

```
url = prolog + "/assay/aid/" + "1207599" + "/cids/txt"
res = requests.get(url)
cids = res.text.split()
print(len(cids))
print(cids)
```

```
791
['6175', '6197', '8547', '10219', '14169', '17558', '21389', '68050', '84677', '95783
```

If you are interested in only the compounds that are tested "active" in a given assay, set the "**cids_type**" parameter to "**active**", as shown in the code below.

In [14]:

```
url = prolog + "/assay/aid/" + "1207599" + "/cids/txt?cids_type=active"
res = requests.get(url)
cids = res.text.split()
print(len(cids))
print(cids)
```

```
435
['6197', '10219', '14169', '17558', '68050', '177894', '182792', '253602', '348623',
```

It is also possible to specify the input assay list implicitly. For example, the following code cell retrieves compounds tested in any assays targeting human Carbonic anhydrase 2 (CA2), whose accession number is P00918.

In [15]:

```
url = prolog + "/assay/target/accession/" + "P00918" + "/cids/txt"
res = requests.get(url)
cids = res.text.split()
print(len(cids))
#print(cids)
```

```
23978
```

Exercise 3a: Find compounds that are tested to be active against human acetylcholinesterase (accession: P08173) and retrieve SMILES strings for those compounds.

- Split the CID list into smaller chunks (with a chunk size of 100).
- Print the retrieved data in a CSV format (CID and SMILES strings in the first and second columns, respectively).

In [16]:

```
# Write your code in this cell.
```

In [ ]:

```

```

2.8: R Assignment

2.8.1: R Assignment 2A

Exploring Chemical Identity

J. Cuadros

August 5th, 2019

Downloadable Files

L02_ChemicalIdentity.Rmd

 RL02_ChemIdentity.pynb

- You can use the R-studio you created in section 1.4 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
- This page is an html version of the above R file.
  - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. If you are not on the discussion group you should contact your instructor for the link to join.
- .pynb is a Jupyter Notebook that opens with an R Kernal

Objectives

- Understand the problem of chemical identity.
- Explore some chemical substances identifiers.
- Understand the layered model of the InChI as a model to chemical identity.

1. The Problem of Chemical Identity

Since the XVIII century, when chemists started to understand chemical substances have a fixed composition, we have faced the need to identify and discriminate them. In the analogic word, we use names (both traditional and systematic) and formulas to fullfill this role. In the digital world, registry numbers and line notations are commonly used for substance identification.

Chemistry information and documentation recurrently face the problem of having to classify substance records (data pieces) in a systematic way. But as we will explore in this activity, this is trickier than it seems.

Let's start facing the problem. You are given 18 substance records (that's not much, PubChem holds more than 200 milions of these, https://pubchemdocs.ncbi.nlm.nih.gov/statistics) and you are asked to decide which correspond to the same substance (and why).

Here are the records:

- https://pubchem.ncbi.nlm.nih.gov/substance/227885365
- https://pubchem.ncbi.nlm.nih.gov/substance/242744695
- https://pubchem.ncbi.nlm.nih.gov/substance/329830556
- https://pubchem.ncbi.nlm.nih.gov/substance/341141642
- https://pubchem.ncbi.nlm.nih.gov/substance/341193751
- https://pubchem.ncbi.nlm.nih.gov/substance/342898240
- https://pubchem.ncbi.nlm.nih.gov/substance/355138175
- https://pubchem.ncbi.nlm.nih.gov/substance/355178551
- https://pubchem.ncbi.nlm.nih.gov/substance/369730804
- https://pubchem.ncbi.nlm.nih.gov/substance/376125581
- https://pubchem.ncbi.nlm.nih.gov/substance/376145687
- https://pubchem.ncbi.nlm.nih.gov/substance/376602811
- https://pubchem.ncbi.nlm.nih.gov/substance/383210891
- https://pubchem.ncbi.nlm.nih.gov/substance/383219135
- https://pubchem.ncbi.nlm.nih.gov/substance/383428756
- https://pubchem.ncbi.nlm.nih.gov/substance/384452886
- https://pubchem.ncbi.nlm.nih.gov/substance/384647147

- https://pubchem.ncbi.nlm.nih.gov/substance/385112115

Exercise 1a: Browse these records and make a table that includes, for each record, the given name, its molecular formula and its structural formula.

```
sids <- c(227885365, 242744695, 329830556, 341141642, 341193751, 342898240, 355138175
         376602811, 383210891, 383219135, 383428756, 384452886, 384647147, 385112115)

paste("# Number of SIDs:", length(sids) )

pugrest <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper <- "cids"
pugout  <- "txt"

pugin <- paste("substance/sid/", paste(sids[1:length(sids)],collapse=","),sep="")
url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
cids <- readLines(url)

cids <- (unique(cids))
paste("# Number of CIDs:", length(cids) )

pugin <- paste("compound/cid/", paste(cids[1:length(cids)],collapse=","),sep="")
pugoper <- "property/IUPACName,MolecularFormula,IsomericSMILES"
pugout  <- "csv"
url <- paste(pugrest,pugin,pugoper,pugout,sep="/")

df <- read.table(url,sep=",",header=TRUE)
print(df)
```

run restart restart & run all

Exercise 1b: How many different substances do you think there are in this set? How would you classify them?

2. Digital Identifiers

In the digital world, identity is usually associated with the use of an identifier; when two identifiers coincide when we can say that both data pieces belong to the same entiy.

For substance, and besides registry numbers, two common identifiers are SMILES and InChI.

Exercise 2a: For each record, check if the data providers included any SMILES or InChI information. Collect this information when available.

Exercise 2b: For each record, use a molecular drawing program to compute the SMILES, the standard InChI and the InChIKey. Make a table with them. If you don’t have a molecular drawing program at hand, you may consider using MolView (http://molview.org/) or the drawing tool included in the Chemical Identifier Resolver (https://cactus.nci.nih.gov/chemical/structure).

Exercise 2c: Compare the provider identifiers with the computed ones. Do you see any differences?

Exercise 2d: Would you reconsider the classification you decided in exercise 1b?

3. The InChI Layered Notation and Identity Matching

If you look carefully at the InChI for the different records, you will notice that some of the identifiers are more similar then others. Some match completely, while some others may match only for some of the layers, especially for the main layer. Sometimes, we consider to be the same substance, any substance where the InChI main layer is coincident. For other applications, some other layers need to be taken into account; for instance, stereochemical information is critical in health-related uses.

Exercise 3a: Classify the records according to the main layer of the InChI.

Exercise 3b: Classify the records again according to the full InChI.

Substance records in PubChem are grouped into compound records. This information appears in each one of the elements of the set.

Exercise 3c: Compare the classification used in PubChem with the InChI-based classifications done in 3a and 3b.

2.8.2: R Assignment 2B

Chemical Structure Inputs for PUG-REST

S. Kim, J. Cuadros
August 21st, 2019

Downloadable Files

▯ L02_StructureInputs.rmd
▯ L02_StructureInputs_wc.rmd
▯ RL02_StructureInputs.pynb

- Note: If clicking on the file opens it in your browser you should rightclick and "save link as" to download the file.
- You can use the R-studio you created in section 1.4 for the R file, your version of Jupyter or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
- This page is an html version of the above R file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. If you are not on the discussion group you should contact your instructor for the link to join.
- _wc is the R file with comments.
- .pynb is a Jupyter Notebook that opens with an R Kernal

Objectives

- Use SMILES and InChI strings to specify the input compound for a PUG-REST request.
- Use a structure-data (SD) file to specify the input compound for a PUG-REST request.
- Learn to submit a PUG-REST request using the HTTP-POST method.

Background

You can use a chemical structure as an input for a PUG-REST request. PUG-REST accepts some popular chemical structure line notations such as SMILES and InChI strings. It is also possible to use an Structure-Data File (SDF) as a structure input.

To learn how to specify the structure input in a PUG-REST request, one needs to know that there are two methods by which data are transferred from clients (users) and servers (PubChem) through PUG-REST. Discussing what these methods are in detail is beyond the scope of this material, and it is enough to know three things:

- When you make a PUG-REST request by typing the request URL in the address bar of your web browser (such as Google Chrome, MS Internet Explorer), the HTTP GET method is used
- The HTTP GET method transfers information encoded in a single-line URL.
- Some chemical structure inputs are not appropriate to encode in a single-line URL (because they may contain special characters not compatible with the URL syntax, span over multiple lines, or too long), and the HTTP POST needs to be used for such cases.

For more information on HTTP GET and POST, read the following documents.

- HTTP request methods (https://www.w3schools.com/tags/ref_httpmethods.asp)
- GET vs. POST (https://www.diffen.com/difference/GET-vs-POST-HTTP-Requests)

Let’s start checking if the `httr` package is available, installing it if needed. Then, we load it.

```
if(!require("httr", quietly=TRUE)) {
  install.packages("httr", repos="https://cloud.r-project.org/",
         quiet=TRUE, type="binary")
  library("httr", quietly=TRUE)
}
```

```
## Warning: package 'httr' was built under R version 3.6.1
```

Packages are the way that libraries (additional functions, data types, constants, data sets…) are distributed in the R environment. In order to use a package, it has to be installed (only once per running environment) with the `install.packages` function. Then, if we load it (in a specific R session) using `library` , its functions can be called as they were in the base environment.

The `require` function checks whether a package has already been installed and loads it if so. It returns a logical value than can be used to install the package if it was not available.

The `httr` package allows a finer grade manipulation of the HTTP communications. That's why we will use it in this activity. A quick introduction to the package is available at https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html.

1. Using the HTTP GET method.

1.1. Structure encoded in the URL path.

In some cases, you can encode a chemical structure in the PUG-REST request URL path as in the following example.

```
prolog <- 'https://pubchem.ncbi.nlm.nih.gov/rest/pug'
smiles1 <- "CC(C)CC1=CC=C(C=C1)C(C)C(=O)O"
url <- paste(prolog, "/compound/smiles/", smiles1, "/cids/txt", sep="")
url
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC(C)CC1=CC=C(C=C1
```

```
res = GET(url)
content(res,"text",encoding="UTF-8")
```

```
## [1] "3672\n"
```

It should be noteworthy that some SMILES strings contain special characters, such as the forward slash ("/"), which is also used in the URL path. These special characters conflict with the PUG-REST request URL syntax, causing an error when used in the PUG-REST request URL.

Also note that the backslash character has to be escaped when used in a string in R. To include a blackslash, we have to write `\\` .

```
smiles2 <- "CC1=C([C@@](SC1=O)(C)/C=C(\\C)/C=C)O"
url <- paste(prolog, "/compound/smiles/", smiles2, "/cids/txt", sep="")
url
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/CC1=C([C@@](SC1=O)
```

```
res <- GET(url)
content(res,"text",encoding="UTF-8")
```

```
## [1] "Status: 400\nCode: PUGREST.BadRequest\nMessage: Unable to standardize the gi
```

1.2. Structure encoded as a URL argument

To circumvent the issue mentioned above, the SMILES string may be encoded as the URL argments (as an optional parameter followed by the "?" character).

```
url2 <- paste(prolog, "/compound/smiles/cids/txt?smiles=", smiles2, sep="")
url2
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1
```

```
res2 <- GET(url2)
content(res2,"text",encoding="UTF-8")
```

```
## [1] "135403829\n"
```

1.3. Structure encoded as a URL-encoded URL argument

It is also possible to pass the structure query as a URL-encoded argument. The following example does the same task as the previous example does.

This is safer in case the argument includes `&` , `?` or other reserved characters.

```
url3 <- paste(prolog, "/compound/smiles/cids/txt?smiles=", URLencode(smiles2,reserved
url3
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1
```

```
res3 <- GET(url3)
content(res3,"text",encoding="UTF-8")
```

```
## [1] "135403829\n"
```

The object returned from a web service request (res, res2, and res3 in our examples) contains information on the request URL through which the data have been retrieved. This information can be accessed using the `$url` attribute of the object, as shown in this example:

```
res2$url   # from (2) structure encoded as a URL argument
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1
```

```
res3$url   # from (3) structure encoded as a URL-encoded URL argument
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt?smiles=CC1
```

Note that URL-encoding does not work for PubChem when including structure information in the URL path.

Exercise 1a: Retrieve the hydrogen bond donor and acceptor counts, TPSA, and XLogP of the chemical represented by the SMILES string: “C1=CC(=C(C=C1Cl)O)OC2=C(C=C(C=C2)Cl)Cl”. When construct a PUG-REST url for this request, encode the structure in the URL path.

```
# Write your code here
```

Exercise 1b: Get the CID corresponding to the following InChI string, using the HTTP GET method.

```
inchi <- "InChI=1S/C17H14O4S/c1-22(19,20)14-9-7-12(8-10-14)15-11-21-17(18)16(15)13-5-

# Write your code here
```

2. Using the HTTP POST method

2.1. Comparison of HTTP POST and GET

All the three examples above use the HTTP GET method, as implied in the use of the `GET` function. Alternatively, one can use the HTTP POST method. For example, the following example returns the identical result as the last two HTTP GET examples.

```
url <- paste(prolog, "/compound/smiles/cids/txt", sep="")
struct <- list(smiles=smiles2)

res <- POST(url, body = struct)
res$url
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/smiles/cids/txt"
```

```
content(res,"text",encoding="UTF-8")
```

```
## [1] "135403829\n"
```

When using the HTTP POST method, information is passed as data through the body argument. Because of this, the URL stored in the `res$url` does not contain structure informaion.

2.2. HTTP POST for multi-line structure input

The HTTP POST method should be used if the input molecular structure for PUG-REST request span over multiple lines (e.g., stored in a structure-data file (SDF) format). The SDF file contains structure information of a molecule in a multi-line format, along with other data.

```
mysdf <- "1983
  -OEChem-07241917072D

 20 20  0     0  0  0  0  0  0999 V2000
    2.8660   -2.5950    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    4.5981    1.4050    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    1.4050    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660    0.4050    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320   -0.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000   -0.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320   -1.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.0000   -1.0950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.8660   -1.5950    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320    1.9050    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320    2.9050    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    4.2690    0.2150    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.4631    0.2150    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.3291    1.7150    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    4.2690   -1.4050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.4631   -1.4050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    4.3520    2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    4.3320    2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.7320    3.5250    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.1120    2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.3291   -2.9050    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  9  1  0  0  0  0
  1 20  1  0  0  0  0
  2 10  2  0  0  0  0
  3  4  1  0  0  0  0
  3 10  1  0  0  0  0
  3 14  1  0  0  0  0
  4  5  2  0  0  0  0
  4  6  1  0  0  0  0
  5  7  1  0  0  0  0
  5 12  1  0  0  0  0
  6  8  2  0  0  0  0
  6 13  1  0  0  0  0
  7  9  2  0  0  0  0
  7 15  1  0  0  0  0
  8  9  1  0  0  0  0
  8 16  1  0  0  0  0
 10 11  1  0  0  0  0
 11 17  1  0  0  0  0
 11 18  1  0  0  0  0
 11 19  1  0  0  0  0
M  END
> <PUBCHEM_COMPOUND_CID>
1983

> <PUBCHEM_COMPOUND_CANONICALIZED>
1

> <PUBCHEM_CACTVS_COMPLEXITY>
139

> <PUBCHEM_CACTVS_HBOND_ACCEPTOR>
2

> <PUBCHEM_CACTVS_HBOND_DONOR>
2

> <PUBCHEM_CACTVS_ROTATABLE_BOND>
1
$$$$
"
```

Multi-line string in R can be written without any special consideration. This multi-line sdf data is used as an input for a PUG-REST request through the HTTP POST.

```
url <- paste(prolog, "/compound/sdf/cids/txt", sep="")
mydata <- list(sdf=mysdf)

res <- POST(url, body = mydata)
res$url
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/sdf/cids/txt"
```

```
content(res,"text",encoding="UTF-8")
```

```
## [1] "1983\n"
```

Note that HTTP POST should be used for the input specification using a SDF format. Although HTTP GET may work if data is URL-encoded, it will be more dependent of URL length limitations.

```
url3 <- paste(prolog, "/compound/sdf/cids/txt?sdf=", URLencode(mysdf,reserved=TRUE),
url3
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/sdf/cids/txt?sdf=1983%0A%
```

```
res3 <- GET(url3)
content(res3,"text",encoding="UTF-8")
```

```
## [1] "1983\n"
```

2.3. HTTP POST for multi-line structure input

One may want to use the structure stored in a file as the input for a PUG-REST request. The following code shows how to read an SDF file into a variable.

```
mysdf <- paste(readLines('Structure2D_CID_5288826.sdf'),collapse="\n")
cat(mysdf)
```

```
## 5288826
##   -OEChem-08171913162D
##
##  40 44  0     1  0  0  0  0  0999 V2000
##     2.2314    0.0528    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
##     2.0000   -2.4021    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
##     2.0000    2.4021    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
##     6.1607   -0.9511    0.0000 N   0  0  3  0  0  0  0  0  0  0  0  0
##     3.6897   -0.4755    0.0000 C   0  0  1  0  0  0  0  0  0  0  0  0
##     4.5133   -0.9511    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
##     5.3370   -0.4755    0.0000 C   0  0  1  0  0  0  0  0  0  0  0  0
##     2.8660   -0.9511    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
##     4.2392    0.2219    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     3.6897    0.4755    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     5.3370    0.4755    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     5.5918    0.2219    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     4.5133    0.9511    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     2.8660   -1.9022    0.0000 C   0  0  2  0  0  0  0  0  0  0  0  0
##     4.5133   -1.9022    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     2.8660    0.9511    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##     3.6897   -2.3777    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##      6.8418   -1.6832    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##      4.5133    1.9022    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##      2.8660    1.9022    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##      3.6897    2.3777    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
##      5.0597   -1.6022    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      5.6284   -1.2740    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      2.0496   -1.1875    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      4.3760    0.8266    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      3.6795    0.4887    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      5.9476    0.3679    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      5.5490    1.0581    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      6.1840    0.4057    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      5.4989    0.8349    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      2.8660   -2.5222    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      5.0503   -2.2122    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      3.6897   -2.9977    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      6.3879   -2.1055    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      7.2641   -2.1371    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      7.2957   -1.2609    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      5.0503    2.2122    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      3.6897    2.9977    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      2.0000   -3.0222    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##      2.0000    3.0222    0.0000 H   0  0  0  0  0  0  0  0  0  0  0  0
##   1  8  1  0  0  0  0
##   1 16  1  0  0  0  0
##  14  2  1  6  0  0  0
##   2 39  1  0  0  0  0
##   3 20  1  0  0  0  0
##   3 40  1  0  0  0  0
##   4  7  1  0  0  0  0
##   4 12  1  0  0  0  0
##   4 18  1  0  0  0  0
##   5  6  1  0  0  0  0
##   5  8  1  0  0  0  0
##   5  9  1  1  0  0  0
##   5 10  1  0  0  0  0
##   6  7  1  0  0  0  0
##   6 15  1  0  0  0  0
##   6 22  1  1  0  0  0
##   7 11  1  0  0  0  0
##   7 23  1  6  0  0  0
##   8 14  1  0  0  0  0
##   8 24  1  1  0  0  0
##   9 12  1  0  0  0  0
##   9 25  1  0  0  0  0
##   9 26  1  0  0  0  0
##  10 13  2  0  0  0  0
##  10 16  1  0  0  0  0
##  11 13  1  0  0  0  0
##  11 27  1  0  0  0  0
##  11 28  1  0  0  0  0
##  12 29  1  0  0  0  0
##  12 30  1  0  0  0  0
##  13 19  1  0  0  0  0
##  14 17  1  0  0  0  0
##  14 31  1  0  0  0  0
##  15 17  2  0  0  0  0
##  15 32  1  0  0  0  0
##  16 20  2  0  0  0  0
##  17 33  1  0  0  0  0
##  18 34  1  0  0  0  0
##  18 35  1  0  0  0  0
##  18 36  1  0  0  0  0
##  19 21  2  0  0  0  0
##  19 37  1  0  0  0  0
##  20 21  1  0  0  0  0
##  21 38  1  0  0  0  0
## M  END
## > <PUBCHEM_COMPOUND_CID>
## 5288826
##
## > <PUBCHEM_COMPOUND_CANONICALIZED>
## 1
##
## > <PUBCHEM_CACTVS_COMPLEXITY>
## 494
##
## > <PUBCHEM_CACTVS_HBOND_ACCEPTOR>
## 4
##
## > <PUBCHEM_CACTVS_HBOND_DONOR>
## 2
##
## > <PUBCHEM_CACTVS_ROTATABLE_BOND>
## 0
##
## > <PUBCHEM_CACTVS_SUBSKEYS>
## AAADceB6MAAAAAAAAAAAAAAAAAAAASAAAAA8YIEAAAAWAEjBAAAAHgAACAAADzzhmAYyBoMABgCAAiBCAA
##
## > <PUBCHEM_IUPAC_OPENEYE_NAME>
## (4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,
##
## > <PUBCHEM_IUPAC_CAS_NAME>
## (4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,
##
## > <PUBCHEM_IUPAC_NAME_MARKUP>
## (4<I>R</I>,4<I>a</I><I>R</I>,7<I>S</I>,7<I>a</I><I>R</I>,12<I>b</I><I>S</I>)-3-met
##
## > <PUBCHEM_IUPAC_NAME>
## (4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,
##
## > <PUBCHEM_IUPAC_SYSTEMATIC_NAME>
## (4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,
##
## > <PUBCHEM_IUPAC_TRADITIONAL_NAME>
## (4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,
##
## > <PUBCHEM_IUPAC_INCHI>
## InChI=1S/C17H19NO3/c1-18-7-6-17-10-3-5-13(20)16(17)21-15-12(19)4-2-9(14(15)17)8-11
##
## > <PUBCHEM_IUPAC_INCHIKEY>
## BQJCRHHNABKAKU-KBQPJGBKSA-N
##
## > <PUBCHEM_XLOGP3>
## 0.8
##
## > <PUBCHEM_EXACT_MASS>
## 285.136493
##
## > <PUBCHEM_MOLECULAR_FORMULA>
## C17H19NO3
##
## > <PUBCHEM_MOLECULAR_WEIGHT>
## 285.34
##
## > <PUBCHEM_OPENEYE_CAN_SMILES>
## CN1CCC23C4C1CC5=C2C(=C(C=C5)O)OC3C(C=C4)O
##
## > <PUBCHEM_OPENEYE_ISO_SMILES>
## CN1CC[C@]23[C@@H]4[C@H]1CC5=C2C(=C(C=C5)O)O[C@H]3[C@H](C=C4)O
##
## > <PUBCHEM_CACTVS_TPSA>
## 52.9
##
## > <PUBCHEM_MONOISOTOPIC_WEIGHT>
## 285.136493
##
## > <PUBCHEM_TOTAL_CHARGE>
## 0
##
## > <PUBCHEM_HEAVY_ATOM_COUNT>
## 21
##
## > <PUBCHEM_ATOM_DEF_STEREO_COUNT>
## 5
##
## > <PUBCHEM_ATOM_UDEF_STEREO_COUNT>
## 0
##
## > <PUBCHEM_BOND_DEF_STEREO_COUNT>
## 0
##
## > <PUBCHEM_BOND_UDEF_STEREO_COUNT>
## 0
##
## > <PUBCHEM_ISOTOPIC_ATOM_COUNT>
## 0
##
## > <PUBCHEM_COMPONENT_COUNT>
## 1
##
## > <PUBCHEM_CACTVS_TAUTO_COUNT>
## -1
##
## > <PUBCHEM_COORDINATE_TYPE>
## 1
## 5
## 255
##
## > <PUBCHEM_BONDANNOTATIONS>
## 10  13  8
## 10  16  8
## 13  19  8
## 16  20  8
## 19  21  8
## 14  2  6
## 20  21  8
## 5  9  5
## 6  22  5
## 7  23  6
## 8  24  5
##
## $$$$
```

`cat` is used in this example instead of `print` (which can be omitted) as it produces a nicer print-out for multiple-line strings.

Now the structure stored in the “mysdf” can be used in a PUG-REST request through HTTP-POST. For example, the code cell below shows how to retrieve various names (also called “synonyms”) of the input structure.

```
url <- paste(prolog, "/compound/sdf/synonyms/txt", sep="")
mydata <- list(sdf=mysdf)

res <- POST(url, body = mydata)
res$url
```

```
## [1] "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/sdf/synonyms/txt"
```

```
cat(content(res,"text",encoding="UTF-8"))
```

```
## morphine
## Morphia
## Morphinum
## Morphium
## Morphina
## Morphin
## (-)-Morphine
## Duromorph
## MS Contin
## DepoDur
## Meconium
## Morphinism
## Morphinism
## Moscontin
## Ospalivina
## Morfina
## l-Morphine
## Dulcontin
## Nepenthe
## Roxanol
## Kadian
## 57-27-2
## MORPHINE SULFATE
## Infumorph
## Dreamer
## Morpho
## Avinza
## Hocus
## Unkie
## Cube juice
## Hard stuff
## Oramorph SR
## Statex SR
## M-Eslon
## Ms Emma
## Morphin [German]
## Morfina [Italian]
## Duramorph
## Morphina [Italian]
## Morphine [BAN]
## Astramorph PF
## Duramorph PF
## CCRIS 5762
## Dolcontin
## HSDB 2134
## (5R,6S,9R,13S,14R)-4,5-Epoxy-N-methyl-7-morphinen-3,6-diol
## UNII-76I7G6D29C
## D-(-)-Morphine
## CHEBI:17303
## CHEMBL70
## EINECS 200-320-2
## 4,5alpha-Epoxy-17-methyl-7-morphinen-3,6alpha-diol
## 7,8-Didehydro-4,5-epoxy-17-methyl-morphinan-3,6-diol
## (7R,7AS,12BS)-3-METHYL-2,3,4,4A,7,7A-HEXAHYDRO-1H-4,12-METHANO[1]BENZOFURO[3,2-E]I
## DEA No. 9300
## Morphine Anhydrate
## 76I7G6D29C
## (5alpha,6alpha)-17-methyl-7,8-didehydro-4,5-epoxymorphinan-3,6-diol
## Morphine (BAN)
## Morphine Forte
## RMS
## Morphine H.P
## (5alpha,6alpha)-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
## Morphinan-3,6-alpha-diol, 7,8-didehydro-4,5-alpha-epoxy-17-methyl-
## Morphine Extra-Forte
## Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl-, (5alpha,6alpha)-
## Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl-, (5alpha,6alpha)-
## 9H-9,9c-Iminoethanophenanthro(4,5-bcd)furan-3,5-diol, 4a,5,7a,8-tetrahydro-12-meth
## methyl[?]diol
## Aguettant
## Dinamorf
## Sevredol
## Dimorf
## MOI
## Epimorph
## Morphitec
## Oramorph
## Rescudose
## Statex Drops
## OMS Concentrate
## RMS Uniserts
## Roxanol UD
## (Morphine)
## Substitol (TN)
## Mscontin, Oramorph
## (4R,4aR,7S,7aR,12bS)-3-methyl-2,4,4a,7,7a,13-hexahydro-1H-4,12-methanobenzofuro[3,
## (-)-(etorphine)
## MSIR
## Roxanol 100
## (-)Morphine sulfate
## Morfina Dosa (TN)
## SDZ202-250
## NSC11441
## SDZ 202-250
## MS/L
## MS/S
## Epitope ID:116646
## Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl- (5alpha,6alpha)-
## SCHEMBL2997
## M.O.S
## BIDD:GT0147
## GTPL1627
## DTXSID9023336
## Morphine 0.1 mg/ml in Methanol
## Morphine 1.0 mg/ml in Methanol
## BQJCRHHNABKAKU-KBQPJGBKSA-N
## ZINC3812983
## BDBM50000092
## AKOS015966554
## DB00295
## AN-23579
## AN-23737
## LS-91748
## C01516
## D08233
## 7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
## UNII-1M5VY6ITRT component BQJCRHHNABKAKU-KBQPJGBKSA-N
## 17-methyl-7,8-didehydro-4,5alpha-epoxymorphinan-3,6alpha-diol
## 7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol(morphine)
## (5A,6A)-7,8-DIDEHYDRO-4,5-EPOXY-17-METHYLMORPHINIAN-3,6-DIOL
## (5alpha,6alpha)-7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
## (5alpha,6beta)-17-methyl-7,8-didehydro-4,5-epoxymorphinan-3,6-diol
## 3-(4-Hydroxy-phenyl)-1-propyl-piperidine-3-carboxylic acid ethyl ester
## 6-tert-Butyl-3-methyl-1,2,3,4,5,6-hexahydro-2,6-methano-benzo[d]azocine
## (-)(5.alpha.,6.alpha.)-7,8-Didehydro-4,5-epoxy-17-methylmorphinan-3,6-diol
## Morphinan-3,6-diol, 7,8-didehydro-4,5-epoxy-17-methyl- (5..alpha.,6.alpha.)-
## Morphine solution, 1.0 mg/mL in methanol, ampule of 1 mL, certified reference mate
## (1S,5R,13R,14S)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10
## (1S,5R,13R,14S,17R)-4-methyl-12-oxa-4-azapentacyclo[9.6.1.0;{1,13}.0;{5,17}.0;{7,1
## (1S,5R,13R,14S,17R)-4-methyl-12-oxa-4-azapentacyclo[9.6.1.0^{1,13}.0^{5,17}.0^{7,1
## (morphine) 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.0
## 2-{4-[2,4-diamino-6-pteridinylmethyl(methyl)amino]phenylcarboxamido}pentanedioic a
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octade
## 4-methyl-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15-tet
## 4-methyl-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15-tet
## 4-methyl-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,18]octadeca-7(18),8,10,15-tet
## 6,11-Dimethyl-3-(3-methyl-but-2-enyl)-1,2,3,4,5,6-hexahydro-2,6-methano-benzo[d]az
## 9H-9,9c-Iminoethanophenanthro(4,5-bcd)furan-3,5-diol, 4alpha,5,7alpha,8-tetrahydro
## MORPHINE, (5A,6A)-7,8-DIDEHYDRO-4,5-EPOXY-17-METHYLMORPHINIAN-3,6-DIOL, MORPHIUM,
## Morphine;4-methyl-(1S,5R,13R,14S,17R)-12-oxa-4-azapentacyclo[9.6.1.01,13.05,17.07,
```

Exercise 2a: Retrieve (in the **CSV** format) the XlogP, molecular weight, hydrogen bond donor count, hydrogen bond aceptor count, and TPSA of the compounds contained in the five sdf files (link to the data files).

- Use a for loop to retrieve the data for each compound.
- Remember to add some sleep time (e.g 0.5 seconds) after retrieving the data for each compound.
- Refer to the **lecture 1** notebook to see how to merge the multiple CSV outputs into a single data frame.

```
files <- c('l02_ex2b_compound1.sdf','l02_ex2b_compound2.sdf','l02_ex2b_compound3.sdf
         'l02_ex2b_compound4.sdf','l02_ex2b_compound5.sdf')

# Write your code here
```

2.9: Mathematica Assignment

There are two parts to this assignment

- Mathematica Assignment 2A: Chemical Structure Inputs for PUG - REST
- Mathematica Assignment 2B: Interconversion Between PubChem Records

2.9.1: Mathematica Assignment 2A

2.9.2: Mathematica Assignment 2B

CHAPTER OVERVIEW

3: Database Resources in Cheminformatics

Hypothes.is Tag= f19OLCCc3

Note: Any annotation tagged **f19OLCCc3** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

This chapter will build on the topics introduced in section 3 of chapter 1 for this text through applications involving databases. The first few sections will cover the basics of using a simple database which will then be applied to the rest of the chapter by showing how this can be used when working with public chemical databases.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

Topic hierarchy

3.1: Database Basics

What is a database

A database is an “organized collection of information.” The information in a database can be in any format, including texts, numbers, images, audios, videos, and many others (and combination of these), but this information must be “organized” for efficient retrieval. According to this definition, a database is not necessarily electronic (i.e., accessible by computers). For example, the collection of names in a phone book or address book may also be considered as a database, because the names are arranged (typically in alphabetical order) to make it easy to search for necessary information (e.g., phone numbers or addresses). However, in computer science and related areas, a database usually means an electronic database. Therefore, the term “database” in this module is used to mean an “electronic database”.

Primary vs. secondary databases

Databases are often categorized into primary and secondary databases.

- **Primary databases** contain experimentally-derived data that are directly submitted by researchers (also called “primary data”). In essence, these databases serve as archives that keep original data. Therefore, they are also known as archival databases.
- **Secondary databases** contain secondary data, which are derived from analyzing and interpreting primary data. These databases often provide value-added information related to the primary data, by using information from other databases and scientific literature. Essentially, secondary databases serve as reference libraries for the scientific community, providing highly curated reviews about primary data. For this reason, they are also known as curated databases, or knowledgebase.

It should be noted that the distinction between primary and secondary databases is not always clear and that many databases have the characteristics of both primary and secondary database. It is very common that a primary database curates its data with information drawn from secondary databases. In addition, because many secondary databases make their value-added information available in the public domain, data exchange and integration among databases very frequently occurs. As a results, virtually all data providers also becomes data consumers these days.

Data provenance

The term “data provenance” refers to a record trail that describes the origin or source of a piece of data and the process by which it entered in a database.[1] Simply put, data provenance deals with the questions “where the data came from” and “how and why the data is in its present place”. Although the data provenance information is critical in the reliability of a data source (and its data), this information is not easy to manage. In addition, information predicted in one database may not be appropriate for use in other databases, but may end up being integrated in them anyway. Therefore, databases need to document the provenance of the data and devise a way to notify users of that information. In turn, users should always pay attention to the data provenance issue when using a database.

References

1. Ram, S.; Liu, J. In *SWPM'09 Proceedings of the First International Conference on Semantic Web in Provenance Management* Washington, D.C. , 2009; Vol. 526, p 35.

3.2: Database Management

Database Design

Planning out how a database will be structured is easily overlooked in the beginning stages of a development project. The key to this process is to start with a template that will allow for both future expansion of your project while also maintaining that only essential information is kept in data storage. The main reason for this is that many types of databases will become very resource intensive as the amount of data grows to a large size. There are several components of database design that will be explained in this section that should be kept in mind.

Storage Options

Databases are often stored on a hard disc which can also be from the same disc containing the applications file system or it may be on a separate drive altogether. For the purpose of this lesson plan, ignore any systems that use a type of RAM (Random Access Memory) storage for holding temporary databases. When choosing the storage options for a database, you want to look at the types of data that you will be working with. Text, numbers and chemical identifiers work well in almost every type of database. However, storing things like images for chemical structures and long text documents can quickly cause database bloating so it may be necessary to design a system that also relies on file storage. A good practice to get into on choosing your storage options is to only design the database to store things like text and numbers. When working with images and documents, assign these items to the file storage and only keep the object link stored in the database.

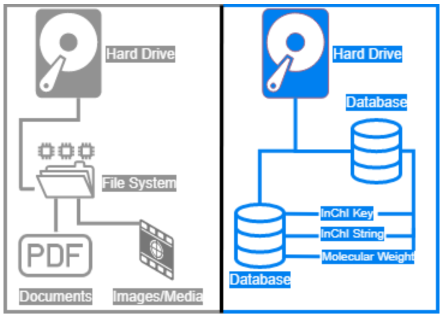


Figure 3.1.4-1 Shown above are two systems for using storage within an application. On the left is a file system which illustrates keeping documents and media stored as a file within a hard drive. This is much more feasible then storing the binary of a file inside of a database which overloads and causes database bloating. The right side shows a database which is also stored on a hard drive, however ,it is an application within itself that allows the structuring and organization of smaller pieces of information such as simple text strings. This allows for quick and easy access to data retrieval and storage without having to open and parse large documents.

Prepare for Expansion

Working with chemical data can present many problems for managing a database. The first and most obvious challenge is to recognize that the amounts of data will continue to expand when new contributions are made. Databases that are poorly designed can be challenging to expand if the available storage limits have been reached. On the other hand, a properly designed database can easily be expanded by just adding more memory or resources. It is the design of the original database that will be important in making sure that the expansions integrate properly.

If the database will be required to house a lot of similar data entries with fewer categories or indexes needed, then a single large capacity database may work fine. Another alternative would be to add new tables with the same structure every time the index number reaches a certain range. This would allow for allocating multiple database engines for faster data retrieval. The actual design of such a system is way beyond the scope of this course, however its important to recognize the process as large online chemical databases may be using similar technology.

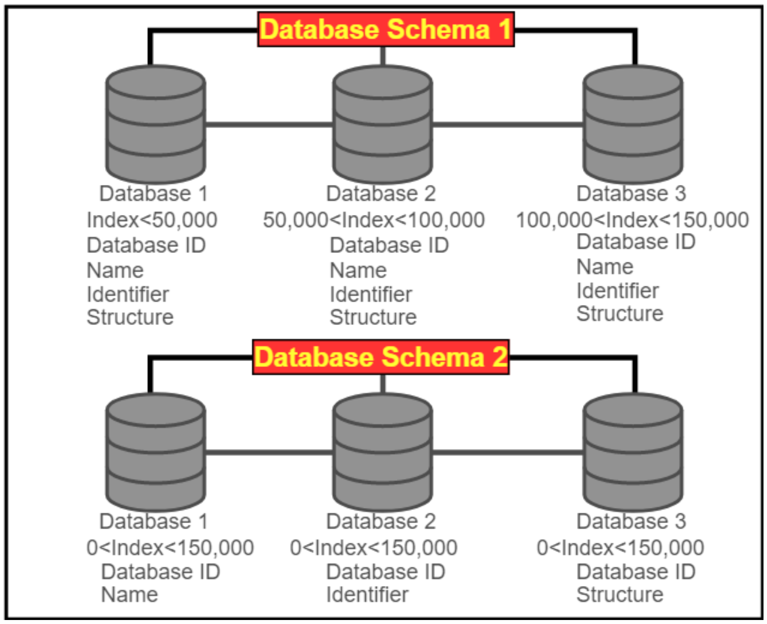


Figure 2 Shown above are two different database schemas for working with large amounts of data. Although both schemas represent only 4 fields for storing data, think of this as a small sample for data that could have hundreds of fields for each chemical entry. Database schema shows that the entries could be broken up simply by having a separate database for each additional 50,000 entries and keeping each chemical with all field entries kept together. Database schema 2 keeps all of the different fields together in the same database, but breaks up separate fields into different databases linked by the chemical database ID for proper retrieval. Both schemas work well for handling large amounts of data, however database schema 1 may be easier to set up for administrators without lots of experience in database design.

Designing Relationships

Challenges to building a database often arise when looking to set up relationships to show how information can be linked. With chemical data, there are endless possibilities on what can be linked and related within a data set. Among the most common relationships would be a chemical needing to be linked with its identifiers and properties such as a melting point or molecular weight. The relationship can be applied directly within a single database table in which the name and other values are only separated by different fields under the same index number. This would be the simplest and most direct way to design a relationship within a database. A problem might arise when the number of fields within this database table grows to a point that it causes data retrieval to slow down.

Solving the problem of building a database with lots of fields, the designer may link different types of tables or possibly using separate databases completely. The benefit to this process is that you can have a field dedicated to linking an index value between separate tables while allowing the resources that power the database to have smaller individual jobs.

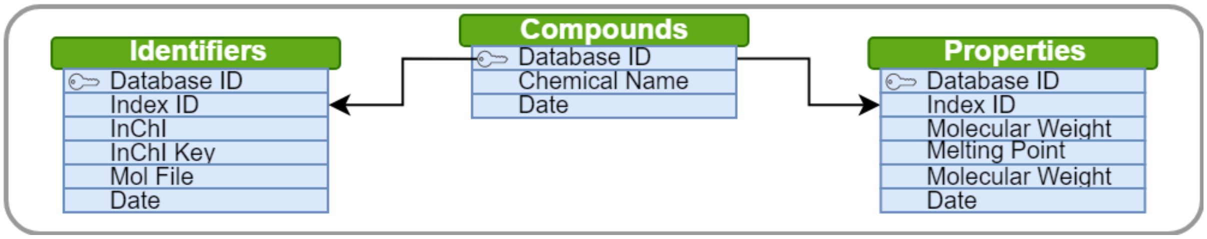


Figure 3.1.4-1 Shown above is an example database relationship. Using the above configuration multiple values for both properties and identifiers could be linked to a table of different chemical names. This might be useful in a situation where only a single InChI matches a particular compound, however that compound may have several different chemical names. This can also be useful in recording the same properties of a compound at different temperatures, pressures, etc.

Creating a Database

There are many options and configurations that can be used when creating a database. To review a few basic types of databases, look at section 1.3 of this text. To look up installing and working with a specific database then it's best to start with the official documentation provided by the creators of the software.

For the purpose of this lesson we will cover the basic concepts of creating the database layout and what to include. Start with a template that you either draw out on paper or electronically that includes some basic fields. The first item to include will be the unique index for all entries. This is most often a number that will automatically increment as data is loaded. It is important that each number be unique as this will be your primary key for the database. When using a spreadsheet or text file for your database, this primary key will usually be your first column and it is what the database will associate all fields with identification of the data. Next, you should define all fields that will be needed to house all of the collected data. These can be text fields, URL placeholders and number fields that will be used to store the related data to your chemical entries.

The last things that should be created are the relationships and indexes which will be used for searching and retrieving the stored data. A relationship should be created to link similar data values and compounds. An example of a popular relationship that is used could be set up through a field that defines types of chemicals. Under this relationship, the database could link all chemicals that fall under a ketone, alcohol, aldehyde, etc. This allows for a database to be queried against all chemicals that share certain functional groups.

Adding Data

Once the layout has been setup for a database, it is important to set up ways to put data into the tables. It is a little beyond the scope of this course, but setting up a web interface is a very common way to add entries into a database. There are many programming languages that can perform this operation such as PHP (Hypertext Preprocessor) and Python. Another common way to add items into a database is through an API (Application Programming Interface). There are many tutorials and videos online for performing each step to connect with a database.

When adding data to a database, it is important to remember that the unique database ID should be created at the same time as the values that are put into their relevant fields. You will want to make sure that the ID does not overwrite any previous entries and that it is set to auto-increment.

Editing Data

Sometimes a database will need to have edits made to previously entered values. A good example would be for a field that stores the location of an image, such as one made to represent a 3D view of a compound. If the user would like to change the image to one of higher quality, then the new uploaded image would have to overwrite the old location to reflect the new image. The commands used to do this should also include deleting the actual image from storage so that space is not used to keep unused files.

Data Removal

It may be bad practice to remove any chemical data, however mistakes do happen and sometimes it may be useful to remove data. This could be for several reasons, but an example could be that the original data may have contained errors and should be removed until a new collection is made. Keep in mind that when removing database entries, a script should also be included that will delete any associated files to that entry. A common practice for deleting database entries is that all deleted data is sent to a temporary storage folder for a short amount of time in case the data needs to be restored. Its always a good idea to keep regular backups of a database should something happen that leaves users unable to access data.

3.3: Public Chemical Databases

PubChem: chemical information repository at the U.S. NIH

PubChem (https://pubchem.ncbi.nlm.nih.gov)[1,2,3] is a public repository of information on small molecules and their biological activities, developed and maintained by the National Library of Medicine (NLM), an institute within the U.S. National Institutes of Health (NIH). Since its launch in 2004 as a component of the NIH's Molecular Libraries Roadmap Initiatives, it has been rapidly growing, and now serves as a key chemical information resource for researchers in many biomedical science areas, including cheminformatics, chemical biology, and medicinal chemistry. Detailed information on PubChem can be found in these three papers:

- PubChem Substance and Compound databases

  S. Kim *et al.*, Nucleic Acids Research **2016**, *44*, D1202-D1213

  (https://doi.org/10.1093/nar/gkv951)
- PubChem BioAssay: 2017 update

  Wang Y. *et al.* Nucleic Acids Research **2017**, *45*, D955-D963

  (https://doi.org/10.1093/nar/gkw1118)
- Getting the most out of PubChem for virtual screening

  S. Kim, Expert Opin. Drug Discov. **2016**, *11*, 843-855

  (http://dx.doi.org/10.1080/17460441.2016.1216967)

As of February 2017, PubChem contains more than 235 million depositor-provided substances, 94 million unique chemical structures, and one million biological assays, which cover about 10 thousand protein target sequences. For efficient use of this vast amount of data, PubChem provides various search and analysis tools. Some of these search tools will be used later in this course for demonstration purposes.

ChemSpider: a chemical database integrated with RSC's publishing process

ChemSpider (http://www.chemspider.com/)[4,5] is a free chemical structure database, containing information on 34 million structures collected from ~500 data sources. It also provides information on chemical reactions through ChemSpider SyntheticPages (CSSP)[6]. ChemSpider uses a crowdsourcing approach that allows registered users for manual comment and correction of ChemSpider records. Owned by the Royal Society of Chemistry (RSC), which publishes ~40 peer-reviewed chemistry journals, ChemSpider is integrated with the RSC publishing process, whereby new chemicals identified in newly published RSC articles also become available in ChemSpider.

ChEMBL: literature-extracted biological activity information

ChEMBL (https://www.ebi.ac.uk/chembl/)[7,8] is a large bioactivity database, developed and maintained by the European Bioinformatics Institute (EBI), which is part of the European Molecular Biology Laboratory (EMBL). The core activity data in ChEMBL are "manually" extracted from the full text of peer-reviewed scientific publications in select chemistry journals, such as *Journal of Medicinal Chemistry*, *Bioorganic Medicinal Chemistry Letters*, and *Journal of Natural products*. From each publication, details of the compounds tested, the assays performed and any target information for these assays are abstracted. ChEMBL also integrates screening results and bioactivity data from other public databases (such as PubChem BioAssay) and information on approved drugs from the U.S. FDA Orange Book[9] and the NLM's DailyMed[10].

ChEBI: a dictionary of small molecular entity

ChEBI (https://www.ebi.ac.uk/chebi/)[11,12] stands for "Chemical Entities of Biological Interest". It is a freely available database of "small" molecular entities, developed at the European Bioinformatics Institute (EBI). The molecular entities in ChEBI are either natural or synthetic products used to intervene the processes of living organisms. As a rule, however, ChEBI does not contain macromolecules directly encoded by genome (e.g., nucleic acids, proteins, and peptides derived from protein by cleavage). ChEBI provides "standardized" descriptions of molecular entities that enable other databases to annotate their entries in a consistent fashion. ChEBI focuses on high-quality manual annotation, non-redundancy, and provision of a chemical ontology rather than full coverage of the vast chemical space. Note that both ChEMBL and ChEBI are developed and maintained by the EMBL-EBI. While ChEMBL focuses on “bioactivity” of a large number of bioactive molecules (currently ~2.0 millions), ChEBI is a “dictionary” that provides high-quality standardized descriptions for a relatively small number of molecules (currently ~50 thousands).

NIST Webbook: thermodynamic and spectroscopic data of chemicals

The U.S. National Institutes of Standards and Technology (NIST) compiles chemical and physical property data for chemical species and distributes them through the web site called the NIST Chemistry WebBook (http://webbook.nist.gov)[13,14]. These data include thermochemical data (e.g., enthalpy of formation, heat capacity, and vapor pressure), reaction thermochemistry data (e.g., enthalpy of reaction and free energy of reaction), spectroscopic data (e.g., IR and UV/Vis spectra), gas chromatographic data, ion energetics data, and so on.

DrugBank: comprehensive information on drug molecules

DrugBank[15,16,17] (http://www.drugbank.ca/) is a comprehensive online database containing biochemical and pharmacological information about ~8,000 drug molecules, including U.S. Food and Drug Administration (FDA)-approved small-molecule drugs and biotech drugs (e.g., protein/peptide drugs) as well as experimental drugs. DrugBank provides a wide range of drug information, including drug targets, mechanism of action, adverse drug reactions, food-drug and drug-drug interactions, experimental and theoretical ADMET properties (*i.e.*, Absorption, Distribution, Metabolism, Excretion, and Toxicity), and many others. Most of these data are curated from primary literature sources, by domain-specific experts and skilled biocurators.

HMDB: the Human Metabolome Database

The Human Metabolome Database (HMDB) (http://www.hmdb.ca)[18,19,20] is comprehensive information on human metabolites and human metabolism data. This database contains curated information derived from scientific literature, as well as experimentally determined metabolite concentrations in human tissue or biofluid (e.g., urine, blood, cerebrospinal fluid and so on). Reference Mass spectra (MS) and nuclear magnetic resonance (NMR) spectra for metabolites are also provided when available. In addition to data for “detected” metabolites (those with measured concentrations or experimental confirmation of their existence), the HMDB also provides information on “expected” metabolites (those for which biochemical pathways are known or human intake/exposure is frequent but the compound has yet to be detected in the body).

TOXNET: a collection of toxicological information

TOXNET (http://toxnet.nlm.nih.gov/)[21,22,23,24] maintained by the National Library of Medicine (NLM) at NIH, is a group of databases covering toxicology, hazardous chemicals, toxic releases, environmental and occupational health, risk assessment. Currently, 16 databases are integrated into the TOXNET system, and users can search all these databases either at once or individually. While all the 16 databases provide valuable information, three of them may be worth mentioning in the context of this course.

- ChemIDPlus[25,26] is a dictionary of over 400,000 chemical records (names, synonyms, and structures) and provides access to the structure and nomenclature files used for the identification of chemical substances in the TOXNET system and other NLM databases.
- The Hazardous Substances Data Bank (HSDB)[27,28] focuses on the toxicology of potentially hazardous chemicals, providing information on human exposure, industrial hygiene, emergency handling procedures, environmental fate, regulatory requirements, nanomaterials, and related areas. All HSDB data are referenced and derived from a core set of books, government documents, technical reports and selected primary journal literature. Importantly, HSDB is peer-reviewed by the Scientific Review Panel (SRP), a committee of experts in the major subject areas within the data bank's scope.
- The Comparative Toxicogenomics Database (CTD)[29,30] contains manually curated data describing interactions of chemicals with genes/proteins and diseases. This database provides insight into the molecular mechanisms underlying variable susceptibility for environmentally influenced diseases.

A brief overview of TOXNET and its databases can be found in the TOXNET Fact Sheet[22] and a recent paper by Fowler and Schnall[24].

Protein Data Bank (PDB): a key source for protein-bound ligand structures

The Protein Data Bank (PDB) is an archive of the experimentally determined 3-D structures of large biological molecules such as proteins and nucleic acids. These structures were determined primarily by using X-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy. While PDB is not a small molecule database, it contains the 3-D structures of many proteins with small-molecule ligands bound to them. PDB allows users to search for proteins that an input small molecule binds to. Considering that it is not possible to experimentally determine how small molecules (such as drug or toxic chemicals) actually bind to their target proteins in a living organism, PDB is the most widely used resource for experimentally determined protein-bound structures of small molecules. The PDB are maintained by the Worldwide PDB (wwPDB)[31], and freely accessible via the websites of its member organizations: PDBe (PDB in Europe)[32,33], PDBj (PDB Japan)[34,35], RCSB PDB (Research Collaboratory for Structural Bioinformatics PDB)[36,37].

References

1. Kim, S.; Thiessen, P. A.; Bolton, E. E.; Chen, J.; Fu, G.; Gindulyte, A.; Han, L. Y.; He, J. E.; He, S. Q.; Shoemaker, B. A.; Wang, J. Y.; Yu, B.; Zhang, J.; Bryant, S. H. *Nucleic Acids Res.* **2016**, *44*, D1202.
2. Wang, Y.; Bryant, S. H.; Cheng, T.; Wang, J.; Gindulyte, A.; Shoemaker, B. A.; Thiessen, P. A.; He, S.; Zhang, J. *Nucleic Acids Res.* **2017**, *45*, D955.
3. Kim, S. *Expert Opinion on Drug Discovery* **2016**, *11*, 843.
4. ChemSpider (http://www.chemspider.com) (Accessed on 2/17/2017).
5. Pence, H. E.; Williams, A. *J. Chem. Educ.* **2010**, *87*, 1123.
6. ChemSpider SyntheticPages (CSSP) (http://cssp.chemspider.com/) (Accessed on 2/17/2017).
7. ChEMBL (https://www.ebi.ac.uk/chembl/) (Accessed on 2/17/2017).
8. Gaulton, A.; Hersey, A.; Nowotka, M.; Bento, A. P.; Chambers, J.; Mendez, D.; Mutowo, P.; Atkinson, F.; Bellis, L. J.; Cibrián-Uhalte, E.; Davies, M.; Dedman, N.; Karlsson, A.; Magariños, M. P.; Overington, J. P.; Papadatos, G.; Smit, I.; Leach, A. R. *Nucleic Acids Res.* **2017**, *45*, D945.
9. Orange Book: Approved Drug Products with Therapeutic Equivalence Evaluations (http://www.accessdata.fda.gov/scripts/cder/ob/default.cfm) (Accessed on 2/17/2017).
10. DailyMed (http://dailymed.nlm.nih.gov/) (Accessed on 2/17/2017).
11. ChEBI (https://www.ebi.ac.uk/chebi/) (Accessed on 2/17/2017).
12. Hastings, J.; de Matos, P.; Dekker, A.; Ennis, M.; Harsha, B.; Kale, N.; Muthukrishnan, V.; Owen, G.; Turner, S.; Williams, M.; Steinbeck, C. *Nucleic Acids Res.* **2013**, *41*, D456.
13. NIST Chemistry Webbook (http://webbook.nist.gov/chemistry/) (Accessed on 2/19/2017).
14. Linstrom, P. J.; Mallard, W. G. *J. Chem. Eng. Data* **2001**, *46*, 1059.
15. DrugBank (http://www.drugbank.ca/) (Accessed on 2/19/2017).
16. About DrugBank (http://www.drugbank.ca/about) (Accessed on 2/19/2017).
17. Law, V.; Knox, C.; Djoumbou, Y.; Jewison, T.; Guo, A. C.; Liu, Y. F.; Maciejewski, A.; Arndt, D.; Wilson, M.; Neveu, V.; Tang, A.; Gabriel, G.; Ly, C.; Adamjee, S.; Dame, Z. T.; Han, B. S.; Zhou, Y.; Wishart, D. S. *Nucleic Acids Res.* **2014**, *42*, D1091.
18. The Human Metabolome Database (HMDB) (http://www.hmdb.ca/) (Accessed on 2/19/2017).
19. About the Human Metabolome Database (HMDB) (http://www.hmdb.ca/about) (Accessed on 2/19/2017).
20. Wishart, D. S.; Jewison, T.; Guo, A. C.; Wilson, M.; Knox, C.; Liu, Y. F.; Djoumbou, Y.; Mandal, R.; Aziat, F.; Dong, E.; Bouatra, S.; Sinelnikov, I.; Arndt, D.; Xia, J. G.; Liu, P.; Yallou, F.; Bjorndahl, T.; Perez-Pineiro, R.; Eisner, R.; Allen, F.; Neveu, V.; Greiner, R.; Scalbert, A. *Nucleic Acids Res.* **2013**, *41*, D801.
21. ToxNet (http://toxnet.nlm.nih.gov/) (Accessed on 2/19/2017).
22. Factsheet - Toxicology Data Network (TOXNET) (http://www.nlm.nih.gov/pubs/factsheets/toxnetfs.html) (Accessed on 2/19/2017).
23. Wexler, P. *Toxicology* **2001**, *157*, 3.
24. Fowler, S.; Schnall, J. G. *Am. J. Nurs.* **2014**, *114*, 61.
25. ChemIDplus (http://chem.sis.nlm.nih.gov/chemidplus/chemidlite.jsp) (Accessed on 2/19/2017).
26. Fact Sheet - ChemIDplus (http://www.nlm.nih.gov/pubs/factsheets/chemidplusfs.html) (Accessed on 2/19/2017).
27. Hazardous Substances Data Bank (HSDB) (http://toxnet.nlm.nih.gov/newtoxnet/hsdb.htm) (Accessed on 2/19/2017).
28. Fact Sheet - Hazardous Substances Data Bank (HSDB) (http://www.nlm.nih.gov/pubs/factsheets/hsdbfs.html) (Accessed on 2/19/2017).
29. Comparative Toxicogenomics Database (CTD) (http://toxnet.nlm.nih.gov/newtoxnet/ctd.htm) (Accessed on 2/19/2017).
30. Fact Sheet - Comparative Toxicogenomics Database (CTD) (http://www.nlm.nih.gov/pubs/factsheets/ctdfs.html) (Accessed on 2/19/2017).
31. Worldwide Protein Data Bank (wwPDB) (http://www.wwpdb.org/) (Accessed on 2/19/2017).

32. Protein Data Bank in Europe (PDBe) (http://www.ebi.ac.uk/pdbe/) (Accessed on 2/19/2017).
33. Gutmanas, A.; Alhroub, Y.; Battle, G. M.; Berrisford, J. M.; Bochet, E.; Conroy, M. J.; Dana, J. M.; Montecelo, M. A. F.; van Ginkel, G.; Gore, S. P.; Haslam, P.; Hendrickx, P. M. S.; Hirshberg, M.; Lagerstedt, I.; Mir, S.; Mukhopadhyay, A.; Oldfield, T. J.; Patwardhan, A.; Rinaldi, L.; Sahni, G.; Sanz-Garcia, E.; Sen, S.; Slowley, R. A.; Velankar, S.; Wainwright, M. E.; Kleywegt, G. J. *Nucleic Acids Res.* **2014**, *42*, D285.
34. Protein Data Bank Japan (PDBj) (http://pdbj.org/) (Accessed on 2/19/2017).
35. Kinjo, A. R.; Suzuki, H.; Yamashita, R.; Ikegawa, Y.; Kudou, T.; Igarashi, R.; Kengaku, Y.; Cho, H.; Standley, D. M.; Nakagawa, A.; Nakamura, H. *Nucleic Acids Res.* **2012**, *40*, D453.
36. RCSB Protein Data Bank (RCSB PDB) (http://www.rcsb.org/pdb/) (Accessed on 2/19/2017).
37. Rose, P. W.; Prlic, A.; Bi, C. X.; Bluhm, W. F.; Christie, C. H.; Dutta, S.; Green, R. K.; Goodsell, D. S.; Westbrook, J. D.; Woo, J.; Young, J.; Zardecki, C.; Berman, H. M.; Bourne, P. E.; Burley, S. K. *Nucleic Acids Res.***2015**, *43*, D345.

3.4: Data Organization in PubChem as a Data Aggregator

PubChem Aggregator Overview

PubChem (https://pubchem.ncbi.nlm.nih.gov) is a data aggregator, meaning that it collects data from other data sources. As of February 2017, PubChem's data are from more than 500 organizations, including government agencies, university labs, pharmaceutical companies, substance vendors, and other databases. An up-to-date list of PubChem's data sources is available at the PubChem Sources page (https://pubchem.ncbi.nlm.nih.gov/sources). To better understand the features of this page, read this article on PubChem Blog:

Data Sources Page

(http://go.usa.gov/xk7xU)

PubChem organizes its data into three inter-linked databases: Substance, Compound, and BioAssay

(See **Table 1**), which can be searched from either the PubChem home page (https://pubchem.ncbi.nlm.nih.gov) or the web page of one of the three PubChem databases.

Table 1. Three inter-linked databases in PubChem.

Database	URL	Identifier
Substance	https://www.ncbi.nlm.nih.gov/pcsubstance	SID
Compound	https://www.ncbi.nlm.nih.gov/pccompound	CID
BioAssay	https://www.ncbi.nlm.nih.gov/pcassay	AID

Individual data contributors deposit information on chemical substances to the Substance database (https://www.ncbi.nlm.nih.gov/pcsubstance). Different data contributors may provide information on the same molecule, hence the same chemical structure may appear multiple times in the Substance database. To provide a non-redundant view, chemical structures in the Substance database are normalized through a process called "standardization" and the unique chemical structures are identified and stored in the Compound database (https://www.ncbi.nlm.nih.gov/pccompound). The difference between the Substance and Compound databases is explained in more detail in this blog post.

Compounds and Substances

What is the difference between a substance and a compound in PubChem?
(http://1.usa.gov/1nl9ePL)

Descriptions of biological experiments on chemical substances are stored in the BioAssay database (https://www.ncbi.nlm.nih.gov/pcassay). The unique identifiers used to locate records in these three databases are called SID (Substance ID), CID (Compound ID), and AID (Assay ID) for the Substance, Compound, and BioAssay databases, respectively.

All information in the Substance database is submitted by individual data depositors. However, the Compound database does contain information that are not submitted by data depositors, but annotated by the PubChem team. [In the context of scientific databases, annotation refers to the process of adding extra information to a database entry (for example, a compound in the Compound database and an assay in the BioAssay database)]. The annotated information is always presented with its provenance information (that is, the source of the information). The list of all the annotation sources used in PubChem is available at the PubChem Sources page (https://pubchem.ncbi.nlm.nih.gov/sources). From this page, one may download all the annotations from a particular source.

3.5: Database Query Introduction

Basic Searches

After visiting a website that provides chemical information such as PubChem, there was most likely a text field available that allows the user to input an alphanumeric name, number or combination of both to retrieve a chemical. Simple text input searches may seem like the most basic way to look for something in a database and often they are, but many allow for inputing characters to refine how the search is performed. Using a simple google search as an example, you can change how the search is performed by simply putting your terms into parenthesis. This tells the search engine that everything in parenthesis must be found before being included in the results. There are some common practices for these alterations that are used among many different search engines, but this should not be interpreted to think that they will all work the same.

Before performing a series of searches, look for documentation provided by the host of the search engine to see what alterations can be made so that the results found have more relevancy to what the user seeks. The reason for this is simply due to the fact that chemical databases can contain a lot of data and this will save on trying to sort through thousands of results. Review section 1.3 of this course to see some other alterations that are often allowed in text searches. Take note of using boolean characters along with some of the different ways chemicals can be represented in chapter 2 of this course.

Custom Search Parameters

Advanced chemical searches may need to have several custom parameters used to narrow down a large list of chemicals into a smaller more relevant pool of results. The user interface for a custom search usually works similar to filling out an electronic form. After navigating to the advanced search on a website, the user will be presented with many options for defining a very specific search. Many of these advanced searches will contain checkboxes for selecting things like functional groups, experimental properties and other details for the compound. The user may also be able to specify ranges for things like density, number of atoms or other properties. Keywords can be valuable in searching the context of a chemical page to find a structure meeting certain categories.

3.6: Special Notes on Using Public Chemical Databases

Availability and Data Exchange

All the databases mentioned in section 3.4 and 3.5 (including PubChem) are public databases that provide their contents free of charge, and in many cases they also provide a way to download data in bulk and integrate them into one's own database. Therefore, it is very common that database groups exchange their information with each other. This often raises some technical concerns. For example, different databases may use different chemical representations to refer to the same molecule. This may result in incorrect chemical structure matching between the databases, leading to incorrect data integration. In addition, when one database has incorrect information, this error often propagates into other databases. The error propagation issue is a serious, but very common, problem.[1,2] Therefore, when using information in these databases, one should keep in mind various data accuracy and quality issues prevalent in these databases. A goal of this course is to help students develop the ability to critically assess chemical information available in public databases.

References

1. Schnoes, A. M.; Brown, S. D.; Dodevski, I.; Babbitt, P. C. *PLoS Comput. Biol.* **2009**, *5*, e1000605.
2. Philippi, S.; Kohler, J. *Nat. Rev. Genet.* **2006**, *7*, 482.

3.7: Mathematica Assignment

3.8: Python Assignment

Compound vs Substance

Downloadable Files

lecture04_Standardization

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

- Requests
- RDKit
- time
- PIL (?)
- IPython.Display

Objectives

- Understand the difference between compounds and substances in PubChem's terminology.
- Learn how chemical structures are represented in a real world.
- Understand the disambiguity of name-structure associations.
- Learn how to draw chemical structures programmatically.

Note

To use the python code in this lesson plan, RDKit must be installed on the system.

Many users can simply run the following code in the command prompt to install RDKit. (be sure you are in the OLCC environment of conda)

conda install -c rdkit rdkit

Access to the full installation instructions can be found at the following link. https://www.rdkit.org/docs/Install.htm

Structure Standardization

PubChem contains more than 200 millions chemical records submitted by hundreds of data contributors. These depositor-provided records are archived in a database called "**Substance**" and each record in this database is called a **substance**. The records in the Substance database are highly redundant, because different data contributors may submit information on the same chemical, independently of each other. Therefore, PubChem extracts unique chemical structures from the Substance database through a process called standardization (https://doi.org/10.1186/s13321-018-0293-8). These unique structures are stored in the **Compound** database and individaual records in this database is called "**compounds**". To learn more about the PubChem compounds and substances, please read this PubChem Blog post (https://go.usa.gov/xVXct).

The code cells below demonstrates the effects of chemical structure standardization.

Step 1. Download a list of the SIDs associated with a given CID

First, let's get a list of SIDs that are associated CID 1174 (uracil).

In [1]:

```python
import requests

cid = 1174

url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/" + str(cid) + "/sids/txt"
res = requests.get(url)
sids = res.text.split()
print(len(sids))
```

```
367
```

The above request returns 360+ substances, all of which are standardized to the same structure (CID 1174).

Step 2. Download the structure data for the SIDs

Now retrieve the depositor-provided structures for the returned substances.

In [2]:

```python
import time

chunk_size = 50

if len(sids) % chunk_size == 0 :
    num_chunks = int( len(sids) / chunk_size )
else :
    num_chunks = int( len(sids) / chunk_size ) + 1

f = open("cid2sids-uracil.sdf", "w")

for i in range(num_chunks):

    print("Processing chunk", i)

    idx1 = chunk_size * i
    idx2 = chunk_size * (i + 1)
    str_sids = ",".join(sids[idx1:idx2])

    url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/sid/" + str_sids + "/record/sdf"
    res = requests.get(url)

    f.write(res.text)
    time.sleep(0.2)

f.close()
```

```
Processing chunk 0
Processing chunk 1
        .
        .
        .
Processing chunk 7
```

Step 3. Convert the structures in the SDF file into the SMILES strings and identify unique SMILES and their frequencies.

In [3]:

```
from rdkit import Chem

unique_smiles_freq = dict()

suppl = Chem.SDMolSupplier('cid2sids-uracil.sdf')

for mol in suppl:

    smiles = Chem.MolToSmiles(mol,isomericSmiles=True)

    unique_smiles_freq[ smiles ] = unique_smiles_freq.get(smiles,0) + 1

sorted_by_freq = [ (v, k) for k, v in unique_smiles_freq.items() ]
sorted_by_freq.sort(reverse=True)
for v, k in sorted_by_freq :
    print(v, k)
```

```
247 O=c1cc[nH]c(=O)[nH]1
87 Oc1ccnc(O)n1
12
7 O=c1ccnc(O)[nH]1
5 O=c1nccc(O)[nH]1
5 O=c1nc(O)cc[nH]1
4 O=c1cc[nH]c(O)n1
```

The above output shows that the 360+ SIDs associated with CID 1174 are represented with six different SMILES strings. In addition, 12 substance records that resulted in an "empty" SMILES strings, implying that the depositors of these substance records did not provide structral information. You may want to what these 12 substances are, but the above code cell does not tell you what they are. This can be done using the following code cell.

In [4]:

```
for mol in suppl:

    smiles = Chem.MolToSmiles(mol,isomericSmiles=True)

    if ( smiles == "" ) :
        print(mol.GetProp('PUBCHEM_SUBSTANCE_ID'), ":", mol.GetProp('PUBCHEM_SUBS_AUTO_STRUCTURE'))
```

```
50608295 : Deposited Substance chemical structure was generated via Synonym "CID1174"
76715622 : Deposited Substance chemical structure was generated via Synonym(s) "uraci
            .
            .
            .
384995482 : Deposited Substance chemical structure was generated via Synonym(s) "66-2
```

Sometimes a data depositor does not provide the structure of a chemical but its chemical synonym(s). In that case, PubChem uses the chemical synonyms to assign a structure to this structure-less record. For example, SID 50608295 (one of the 12 structures without SMILES strings in the above output) did not have a depositor-provided structure, but its depositor-provided synonyms include "CID1174". Therefore, PubChem assigns SID 50608295 to CID 1174, although the depositor did not provide the structure of SID 50608295. (Please check the structure and synonyms for SID 50608295 stored in the SDF file ("cid2sids-uracil.sdf") generated in step 2).

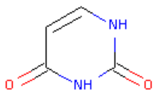
Step 4. Generate the structure images from the SMILES

Now we want to see what these SMILES strings look like, by drawing molecular structures from them.

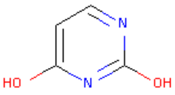
In [5]:

```
from rdkit.Chem import Draw

for mysmiles in unique_smiles_freq.keys() :

    if mysmiles != "" :

        print(mysmiles)
        img = Draw.MolToImage( Chem.MolFromSmiles(mysmiles), size=(150, 150) )
        display(img)
```

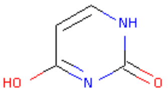
```
O=c1cc[nH]c(=O)[nH]1
```



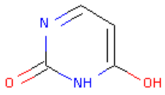
```
Oc1ccnc(O)n1
```



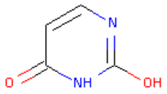
```
O=c1nc(O)cc[nH]1
```



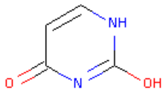
```
O=c1nccc(O)[nH]1
```



```
O=c1ccnc(O)[nH]1
```



```
O=c1cc[nH]c(O)n1
```



You may want to write these molecule images in files, rather than displaying them on this Jupyter notebook.

In [6]:

```python
from rdkit.Chem import Draw

index = 1

for mysmiles in unique_smiles_freq.keys() :

    if mysmiles != "" :

        filename = 'image' + str(index) +'.png'
        Draw.MolToFile( Chem.MolFromSmiles(mysmiles), filename )
        index += 1
```

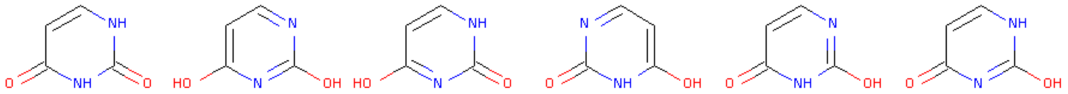
You may also want to display all the images in a single figure.

In [7]:

```python
from PIL import Image
```

In [8]:

```python
images = []

for mysmiles in unique_smiles_freq.keys() :

    if mysmiles != "" :

        img = Draw.MolToImage( Chem.MolFromSmiles(mysmiles), size=(150, 150) )
        images.append(img)

big_img = Image.new('RGB', (900,150))  # enought to arrange six 150x150 images

for i in range(0,len(images)):

    #paste the image at location i,j:
    big_img.paste(images[i], (i*150, 0 ) )

display(big_img)
```



In [9]:

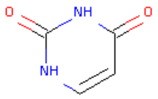
```
big_img.save('image_grid.png')
```

As shown these chemical images, the 360+ substances associated with CID 1174 (uracil) correspond to six tautomeric form of uracil, which differ from each other in the position of "movable" hydrogen atoms. Compare these structures with their standardized structure (CID 1174).

In [10]:

```
res =
requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/1174/proper
img = Draw.MolToImage( Chem.MolFromSmiles( res.text.rstrip() ), size=(150, 150)
img
```

Out[10]:

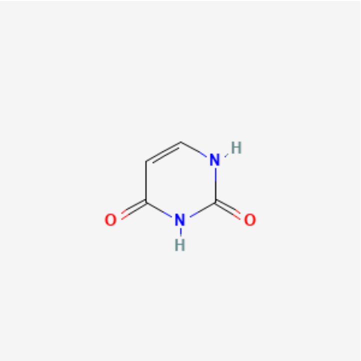


Alternatively, you can get the structure image of CID 1174 from PubChem.

In [11]:

```
from IPython.display import Image
Image(url='https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/1174/record/PN
image_size=300x300')
```

Out[11]:



Exercise 1a: The **MolToSmiles()** function used in **Step 3** generates the canonical SMILES string by default. Read the RDKit manual about the arguments available for this function (https://www.rdkit.org/docs/source/rdkit.Chem.rdmolfiles.html) and write a code that generates non-canonical SMILES strings for the 360+ substance records associated with uracil (CID 1174).

- Ignore/skip structure-less records using a conditional statement (i.e., an if statement).

- Print the number of unique non-canonical SMILES.
- Print unique non-canonical SMILES, sorted by frequency.
- For a given molecule, there may be multiple ways to write SMILES strings: one of them is selected as the "canonical" SMILES and all the others are considered as "non-canonical". However, for the purpose of this exercise, we want to generate only one non-canonical SMILES for each record (because the function will return only one SMILES string (the canonical SMILES or one of possible non-canonical SMILES)).

In [12]:

```
# Write your code in this cell.
```

Exercise 1b: The RDKit function "**MolsToGridImage()**" allows you to draw a "grid image" that shows multiple structures. Read the RDKit manual about "**MolsToGridImage()**" (https://www.rdkit.org/docs/source/rdkit.Chem.Draw.html) and display the structures represented by the unique non-canonical SMILES generated from **Exercise 1a**.

In [13]:

```
# Write your code in this cell.
```

Exercise 1c: Retrieve the substance records associated with guanine (CID 135398634) and display unique structures generated from them, by following these steps:

- Retrieve the SIDs associated CID 135398634
- Download the structure data for the retrieved SIDs (in SDF)
- Generate canonical SMILES strings from the structure data in the SDF file and identify unique canonical SMILES strings
- Draw the structures represented by the unqiue canonical SMILES strings in a single figure.

In [14]:

```
# Write your code in this cell.
```

Exercise 1d: Retrieve the substance records whose synonym is "glucose" and display unique structures generated from them, by following these steps:

- Retrieve the SIDs whose synonym is "glucose".
- Download the structure data for the retrieved SIDs (in SDF)
- Generate canonical SMILES strings from the structure data in the SDF file and identify unique canonical SMILES strings
- Draw the structures represented by the unqiue canonical SMILES strings in a single figure.

In [15]:

```
# Write your code in this cell.
```

Exercise 1e: Retrieve the compound records associated with the SIDs retrieved in **Exercise 1d** and display unique structures generated from them, by following these steps:

- Retrieve the CIDs associated with the SIDs whose name is "glucose", using a single PUG-REST request (i.e., using the list conversion covered in the previous notebook, "lecture03-list-conversion.ipynb").
- Identify unique CIDs from the returned CIDs, using the **set()** function in python.
- Retrieve the isomeric SMILES for the unique CIDs through PUG-REST.
- Draw the structures represented by the returned SMILES strings in a single figure.

In [16]:

```
# Write your code in this cell.
```

3.9: R Assignment

Standardization

S. Kim, J. Cuadros

November 2nd, 2019

Objectives

- Understand the difference between compounds and substances in PubChem's terminology.
- Learn how chemical structures are represented in a real world.
- Understand the disambiguity of name-structure associations.
- Learn how to draw chemical structures programmatically.

In this task, we will use some cheminformatics packages to ease some processes. In R, some options are rcdk , ChemmineR and ChemmineOB . In Python, a useful package is RDKit ; in R, we'll make use of it online version, the Beaker API of ChEMBL (https://chembl.gitbook.io/chembl-interface-documentation/web-services).

Structure Standardization

PubChem contains more than 200 millions chemical records submitted by hundreds of data contributors. These depositor-provided records are archived in a database called "Substance" and each record in this database is called a substance. The records in the Substance database are highly redundant, because different data contributors may submit information on the same chemical, independently of each other. Therefore, PubChem extracts unique chemical structures from the Substance database through a process called standardization (https://doi.org/10.1186/s13321-018-0293-8). These unique structures are stored in the Compound database and individaual records in this database is called "compounds". To learn more about the PubChem compounds and substances, please read this PubChem Blog post (https://go.usa.gov/xVXct (https://go.usa.gov/xVXct)). The code cells below demonstrates the effects of chemical structure standardization.

Step 1. Download a list of the SIDs associated with a given CID First, let's get a list of SIDs that are associated CID 1174 (uracil).

```
pugrest <- 'https://pubchem.ncbi.nlm.nih.gov/rest/pug'
pugin <- 'compound/cid/1174'
pugoper <- 'sids'
pugout <- 'txt'

url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
sids <- readLines(url)
length(sids)
```

The above request returns 300+ substances, all of which are standardized to the same structure (CID 1174).

Step 2. Download the structure data for the SIDs Now retrieve the depositor-provided structures for the returned substances.

```
chunk_size <- 50
num_chunks <- ceiling(length(sids)/chunk_size)

sdf <- character(0)
for(i in seq(num_chunks)) {

 print(paste("Processing chunk", i))

 idx1 <- chunk_size * (i - 1) + 1
 idx2 <- chunk_size * i
 str_sids <- paste(sids[idx1:min(idx2,length(sids))], collapse=",")

 url <- paste("https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/sid",
 str_sids, "record/sdf", sep="/")
 sdf <- c(sdf,readLines(url))
 Sys.sleep(0.2)
}
```

```
 writeLines(sdf,"uracil_from_sids.sdf")
```

Step 3. Convert the structures in the SDF file into the SMILES strings and identify unique SMILES and their frequencies.

```
if (!require("BiocManager", quietly=TRUE)) {
 install.packages("BiocManager", repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
library("BiocManager")
}
if (!require("ChemmineR", quietly=TRUE)) {
 BiocManager::install("ChemmineR",ask=FALSE)
 library("ChemmineR")
}
if (!require("ChemmineOB", quietly=TRUE)) {
 BiocManager::install("ChemmineOB",ask=FALSE)
 library("ChemmineOB")
}
```

```
 convertFormatFile("SDF","CAN","uracil_from_sids.sdf","uracil_from_sids.smi")

#convertFormatFile("SDF","CAN","https://chem.libretexts.org/@api/deki/files/346075/ur

smis <- read.table("uracil_from_sids.smi",header=F,sep="\t")
tabSMIs <- table(smis[,1])
tabSMIs <- sort(tabSMIs,decreasing = TRUE)
tabSMIs
```

The above output shows that the 300+ SIDs associated with CID 1174 are represented with six different SMILES strings. In addition, some substance records that resulted in an “empty” SMILES strings, implying that the depositors of these substance records did not provide structral information. You may want to what these substances are, but this can be recovered from the SDF file as follows.

```
uracilSDFset <- read.SDFset("uracil_from_sids.sdf")
```

```
 nonvalidSDF <- uracilSDFset[!validSDF(uracilSDFset)]
 unlist(lapply(nonvalidSDF@SDF,
     function(x) paste(x@datablock["PUBCHEM_SUBSTANCE_ID"],
     x@datablock["PUBCHEM_SUBS_AUTO_STRUCTURE"], sep=": ")))
```

Sometimes a data depositor does not provide the structure of a chemical but its chemical synonym(s). In that case, PubChem uses the chemical synonyms to assign a structure to this structure-less record. For example, SID 50608295 (one of the 12 structures without SMILES strings in the above output) did not have a depositorprovided structure, but its depositor-provided synonyms include “CID1174”. Therefore, PubChem assigns SID 50608295 to CID 1174, although the depositor did not provide the structure of SID 50608295. (Please check the structure and synonyms for SID 50608295 stored in the SDF file (“cid2sids-uracil.sdf”) generated in step 2).

Step 4. Generate the structure images from the SMILES

Now we want to see what these SMILES strings look like, by drawing molecular structures from them.

```
if(!require("httr", quietly=TRUE)) {
 install.packages("httr", repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("httr")
}
if(!require("png")) {
 install.packages(("png"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("png")
}
if(!require("grid")) {
 install.packages(("grid"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("grid")
}
if(!require("gridExtra")) {
 install.packages(("gridExtra"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("gridExtra")
}
```

```
 tabSMIs <- tabSMIs[names(tabSMIs)!=""]
names(tabSMIs)
```

```
 vecSMI <- names(tabSMIs)
names(vecSMI) <- names(tabSMIs)

sdf <- smiles2sdf(vecSMI)

# ChemmineR has a function to print the molecules, but
# results are far from acceptable
# png("plots.png",width=400,height=2000)
# ChemmineR::plot(sdf, griddim=c(ceiling(length((vecSMI))/2),2),
# regenCoords=TRUE,atomcex=1 )
```

```
#
# dev.off()
# A better option is to use the ChEMBL Beaker API (RDKit online)

write.SDF(sdf,"uracil_unique.sdf")
url <- "https://www.ebi.ac.uk/chembl/api/utils/ctab2image?size=300"
mydata <- list(sdf=upload_file("uracil_unique.sdf"))
res <- POST(url, body = mydata)
img <- readPNG(res$content, native=TRUE)
grid.arrange(rasterGrob(img))
writePNG(img,"uracil_unique.png")
```

You may want to write these molecule images in separate files.

```
dir.create("uracil_unique",showWarnings = FALSE)
fileSDF <- write.SDFsplit(sdf,"uracil_unique/",1)$filename
url <- "https://www.ebi.ac.uk/chembl/api/utils/ctab2image?size=300"

for(i in seq(fileSDF)) {
 mydata <- list(sdf=upload_file(fileSDF[i]))
 res <- POST(url, body = mydata)
 img <- readPNG(res$content, native=TRUE)
 writePNG(img,paste("uracil_unique/",
 formatC(i,format="d",width=2,flag="0"),".png",sep=""))
 Sys.sleep(1)
}

dir("uracil_unique",pattern=".*[.]png")
```

As shown these chemical images, the 300+ substances associated with CID 1174 (uracil) correspond to 6 tautomeric forms of uracil, which differ from each other in the position of “movable” hydrogen atoms. Compare these structures with their standardized structure (CID 1174).

```
img <- readPNG(GET('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/1174/PNG?i
grid.arrange(rasterGrob(img))
```

Exercise 1a: The function used in Step 3 generates the canonical SMILES string by default. OpenBabel supports a second format (“SMI”) that does not use a canonical numbering when creating the SMILES. Other options can be added. Write a code that generates non-canonical SMILES strings for the 300+ substance records associated with uracil (CID 1174).

- Ignore/skip structure-less records using a conditional statement (i.e., an if statement).
- Print the number of unique non-canonical SMILES.
- Print unique non-canonical SMILES, sorted by frequency.

For a given molecule, there may be multiple ways to write SMILES strings: one of them is selected as the “canonical” SMILES and all the others are considered as “non-canonical”. However, for the purpose of this exercise, we want to generate only one non-canonical SMILES for each record (because the function will return only one SMILES string, the canonical SMILES or one of possible non-canonical SMILES).

```
#Write your code here.
```

Exercise 1b: NOT INCLUDED IN THE R VERSION

Exercise 1c: Retrieve the substance records associated with guanine (CID 135398634) and display unique structures generated from them, by following these steps:

- Retrieve the SIDs associated CID 135398634
- Download the structure data for the retrieved SIDs (in SDF)
- Generate canonical SMILES strings from the structure data in the SDF file and identify unique canonical SMILES strings
- Draw the structures represented by the unqiue canonical SMILES strings in a single figure.

```
#Write your code here.
```

Exercise 1d: Retrieve the substance records whose synonym is “glucose” and display unique structures generated from them, by following these steps: Retrieve the SIDs whose synonym is “glucose”. Download the structure data for the retrieved SIDs (in SDF) Generate canonical SMILES strings from the structure data in the SDF file and identify unique canonical SMILES strings Draw the structures represented by the unique canonical SMILES strings in a single figure.

```
#Write your code here.
```

Exercise 1e: Retrieve the compound records associated with the SIDs retrieved in Exercise 1d and display unique structures generated from them, by following these steps: Retrieve the CIDs associated with the SIDs whose name is “glucose”, using a single PUG-REST request

(i.e., using the list conversion covered in the previous activity, “Interconversion between PubChem records”).

- Identify unique CIDs from the returned CIDs.
- Retrieve the isomeric SMILES for the unique CIDs through PUG-REST.
- Draw the structures represented by the returned SMILES strings in a single figure.

```
#Write your code here.
```

3.10: R Assignment (binder test)

Standardization

S. Kim, J. Cuadros

November 2nd, 2019

Objectives

- Understand the difference between compounds and substances in PubChem's terminology.
- Learn how chemical structures are represented in a real world.
- Understand the disambiguity of name-structure associations.
- Learn how to draw chemical structures programmatically.

In this task, we will use some cheminformatics packages to ease some processes. In R, some options are rcdk , ChemmineR and ChemmineOB . In Python, a useful package is RDKit ; in R, we'll make use of it online version, the Beaker API of ChEMBL (https://chembl.gitbook.io/chembl-interface-documentation/web-services).

Structure Standardization

PubChem contains more than 200 millions chemical records submitted by hundreds of data contributors. These depositor-provided records are archived in a database called "Substance" and each record in this database is called a substance. The records in the Substance database are highly redundant, because different data contributors may submit information on the same chemical, independently of each other. Therefore, PubChem extracts unique chemical structures from the Substance database through a process called standardization (https://doi.org/10.1186/s13321-018-0293-8). These unique structures are stored in the Compound database and individaual records in this database is called "compounds". To learn more about the PubChem compounds and substances, please read this PubChem Blog post (https://go.usa.gov/xVXct (https://go.usa.gov/xVXct)). The code cells below demonstrates the effects of chemical structure standardization.

Step 1. Download a list of the SIDs associated with a given CID First, let's get a list of SIDs that are associated CID 1174 (uracil).

```
pugrest <- 'https://pubchem.ncbi.nlm.nih.gov/rest/pug'
pugin <- 'compound/cid/1174'
pugoper <- 'sids'
pugout <- 'txt'

url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
sids <- readLines(url)
length(sids)
```

run | restart | restart & run all

The above request returns 300+ substances, all of which are standardized to the same structure (CID 1174).

Step 2. Download the structure data for the SIDs Now retrieve the depositor-provided structures for the returned substances.

```
chunk_size <- 50
num_chunks <- ceiling(length(sids)/chunk_size)

sdf <- character(0)
for(i in seq(num_chunks)) {

 print(paste("Processing chunk", i))

 idx1 <- chunk_size * (i - 1) + 1
 idx2 <- chunk_size * i
 str_sids <- paste(sids[idx1:min(idx2,length(sids))], collapse=",")
 url <- paste("https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/sid",
 str_sids, "record/sdf", sep="/")
 sdf <- c(sdf,readLines(url))
 Sys.sleep(0.2)
}
```

run restart restart & run all

```
writeLines(sdf,"uracil_from_sids.sdf")
```

run restart restart & run all

##### -- TEST -- #####

```
pugrest <- 'https://pubchem.ncbi.nlm.nih.gov/rest/pug'
pugin <- 'compound/cid/1174'
pugoper <- 'sids'
pugout <- 'txt'

url <- paste(pugrest,pugin,pugoper,pugout,sep="/")
sids <- readLines(url)
length(sids)

chunk_size <- 50
num_chunks <- ceiling(length(sids)/chunk_size)

sdf <- character(0)
for(i in seq(num_chunks)) {

 print(paste("Processing chunk", i))

 idx1 <- chunk_size * (i - 1) + 1
 idx2 <- chunk_size * i
 str_sids <- paste(sids[idx1:min(idx2,length(sids))], collapse=",")

 url <- paste("https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/sid",
 str_sids, "record/sdf", sep="/")
 sdf <- c(sdf,readLines(url))
 Sys.sleep(0.2)
}

writeLines(sdf,"uracil_from_sids.sdf")
```

run restart restart & run all

Step 3. Convert the structures in the SDF file into the SMILES strings and identify unique SMILES and their frequencies.

```
if (!require("BiocManager", quietly=TRUE)) {
 install.packages("BiocManager", repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
library("BiocManager")
}
if (!require("ChemmineR", quietly=TRUE)) {
 BiocManager::install("ChemmineR",ask=FALSE)
 library("ChemmineR")
}
if (!require("ChemmineOB", quietly=TRUE)) {
 BiocManager::install("ChemmineOB",ask=FALSE)
 library("ChemmineOB")
}
```

run restart restart & run all

```
convertFormatFile("SDF","CAN","uracil_from_sids.sdf","uracil_from_sids.smi")

#convertFormatFile("SDF","CAN","https://chem.libretexts.org/@api/deki/files/346075/ura

smis <- read.table("uracil_from_sids.smi",header=F,sep="\t")
tabSMIs <- table(smis[,1])
tabSMIs <- sort(tabSMIs,decreasing = TRUE)
tabSMIs
```

run restart restart & run all

The above output shows that the 300+ SIDs associated with CID 1174 are represented with six different SMILES strings. In addition, some substance records that resulted in an “empty” SMILES strings, implying that the depositors of these substance records did not provide structral information. You may want to what these substances are, but this can be recovered from the SDF file as follows.

```
uracilSDFset <- read.SDFset("uracil_from_sids.sdf")
```

run restart restart & run all

```
nonvalidSDF <- uracilSDFset[!validSDF(uracilSDFset)]
 unlist(lapply(nonvalidSDF@SDF,
     function(x) paste(x@datablock["PUBCHEM_SUBSTANCE_ID"],
     x@datablock["PUBCHEM_SUBS_AUTO_STRUCTURE"], sep=": ")))
```

run restart restart & run all

Sometimes a data depositor does not provide the structure of a chemical but its chemical synonym(s). In that case, PubChem uses the chemical synonyms to assign a structure to this structure-less record. For example, SID 50608295 (one of the 12 structures without SMILES strings in the above output) did not have a depositorprovided structure, but its depositor-provided synonyms include “CID1174”. Therefore, PubChem assigns SID 50608295 to CID 1174, although the depositor did not provide the structure of SID 50608295. (Please check the structure and synonyms for SID 50608295 stored in the SDF file (“cid2sids-uracil.sdf”) generated in step 2).

Step 4. Generate the structure images from the SMILES

Now we want to see what these SMILES strings look like, by drawing molecular structures from them.

```
if(!require("httr", quietly=TRUE)) {
 install.packages("httr", repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("httr")
}
if(!require("png")) {
 install.packages(("png"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("png")
}
if(!require("grid")) {
 install.packages(("grid"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("grid")
}
if(!require("gridExtra")) {
 install.packages(("gridExtra"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("gridExtra")
}
```

run restart restart & run all

```
tabSMIs <- tabSMIs[names(tabSMIs)!=""]
names(tabSMIs)
```

run restart restart & run all

```
vecSMI <- names(tabSMIs)
names(vecSMI) <- names(tabSMIs)

sdf <- smiles2sdf(vecSMI)

# ChemmineR has a function to print the molecules, but
# results are far from acceptable
# png("plots.png",width=400,height=2000)
# ChemmineR::plot(sdf, griddim=c(ceiling(length((vecSMI))/2),2),
# regenCoords=TRUE,atomcex=1 )
#
# dev.off()
# A better option is to use the ChEMBL Beaker API (RDKit online)

write.SDF(sdf,"uracil_unique.sdf")
url <- "https://www.ebi.ac.uk/chembl/api/utils/ctab2image?size=300"
mydata <- list(sdf=upload_file("uracil_unique.sdf"))
res <- POST(url, body = mydata)
img <- readPNG(res$content, native=TRUE)
grid.arrange(rasterGrob(img))
writePNG(img,"uracil_unique.png")
```

run restart restart & run all

You may want to write these molecule images in separate files.

```
dir.create("uracil_unique",showWarnings = FALSE)
fileSDF <- write.SDFsplit(sdf,"uracil_unique/",1)$filename
url <- "https://www.ebi.ac.uk/chembl/api/utils/ctab2image?size=300"

for(i in seq(fileSDF)) {
 mydata <- list(sdf=upload_file(fileSDF[i]))
 res <- POST(url, body = mydata)
```

```
  img <- readPNG(res$content, native=TRUE)
  writePNG(img,paste("uracil_unique/",
  formatC(i,format="d",width=2,flag="0"),".png",sep=""))
  Sys.sleep(1)
}

dir("uracil_unique",pattern=".*[.]png")
```

run restart restart & run all

As shown these chemical images, the 300+ substances associated with CID 1174 (uracil) correspond to 6 tautomeric forms of uracil, which differ from each other in the position of “movable” hydrogen atoms. Compare these structures with their standardized structure (CID 1174).

```
img <- readPNG(GET('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/1174/PNG?i
grid.arrange(rasterGrob(img))
```

run restart restart & run all

Exercise 1a: The function used in Step 3 generates the canonical SMILES string by default. OpenBabel supports a second format (“SMI”) that does not use a canonical numbering when creating the SMILES. Other options can be added. Write a code that generates non-canonical SMILES strings for the 300+ substance records associated with uracil (CID 1174).

- Ignore/skip structure-less records using a conditional statement (i.e., an if statement).
- Print the number of unique non-canonical SMILES.
- Print unique non-canonical SMILES, sorted by frequency.

For a given molecule, there may be multiple ways to write SMILES strings: one of them is selected as the “canonical” SMILES and all the others are considered as “non-canonical”. However, for the purpose of this exercise, we want to generate only one non-canonical SMILES for each record (because the function will return only one SMILES string, the canonical SMILES or one of possible non-canonical SMILES).

run restart restart & run all

Exercise 1b: NOT INCLUDED IN THE R VERSION

Exercise 1c: Retrieve the substance records associated with guanine (CID 135398634) and display unique structures generated from them, by following these steps:

- Retrieve the SIDs associated CID 135398634
- Download the structure data for the retrieved SIDs (in SDF)
- Generate canonical SMILES strings from the structure data in the SDF file and identify unique canonical SMILES strings
- Draw the structures represented by the unqiue canonical SMILES strings in a single figure.

run restart restart & run all

Exercise 1d: Retrieve the substance records whose synonym is “glucose” and display unique structures generated from them, by following these steps: Retrieve the SIDs whose synonym is “glucose”. Download the structure data for the retrieved SIDs (in SDF) Generate canonical SMILES strings from the structure data in the SDF file and identify unique canonical SMILES strings Draw the structures represented by the unique canonical SMILES strings in a single figure.

run restart restart & run all

Exercise 1e: Retrieve the compound records associated with the SIDs retrieved in Exercise 1d and display unique structures generated from them, by following these steps: Retrieve the CIDs associated with the SIDs whose name is “glucose”, using a single PUG-REST request
(i.e., using the list conversion covered in the previous activity, “Interconversion between PubChem records”).

- Identify unique CIDs from the returned CIDs.
- Retrieve the isomeric SMILES for the unique CIDs through PUG-REST.
- Draw the structures represented by the returned SMILES strings in a single figure.

run restart restart & run all

3.11: Assignments

1. Read these articles and answer the following questions.

- What is the difference between a substance and a compound in PubChem?

  (https://pubchemblog.ncbi.nlm.nih.gov/2014/06/19/what-is-the-difference-between-a-substance-and-a-compound-in-pubchem/)

- Compound Summary Page Redesigned

  (https://pubchemblog.ncbi.nlm.nih.gov/2014/10/20/compound-summary-page-redesigned/)

- Substance Record Page Released

  (https://pubchemblog.ncbi.nlm.nih.gov/2015/04/09/substance-record-page-released/)

- PubChem adds a “legacy” designation for outdated data

  (https://pubchemblog.ncbi.nlm.nih.gov/2015/11/16/pubchem-adds-a-legacy-designation-for-outdated-data/)

- “§2.4. Availability of compounds for subsequent experiments” in “Getting the most out of PubChem for virtual screening”

  (http://www.tandfonline.com/doi/full/10.1080/17460441.2016.1216967) [If you don’t have access to this article, Author’s original manuscript for this paper is available as an attachment at the end of Module 4.]

(a) Explain the difference between the PubChem Substance and Compound databases in two or three sentences.

(b) Explain what the Compound Summary page of a compound is.

(c) Explain what the Substance Record page of a substance is.

(d) Explain the reason why the “legacy” designation was introduced in PubChem in two or three sentences.

(e) Among the menus available on the top of the PubChem home page (https://pubchem.ncbi.nlm.nih.gov) is “Today’s Statistics”. The number of compounds/substances/assays shown under this menu does not include “non-live” records. What does “non-live” mean here?

2. While the PubChem Substance database is an archive in nature, data providers often want to update their substance information archived in PubChem. For this reason, PubChem keeps all different “versions” of a substance record and shows the most recent version on its Substance Record page by default (Click **here** to read about what the Substance Record page is). Go to the PubChem home page (https://pubchem.ncbi.nlm.nih.gov) and follow the steps described below.

(a) After selecting the “Compound” tab above the search box, type “60823” in the search box and click the “Go” button. This will direct you to the Compound Summary page for CID 60823 (atorvastatin). (Click **here** to read about what the Compound Summary page is.) [You will learn how to search PubChem in much more detail for next two modules (Modules 5 and 6).]

(b) Scroll down until you see “Contents” on the left column. Expand this table of contents by clicking the “+” sign before “Contents”. Locate the “Related Substances” section and click the record count for the “Same” item under that section.

CHAPTER OVERVIEW

4: Searching Databases for Chemical Information

Hypothes.is Tag= f19OLCCc4

Note: Any annotation tagged **f19OLCCc4** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

Topic hierarchy

4.1: PubChem Web Interfaces for Text

PubChem Homepage

The PubChem homepage (https://pubchem.ncbi.nlm.nih.gov) provides a search interface that allow users to perform any term/keyword/identifier search against all three major databases of PubChem[1,2,3]: Compound, Substance, BioAssay. If a search returns multiple hits, they are presented on an Entrez DocSum page and will be explained in more detail later in this chapter. If the search returns a single record, the user will be directed to the web page that presents information on that record. This page is called the Compound Summary, Substance Record, or BioAssay Record page, depending on the record type (i.e., compound, substance, or assay). In addition, the PubChem homepage provides launch points to various PubChem services, tools, help documents, and more. In general, the PubChem homepage is a central location for all PubChem services.

Entrez Search and Retrieval System

NCBI's Entrez[4,5,6,7] is a database retrieval system that integrates PubChem's three major databases as well as other NCBI's major databases, including PubMed, Nucleotide and Protein Sequences, Protein Structures, Genome, Taxonomy, BioSystems, Gene Expression Omnibus (GEO) and many others. Entrez provides users with an integrated view of biomedical data and their relationships. This section focuses on search and retrieval of PubChem data using the Entrez system. A more detailed description on the Entrez system is given in the following documents:

- The Entrez Search and Retrieval System (http://www.ncbi.nlm.nih.gov/books/NBK184582/)
- Entrez Help (https://www.ncbi.nlm.nih.gov/books/NBK3836/)

Entry points to Entrez

One can search the PubChem databases through Entrez, by initiating a search from the NCBI home page (http://www.ncbi.nlm.nih.gov). By default, if a specific database is not selected in the search menu, Entrez searches all Entrez databases available, and lists the number of records in each database that are returned for this "global query". The following link directs you to the global query result page for the term "AIDS" against all databases integrated in the Entrez system.

https://www.ncbi.nlm.nih.gov/gquery/?term=AIDS

Simply by selecting one of the three PubChem databases from the global query results page (under the Chemical section), one can see the query results specific to that database.

Alternatively, one can start from the PubChem home page (http://pubchem.ncbi.nlm.nih.gov), where a search of one of the three PubChem databases may be initiated through the search box at the top. It is also possible to initiate an Entrez search against a PubChem database from the following pages:

- https://www.ncbi.nlm.nih.gov/pccompound/ (to search the Compound database)
- https://www.ncbi.nlm.nih.gov/pcsubstance/ (to search the Substance database)
- https://www.ncbi.nlm.nih.gov/pcassay/ (to search the BioAssay database)

Entrez DocSums

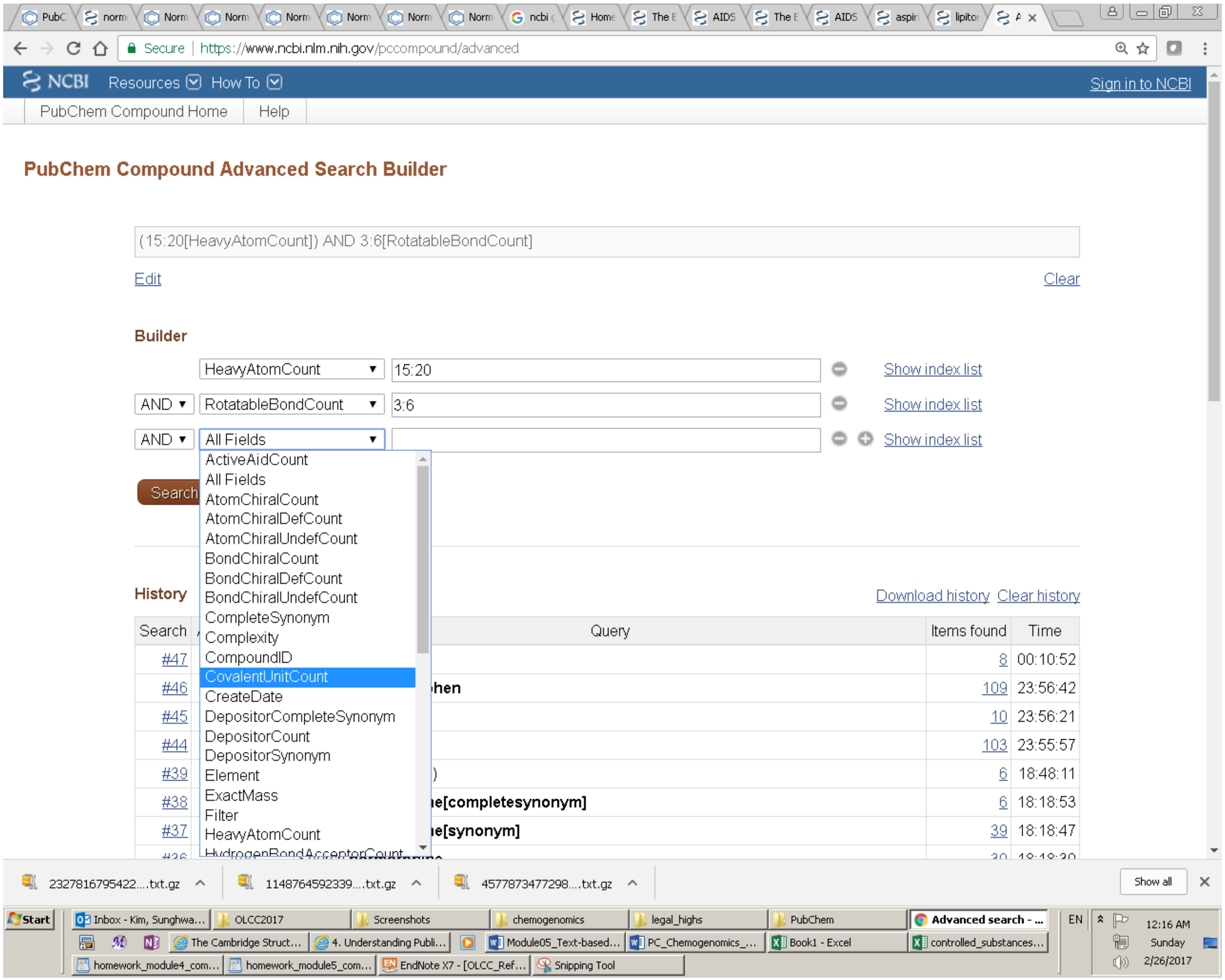
If an Entrez search for a query against any of the three PubChem databases returns a single record, the user will be directed to the Compound Summary, Substance Record, or BioAssay Record page for that record (depending on whether the record is a compound, substance, or assay). If it returns multiple records, Entrez will display a document summary report (also called "DocSum" page). The following link directs you to the DocSum page for a search for the term "lipitor" against the PubChem Compound database:

https://www.ncbi.nlm.nih.gov/pccompound?term=lipitor

In this example, the DocSum page displays a list of the compound records returned from the search. For each record, some data-specific information is provided with a link to the summary page for that record. The DocSum page contains controls to change the display type, to sort the results by various means, or to export the page to a file or printer. Additional controls that operate on a query result list are available on the right column of the DocSum page. The DocSum page for the other two PubChem databases look similar to this example for the Compound database.

Entrez Indices

Entrez indices, tied to individual records in an Entrez database, include information on particular aspects (often referred to as fields) of the records. These indices may have text, numeric or date values, and some indices may have multiple values for each record. The available fields and their indexed terms in any Entrez database can be found from the drop-down menus on the Advanced Search Builder page (which can be accessed by clicking the "Advanced" link next to the "Go" button on the PubChem Home page).



https://www.ncbi.nlm.nih.gov/pccompound/advanced

PubChem Compound Advanced Search Builder

(15:20[HeavyAtomCount]) AND 3:6[RotatableBondCount]

Builder

HeavyAtomCount ▾ 15:20

AND ▾ RotatableBondCount ▾ 3:6

AND ▾ All Fields ▾

History

Search	Query	Items found	Time
#47		8	00:10:52
#46	…hen	109	23:56:42
#45		10	23:56:21
#44		103	23:55:57
#39	…)	6	18:48:11
#38	…e[completesynonym]	6	18:18:53
#37	…e[synonym]	39	18:18:47

When the user enters a query in the Entrez search interface, the Entrez indices are matched directly to that query. By default, in an Entrez search with a simple query, all indexed fields are matched against the query, usually resulting in the largest number of returned records including many unwanted results. One can narrow the search to a particular indexed field, by adding the index name in brackets after the term itself (e.g., "**lipitor[synonym]**"). For numeric indices, a search for a range of values can be done by using minimum and maximum values separated by a colon and followed by the bracketed index name (e.g., "**100:105[MolecularWeight]**"). Multiple indices may be searched simultaneously using Entrez's Boolean operators (e.g., "**AND**", "**OR**" and "**NOT**").

A complete list of the Entrez indices available for the three PubChem databases can be retrieved in the XML format, using the eInfo functionality in E-Utilities (which will be covered in Module 7):

- http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=pccompound (for Compound)
- http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=pcsubstance (for Substance)
- http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=pcassay (for BioAssay).

Additional information on the PubChem Entrez indices is available in the “Indices and Filters in Entrez” section of the help documentation:

https://pubchem.ncbi.nlm.nih.gov/help.html#PubChem_Index

Entrez Links

Entrez links are cross links or associations between records in different Entrez databases, or within the same database. These links may be applied to an entire search result list (via the “find related data” section at the right column of a DocSum page) or to an individual record (via links at the bottom of each record presented on the DocSum page). The Entrez links provide a way to discover relevant information in other Entrez databases based on a user’s specific interests. Equivalently, one may think of this as a way to transform an identifier list from one database to another based on a particular criterion. Note that there are limits to how many records may be used as input in a link operation. To process a large amount of input records and/or to expect a large amount of output records associated with the input records, one should use the FLink tool (https://www.ncbi.nlm.nih.gov/Structure/flink/flink.cgi).

A complete list of the Entrez links available for the three PubChem databases can be retrieved in the XML format through these links

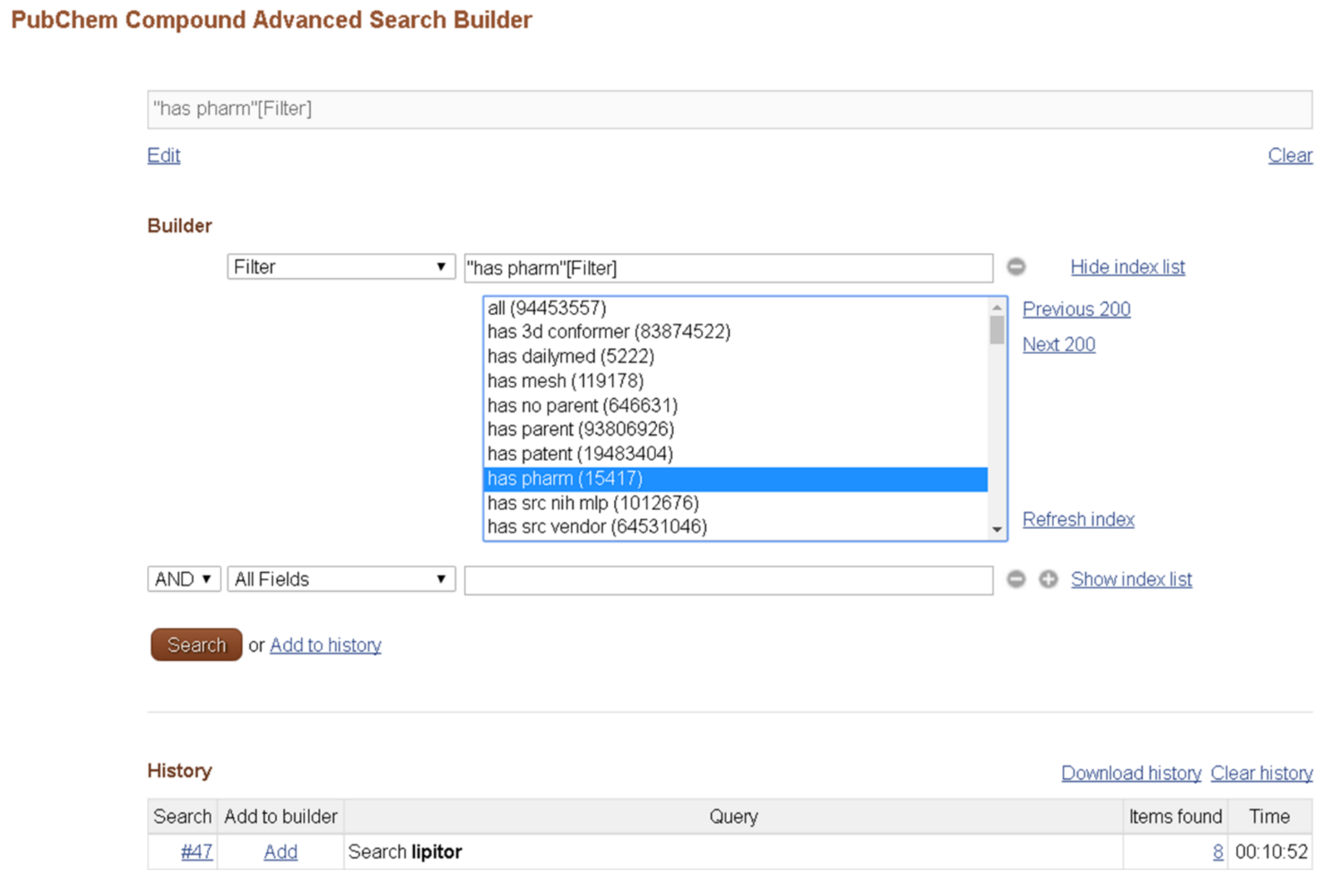
- http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=pccompound (for Compound)
- http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=pcsubstance (for Substance)
- http://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=pcassay (for BioAssay).

Entrez Filters

Entrez filters are essentially Boolean bits (true or false) for all records in a database that indicate whether or not a given record has a particular property. The Entrez filters may be used to subset other Entrez searches according to this property, by adding the filter to the query string.

Entrez filters are closely related to links in that the majority of Entrez filters in the PubChem databases are generated automatically based on whether PubChem records have Entrez links to a given database. However, some special filters, such as the "lipinski rule of 5" filter, or the “all” filter, are not link-based.

The Entrez filters available for each Entrez database may be found on the Advanced Search Builder page by selecting “Filter” from the “All Fields” dropdown and clicking “Show index list”.



PubChem Compound Advanced Search Builder

"has pharm"[Filter]

Builder

Filter: "has pharm"[Filter]

all (94453557)
has 3d conformer (83874522)
has dailymed (5222)
has mesh (119178)
has no parent (646631)
has parent (93806926)
has patent (19483404)
has pharm (15417)
has src nih mlp (1012676)
has src vendor (64531046)

AND All Fields

History

Search	Add to builder	Query	Items found	Time
#47	Add	Search **lipitor**	8	00:10:52

More detailed description of the Entrez filters available for the three PubChem databases are given in the “Indices and Filters in Entrez” section of the help documentation:

https://pubchem.ncbi.nlm.nih.gov/help.html#PubChem_Index

Entrez History

Entrez has a history mechanism (Entrez history) that automatically keeps track of a user’s searches, temporarily caches them (for eight hours), and allows one to combine search result sets with Boolean logic (i.e., “AND”, “OR”, and “NOT”). The Entrez history allows one to limit a search to a subset of records returned from a previous search. Use of Entrez history can help users avoid sending and receiving (potentially) very large lists of identifiers. In addition, through the Entrez history, one can use the search results as an input to various PubChem tools for further manipulation and analysis.

References

(1) Kim, S.; Thiessen, P. A.; Bolton, E. E.; Chen, J.; Fu, G.; Gindulyte, A.; Han, L. Y.; He, J. E.; He, S. Q.; Shoemaker, B. A.; Wang, J. Y.; Yu, B.; Zhang, J.; Bryant, S. H. *Nucleic Acids Res.* **2016**, *44*, D1202.

(2) Wang, Y.; Bryant, S. H.; Cheng, T.; Wang, J.; Gindulyte, A.; Shoemaker, B. A.; Thiessen, P. A.; He, S.; Zhang, J. *Nucleic Acids Res.* **2017**, *45*, D955.

(3) Kim, S. *Expert Opinion on Drug Discovery* **2016**, *11*, 843.

(4) Schuler, G. D.; Epstein, J. A.; Ohkawa, H.; Kans, J. A. *Methods Enzymol.* **1996**, *266*, 141.

(5) McEntyre, J. *Trends in genetics : TIG* **1998**, *14*, 39.

(6) The Entrez Search and Retrieval System (https://www.ncbi.nlm.nih.gov/books/NBK184582/) (Accessed on.

(7) Entrez Help (https://www.ncbi.nlm.nih.gov/books/NBK3836/) (Accessed on.

4.2: Text Search in PubChem

Basics

Text search allows one to find chemical structures using one or more textual keywords, which may be chemical names (e.g., "aspirin") or any word or phrase that describe molecules of interest (e.g., "cyclooxygenase inhibitors"). One can perform a text search from the PubChem homepage, by providing a text query in the search box. If the query is **a phrase or a name with non-alphanumeric characters**, **double quotes should be used around the query**. Various indices can be individually searched by suffixing a text query with an appropriate index enclosed by square brackets (for example, the query ***"N-(4-hydroxyphenyl)acetamide"[iupacname]***). Numeric range searches of appropriate index fields can be performed using a ":" delimiter (for example, the query ***100.5:200[molecularweight]*** for a molecular weight range search between 100.5 and 200.0 g/mol). One can see what search indices are available in PubChem from the drop-down menu on the "PubChem Compound Advanced Search Builder", which can be accessed by clicking the "advanced" link (next to the "Go" button) on the PubChem homepage. Queries may be combined using the Boolean operators "**AND**", "**OR**", and "**NOT**". These Boolean operators must be capitalized.

Depositor-supplied synonyms

Conceptually, data in a database are stored in the same way as we would record them in a table or excel spreadsheet. The rows in the table correspond to compounds, and the columns correspond to properties or descriptions for those compounds (e.g., melting and boiling points, chemical names, toxicity, bioactivity, target proteins, and so on). These columns are commonly called "data fields". You may want to perform a search against all data fields or only a particular field. To search the (depositor-provided) chemical name field of the records in the PubChem Compound database, a chemical name query needs to be suffixed with either of the "[synonym]" or "[completesynonym]" index. The "[synonym]" index invokes search for molecules whose names contain the query chemical name as a part (that is, **partial matching**), and the "[completesynonym]" index invokes search for those whose names completely match the query (that is, **exact matching**). If no index is given after the query, PubChem will search all data fields. Compare the following searches for "aspirin" against the PubChem Compound database.

- **aspirin[completesynonym] (1 hit, as of Feb. 26, 2017)**
  https://www.ncbi.nlm.nih.gov/pccompound/?term=aspirin%5Bcompletesynonym%5D
- **aspirin[synonym] (98 hits)**
  https://www.ncbi.nlm.nih.gov/pccompound/?term=aspirin%5Bsynonym%5D
- **aspirin (103 hits)**
  https://www.ncbi.nlm.nih.gov/pccompound/?term=aspirin

Note that the URLs for these searches contain the query strings (following the string "?term="), and that the square brackets enclosing the Entrez indices "completesynonym" and "synonym" are replaced with the strings "%5B" and "%5D". Because the first query resulted in only one hit, the user is directed to the Compound Summary page for the hit compound (CID 2244). On the other hand, because the other two queries result in multiple hits, the results are presented on the DocSum pages.

When either "[completesynonym]" or "[synonym]" is used, it is the **"depositor-provided synonyms"** fields of the compound records in PubChem that is searched for the query string. The depositor-provided synonyms field for a compound contains a filtered list of chemical names (synonyms) provided by individual data providers for the substances associated with that compound. These synonyms are presented in the "Depositor-provided synonyms" section on a Compound Summary page. To see the variety of synonyms for a compound, check the following link [to the Depositor-provided synonyms" section of the Compound Summary page for CID 2244 (aspirin)]:

https://pubchem.ncbi.nlm.nih.gov/compound/2244#section=Depositor-Supplied-Synonyms

For CID 2244, there are more than 700 depositor-supplied synonyms. These synonyms include not only those commonly used in chemistry class (e.g., common names, IUPAC names, CAS registry numbers) but also those used in many other places (e.g., database identifiers, chemical vendor catalogues, the name of products that contains the chemical, code numbers internally used in a company, and so on).

As mentioned above, the search for aspirin with the "[completesynonym]" index specified returns only one compound (CID 2244). It means that one of many names of this compound exactly matches the query string "aspirin". On the other hand, the search for aspirin with the "[synonym]" index returns additional 97 compounds. It means that at least one of the names of each these compound partially match the query string (that is, the compound contains the string “aspirin” in one of its names). Interestingly, the results from the last two queries include acetaminophen (CID 1983), which is the active ingredient of Tylenol. Check the following link to the depositor-provided synonyms section of CID 1983 to see what synonyms of Tylenol contains the string “aspirin”:

https://pubchem.ncbi.nlm.nih.gov/compound/1983#section=Depositor-Supplied-Synonyms

Some of the synonyms of Tylenol contains the phrase “aspirin-free” or “non-aspirin”. Note that Tylenol was returned from a search for “aspirin” (through partial matching using the [synonym] index).

MeSH Synonyms

The National Library of Medicine (NLM)’s Medical Subject Headings (MeSH)[1,2]

is a controlled vocabulary thesaurus of medical terms arranged in a hierarchical structure. It is used for indexing scientific articles from biomedical journals for PubMed and cataloging medical books, documents, and audiovisual materials, in order to facilitate retrieval of medical information at various levels of specificity.

Many of MeSH terms are chemical names (e.g., for drugs, nutrients, metabolites, toxic chemicals, and so on). PubChem performs an automated annotations of PubChem records with MeSH terms (by means of chemical name matching), creating associations between PubChem records and PubMed articles that share the same MeSH annotation. The MeSH term that match a (depositor-provided) synonym of a compound in PubChem is presented with its entry terms under the “MeSH Synonyms” section of the Compound Summary page of that compound.

Go to the Compound Summary page for CID 171511 via the following link to check the MeSH synonyms and Depositor-supplied Synonyms sections.

References

(1) Medical Subject Headings (MeSH) (https://www.nlm.nih.gov/mesh/)

(2) Medical Subject Headings (MeSH®) Fact Sheet (https://www.nlm.nih.gov/pubs/factsheets/mesh.html)

4.3: Additional Data Retrieval Approaches in PubChem

Classification Browser

The PubChem Classification Browser, which allows the user to navigate or search PubChem records associated to a hierarchical classification system of interest, is available via URL:

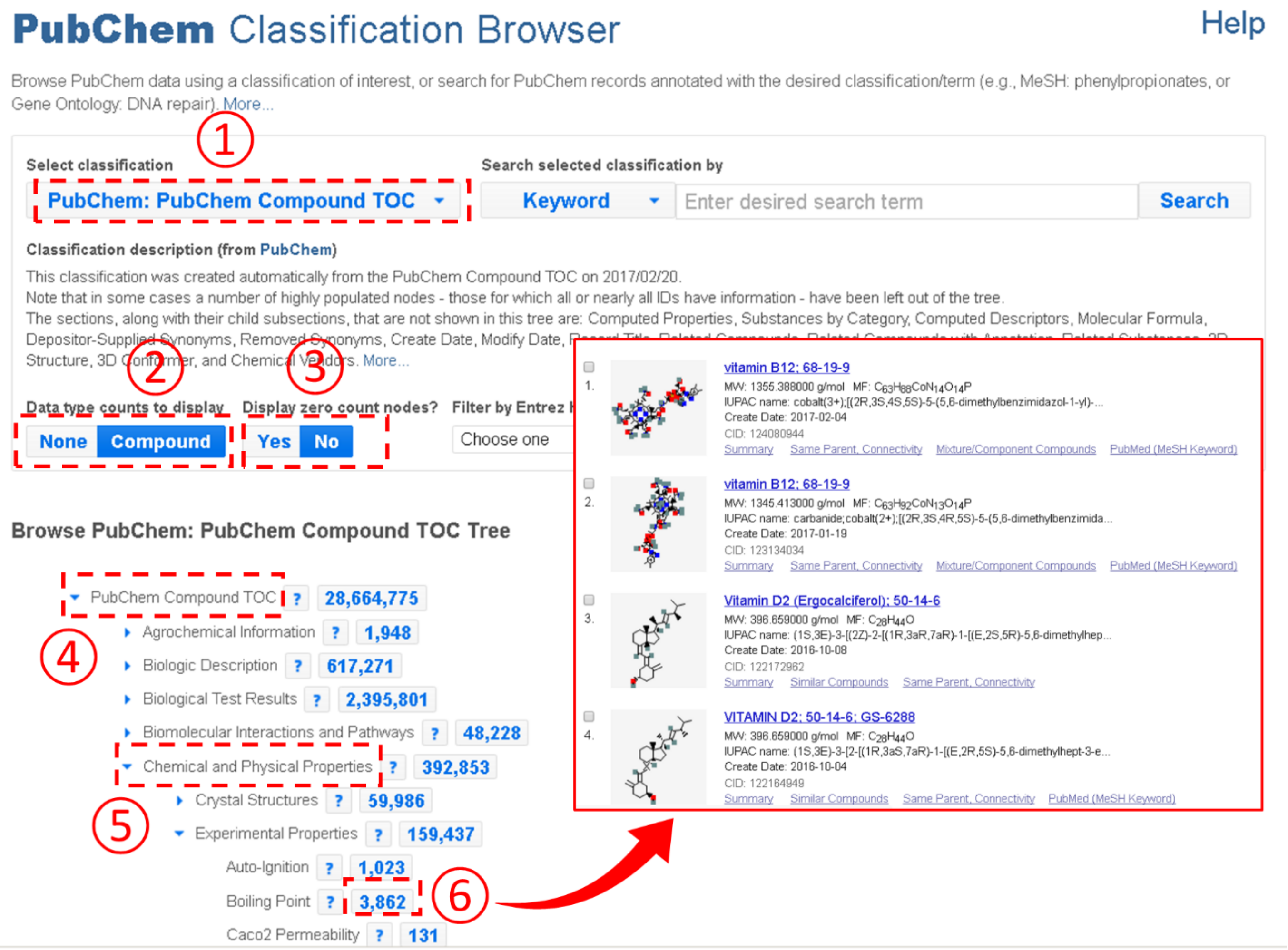
http://pubchem.ncbi.nlm.nih.gov/classification

The Classification Browser can also be accessed from the PubChem home page (through the “Services” menu at the top or the “Classification” icon on the right column of the page). Currently, the Classification Browser can retrieve records annotated with terms in the following classification systems:

- MeSH (Medical Subject Headings)
- ChEBI
- FDA Pharmacological Classification
- KEGG
- LIPID MAPS
- World Health Organization (WHO)’s Anatomical Therapeutic Chemical (ATC)
- World Intellectual Property Organization (WIPO)’s IPC (International Patent Classification)

The Classification Browser provides a powerful way to quickly and visually find a desired subset of PubChem records. The output can be displayed in Tree view or List view.

An important feature of the Classification Browser is that **the Table of Contents presented on the Compound Summary is integrated into the Classification Browser, allowing users to quickly retrieve compounds with a particular type of information available**. For example, the figure below shows how to retrieve all compounds with the boiling point information from PubChem.



PubChem Classification Browser

Browse PubChem data using a classification of interest, or search for PubChem records annotated with the desired classification/term (e.g., MeSH: phenylpropionates, or Gene Ontology: DNA repair). More...

Select classification: PubChem: PubChem Compound TOC

Search selected classification by: Keyword | Enter desired search term | Search

Classification description (from PubChem)

This classification was created automatically from the PubChem Compound TOC on 2017/02/20.
Note that in some cases a number of highly populated nodes - those for which all or nearly all IDs have information - have been left out of the tree.
The sections, along with their child subsections, that are not shown in this tree are: Computed Properties, Substances by Category, Computed Descriptors, Molecular Formula, Depositor-Supplied Synonyms, Removed Synonyms, Create Date, Modify Date, R
Structure, 3D Conformer, and Chemical Vendors. More...

Data type counts to display: None [✓ Compound]

Display zero count nodes? Yes [✓ No]

Filter by Entrez: Choose one

Browse PubChem: PubChem Compound TOC Tree

- PubChem Compound TOC ? 28,664,775
  - Agrochemical Information ? 1,948
  - Biologic Description ? 617,271
  - Biological Test Results ? 2,395,801
  - Biomolecular Interactions and Pathways ? 48,228
  - Chemical and Physical Properties ? 392,853
    - Crystal Structures ? 59,986
    - Experimental Properties ? 159,437
      - Auto-Ignition ? 1,023
      - Boiling Point ? 3,862
      - Caco2 Permeability ? 131

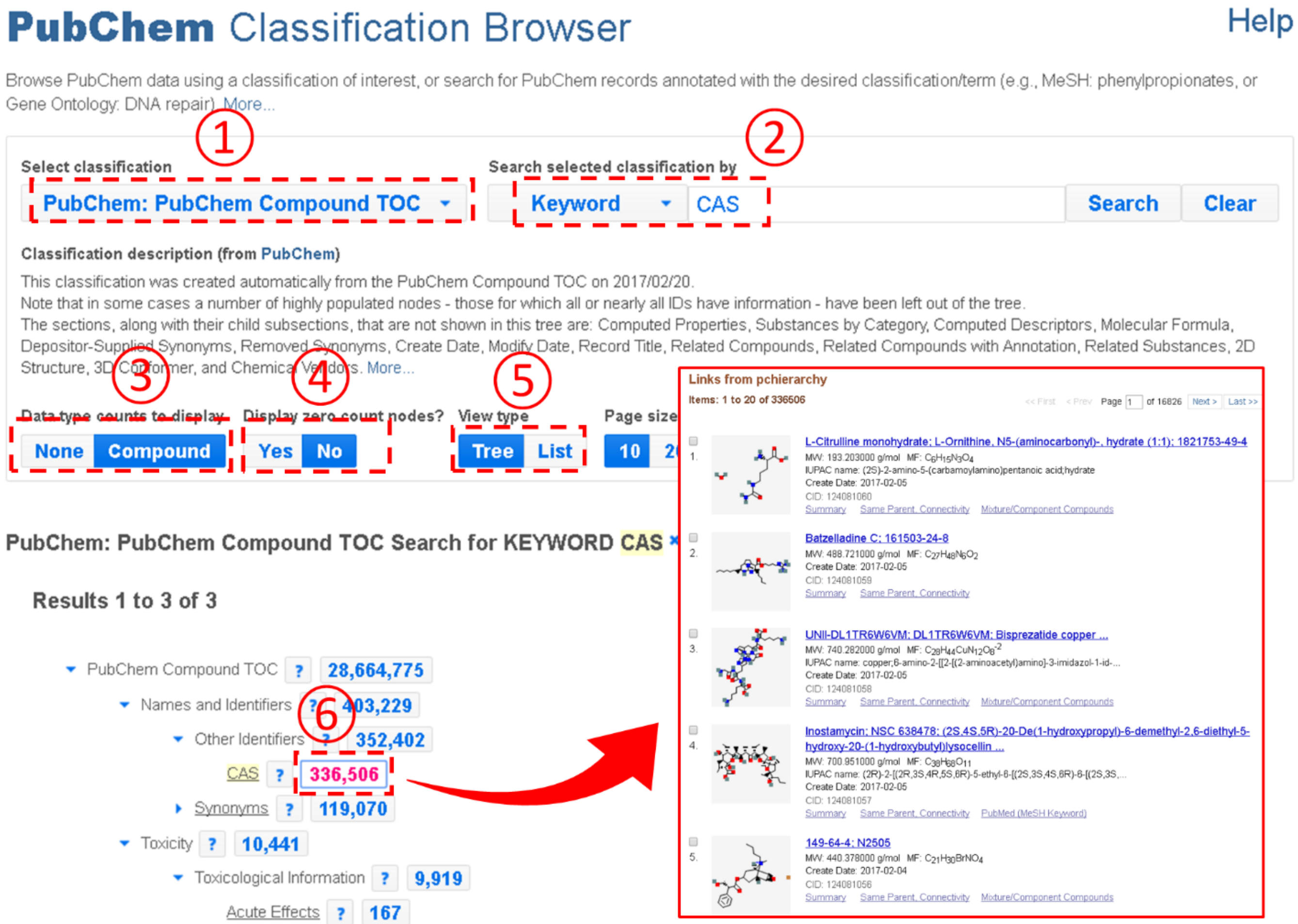
1. vitamin B12; 68-19-9
MW: 1355.388000 g/mol MF: $C_{63}H_{88}CoN_{14}O_{14}P$
IUPAC name: cobalt(3+);[(2R,3S,4S,5S)-5-(5,6-dimethylbenzimidazol-1-yl)-...
Create Date: 2017-02-04
CID: 124080944
Summary Same Parent, Connectivity Mixture/Component Compounds PubMed (MeSH Keyword)

2. vitamin B12; 68-19-9
MW: 1345.413000 g/mol MF: $C_{63}H_{92}CoN_{13}O_{14}P$
IUPAC name: carbanide;cobalt(2+);[(2R,3S,4R,5S)-5-(5,6-dimethylbenzimida...
Create Date: 2017-01-19
CID: 123134034
Summary Same Parent, Connectivity Mixture/Component Compounds PubMed (MeSH Keyword)

3. Vitamin D2 (Ergocalciferol); 50-14-6
MW: 396.659000 g/mol MF: $C_{28}H_{44}O$
IUPAC name: (1S,3E)-3-[(2Z)-2-[(1R,3aR,7aR)-1-[(E,2S,5R)-5,6-dimethylhep...
Create Date: 2016-10-08
CID: 122172962
Summary Similar Compounds Same Parent, Connectivity

4. VITAMIN D2; 50-14-6; GS-6288
MW: 396.659000 g/mol MF: $C_{28}H_{44}O$
IUPAC name: (1S,3E)-3-[2-[(1R,3aS,7aR)-1-[(E,2R,5S)-5,6-dimethylhept-3-e...
Create Date: 2016-10-04
CID: 122164949
Summary Similar Compounds Same Parent, Connectivity PubMed (MeSH Keyword)

In the example above, users need to expand the Table of Contents tree to locate the boiling point node. However, this task may not be easy to some users who do not have prior knowledge about where the node that they want to find is located in the Table of Contents tree system. To assist these users, the Classification Brower supports a keyword search against the node names and descriptions of the classification trees. For example, the example below shows how to retrieve compounds with the CAS Registry number. Note that this task involves a search for the term “CAS”.



PubChem Classification Browser

Help

Browse PubChem data using a classification of interest, or search for PubChem records annotated with the desired classification/term (e.g., MeSH: phenylpropionates, or Gene Ontology: DNA repair). More...

Select classification: PubChem: PubChem Compound TOC

Search selected classification by: Keyword CAS

Search Clear

Classification description (from PubChem)

This classification was created automatically from the PubChem Compound TOC on 2017/02/20.
Note that in some cases a number of highly populated nodes - those for which all or nearly all IDs have information - have been left out of the tree.
The sections, along with their child subsections, that are not shown in this tree are: Computed Properties, Substances by Category, Computed Descriptors, Molecular Formula, Depositor-Supplied Synonyms, Removed Synonyms, Create Date, Modify Date, Record Title, Related Compounds, Related Compounds with Annotation, Related Substances, 2D Structure, 3D Conformer, and Chemical Vendors. More...

Data type counts to display: None [✓ Compound]

Display zero-count nodes? Yes [✓ No]

View type: [✓ Tree] List

Page size: [✓ 10] 20

PubChem: PubChem Compound TOC Search for KEYWORD CAS

Results 1 to 3 of 3

- PubChem Compound TOC ? 28,664,775
  - Names and Identifiers ? 403,229
    - Other Identifiers ? 352,402
      - CAS ? 336,506
      - Synonyms ? 119,070
  - Toxicity ? 10,441
    - Toxicological Information ? 9,919
      - Acute Effects ? 167

Links from pchierarchy

Items: 1 to 20 of 336506

<< First < Prev Page 1 of 16826 Next > Last >>

1. L-Citrulline monohydrate; L-Ornithine, N5-(aminocarbonyl)-, hydrate (1:1); 1821753-49-4
MW: 193.203000 g/mol MF: $C_6H_{15}N_3O_4$
IUPAC name: (2S)-2-amino-5-(carbamoylamino)pentanoic acid;hydrate
Create Date: 2017-02-05
CID: 124081060
Summary Same Parent, Connectivity Mixture/Component Compounds

2. Batzelladine C; 161503-24-8
MW: 488.721000 g/mol MF: $C_{27}H_{48}N_6O_2$
Create Date: 2017-02-05
CID: 124081059
Summary Same Parent, Connectivity

3. UNII-DL1TR6W6VM; DL1TR6W6VM; Bisprezatide copper ...
MW: 740.282000 g/mol MF: $C_{28}H_{44}CuN_{12}O_8^{-2}$
IUPAC name: copper;6-amino-2-[[2-[(2-aminoacetyl)amino]-3-imidazol-1-id-...
Create Date: 2017-02-05
CID: 124081058
Summary Same Parent, Connectivity Mixture/Component Compounds

4. Inostamycin; NSC 638478; (2S,4S,5R)-20-De(1-hydroxypropyl)-6-demethyl-2,6-diethyl-5-hydroxy-20-(1-hydroxybutyl)lysocellin ...
MW: 700.951000 g/mol MF: $C_{38}H_{68}O_{11}$
IUPAC name: (2R)-2-[(2R,3S,4R,5S,6R)-5-ethyl-6-[(2S,3S,4S,6R)-6-[(2S,3S,...
Create Date: 2017-02-05
CID: 124081057
Summary Same Parent, Connectivity PubMed (MeSH Keyword)

5. 149-64-4; N2505
MW: 440.378000 g/mol MF: $C_{21}H_{30}BrNO_4$
Create Date: 2017-02-04
CID: 124081056
Summary Same Parent, Connectivity Mixture/Component Compounds

The Classification Browser also supports the **PubChem BioAssay Classification Tree**, providing an additional approach to browse, search, and access the BioAssay data. More detailed information on the Classification Browser is available at the URL:

http://pubchem.ncbi.nlm.nih.gov//classification/docs/classification_help.html

Identifier Exchange Service

The Identifier Exchange Service can be found at the following URL: http://pubchem.ncbi.nlm.nih.gov/idexchange

This service allows the user to convert one type of identifiers for a given set of chemical structures into a different type of identifiers for identical or similar chemical structures. Currently, it supports seven types of identifiers: CID, SID, InChI, InChIKey, SMILES, synonyms, Registry ID. When Registry ID is selected as an input or output identifier type, the DSN (Data Source Name) should also be provided.

The input identifier list may be provided using a string, a text file, or Entrez history. When a service request is submitted, it will be queued on PubChem servers. Once the actual task starts to run, the input identifiers will be converted into CIDs (called input CIDs) during the computation, and the CIDs (called output CIDs) that satisfy the condition specified by one of the following operation types will be retrieved:

- **Same CID**: Same CIDs as input CIDs.
- **Same, Stereochemistry**: CIDs that have same stereo centers as input CIDs.
- **Same, Isotopes**: CIDs that have the same isotopes as input CIDs.
- **Same, Connectivity**: CIDs that have the same connectivity as input CIDs.
- **Same parent**: CIDs that have the same parents as input CIDs.
- **Same parent, Stereochemistry**: CIDs that have the same stereo centers and parents as input CIDs.
- **Same parent, Isotopes**: CIDs that have the same isotopes and parents as input CIDs.

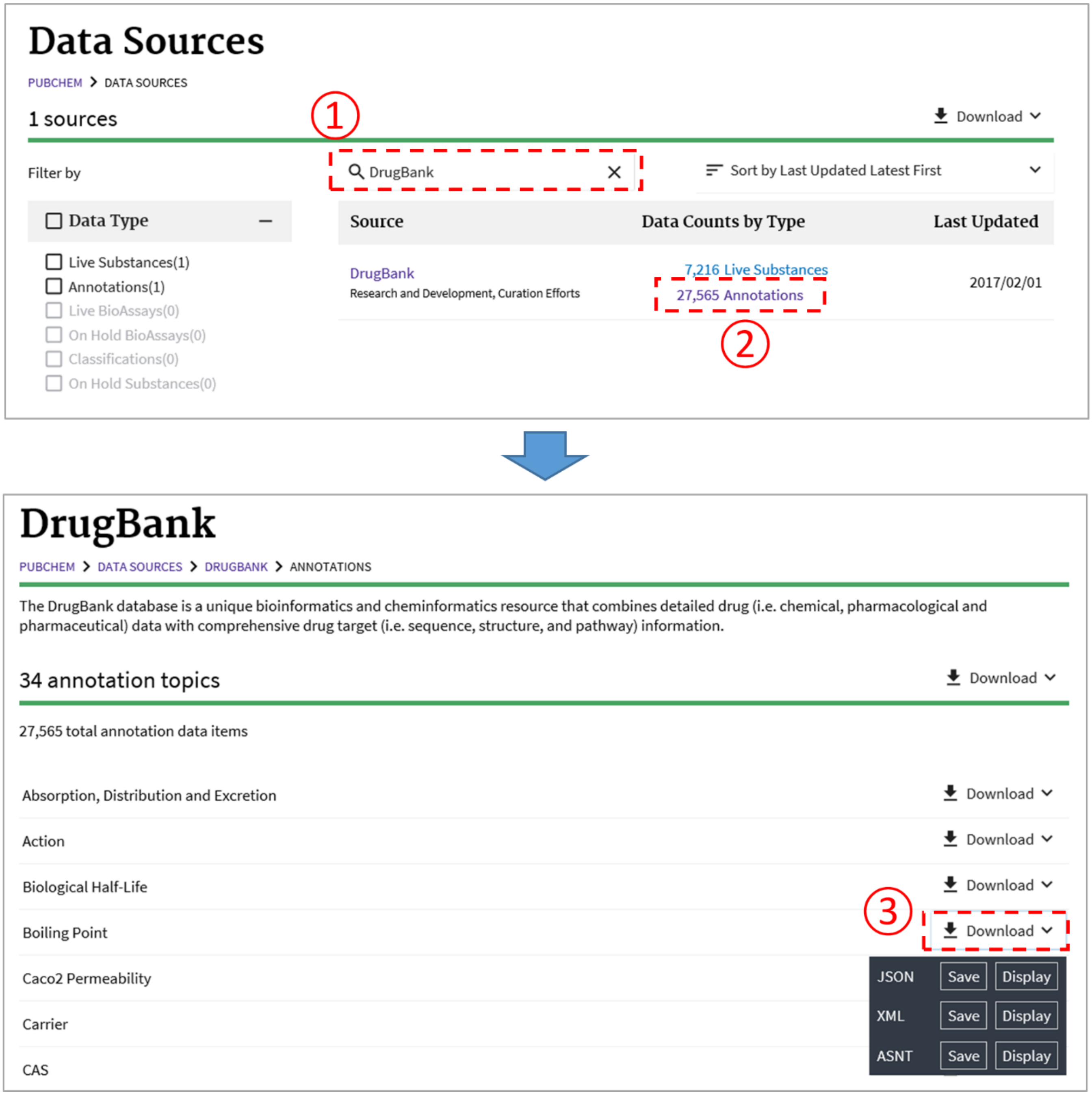
- **Same parent, Connectivity**: CIDs that have the same connectivity and parents as input CIDs.
- **Similar 2D compounds**: CIDs similar to the input CIDs in PubChem's 2-D similarity.
- **Similar 3D conformers**: CIDs similar to the input CIDs in PubChem's 3-D similarity.

These output CIDs are then converted into the identifier type specified by the user and written into a file or sent to **Entrez history**. In practice, the identifier exchange service may be used as a quick approach to search the PubChem Compound database using multiple queries, although this type of task may be performed programmatically (for example, using PUG-REST,[1] which will be discussed in Module 7). A more detailed information is available at the URL:

> http://pubchem.ncbi.nlm.nih.gov//idexchange/idexchange-help.html

The PubChem Data Sources page

As discussed in Section 3.4, the PubChem Data Sources page (https://pubchem.ncbi.nlm.nih.gov/sources/) helps users determine who provided what information. This page can be used to retrieve the data provided by a data depositor or to download the annotations collected from a data source. For example, the following figure illustrates how to download the boiling point data collected from DrugBank.[2]



Data Sources

PUBCHEM > DATA SOURCES

1 sources

Download

Filter by

DrugBank

Sort by Last Updated Latest First

□ Data Type

□ Live Substances(1)
□ Annotations(1)
□ Live BioAssays(0)
□ On Hold BioAssays(0)
□ Classifications(0)
□ On Hold Substances(0)

Source	Data Counts by Type	Last Updated
DrugBank Research and Development, Curation Efforts	7,216 Live Substances 27,565 Annotations	2017/02/01

DrugBank

PUBCHEM > DATA SOURCES > DRUGBANK > ANNOTATIONS

The DrugBank database is a unique bioinformatics and cheminformatics resource that combines detailed drug (i.e. chemical, pharmacological and pharmaceutical) data with comprehensive drug target (i.e. sequence, structure, and pathway) information.

34 annotation topics

Download

27,565 total annotation data items

Absorption, Distribution and Excretion — Download
Action — Download
Biological Half-Life — Download
Boiling Point — Download
Caco2 Permeability
Carrier
CAS

To obtain a particular kind of annotated information (e.g., boiling points) through the PubChem Data Sources page, one may need to know "in advance" which depositors provide that information. This can be done through a PUG-REST request[1] (to be discussed in detail in Module 7). For example, the following PUG-REST request returns all data sources that provide the boiling point information for chemicals.

https://pubchem.ncbi.nlm.nih.gov/rest/pug/annotations/heading/boiling%20point/TXT

On the other hand, one may want to know what kind of information is provided by a given data source. This can also be done using a PUG-REST request:

https://pubchem.ncbi.nlm.nih.gov/rest/pug/annotations/sourcename/DrugBank/TXT

This example retrieves all types of annotations collected from DrugBank.

References

(1) Kim, S.; Thiessen, P. A.; Bolton, E. E.; Bryant, S. H. *Nucleic Acids Res.* **2015**, *43*, W605.

(2) Law, V.; Knox, C.; Djoumbou, Y.; Jewison, T.; Guo, A. C.; Liu, Y. F.; Maciejewski, A.; Arndt, D.; Wilson, M.; Neveu, V.; Tang, A.; Gabriel, G.; Ly, C.; Adamjee, S.; Dame, Z. T.; Han, B. S.; Zhou, Y.; Wishart, D. S. *Nucleic Acids Res.* **2014**, *42*, D1091.

4.4: Searching PubChem Using a Non-Textual Query

This section describes various searches that can be performed in PubChem.[1,2,3] Currently PubChem has three different search interfaces:

1. PubChem homepage (http://pubchem.ncbi.nlm.nih.gov)
2. PubChem Chemical Structure Search (https://pubchem.ncbi.nlm.nih.gov/search/search.cgi)
3. PubChem Search (https://pubchem.ncbi.nlm.nih.gov/search/).

As explained in Section 4.1, the PubChem homepage provides a search interface for all three primary databases (e.g., Substance, Compound, and BioAssay). However, the search box on the PubChem homepage can accepts textual keywords only, and it is difficult to input non-textual queries (such as chemical structures). The PubChem Chemical Structure Search allows users to perform various searches using both textual and non-textual queries. This search interface is integrated with PubChem Sketcher,[4] which enables users to provide the 2-D structure of a molecule as a query for chemical structure search. While the PubChem Chemical Structure Search is limited to search for chemical structures, the PubChem Search allows users to search for bioassays, bioactivities, patents, and targets as well as chemical structures, but it is still in beta testing. In this module, we use the Chemical Structure Search for chemical structure search.

Molecular formula search

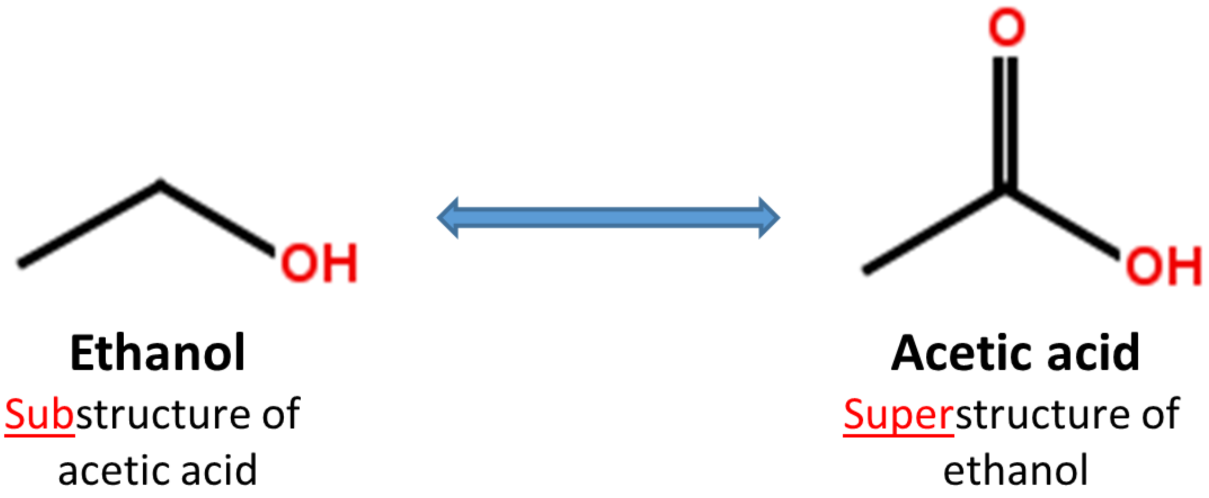
Molecular formula search allows one to find molecules that contain a certain number and type of elements. Typically, molecular formula search returns by default molecules that exactly match the queried stoichiometry. For example, a query of “C_6H_6” will return all structures containing six carbon atoms, six hydrogen atoms and nothing else. However, molecular formula search implemented in some databases, including PubChem Chemical Structure Search, has an option to allow other elements in returned hits (e.g., C_6H_6O or $C_6H_6N_2O$ for the “C_6H_6” query).

Identity search

Identity search is to locate a particular chemical structure that is “identical” to the query chemical structure. Although identity search seems conceptually straightforward, one should keep in mind that the word “identical” can have different notions. For example, if a molecule exists as multiple tautomeric forms in equilibrium, do you want to consider all these tautomers identical and search the database for all of them? If your query molecule has a chiral stereo center, should you consider both R- and S-forms in your search? In your identity search, do you want to include isotopically substituted species of the provided query molecule as well as the query itself? Depending on how to deal with these nuances of chemical structures, identical search will return different results. The identity search in the PubChem Chemical Structure Search allows users to choose a desired degree of “sameness” from several predefined options. To see these options, one need to expand the options section by clicking the “plus” button next to the “option” section heading.

Substructure and superstructure search

When a chemical structure occurs as a part of a bigger chemical structure, the former is called a *substructure* and the latter is referred to as a *superstructure*. For example, ethanol is a substructure of acetic acid, and acetic acid is a superstructure of ethanol.



In substructure search, one provides an input substructure as a query to find molecules that contain the query substructure (that is, superstructures that contain the query substructure). On the contrary, superstructure search returns molecules that comprise or make up the provided chemical structure query (that is, substructures that is contained in the query superstructure). It should be noted that substructure search does *not* give you substructures of the query and that superstructure search does *not* return superstructures of the query.

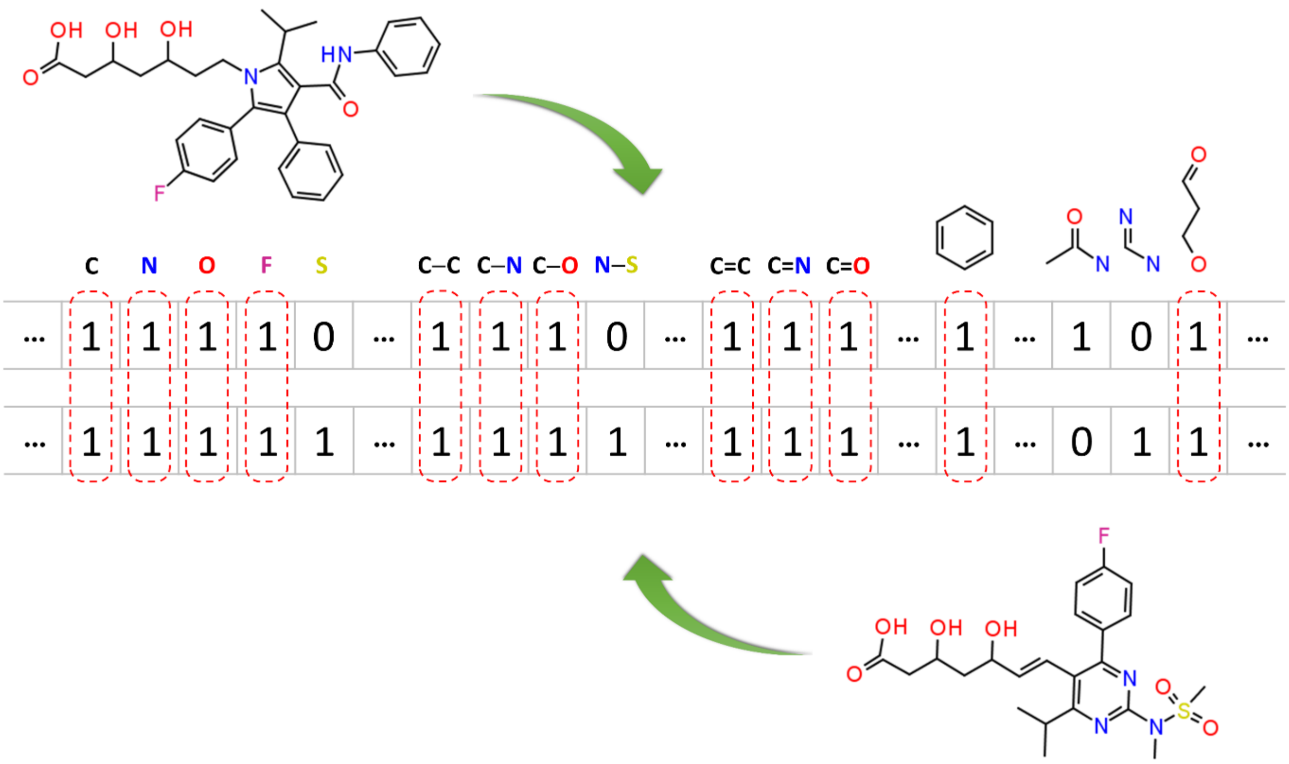
It is possible to include explicit hydrogen atoms as part of the pattern being searched. For example, if you choose to do so, the SMILES queries [CH2][CH2][OH] and [CH3][CH][OH] will return molecules whose formula are R-CH$_2$-CH$_2$-OH and CH$_3$-CH(R)-OH, respectively. Substructure/superstructure searches implemented in some databases remove by default explicit hydrogens from the query molecule prior to search, the two SMILES queries [CH2][CH2][OH] and [CH3][CH][OH] may give you the same result as what the SMILES query CCO does, unless you specify that explicit hydrogens should be included in pattern matching.

In addition to explicit hydrogen atoms, there are additional factors that may affect results of substructure/superstructure searches, for example, whether to ignore stereochemistry, isotopism, tautomerism, formal charge, and so on.

Similarity search

Molecular similarity (also called chemical similarity or chemical structure similarity) is a fundamental concept in cheminformatics, playing an important role in computational methods for predicting properties of chemical compounds as well as designing chemicals with desired properties. The underlying assumption in these computational methods is that structurally similar molecules are likely to have similar biological and physicochemical properties (commonly called the similarity principle).[5] Molecular similarity is a straightforward and easy-to-understand concept, but there is no absolute, mathematical definition of molecular similarity that everyone agrees on. As a result, there are a virtually infinite number of molecular similarity methods, which quantify molecular similarity. Similarity search uses a molecular similarity method to find molecules similar to the query structure.

Two-dimensional (2-D) similarity methods



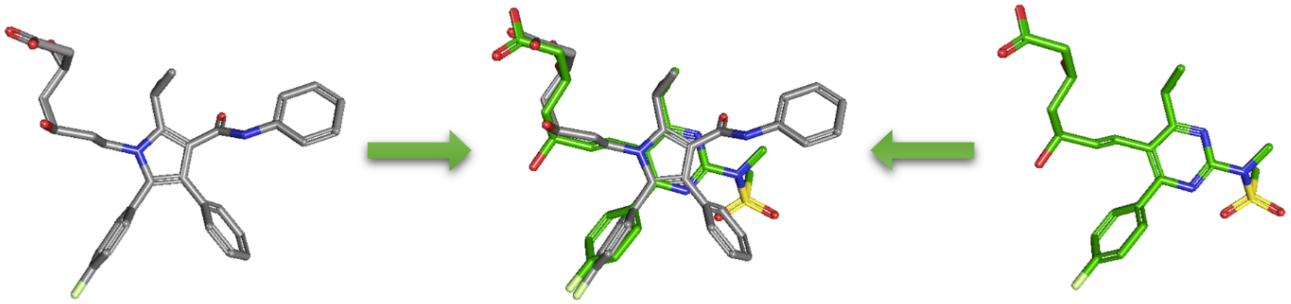
Molecular similarity methods can be broadly classified into two-dimensional (2-D) and three-dimensional (3-D) similarity methods. Typically, 2-D similarity methods use so-called molecular fingerprints. The most common types of molecular fingerprints are structural keys, which encode structural information of a molecule into a binary string (*that is*, a string of 0’s and 1’s). The position of each number in this string corresponds to a particular fragment. If the molecule has a particular fragment, the corresponding bit position is set to 1, and otherwise to 0. Note that there are many different ways to design molecular fingerprints, depending on what fragments are included in the fingerprint definition. PubChem uses its own fingerprint called PubChem subgraph fingerprints.

In 2-D similarity methods, structural similarity between two molecules is estimated by comparing their molecular fingerprints. Their similarity is quantified as a so-called similarity score or similarity coefficient. While several different methods can be used for computation of a similarity score, the underlying ideas are the same as each other: if the two fingerprints have 1’s at the same position, it means that both compounds have the same fragment, and if the molecules share more common fragments, they are considered to be more similar. In conjunction with the PubChem subgraph fingerprints, PubChem 2-D similarity method use the Tanimoto coefficient[6-8]

$$Tanimoto = \frac{N_{AB}}{N_A + N_B - N_{AB}} \tag{4.4.1}$$

where N_A and N_B are the number of bits set in the fingerprints for molecules A and B, respectively, and N_{AB} is the number of bits set in both fingerprints. The Tanimoto score ranges from 0 (for no similarity) to 1 (for identical molecules). 2-D Similarity search returns molecules whose similarity scores with the query molecule are greater than or equal to a given Tanimoto cut-off value.

PubChem 3-D similarity method



As an alternative to 2-D similarity search, 3-D similarity search can also be performed using the “3D conformer” tab in PubChem Chemical Structure Search. 3-D similarity methods use the 3-D structures (that is, conformations) of molecules. PubChem’s 3-D similarity method is based on the atom-centered Gaussian-shape comparison method by Grant and coworkers,[9,10,11,12] implemented in the Rapid Overlay of Chemical Structures (ROCS).[13,14] While the underlying mathematics of this approach is beyond the scope of this module, what this method essentially does is to find the “best” alignment of the 3-D structures of two molecules, which gives the maximized overlap between them. The 3-D similarity method quantifies the 3-D molecular similarity using three metrics.

- **Shape-Tanimoto (ST)**: quantifies steric shape similarity between two conformers.
- **Color-Tanimoto (CT)**: quantifies the overlap of functional groups between two conformers, such as hydrogen bond donors and acceptors, cations, anions, rings, and hydrophobes.
- **Combo-Tanimoto (ComboT)**: the sum of ST and CT scores between two conformers. It takes into account the shape similarity (ST) and functional group similarity (CT) simultaneously.

Because both the ST and CT scores range from 0 (for no similarity) to 1 (for identical molecules), the ComboT score may have a value from 0 to 2 (without normalization to unity). Note that the ST, CT and ComboT scores between two molecules can be evaluated in two different molecular superpositions: (1) in the ST- or shape-optimized superpositions, and (2) in the CT- or feature-optimization superpositions. In the ST-optimization approach, the shape overlap between the molecules (that is, the ST score) are maximized and the single-point CT score is evaluated at that superposition. On the contrary, the CT-optimization considers both ST and CT scores to find the best superposition between molecules, and the single-point ST score is computed at that superposition.

The 3-D similarity method used in PubChem requires the 3-D structures of molecules. PubChem generates a conformer ensemble containing up to 500 conformers for each compound that satisfy the following conditions[15,16,17]:

- Not too big or too flexible (with £ 50 non-hydrogen atoms and ≤ 15 rotatable bonds).
- Have only a single covalent unit (i.e., not a salt or a mixture).
- Consist of only supported elements (H, C, N, O, F, Si, P, S, Cl, Br, and I).
- Contain only atom types recognized by the MMFF94s force field.
- Fewer than six undefined atom or bond stereo centers.

About 90% of compounds in PubChem have computationally generated conformer models. Although each compound has up to 500 conformers (depending on the molecular size and flexibility), many PubChem tools and services support up to 10 conformers per compound. It should be emphasized that these conformers are not energy-minimized but sampled from the conformational space of a given molecule in such a way that the sampled conformers represent the overall diversity of shape and feature of the molecule.[15,16,17] These conformer models aim to generate bioactive conformers, which would be found in protein-ligand complexes. For this reason, these conformers are often very different from their experimental structures determined in the gas phase.

References

(1) Kim, S.; Thiessen, P. A.; Bolton, E. E.; Chen, J.; Fu, G.; Gindulyte, A.; Han, L. Y.; He, J. E.; He, S. Q.; Shoemaker, B. A.; Wang, J. Y.; Yu, B.; Zhang, J.; Bryant, S. H. *Nucleic Acids Res.* **2016**, *44*, D1202.

(2) Wang, Y.; Bryant, S. H.; Cheng, T.; Wang, J.; Gindulyte, A.; Shoemaker, B. A.; Thiessen, P. A.; He, S.; Zhang, J. *Nucleic Acids Res.* **2017**, *45*, D955.

(3) Kim, S. *Expert Opinion on Drug Discovery* **2016**, *11*, 843.

(4) Ihlenfeldt, W. D.; Bolton, E. E.; Bryant, S. H. *J. Cheminform.* **2009**, *1*, 20.

(5) *Concepts and Applications of Molecular Similarity*; Johnson, M. A.; Maggiora, G. M., Eds.; John Wiley & Sons, Inc.: New York, NY, 1990.

(6) Chen, X.; Reynolds, C. H. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 1407.

(7) Holliday, J. D.; Hu, C. Y.; Willett, P. *Combinatorial Chemistry & High Throughput Screening* **2002**, *5*, 155.

(8) Holliday, J. D.; Salim, N.; Whittle, M.; Willett, P. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 819.

(9) Grant, J. A.; Pickup, B. T. *Journal of Physical Chemistry* **1995**, *99*, 3503.

(10) Grant, J. A.; Gallardo, M. A.; Pickup, B. T. *Journal of Computational Chemistry* **1996**, *17*, 1653.

(11) Grant, J. A.; Pickup, B. T. *Journal of Physical Chemistry* **1996**, *100*, 2456.

(12) Grant, J. A.; Pickup, B. T. In *Computer Simulation of Biomolecular Systems*; van Gunsteren, W. F., Weiner, P. K., Wilkinson, A. J., Eds.; Kluwer Academic Publishers: Dordrecht, 1997, p 150.

(13) Rush, T. S.; Grant, J. A.; Mosyak, L.; Nicholls, A. *Journal of Medicinal Chemistry* **2005**, *48*, 1489.

(14) 3.1.0 ed.; OpenEye Scientific Software, Inc.: Santa Fe, NM, 2010.

(15) Bolton, E. E.; Chen, J.; Kim, S.; Han, L. Y.; He, S. Q.; Shi, W. Y.; Simonyan, V.; Sun, Y.; Thiessen, P. A.; Wang, J. Y.; Yu, B.; Zhang, J.; Bryant, S. H. *J. Cheminform.* **2011**, *3*, 32.

(16) Bolton, E. E.; Kim, S.; Bryant, S. H. *J. Cheminform.* **2011**, *3*, 4.

(17) Kim, S.; Bolton, E. E.; Bryant, S. H. *J. Cheminform.* **2013**, *5*, 1.

4.5: Programming Topics

- Know how to formulate a PUG-REST request URL.
- Know how to access PubChem data from a spread sheet (in Google Sheet)
- Know how to access PubChem data from a python script.

Useful Resources for Students with no Programming Background

In this module, we will learn how to “programmatically” access PubChem and other public databases. Cheminformaticians often write computer programs to process a large amount of chemical data and to automate some tasks that are routinely performed. Therefore, computer programming is an essential skill for future cheminformaticians. While this module does not require prior knowledge of computer programming, it is highly recommended that students with no or little programming background build some basic computer programming skills. There are many online resources that help you learn computer programming and below are some of them.

- w3schools.com (https://www.w3schools.com)
- Tutorialspoint.com (https://www.tutorialspoint.com/index.htm)
- Code.org (https://code.org)
- Code Academy (https://www.codecademy.com)
- Exercism (http://exercism.io)
- Cloud9 (https://c9.io)
- Ideone (https://ideone.com)
- LearnPython.org (https://www.learnpython.org)
- Python Challenge (http://www.pythonchallenge.com)

In addition, an introductory material about Web APIs (Application Programming Interfaces) is available at the OLCC web site (as a Special Topic Module).

- **Programmatic Access to Data (by Jordi Cuadros)** (http://olcc.ccce.divched.org/Spring2017OLCCSpecialTopic-Programmatic%20Access%20to%20Data)

The present module focuses on accessing PubChem data, but many other public chemical information resources also provide programmatic access to their data. Some examples complied by the OLCC Cheminformatics Faculty are available at this URL:

- **Programmatic Access to Web-Based Chemical Information – Examples** (http://olcc.ccce.divched.org/Spring2017OLCCModule7TLO1)

Overview of Programmatic Access to PubChem

Currently PubChem[1,2,3] contains more than 235 million depositor-provided substance descriptions, 94 million unique chemical structures and 232 million biological test results from one million assays, covering more than 10 thousand unique protein target sequences. Many researchers in the biomedical science community have a great interest in programmatic access to this vast amount of data because it presents new opportunities for data-driven research in a “big data” era. PubChem provides several ways for programmatic access to its data, including:

- Entrez Utilities (also called E-Utilities or E-Utils)
- Power User Gateway (PUG)
- PUG-SOAP
- PUG-REST

E-Utilities, used for programmatic access to information contained in the Entrez system, are suited for accessing text or numeric-fielded data, they cannot deal with more complex types of data specific to PubChem, such as chemical structures and tabular bioactivity data. Thus, PubChem provides additional programmatic access routes specialized for PubChem data and analysis services: PUG, PUG-SOAP, and PUG-REST. While suitable for low-level programmatic access to PubChem, PUG exchanges data through a complex eXtended Markup Langauge (XML) schema that requires some expertise to use. For the sake of user-friendliness and for integration with a variety of third party tools, PubChem provides two easier-to-use web service access methods: PUG-SOAP, which uses the simple object access protocol (SOAP) and PUG-REST, which is a Representational State Transfer (REST)-style interface. An overview of these programmatic access routes to PubChem is given in the following paper:[4]

- **PUG-SOAP and PUG-REST: Web Services for Programmatic Access to Chemical Information in PubChem**
  Kim *et al.*, *Nucleic Acids Res.* **2015**, 43(W1), W605-W611.
  (http://dx.doi.org/10.1093/nar/gkv396).

In this module, we will learn how to access PubChem using PUG-REST, because it is the simplest to use and learn. More in-depth information on programmatic access to PubChem is described in these documents:

- **PUG-REST**
  - PUG-REST Help (http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html)
  - PUG-REST Tutorial (http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html)
  - Programmatic Retrieval of Small Molecule Information from PubChem Using PUG-REST (in press).
    [Available to logged-in students at bottom of this module.]
- **E-Utilities**
  - Entrez Programming Utilities Help (http://www.ncbi.nlm.nih.gov/books/NBK25501/)
  - E-Utilities Quick Start (http://www.ncbi.nlm.nih.gov/books/NBK25500/)
- **PUG**
  - PUG Help (https://pubchem.ncbi.nlm.nih.gov/pug/pughelp.html)
- **PUG-SOAP**
  - PUG-SOAP Help (http://pubchem.ncbi.nlm.nih.gov/pug_soap/pug_soap_help.html
  - PUG-SOAP Client Help (http://pubchem.ncbi.nlm.nih.gov/pug_soap/client_help.html)
  - PUG-SOAP Web Service Reference (http://pubchem.ncbi.nlm.nih.gov/pug_soap/PUG_SOAP.html)

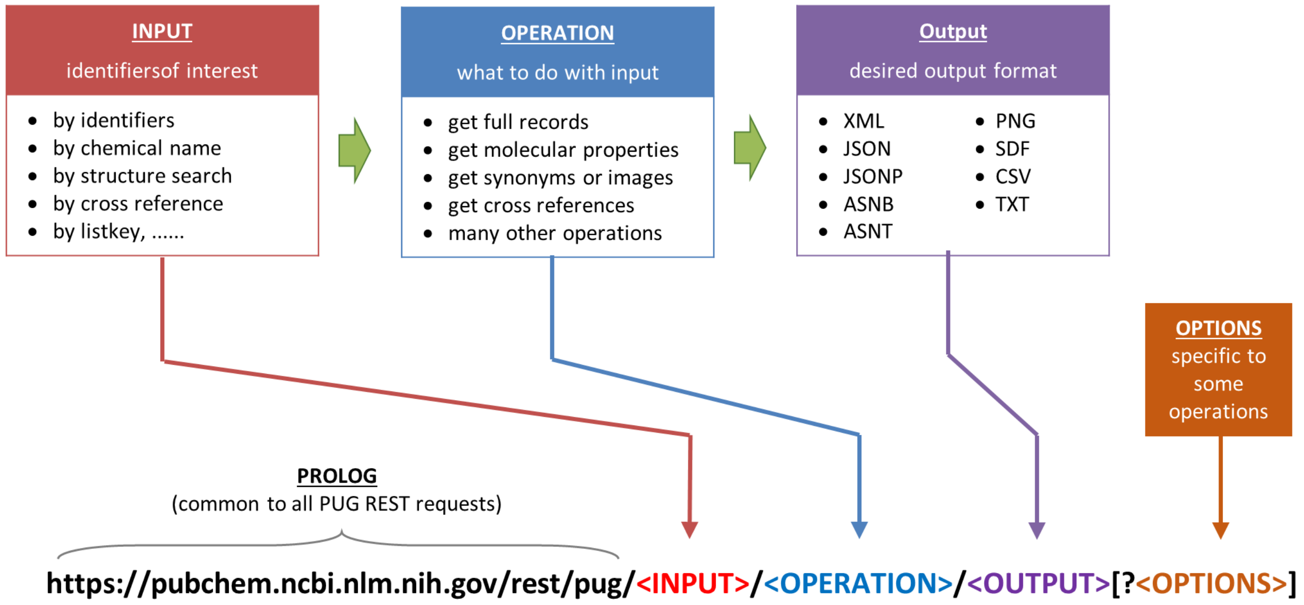
PUG-REST

Concepts and Syntax of PUG-REST requests

PUG-REST is the simplest to use and learn among the existing programmatic access methods to PubChem. Importantly, because information necessary for a PUG-REST request can be encoded into a single Uniform Resource Locator (URL) that can be written by hand without programming expertise. Conceptually, a web service request from the user to PubChem requires three pieces of information:

- **input**: a list of PubChem identifiers of interest (e.g., CID, AID, SID).
- **operation**: what to do with the input identifiers.
- **output**: the format of the output from the operation.

In PUG-REST, these three pieces of information are encoded into an URL in the following format:



INPUT
identifiersof interest
- by identifiers
- by chemical name
- by structure search
- by cross reference
- by listkey, ……

OPERATION
what to do with input
- get full records
- get molecular properties
- get synonyms or images
- get cross references
- many other operations

Output
desired output format
- XML
- JSON
- JSONP
- ASNB
- ASNT
- PNG
- SDF
- CSV
- TXT

OPTIONS
specific to some operations

PROLOG
(common to all PUG REST requests)

https://pubchem.ncbi.nlm.nih.gov/rest/pug/<INPUT>/<OPERATION>/<OUTPUT>[?<OPTIONS>]

[Example]

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/853/record/XML?record_type=3d

Some tasks require additional pieces of information that do not fit into the three-part PUG-REST URL. They should be provided as a list of ‘&’-separated option name and option value pairs, following the question mark (“?”) appended at the end of the request URL. Some examples are presented in next section, but there are much more things that users can do through PUG-REST. To get more detailed information on PUG-REST, read the following four articles:

- **PUG-SOAP and PUG-REST: Web Services for Programmatic Access to Chemical Information in PubChem**
  Kim *et al.*, *Nucleic Acids Res.* **2015**, 43(W1), W605-W611.
  (http://dx.doi.org/10.1093/nar/gkv396).
- **Programmatic Retrieval of Small Molecule Information from PubChem Using PUG-REST** (in press).
  [Available to logged-in students at bottom of this module.]
- **PUG-REST Help** (http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html)
- **PUG-REST Tutorial** (http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html)

PUG-REST Examples

1. Getting the full record of a compound record.

One of the most common tasks requested through PUG-REST is to retrieve all computed properties for a given chemical or chemicals. The following example URLs retrieve the full record of acetone in an XML format.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/180/record/XML
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/acetone/record/XML
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/inchikey/CSCPPACGZOOCGX-UHFFFAOYSA-N/record/XML
d. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/inchikey/CSCPPACGZOOCGX-UHFFFAOYSA-N/record/XML?record_type=2d
e. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/inchikey/CSCPPACGZOOCGX-UHFFFAOYSA-N/record/XML?record_type=3d

In the above examples, the input compound is specified by CID (CID 180), name (acetone), and InChiKey (CSCPPACGZOOCGX-UHFFFAOYSA-N). The input identifiers can also be specified by SMILES or InChI strings, although special care needs to be taken because these identifiers contain special characters (such as “/”) that cause conflicts with the URL syntax.[4] In addition, the hits returned from a structure search (e.g., identity/similarity search or sub/superstructure search) can be used as the input identifiers for a PUG-REST request. [It is highly recommended to use the synchronous “fast” inputs for structure searches (e.g., fastidentity, fastsimilarity_2d, fastsimilarity_3d, fastsubstructure, fastsuperstructure, and fastformula). Read the "Asynchronous operations in PUG-REST" section of this paper.]

The keyword "record", followed by the input identifier, invokes the full record retrieval of the input compound. By default, the "record_type" option is set to "2d", meaning that the information derived from the 2-D structure of the input compound will be returned. By setting this option to "3d", one can get the information derived from the 3-D structure. Therefore, examples (a) through (d) returns 2-D information, and example (e) returns 3-D information.

2. Getting the 2-D and 3-D structure image of a compound

Below are examples of the molecular structure image retrieval through PUG-REST.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/180/record/PNG
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/180/record/PNG?record_type=2d
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/180/record/PNG?record_type=3d

When the output file format for full record retrieval is set to "PNG", the image of the input compound is retrieved. The 3-D image of the input compound can be retrieved by setting the "record_type" option to "3d". When a list of compounds are specified as the input, the image of only the first compound on the list will be returned.

3. Getting molecular properties of a set of compounds

One may download molecular properties for a set of compounds, as in the following example:

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/887,702,1031,263/property/MolecularFormula,MolecularWeight,CanonicalSMILES,HeavyAtomCount,XLOGP/CSV

In this example, the molecular formula, molecular weight, canonical SMILES, Heavy Atom Count, and XLOGP value for 4 compounds are retrieved in a CSV format, which can be read in a spread sheet program like Excel and Google Sheet. A list of the molecular properties available through PUG-REST can be found in the PUG-REST specification document:

https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html#_Toc458584223

4. Getting a list of CIDs whose synonym is or contains "atorvastatin"

Through PUG-REST, one can perform a search by synonym.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/atorvastatin/cids/txt
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/atorvastatin/cids/txt?name_type=complete
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/atorvastatin/cids/txt?name_type=word

By default, the "name_type" option is set to "complete", meaning that only those whose synonym completely matches the input chemical name will be returned. One can perform partial synonym matching, by setting this option to "word". These search options are conceptually equivalent to an Entrez search with Entrez indices "[completesynonym]" and "[synonym]".

5. Getting a list of CIDs for compounds identical to a query compound

Below are examples of PUG-REST request URLs for identity search.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastidentity/cid/4594/cids/TXT
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastidentity/cid/4594/cids/TXT?identity_type=same_stereo_isotope
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastidentity/cid/4594/cids/TXT?identity_type=same_connectivity
d. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastidentity/cid/4594/cids/TXT?identity_type=same_isotope
e. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastidentity/cid/4594/cids/TXT?identity_type=same_stereo

Note that various "contexts" of identity can be selected using the "identity_type" option. By default, this option is set to "same_stereo_isotope", which returns compounds identical to the query compound in both stereochemistry and isotopism. When it is set to "same_connectivity", the identity search will returns molecules with the same connectivity (ignoring stereochemistry and isotopism).

6. Getting a list of CIDs for compounds with a given substructure

The following examples perform substructure searches through PUG-REST.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastsubstructure/cid/6857523/cids/TXT

b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastsubstructure/cid/6857523/cids/TXT?StripHydrogen=false
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastsubstructure/cid/6857523/cids/TXT?StripHydrogen=true

By default, the substructure search will keep explicit hydrogen atoms in the query substructure. By setting the StripHydrogen parameter to "true", one can strip off hydrogen atoms from the query substructure before performing a substructure search.

7. Getting a list of CIDs for compounds with a given molecular formula

The following examples show how to perform a molecular formula search through PUG-REST.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastformula/C6H12O6/cids/TXT
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/fastformula/C6H12O6/cids/TXT?AllowOtherElements=true

By default, the search results from a molecular formula search will exactly match the entered stoichiometry. One may allow other elements in the returned results by using the "AllowOtherElements=true" option.

8. Getting a list of AIDs for assays that target a given protein.

One can retrieve assays that are performed against a particular target, which can be specified by gene symbol, NCBI's Gene ID, or NCBI's global identifier (gi) (for protein sequences).

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/target/genesymbol/HMGCR/aids/TXT
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/target/geneid/3156/aids/TXT
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/target/gi/118573791/aids/TXT

9. Getting a list of compounds tested in an assay.

The following examples show how to get a list of compounds tested in a given assay.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/1053202/cids/XML
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/1053202/cids/XML?cids_type=all
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/1053202/cids/XML?cids_type=active
d. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/1053202/cids/XML?cids_type=inactive

Using the "cids_type" option, one can download all compounds tested in an assay or only those compounds tested to be active (or inactive).

10. Getting bioactivity data determined in an assay.

It is possible to download the bioactivity data for a given assay through PUG-REST.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/1053202/record/CSV
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/1053202/record/CSV?cid=823611,1270560,2104765

It is also possible to download the bioactivity data for a particular compound or compounds tested in the input assay by providing the comma-separated list of CIDs of interest after the question mark.

11. Getting a list of assays in which a compound was tested.

Through PUG-REST, one can get a list of assays in which a given compound was tested. Below are some examples.

a. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/5329102/aids/XML
b. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/5329102/aids/XML?aids_type=all
c. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/5329102/aids/XML?aids_type=active
d. https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/5329102/aids/XML?aids_type=inactive

By adjusting the "aids_type" parameter to "active" (or "inactive), one can retrieve assays in which the compound is tested to be active (or inactive). By default, any assays in which the input compound is tested are retrieved.

Accessing PubChem Data From a Spread Sheet Program

Through PUG-REST, one can access PubChem data from a spread sheet software, such as Google Sheet or MicroSoft Excel. Below is an example Google Sheet file that shows how to auto-populate data from PubChem into the spread sheet.

https://docs.google.com/spreadsheets/d/12-UkxgCxNVp2Ceim47TRffhqquLi2TFPKj3PJpii7bY/edit?usp=sharing

In this example, the function **IMAGE()** is used to retrieve the molecular images in the PNG format, and **IMPORTXML()**is used to get the record name and some molecular properties. While the **IMPORTXML()** function is used to get data in an XML format, the **IMPORTDATA()** function should be used to get PubChem data in CSV or TXT format. Note that the PUG-REST request URL used in the **IMPORTXML()** contains a comma-separated list of the input CIDs to retrieve data for all CIDs at a single request. While it is possible to make ten PUG-REST requests (one for each CID) to get the same data, it is highly recommended that users should minimize the number of requests. On the other hand, the image retrieval through PUG-REST supports only one input CID at a time, and therefore ten PUG-REST requests are made to get the images for all CIDs.

It is also possible to use MicroSoft Excel to do the same task as done in the above Google Sheet example. The **WEBSERVICE()** function in MicroSoft Excel 2013/2016 is equivalent to **IMPORTXML()** and **IMPORTDATA()** in Google Sheet. It returns the data retrieved from a web service request (in both XML and CSV/TXT). However, MicroSoft Excel does not have a built-in function equivalent to **IMAGE()** in Google Sheet, one needs to write a script for the image retrieval in VBA (Visual Basic for Applications), which is beyond the scope of this module.

Accessing PubChem Data From a Script

This section introduces how to make a PUG-REST request from a script, with example Python scripts. These examples are simple and short enough for students with no programming background to understand. You can modify and run these scripts on your web browser. While the examples are written in Python, it is possible to do the same tasks using other programming languages.

Example 1: Search by chemical name (https://trinket.io/library/trinkets/f009cf46ee)

This example searches PubChem for compounds whose name is “atorvastatin” and retrieve their full record. All lines beginning with the “#” characters are comments, which are ignored during the execution of a script. These comments are usually used to provide human-readable explanations about the script.

Without the comments and blank lines, only the remaining three lines are actually executed. In line 8, the “urllib.request” module is imported, which contains the definition of the function “urllib.request.urlopen()” in line 11. This function makes a web service request to the URL provided within the parentheses. The returned result is stored in the “request” object. In line 14, the returned data are read from the object and printed out.

Example 2: Molecular property retrieval (https://trinket.io/library/trinkets/a42e5b43e8)

This example illustrates how to retrieve molecular properties of a list of CIDs. In this example, the three pieces of information required for a PUG-REST request (that is, the input, operation, and output) are stored in respective variables (Lines 12-14). These variables are used to construct a PUG-REST request URL, which is stored in a variable called “url” (Line 17). In Line 20, this “url” variable is used as an argument in the function “urllib.request.urlopen()”, which makes the PUG-REST request.

Note that Example 2 uses additional variables to store different parts of the PUG-REST request URL, making it longer than Example 1. It is possible to reduce Example 2 into a three-line script without using these additional variables, by directly specifying the PUG-REST URL in the parentheses after “urllib.request.urlopen” as shown in Example 1.

Example 3: 2-D similarity search using CID queries (https://trinket.io/library/trinkets/490a1841a3)

In this example, 2-D similarity search is repeatedly performed using a list of CIDs as queries. The query compounds are defined in Line 17. Because 2-D similarity search can take only one query compound at a time, it should be repeated using a “for” loop as shown in Lines 20 through 28. In this “for” loop, each CID in “queries” (defined in Line 17) is assigned to the variable “mycid”, which is used to construct a PUG-REST request URL in Lines 24-25.

Note that the input CID is converted into a string [using the str() function at line 24] before it is concatenated with the other parts of the URL. This conversion is necessary because the input identifiers are provided as numbers at Line 17. If they are provided as strings (with each of them enclosed in quotes), they could be directly used in the URL.

Example 4: 2-D similarity search using isomeric SMILES queries (https://trinket.io/library/trinkets/3437cdefb0)

This example also performs 2-D similarity search using a list of SMILES strings as queries. An important difference between Examples 3 and 4 is the way in which the input identifiers are encoded in the PUG-REST request URL. Some special characters used as input identifiers are also reserved for the URL syntax, causing issues when directly encoded in an URL path. An example is the “/” (forward slash) character used in isomeric SMILES and InChI strings. To avoid conflicts with URL syntax, these line notations need to be included in the options part of the URL (after the “?” mark) (as in Lines 27-28).

References

(1) Kim, S.; Thiessen, P. A.; Bolton, E. E.; Chen, J.; Fu, G.; Gindulyte, A.; Han, L. Y.; He, J. E.; He, S. Q.; Shoemaker, B. A.; Wang, J. Y.; Yu, B.; Zhang, J.; Bryant, S. H. *Nucleic Acids Res.* **2016**, *44*, D1202.

(2) Wang, Y.; Bryant, S. H.; Cheng, T.; Wang, J.; Gindulyte, A.; Shoemaker, B. A.; Thiessen, P. A.; He, S.; Zhang, J. *Nucleic Acids Res.* **2017**, *45*, D955.

(3) Kim, S. *Expert Opinion on Drug Discovery* **2016**, *11*, 843.

(4) Kim, S.; Thiessen, P. A.; Bolton, E. E.; Bryant, S. H. *Nucleic Acids Res.* **2015**, *43*, W605.

4.6: Python Assignments

Structure Search

Downloadable Files

lecture05_Structure_Search.ipynb

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

- Requests
- RDKit
- time
- io

NOTE: This is a 2 week assignment

Objectives

- Learn various types of structure searches including identity search, similarity search, substructure and super structure searches.
- Learn the optional parameters available for each search type.

Using PUG-REST, one can perform various types of structure searches (https://bit.ly/2lPznCo), including:

- identity search
- similarity search
- super/substructure search
- molecular formula search

As explained in a PubChem paper (https://bit.ly/2kirxky), whereas structure search can be performed in either an 'asynchronous' or 'synchronous' way, it is highly recommended to use the synchronous approach.
The synchronous searches are invoked by using the keywords prefixed with ‘fast’, such as **fastidenity**, **fastsimilarity_2d**, **fastsimilarity_3d**, **fastsubstructure**, **fastsuperstructure**, and **fastformula**.

Note: To use the python code in this lesson plan, RDKit must be installed on the system.

Many users can simply run the following code to install RDKit.

conda install -c rdkit rdkit

Access to the full installation instructions can be found at the following link. https://www.rdkit.org/docs/Install.html

PUG-REST allows you to search the PubChem Compound database for molecules identical to the query molecule. PubChem's identity search supports different contexts of chemical identity, which the user can specify using the optional parameter, "identity_type". Here are some commonly-used chemical identity contexts.

- **same_connectivity**: returns compounds with the same atom connectivity as the query molecule, ignoring stereochemistry and isotope information.
- **same_isotope**: returns compounds with the same isotopes (as well as the same atom connectivity) as the query molecule. Stereochemistry will be ignored.
- **same_stereo**: returns compounds with the same stereochemistry (as well as the same atom connectivity) as the query molecule. Isotope information will be ignored.

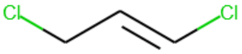
- **same_stereo_isotope**: returns compounds with the same stereochemistry AND isotope information (as well as the same atom connectivity). This is the default.

The following code cell demonstrates how these different contexts of chemical sameness affects identity search in PubChem.

In [1]:

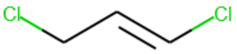
```
import requests
import time
import io

from rdkit import Chem
from rdkit.Chem import Draw

prolog = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

mydata = { 'smiles' : 'C(/C=C/Cl)Cl' }
options = [ 'same_stereo_isotope', # This is the default
            'same_stereo',
            'same_isotope',
            'same_connectivity']

for myoption in ( options ) :

    print("#### Identity_type:", myoption)

    url = prolog + '/compound/fastidentity/smiles/property/isomericsmiles/csv?identity_type=' + myoption
    res = requests.post(url, data=mydata)

    mycids = []
    mysmiles = []

    file = io.StringIO(res.text)
    file.readline()                 # Skip the first line (column heads)

    for line in file :

        ( cid_tmp, smiles_tmp ) = line.rstrip().split(',')
        print(cid_tmp, smiles_tmp)

        mycids.append( cid_tmp )
        mysmiles.append( smiles_tmp.replace('"',"") )

    mols = []

    for x in mysmiles :

        mol = Chem.MolFromSmiles(x)
        Chem.FindPotentialStereoBonds(mol)     # Identify potential stereo bonds!
        mols.append(mol)

    img = Draw.MolsToGridImage(mols, molsPerRow=3, subImgSize=(200,200), legends=mycids)
    display(img)

    time.sleep(0.2)
```

```
#### Identity_type: same_stereo_isotope
24726 "C(/C=C/Cl)Cl"
```

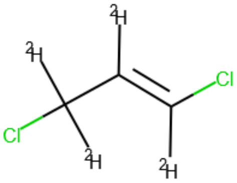


24726

```
#### Identity_type: same_stereo
24726 "C(/C=C/Cl)Cl"
102602172 "[2H]/C(=C(/[2H])\Cl)/C([2H])([2H])Cl"
```

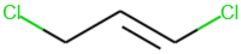


24726

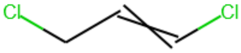


102602172

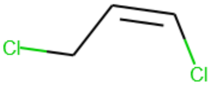
```
#### Identity_type: same_isotope
24726 "C(/C=C/Cl)Cl"
24883 "C(C=CCl)Cl"
5280970 "C(/C=C\Cl)Cl"
```



24726

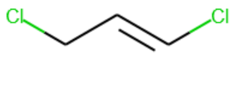


24883

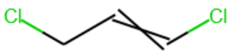


5280970

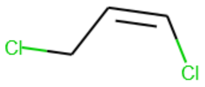
```
#### Identity_type: same_connectivity
24726 "C(/C=C/Cl)Cl"
24883 "C(C=CCl)Cl"
5280970 "C(/C=C\Cl)Cl"
102602172 "[2H]/C(=C(/[2H])\Cl)/C([2H])([2H])Cl"
131875718 "[2H]C(=C([2H])Cl)C([2H])([2H])Cl"
```



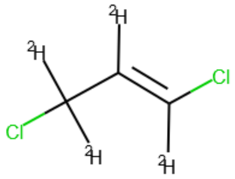
24726



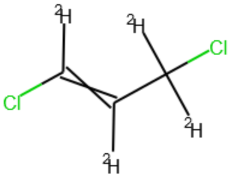
24883



5280970



102602172



131875718

Exercise 1a: Find compounds that has the same atom connectivity and isotope information as the query molecule.

In [2]:

```
query = "CC1=CN=C(C(=C1OC)C)C[S@](=O)C2=NC3=C(N2)C=C(C=C3)OC"
```

For each compound returned from the search, retrieve the following information.

- CID
- Isomeric SMILES string
- chemical synonyms (for simplicity, print only the five synonyms that first occur in the name list retrieved for each compound)
- Structure image

In [3]:

```
# Write your code in this cell.
```

Similarity search

PubChem supports 2-dimensional (2-D) and 3-dimensional (3-D) similarity searches. Because molecular similarity is not a measurable physical observable but a subjective concept, many approaches have been developed to evaluate it. Detailed discussion on how PubChem quantifyies molecular similarity, read the following LibreTexts page:

Searching PubChem Using a Non-Textual Query (https://bit.ly/2lPznCo)

The code cell below demonstrates how to perform 2-D and 3-D similarity searches.

In [4]:

```
mydata = { 'smiles' : "C1COCC(=O)N1C2=CC=C(C=C2)N3C[C@@H]
(OC3=O)CNC(=O)C4=CC=C(S4)Cl" }
url = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=99"
res = requests.post(url,data=mydata)
cids = res.text.split()

print("# Number of CIDs:", len(cids))
print(cids)
```

```
# Number of CIDs: 29
['9875401', '6433119', '11524901', '68152323', '25190310', '25164166', '123868009', '
```

It is worth mentioning that the parameter name "Threshold" is **case-sensitive**. If "threshold" is used (rather than "Threshold"), it will be ignored and the default value (0.90) will be used for the parameter. [As a matter of fact, all optional parameter names in PUG-REST are case-sensitive.]

In [5]:

```
url1 = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=95"
url2 = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt?threshold=95"  # "threshold=95" is ignored.

res1 = requests.post(url1,data=mydata)
res2 = requests.post(url2,data=mydata)
cids1 = res1.text.split()
cids2 = res2.text.split()

print("# Number of CIDs:", len(cids1), "vs.", len(cids2))
```

```
# Number of CIDs: 165 vs. 763
```

It is possible to run 3-D similarity search using PUG-REST. However, because 3-D similarity search takes much longer than 2-D similarity search, it often exceeds the 30-second time limit and returns a time-out error, especially when the query molecule is big.

In addition, for 3-D similarity search, it is **not** possible to adjust the similarity threshold (that is, the optional "Threshold" parameter does not work). 3-D similarity search uses a shape-Tanimoto (ST) of >=0.80 and a color-Tanimoto (CT) of >=0.50 as a similarity threshold. Read the libreTexts page for more details (https://bit.ly/2lPznCo).

In [6]:

```
mydata = { 'smiles' : 'CC(=O)OC1=CC=CC=C1C(=O)O'}
url = prolog + "/compound/fastsimilarity_3d/smiles/cids/txt"
res = requests.post(url, data=mydata)
cids = res.text.split()
print(len(cids))
```

```
21424
```

Exercise 2a: Perform 2-D similarity search with the following query, using a threshold of 0.80 and find the macromolecule targets of the assays in which the returned compounds were tested. You will need to take these steps.

- Run 2-D similarity search using the SMILES string as a query (with Threshold=80).
- Retrieve the AIDs in which any of the returned CIDs was tested "active".
- Retrieve the gene symbols of the targets for the returned AIDs.

In [7]:

```
query='[C@@H]23C(=O)[C@H](N)C(C)[C@H](CCC1=COC=C1)[C@@]2(C)CCCC3(C)C'
```

In [8]:

```
# Write your code in this cell.
```

Substructure/Superstructure search

When a chemical structure occurs as a part of a bigger chemical structure, the former is called a substructure and the latter is referred to as a superstructure (https://bit.ly/2lPznCo). PUG-REST supports both substructure and superstructure searches. For example, below is an example for substructure search using the core structure of antibiotic drugs called cephalosporins as a query (https://en.Wikipedia.org/wiki/Cephalosporin).

In [9]:

```
query = 'C12(SCC(=C(N1C([C@H]2NC(=O)[*])=O)C(=O)O[H])[*])[H]'

mydata = { 'smiles' : query }
url = prolog + "/compound/fastsubstructure/smiles/cids/txt?Stereo=exact"
res = requests.post(url, data=mydata)
cids = res.text.split()

print("# Number of CIDs:", len(cids))
#print(cids)
```

```
# Number of CIDs: 21810
```

An important thing to remember about substructure search is that, if the query structure is not specific enough (that is, not big enough), it will return too many hits for the PubChem server can handle. For example, if you perform substructure search using the "C-C" as a query, it will give you an error, because PubChem has ~96 million (organic) compounds with more than two carbon atoms and most of them will have the "C-C" unit. Therefore, if you get an "time-out" error while doing substructure search, consider providing more specific structure as an input query.

Exercise 3a: Below is the SMILES string for a HCV (Hepatitis C Virus) drug (Sovaldi). Perform substructure search using this SMILES string as a query, identify compounds that are mentioned in patent documents, and create a list of the patent documents that mentioning them.

- Use the default options for substructure search.
- Use the "XRefs" operation to retrieve Patent IDs associated with the returned compounds.
- For simplicity, ignore the CID-Patent ID mapping. (That is, no need to track which CID is associated with which patent document.)

In [10]:

```
query="C[C@@H](C(=O)OC(C)C)N[P@](=O)(OC[C@@H]1[C@H]([C@@]([C@@H](O1)N2C=CC(=O)NC2=O)(C)F)O)OC3=CC=CC=C3"
```

In [11]:

```
# Write your code in this cell.
```

Molecular formula search

Strictly speaking, molecular formula search is not structure search, but its PUG-REST request URL is constructed in a similar way to structure searches like identity, similarity, and substructure/superstructure searches.

In [12]:

```
query = 'C22H28FN3O6S'     # Molecular formula for Crestor (Rosuvastatin: CID 446157)

url = prolog + "/compound/fastformula/"+ query + "/cids/txt"
res = requests.get(url)
cids = res.text.split()
print("# Number of CIDs:", len(cids))
#print(cids)
```

```
# Number of CIDs: 179
```

It is possible to allow other elements to be present in addition to those specified by the query formula, as shown in the following example.

In [13]:

```
url = prolog + "/compound/fastformula/"+ query + "/cids/txt?AllowOtherElements=true"
res = requests.get(url)
cids = res.text.split()
print("# Number of CIDs:", len(cids))
#print(cids)
```

```
# Number of CIDs: 200
```

Exercise 4a: The general molecular formula for alcohols is CnH(2n+2)OCnH(2n+2)O [for example, CH4O (methanol), C2H6O (ethanol), C3H8O (propanol), etc]. Run molecular formula search using this general formula for n=1 through 20 and retrieve the XLogP values of the returned compounds for each value of n. Print the minimum and maximum XLogP values for each n value.

In [14]:

```
# Write your code in this cell.
```

4.7: R Assignment

Structure Search

S. Kim, J. Cuadros

Objectives

- Learn various types of structure searches including identity search, similarity search, substructure and super structure searches.
- Learn the optional parameters available for each search type.

Using PUG-REST, one can perform various types of structure searches (https://bit.ly/2lPznCo), including: - identity search - similarity search - super/substructure search - molecular formula search

As explained in a PubChem paper (https://bit.ly/2kirxky), whereas structure search can be performed in either an ‘asynchronous’ or ‘synchronous’ way, it is highly recommended to use the synchronous approach.

The synchronous searches are invoked by using the keywords prefixed with ‘fast’, such as fastidenity, fastsimilarity_2d, fastsimilarity_3d, fastsubstructure, fastsuperstructure, and fastformula.

In this task, we will use some cheminformatics packages to ease some processes. In R, some options are rcdk , ChemmineR and ChemmineOB . In Python, a useful package is RDKit ; in R, we’ll make use of it online version, the Beaker API of ChEMBL (https://chembl.gitbook.io/chembl-interface-documentation/web-services).

1. Identity Search

PUG-REST allows you to search the PubChem Compound database for molecules identical to the query molecule. PubChem’s identity search supports different contexts of chemical identity, which the user can specify using the optional parameter, “identity_type”. Here are some commonly-used chemical identity contexts. - same_connectivity: returns compounds with the same atom connectivity as the query molecule, ignoring stereochemistry and isotope information. - same_isotope: returns compounds with the same isotopes (as well as the same atom connectivity) as the query molecule. Stereochemistry will be ignored. - same_stereo: returns compounds with the same stereochemistry (as well as the same atom connectivity) as the query molecule. Isotope information will be ignored. - same_stereo_isotope: returns compounds with the same stereochemistry AND isotope information (as well as the same atom connectivity). This is the default.

The following code cell demonstrates how these different contexts of chemical sameness affects identity search in PubChem.

```
if(!require("httr")) {
 install.packages(("httr"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("httr")
}
if(!require("jsonlite")) {
 install.packages(("jsonlite"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
library("jsonlite")
}
if(!require("png")) {
 install.packages(("png"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("png")
}
if(!require("grid")) {
 install.packages(("grid"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("grid")
}
if(!require("gridExtra")) {
 install.packages(("gridExtra"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("gridExtra")
}
```

```
prolog <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
smiles <- "C(/C=C/Cl)Cl"
options <- c('same_stereo_isotope',
 'same_stereo',
 'same_isotope',
 'same_connectivity') # same_stereo_isotope is the default

for(opt in options) {
 print(paste ("#### Identity_type:", opt))
 url <- paste(prolog,
 "/compound/fastidentity/smiles/",
 "property/isomericsmiles/csv?smiles=",
 URLencode(smiles,reserved = T),
 "&identity_type=",
 opt, sep="")
 dfChem <- read.table(url, sep=",", header=T)
 print(dfChem)

 url_img <- paste("https://www.ebi.ac.uk/chembl/api/utils/smiles2image",
 "?size=300&engine=rdkit",sep="")
 res <- POST(url_img,
 body=list(smiles=paste(dfChem[,2],collapse="\n")))
 img <- readPNG(res$content, native=TRUE)
 grid.arrange(rasterGrob(img))

 Sys.sleep(.5)
}
```

Exercise 1a: Find compounds that has the same atom connectivity and isotope information as the query molecule. query <- "CC1=CN=C(C(=C1OC)C)CS@C2=NC3=C(N2)C=C(C=C3)OC" For each compound returned from the search, retrieve the following information.

- CID
- Isomeric SMILES string
- chemical synonyms (for simplicity, print only the five synonyms that first occur in the name list retrieved for each compound)
- Structure image

```
# Write your code here
```

2. Similarity search

PubChem supports 2-dimensional (2-D) and 3-dimensional (3-D) similarity searches. Because molecular similarity is not a measurable physical observable but a subjective concept, many approaches have been developed to evaluate it. Detailed discussion on how PubChem quantifyies molecular similarity, read the following LibreTexts page:

Searching PubChem Using a Non-Textual Query (https://bit.ly/2lPznCo)

The code cell below demonstrates how to perform 2-D and 3-D similarity searches.

```
mydata <- list(smiles="C1COCC(=O)N1C2=CC=C(C=C2)N3C[C@@H](OC3=O)CNC(=O)C4=CC=C(S4)Cl"
url <- paste(prolog,
 "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=99",
 sep="")
res <- POST(url,body=mydata)
cids <- unlist(strsplit(rawToChar(res$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids)))
```

```
print(cids)
```

IMPORTANT: It is worth mentioning that the parameter name “Threshold” is case-sensitive. If “threshold” is used (rather than “Threshold”), it will be ignored and the default value (0.90) will be used for the parameter. [As a matter of fact, all optional parameter names in PUG-REST are case-sensitive.]

```
 url1 <- paste(prolog, "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=95", se
url2 <- paste(prolog, "/compound/fastsimilarity_2d/smiles/cids/txt?threshold=95", sep
# "threshold=95" is ignored.
```

```
res1 <- POST(url1,body=mydata)
res2 <- POST(url2,body=mydata)
cids1 <- unlist(strsplit(rawToChar(res1$content),"\n",fixed=T))
cids2 <- unlist(strsplit(rawToChar(res2$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids1), "vs.", length(cids2)))
```

It is possible to run 3-D similarity search using PUG-REST. However, because 3-D similarity search takes much longer than 2-D similarity search, it often exceeds the 30-second time limit and returns a time-out error, especially when the query molecule is big.

In addition, for 3-D similarity search, it is not possible to adjust the similarity threshold (that is, the optional “Threshold” parameter does not work). 3-D similarity search uses a shape-Tanimoto (ST) of >=0.80 and a color-Tanimoto (CT) of >=0.50 as a similarity threshold. Read the libreTexts page for more details (https://bit.ly/2lPznCo).

```
mydata <- list(smiles="CC(=O)OC1=CC=CC=C1C(=O)O")
url <- paste(prolog,
 "/compound/fastsimilarity_3d/smiles/cids/txt",
 sep="")
res <- POST(url,body=mydata)
cids <- unlist(strsplit(rawToChar(res$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids)))
```

Exercise 2a: Perform 2-D similarity search with the following query, using a threshold of 0.80 and find the macromolecule targets of the assays in which the returned compounds were tested. You will need to take these steps.

- Run 2-D similarity search using the SMILES string as a query (with Threshold=80).
- Retrieve the AIDs in which any of the returned CIDs was tested “active”.
- Retrieve the gene symbols of the targets for the returned AIDs.

```
query <- "[C@@H]23C(=O)[C@H](N)C(C)[C@H](CCC1=COC=C1)[C@@]2(C)CCCC3(C)C"
```

```
 # Write your code here
```

3. Substructure/Superstructure search

When a chemical structure occurs as a part of a bigger chemical structure, the former is called a substructure and the latter is referred to as a superstructure (https://bit.ly/2lPznCo). PUG-REST supports both substructure and superstructure searches. For example, below is an example for substructure search using the core structure of antibiotic drugs called cephalosporins as a query (https://en.Wikipedia.org/wiki/Cephalosporin).

```
mydata <- list(smiles="C12(SCC(=C(N1C([C@H]2NC(=O)[*])=O)C(=O)O[H])[*])[H]")
url <- paste(prolog,
 "/compound/fastsubstructure/smiles/cids/txt?Stereo=exact",
 sep="")
res <- POST(url,body=mydata)
cids <- unlist(strsplit(rawToChar(res$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids)))
```

An important thing to remember about substructure search is that, if the query structure is not specific enough (that is, not big enough), it will return too many hits for the PubChem server can handle. For example, if you perform substructure search using the “C-C” as a query, it will give you an error, because PubChem has ~96 million (organic) compounds with more than two carbon atoms and most of them will have the “C-C” unit. Therefore, if you get an “time-out” error while doing substructure search, consider providing more specific structure as an input query.

Exercise 3a: Below is the SMILES string for a HCV (Hepatitis C Virus) drug (Sovaldi). Perform substructure search using this SMILES string as a query, identify compounds that are mentioned in patent documents, and create a list of the patent documents that mentioning them.

Use the default options for substructure search. Use the “XRefs” operation to retrieve Patent IDs associated with the returned compounds. For simplicity, ignore the CID-Patent ID mapping. (That is, no need to track which CID is associated with which patent document.)

```
query <- "C[C@@H](C(=O)OC(C)C)N[P@](=O)(OC[C@@H]1[C@H]([C@@]([C@@H](O1)N2C=CC(=O)NC2=
O)OC3=CC=CC3"
```

```
# Write your code here
```

4. Molecular formula search

Strictly speaking, molecular formula search is not structure search, but its PUG-REST request URL is constructed in a similar way to structure searches like identity, similarity, and substructure/superstructure searches.

```
query <- "C22H28FN3O6S" # Molecular formula for Crestor (Rosuvastatin: CID 446157)
url <- paste(prolog, "/compound/fastformula/",
 query, "/cids/txt", sep="")
```

```
cids <- readLines(url)
print(paste("# Number of CIDs:", length(cids)))
```

It is possible to allow other elements to be present in addition to those specified by the query formula, as shown in the following example.

```
query <- "C22H28FN3O6S" # Molecular formula for Crestor (Rosuvastatin: CID 446157)
url <- paste(prolog, "/compound/fastformula/",
 query, "/cids/txt?AllowOtherElements=true", sep="")
cids <- readLines(url)
print(paste("# Number of CIDs:", length(cids)))
```

Exercise 4a: The general molecular formula for alcohols is [for example, CH4O (methanol), C2H6O (ethanol), C3H8O (propanol), etc]. Run molecular formula search using this general formula for n=1 through 20 and retrieve the XLogP values of the returned compounds for each value of n. Print the minimum and maximum XLogP values for each n value.

```
 # Write your code here
```

4.8: Mathematica Assignment

CHAPTER OVERVIEW

5: Quantitative Structure Property Relationships

Hypothes.is Tag= f19OLCCc6

Note: Any annotation tagged **f19OLCCc6** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

This page is under construction, and will hold content for module 6 of the Fall 2019 Cheminformatics OLCC.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

Topic hierarchy

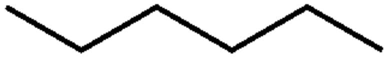
5.1: Quantitative Structure-Property Relationships

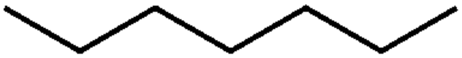
Molecular similarity (also called chemical similarity or chemical structure similarity) is a fundamental concept in cheminformatics, playing an important role in computational methods for predicting properties of chemical compounds as well as designing chemicals with desired properties. The underlying assumption in these computational methods is that structurally similar molecules are likely to have similar biological and physicochemical properties (commonly called the similarity principle). Molecular similarity is a straightforward and easy-to-understand concept, but there is no absolute, mathematical definition of molecular similarity that everyone agrees on. As a result, there are a virtually infinite number of molecular similarity methods, which quantify molecular similarity.

5.2: Similar-Structure, Similar-Property Principle

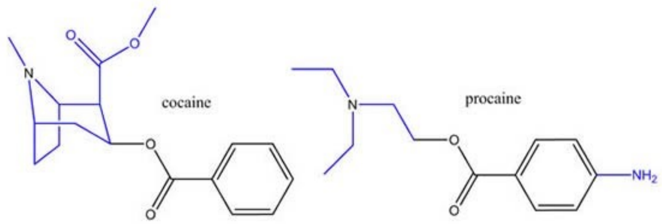
The Similar-Structure, Similar Property Principle is the fundamental assertion that similar molecules will also tend to exhibit similar properties. These properties can either be physical (e.g. boiling points) or biological (e.g. activity).

Example 1: Hexane and heptane should have similar boiling points and water solubility.





Example 2: Cocaine and procaine are both local anesthetics



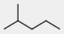
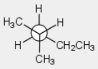
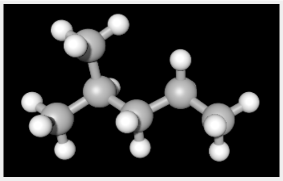
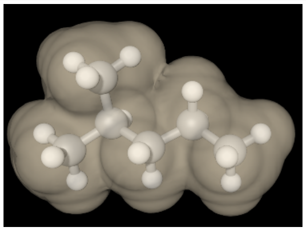
Quantitative Structure-Property Relationships (QSPR) and Quantitative Structure-Activity Relationships (QSAR) use statistical models to relate a set of predictor values to a response variable. Molecules are described using a set of descriptors, and then mathematical relationships can be developed to explain observed properties. In QSPR and QSAR physico-chemical properties of theoretical descriptors of chemicals are used to predict either a physical property or a biological outcome.

In either case, a set of known molecules is used to create a **training set** that a statistical model can be derived from. These molecules have known properties or activities. An outside **test set** is used to validate the model. The test set consists of other molecules with known properties that are excluded from the training set. After the model is validated, it can be used to predict properties or activities of molecules that are outside the previous sets. One caveat- new test molecules cannot be sufficiently different from the ones used in previous sets.

5.3: Molecular Descriptors

If we want to develop a computational model to predict properties, we need to be able to describe them in ways that can be tied to a biological or physical properties. There are many ways that we can represent organic molecules.

Example 1: Representing 2-methylpentane

2-methylpentane (IUPAC Name)	Isohexane (synonym)	$CH_3CH(CH_3)CH_2CH_2CH_3$(condensed structure)
 (Skeletal Line drawing)	 (Newman projection)	 (Ball and Stick Model)
 (Van der Waals surface)	CCCC(C)C (SMILES)	InChI=1S/C6H14/c1-4-5-6(2)3/h6H,4-5H2,1-3H3 (IUPAC InChI)
AFABGHUZZDYHJO-UHFFFAOYSA-N (IUPAC InChI Key)	C_6H_{14} (Molecular Formula)	86.18 g/mol (Molecular weight)

Each of these representations provides some clue about the nature of the molecule. Some representations can be inferred from others. For example, molecular weight can be calculated from the molecular formula, the SMILES, the condensed structure, or the skeletal drawing. Some representations tell you about the relative position of atoms in either 2D or 3D space. Some of these are inherently easy for humans to read and write, but present challenges for computer processing.

To make a reasonable prediction for any set of molecules, the physical or biological data must be related to the molecule through a series of descriptors. These descriptors can be structural, relating data about the relative position of atoms and types, or calculated data such as electron density using quantum chemical methods.

Descriptors can be classified by the following representations:

Molecular representation	examples
0D	Atom types, molecular weight, bond types
1D	Counts of atom types, counts of hydrogen bond donors or acceptors, number of rings, number of functional groups by type
2D	Mathematical representations by graph theory or calculated values such as lipophilicity or topological polar surface area
3D	Geometrical descriptors or polar surface area

In this chapter we will ignore 3D descriptors for now.

0D molecular descriptors

Molecules can be described in a data table by presence or absence or total number of atoms present. The total number of carbon, nitrogen, oxygen or halogen atoms can potentially adequately describe a molecule. For example, in organic chemistry much can be predicted about how a molecule will react or what physical properties it will have just by classifying it as an alkane, an alcohol or an aromatic molecule. Molecular weight in a series of like molecule can be useful to explain difference in boiling points even though that is not fundamental to the property.

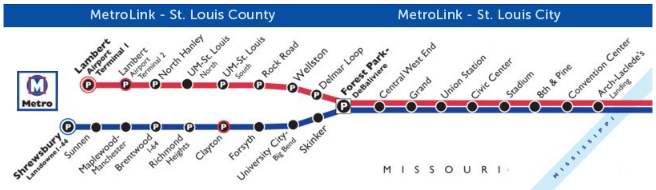
1D molecular descriptors

In addition to the types of atoms present, molecules can be further represented by bonding or bonding fragments. Molecules can be described by the number of sp^3, sp^2, or *sp* hybridized carbons present. These can also be included in a data table to indicate if they are bonded to an oxygen in the form of an alcohol or a carbonyl. Other functional groups can also be used to adequately describe a molecule by similarity. Indication of presence of C-N, C-S, C=N, or amide or ester functional groups can also tell a lot about how a molecule will interact with solvents or biological systems.

Topological vs topographical descriptors

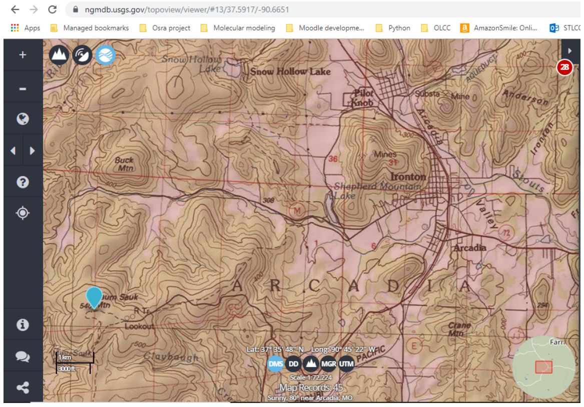
In cartography, maps are provided that tell you either the relative positions of features on a map (Topological) or the specific distances and elevations of features on a map. For example, public transportation maps usually only represent the stops on a bus or train line, but do not indicate the distance.

Example 2: Topological Map- Metrolink of St. Louis, Missouri https://www.metrostlouis.org/wp-content/uploads/2018/08/MK180468redblueline_update_CORTEX.jpg



A rider can know how many stops are between two points on the map, but not know that the distance between stops may be many miles.

Example 3: Topographical Map- https://ngmdb.usgs.gov/topoview/viewer/#13/37.5917/-90.6651



In this case, a person can know using the scale and the topological lines on the map, how far Taum Sauk Mountain is from Buck Mountain and the elevation change between the two.

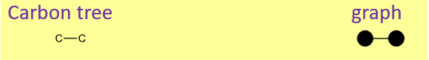
Molecules can also be described by topological (two-dimensional 2D) descriptors or topographical (geometrical, three-dimensional 3D) descriptors.

2D Molecular Descriptors

You were introduced to chemical graph theory in section 2.1 of this Libretext. Mathematical notations provide a method for describing chemical structures, and allow for computational processing of molecules in a data set. These are essentially 2D descriptors.

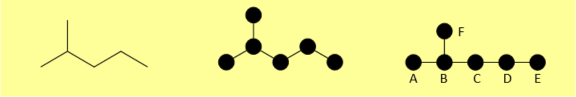
A graph is an abstract structure that contains nodes connected by edges. In representing molecules nodes are the atoms, and edges are the bonds. Hydrogen atoms are usually omitted and thus called “hydrogen depleted molecular graphs.”

Example: Ethane



Note that ethane is described here as a topological map- the connectivity of the molecule is given as relative locations, not exact locations (e.g. atomic size or bond length is excluded).

More complicated example- 2-methylpentane



Wiener Index

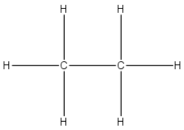
One of the first mathematical representations of chemical structure used for prediction of properties was developed in 1947 by Harold Weiner. It is defined at the sum of distances between any two carbon atoms (pairs of nodes) in the molecule. Mathematically it is represented as:

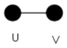
$$W(G) = \frac{1}{2} \sum_{u,v \in G} d(u,v)$$

Where G represents the total atoms in the molecule, u and v are individual carbon atoms and *d*(u,v) is the distance in bonds between any two carbon atoms in the shortest path between any two atoms.

In using this index, Weiner showed that the index value is closely correlated with the boiling point of a series of alkanes. Further work also showed that it correlated with other physical properties such as density, surface tension and viscosity.

To calculate the Wiener index for a molecule, for each pair of atoms in the structure, count the distance between atoms. Take the sum of all distances and divide by two. For example in the case of ethane, which only has two nodes:

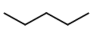


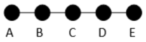


	u	**v**
u	0	1
v	1	0

$$W(G) = \frac{1}{2}(1+1) = \frac{1}{2}(2) = 1$$

A more complicated example is pentane:





Pentane has 5 nodes, and distances between each node are calculated and summed.

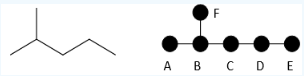
	A	**B**	**C**	**D**	**E**	**total**
A	0	1	2	3	4	10
B	1	0	1	2	3	7

C	2	1	0	1	2	6
D	3	2	1	0	1	7
E	4	3	2	1	0	10

$$W(G) = \frac{1}{2}(10+7+6+7+10) = \frac{1}{2}(40) = 20$$

✓ Activity 5.1

Determine the Wiener index for 2-methylpentane.



Click here for solution

Zagreb Indices

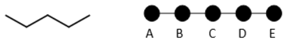
The first and second Zagreb indices (M_1 and M_2) are another set of classic vertex based descriptors developed in 1972 and 1975, respectively. They were called the Zagreb group indices as their authors were members of the "Rudjer Bošković" Institute in Zagreb, Croatia.

In these indices one counts the connections from each vertex (node, carbon). The first Zagreb index $M_1(G)$ is equal to the sum of squares of the degrees of the vertices, and the second Zagreb index $M_2(G)$ is equal to the sum of the products of the degrees of pairs of adjacent vertices of the underlying molecular graph G.

$$M_1 = \sum_{i=1}^{n} \delta_i^2$$

$$M_2 = \sum \delta_i \delta_j$$

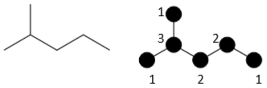
For pentane, each would be calculated as:



$M_1 = 1^2 + 2^2 + 2^2 + 2^2 + 1^2 = 1 + 4 + 4 + 4 + 1 = 14$

$M_2 = 1x2 + 2x2 + 2x2 + 2x1 = 2+4+4+2 = 12$

For 2-methylpentane, each would be calculated as:



$M_1 = 1^2 + 1^2 + 3^2 + 2^2 + 2^2 + 1^2 = 1 + 1 + 9 + 4 + 4 + 1 = 20$

$M_2 = 1x3 + 1x3 + 3x2 + 2x2 + 2x1 = 3+3+6+4+2 = 18$

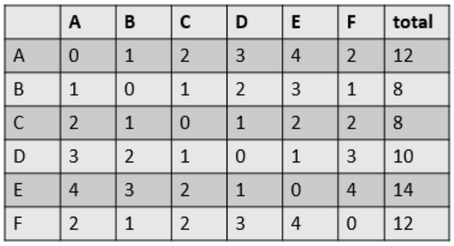
✓ Activity 5.2

Determine the Zagreb indices for 2,3-dimethylbutane.

Click here for solution

There are thousands of 2D descriptors that are frequently applied in modeling or predicting properties or biological functions. What is interesting is that these graphs are often descriptors that are reduced to a single value that can be used to make meaning of the physical world.

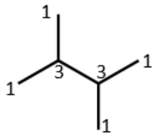
5.3.1: Exercise 5.1 solution



	A	B	C	D	E	F	total
A	0	1	2	3	4	2	12
B	1	0	1	2	3	1	8
C	2	1	0	1	2	2	8
D	3	2	1	0	1	3	10
E	4	3	2	1	0	4	14
F	2	1	2	3	4	0	12

$$W(G) = \frac{1}{2}(12 + 8 + 8 + 10 + 14 + 12) = \frac{1}{2}(64) = 32$$

5.3.2: Exercise 5.2 solution



Zagreb M_1	Zagreb M_2
$M_1 = 1^2 + 3^2 + 3^2 + 1^2 + 1^2 + 1^2$ $M_1 = 1 + 9 + 9 + 1 + 1 + 1$ $M_1 = 22$	$M_2 = 1*3+1*3+3*3+3*1+3*1$ $M_2 = 3+3+9+3+3$ $M_2 = 21$

5.4: Mathematica Assignment

5.5: Python Assignment

Quantitative Structure-Property Relationships

Downloadable Files

 QSPR.ipynb

 Excel_multiple_linear_regression_assignment.docx

 BP.csv

 102BP.csv

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.
- You will need to have the BP.csv and 102BP.csv files in the same directory as the QSPR.ipynb Jupyter notebook file.
- Your instructor may have you use Excel and multiple linear regression assignment in addition to this notebook.

Required Modules

- RDKit
- mordred (in your conda environment type- conda install -c mordred-descriptor mordred)
- pandas
- statsmodels (in your conda environment type- conda install statsmodels)

Objectives

- Use RDKit to calculate molecular descriptors
- Use molecular descriptors to create a predictive model for boiling points of alkanes.
- Use statsmodels to visualize data

Quantitative Structure-Property Relationships

Quantitative Structure-Property Relationships (QSPR) and Quantitative Structure-Activity Relationships (QSAR) use statistical models to relate a set of predictor values to a response variable. Molecules are described using a set of *descriptors*, and then mathematical relationships can be developed to explain observed properties. In QSPR and QSAR, physico-chemical properties of theoretical descriptors of chemicals are used to predict either a physical property or a biological outcome.

Molecular Descriptors

A molecular descriptor is “final result of a logical and mathematical procedure, which transforms chemical information encoded within a symbolic repre-sentation of a molecule into a useful number or the result of some standardized experiment” (Todeschini, R.; Consonni, V. *Molecular descriptors for chemoinformatics* **2009** Wiley-VCH, Weinheim). You are already familiar with descriptors such as molecular weight or number of heavy atoms and we have queried PubChem for data such as XLogP. We’ll examine just a few simple descriptors, but thousands have been developed for applications in QSPR.

Using rdkit and mordred to calculate descriptors

Clearly we have been using algorithms for calculating these indices. This is time consuming for an individual, but programs can be used to complete this much easier. We will use the rdkit and mordred python libraries to help us out.

In [1]:

```
from rdkit import Chem                              # imports the Chem module from rdki
from mordred import Calculator, descriptors         # imports mordred descriptor librar
calc = Calculator(descriptors, ignore_3D=True)      # sets up a function reading descri
len(calc.descriptors) # tells us how many different types of descriptors are availabl
```

Out[1]:

```
1613
```

Wiener Index

We already calculated the Wiener index for *n*-pentane and 2-methylpentane. Now let’s have mordred do it for us.

In [2]:

```
from mordred import WienerIndex
pentane = Chem.MolFromSmiles('CCCCC')                   # Use rdkit to create a mol fi
methyl_pentane = Chem.MolFromSmiles('CCCC(C)C')         #  and for 2-methylpentane
wiener_index = WienerIndex.WienerIndex()                # create descriptor instance f
result1 = wiener_index(pentane)                         # calculate wiener index for n
result2 = wiener_index(methyl_pentane)                  #  and for 2-methylpentane
print("The Wiener index for n-pentane is: ", result1)  # display result
print("The Wiener index for 2-methylpentane is: ", result2)
```

```
The Wiener index for n-pentane is:  20
The Wiener index for 2-methylpentane is:  32
```

Zagreb Indices

And we can do the same for the different Zagreb indices for *n*-pentane and 2-methylpentane.

In [3]:

```
from mordred import ZagrebIndex

zagreb_index1 = ZagrebIndex.ZagrebIndex(version = 1)            # create descriptor i
zagreb_index2 = ZagrebIndex.ZagrebIndex(version = 2)            # create descriptor i

result_Z1 = zagreb_index1(pentane)                              # calculate Z1 descri
result_Z2 = zagreb_index2(pentane)                              # calculate Z2 descri
print("The Zagreb index 1 for n-pentane is:", result_Z1)
print("The Zagreb index 2 for n-pentane is:", result_Z2)

result_Z1 = zagreb_index1(methyl_pentane)                       # and for 2-methylpen
result_Z2 = zagreb_index2(methyl_pentane)
print("The Zagreb index 1 for 2-methylpentane is:", result_Z1)
print("The Zagreb index 2 for 2-methylpentane is:", result_Z2)
```

```
The Zagreb index 1 for n-pentane is: 14.0
The Zagreb index 2 for n-pentane is: 12.0
The Zagreb index 1 for 2-methylpentane is: 20.0
The Zagreb index 2 for 2-methylpentane is: 18.0
```

As you can see from the code above, each index will have different code that needs to be followed for programming. Each descriptor and the resulting code syntax can be found here http://mordred-descriptor.github.io/documentation/master/api/modules.html

Looping through a list of molecules

Now that we have an understanding on how rdkit and mordred work to get our descriptors, let's simplify the code using a looping structure:

In [4]:

```
smiles = ["CCC", "CCCC", "CCCCC", "CCCC(C)C","CC(C)C(C)C"]         #store smiles stri

for smile in smiles:
    mol = Chem.MolFromSmiles(smile)                      # convert smiles string to m
    result_Z1 = zagreb_index1(mol)                       # calculate Z1 descriptor va
    result_Z2 = zagreb_index2(mol)                       # calculate Z2 descriptor va
    print("The Zagreb index 1 for", smile, "is:", result_Z1)
    print("The Zagreb index 2 for", smile, "is:", result_Z2)
    print()
```

```
The Zagreb index 1 for CCC is: 6.0
The Zagreb index 2 for CCC is: 4.0

The Zagreb index 1 for CCCC is: 10.0
The Zagreb index 2 for CCCC is: 8.0

The Zagreb index 1 for CCCCC is: 14.0
The Zagreb index 2 for CCCCC is: 12.0

The Zagreb index 1 for CCCC(C)C is: 20.0
The Zagreb index 2 for CCCC(C)C is: 18.0

The Zagreb index 1 for CC(C)C(C)C is: 22.0
The Zagreb index 2 for CC(C)C(C)C is: 21.0
```

Using descriptors to predict molecular properties

For this exercise we will take a series of alkanes and create an equation that will allow us to predict boiling points. We will start with a 30 molecule alkane training set. We will obtain various descriptors and see how they can predict the physical property boiling point.

For this exercise we will be using the pandas (Python Data Analysis) library to help us read, write and manage data. We will also use matplotlib to generate graphs.

Boiling Point data

Let's start by reading and graphing a set of boiling point data. First we read our csv file into a pandas "dataframe". Notice that we can generate a nicely formatted table from our dataframe by just entering the name of the dataframe on the last line.

In [5]:

```
import pandas as pd          # import the Python Data Analysis Library with the shorten
df = pd.read_csv("BP.csv") # read in the file into a pandas dataframe
df                          # print the dataframe
```

Out[5]:

	compound	name	BP_C	BP_K	SMILES	MW
0	1	Methane	-162.2	110.95	C	16.043
1	2	Ethane	-88.6	184.55	CC	30.070
2	3	propane	-42.2	230.95	CCC	44.100
3	4	butane	-0.1	273.05	CCCC	58.120
4	5	2-methylpropane	-11.2	261.95	CC(C)C	58.120
5	6	pentane	36.1	309.25	CCCCC	72.150
6	7	2-methylbutane	27.0	300.15	CC(C)CC	72.150
7	8	2,2-dimethylpropane	9.5	282.65	CC(C)(C)C	72.150
8	9	hexane	68.8	341.95	CCCCCC	86.180
9	10	2-methylpentane	60.9	334.05	CC(C)CCC	86.180
10	11	3-methylpentane	63.3	336.45	CC(CC)CC	86.180
11	12	2,2-dimethylbutane	49.8	322.95	CC(C)(CC)C	86.180
12	13	2,3-dimethylbutane	58.1	331.25	CC(C)C(C)C	86.180
13	14	heptane	98.5	371.65	CCCCCCC	100.200
14	15	3-ethylpentane	93.5	366.65	C(C)C(CC)CC	100.200
15	16	2,2-dimethylpentane	79.2	352.35	CC(C)(CCC)C	100.200
16	17	2,3-dimethylpentane	89.8	362.95	CC(C)C(CC)C	100.200
17	18	2,4-dimethylpentane	80.6	353.75	CC(C)CC(C)C	100.200
18	19	2-methylhexane	90.1	363.25	CC(C)CCCC	100.205
19	20	3-methylhexane	91.8	364.95	CC(CC)CCC	100.200
20	21	octane	125.6	398.75	CCCCCCCC	114.230
21	22	3-methylheptane	118.9	392.05	CC(CC)CCCC	114.232
22	23	2,2,3,3-tetramethylbutane	106.5	379.65	CC(C)(C(C)(C)C)C	114.230

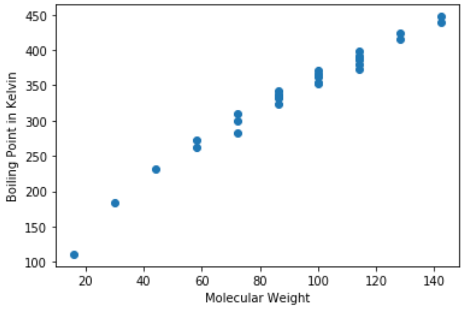
	compound	name	BP_C	BP_K	SMILES	MW
23	24	2,3,3-trimethylpentane	114.7	387.85	CC(C)C(CC)(C)C	114.230
24	25	2,3,4-trimethylpentane	113.7	386.85	CC(C)C(C(C)C)C	114.230
25	26	2,2,4-trimethylpentane	99.3	372.45	CC(C)(CC(C)C)C	114.230
26	27	nonane	150.7	423.85	CCCCCCCCC	128.250
27	28	2-methyloctane	143.0	416.15	CC(C)CCCCCC	128.259
28	29	decane	174.2	447.35	CCCCCCCCCC	142.280
29	30	2-methylnonane	166.9	440.05	CC(C)CCCCCCC	142.280

Graphing the data

Now we can graph the data using matplotlib.

In [16]:

```
import matplotlib.pyplot as plt

plt.scatter(df.MW, df.BP_K)     # plot of boiling point (in K) vs molecular weight
plt.xlabel('Molecular Weight')
plt.ylabel('Boiling Point in Kelvin')
plt.show()
```



Clearly from the data we can see that we have multiple molecules with the same molecular weight, but different boiling points. Molecular weight is therefore not the best predictor of boiling point. We can see if there are other descriptors that we can use such as Weiner or Zagreb. Let’s add various descriptors to the dataframe.

Adding descriptors to the dataset

We can now calculate the Wiener and Zagreb indices for each of our hydrocarbons and add them to the dataframe.

In [7]:

```
# create new lists to store results we calculate
result_Wiener= []
result_Z1= []
result_Z2= []
```

```
for index, row in df.iterrows():                # iterate through each row of the CSV
    SMILE = row['SMILES']                       # get SMILES string from row
    mol = Chem.MolFromSmiles(SMILE)             # convert smiles string to mol file
    result_Wiener.append(wiener_index(mol))     # calculate Wiener index descripter va
    result_Z1.append(zagreb_index1(mol))        # calculate zagreb (Z1) descriptor va
    result_Z2.append(zagreb_index2(mol))        # calculate zagreb (Z2) descriptor va

df['Wiener'] = result_Wiener                 # add the results for WienerIndex to dataframe
df['Z1'] = result_Z1                         # add the results for Zagreb 1 to dataframe
df['Z2'] = result_Z2                         # add the results for Zagreb 2 to dataframe
df                                           # print the updated dataframe
```

Out[7]:

	compound	name	BP_C	BP_K	SMILES	MW	Wiener	Z1	Z2
0	1	Methane	-162.2	110.95	C	16.043	0	0.0	0.0
1	2	Ethane	-88.6	184.55	CC	30.070	1	2.0	1.0
2	3	propane	-42.2	230.95	CCC	44.100	4	6.0	4.0
3	4	butane	-0.1	273.05	CCCC	58.120	10	10.0	8.0
4	5	2-methylpropane	-11.2	261.95	CC(C)C	58.120	9	12.0	9.0
5	6	pentane	36.1	309.25	CCCCC	72.150	20	14.0	12.0
6	7	2-methylbutane	27.0	300.15	CC(C)CC	72.150	18	16.0	14.0
7	8	2,2-dimethylpropane	9.5	282.65	CC(C)(C)C	72.150	16	20.0	16.0
8	9	hexane	68.8	341.95	CCCCCC	86.180	35	18.0	16.0
9	10	2-methylpentane	60.9	334.05	CC(C)CCC	86.180	32	20.0	18.0
10	11	3-methylpentane	63.3	336.45	CC(CC)CC	86.180	31	20.0	19.0
11	12	2,2-dimethylbutane	49.8	322.95	CC(C)(CC)C	86.180	28	24.0	22.0
12	13	2,3-dimethylbutane	58.1	331.25	CC(C)C(C)C	86.180	29	22.0	21.0

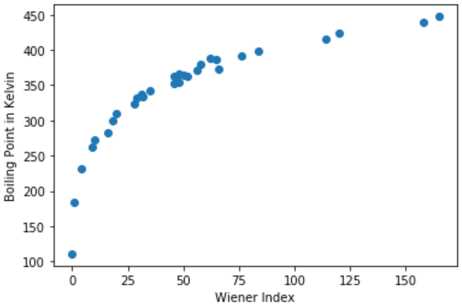
	compound	name	BP_C	BP_K	SMILES	MW	Wiener	Z1	Z2
13	14	heptane	98.5	371.65	CCCCCCC	100.200	56	22.0	20.0
14	15	3-ethylpentane	93.5	366.65	C(C)C(CC)CC	100.200	48	24.0	24.0
15	16	2,2-dimethylpentane	79.2	352.35	CC(C)(CCC)C	100.200	46	28.0	26.0
16	17	2,3-dimethylpentane	89.8	362.95	CC(C)C(CC)C	100.200	46	26.0	26.0
17	18	2,4-dimethylpentane	80.6	353.75	CC(C)CC(C)C	100.200	48	26.0	24.0
18	19	2-methylhexane	90.1	363.25	CC(C)CCCC	100.205	52	24.0	22.0
19	20	3-methylhexane	91.8	364.95	CC(CC)CCC	100.200	50	24.0	23.0
20	21	octane	125.6	398.75	CCCCCCCC	114.230	84	26.0	24.0
21	22	3-methylheptane	118.9	392.05	CC(CC)CCCC	114.232	76	28.0	27.0
22	23	2,2,3,3-tetramethylbutane	106.5	379.65	CC(C)(C(C)(C)C)C	114.230	58	38.0	40.0
23	24	2,3,3-trimethylpentane	114.7	387.85	CC(C)C(CC)(C)C	114.230	62	34.0	36.0
24	25	2,3,4-trimethylpentane	113.7	386.85	CC(C)C(C(C)C)C	114.230	65	32.0	33.0
25	26	2,2,4-trimethylpentane	99.3	372.45	CC(C)(CC(C)C)C	114.230	66	34.0	32.0
26	27	nonane	150.7	423.85	CCCCCCCCC	128.250	120	30.0	28.0
27	28	2-methyloctane	143.0	416.15	CC(C)CCCCCC	128.259	114	32.0	30.0

	compound	name	BP_C	BP_K	SMILES	MW	Wiener	Z1	Z2
28	29	decane	174.2	447.35	CCCCCCCCCC	142.280	165	34.0	32.0
29	30	2-methylnonane	166.9	440.05	CC(C)CCCCCCC	142.280	158	36.0	34.0

Now we can see how each of these descriptors are related to the boiling points of their respective compounds.

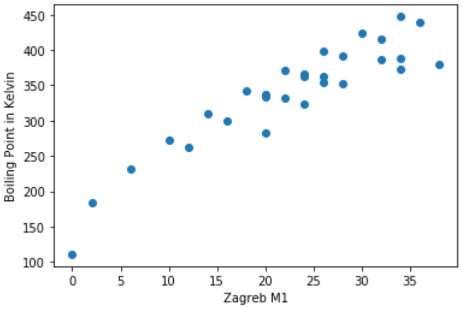
In [8]:

```
plt.scatter(df.Wiener, df.BP_K) # plot of BP versus Wiener index
plt.xlabel('Wiener Index')
plt.ylabel('Boiling Point in Kelvin')
plt.show()
```



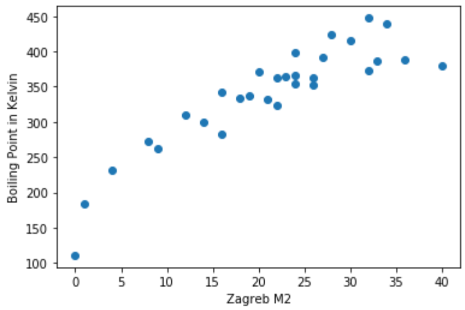
In [9]:

```
plt.scatter(df.Z1, df.BP_K) # plot of BP versus Zagreb M1
plt.xlabel('Zagreb M1')
plt.ylabel('Boiling Point in Kelvin')
plt.show()
```



In [10]:

```
plt.scatter(df.Z2, df.BP_K) # plot of BP versus Zagreb M2
plt.xlabel('Zagreb M2')
plt.ylabel('Boiling Point in Kelvin')
plt.show()
```



Clearly molecular weight was somewhat predictive, but problematic. It looks like using the other indicators we have have some other ways to predict boiling point.

One option is write this data to a new CSV file and work in Microsoft Excel to perform a regression analysis. Exporting the data is straightforward and your instructor may provide instructions on how to analyze the data using Excel.

In [11]:

```
df.to_csv('bp_descriptor_data.csv', encoding='utf-8', index=False)
```

Mulitple regression analysis using statsmodels

The statsmodels package provides numerous tools for performaing statistical analysis using Python. In this case, we want to perform a *multiple linear regression* using all of our descriptors (molecular weight, Wiener index, Zagreb indices) to help predict our boiling point.

In [12]:

```
import statsmodels.api as sm           # import the statsmodels library as sm
X = df[["MW", "Wiener", "Z1", "Z2"]]   # select our independent variables
X = sm.add_constant(X)                 # add an intercept to our model
y = df[["BP_K"]]                       # select BP as our dependent variable
model = sm.OLS(y,X).fit()              # set up our model
predictions = model.predict(X)         # make the predictions
print(model.summary())                 # print out statistical summary
```

```
C:\ProgramData\Miniconda3\envs\OLCC2019\lib\site-packages\numpy\core\fromnumeric.py:2
  return ptp(axis=axis, out=out, **kwargs)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   BP_K   R-squared:                       0.994
Model:                            OLS   Adj. R-squared:                  0.994
Method:                 Least Squares   F-statistic:                     1124.
Date:                Wed, 16 Oct 2019   Prob (F-statistic):           8.16e-28
Time:                        19:04:48   Log-Likelihood:                -93.019
No. Observations:                  30   AIC:                             196.0
Df Residuals:                      25   BIC:                             203.0
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         55.5695      6.745      8.238      0.000      41.677      69.462
MW             4.4325      0.203     21.853      0.000       4.015       4.850
Wiener        -0.6411      0.064     -9.960      0.000      -0.774      -0.509
Z1            -4.3920      1.238     -3.549      0.002      -6.941      -1.843
Z2             0.2982      0.943      0.316      0.754      -1.644       2.240
==============================================================================
Omnibus:                        3.756   Durbin-Watson:                   1.285
Prob(Omnibus):                  0.153   Jarque-Bera (JB):                2.259
Skew:                          -0.583   Prob(JB):                        0.323
Kurtosis:                       3.671   Cond. No.                         755.
==============================================================================


Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

Note above that we now have coeffiecients for an equation that can be used for prediction of boiling points for molecules not included in our dataset. The equation would be:

```
Predicted BP = 4.4325 * MW - 0.6411 * Weiner - 4.3920 * Z1 + 0.2982 * Z2 + 55.5695
```

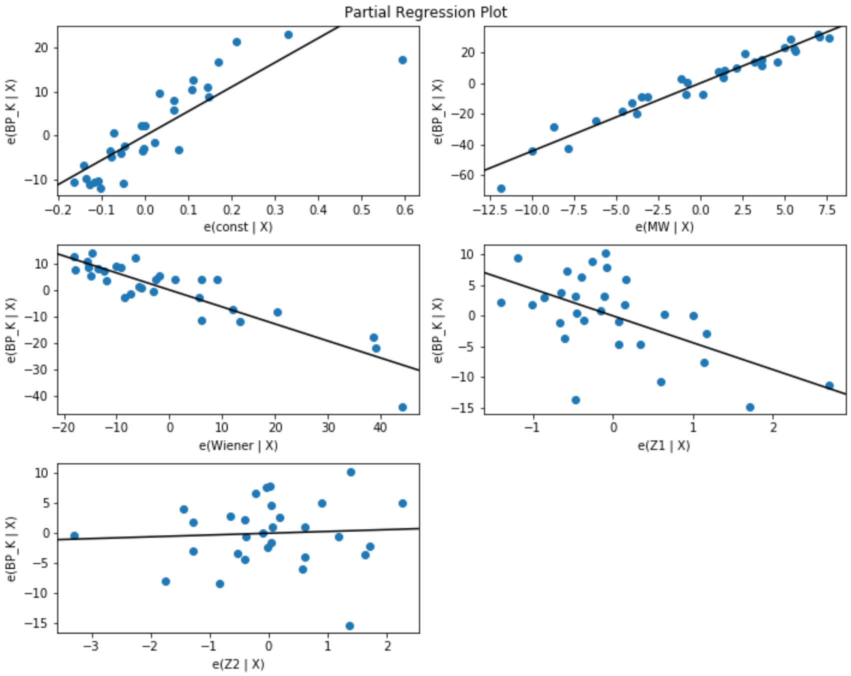
We can use this equation to predict the boiling point of a new molecule. However, before we do, we need to explore the validity of the model.

Model summary and analysis using partial regression plots

A quick look at the results summary shows that the model has an excellent R-squared value. Upon more careful examination, you may notice that one of our descriptors has a very large P value. This would indicate that perhaps the Z2 descriptor is not working well in this case. We can generate a more graphical interpretation that will make this more obvious.

In [13]:

```
fig = plt.figure(figsize=(10,8))
fig = sm.graphics.plot_partregress_grid(model, fig=fig)
```



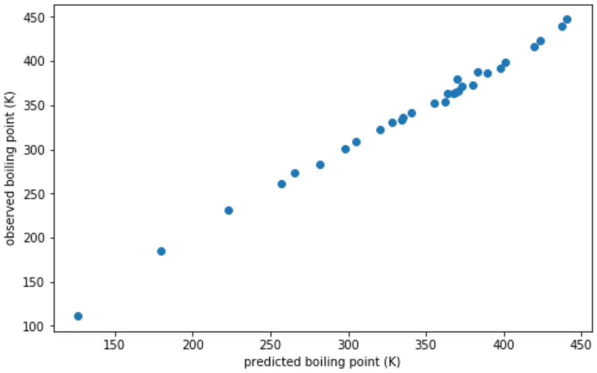
Part of the reason that Z2 may not be predictive in this model is that there is colinearity with the Z1 descriptor. Both descriptors have similar calculations (as outlined in the Libretexts page for this activity). Later on in this exercise we can explore dropping this descriptor.

How good is our model?

If we look at a plot of actual versus predicted boiling points

In [14]:

```
pred_bp = model.fittedvalues.copy()        # use our model to create a set of predicte
fig, ax = plt.subplots(figsize=(8, 5))
lmod = sm.OLS(pred_bp, df.BP_K)            # linear regression of observed vs predicte
res = lmod.fit()                           # run fitting
plt.scatter(pred_bp, df.BP_K)              # plot of of observed vs predicted bp's
plt.ylabel('observed boiling point (K)')
plt.xlabel('predicted boiling point (K)')
plt.show()
print(res.summary())                       # print linear regression stats summary
```



```
                            OLS Regression Results
======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   1.0
Model:                            OLS   Adj. R-squared (uncentered):              1.0
Method:                 Least Squares   F-statistic:                          1.213e+
Date:                Wed, 16 Oct 2019   Prob (F-statistic):                    4.52e-
Time:                        19:05:22   Log-Likelihood:                        -93.0
No. Observations:                  30   AIC:                                      188
Df Residuals:                      29   BIC:                                      189
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
BP_K           0.9998      0.003    348.263      0.000       0.994       1.006
==============================================================================
Omnibus:                        3.635   Durbin-Watson:                   1.284
Prob(Omnibus):                  0.162   Jarque-Bera (JB):                2.167
Skew:                           0.574   Prob(JB):                        0.338
Kurtosis:                       3.643   Cond. No.                         1.00
==============================================================================


Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

The model appears to have very good predictability (R-squared = 1.000) within the original 30 molecule data set. One way to test this model is to use a new molecule with its descriptors to see how well it is predicted. One molecule in the dataset is 2-methylheptane. It has the following data: MW = 114.232 Wiener Index = 79 Z1 = 28 Z2 = 26 Boiling Point = 390.6 K

Using the equation from above we can determine that the boiling point from the equation Predicted BP = 4.4325 *MW - 0.6411* Weiner - 4.3920 *Z1 + 0.2982* Z2 + 55.5695 is 396.0 K. The model gives a 1.4% error for prediction of the boiling point outide the training set.

We had mentioned earlier that Z2 may not be very predictive in this model. We can remove the variable an rerun the analysis to see if we can improve the predictability of the model.

In [15]:

```
import statsmodels.api as sm           # import the statsmodels library as sm
X = df[["MW", "Wiener", "Z1"]]         # select our independent variables, this time 
X = sm.add_constant(X)                 # add an intercept to our model
y = df[["BP_K"]]                       # select BP as our dependent variable
model = sm.OLS(y,X).fit()              # set up our model
predictions = model.predict(X)         # make the predictions
print(model.summary())                 # print out statistical summary
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   BP_K   R-squared:                       0.994
```

```
Model:                            OLS   Adj. R-squared:                  0.994
Method:                 Least Squares   F-statistic:                     1552.
Date:                Wed, 16 Oct 2019   Prob (F-statistic):           1.99e-29
Time:                        19:05:30   Log-Likelihood:                -93.078
No. Observations:                  30   AIC:                             194.2
Df Residuals:                      26   BIC:                             199.8
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         55.4979      6.624      8.378      0.000      41.882      69.113
MW             4.4114      0.188     23.433      0.000       4.024       4.798
Wiener        -0.6397      0.063    -10.139      0.000      -0.769      -0.510
Z1            -4.0260      0.430     -9.364      0.000      -4.910      -3.142
==============================================================================
Omnibus:                        2.917   Durbin-Watson:                   1.326
Prob(Omnibus):                  0.233   Jarque-Bera (JB):                1.624
Skew:                          -0.507   Prob(JB):                        0.444
Kurtosis:                       3.521   Cond. No.                         741.
==============================================================================


Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

IThe model appears to have very good predictability (R-squared = 0.994) within the original 30 molecule data set. Let's reexamine 2-methylheptane: MW = 114.232 Wiener Index = 79 Z1 = 28 Boiling Point = 390.6 K

Using the equation from above we can determine that the boiling point from the equation Predicted BP = 4.4114 *MW - 0.6397* Weiner - 4.0260 * Z1 + 55.4979 is 396.2 K. The model also gives a 1.4% error for prediction of the boiling point outide the training set.

We can see here that Z2 doesn't really change the result of the calculation, and can probably be best left out to simplify the model.

Keep in mind that we have completed this analysis with only a training set of 30 molecules. If the training set had more molecules, you should be able to develop a better model.

Assignment

You originally ran this analysis on a 30 molecule data set (BP.CSV). You also have available to you a 102 molecule data set (102BP.CSV).

- Complete the above analysis using the expanded data set to determine if a better predictive model can be obtained with a larger training set. Note that 2-methylheptane is in this new dataset so you will need to choose a new test molecule.

When you have completed the analysis, you will create a new analysis:

- Choose four new topological and other calculated descriptors found in Mordred http://mordred-descriptor.github.io/documentation/master/api/modules.html
- Complete simple linear analysis for each of your new descriptors.
- Complete a multiple linear regression to create an equation that best represents the data boiling point data and your descriptors.
- Create a separate sheet that has your regression data.
- Make a plot of Actual vs Predicted BP for your regression.
- Choose a new molecule not in the dataset (not 2-methylheptane, be creative and use chemical intuition).

- Use your multiple linear equation to predict this molecule’s BP and look of the literature value.
- Write a short one-two page paper that includes:
  - What your new chosen descriptors mean
  - Which new chosen descriptors correlate
  - What is the overall equation calculated
  - How to choose the molecule to test
  - How close this multiple linear regression predicts your boiling point of your molecule

5.6: R Assignment

Structure Search

S. Kim, J. Cuadros

Objectives

- Learn various types of structure searches including identity search, similarity search, substructure and super structure searches.
- Learn the optional parameters available for each search type.

Using PUG-REST, one can perform various types of structure searches (https://bit.ly/2lPznCo), including: - identity search - similarity search - super/substructure search - molecular formula search

As explained in a PubChem paper (https://bit.ly/2kirxky), whereas structure search can be performed in either an ‘asynchronous’ or ‘synchronous’ way, it is highly recommended to use the synchronous approach.

The synchronous searches are invoked by using the keywords prefixed with ‘fast’, such as fastidenity, fastsimilarity_2d, fastsimilarity_3d, fastsubstructure, fastsuperstructure, and fastformula.

In this task, we will use some cheminformatics packages to ease some processes. In R, some options are rcdk , ChemmineR and ChemmineOB . In Python, a useful package is RDKit ; in R, we’ll make use of it online version, the Beaker API of ChEMBL (https://chembl.gitbook.io/chembl-interface-documentation/web-services).

1. Identity Search

PUG-REST allows you to search the PubChem Compound database for molecules identical to the query molecule. PubChem’s identity search supports different contexts of chemical identity, which the user can specify using the optional parameter, “identity_type”. Here are some commonly-used chemical identity contexts. - same_connectivity: returns compounds with the same atom connectivity as the query molecule, ignoring stereochemistry and isotope information. - same_isotope: returns compounds with the same isotopes (as well as the same atom connectivity) as the query molecule. Stereochemistry will be ignored. - same_stereo: returns compounds with the same stereochemistry (as well as the same atom connectivity) as the query molecule. Isotope information will be ignored. - same_stereo_isotope: returns compounds with the same stereochemistry AND isotope information (as well as the same atom connectivity). This is the default.

The following code cell demonstrates how these different contexts of chemical sameness affects identity search in PubChem.

```
if(!require("httr")) {
 install.packages(("httr"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("httr")
}
if(!require("jsonlite")) {
 install.packages(("jsonlite"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
library("jsonlite")
}
if(!require("png")) {
 install.packages(("png"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("png")
}
if(!require("grid")) {
 install.packages(("grid"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("grid")
}
if(!require("gridExtra")) {
 install.packages(("gridExtra"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("gridExtra")
}
```

```
prolog <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
smiles <- "C(/C=C/Cl)Cl"
options <- c('same_stereo_isotope',
 'same_stereo',
 'same_isotope',
 'same_connectivity') # same_stereo_isotope is the default

for(opt in options) {
 print(paste ("#### Identity_type:", opt))
 url <- paste(prolog,
 "/compound/fastidentity/smiles/",
 "property/isomericsmiles/csv?smiles=",
 URLencode(smiles,reserved = T),
 "&identity_type=",
 opt, sep="")
 dfChem <- read.table(url, sep=",", header=T)
 print(dfChem)

 url_img <- paste("https://www.ebi.ac.uk/chembl/api/utils/smiles2image",
 "?size=300&engine=rdkit",sep="")
 res <- POST(url_img,
 body=list(smiles=paste(dfChem[,2],collapse="\n")))
 img <- readPNG(res$content, native=TRUE)
 grid.arrange(rasterGrob(img))

 Sys.sleep(.5)
}
```

Exercise 1a: Find compounds that has the same atom connectivity and isotope information as the query molecule. query <- "CC1=CN=C(C(=C1OC)C)CS@C2=NC3=C(N2)C=C(C=C3)OC" For each compound returned from the search, retrieve the following information.

- CID
- Isomeric SMILES string
- chemical synonyms (for simplicity, print only the five synonyms that first occur in the name list retrieved for each compound)
- Structure image

```
# Write your code here
```

2. Similarity search

PubChem supports 2-dimensional (2-D) and 3-dimensional (3-D) similarity searches. Because molecular similarity is not a measurable physical observable but a subjective concept, many approaches have been developed to evaluate it. Detailed discussion on how PubChem quantifyies molecular similarity, read the following LibreTexts page:

Searching PubChem Using a Non-Textual Query (https://bit.ly/2lPznCo)

The code cell below demonstrates how to perform 2-D and 3-D similarity searches.

```
mydata <- list(smiles="C1COCC(=O)N1C2=CC=C(C=C2)N3C[C@@H](OC3=O)CNC(=O)C4=CC=C(S4)Cl"
url <- paste(prolog,
 "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=99",
 sep="")
res <- POST(url,body=mydata)
cids <- unlist(strsplit(rawToChar(res$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids)))
```

```
 print(cids)
```

IMPORTANT: It is worth mentioning that the parameter name “Threshold” is case-sensitive. If “threshold” is used (rather than “Threshold”), it will be ignored and the default value (0.90) will be used for the parameter. [As a matter of fact, all optional parameter names in PUG-REST are case-sensitive.]

```
url1 <- paste(prolog, "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=95", sep
url2 <- paste(prolog, "/compound/fastsimilarity_2d/smiles/cids/txt?threshold=95", sep
# "threshold=95" is ignored.
```

```
 res1 <- POST(url1,body=mydata)
res2 <- POST(url2,body=mydata)
cids1 <- unlist(strsplit(rawToChar(res1$content),"\n",fixed=T))
cids2 <- unlist(strsplit(rawToChar(res2$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids1), "vs.", length(cids2)))
```

It is possible to run 3-D similarity search using PUG-REST. However, because 3-D similarity search takes much longer than 2-D similarity search, it often exceeds the 30-second time limit and returns a time-out error, especially when the query molecule is big.

In addition, for 3-D similarity search, it is not possible to adjust the similarity threshold (that is, the optional “Threshold” parameter does not work). 3-D similarity search uses a shape-Tanimoto (ST) of >=0.80 and a color-Tanimoto (CT) of >=0.50 as a similarity threshold. Read the libreTexts page for more details (https://bit.ly/2lPznCo).

```
mydata <- list(smiles="CC(=O)OC1=CC=CC=C1C(=O)O")
url <- paste(prolog,
 "/compound/fastsimilarity_3d/smiles/cids/txt",
 sep="")
res <- POST(url,body=mydata)
cids <- unlist(strsplit(rawToChar(res$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids)))
```

Exercise 2a: Perform 2-D similarity search with the following query, using a threshold of 0.80 and find the macromolecule targets of the assays in which the returned compounds were tested. You will need to take these steps.

- Run 2-D similarity search using the SMILES string as a query (with Threshold=80).
- Retrieve the AIDs in which any of the returned CIDs was tested "active".
- Retrieve the gene symbols of the targets for the returned AIDs.

```
query <- "[C@@H]23C(=O)[C@H](N)C(C)[C@H](CCC1=COC=C1)[C@@]2(C)CCCC3(C)C"
```

```
 # Write your code here
```

3. Substructure/Superstructure search¶

When a chemical structure occurs as a part of a bigger chemical structure, the former is called a substructure and the latter is referred to as a superstructure (https://bit.ly/2lPznCo). PUG-REST supports both substructure and superstructure searches. For example, below is an example for substructure search using the core structure of antibiotic drugs called cephalosporins as a query (https://en.Wikipedia.org/wiki/Cephalosporin).

```
mydata <- list(smiles="C12(SCC(=C(N1C([C@H]2NC(=O)[*])=O)C(=O)O[H])[*])[H]")
url <- paste(prolog,
 "/compound/fastsubstructure/smiles/cids/txt?Stereo=exact",
 sep="")
res <- POST(url,body=mydata)
cids <- unlist(strsplit(rawToChar(res$content),"\n",fixed=T))
print(paste("# Number of CIDs:", length(cids)))
```

An important thing to remember about substructure search is that, if the query structure is not specific enough (that is, not big enough), it will return too many hits for the PubChem server can handle. For example, if you perform substructure search using the "C-C" as a query, it will give you an error, because PubChem has ~96 million (organic) compounds with more than two carbon atoms and most of them will have the "C-C" unit. Therefore, if you get an "time-out" error while doing substructure search, consider providing more specific structure as an input query.

Exercise 3a: Below is the SMILES string for a HCV (Hepatitis C Virus) drug (Sovaldi). Perform substructure search using this SMILES string as a query, identify compounds that are mentioned in patent documents, and create a list of the patent documents that mentioning them.

Use the default options for substructure search. Use the "XRefs" operation to retrieve Patent IDs associated with the returned compounds. For simplicity, ignore the CID-Patent ID mapping. (That is, no need to track which CID is associated with which patent document.)

```
 query <- "C[C@@H](C(=O)OC(C)C)N[P@](=O)(OC[C@@H]1[C@H]([C@@]([C@@H](O1)N2C=CC(=O)NC2
O)OC3=CC=C3"
```

```
 # Write your code here
```

4. Molecular formula search

Strictly speaking, molecular formula search is not structure search, but its PUG-REST request URL is constructed in a similar way to structure searches like identity, similarity, and substructure/superstructure searches.

```
query <- "C22H28FN3O6S" # Molecular formula for Crestor (Rosuvastatin: CID 446157)
url <- paste(prolog, "/compound/fastformula/",
 query, "/cids/txt", sep="")
```

```
cids <- readLines(url)
print(paste("# Number of CIDs:", length(cids)))
```

It is possible to allow other elements to be present in addition to those specified by the query formula, as shown in the following example.

```
query <- "C22H28FN3O6S" # Molecular formula for Crestor (Rosuvastatin: CID 446157)
url <- paste(prolog, "/compound/fastformula/",
 query, "/cids/txt?AllowOtherElements=true", sep="")
cids <- readLines(url)
print(paste("# Number of CIDs:", length(cids)))
```

Exercise 4a: The general molecular formula for alcohols is [for example, CH4O (methanol), C2H6O (ethanol), C3H8O (propanol), etc]. Run molecular formula search using this general formula for n=1 through 20 and retrieve the XLogP values of the returned compounds for each value of n. Print the minimum and maximum XLogP values for each n value.

```
# Write your code here
```

CHAPTER OVERVIEW

6: Molecular Similarity

Hypothes.is Tag= f19OLCCc6

Note: Any annotation tagged **f19OLCCc6** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

This page is under construction, and will hold content for module 5 of the Fall 2019 Cheminformatics OLCC.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

Topic hierarchy

6.1: Molecular Descriptors

Molecular Similarity

Molecular similarity [1-3] is one of the most heavily exploited concepts in cheminformatics and related areas (such as medicinal chemistry and drug discovery). It is applied to multiple tasks, including similarity searching [1], property prediction [4], synthesis design [5], virtual screening [2,3,6], cluster analysis [7,8], and molecular diversity analysis [9-11]. However, because molecular similarity is a concept, not a physical observable, "measuring" molecular similarity is inherently subjective and context-dependent. There is no correct or authoritative measure of molecular similarity. As a result, various similarity measures have been proposed to quantify the degree of structural similarity between molecules. In general, these measures involve two principal components [12]:

- **Molecular descriptors** that represent the structures of the molecules being compared.
- **Similarity coefficient** (metric) used to compute a quantitative score for the degree of similarity based on the weighted values of structural descriptors.

The molecular descriptors may need to be pre-processed before the similarity calculation, using a weighting scheme that assigns differing degrees of importance to various components of molecular descriptors. For this reason, some papers list the weighting scheme as a third component of similarity measures [1,13]. While some studies [14,15] have focused on the effects of the weighting schemes upon similarity calculations, much more attention has been given to molecular descriptors and similarity coefficients. Therefore, this chapter also focuses on these two components.

Molecular descriptors

There are many molecular descriptors that capture different aspects of molecules, but they are broadly classified according to their "dimensionality" [16]. One-dimensional (1-D) descriptors include bulk properties and physicochemical parameters (e.g., log P, molecular weight, polar surface area). Two-dimensional (2-D) descriptors include structural fragments or connectivity indices derived from the 2-D representation of the molecule. Three-dimensional (3-D) descriptors, such as molecular shape, are derived from 3-D molecular structures (i.e., 3-D coordinates of the atoms in the molecule). In this chapter, we focus on 2-D molecular fingerprints, which encodes the 2-D structure of molecules. While many molecular fingerprints have been developed, we discuss two types of molecular fingerprints, structural keys and hashed fingerprints, because they are more widely used than others.

Structural keys

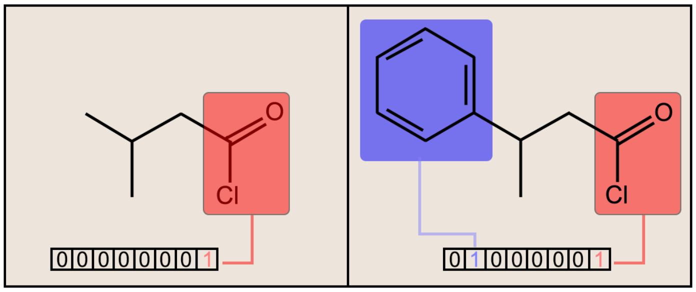


Fig. 1. (above) Two molecules are shown along with the respective bit substructures highlighted for comparison. The number of bits and designations used for this figure is simply for display and illustrative purposes. The true fingerprint would be much longer.

In structural keys, the structure of a molecule is encoded into a binary bit string (that is, a sequence of 0's and 1's), each bit of which corresponds to a "pre-defined" structural feature (e.g., substructure or fragment). If the molecule has a pre-defined feature, the bit position corresponding to this feature is set to 1 (ON). Otherwise, it is set to 0 (OFF). It is important to understand that structural keys cannot encode structural features that are not pre-defined in the fragment library. Examples are the MACCS keys [17,18] and PubChem Fingerprints [19].

- MACCS keys

  The MACCS (Molecular ACCess System) keys [17,18] are one of the most commonly used structural keys. They are sometimes referred to as the MDL keys, named after the company that developed them [the MDL Information Systems (now BIOVIA)]. While there are two sets of MACCS keys (one with 960 keys and the other containing a subset of 166 keys), only the shorter fragment definitions are available to the public. These 166 public keys are implemented in popular open-source cheminformatics software packages, including RDKit [20], OpenBabel [21,22], CDK [23,24], etc. The fragment definitions for the MACCS 166 keys can be found in this document:

https://github.com/rdkit/rdkit/blob/master/rdkit/Chem/MACCSkeys.py

- PubChem fingerprints

The PubChem fingerprint [19] is a 881-bit-long structural key, which is used by PubChem for similarity searching (interactively through the PubChem Homepage or programmatically through PUG-REST). It is also used for structure neighboring, which "pre-computes" a list of similar chemical structure for each compound. This pre-computed list is accessible through the Compound Summary page (the Related Compounds and Related Compounds with Annotation sections). The fragment dictionary of the PubChem fingerprint is organized in seven sections, as described in the following document:

ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf

Hashed Fingerprints

An alternative to structural keys is hashed fingerprints. Contrary to structural keys, hashed fingerprints do not require a pre-defined fragment library. Instead, they are generated by enumerating through the molecule all possible fragments that are not bigger than a certain size and then converting these fragments into numeric values using a "hash" function (https://en.Wikipedia.org/wiki/Hash_function). These numeric values can be used to indicate bit positions in the hashed fingerprints.

Hash functions are used to map data of arbitrary size to "fixed-size" values. Enumerating all possible fragments with a molecule may result in a very large number of fragments. Hashing them into values within a fixed range inevitably results in "bit collisions", in which different fragments are converted into the same numeric value (and the same bit position). Because of this, there is no one-to-one correspondence between fragments and fingerprint bits (contrary to structural keys).

Hashed fingerprints may be further classified into topological or path-based fingerprints and circular fingerprints, according to the way by which the fragments are enumerated.

- **Path-based fingerprints**

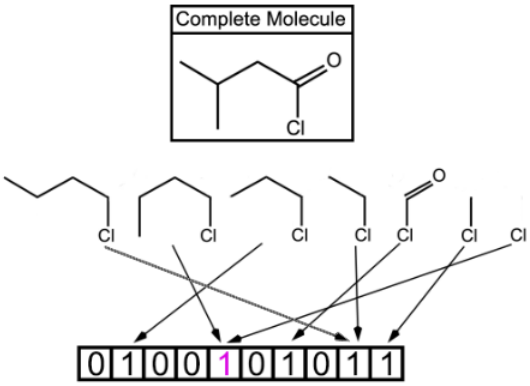


Fig. 2. Shown above is a topological fingerprint with multiple collisions between fragments. A bit collision is represented by having two or more arrows from the molecular fragments pointing to the same bit value. Starting with the chlorine atom, all of the possible fragments are shown. However in a true fingerprint, each atom could be the starting point which would allow for many more fragments than this example shows. The more bits allowed, the less likely for the bit collisions, which is represented by having two collisions due to only 10 bits being used.

In this type of fingerprints, fragments of the molecule are generated by following a (usually linear) path up to a certain number of bonds within the molecule. The most well-known example of path-based fingerprints is the Daylight fingerprint [25,26].

-

Circular fingerprints

Circular fingerprints are generated by considering the “circular” environment of each atom up to a given “radius” or “diameter”. Examples of circular fingerprints are extended-connectivity fingerprints (ECFPs) [27]. ECFPs are generated using a variant of the Morgan algorithm [28], which is a method for solving the molecular isomorphism problem (i.e., how to identify identical molecules that have different atom numberings). Different flavors of ECFPs may be generated by selecting different maximum diameter of the circular atom neighborhood and they are referred to as ECFP2, ECFP4, ECFP6, etc., where the digit at the end indicates the maximum diameter value employed to generate the fingerprint. The most commonly used ones are ECFP4 and ECFP6.

Another example of circular fingerprints is functional-class fingerprints (FCFPs) [27], which are a variation of ECFPs. FCFPs are further abstracted in that FCFPs encodes atom’s roles (not atoms). At the initial stage of FCFP generation, each atom in the molecule is assigned a special code that represents one of the atom roles (e.g., hydrogen-bond acceptor and donor, negatively or positively ionizable, aromatic, and halogen), and these codes (not the atoms) are used to generate FCFPs, through the same process as ECFPs.

References

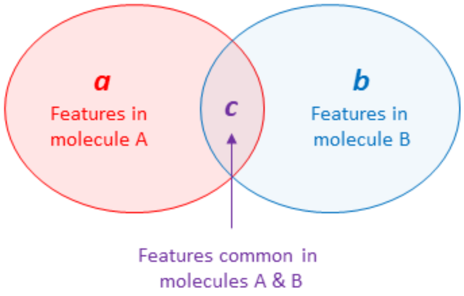
1. Willett P, Barnard JM, Downs GM: **Chemical similarity searching.** *J Chem Inf Comput Sci* 1998, **38:**983-996.
2. Cereto-Massague A, Ojeda MJ, Valls C, Mulero M, Garcia-Vallve S, Pujadas G: **Molecular fingerprint similarity search in virtual screening.** *Methods* 2015, **71:**58-63.
3. Muegge I, Mukherjee P: **An overview of molecular fingerprint similarity search in virtual screening.** *Expert Opin Drug Discov* 2016, **11:**137-148.
4. Brown RD, Martin YC: **Use of structure Activity data to compare structure-based clustering methods and descriptors for use in compound selection.** *J Chem Inf Comput Sci* 1996, **36:**572-584.
5. Wipke WT, Rogers D: **ARTIFICIAL-INTELLIGENCE IN ORGANIC-SYNTHESIS - SST - STARTING MATERIAL SELECTION-STRATEGIES - AN APPLICATION OF SUPERSTRUCTURE SEARCH.** *J Chem Inf Comput Sci* 1984, **24:**71-81.
6. Eckert H, Bojorath J: **Molecular similarity analysis in virtual screening: foundations, limitations and novel approaches.** *Drug Discov Today* 2007, **12:**225-233.
7. Cruz R, Lopez N, Quintero M, Rojas G: **Cluster analysis from molecular similarity matrices using a non-linear neural network.** *J Math Chem* 1996, **20:**385-394.
8. Pan DH, Iyer M, Liu JZ, Li Y, Hopfinger AJ: **Constructing optimum blood brain barrier QSAR models using a combination of 4D-molecular similarity measures and cluster analysis.** *J Chem Inf Comput Sci* 2004, **44:**2083-2098.
9. Golbraikh A: **Molecular dataset diversity indices and their applications to comparison of chemical databases and QSAR analysis.** *J Chem Inf Comput Sci* 2000, **40:**414-425.
10. Klein CT, Kaiser D, Ecker G: **Topological distance based 3D descriptors for use in QSAR and diversity analysis.** *J Chem Inf Comput Sci* 2004, **44:**200-209.
11. Koutsoukas A, Paricharak S, Galloway W, Spring DR, Ijzerman AP, Glen RC, Marcus D, Bender A: **How Diverse Are Diversity Assessment Methods? A Comparative Analysis and Benchmarking of Molecular Descriptor Space.** *J Chem Inf Model* 2014, **54:**230-242.
12. Holliday JD, Hu CY, Willett P: **Grouping of coefficients for the calculation of inter-molecular similarity and dissimilarity using 2D fragment bit-strings.** *Comb Chem High Throughput Screen* 2002, **5:**155-166.
13. Chen X, Reynolds CH: **Performance of similarity measures in 2D fragment-based similarity searching: Comparison of structural descriptors and similarity coefficients.** *J Chem Inf Comput Sci* 2002, **42:**1407-1414.
14. Bath PA, Morris CA, Willett P: **EFFECT OF STANDARDIZATION ON FRAGMENT-BASED MEASURES OF STRUCTURAL SIMILARITY.** *J Chemometr* 1993, **7:**543-550.
15. Turner DB, Willett P, Ferguson AM, Heritage TW: **Similarity Searching in Files of Three-Dimensional Structures: Evaluation of Similarity Coefficients and Standardisation Methods for Field-Based Similarity Searching.** *SAR and QSAR in Environmental Research* 1995, **3:**101-130.
16. Xue L, Bajorath J: **Molecular descriptors in chemoinformatics, computational combinatorial chemistry, and virtual screening.** *Comb Chem High Throughput Screen* 2000, **3:**363-372.
17. Durant JL, Leland BA, Henry DR, Nourse JG: **Reoptimization of MDL keys for use in drug discovery.** *J Chem Inf Comput Sci* 2002, **42:**1273-1280.

18. **THE KEYS TO UNDERSTANDING MDL KEYSET TECHNOLOGY.** https://www.3dsbiovia.com/products/pdf/keys-to-keyset-technology.pdf. Accessed Oct. 2019.
19. **PubChem Substructure Fingerprint.** ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf. Accessed Oct. 2019.
20. **RDKit.** https://www.rdkit.org/. Accessed October 2019.
21. O'Boyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR: **Open Babel: An open chemical toolbox.** *J Cheminformatics* 2011, **3:**33.
22. **The Open Babel Package.** https://openbabel.org. Accessed October, 2019.
23. **Chemistry Development Kit (CDK).** https://cdk.github.io/. Accessed October 2019.
24. Willighagen EL, Mayfield JW, Alvarsson J, Berg A, Carlsson L, Jeliazkova N, Kuhn S, Pluskal T, Rojas-Cherto M, Spjuth O, Torrance G, Evelo CT, Guha R, Steinbeck C: **The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching.** *J Cheminformatics* 2017, **9:**33.
25. **Daylight Theory: Fingerprints.** https://www.daylight.com/dayhtml/doc/theory/theory.finger.html. Accessed October 2019.
26. **Daylight Fingerprints.** https://www.daylight.com/meetings/summerschool01/course/basics/fp.html. Accessed October 2019.
27. Rogers D, Hahn M: **Extended-Connectivity Fingerprints.** *J Chem Inf Model* 2010, **50:**742-754.
28. Morgan HL: **GENERATION OF A UNIQUE MACHINE DESCRIPTION FOR CHEMICAL STRUCTURES-A TECHNIQUE DEVELOPED AT CHEMICAL ABSTRACTS SERVICE.** *Journal of Chemical Documentation* 1965, **5:**107-&.

Tags recommended by the template: article:topic

6.2: Similarity Coefficients

Many similarity metrics have been proposed and some commonly used metrics in cheminformatics are listed below, along with their mathematical definitions for binary features.



Metric name	Formula for binary variables	Minimum	Maximum
Tanimoto (Jaccard) coefficient	$S_{AB} = \frac{C}{A+B-C}$	0	1
Dice coefficient (Hodgkin index)	$S_{AB} = \frac{2C}{A+B}$	0	1
Cosine coefficient (Carbo index)	$S_{AB} = \frac{C}{\sqrt{ab}}$	0	1
Soergel distance	$D_{AB} = \frac{a+b-2c}{a+b-c}$	0	1
Euclidean distance	$D_{AB} = \sqrt{a+b-2c}$	0	N^{α}
Hamming (Manhattan or city-block) distance	$D_{AB} = a+b-2c$	0	N^{α}

$^{\alpha}$ The length of molecular fingerprints.

In the above table, the first three metrics (Tanimoto, Dice, and Cosine coefficients) are similarity metrics (S_{AB}), which evaluates how similar two molecules are to each other. The other three (Soergel, Euclidean, and Hamming coefficients) are distance or dissimilarity metrics (D_{AB}), which quantify how dissimilar the molecules are. These dissimilarity measures can be converted into similarity measures in a simple way. For example, for dissimilarity metrics whose possible values range from 0 to 1 (e.g., Soergel distance), the similarity score (S_{AB}) between two molecules can be computed simply by subtracting the dissimilarity score from unity:

$$S_{AB} = 1 - D_{AB}$$

Note that the Soergel distance between two molecules is the complement of their Tanimoto coefficient (that is, the sum of the two metrics is 1), while they are developed independently of each other.

If a distance metric has an upper-bound value greater than 1, (e.g., Euclidean or Hamming distance), the following equation [1] can be used to convert the dissimilarity score to the similarity score:

$$S_{AB} = \frac{1}{1 + D_{AB}}$$

According to this equation, if two molecules are identical to each other, the distance (D_{AB}) between them is zero, and their similarity score (S_{AB}) becomes 1. On the other hand, as the D_{AB} value increases (i.e., for dissimilar molecules), the S_{AB} score approaches to 0.

An important question about molecular similarity evaluation is “how similar is similar?”. To answer this question, it is necessary to have a similarity threshold that can be used to determine whether molecules are similar enough. In 1996, Patterson et al. [2] analyzed sets of active compounds selected from scientific articles and showed that a Tanimoto coefficient of 0.85 or greater reflected a high probability of two compounds having the same activity. Since then, this Tanimoto value of 0.85 has been used in many studies as a general threshold for molecular similarity evaluation. However, as demonstrated in several studies [3], different molecular fingerprints give different similarity score distributions. For example, the Tanimoto score of 0.85 computed from MACCS keys have a different probability of the two compounds sharing the same activity than the probability represented by the same Tanimoto value (0.85) computed from ECFPs. The programming assignments for this chapter will help understand the impact of different molecular fingerprints upon computed similarity coefficient values.

References

1. Todeschini R, Ballabio D, Consonni V, Mauri A, Pavan M: **CAIMAN (Classification And Influence Matrix Analysis): A new approach to the classification based on leverage-scaled functions.** Chemometrics Intell Lab Syst 2007, 87:3-17.
2. Patterson DE, Cramer RD, Ferguson AM, Clark RD, Weinberger LE: **Neighborhood behavior: A useful concept for validation of ''molecular diversity'' descriptors.** J Med Chem 1996, 39:3049-3059.
3. Jasial S, Hu Y, Vogt M, Bajorath J: **Activity-relevant similarity values for fingerprints and implications for similarity searching [version 2; peer review: 3 approved]**. F1000Research 2016, 5

6.3: Discussion

While there are many molecular fingerprints and similarity coefficients, it is not feasible to use all possible combination of them for a given project with limited time and resources. For this reason there have been many studies that compared performances among different fingerprints and similarity coefficients. In their large-scale analysis of 37 molecular descriptors [1], Bender and coworkers evaluated similarity between the descriptors and identified four broad descriptor classes: (1) circular fingerprints, (2) circular fingerprints considering counts, (3) path-based fingerprints and structural keys, and (4) pharmacophoric descriptors. This study suggests that the performance of the descriptors is much more defined by those four classes than the particular parametrization used or individual descriptors. This implies that descriptors that belong to the same class are likely to give similar results (e.g., similar hit compound lists) when they are used for molecular similarity evaluation.

In general, the Tanimoto coefficient is a preferred metric for molecular similarity comparison, but Dice and Cosine coefficients are considered as good alternatives [2]. For example, a study by Bajusz and Héberger [2] compared eight well-known similarity distance metrics on a large data set of molecular fingerprints. This study concluded that the Tanimoto, Dice, Cosine, and Soergel coefficients are the best metrics for similarity calculation, in the sense that they produce the most similar rankings to those averaged over the rankings produced by the eight similarity metrics considered. The Euclidean and Manhattan distances were found to be not optimal because they gave different rankings from other metrics.

Further Reading

- Molecular Similarity in Medicinal Chemistry

  https://doi.org/10.1021/jm401411z

- Molecular similarity: a key technique in molecular informatics

  https://doi.org/10.1039/B409813G

- Daylight Theory: Fingerprints

  https://www.daylight.com/dayhtml/doc/theory/theory.finger.html

- How Similar Are Similarity Searching Methods? A Principal Component Analysis of Molecular Descriptor Space

  https://doi.org/10.1021/ci800249s

- Extended-Connectivity Fingerprints

  https://doi.org/10.1021/ci100050t

References

1. Bender A, Jenkins JL, Scheiber J, Sukuru SCK, Glick M, Davies JW: **How Similar Are Similarity Searching Methods? A Principal Component Analysis of Molecular Descriptor Space.** J Chem Inf Model 2009, 49:108-119.
2. Bajusz D, Racz A, Heberger K: **Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations?** J Cheminformatics 2015, 7:20.

6.4: Python Assignment

Molecular Similarity

Downloadable Files

 lecture06_Molecular Similarity.ipynb

- Download the ipynb file and run your Jupyter notebook.
    - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
    - This page is an html version of the above .ipynb file.
        - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

- Requests
- RDKit

NOTE: This is a 2 week assignment

Objectives

- Generete molecular fingerprints for a given molecule.
- Evaluate structural similarity between molecules using different molecular fingerprints and similarity metrics.

Many useful documents/papers describe various aspects of molecular similarity, including molecular fingerprints and similarity measures. Please read these if you need more details.

- Getting Started with the RDKit in Python (https://www.rdkit.org/docs/GettingStartedInPython.html#fingerprinting-and-molecular-similarity)
- Fingerprint Generation, GraphSim Toolkit 2.4.2 (https://docs.eyesopen.com/toolkits/python/graphsimtk/fingerprint.html)
- Chemical Fingerprints (https://docs.chemaxon.com/display/docs/Chemical+Fingerprints)
- Extended-Connectivity Fingerprints (https://doi.org/10.1021/ci100050t)

1. Fingerprint Generation

In [39]:

```
from rdkit import Chem
```

In [40]:

```
mol = Chem.MolFromSmiles('CC(C)C1=C(C(=C(N1CC[C@H](C[C@H](CC(=O)O)O)O)C2=CC=C(C=C2)F)
```

1-(1) MACCS keys

The MACCS key is a binary fingerprint (a string of 0's and 1's). Each bit position represents the presence (=1) or absence (=0) of a pre-defined structural feature. The feature definitions for the MACCS keys are available at: https://github.com/rdkit/rdkit/blob/master/rdkit/Chem/MACCSkeys.py

In [41]:

```
from rdkit.Chem import MACCSkeys
fp = MACCSkeys.GenMACCSKeys(mol)
```

In [42]:

```
print(type(fp))

for i in range(len(fp)):
    print(fp[i], end='')

fp.ToBitString()    # Alternative, easier way to convert it to a bitstring.
```

```
<class 'rdkit.DataStructs.cDataStructs.ExplicitBitVect'>
000000000000000000000000000000000000000000100000000001000000001001000000001100001001
```

Out[42]:

```
'000000000000000000000000000000000000000000100000000001000000001001000000001100001001
```

In [43]:

```
len(fp)
```

Out[43]:

```
167
```

Note that the MACCS key is **166-bit-long**, but RDKit generates a 167-bit-long fingerprint. It is because the index of a list/vector in many programming languages (including python) begins at 0. To use the original numbering of the MACCS keys (1-166) (rather than 0-165), the MACCS keys were implemented to be 167-bit-long, with Bit 0 being always zero. Because Bit 0 is set to OFF for all compounds, it does not affect the evaluation of molecular similarity.

These are some methods that allow you to get some additional information on the MACCS Keys.

In [44]:

```
print(fp.GetNumBits())
print(fp.GetNumOffBits())
print(fp.GetNumOnBits())
print(fp.ToBinary())
```

```
167
105
62
b'\xe0\xff\xff\xff\xa7\x00\x00\x00>\x00\x00\x00T\x14\x10\x04\x10\x00\x08\x04\x02\x02\
```

Exercise 1a: Generate the MACCS keys for the molecules represented by the following SMILES, and get the positions of the bits set to ON in each of the three fingerprints. What fragments do these bit positions correspond to? (the bit definitions are available at

In [45]:

```
smiles = [ 'C1=CC=CC=C1', # Benzene (Kekule)
           'c1ccccc1',    # Benzene ("Aromatized" carbons)
           'C1CCCCC1']     # Cyclohexene
```

In [46]:

```
# Write your code in this cell.
```

Write the fragment definition of the bits ON (one is already provided for you as an example).

- 118: ACH2CH2A > 1

1-(2) Circular Fingerprints

Circular fingerprints are hashed fingerprints. They are generated by exhaustively enumerating "circular" fragments (containing all atoms within a given radius from each heavy atom of the molecule) and then hashing these fragments into a fixed-length bitstring. (Here, the "radius" from an atom is measured by the number of bonds that separates two atoms).

Examples of circular fingerprints are the extended-connectivity fingerprint (ECFPs) and their variant called FCFPs (Functional-Class Fingerprints), originally described in a paper by Rogers and Hahn (https://doi.org/10.1021/ci100050t). The RDKit implementation of these fingerprints are called "Morgan Fingerprints" (https://www.rdkit.org/docs/GettingStartedInPython.html#morgan-fingerprints-circular-fingerprints).

In [47]:

```
from rdkit.Chem import AllChem
fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=1024).ToBitString()
print(fp)
```

```
010000000000000000000000000000100011000000000000000000100000000001000000000000000100000
```

When comparing the RDK's Morgan fingerprints with the ECFP/FCFP fingerprints, it is important to remember that the name of ECFP/FCFP fingerprints are suffixed with the **diameter** of the atom environments considered, while the Morgan Fingerprints take a **radius** parameter (e.g., the second argument "2" of GetMorganFingerprintAsBitVect() in the above code cell). The Morgan fingerprint generated above (with a radius of 2) is comparable to the ECFP4 fingerprint (with a diameter of 4).

Exercise 1b: For the moleculess below, generate the 512-bit-long Morgan Fingeprint comparable to the **FCFP6** fingerprint.

- Search for the compounds by name and get their SMILES strings.
- Generate the molecular fingerprints from the SMILES strings.
- Print the generated fingerprints.
- To generate FCFP (not ECFP), read the following document: https://www.rdkit.org/docs/GettingStartedInPython.html#morgan-fingerprints-circular-fingerprints

In [48]:

```
synonyms = [ 'diphenhydramine', 'cetirizine', 'fexofenadine', 'loratadine' ]
```

In [49]:

```
# Write your code in this cell
```

1-(3) Path-Based Fingerprints

Path-based fingerprints are also hashed fingerprints. They are generated by enumerating linear fragments of a given length and hashing them into a fixed-length bitstring. An example is the RDKit's topological fingeprint. As described in the RDK documentation (https://www.rdkit.org/docs/GettingStartedInPython.html#topological-fingerprints), while this fingerprint can be generated using FingerprintMols.FingerprintMol(), it is recommended to use rdmolops.RDKFingerprint() to generate the fingerprint using non-default parameter values.

In [50]:

```
from rdkit.Chem import rdmolops
fp = rdmolops.RDKFingerprint(mol, fpSize=2048, minPath=1, maxPath=7).ToBitString()
print(fp)
```

```
1101110111110101011111001011111010111101111010011111111111011100111111110110111101111
```

1-(4) PubChem Fingerprint

The PubChem Fingerprint is a 881-bit-long binary fingerprint (ftp://ftp.ncbi.nlm.nih.gov/pubchem/s...ngerprints.pdf). Similar to the MACCS keys, it uses a pre-defined fragment dictionary. The PubChem fingerprint for each compound in PubChem can be downloaded from PubChem. However, because they are base64-encoded, they should be decoded into binary bitstrings or bitvectors.

Details about how to decode base64-encoded PubChem fingerprints is described on the last page of the PubChem Fingerprint specification (ftp://ftp.ncbi.nlm.nih.gov/pubchem/s...ngerprints.pdf). Below is a user-defined function that decodes a PubChem fingerprint into a bit string.

In [51]:

```
from base64 import b64decode

def PCFP_BitString(pcfp_base64) :

    pcfp_bitstring = "".join( ["{:08b}".format(x) for x in b64decode( pcfp_base64 )]
    return pcfp_bitstring

```

In [52]:

```
pcfps = 'AAADcYBgAAAAAAAAAAAAAAAAAAAAAAAAAAAwAAAAAAAAAAAABAAAAGAAAAAAACACAEAAwAIAAAACA
```

In [53]:

```
print( len(PCFP_BitString(pcfps)) )
print(PCFP_BitString(pcfps))
```

```
881
1000000001100000000000000000000000000000000000000000000000000000000000000000000000000
```

The generated bitstring can be converted to a bitvector that can be used for molecular similarity computation in RDKit (to be discussed in the next section).

In [54]:

```
from rdkit import DataStructs
bitvect = DataStructs.CreateFromBitString(PCFP_BitString(pcfps))
type(bitvect)
```

Out[54]:

```
rdkit.DataStructs.cDataStructs.ExplicitBitVect
```

2. Computation of similarity scores

In [55]:

```
import requests
import time
```

In [56]:

```
cids = [    54454,  # Simvastatin (Zocor)
            54687,  # Pravastatin (Pravachol)
            60823,  # Atorvastatin (Lipitor)
           446155,  # Fluvastatin (Lescol)
           446157,  # Rosuvastatin (Crestor)
          5282452,  # Pitavastatin (Livalo)
         97938126 ] # Lovastatin (Altoprev)
```

Let's get the SMILES strings from PubChem, generate Mol objects from them, and draw their chemical structures.

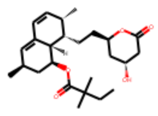
In [57]:

```
prolog = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

str_cid = ",".join([ str(x) for x in cids])

url = prolog + "/compound/cid/" + str_cid + "/property/isomericsmiles/txt"
res = requests.get(url)
smiles = res.text.split()
```

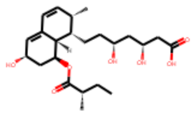
In [58]:

```
from rdkit import Chem
from rdkit.Chem import Draw

mols = [ Chem.MolFromSmiles(x) for x in smiles ]
Chem.Draw.MolsToGridImage(mols, molsPerRow=4, subImgSize=(200,200), legends=[str(x) f
```

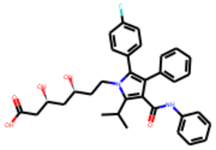
Out[58]:



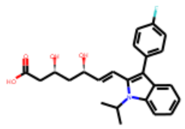
54454



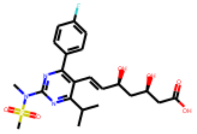
54687



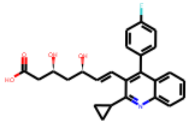
60823



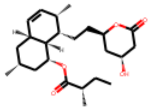
446155



446157



5282452



97938126

Now generate MACCS keys for each compound.

In [59]:

```
from rdkit import DataStructs
from rdkit.Chem import MACCSkeys

fps = [ MACCSkeys.GenMACCSKeys(x) for x in mols ]
```

Now let's compute the pair-wise similarity scores among them. To make higher scores easier to find, they are indicated with the "*" character(s).

In [60]:

```
for i in range(0, len(fps)) :
    for j in range(i+1, len(fps)) :

        score = DataStructs.FingerprintSimilarity(fps[i], fps[j])
        print(cids[i], "vs.", cids[j], ":", round(score,3), end='')

        if ( score >= 0.85 ):
            print(" ****")
        elif ( score >= 0.75 ):
            print(" ***")
        elif ( score >= 0.65 ):
            print(" **")
        elif ( score >= 0.55 ):
            print(" *")
        else:
            print(" ")
```

```
54454 vs. 54687 : 0.812 ***
54454 vs. 60823 : 0.354
54454 vs. 446155 : 0.379
54454 vs. 446157 : 0.307
54454 vs. 5282452 : 0.4
```

```
54454 vs. 97938126 : 0.867 ****
54687 vs. 60823 : 0.387
54687 vs. 446155 : 0.397
54687 vs. 446157 : 0.287
54687 vs. 5282452 : 0.421
54687 vs. 97938126 : 0.8 ***
60823 vs. 446155 : 0.662 **
60823 vs. 446157 : 0.535
60823 vs. 5282452 : 0.507
60823 vs. 97938126 : 0.365
446155 vs. 446157 : 0.526
446155 vs. 5282452 : 0.735 **
446155 vs. 97938126 : 0.393
446157 vs. 5282452 : 0.473
446157 vs. 97938126 : 0.298
5282452 vs. 97938126 : 0.444
```

By default, the similarity score is generated using the **Tanimoto** equation. RDKit also supports other similarity metrics, including Dice, Cosine, Sokal, Russel, Kulczynski, McConnaughey, and Tversky. The definition of these metrics is available at the LibreTexts page (https://bit.ly/2kx9NCd).

In [61]:

```
print("Tanimoto    :", round(DataStructs.TanimotoSimilarity(fps[0], fps[1]), 4))
print("Dice        :", round(DataStructs.DiceSimilarity(fps[0], fps[1]), 4))
print("Cosine      :", round(DataStructs.CosineSimilarity(fps[0], fps[1]), 4))
print("Sokal       :", round(DataStructs.SokalSimilarity(fps[0], fps[1]), 4))
print("McConnaughey:", round(DataStructs.McConnaugheySimilarity(fps[0], fps[1]), 4))
```

```
Tanimoto    : 0.8125
Dice        : 0.8966
Cosine      : 0.8971
Sokal       : 0.6842
McConnaughey: 0.7952
```

The Tversky score is an asymmetric similarity measure, and its computation requires the weightings of the two molecules being compared.

In [62]:

```
for i in range(0,11) :

    alpha = round( i * 0.1, 1 )
    beta = round( 1 - alpha, 1 )
    print("(alpha, beta) = (", alpha, ",", beta, ") : ", end='')
    print(round(DataStructs.TverskySimilarity( fps[0], fps[1], alpha, beta ), 4))
```

```
(alpha, beta) = ( 0.0 , 1.0 ) : 0.9286
(alpha, beta) = ( 0.1 , 0.9 ) : 0.922
```

```
(alpha, beta) = ( 0.2 , 0.8 ) : 0.9155
(alpha, beta) = ( 0.3 , 0.7 ) : 0.9091
(alpha, beta) = ( 0.4 , 0.6 ) : 0.9028
(alpha, beta) = ( 0.5 , 0.5 ) : 0.8966
(alpha, beta) = ( 0.6 , 0.4 ) : 0.8904
(alpha, beta) = ( 0.7 , 0.3 ) : 0.8844
(alpha, beta) = ( 0.8 , 0.2 ) : 0.8784
(alpha, beta) = ( 0.9 , 0.1 ) : 0.8725
(alpha, beta) = ( 1.0 , 0.0 ) : 0.8667
```

Exercise 2a: Compute the Tanimoto similarity scores between the seven compounds used in this section, using the PubChem fingerprints

- Download the PubChem Fingerprint for the seven CIDs.
- Convert the downloaded fingerprints into bit vectors.
- Compute the pair-wise Tanimoto scores using the bit vectors.

In [63]:

```
# Write your code in this cell
```

3. Interpretation of similarity scores

Using molecular fingeprints. we can compute the similarity scores between molecules. However, how should these scores be interpreted? For example, the Tanimoto score between CID 60823 and CID 446155 is computed to be 0.662, but does it mean that the two compounds are similar? How similar is similar? The following analysis would help answer these questions.

Step 1. Randomly select 1,000 compounds from PubChem and download their SMILES strings.

In [64]:

```
import random
random.seed(0)

cid_max = 138962044    # The maximum CID in PubChem as of September 2019

cids = []

for x in range(1000):
    cids.append(random.randint(1, cid_max + 1))

chunk_size = 100

if len(cids) % chunk_size == 0 :
    num_chunks = int( len(cids) / chunk_size )
else :
    num_chunks = int( len(cids) / chunk_size ) + 1

smiles = []

for i in range(num_chunks):
    if (i == 0):
        print("Processing chunk ", end='')

    print(i, end=' ')

    idx1 = chunk_size * i
    idx2 = chunk_size * (i + 1)
    str_cids = ",".join([ str(x) for x in cids[idx1:idx2]])

    url = prolog + "/compound/cid/" + str_cids + "/property/isomericsmiles/txt"
    res = requests.get(url)

    if ( res.status_code == 200) :
        smiles.extend( res.text.split() )
    else :
        print("Chunk", i, "Failed to get SMILES.")

    time.sleep(0.2)

print("Done!")
print("# Number of SMILES : ", len(smiles))
```

```
Processing chunk 0 1 2 3 4 5 6 7 8 9 Done!
# Number of SMILES :  1000
```

Step 2. Generate the MACCSKeys for each compound.

In [65]:

```
from rdkit import Chem

mols = [ Chem.MolFromSmiles(x) for x in smiles if x != None ]
fps  = [ MACCSkeys.GenMACCSKeys(x) for x in mols if x != None ]
print("# Number of compounds:", len(mols))
print("# Number of fingerprints:", len(fps))
```

```
# Number of compounds: 1000
# Number of fingerprints: 1000
```

In [66]:

```
# Run this cell if the number of compounds != the number of fingerprints.
#if ( len(cids) != len(fps) ):
#    print("SMILES at index", mols.index(None), ":", smiles[ mols.index(None) ])
```

Step 3. Compute the Tanimoto scores between compounds.

In [67]:

```
print("# The number of compound pairs:", (len(fps) * (len(fps) - 1))/2 )
```

```
# The number of compound pairs: 499500.0
```

In [68]:

```
scores = []

for i in range(0, len(fps)) :

    if (i == 0) :
        print("Processing compound ", end='')

    if (i % 100 == 0) :
        print(i, end=' ')

    for j in range(i+1, len(fps)) :
        scores.append(DataStructs.FingerprintSimilarity(fps[i], fps[j]))

print("Done!")
print("# Number of scores : ", len(scores))
```

```
Processing compound 0 100 200 300 400 500 600 700 800 900 Done!
# Number of scores :  499500
```

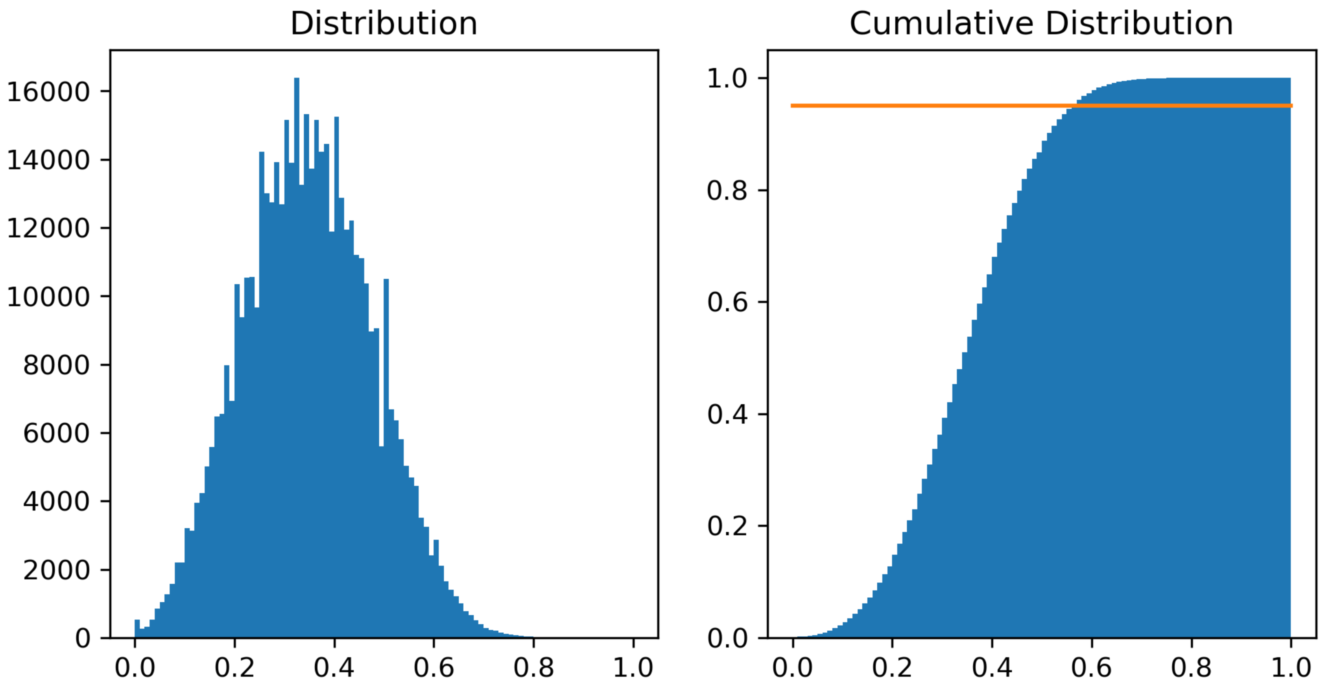
Step 4. Generate a histogram that shows the distribution of the pair-wise scores.

In [69]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [70]:

```
mybins = [ x * 0.01 for x in range(101)]

fig = plt.figure(figsize=(8,4), dpi=300)

plt.subplot(1, 2, 1)
plt.title("Distribution")
plt.hist(scores, bins=mybins)

plt.subplot(1, 2, 2)
plt.title("Cumulative Distribution")
plt.hist(scores, bins=mybins, density=True, cumulative=1)
plt.plot([0,1],[0.95,0.95]);
```



In [71]:

```
for i in range(21) :

    thresh = i / 20
    num_similar_pairs = len([x for x in scores if x >= thresh])
    prob = num_similar_pairs / len(scores) * 100
    print("%.3f %8d (%8.4f %%)" % (thresh, num_similar_pairs, round(prob,4)))
```

```
0.000   499500 (100.0000 %)
0.050   497010 ( 99.5015 %)
0.100   488714 ( 97.8406 %)
0.150   469193 ( 93.9325 %)
0.200   435672 ( 87.2216 %)
0.250   385198 ( 77.1167 %)
0.300   318637 ( 63.7912 %)
0.350   245621 ( 49.1734 %)
0.400   175158 ( 35.0667 %)
0.450   111668 ( 22.3560 %)
0.500    66599 ( 13.3331 %)
0.550    32209 (  6.4482 %)
0.600    13897 (  2.7822 %)
0.650     4668 (  0.9345 %)
0.700     1351 (  0.2705 %)
0.750      355 (  0.0711 %)
0.800       76 (  0.0152 %)
0.850       24 (  0.0048 %)
0.900        6 (  0.0012 %)
```

```
0.950         1 (  0.0002 %)
1.000         0 (  0.0000 %)
```

In [72]:

```
print("Average:", sum(scores)/len(scores))
```

```
Average: 0.3488195436976387
```

From the distribution of the similarity scores among 1,000 compounds, we observe the following:

- If you randomly select two compounds from PubChem, the similarity score between them (computed using the Tanimoto equation and MACCS keys) is ~0.35 on average.
- About %5 of randomly selected compound pairs have a similarity score greater than 0.55.
- About %1 of randomly selected compound pairs have a similarity score greater than 0.65.

If two compounds have a Tanimoto score of 0.35, it is close to the avaerage Tanimoto score between randomly selected compounds and there is a 50% chance that you will get a score of 0.35 or greater just by selecting two compounds from PubChem. Therefore, it is reasonable to consider the two compounds are not similar.

The Tanimoto index may have a value ranging from 0 (for no similarity) to 1 (for identical molecules) and the midpoint of this value range is 0.5. Because of this, a Tanimoto score of **0.55** may not sound great enough to consider two compounds to be similar. However, according to the score distribution curve generated here, only **~5%** of randomly selected compound pairs will have a score greater than this.

In the previous section, we computed the similarity scores between some cholesterol-lowering drugs, and CID 60823 and CID 446155 had a Tanimoto score of **0.662**. Based on the score distribution curve generated in the second section, we can say that the probablilty of two randomly selected compounds from PubChem having a Tanimoto score greater than 0.662 is **less than 1%**.

The following code cell demonstrates how to find an appropriate similarity score threshold above which a given percentage of the compound pairs will be considered to be similar to each other.

In [73]:

```
scores.sort()    # Sort the scores in an increasing order.
```

In [74]:

```
# to find a threshold for top 3% compound pairs (i.e., 97% percentile)
print("# total compound pairs:   ", len(scores))
print("# 95% of compound pairs:  ", len(scores) * 0.97)
print("# score at 95% percentile:", scores[ round(len(scores) * 0.97) ] )
```

```
# total compound pairs:    499500
# 95% of compound pairs:   484515.0
# score at 95% percentile: 0.5945945945945946
```

Exercise 3a: In this exercise, we want to generate the distribution of the similarity scores among 1,000 compounds randomly selected from PubChem, using different molecular fingeprints and similarity metrics.
For molecular fingerprints, use the following:

- PubChem Fingerprint
- MACCS keys
- Morgan Fingerprint (ECFP4 analogue, 1024-bit-long)

For similarity metrics, use the following:

- Tanimoto similarity
- Dice similarity
- Cosine similarity

As a result, a total of 9 distribution curves need to be generated.

Here are additional instructions to follow:

- When generating the histograms, bin the scores from 0 to 1 with an increment of 0.01.
- For each distribution curve, determine the similarity score threshold so that **1%** of the compound pairs have a similarity score greater than or equal to this threshold.
- Use RDKit to generate the MACCS keys and Morgan fingerprint and download the PubChem fingerprints from PubChem.
- For reproducibility, use **random.seed(2019)** before you generate random CIDs.

Step 1: Generate 1,000 random CIDs, download the isomeric SMILES for them, and create the RDKit mol objects from the downloaded SMILES strings.

In [75]:

```
# Write your code in this cell
```

Step 2: Generate the fingerprints, compute the similarity scores, determine similarity thresholds, and make histograms.

In [76]:

```
# Write your code in this cell
```

In [ ]:

```

```

6.5: R Assignment

Molecular Similarity

S. Kim, J. Cuadros

November 23rd, 2019

Objectives

- Generate molecular fingerprints for a given molecule.
- Evaluate structural similarity between molecules using different molecular fingerpints and similarity metrics.

Many useful documents/papers describe various aspects of molecular similarity, including molecular fingerprints and similarity measures. Please read these if you need more details.

- Getting Started with the RDKit in Python (https://www.rdkit.org/docs/GettingStartedInPython.html#fingerprinting-and-molecular-similarity)
- Fingerprint Generation, GraphSim Toolkit 2.4.2 (https://docs.eyesopen.com/toolkits/python/graphsimtk/fingerprint.html)
- Chemical Fingerprints (https://docs.chemaxon.com/display/docs/Chemical+Fingerprints)
- Extended-Connectivity Fingerprints (https://doi.org/10.1021/ci100050t)

1. Fingerprint Generation

```
if (!require("rcdk", quietly=TRUE)) {
 install.packages("rcdk", repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("rcdk")
}
if (!require("fingerprint", quietly=TRUE)) {
 install.packages("fingerprint", repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("fingerprint")
}
```

```
smi <- "CC(C)C1=C(C(=C(N1CC[C@H](C[C@H](CC(=O)O)O)O)C2=CC=C(C=C2)F)C3=CC=CC=C3)C(=O)NC
mol <- parse.smiles(smi)
```

1-(1) MACCS keys

The MACCS key is a binary fingerprint (a string of 0’s and 1’s). Each bit position represents the presence (=1) or absence (=0) of a pre-defined structural feature. The feature definitions for the MACCS keys are available at:

- https://github.com/rdkit/rdkit/blob/master/rdkit/Chem/MACCSkeys.py
- http://www.mayachemtools.org/docs/modules/html/MACCSKeys.html

```
fp <- get.fingerprint(mol[[1]], type = 'maccs',
 fp.mode = 'bit', verbose=FALSE)
str(fp)
```

```
 as.character(fp)
```

```
 nchar(as.character(fp))
```

These are some methods that allow you to get some additional information on the fingerprint.

```
print(paste("Number of bits:", length(fp)))
```

```
 print(paste("Number of ON bits:", length(fp@bits)))
```

```
 print(paste("Number of OFF bits:", fp@nbit-length(fp@bits)))
```

```
 as.character(fp)
```

```
 (fp_bin <- unlist(strsplit(as.character(fp),"")))
```

```
 paste(fp_bin, collapse="")
```

```
 fp_bytes <- substring(paste("00", as.character(fp), sep=""),
 seq(1, length(fp)+2, 8), seq(8, length(fp)+2, 8))
# bits in bytes are read right to left, https://code.google.com/archive/p/chem-finger
fp_bytes <- sapply(fp_bytes, function(x)
 paste(rev(strsplit(x,"")[[1]]),collapse=""))

(fp_bytes <- strtoi(fp_bytes,base=2))
```

```
(fp_hex <- as.raw(fp_bytes))
```

```
 paste(fp_hex, collapse = "")
```

```
 fp_bin2 <- unlist(lapply(paste("0x",fp_hex,sep=""),function(x) rawToBits(as.raw(x)))
fp_bin2 <- as.numeric(fp_bin2)
substring(paste(fp_bin2, collapse=""),3,168)
```

Exercise 1a: Generate the MACCS keys for the molecules represented by the following SMILES, and get the positions of the bits set to ON in each of the three fingerprints. What fragments do these bit positions correspond to?

```
smiles <- c('C1=CC=CC=C1', # Benzene (Kekule)
 'c1ccccc1', # Benzene ("Aromatized" carbons)
 'C1CCCCC1') # Cyclohexene
```

```
 # Write your code here
```

Write the fragment definition of the bits ON (one is already provided for you as an example). - 118: ACH2CH2A > 1

```

```

1-(2) Circular Fingerprints

Circular fingerprints are hashed fingerprints. They are generated by exhaustively enumerating “circular” fragments (containing all atoms within a given radius from each heavy atom of the molecule) and then hashing these fragments into a fixed-length bitstring. (Here, the “radius” from an atom is measured by the number of bonds that separates two atoms).

Examples of circular fingerprints are the extended-connectivity fingerprint (ECFPs) and their variant called FCFPs (Functional-Class Fingerprints), originally described in a paper by Rogers and Hahn (https://doi.org/10.1021/ci100050t). Sometimes, for instance in RDKit, these fingerprints are called “Morgan Fingerprints” (https://www.rdkit.org/docs/GettingStartedInPython.html#morganfingerprints-circular-fingerprints).

CDK can compute a ECFP6 fingerprint.

```
fp <- get.fingerprint(mol[[1]], type = 'circular',
 fp.mode = 'bit', verbose=FALSE)
as.character(fp)
```

Morgan fingerprints can be obtained from the ChEMBL webservice, which is based on RDKit.

```
if(!require("httr")) {
 install.packages(("httr"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("httr")
}
```

```
sdf <- readLines(paste("https://cactus.nci.nih.gov/chemical/structure/",
 URLencode(smi,reserved = T), "/SDF", sep=""))
sdf <- paste(sdf, collapse="\n")
url <- paste("https://www.ebi.ac.uk/chembl/api/utils/sdf2fps",
 "?n_bits=1024&radius=2",sep="")
res <- POST(url,
 body=sdf)
response <- rawToChar(res$content)
fp_hex <- strsplit(response, "\n")[[1]][4]
(fp_hex <- strsplit(fp_hex, "\t")[[1]][1])
```

```
 fp_hex <- substring(fp_hex, seq(1,nchar(fp_hex),2), seq(2,nchar(fp_hex),2))
fp_bin <- unlist(lapply(paste("0x",fp_hex,sep=""),function(x) rawToBits(as.raw(x))))
(fp_bin <- as.numeric(fp_bin))
```

```
 (fp <- paste(fp_bin, collapse=""))
```

```
 # Using the tools of the fingerprint package
fp_obj <- fps.lf(strsplit(response, "\n")[[1]][4])
fp <- new("fingerprint", nbit = 1024,
 bits = as.numeric(fp_obj[[2]]),
 folded = FALSE,
 provider = gsub("#software=","",
 strsplit(response, "\n")[[1]][3], fixed=T),
 name = fp_obj[[1]],
 misc = list())
str(fp)
```

```
as.character(fp)
```

When comparing the RDKit's Morgan fingerprints with the ECFP/FCFP fingerprints, it is important to remember that the name of ECFP/FCFP fingerprints are suffixed with the diameter of the atom environments considered, while the Morgan Fingerprints take a radius parameter (e.g., the second argument "2" of GetMorganFingerprintAsBitVect() in the above code cell). The Morgan fingerprint generated above (with a radius of 2) is comparable to the ECFP4 fingerprint (with a diameter of 4).

Exercise 1b: For the moleculess below, generate the 512-bit-long Morgan Fingeprint.

- Search for the compounds by name and get their SMILES strings.
- Generate the molecular fingerprints from the SMILES strings.
- Print the generated fingerprints.

```
 synonyms <- c('diphenhydramine', 'cetirizine', 'fexofenadine', 'loratadine')
```

```
# Write your code here
```

1-(3) Path-Based Fingerprints

Path-based fingerprints are also hashed fingerprints. They are generated by enumerating linear fragments of a given length and hashing them into a fixed-length bitstring. An example of this, is the standard fingerprint in CDK. Another example is the RDKit's topological fingerprint.

In CDK, size and depth allow specifying fingerprint and maximum path size, used for constructing the fingerprint. They default to 1024 bits and depth 6.

```
fp <- get.fingerprint(mol[[1]], type = 'standard',
 fp.mode = 'bit', size = 128, depth= 4, verbose=FALSE)
str(fp)
```

```
as.character(fp)
```

```
 nchar(as.character(fp))
```

```
 length(fp@bits)/length(fp)
```

```
 fp <- get.fingerprint(mol[[1]], type = 'standard',
 fp.mode = 'bit', size = 2048, depth= 7, verbose=FALSE)
str(fp)
```

```
 as.character(fp)
```

```
 nchar(as.character(fp))
 length(fp@bits)/length(fp)
```

1-(4) PubChem Fingerprint

The PubChem Fingerprint is a 881-bit-long binary fingerprint (ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf). Similar to the MACCS keys, it uses a pre-defined fragment dictionary. The PubChem fingerprint for each compound in PubChem can be downloaded from PubChem. However, because they are base64-encoded, they should be decoded into binary bitstrings or bitvectors.

Details about how to decode base64-encoded PubChem fingerprints is described on the last page of the PubChem Fingerprint specification (ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf). Below is a user-defined function that decodes a PubChem fingerprint into a bit string.

```
if(!require("jsonlite")) {
 install.packages(("jsonlite"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("jsonlite")
}
```

```
 pcfps <- 'AAADcYBgAAAAAAAAAAAAAAAAAAAAAAAAAAAAwAAAAAAAAAAAABAAAAGAAAAAAACACAEAAwAIAAAA
CAAAgAAAIiAAAAIgIICKAERCAIAAggAAIiAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=='
pcfps_raw <- base64_dec(pcfps)
rawToBits(base64_dec("AAAD"))
```

```
 sapply(base64_dec("AAAD"),function(x) paste(as.numeric(rawToBits(x)),collapse=""))
```

```
 pcfps_bin <- sapply(pcfps_raw,
 function(x) paste(as.numeric(rev(rawToBits(x))),collapse=""))
pcfps_bin <- substring(paste(pcfps_bin,collapse=""),33,913)
nchar(pcfps_bin)
```

```
 pcfps_bin
```

The generated bitstring can be converted to a bitvector that can be used for molecular similarity computations (to be discussed in the next section).

```
(binvect <- as.numeric(unlist(strsplit(pcfps_bin,""))))
```

2. Computation of similarity scores

```
cids <- c(54454, # Simvastatin (Zocor)
          54687, # Pravastatin (Pravachol)
          60823, # Atorvastatin (Lipitor)
          446155, # Fluvastatin (Lescol)
          446157, # Rosuvastatin (Crestor)
          5282452, # Pitavastatin (Livalo)
          97938126) # Lovastatin (Altoprev)
```

Let’s get the SMILES strings from PubChem, generate Mol objects from them, and draw their chemical structures.

```
prolog <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
str_cid <- paste(as.character(cids), collapse=",")
url <- paste(prolog, "/compound/cid/", str_cid, "/property/isomericsmiles/txt", sep="
smiles <- readLines(url)
```

```
if(!require("httr")) {
 install.packages(("httr"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("httr")
}
if(!require("jsonlite")) {
 install.packages(("jsonlite"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("jsonlite")
}
if(!require("png")) {
 install.packages(("png"), repos="https://cloud.r-project.org/",#
 quiet=TRUE, type="binary")
 library("png")
}
if(!require("grid")) {
 install.packages(("grid"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("grid")
}
if(!require("gridExtra")) {
 install.packages(("gridExtra"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("gridExtra")
}
```

```
url_img <- paste("https://www.ebi.ac.uk/chembl/api/utils/smiles2image",
 "?size=300&engine=rdkit",sep="")
res <- POST(url_img,
 body=list(smiles=paste(smiles,collapse="\n")))
img <- readPNG(res$content, native=TRUE)
grid.arrange(rasterGrob(img))
```

Now generate MACCS keys for each compound.

```
mols <- parse.smiles(smiles)
fps <- sapply(mols,
 function(x) get.fingerprint(x, type="maccs"))
fps_bin <- sapply(fps, as.character)
fps_bin
```

Now let’s compute the pair-wise similarity scores among them. To make higher scores easier to find, they are indicated with the “*” character(s).

```
 dfDistance <- data.frame(cid1 = numeric(0), cid2 = numeric(0),
 score = numeric(0))

for(i in 1:(length(cids)-1)) {
 for(j in (i+1):length(cids)) {
 score <- distance(fps[[i]], fps[[j]], method="tanimoto")
 dfDistance <- rbind(dfDistance,
 list(cid1 = cids[i], cid2 = cids[j],
 score = score))
 }
}

dfDistance$value <- ""
dfDistance$value[dfDistance$score >= .55] <- "*"
dfDistance$value[dfDistance$score >= .65] <- "**"
dfDistance$value[dfDistance$score >= .75] <- "***"
dfDistance$value[dfDistance$score >= .85] <- "****"

dfDistance
```

By default, the similarity score is generated using the Tanimoto equation. fingerprint::distance also supports other similarity metrics, including Dice, Cosine, Russel, SOkal-Michener (also known as simple matching), Kulczynski, McConnaughey, and Tversky. The definition of these metrics is available at the LibreTexts page (https://bit.ly/2kx9NCd).

```
print(paste("Tanimoto: ",
 distance(fps[[1]], fps[[2]], method="tanimoto")))
```

```
print(paste("Dice: ",
 distance(fps[[1]], fps[[2]], method="dice")))
```

```
 print(paste("Cosine: ",
 distance(fps[[1]], fps[[2]], method="cosine")))
```

```
 print(paste("Simple: ",
 distance(fps[[1]], fps[[2]], method="simple")))
```

```
print(paste("McConnaughey: ",
 distance(fps[[1]], fps[[2]], method="mcconnaughey")))
```

```
contTable <- as.matrix(table(seq(length(fps[[1]])) %in% fps[[1]]@bits,
 seq(length(fps[[2]])) %in% fps[[2]]@bits))
a <- contTable[2,2]
```

```
b <- contTable[2,1]
c <- contTable[1,2]
d <- contTable[1,1]
dist <- (a^2 - b * c)/((a + b) * (a + c))

print(paste("McConnaughey: ",
 dist))
```

The Tversky score is an asymmetric similarity measure, and its computation requires the weightings of the two molecules being compared.

```
for(i in 0:10) {
 print(paste("Tversky (alpha = ", i * 0.1, ", beta = ", 1-i * .1, ") = ",
 distance(fps[[1]], fps[[2]], a= i * .1, b = 1 - i * .1,
 method="tversky"),
 sep = ""))
}
```

Exercise 2a: Compute the Tanimoto similarity scores between the seven compounds used in this section, using the PubChem fingerprints

- Download the PubChem Fingerprint for the seven CIDs.
- Convert the downloaded fingerprints into bit vectors.
- Compute the pair-wise Tanimoto scores using the bit vectors

```
# Write your code here
```

3. Interpretation of similarity scores

Using molecular fingeprints. we can compute the similarity scores between molecules. However, how should these scores be interpreted? For example, the Tanimoto score between CID 60823 and CID 446155 is computed to be 0.662, but does it mean that the two compounds are similar? How similar is similar? The following analysis would help answer these questions.

Step 1. Randomly select 1,000 compounds from PubChem and download their SMILES strings.

```
prolog <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
set.seed(0)
cid_max <- 138962044 # The maximum CID in PubChem as of September 2019

cids <- sample(seq(cid_max),1000)

chunk_size <- 100
num_chunks <- ceiling(length(cids) / chunk_size)

smiles = character(length(cids))

for(i in seq(num_chunks)) {
 print(i)

 idx1 <- chunk_size * (i - 1) + 1
 idx2 <- chunk_size * i
 str_cids <- paste(cids[idx1:idx2], collapse=",")
 url <- paste(prolog, "/compound/cid/", str_cids, "/property/isomericsmiles/txt", sep
 smiles[idx1:idx2] <- readLines(url)

 Sys.sleep(0.5)
}
print("Done!")
```

```
 print(paste("# Number of SMILES :", length(na.omit(smiles))))
```

Step 2. Generate the MACCSKeys for each compound.

```
mols <- parse.smiles(smiles)
fps <- lapply(mols, function(x) get.fingerprint(x, type = 'maccs',
 fp.mode = 'bit', verbose=FALSE))

str(fps[[1]])
```

```
print(paste("Number of compounds:", length(mols)))
```

```
 print(paste("Number of Fingerprints:", length(fps)))
```

Step 3. Compute the Tanimoto scores between compounds.

```
print(paste("Number of compound pairs:", (length(fps) * (length(fps) - 1))/2))
```

```
 scores <- numeric((length(fps) * (length(fps) - 1))/2)

k <- 1
for(i in 1:(length(fps)-1)) {
 for(j in (i+1):length(fps)) {
 scores[k] <- distance(fps[[i]], fps[[j]], method="tanimoto")
 k <- k + 1
 }
}

summary(scores)
```

Step 4. Generate a histogram that shows the distribution of the pair-wise scores.

```
if(!require("tidyverse")) {
 install.packages(("tidyverse"), repos="https://cloud.r-project.org/",
 quiet=TRUE, type="binary")
 library("tidyverse")
}
```

```
 ggplot(NULL, aes(x=scores, y=..density..)) +
 geom_histogram(fill="lightgrey", color="black", binwidth=.05) +
 geom_density()
```

```
dfScoresTab <- data.frame(limit = 0:20*0.05,
 countLT = sapply(0:20*0.05, function(x) sum(scores>=x)))
dfScoresTab$propLT <- dfScoresTab$countLT / length(scores)
dfScoresTab
```

```
 print(paste("Average:", sum(scores)/length(scores)))
```

From the distribution of the similarity scores among 1,000 compounds, we observe the following:

- If you randomly select two compounds from PubChem, the similarity score between them (computed using the Tanimoto equation and MACCS keys) is ~0.35 on average.
- About %5 of randomly selected compound pairs have a similarity score greater than 0.55.
- About %1 of randomly selected compound pairs have a similarity score greater than 0.65.

If two compounds have a Tanimoto score of 0.35, it is close to the avaerage Tanimoto score between randomly selected compounds and there is a 50% chance that you will get a score of 0.35 or greater just by selecting two compounds from PubChem. Therefore, it is reasonable to consider the two compounds are not similar.

The Tanimoto index may have a value ranging from 0 (for no similarity) to 1 (for identical molecules) and the midpoint of this value range is 0.5. Because of this, a Tanimoto score of 0.55 may not sound great enough to consider two compounds to be similar. However, according to the score distribution curve generated here, only ~5% of randomly selected compound pairs will have a score greater than this.

In the previous section, we computed the similarity scores between some cholesterol-lowering drugs, and CID 60823 and CID 446155 had a Tanimoto score of 0.662. Based on the score distribution curve generated in the second section, we can say that the probablilty of two randomly selected compounds from PubChem having a Tanimoto score greater than 0.662 is less than 1%.

The following code cell demonstrates how to find an appropriate similarity score threshold above which a given percentage of the compound pairs will be considered to be similar to each other.

```
# to find a threshold for top 3% compound pairs (i.e., 97% percentile)
quantile(scores,.97)
```

Exercise 3a: In this exercise, we want to generate the distribution of the similarity scores among 1,000 compounds randomly selected from PubChem, using different molecular fingeprints and similarity metrics. For molecular fingerprints, use the following: - PubChem Fingerprint - MACCS keys - Morgan Fingerprint (ECFP4 analogue, 1024-bit-long)

For similarity metrics, use the following: - Tanimoto similarity - Dice similarity - Cosine similarity

As a result, a total of 9 distribution curves need to be generated.

Here are additional instructions to follow: - When generating the histograms, bin the scores from 0 to 1 with an increment of 0.01. - For each distribution curve, determine the similarity score threshold so that 1% of the compound pairs have a similarity score greater than or equal to this threshold. - Use RDKit to generate the MACCS keys and Morgan fingerprint and download the PubChem fingerprints from PubChem. - For reproducibility, use random.seed(2019) before you generate random CIDs.

Step 1: Generate 1,000 random CIDs, download the isomeric SMILES for them, and create the RDKit mol objects from the downloaded SMILES strings.

```
# Write your code here
```

Step 2: Generate the fingerprints, compute the similarity scores, determine similarity thresholds, and make histograms.

```
# Write your code here
```

6.6: Mathematica Assignment

CHAPTER OVERVIEW

7: Computer-Aided Drug Discovery and Design

Hypothes.is Tag= f19OLCCc7

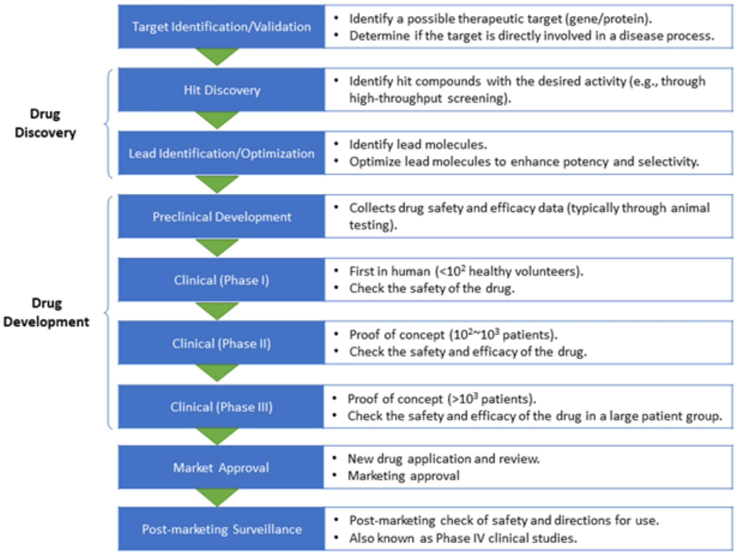
Note: Any annotation tagged **f19OLCCc7** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

7.1: Reading

7.1. Drug discovery & development process

The drug discovery and development process is very resource-intensive and time-consuming. Bringing a new drug to market (from drug discovery through clinical trials to marketing approval) typically takes between 10 and 15 years and costs $1.395 billion (2013 dollars) on average [1]. Figure 1 shows a schematic diagram of the drug discovery and development process.


Target Identification/Validation
• Identify a possible therapeutic target (gene/protein).
• Determine if the target is directly involved in a disease process.
Drug Discovery
Hit Discovery
• Identify hit compounds with the desired activity (e.g., through high-throughput screening).
Lead Identification/Optimization
• Identify lead molecules.
• Optimize lead molecules to enhance potency and selectivity.
Drug Development
Preclinical Development
• Collects drug safety and efficacy data (typically through animal testing).
Clinical (Phase I)
• First in human (<10^2 healthy volunteers).
• Check the safety of the drug.
Clinical (Phase II)
• Proof of concept (10^2~10^3 patients).
• Check the safety and efficacy of the drug.
Clinical (Phase III)
• Proof of concept (>10^3 patients).
• Check the safety and efficacy of the drug in a large patient group.
Market Approval
• New drug application and review.
• Marketing approval
Post-marketing Surveillance
• Post-marketing check of safety and directions for use.
• Also known as Phase IV clinical studies.

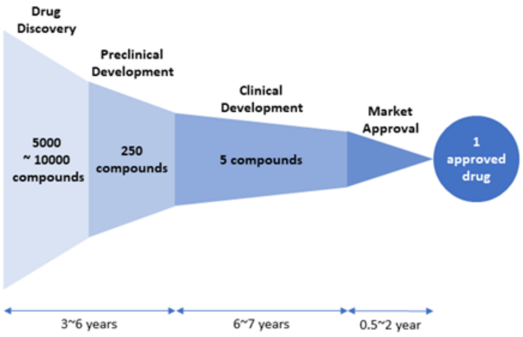
Copy and Paste Caption here

Figure 1. Drug discovery and development process.

The process presented in Figure 1 is simplified, ignoring many details within each stage. In reality, the drug discovery and development process is much more complicated, as illustrated in this Figure (https://www.nature.com/articles/nrd.2017.217/figures/1) [2].

This chapter describes some computational approaches used in the drug discovery stage. To further discuss computer-aided (or computer-assisted) drug discovery, it is necessary to learn the following commonly used terms.

- **Actives**
  Substances that meet a threshold level of activity in a primary screen, which typically measures the activity of compounds against the target at a single concentration. Because the activity was measured only at a single concentration, it is not possible to tell whether a compound can interact with the target in a dose-response way. Often the structure and purity of screening substances are not confirmed.
  - **Hits**
    Hits are compounds with intrinsic activity (IC50, EC50, etc.) against the target and they are characterized through secondary assays (which measure compounds' activity at multiple concentrations). In general, because hits have limited potency and/or selectivity, they are not suitable for in vivo studies (in animals or humans). However, they provide a starting point for structure activity relationship (SAR) analysis to help improve potency/selectivity and other drug properties.
- **Leads**
  A Lead represents a compound series that shows a relationship between chemical structure and target-based activity (in biochemical and cell-based assays). Compounds within the series have physicochemical properties, potency and selectivity deemed appropriate for in vivo evaluation.
- **Drug candidates**
  Compounds with strong therapeutic potential and whose activity and specificity have been optimized through the lead optimization step. These compounds move to the preclinical development stage for in vivo animal testing.



Copy and Paste Caption here

Figure 2. Drug attrition rate. Of 5,000–10,000 compounds experimentally screened in the drug discovery stage, approximately 250 will enter preclinical testing and 5 will enter clinical testing. Only one compound of them will be approved. [Adapted from "New Drug Approvals in 2011" (http://phrma-docs.phrma.org/sites/default/files/pdf/nda2011.pdf)].

7.2. What is virtual screening?

Virtual screening (VS) is a computational technique used in drug discovery to prioritize the compound selection from a large compound library for subsequent experimental assays. VS aims to ensure that those molecules with the largest a priori probabilities of being active against the target (protein/gene/disease) are tested first in a lead discovery program. VS is a cost-effective approach that complements high-throughput screening and it is routinely used in drug discovery projects.

Depending on the information available about the target and/or its ligands at the beginning of the VS campaign, VS can be broadly divided into two main approaches: structure-based and ligand-based approaches. The structure-based approaches, such as molecular docking, use the 3-D structure of the target macromolecule (protein/gene) to dock the candidate molecules and rank them based on their predicted binding affinity or complementarity to the binding site. On the other hand, the ligand-based approaches use a set of known actives against the target to identify database compounds that are likely to be active based on similarity/commonality between the known actives and database compounds. Because the structure-based approaches require the 3-D structural information of the target macromolecules, they are not a feasible option when the 3-D structure of the target is not available. In contrast, the ligand-based approaches can be used regardless of whether the target 3-D structure is available or not.

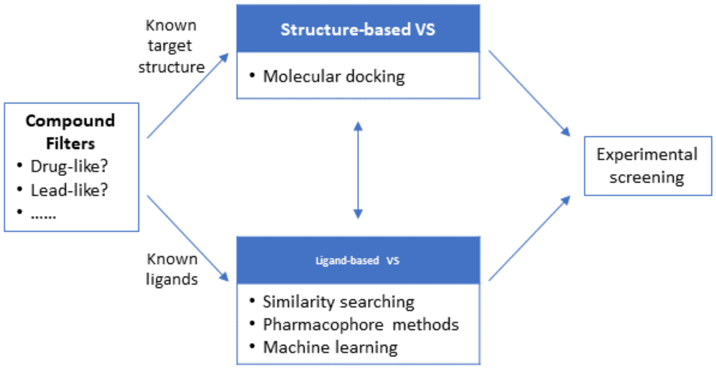


Figure 3. A schematic diagram of the virtual screening process, which involves many computational approaches to prioritize the compound selection for further experiments

.

7.3. Compound Filters

When screening a large compound library for drug discovery, it is desirable to identify problematic compounds that are not likely to lead to successful drug discovery campaign and exclude them from further computational or experimental screening. For example, if a molecule is *not* absorbed very well by the human body, that molecule may not be a good candidate for an orally administered drug, even if it shows a good activity against the target protein in *in vitro* testing. Therefore, one may want to exclude this compound from further testing.

Lipinski's rule of five (also known as the Pfizer's rule of five) (RO5) [3] is a rule of thumb to evaluate “drug-likeness” and helps determine whether a compound has good solubility and permeability that would make the compound a likely orally administered drug in human. According to RO5, drug-like compounds have:

- No more than 5 hydrogen-bond donors
- No more than 10 hydrogen-bond acceptors
- Molecular weight of 500 or less
- Calculated logP (CLogP) less than 5

Lipinski and coworkers [3] analyzed 2245 compounds that had reached phase II trials or higher and formulated RO5 based on the observation that most orally administered drugs are relatively small and moderately lipophilic.

While RO5 provides a fast and simple way to estimate oral bioavailability of molecules, it is a rule of thumb, not accurate and hard-set criteria. For example, ~16% of orally administrated drugs violate at least one criterion of RO5 [4]. For this reason, several variants of RO5 have been proposed as explained in detail in a recent review article [5].

An interesting concept related to drug-likeness is “lead-likeness”[6-9]. In the analysis of 470 lead-drug pairs, Hann et al [6]. found that, on average, lead compounds had lower molecular weight and logP and fewer hydrogen bond acceptors and more hydrogen bond donors, compared to the respective drugs developed from them. Therefore, it can be problematic to apply drug-likeness filters to compound libraries designed for lead discovery. This led to the development of “lead-likeness” filters. An example is Congreve’s rule of 3 (RO3) for lead-like compounds [10], which include:

- The number of hydrogen bond donors ≤3
- The number of hydrogen bond acceptors ≤ 3
- Molecular weight < 300
- ClogP ≤ 3

In general, the lead optimization step in drug discovery involves the modification of the lead molecule to give the final drug and this typically increases the molecular “complexity”, as reflected in the differences between RO3 and RO5. For this reason, some people argue that lead-like filters, rather than drug-like filters, should be used when performing virtual screening [6,7]. More detailed discussion about drug-likeness and lead-likeness can be found elsewhere [6-9].

7.4. Structure-based virtual screening

Structural-based approaches, exemplified by molecular docking, require the 3-D structure of the target macromolecule, which can be either experimentally determined through X-ray crystallography or computationally predicted through homology modeling. In molecular docking, the most likely binding mode for each compound is identified and assign a priority order to the molecules. While a large number of molecular docking methods have been proposed, all of them have two essential components to solve the molecular docking problem:

· **docking algorithm**, used to possible protein-ligand geometries (also called “poses”), and

· **scoring function**, which is a mathematical formula used to score or rank these poses.

Docking algorithms

Molecular docking methods can be classified into rigid-body docking and flexible docking, depending on the degree to which the flexibility of ligands and their macromolecule target is considered during the docking process. The rigid-body docking uses “fixed” structures of the ligands and target, treating them as rigid bodies. While the earliest docking programs used the rigid-body docking approach, more recent programs can explore the conformational space of the ligands (by changing the torsional angles of the ligands during the docking process). Some programs also permit conformational flexibility to the protein. Because rigid-body docking is faster than flexible docking, rigid docking may be preferred for initial quick screening a very large compound library. However, poses from such initial rigid docking need to be refined and optimized through flexible docking methods. In addition, because of the advance of computational resources and efficiency, flexible docking is becoming more popular. There are several approaches to take ligand flexibility in molecular docking, including:

· **Systematic methods**: incorporate ligand flexibility by gradually changing structural parameters of the ligands (such as torsional angles, translational and rotational degrees of freedom). Because large conformational space prohibits an exhaustive systematic search, some algorithms use heuristics to focus on regions on conformational space that are likely to contain good poses. While effective in conformational search, it can find a local minimum, rather than the global minimum.

· **Stochastic methods**: makes random changes to the ligand structure to perform the conformational search. It generates an ensemble of conformations that populates a wide range of the energy landscape. While this avoids trapping the final structure at a local energy minimum and increases the probability of finding the global minimum. However, it covers a broader range of the energy landscape, it is computationally more expensive.

· **Genetic algorithms** [11,12]: use the concepts of evolution and natural selection. It begins with encoding structural parameters of the initial structure in a vector (called a chromosome) and generating an ensemble (or population) of chromosomes using the random search algorithm. Each chromosome in this population is evaluated and the most "adapted" ones (with the lowest energy values) are selected as "templates" for the generation of next population. Each iteration of this process will lead to lower-energy binding poses than those from the previous iteration and after a reasonable number of iterations, the chromosome population will converge to a chromosome corresponding to the global energy minimum.

Scoring Functions

Scoring functions [13] are used to evaluate the binding affinity between the ligand and its target. different scoring functions have been developed over the years, and they can be broadly classified into four groups: (1) force-field-based, (2) empirical scoring functions, (3) knowledge-based functions, and (4) consensus-scoring functions.

· **Force-field scoring functions**
Force-field scoring functions use force field parameters used in molecular mechanics calculations. These parameters are derived from experimental data and ab initio quantum mechanical calculations. The force field scorning functions estimate the binding energy by summing the contributions of various bonded terms (e.g., stretching, bending, and torsional forces) and non-bonded terms (e.g., electrostatic and van der Waals interactions). An example of this type of scoring functions is the one used by DOCK [14], whose energy parameters are taken from the AMBER force fields [15,16].

· **Empirical scoring functions**
Empirical scoring functions [17-19] express the binding energy of a protein-ligand complex as a weighted sum of terms that represent physical events involved in the complex formation, such hydrogen bonding, hydrophobic contact, desolvation effects (due to the loss of solvent (water) molecules that stabilizes the ligand), and entropy penalty (due to the loss of ligand flexibility upon the complex formation). The weight and parameters in each term are empirically determined through regression analysis of a set of protein-ligand complexes with known binding affinities. While the performance of empirical scoring functions relies on the accuracy of the experimental data used to develop them, they are faster than force-field-based scoring functions. Empirical scoring functions are used in some popular molecular docking programs like Surflex [20] and FlexX [21,22].

· **Knowledge-based functions**
Knowledge-based functions [23-27] exploit information contained in experimentally determined 3-D structures of protein-ligand complexes. For each ligand-protein atom pair, an interaction potential is generated, which gives the pairwise interaction energy as a function of their separation. These potentials are generated through statistical analysis of the interatomic distance distribution observed in known protein-ligand complexes. The underlying assumption in this approach is that interatomic contacts observed more often in the data set are likely to represent favorable contacts that increase the binding affinity, while those contacts occurring with less frequency are unfavorable and likely to decrease the binding affinity. The final score is computed from these individual interactions.

· **Consensus scoring functions**
Because each scoring function has its strength and weakness, consensus scoring [28-33] has been gaining popularity more recently, which simultaneously uses multiple scoring approaches together to achieve improved accuracies. Many consensus-scoring strategies have been proposed and examples are MultiScore [29], X-CSCORE [30], GFscore [31], supervised consensus scoring (SCS) [32], and SeleX-CS [33].

7.5. Ligand-based virtual screening

Ligand-based approaches use a set of active compounds that are known to interact with the target protein. These approaches are based on the Similar Property Principle, which states structurally similar molecules are likely to have similar (physicochemical and biological) properties. In contrast to structure-based approaches, the ligand-based approaches can be applied when the structure of the target macromolecule is not known. Examples of ligand-based approaches are:

· Pharmacophore methods: identify the pharmacophore pattern common to a set of known actives and uses this pattern in a subsequent substructure search. IUPAC defines a pharmacophore as “the ensemble of steric and electronic features that is necessary to ensure the optimal supramolecular interactions with a specific biological target structure and to trigger (or to block) its biological response.”[34]

· Machine learning methods: use prediction models developed from a training set containing known actives and known inactives. These methods will be discussed in Chapter 8.

· Similarity methods: find molecules structurally similar to known active molecules, based on similarity measures. This topic was discussed in Chapter 6.

7.6. Prediction of ADMET Properties

During the drug discovery and development process, it is very important to consider not only how tightly a potential drug molecule can bind to the target protein (pharmacodynamics), but also how the molecule reach its site of action within the body (pharmacokinetics). This involves the absorption of the drug molecule by the body and the drug transport to the target organ/tissue. While a sufficient number of drug molecules should be available in the body to give the desired therapeutic effect but they must ultimately be removed from the body through metabolism and excretion. In addition, neither the drug nor its metabolites should be toxic. In pharmacology, these properties are often referred to as ADMET properties (which stands for Absorption, Distribution, Metabolism, Excretion, and Toxicity).

During the drug discovery process, various computational tools are routinely used to predict a wide range of ADMET properties of compounds [35-38], including:

· Solubility in water, which affects oral bioavailability of the drug.

· Caco-2 cell monolayer permeability, which is an experimental model for evaluating the intestinal absorption of drugs.

· Permeability through blood-brain barrier between the systemic circulation and the brain.

· Interaction with cytochrome P450 (CYP) proteins involved in drug metabolism.

· Binding affinity to Human ether-a-go-go related gene (hERG) protein, which is responsible for cardiotoxicity of many drugs.

· Interaction with P-glycoprotein efflux pump, involved in the active transport of various compounds out of cells [39].

· Plasma-protein binding, which help estimates the amount of free drugs that can cross membranes.

7.7. Further Reading

· Principles of early drug discovery

https://doi.org/10.1111/j.1476-5381.2010.01127.x

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3058157/

· Recognizing Pitfalls in Virtual Screening: A Critical Review
https://doi.org/10.1021/ci200528d

· ADMET IN SILICO MODELLING: TOWARDS PREDICTION PARADISE?
https://doi.org/10.1038/nrd1032

· Computational Methods in Drug Discovery

https://doi.org/10.1124/pr.112.007336

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3880464/

· Scoring functions and their evaluation methods for protein–ligand docking: recent advances and future directions

https://doi.org/10.1039/C0CP00151A

· Empirical Scoring Functions for Structure-Based Virtual Screening: Applications, Critical Aspects, and Challenges
https://doi.org/10.3389/fphar.2018.01089

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6165880/

References

1. DiMasi JA, Grabowski HG, Hansen RW: **Innovation in the pharmaceutical industry: New estimates of R&D costs.** *J Health Econ* 2016, **47:**20-33.

2. Wagner J, Dahlem AM, Hudson LD, Terry SF, Altman RB, Gilliland CT, DeFeo C, Austin CP: **A dynamic map for learning, communicating, navigating and improving therapeutic development.** *Nat Rev Drug Discov* 2018, **17:**151-153.

3. Lipinski CA, Lombardo F, Dominy BW, Feeney PJ: **Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings.** *Adv Drug Deliv Rev* 1997, **23:**3-25.

4. Bickerton GR, Paolini GV, Besnard J, Muresan S, Hopkins AL: **Quantifying the chemical beauty of drugs.** *Nat Chem* 2012, **4:**90-98.

5. Mignani S, Rodrigues J, Tomas H, Jalal R, Singh PP, Majoral JP, Vishwakarma RA: **Present drug-likeness filters in medicinal chemistry during the hit and lead optimization process: how far can they be simplified?** *Drug Discov Today* 2018, **23:**605-615.

6. Hann MM, Leach AR, Harper G: **Molecular complexity and its impact on the probability of finding leads for drug discovery.** *J Chem Inf Comput Sci* 2001, **41:**856-864.

7. Teague SJ, Davis AM, Leeson PD, Oprea T: **The design of leadlike combinatorial libraries.** *Angew Chem-Int Edit* 1999, **38:**3743-3748.

8. Oprea TI, Davis AM, Teague SJ, Leeson PD: **Is there a difference between leads and drugs? A historical perspective.** *J Chem Inf Comput Sci* 2001, **41:**1308-1315.

9. Oprea TI, Allu TK, Fara DC, Rad RF, Ostopovici L, Bologa CG: **Lead-like, drug-like or "pub-like": how different are they?** *J Comput-Aided Mol Des* 2007, **21:**113-119.

10. Congreve M, Carr R, Murray C, Jhoti H: **A rule of three for fragment-based lead discovery?** *Drug Discov Today* 2003, **8:**876-877.

11. Morris GM, Goodsell DS, Halliday RS, Huey R, Hart WE, Belew RK, Olson AJ: **Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function.** *J Comput Chem* 1998, **19:**1639-1662.

12. Jones G, Willett P, Glen RC, Leach AR, Taylor R: **Development and validation of a genetic algorithm for flexible docking.** *J Mol Biol* 1997, **267:**727-748.

13. Huang SY, Grinter SZ, Zou XQ: **Scoring functions and their evaluation methods for protein-ligand docking: recent advances and future directions.** *Phys Chem Chem Phys* 2010, **12:**12899-12908.

14. Meng EC, Shoichet BK, Kuntz ID: **AUTOMATED DOCKING WITH GRID-BASED ENERGY EVALUATION.** *J Comput Chem* 1992, **13:**505-524.

15. Weiner SJ, Kollman PA, Case DA, Singh UC, Ghio C, Alagona G, Profeta S, Weiner P: **A NEW FORCE-FIELD FOR MOLECULAR MECHANICAL SIMULATION OF NUCLEIC-ACIDS AND PROTEINS.** *J Am Chem Soc* 1984, **106:**765-784.

16. Weiner SJ, Kollman PA, Nguyen DT, Case DA: **AN ALL ATOM FORCE-FIELD FOR SIMULATIONS OF PROTEINS AND NUCLEIC-ACIDS.** *J Comput Chem* 1986, **7:**230-252.

17. Murray CW, Auton TR, Eldridge MD: **Empirical scoring functions. II. The testing of an empirical scoring function for the prediction of ligand-receptor binding affinities and the use of Bayesian regression to improve the quality of the model.** *J Comput-Aided Mol Des* 1998, **12:**503-519.

18. Guedes IA, Pereira FSS, Dardenne LE: **Empirical Scoring Functions for Structure-Based Virtual Screening: Applications, Critical Aspects, and Challenges.** *Front Pharmacol* 2018, **9:**18.

19. Eldridge MD, Murray CW, Auton TR, Paolini GV, Mee RP: **Empirical scoring functions .1. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes.** *J Comput-Aided Mol Des* 1997, **11:**425-445.

20. Jain AN: **Surflex: Fully automatic flexible molecular docking using a molecular similarity-based search engine.** *J Med Chem* 2003, **46:**499-511.

21. Kramer B, Rarey M, Lengauer T: **Evaluation of the FLEXX incremental construction algorithm for protein-ligand docking.** *Proteins* 1999, **37:**228-241.

22. Rarey M, Kramer B, Lengauer T, Klebe G: **A fast flexible docking method using an incremental construction algorithm.** *J Mol Biol* 1996, **261:**470-489.

23. Muegge I, Martin YC: **A general and fast scoring function for protein-ligand interactions: A simplified potential approach.** *J Med Chem* 1999, **42:**791-804.

24. Mitchell JBO, Laskowski RA, Alex A, Thornton JM: **BLEEP - Potential of mean force describing protein-ligand interactions: I. Generating potential.** *J Comput Chem* 1999, **20:**1165-1176.

25. Mitchell JBO, Laskowski RA, Alex A, Forster MJ, Thornton JM: **BLEEP - Potential of mean force describing protein-ligand interactions: II. Calculation of binding energies and comparison with experimental data.** *J Comput Chem* 1999, **20:**1177-1185.

26. Huang SY, Zou XQ: **An iterative knowledge-based scoring function to predict protein-ligand interactions: I. Derivation of interaction potentials.** *J Comput Chem* 2006, **27:**1866-1875.

27. Huang SY, Zou XQ: **An iterative knowledge-based scoring function to predict protein-ligand interactions: II. Validation of the scoring function.** *J Comput Chem* 2006, **27:**1876-1882.

28. O'Boyle NM, Liebeschuetz JW, Cole JC: **Testing Assumptions and Hypotheses for Rescoring Success in Protein-Ligand Docking.** *J Chem Inf Model* 2009, **49:**1871-1878.

29. Terp GE, Johansen BN, Christensen IT, Jorgensen FS: **A new concept for multidimensional selection of ligand conformations (MultiSelect) and multidimensional scoring (MultiScore) of protein-ligand binding affinities.** *J Med Chem* 2001, **44:**2333-2343.

30. Wang RX, Lai LH, Wang SM: **Further development and validation of empirical scoring functions for structure-based binding affinity prediction.** *J Comput-Aided Mol Des* 2002, **16:**11-26.

31. Betzi S, Suhre K, Chetrit B, Guerlesquin F, Morelli X: **GFscore: A general nonlinear consensus scoring function for high-throughput docking.** *J Chem Inf Model* 2006, **46:**1704-1712.

32. Teramoto R, Fukunishi H: **Supervised consensus scoring for docking and virtual screening.** *J Chem Inf Model* 2007, **47:**526-534.

33. Bar-Haim S, Aharon A, Ben-Moshe T, Marantz Y, Senderowitz H: **SeleX-CS: A New Consensus Scoring Algorithm for Hit Discovery and Lead Optimization.** *J Chem Inf Model* 2009, **49:**623-633.

34. Wermuth G, Ganellin CR, Lindberg P, Mitscher LA: **Glossary of terms used in medicinal chemistry (IUPAC Recommendations 1998).** *Pure Appl Chem* 1998, **70:**1129-1143.

35. Norinder U, Bergstrom CAS: **Prediction of ADMET properties.** *ChemMedChem* 2006, **1:**920-937.

36. Moroy G, Martiny VY, Vayer P, Villoutreix BO, Miteva MA: **Toward in silico structure-based ADMET prediction in drug discovery.** *Drug Discov Today* 2012, **17:**44-55.

37. Gleeson MP: **Generation of a set of simple, interpretable ADMET rules of thumb.** *J Med Chem* 2008, **51:**817-834.

38. van de Waterbeemd H, Gifford E: **ADMET in silico modelling: Towards prediction paradise?** *Nat Rev Drug Discov* 2003, **2:**192-204.

39. Li D, Chen L, Li YY, Tian S, Sun HY, Hou TJ: **ADMET Evaluation in Drug Discovery. 13. Development of in Silico Prediction Models for P-Glycoprotein Substrates.** *Mol Pharm* 2014, **11:**716-726.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

7.2: Mathematica Assignment

7.3: Python Assignment-Virtual Screening

Downloadable Files

lecture07-1-virtual-screening_v0.ipynb

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

Objectives

- Perform virtual screening against PubChem using ligand-based approach
- Apply filters to prioritize virtual screening hit list.
- Learn how to use pandas' data frame.

In this notebook, we perform virtual screening against PubChem using a set of known ligands for muscle glycogen phosphorylase. Compound filters will be applied to identify drug-like compounds and unique structures in terms of canonical SMILES will be selected to remove redundant structures. For some top-ranked compounds in the list, their binding mode will be predicted using molecular docking (which will be covered in a separate assignment).

Read known ligands from a file.

As a starting point, let's download a set of known ligands against muscle glycogen phosphorylase. These data are obtained from the DUD-E (Directory of Useful Decoys, Enhanced) data sets (http://dude.docking.org/), which contain known actives and inactives for 102 protein targets. The DUD-E sets are widely used in benchmarking studies that compare the performance of different virtual screening approaches (https://doi.org/10.1021/jm300687e).

Go to the DUD-E target page (http://dude.docking.org/targets) and find muscle glycogen phosphorylase (Target Name: PYGM, PDB ID: 1c8k) from the target list. Clicking the target name "PYGM" directs you to the page that lists various files (http://dude.docking.org/targets/pygm). Download file "**actives_final.ism**", which contains the SMILES strings of known actives. Rename the file name as "**pygm_1c8k_actives.ism**". [Open the file in WordPad or other text viewer/editor to check the format of this file].

Now read the data from the file using the pandas library (https://pandas.pydata.org/). Please go through some tutorials available at https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html

In [1]:

```
import pandas as pd
```

In [2]:

```
colnames = ['smiles','dat', 'id']
df_act = pd.read_csv("pygm_1c8k_actives.ism", sep=" ", names=colnames)
df_act.head(5)
```

Out[2]:

	smiles	dat	id

	smiles	dat	id
0	c1ccc2cc(c(cc2c1)NC(=O)c3cc(ccn3)N(=O)=O)Oc4cc...	220668	CHEMBL134802
1	CC1=C(C(C(=C(N1Cc2ccc(cc2)Cl)C(=O)O)C(=O)O)c3c...	189331	CHEMBL115651
2	CCN1C(=C(C(C(=C1C(=O)O)C(=O)O)c2ccccc2Cl)C(=O)...	188996	CHEMBL113736
3	c1cc(c(c(c1)F)NC(=O)c2cc(ccn2)N(=O)=O)Oc3ccc(c...	219845	CHEMBL133911
4	CC1=C(C(C(=C(N1Cc2cccc(c2)N(=O)=O)C(=O)O)C(=O)...	189034	CHEMBL423509

In [3]:

```
print(len(df_act))    # Show how many structures are in the "data frame"
```

```
77
```

Similarity Search against PubChem

Now, let's perform similarity search against PubChem using each known active compound as a query. There are a few things to mention in this step:

- The isomeric SMILES string is available for each query compound. This string will be used to specify the input structure, so HTTP POST should be used. (Please review lecture02-structure-inputs.ipynb)
- During PubChem's similarity search, molecular similarity is evaluated using the **PubChem fingerprints** and **Tanimoto** coefficient. By default, similarity search will return compounds with Tanimoter scores of **0.9 or higher**. While we will use the default threshold in this practice, it is noteworthy that it is adjustable. If you use a higher threshold (e.g., 0.99), you will get a fewer hits, which are too similar to the query compounds. If you use a lower threshold (e.g., 0.88), you will get more hits, but they will include more false positives.
- PubChem's similarity search does **not** return the similarity scores between the query and hit compounds. Only the hit compound list is returned, which makes it difficult to rank the hit compounds for compound selection. To address this issue, for each hit compound, we compute **the number of query compounds that returned that compound as a hit.** [Because we are using multiple query compounds for similarity search, it is possible for different query compounds to return the same compound as a hit. That is, the hit compound may be similar to multiple query compounds. The underlying assumption is that hit compounds returned multiple times from different queries are more likely to be active than those returned only once from a single query.]
- Add "time.sleep()" to avoid overloading PubChem servers and getting blocked.

In [4]:

```
smiles_act = df_act.smiles.to_list()
```

In [5]:

```
import time
import requests

prolog = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

cids_hit = dict()
```

```
for idx, mysmiles in enumerate(smiles_act) :

    mydata = { 'smiles' : mysmiles }

    url = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt"
    res = requests.post(url, data=mydata)

    if ( res.status_code == 200 ) :
        cids = res.text.split()
        cids = [ int(x) for x in cids ]    # Convert CIDs from string to integer.
    else :
        print("Error at", idx, ":", df_act.loc[idx,'id'], mysmiles )
        print(res.status_code)
        print(res.content)

    for mycid in cids:
        cids_hit[mycid] = cids_hit.get(mycid, 0) + 1

    time.sleep(0.2)
```

In [6]:

```
len(cids_hit)    # Show the number of compounds returned from any query.
```

Out[6]:

```
23981
```

In the above code cells, the returned hits are stored in a dictionary, along with the number of times they are returned. Let's print the top 10 compounds that are returned the most number of times from the search.

In [7]:

```
sorted_by_freq = [ (v, k) for k, v in cids_hit.items() ]
sorted_by_freq.sort(reverse=True)

for v, k in enumerate(sorted_by_freq) :

    if v == 10 :
        break

    print(v, k) # Print (frequency, CID)

```

```
0 (16, 44354348)
1 (15, 44354370)
2 (15, 44354349)
```

```
3 (15, 44354322)
4 (13, 44357907)
5 (12, 44357938)
6 (12, 44357937)
7 (12, 44354455)
8 (12, 44354454)
9 (12, 44354362)
```

Exclude the query compounds from the hits

In the previous step, we repeated similarity searches using multiple query molecules. This may result in a query molecule being returned as a hit from similarity search using another query molecule. Therefore, we want to check if the hit compound list has any query compounds and if any, we want to remove them. Below, we search PubChem for compounds identical to the query molecules and remove them from the hit compound list.

Note that the identity_type parameter in the PUG-REST request is set to "**same_connectivity**", which will return compounds with the same connectivity with the query molecule (ignoring stereochemistry and isotope information). The default for this parameter is "same_stereo_isotope", which returns compounds with the same stereochemistry AND isotope information.

In [8]:

```
cids_query = dict()

for idx, mysmiles in enumerate(smiles_act) :

    mydata = { 'smiles' : mysmiles }
    url = prolog + "/compound/fastidentity/smiles/cids/txt?identity_type=same_connect
    res = requests.post(url, data=mydata)

    if ( res.status_code == 200 ) :
        cids = res.text.split()
        cids = [ int(x) for x in cids]
    else :
        print("Error at", idx, ":", df_act.loc[idx,'id'], mysmiles )
        print(res.status_code)
        print(res.content)

    for mycid in cids:
        cids_query[mycid] = cids_query.get(mycid, 0) + 1

    time.sleep(0.2)
```

In [9]:

```
len(cids_query.keys())    # Show the number of CIDs that represent the query compound
```

Out[9]:

```
133
```

Now remove the query compounds from the hit list (if they are found in the list)

In [11]:

```
for mycid in cids_query.keys() :

    cids_hit.pop(mycid, None)
```

In [12]:

```
len(cids_hit)
```

Out[12]:

```
23848
```

Print the top 10 compounds in the current hit list and compare them with the old ones.

In [13]:

```
sorted_by_freq = [ (v, k) for k, v in cids_hit.items() ]
sorted_by_freq.sort(reverse=True)

for v, k in enumerate(sorted_by_freq) :

    if v == 10 :
        break

    print(v, k)    # Print (frequency, CID)
```

```
0 (12, 11779854)
1 (11, 118078858)
2 (11, 93077065)
3 (11, 93077064)
4 (11, 53013349)
5 (11, 51808718)
6 (11, 45369696)
7 (11, 17600716)
8 (10, 131851009)
9 (10, 129567524)
```

Filtering out non-drug-like compounds

In this step, non-drug-like compounds are filtered out from the list. To do that, four molecular properties are downloaded from PubChem and stored in CSV.

In [15]:

```
chunk_size = 100

if ( len(cids_hit) % chunk_size == 0 ) :
    num_chunks = len(cids_hit) // chunk_size
else :
    num_chunks = len(cids_hit) // chunk_size + 1

cids_list = list(cids_hit.keys())

print("# Number of chunks:", num_chunks )

csv = ""   #sets a variable called csv to save the comma separated output

for i in range(num_chunks) :

    print(i, end=" ")

    idx1 = chunk_size * i
    idx2 = chunk_size * (i + 1)

    cids_str = ",".join([ str(x) for x in cids_list[idx1:idx2] ]) # build pug input f
    url = prolog + "/compound/cid/" + cids_str + "/property/HBondDonorCount,HBondAcce

    res = requests.get(url)

    if ( i == 0 ) : # if this is the first request, store result in empty csv variabl
        csv = res.text
    else :          # if this is a subsequent request, add the request to the csv var
        csv = csv + "\n".join(res.text.split()[1:]) + "\n"

    time.sleep(0.2)

#print(csv)
```

```
# Number of chunks: 239
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

Downloaded data (in CSV) are loaded into a pandas data frame.

In [16]:

```
from io import StringIO

csv_file = StringIO(csv)

df_raw = pd.read_csv(csv_file, sep=",")

df_raw.shape    # Show the shape (dimesnion) of the data frame
```

Out[16]:

```
(23848, 7)
```

In [17]:

```
df_raw.head(5)     # Show the first 5 rows of the data frame
```

Out[17]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES
0	1731763	0	5	454.0	5.5	CCOC(=O)C1=C(N(C(=C(C1C2=CC=C(C=C2)Cl)C(=O)OCC...	CCOC(=O)C1=C(N(C(=C(C1C2=CC=C(C=C2)Cl)C(=O)OCC...
1	21795259	0	5	454.0	5.5	CCOC(=O)C1=C(N(C(=C(C1C2=CC=CC=C2Cl)C(=O)OCC)C...	CCOC(=O)C1=C(N(C(=C(C1C2=CC=CC=C2Cl)C(=O)OCC)C...
2	9910160	3	4	422.9	4.4	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...
3	70157737	2	4	436.9	4.6	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...
4	70074958	2	6	448.9	3.9	CC1=C(C(C(=C(N1)C)C(=O)OC)C2=CC=CC=C2Cl)C(=O)N...	CC1=C([C@@H](C(=C(N1)C)C(=O)OC)C2=CC=CC=C2Cl)C...

Note that some compounds do not have computed XLogP values (because XLogP algorithm cannot handle inorganic compounds, salts, and mixtures) and we want to remove them.

In [18]:

```
df_raw.isna().sum()     # Check if there are any NULL values.
```

Out[18]:

```
CID                     0
HBondDonorCount         0
HBondAcceptorCount      0
```

```
MolecularWeight          0
XLogP                  477
CanonicalSMILES          0
IsomericSMILES           0
dtype: int64
```

In [19]:

```
len(df_raw)    # Check the number of rows (which is equals to the number of CIDs)
```

Out[19]:

```
23848
```

For convenience, add the information contained in the **cids_hit** dictionary to this data frame

In [20]:

```
# First load the cids_hit dictionary into a data frame.
df_freq = pd.DataFrame( cids_hit.items(), columns=['CID','HitFreq'])
df_freq.head(5)
```

Out[20]:

	CID	HitFreq
0	1731763	2
1	21795259	4
2	9910160	3
3	70157737	3
4	70074958	3

In [21]:

```
# Double-check if the data are loaded correctly
# Compare the data with those from Cell [12]
df_freq.sort_values(by=['HitFreq', 'CID'], ascending=False).head(10)
```

Out[21]:

	CID	HitFreq
1022	11779854	12
372	118078858	11
2631	93077065	11
2630	93077064	11
1983	53013349	11
1941	51808718	11

	CID	HitFreq
1707	45369696	11
1467	17600716	11
3072	131851009	10
3067	129567524	10

In [22]:

```
# Create a new data frame called "df" by joining the df and df_freq data frames
df = df_raw.join(df_freq.set_index('CID'), on='CID')
df.shape
```

Out[22]:

```
(23848, 8)
```

In [23]:

```
df.sort_values(by=['HitFreq', 'CID'], ascending=False).head(10)
```

Out[23]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
1022	11779854	3	2	305.3	2.6	C1C(C(=O)NC2=CC=CC=C21)NC(=O)C3=CC4=CC=CC=C4N3	C1C(C(=O)NC2=CC=CC=C21)NC(=O)C3=CC4=CC=CC=C4N3	12
372	118078858	1	2	333.4	3.0	CC1CN(C2=CC=CC=C2N1C(=O)C)C(=O)C3=CC4=CC=CC=C4N3	C[C@H]1CN(C2=CC=CC=C2N1C(=O)C)C(=O)C3=CC4=CC=C...	11
2631	93077065	2	2	409.5	4.8	C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...	C1CC2=CC=CC=C2N(C1)C(=O)[C@@H](C3=CC=CC=C3)NC(...	11

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
2630	93077064	2	2	409.5	4.8	C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...	C1CC2=CC=CC=C2N(C1)C(=O)[C@H](C3=CC=CC=C3)NC(=...	11
1983	53013349	2	2	409.5	4.8	C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...	C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...	11
1941	51808718	2	3	390.5	2.9	CC(C(=O)N1CCN(CC1)CC2=CC=CC=C2)NC(=O)C3=CC4=CC...	C[C@H](C(=O)N1CCN(CC1)CC2=CC=CC=C2)NC(=O)C3=CC...	11
1707	45369696	2	3	390.5	2.9	CC(C(=O)N1CCN(CC1)CC2=CC=CC=C2)NC(=O)C3=CC4=CC...	CC(C(=O)N1CCN(CC1)CC2=CC=CC=C2)NC(=O)C3=CC4=CC...	11
1467	17600716	2	3	390.5	2.9	CC(C(=O)N1CCN(CC1)CC2=CC=CC=C2)NC(=O)C3=CC4=CC...	C[C@@H](C(=O)N1CCN(CC1)CC2=CC=CC=C2)NC(=O)C3=C...	11
3072	131851009	1	2	347.4	3.1	CN1CCN(C(C1=O)CC2=CC=CC=C2)C(=O)C3=CC4=CC=CC=C4N3	CN1CCN([C@H](C1=O)CC2=CC=CC=C2)C(=O)C3=CC4=CC=...	10
3067	129567524	1	2	347.4	2.6	CC(=O)N1CCN(CC1C2=CC=CC=C2)C(=O)C3=CC4=CC=CC=C4N3	CC(=O)N1CCN(C[C@@H]1C2=CC=CC=C2)C(=O)C3=CC4=CC...	10

Now identify and remove those compounds that satisfy all criteria of Lipinski's rule of five.

In [24]:

```
len(df[ df['HBondDonorCount'] <= 5 ])
```

Out[24]:

```
23803
```

In [25]:

```
len(df[ df['HBondAcceptorCount'] <= 10 ])
```

Out[25]:

```
23830
```

In [26]:

```
len(df[ df['MolecularWeight'] <= 500 ])
```

Out[26]:

```
23238
```

In [27]:

```
len(df[ df['XLogP'] < 5 ])
```

Out[27]:

```
21105
```

In [28]:

```
df = df[ ( df['HBondDonorCount'] <= 5) &
         ( df['HBondAcceptorCount'] <= 10 ) &
         ( df['MolecularWeight'] <= 500 ) &
         ( df['XLogP'] < 5 ) ]
```

In [29]:

```
len(df)
```

Out[29]:

```
20827
```

5. Draw the structures of the top 10 compounds

Let's check the structure of the top 10 compounds in the hit list.

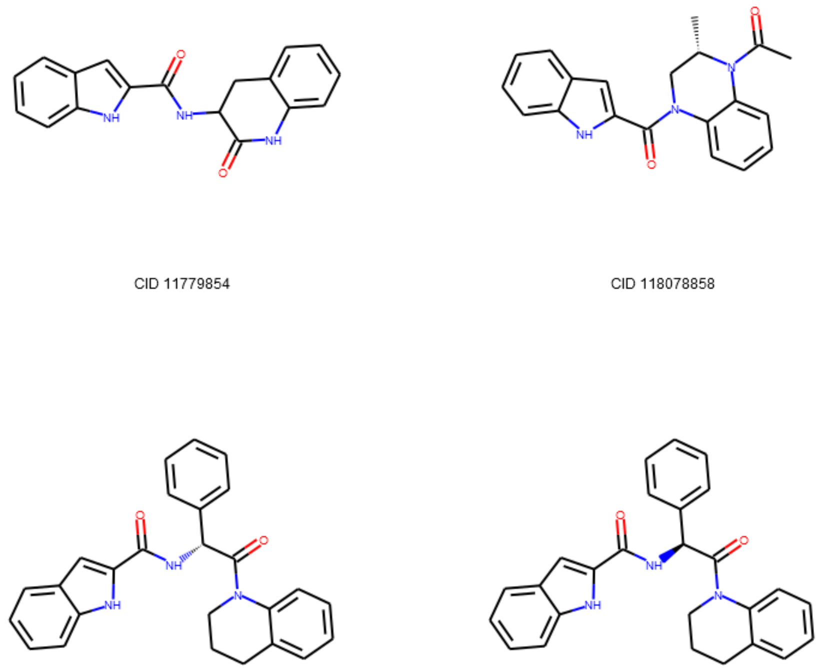
In [30]:

```
cids_top = df.sort_values(by=['HitFreq', 'CID'], ascending=False).head(10).CID.to_lis
```

In [31]:

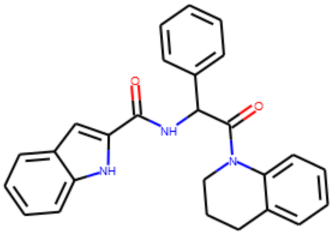
```
from rdkit import Chem
from rdkit.Chem import Draw

mols = []

for mycid in cids_top :

    mysmiles = df[ df.CID==mycid ].IsomericSMILES.item()

    mol = Chem.MolFromSmiles( mysmiles )
    Chem.FindPotentialStereoBonds(mol)    # Identify potential stereo bonds!
    mols.append(mol)

mylegends = [ "CID " + str(x) for x in cids_top ]
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(400,400), legends=mylegend
display(img)
```

```
RDKit WARNING: [10:47:56] Enabling RDKit 2019.09.1 jupyter extensions
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
```

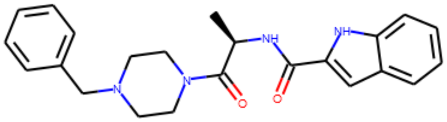


CID 93077065

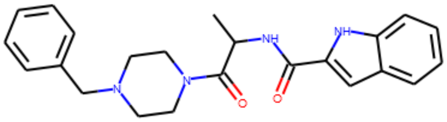
CID 93077064



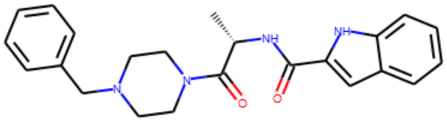
CID 53013349



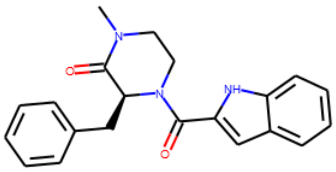
CID 51808718



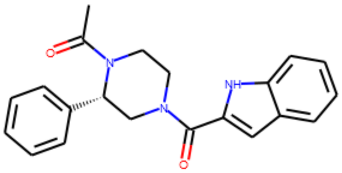
CID 45369696



CID 17600716



CID 131851009



CID 129567524

An important observation from these images is that the hit list contains multiple compounds with the same connectivity. For example, CIDs 93077065 and 93077064 are stereoisomers of each other and CID 53013349 has the same connectivity as the two CIDs, but with its stereocenter being unspecified. When performing a screening with limited resources in the early stage of drug discovery, you may want to test as diverse molecules as possible, avoiding testing too similar structures.

To do so, let's look into PubChem's canonical SMILES strings, which do not encode the stereochemisry and isotope information. Chemicals with the same connectivity but with different stereochemistry/isotopes should have the same canonical SMILES. In the next section, we select unique compounds in terms of canonical SMILES to reduce the number of compounds to screen.

Extract unique compounds in terms of canonical SMILES

The next few cells show how to get unique values within a column (in this case, unique canonical SMILES).

In [32]:

```
len(df)
```

Out[32]:

```
20827
```

In [33]:

```
len(df.CanonicalSMILES.unique())
```

Out[33]:

```
16543
```

In [34]:

```
canonical_smiles = df.CanonicalSMILES.unique()
```

In [35]:

```
df[ df.CanonicalSMILES == canonical_smiles[0] ]
```

Out[35]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
2	9910160	3	4	422.9	4.4	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	3

In [36]:

```
df[ df.CanonicalSMILES == canonical_smiles[1] ]
```

Out[36]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
3	70157737	2	4	436.9	4.6	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	3
6	18318317	2	4	436.9	4.6	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...	3

In [37]:

```
df[ df.CanonicalSMILES == canonical_smiles[1] ].IsomericSMILES.to_list()
```

Out[37]:

```
['CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)N(C)CC=CC3=CC=CC=C3',
 'CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)N(C)C/C=C/C3=CC=CC=C3']
```

Now let's generate a list of unique compounds in terms of canonical SMILES. If multiple compounds have the same canonical SMILES, the one that appears very first will be included in the unique compound list.

In [38]:

```
idx_to_include = []

for mysmiles in canonical_smiles :

    myidx = df[ df.CanonicalSMILES == mysmiles ].index.to_list()[0]

    idx_to_include.append( myidx )
```

In [39]:

```
len(idx_to_include)
```

Out[39]:

```
16543
```

In [40]:

```
# Create a new column 'Include'
# All values initialized to 0 (not include)
```

```
df['Include'] = 0
df['Include'].sum()
```

Out[40]:

```
0
```

In [41]:

```
# Now the "Include" column's value is modified if the record is in the idx_to_include
df.loc[idx_to_include,'Include'] = 1
df['Include'].sum()
```

Out[41]:

```
16543
```

In [42]:

```
df[['CID','Include']].head(10)
```

Out[42]:

	CID	Include
2	9910160	1
3	70157737	1
4	70074958	1
5	70073800	0
6	18318317	0
7	18318313	1
9	18318274	1
10	18318270	1
11	15838576	0
12	15838575	1

Now draw the top 10 unique compounds (in terms of canonical SMILES). Note the, the structure figures are drawn using isomeric SMILES, but canonical SMILES strings could be used.

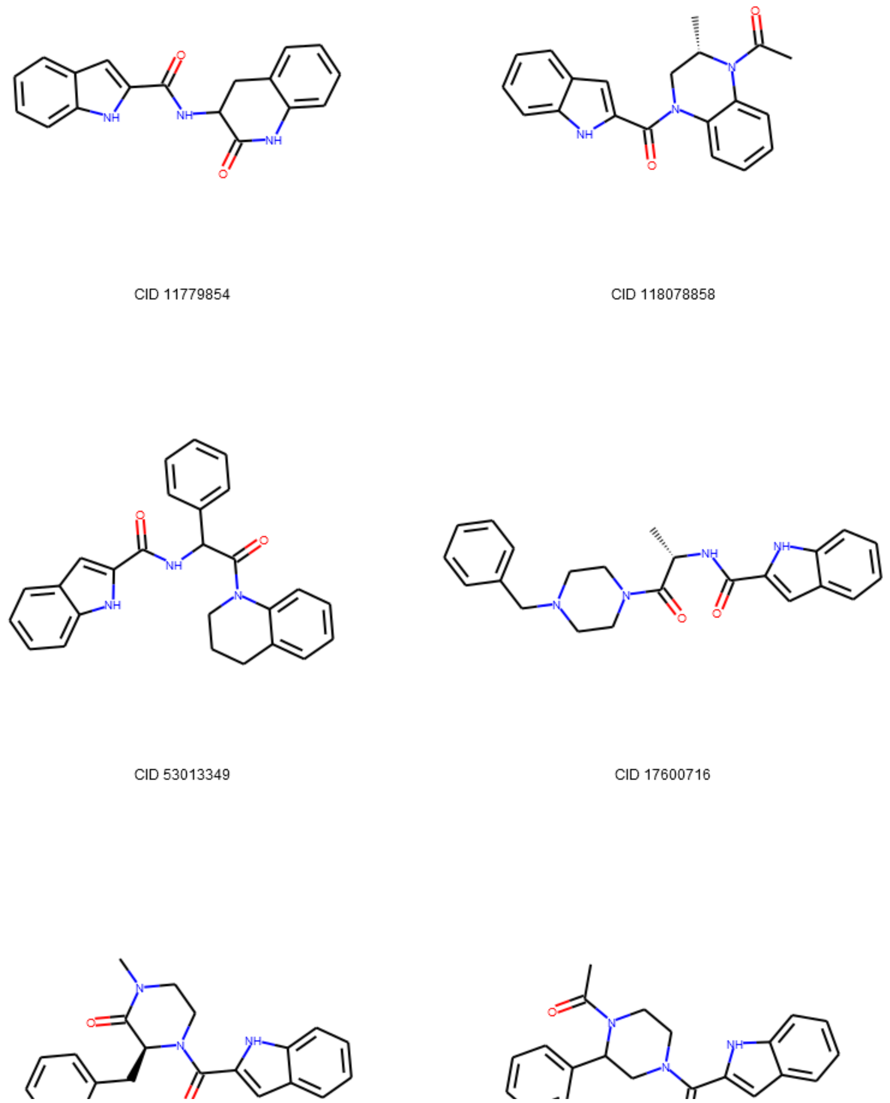
In [43]:

```
cids_top = df[ df['Include'] == 1].sort_values(by=['HitFreq', 'CID'], ascending=False
```

In [44]:

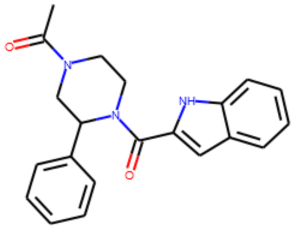
```
mols = []

for mycid in cids_top :
    mysmiles = df[ df.CID==mycid ].IsomericSMILES.item()

    mol = Chem.MolFromSmiles( mysmiles )
    Chem.FindPotentialStereoBonds(mol)    # Identify potential stereo bonds!
    mols.append(mol)

mylegends = [ "CID " + str(x) for x in cids_top ]
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(400,400), legends=mylegend
display(img)
```

```
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
  """
```

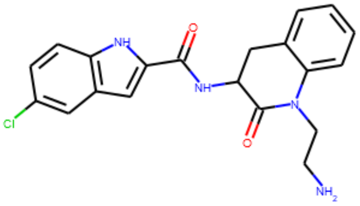


CID 131851009

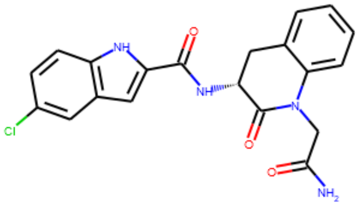
CID 110098384



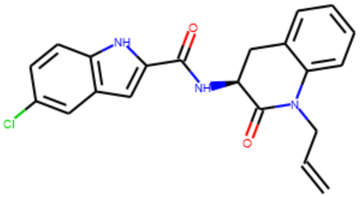
CID 110098233



CID 69517802



CID 69196722



CID 69195207

Saving molecules in files

Now save the molecules in the cids_top list in files, which will be used in molecular docking experiments. For simplicity, we will use only the **top 3** compounds in the list.

In [45]:

```
from rdkit.Chem import AllChem

for idx, mycid in enumerate( cids_top ) :

    if idx == 3 :
        break

    mysmiles = df[ df['CID'] == mycid ].IsomericSMILES.item()

    mymol = Chem.MolFromSmiles(mysmiles)
    mymol = Chem.AddHs(mymol)
    AllChem.EmbedMolecule(mymol)
    AllChem.MMFFOptimizeMolecule(mymol)

    filename = "pygm_lig" + str(idx) + "_" + str(mycid) + ".mol"
    Chem.MolToMolFile(mymol, filename)
```

```
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
```

To save all data in the **df** data frame (in CSV)

In [46]:

```
df.to_csv('pygm_df.csv')
```

Exersises

1. From the DUD-E target list (http://dude.docking.org/targets), find cyclooxigenase-2 (Target Name: PGH2, PDB ID: 3ln1). Download the "**actives_final.ism**" file and save it as "**pgh2_3ln1_actives.ism**". Load the data into a data frame called **df_act**. After loading the data, show the following information:

- the number of rows of the data frame.
- the first five rows of the data frame.

In [47]:

```
# Write your code in this cell
```

In [48]:

```
df_act.head(5)
```

Out[48]:

	smiles	dat	id
0	c1ccc2cc(c(cc2c1)NC(=O)c3cc(ccn3)N(=O)=O)Oc4cc...	220668	CHEMBL134802
1	CC1=C(C(C(=C(N1Cc2ccc(cc2)Cl)C(=O)O)C(=O)O)c3c...	189331	CHEMBL115651
2	CCN1C(=C(C(C(=C1C(=O)O)C(=O)O)c2ccccc2Cl)C(=O)...	188996	CHEMBL113736
3	c1cc(c(c(c1)F)NC(=O)c2cc(ccn2)N(=O)=O)Oc3ccc(c...	219845	CHEMBL133911
4	CC1=C(C(C(=C(N1Cc2cccc(c2)N(=O)=O)C(=O)O)C(=O)...	189034	CHEMBL423509

2. Perform similarity search using each of the isomeric SMILES contained in the loaded data frame.

- As we did for PYGM ligands in this notebook, track the number of times a particular hit is returned from multiple queries, using a dictionary named **cids_hit** (CIDs as keys and the frequencies as values). This information will be used to rank the hit compounds.

- Make sure that the CIDs are recognized as integers when they are used as keys in the dictionary.
- Print the total number of hits returned from this step (which is the same as the number of CIDs in **cids_hit**.
- Add **time.sleep()** to avoid overloading PubChem servers.

In [49]:

```
prolog = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

cids_hit = dict()

for idx, mysmiles in enumerate(df_act.smiles.to_list()) :

    mydata = { 'smiles' : mysmiles }

    url = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt"
    res = requests.post(url, data=mydata)

    if ( res.status_code == 200 ) :
        cids = res.text.split()
        cids = [ int(x) for x in cids ]
    else :
        print("Error at", idx, ":", df_act.loc[idx,'id'], mysmiles )
        print(res.status_code)
        print(res.content)

    for mycid in cids:
        cids_hit[mycid] = cids_hit.get(mycid, 0) + 1

    time.sleep(0.2)
```

In [50]:

```
len(cids_hit)
```

Out[50]:

```
23981
```

3. The hit list from the above step may contain the query compounds themselves. Get the CIDs of the query compounds through idenitity search and remove them from the hit list.

- Set the optional parameter "**identity_type**" to "**same_connectivity**".
- Add **time.sleep()** to avoid overloading PubChem servers.
- Print the number of CIDs corresponding to the query compounds.
- Print the number of the remaining hit compounds, after removing the query compounds from the hit list.

In [51]:

```
# Write your code in this cell
```

In [52]:

```
for mycid in cids_query.keys() :

    cids_hit.pop(mycid, None)
```

In [53]:

```
len(cids_query)
```

Out[53]:

```
133
```

In [54]:

```
len(cids_hit)
```

Out[54]:

```
23848
```

4. Download the hydrogen donor and acceptor counts, molecular weights, XlogP, and canonical and isomeric SMILES for each compound in **cids_hit**. Load the downloaded data into a new data frame called **df_raw**. Print the size (or dimension) of the data frame using **.shape**.

In [55]:

```
# Write your code in this cell
```

In [56]:

```
from io import StringIO

csv_file = StringIO(csv)
df_raw = pd.read_csv(csv_file, sep=",")
df_raw.shape
```

Out[56]:

```
(23848, 7)
```

5. Create a new data frame called **df**, which combines the data stored in **cids_hit** and **df_raw**.

- First load the frequency data into a new data frame called **df_freq** and then join **df_raw** and **df_freq** into **df**
- Print the shape (dimension) of **df**

In [57]:

```
# Write your code in this cell
```

6. Remove from **df** the compounds that violate any criterion of Congreve's rule of 3 and show the number of remaining compounds (the number of rows of **df**).

In [58]:

```
# Write your code in this cell
```

7. Get the unique canonical SMILES strings from the **df**. Add to the **df** a column named '**Include**', which contains a flag set to 1 for the lowest CID associated with each unique CID and set to 0 for other CIDs. Show the number of compounds for which this flag is set to 1.

In [59]:

```
# Write your code in this cell
```

In [60]:

```
idx_to_include = []

for mysmiles in canonical_smiles :

    myidx = df[ df.CanonicalSMILES == mysmiles ].index.to_list()[0]

    idx_to_include.append( myidx )
```

In [61]:

```
df['Include'] = 0
df.loc[idx_to_include,'Include'] = 1

df['Include'].sum()
```

Out[61]:

```
16543
```

8. Among those with the "Include" flag set to 1, identify the top 10 compounds that were returned from the largest number of query compounds.

- Sort the data frame by the number of times returned (in descending order) and then by CID (in ascending order)
- For each of the 10 compounds, print its CID, isomeric SMILES, and the number of times it was returned.
- For each of the 10 compounds, draw its structure (using isomeric SMILES).

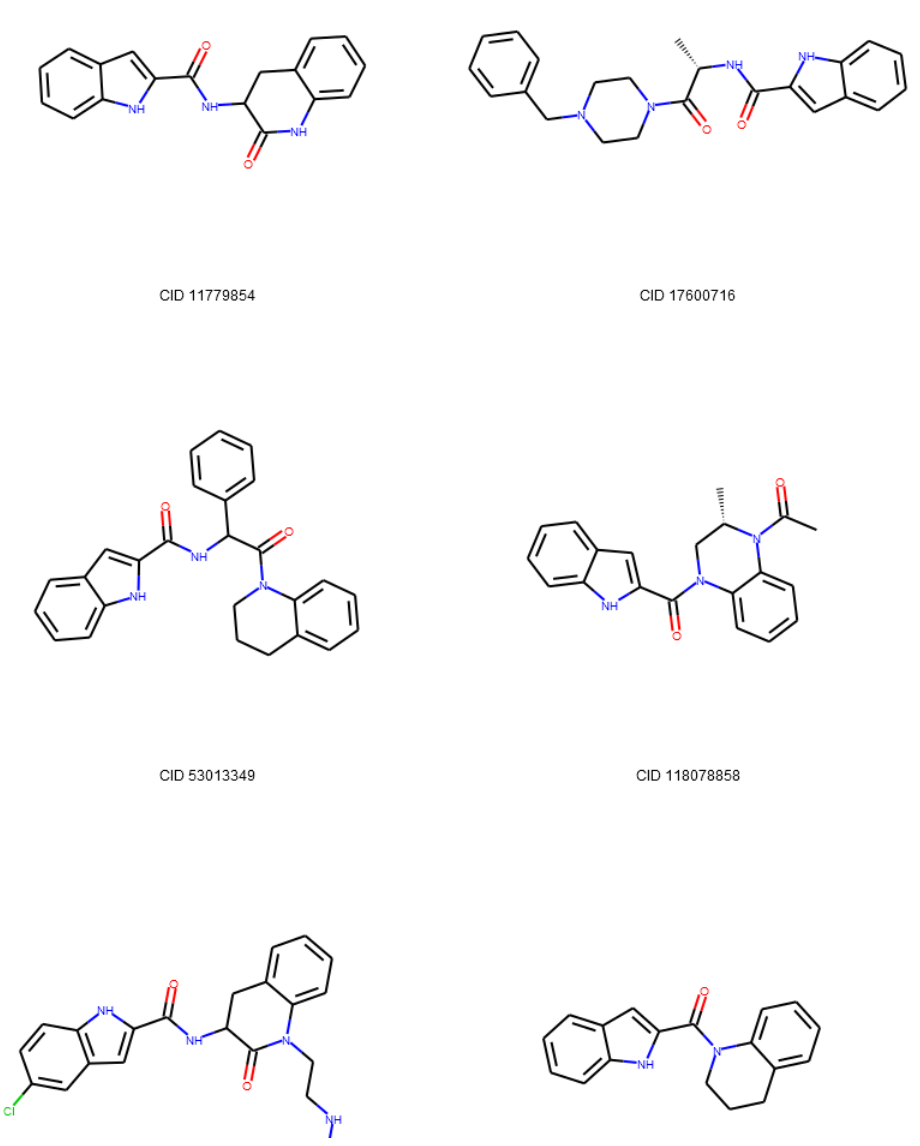
In [62]:

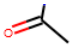
```
# Write your code in this cell
```

In [63]:

```
cids_top = df[ df['Include'] == 1].sort_values(by=['HitFreq', 'CID'], ascending=[Fals

mols = []

for mycid in cids_top :
    mysmiles = df[ df.CID==mycid ].IsomericSMILES.item()

    mol = Chem.MolFromSmiles( mysmiles )
    Chem.FindPotentialStereoBonds(mol)    # Identify potential stereo bonds!
    mols.append(mol)

mylegends = [ "CID " + str(x) for x in cids_top ]
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(400,400), legends=mylegend
display(img)
```

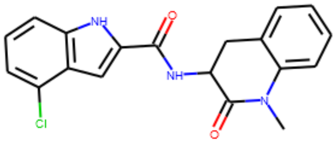
```
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
  import sys
```



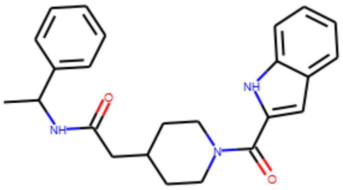


CID 10127135

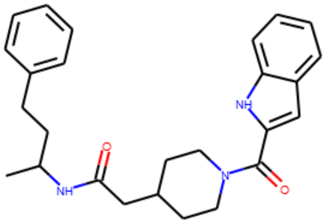
CID 43397595



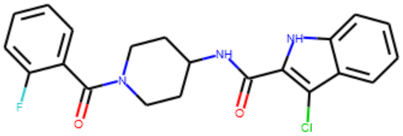
CID 44358150



CID 46376317



CID 50816864



CID 55961297

7.4: R Assignment

Virtual Screening

Objectives

- Perform virtual screening against PubChem using ligand-based approach
- Apply filters to prioritize virtual screening hit list.
- Learn how to use pandas' data frame.

In this notebook, we perform virtual screening against PubChem using a set of known ligands for muscle glycogen phosphorylase. Compound filters will be applied to identify drug-like compounds and unique structures in terms of canonical SMILES will be selected to remove redundant structures. For some top-ranked compounds in the list, their binding mode will be predicted using molecular docking (which will be covered in a separate assignment).

1. Read known ligands from a file.

As a starting point, let's download a set of known ligands against muscle glycogen phosphorylase. These data are obtained from the DUD-E (Directory of Useful Decoys, Enhanced) data sets (http://dude.docking.org/), which contain known actives and inactives for 102 protein targets. The DUD-E sets are widely used in benchmarking studies that compare the performance of different virtual screening approaches (https://doi.org/10.1021/jm300687e).

Go to the DUD-E target page (http://dude.docking.org/targets) and find muscle glycogen phosphorylase (Target Name: PYGM, PDB ID: 1c8k) from the target list. Clicking the target name "PYGM" directs you to the page that lists various files (http://dude.docking.org/targets/pygm). Download file "**actives_final.ism**", which contains the SMILES strings of known actives. Rename the file name as "**pygm_1c8k_actives.ism**". [Open the file in WordPad or other text viewer/editor to check the format of this file].

Now read the data from the file using the pandas library (https://pandas.pydata.org/). Please go through some tutorials available at https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html

```
colnames <- c('smiles','dat', 'id')
df_act <- read.delim("pygm_1c8k_actives.ism", header = FALSE, sep = " ")
colnames(df_act) <- colnames

head(df_act)
```

```
 print(nrow(df_act))    # Show how many structures are in the "data frame"
```

2. Similarity Search against PubChem

Now, let's perform similarity search against PubChem using each known active compound as a query. There are a few things to mention in this step:

- The isomeric SMILES string is available for each query compound. This string will be used to specify the input structure, so HTTP POST should be used. (Please review lecture02-structure-inputs.ipynb)
- During PubChem's similarity search, molecular similarity is evaluated using the **PubChem fingerprints** and **Tanimoto** coefficient. By default, similarity search will return compounds with Tanimoter scores of **0.9 or higher**. While we will use the default threshold in this practice, it is noteworthy that it is adjustable. If you use a higher threshold (e.g., 0.99), you will get a fewer hits, which are too similar to the query compounds. If you use a lower threshold (e.g., 0.88), you will get more hits, but they will include more false positives.
- PubChem's similarity search does **not** return the similarity scores between the query and hit compounds. Only the hit compound list is returned, which makes it difficult to rank the hit compounds for compound selection. To address this issue, for each hit compound, we compute **the number of query compounds that returned that compound as a hit.** [Because we are using multiple query compounds for similarity search, it is possible for different query compounds to return the same compound as a

hit. That is, the hit compound may be similar to multiple query compounds. The underlying assumption is that hit compounds returned multiple times from different queries are more likely to be active than those returned only once from a single query.]

- Add "sys.sleep()" to avoid overloading PubChem servers and getting blocked.

```
smiles_act <- list(df_act$smiles)
```

WIP WIP WIP

7.5: Molecular Docking Experiments

Molecular Docking Experiments

This tutorial explains how to perform molecular docking experiments using Autodock Vina (molecular docking software) and UCSF Chimera (molecular visualization software), both of which are freely available for academic users. In this tutorial, we will use the 3-D structure of muscle glycogen phosphorylase from rabbit (PDB ID: 1c8k) as a target macromolecule template and three ligand molecules (stored in the files generated using the jupyter notebook for Chapter 7 (lecture07-virtual-screening_v0.ipynb)).

Step 0: Installation of Autodock Vina and Chimera

(0-A) Download Autodock Vina (http://vina.scripps.edu/download.html) and install it on your computer.

(0-B) Download UCSF Chimera (http://www.cgl.ucsf.edu/chimera/download.html) and install it on your computer.

· For both programs, make sure that you download a correct version, depending on the operation system of your computer (Windows, Mac, or Linux) and whether it is 64-bit or 32-bit.)

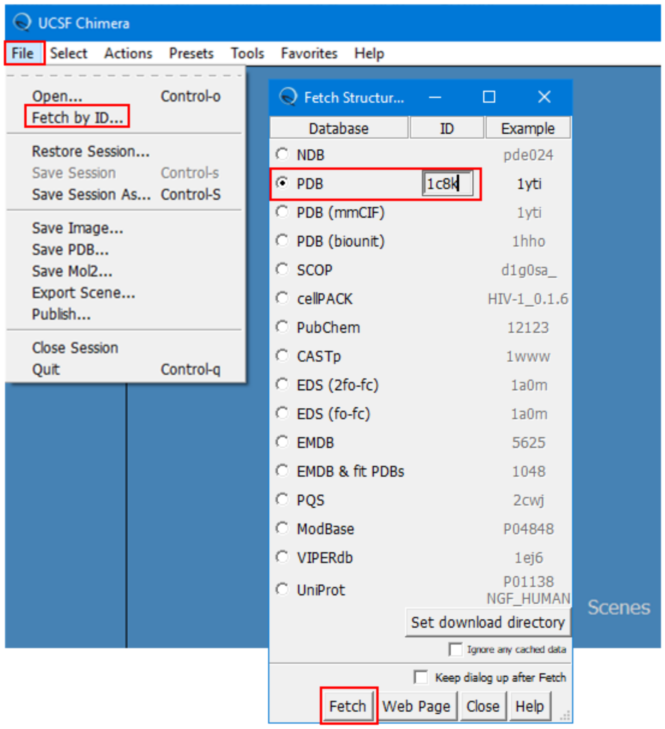
(0-C) Read the tutorial “UCSF Chimera – Getting Started“ (https://www.cgl.ucsf.edu/Outreach/Tutorials/GettingStarted.html) to learn how to use chimera. Other tutorials are also available at https://www.cgl.ucsf.edu/chimera/tutorials.html.

(0-D) Watch this YouTube video (https://www.youtube.com/watch?v=cLGQ-951FDk). This video explains how to perform a docking experiment using Autodock Vina and Chimera.

Step 1: Loading the 3-D structure of the co-crystallized target-ligand complex.

(1A) Open Chimera.

(1B) Select “FILE” --> “Fetch by ID” --> “PDB”, provide the PDB ID (“1c8k”), and click “Fetch” at the bottom. This will load the 3-D structure of “1c8k”.



Database	ID	Example
○ NDB		pde024
● PDB	1c8k	1yti
○ PDB (mmCIF)		1yti
○ PDB (biounit)		1hho
○ SCOP		d1g0sa_
○ cellPACK		HIV-1_0.1.6
○ PubChem		12123
○ CASTp		1www
○ EDS (2fo-fc)		1a0m
○ EDS (fo-fc)		1a0m
○ EMDB		5625
○ EMDB & fit PDBs		1048
○ PQS		2cwj
○ ModBase		P04848
○ VIPERdb		1ej6
○ UniProt		P01138 NGF_HUMAN

(1C) Save this structure as “pygm_1c8k.pdb” (for your reference) by selecting “FILE” --> “Save PDB …” and providing the filename. [Here, “pygm” is the symbol of the gene that encodes the target protein (glycogen phosphorylase, muscle associated).]

Step 2: Prepare a ligand structure for docking, using the inhibitor co-crystallized with the target in 1c8k. This docking example will allow us to compare a predicted pose of the inhibitor with the experimental pose. Note that the 1c8k protein-ligand complex has two small molecules bound to the target protein. We need to figure out which one is the inhibitor.

(2A) Review the ligand information for PDB ID 1c8k, available at: https://www.rcsb.org/structure/1C8K. This protein-ligand complex has two small molecules bound to the protein: Flavopyridol (CPB) and Vitamin B6 Phosphate (PLP). According to the abstract available on this page, CPB is an inhibitor and PLP is a substrate of the protein. In this docking experiment, we will dock ligands to the inhibition site of the protein.

(2B) Investigate the structure of the inhibition site. Five amino acid residues with close contacts to the CPB ligand are already shown by default. Place the mouse pointer on each of these residues and record their names and IDs. (For example, PHE285, GLU382, …)

(2C) Delete all residues and ligand/water molecules, **except for CPB**. This can be done by selecting:

(2C-1) Select --> Residue --> CPB

(2C-2) Select --> Invert (all models)

(2C-3) Actions --> Atoms/Bonds --> Delete

(2D) Save the ligand molecule by following these steps:

(2D-1) select File --> Save PDB…

(2D-2) provide the file name "**pygm_1c8k_cpb**"

(2D-3) uncheck "Use untransformed coordinates"

(2D-4) press "Save"

(2E) Close the current session by selecting: File --> Close session.

Step 3: Prepare a protein structure for docking.

(3A) Load the 1c8k structure [in the same way as Step (1A)-(1B)].

(3B) Delete all ligand/water molecules, **except for protein**. This can be done by selecting:

(3B-1) Select --> Structure --> Protein

(3B-2) Select --> Invert (all models)

(3B-3) Actions --> Atoms/Bonds --> Delete

(3C) Save the protein structure by following these steps:

(3C-1) select File --> Save PDB…

(3C-2) provide the file name "**pygm_1c8k_protein.pdb**"

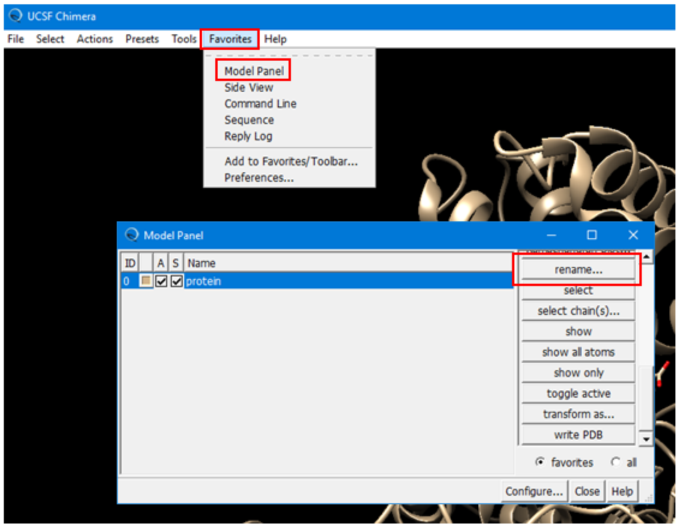
(3C-3) uncheck "Use untransformed coordinates"

(3C-4) press "Save"

NOTE: DO NOT CLOSE THE CURRENT SESSION NOW. WE WILL USE THIS PROTEIN STRUCTURE TO SET UP THE DOCKING EXPERIMENT.

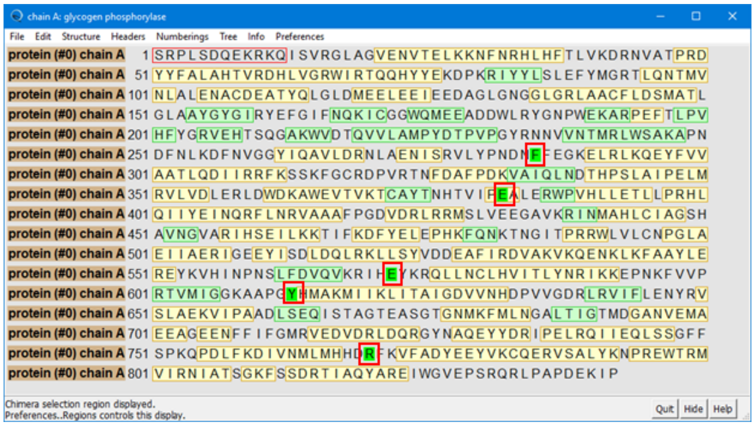
Step 4: Set up a molecular docking experiment.

(4A) Open the Model Panel by selecting: Favorites --> Model Panel (Alternatively, Tools --> General Controls --> Model Panel.) Rename the existing protein structure to "protein".



(4B) Create a named selection for the inhibition site, using the residue names/IDs from **Step 2B**.

(4B-1) Open the Sequence window by choosing: Tools --> Sequence --> Sequence and then choosing "protein", which will be used as a template for docking experiments.

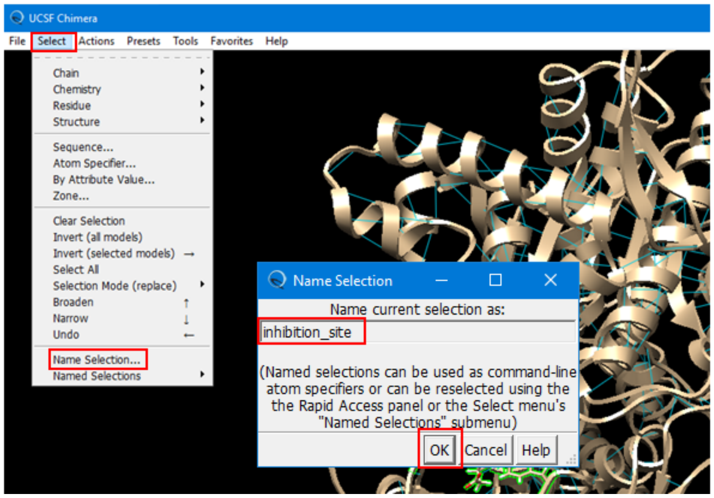

chain A: glycogen phosphorylase
File Edit Structure Headers Numberings Tree Info Preferences
Quit Hide Help

```
protein (#0) chain A   1 SRPLSDQEKRKQISVRGLAGVENVTELKKNFNRHLHFTLVKDRNVATPRD
protein (#0) chain A  51 YYFALAHTVRDHLVGRWIRTQQHYYEKDPKRIYYLSLEFYMGRTLQNTMV
protein (#0) chain A 101 NLALENACDEATYQLGLDMEELEEIEEDAGLGNGGLGRLAACFLDSMATL
protein (#0) chain A 151 GLAAYGYGIRYEFGIFNQKICGGWQMEEADDWLRYGNPWEKARPEFTLPV
protein (#0) chain A 201 HFYGRVEHTSQGAKWVDTQVVLAMPYDTPVPGYRNNVVNTMRLWSAKAPN
protein (#0) chain A 251 DFNLKDFNVGGYIQAVLDRNLAENISRVLYPNDNFFEGKELRLKQEYFVV
protein (#0) chain A 301 AATLQDIIRRFKSSKFGCRDPVRTNFDAFPDKVAIQLNDTHPSLAIPELM
protein (#0) chain A 351 RVLVDLERLDWDKAWEVTVKTCAYTNHTVIPEALERWPVHLLETLLPRHL
protein (#0) chain A 401 QIIYEINQRFLNRVAAAFPGDVDRLRRMSLVEEGAVKRINMAHLCIAGSH
protein (#0) chain A 451 AVNGVARIHSEILKKTIFKDFYELEPHKFQNKTNGITPRRWLVLCNPGLA
protein (#0) chain A 501 EIIAERIGEEYISDLDQLRKLLSYVDDEAFIRDVAKVKQENKLKFAAYLE
protein (#0) chain A 551 REYKVHINPNSLFDVQVKRIHEYKRQLLNCLHVITLYNRIKKEPNKFVVP
protein (#0) chain A 601 RTVMIGGKAAPGYHMAKMIIKLITAIGDVVNHDPVVGDRLRVIFLENYRV
protein (#0) chain A 651 SLAEKVIPAADLSEQISTAGTEASGTGNMKFMLNGALTIGTMDGANVEMA
protein (#0) chain A 701 EEAGEENFFIFGMRVEDVDRLDQRGYNAQEYYDRIPELRQIIEQLSSGFF
protein (#0) chain A 751 SPKQPDLFKDIVNMLMHHDRFKVFADYEEYVKCQERVSALYKNPREWTRM
protein (#0) chain A 801 VIRNIATSGKFSSDRTIAQYAREIWGVEPSRQRLPAPDEKIP
```

Chimera selection region displayed.
Preferences..Regions controls this display.

(4B-2) Select the five residues identified in Step 2B. When the mouse point is placed on a residue, their symbol and ID will be shown at the right-bottom of the windown (e.g., ILE 584). Use "control+drag" to add new region and "shift+drag" to add to region. Selected residues will be highlighted in green.

(4B-3) Go to the main Chimera window and choose: Select --> Name selection and provide "inhibition_site" as an alias for the five selected residues. After naming this selection, "inhibition_site" will be listed under the "Named Selections". This will be used later to display the inhibition site.



Name current selection as:

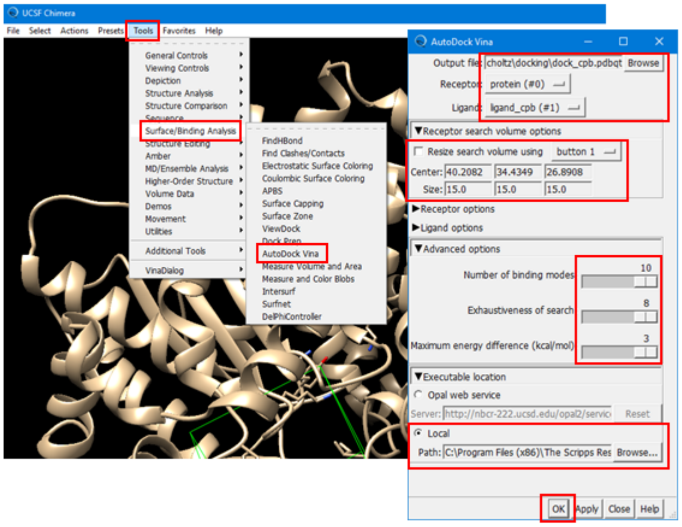
inhibition_site

(Named selections can be used as command-line atom specifiers or can be reselected using the the Rapid Access panel or the Select menu's "Named Selections" submenu)

(4C) Without closing the current session (which has the protein structure), open the ligand file prepared in Step 2 (“pygm_1c8k_cpb.pdb”).

(4D) Check the location of the ligand by using the “Model Panel”, which can be accessed by selecting: Favorites --> Model Panel (alternatively, by selecting: Tools --> General Controls --> Model Panel. Note that the ligand is not necessarily positioned at the binding pocket (because the “Use untransformed coordinates” box was unchecked when the ligand was saved). Rename the ligand as “ligand_cpb”.

(4E) Open the Autodock Vina window by selecting: Tools --> Surface/Binding Analysis --> Autodock Vina.



Output file: choltz\docking\dock_cpb.pdbqt

Receptor: protein (#0)

Ligand: ligand_cpb (#1)

□ Resize search volume using button 1

Center: 40.2082 34.4349 26.8908

Size: 15.0 15.0 15.0

Number of binding modes 10

Exhaustiveness of search 8

Maximum energy difference (kcal/mol) 3

○ Opal web service

Server: http://nbcr-222.ucsd.edu/opal2/servic

● Local

Path: C:\Program Files (x86)\The Scripps Res

(4E-1) Set the output file name to “dock_cpb.pdbqt”. [Note that the file extension is “pdbqt”].

(4E-2) Select “protein” as the receptor and “ligand_cpb” as the ligand.

(4E-3) Define the simulator box using the “Receptor Search Box Options”. This box defines the location of the binding site for the ligand and it tells the docking program where it should search to predict the best binding pose of the ligand. Once you check the box for “Resize search volume using Button X” (where X can be 1, 2, or 3 or Ctrl+1, Ctrl+2, Ctrl3, depending on your machine), you can draw, move, resize the simulator box. [To resize the box, you need to click and drag one of the faces of the box while holding down the left mouse button]. Make sure that all residues to which the CPB ligand was bound should be contained in the simulator box. If you have difficulty in setting up the box, use the following numbers:

Center: 40.2082, 34.4349, 26.8908

Size: 15.00, 15.00, 15.00

[These numbers were used to get the docking results shown in the images included in this tutorial.]

(4E-4) Select “Advanced Option” and adjust the parameters to the largest possible values.

· 10 for Number of binding mode

· 8 for Exhaustiveness of search

· 3 for maximum energy difference (kcal/mol)

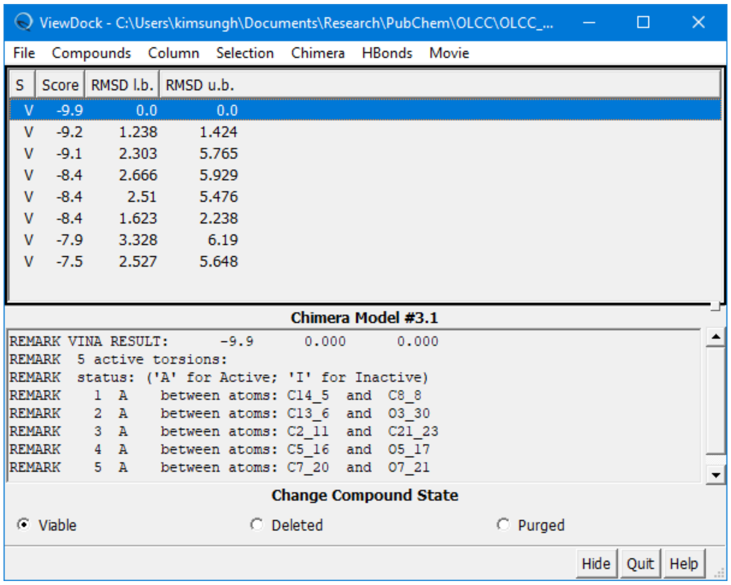
In this exercise, we are testing only a few ligands, which does not take too much computational resources, so it is okay to adjust these parameters to the maximum possible values. However, if you are performing a large-scale screening, they need to be adjusted accordingly.

(4E-5) Select “Executable Location” and then “Local” and provide the path of the “vina.exe” installed on you PC. It should be something similar to:

C:\Program Files (x86)\The Scripps Research Institute\Vina\vina.exe

(4E-6) Press “OK” to start the docking experiment. This will take a few minutes, depending on your machine. You may encounter some warning messages, but the computation will continue until it finds the requested number of poses.

(4E-7) Once the docking experiment is done, the **View Dock** window will be presented, which shows the list of poses predicted, along with additional information.



S	Score	RMSD l.b.	RMSD u.b.
V	-9.9	0.0	0.0
V	-9.2	1.238	1.424
V	-9.1	2.303	5.765
V	-8.4	2.666	5.929
V	-8.4	2.51	5.476
V	-8.4	1.623	2.238
V	-7.9	3.328	6.19
V	-7.5	2.527	5.648

```
REMARK VINA RESULT:      -9.9      0.000      0.000
REMARK  5 active torsions:
REMARK  status: ('A' for Active; 'I' for Inactive)
REMARK    1  A    between atoms: C14_5  and  C8_8
REMARK    2  A    between atoms: C13_6  and  O3_30
REMARK    3  A    between atoms: C2_11  and  C21_23
REMARK    4  A    between atoms: C5_16  and  O5_17
REMARK    5  A    between atoms: C7_20  and  O7_21
```

Step 5: Review the docking results

(5A) Look into the 10 predicted poses by selecting them from the ViewDock window.

(5B) Overlap the experimentally determined pose with the best predicted pose by opening the structure of PDB 1c8k [see **Step (1B)**].

(5C) Show only the CPB ligands (from the docking and the PDB), by taking these steps:

(5C-1) Select --> Residue --> CPB

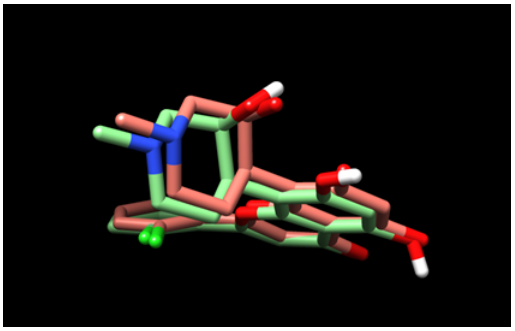
(5C-2) Select --> Invert (all models)

(5C-3) Actions --> Atoms/Bonds --> Hide

(5C-4) Actions --> Ribbon --> Hide

The resulting view will be similar to this:

Here, the predicted pose is in green and the experimental pose is in red.



(5D) Save the current view as “dock_cpb_img.png” by selecting File --> Save Image… and providing the file name.
(5E) Show the protein and other ligands that were hidden for image generation [in **(5C)-(5D)**], by taking these steps:

(5E-1) Select --> Structure --> Protein

(5E-2) Actions --> Ribbon --> Show

(5E-3) Select --> “Clear Selection”.

Step 6: Now dock one of the ligand molecules returned from the jupyter notebook.

(6A) Open the file containing the hit compound with the highest rank (pygm_lig0_11779854.mol).

(6B) Check the location of the ligand using the Model Panel [see **Step (4A)**] and rename the ligand as “ligand0”.

(6C) Perform the docking experiment using “ligand0”, in the similar manner to **Step (4E)**.

· Make sure that the **output file** should be set differently from the previous one in order not to overwrite the previous data. (For example, set the output file name “**dock_ligand0.pdbqt**”.

· Make sure that “**ligand0**” is selected as the ligand.

· Use the same simulator box location and size.

· Use the same values for other parameters.

Note: the ligands contained the .mol files are already energy-minimized and ready for docking. However, if a ligand does not have a 3-D structure (e.g., only the 2-D structure or SMILES string is available), a reasonable 3-D structure should be generated using the “Minimize Structure” and “Dock Prep” tools in Chimera.

(6D) Visualize the binding pose

(6D-1) Hide the protein structure by doing:

· Select --> Structure --> Protein, and

· Actions --> Ribbon --> Hide

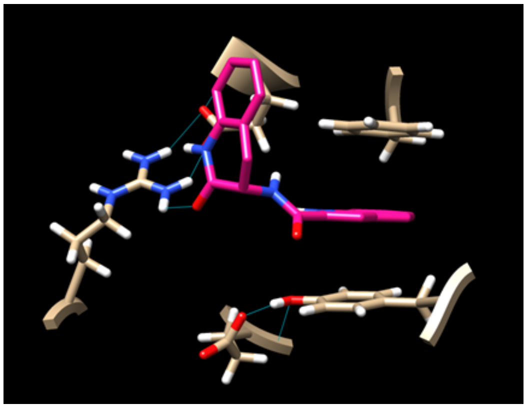
(6D-2) Show only the residues at the binding site by doing:

· Select --> Named Selections --> inhibition_site

· Actions --> Atoms/Bonds --> Show

· Actions --> Ribbon --> Show

(6D-3) Now you have a view similar to this:

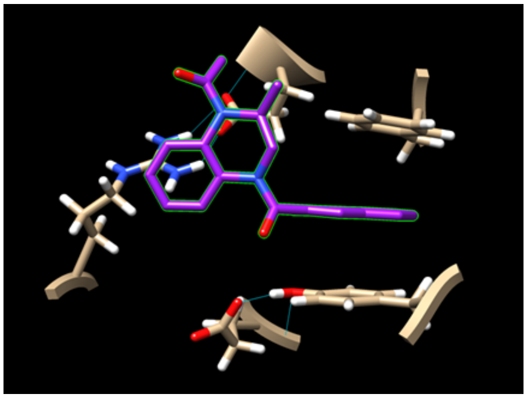


An aromatic ring of the ligand is stacked with the rings of TYR 163 and PHE 285, indicating some hydrophobic interaction. There is also a hydrogen bond between the ligand and ARG 770.

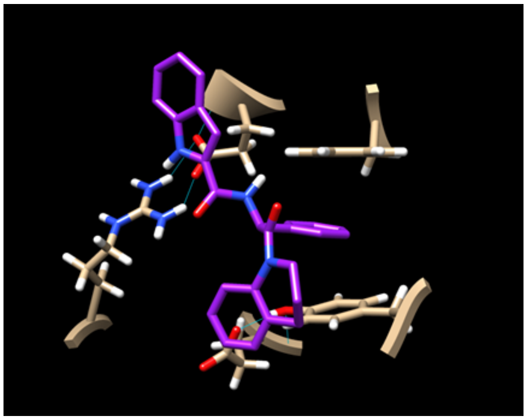
Save this image as “dock_ligand0_img.png”.

Step 7: For the remaining two ligands, repeat the docking experiment and generate the image of their best binding poses.

· dock_ligand1_img.png (for pygm_lig1_118078858.mol)



· dock_ligand2_img.png (for pygm_lig2_53013349.mol)



CHAPTER OVERVIEW

8: Machine-learning Basics

Hypothes.is Tag= f19OLCCc8

Note: Any annotation tagged **f19OLCCc8** on any open access page on the web will show at the bottom of this page.
You need to log in to https://web.hypothes.is/ to see annotations to the group 2019OLCCStu.

This page is under construction, and will hold content for module 8 of the Fall 2019 Cheminformatics OLCC.

Contact Bob Belford, rebelford@ualr.edu if you have any questions.

Topic hierarchy

8.1: Machine Learning Basics

Machine learning

Machine learning [1,2] is an application of artificial intelligence (AI) that provides computer systems with the ability to automatically learn from data, identify patterns, and make predictions or decisions with minimal human intervention. It focuses on the development of computational models that perform a specific task without using explicit instructions. Machine learning algorithms are now used in a wide variety of applications in many areas. This chapter reviews commonly used machine learning algorithms for classification, which will be used with bioactivity data archived in PubChem to build a predictive model for bioactivity of small molecules in the assignment (link to the lecture 8 notebook). There are several articles that provide thorough reviews on the application of machine learning in drug discovery and development, including the following papers:

- Concepts of Artificial Intelligence for Computer-Assisted Drug Discovery Yang, et al., Chem. Rev. 2019, 119, 10520−10594 doi:10.1021/acs.chemrev.8b00728
- Applications of machine learning in drug discovery and development, Vamathevan et al., Nat. Rev. Drug Discov. 2019, 18, 463-477. doi:10.1038/s41573-019-0024-5, PMCID: PMC6552674

Supervised and Unsupervised learning

There are two main categories of machine learning techniques: supervised learning and unsupervised learning.

Supervised learning

In supervised learning, a model is built from a training data set, which consists of a set of input data (represented with “descriptors” or “features”) and their known outputs (also called “labels” or “targets”). This model is used to predict an output for new input data (that are not included in the training data set). Simply put, supervised learning is about building a model, y=f(X), that predicts the value of y from input variables (X). [Note that X is in uppercase to reflect that input data are typically represented with a “vector” of multiple descriptors.] An example of supervised learning is to construct a model that predicts the binding affinity of small molecules against a given protein based on their molecular structures represented with molecular fingerprints. [Here, the molecular fingerprints correspond to the input and the binding affinity corresponds to the output.]

Supervised learning algorithms can be further divided into two categories (regression algorithms and classification algorithms), according to the type of the output data that supervised learning aims to predict.

- **Regression**
  Regression algorithms aim to build a mapping function from the input variable(s) (X) to the numerical or continuous output variable (y). The output variable can be an integer or a floating-point value and it usually represents a quantity, size, or strength. An example of regression problems is to predict the IC50 value of a compound against a target protein from its molecular structure.
- **Classification**
  Classification algorithms attempt to predict the “categorical” variable from the input variables. An example of classification problems is to predict whether a compound is agonistic, antagonistic, or inactive against a target protein.

Unsupervised learning

Unsupervised learning methods identify hidden patterns or intrinsic characteristics in the input data (X) and use them to cluster the data. Contrary to supervised learning, unsupervised learning does not use assigned labels to the input training data [that is, no output/target/label values (y) associated with the input data]. Therefore, it can be used to analyze unlabeled data. An example of the problems that unsupervised learning can handle is to group a set of compounds into small clusters according to their structural similarity (computed using molecular fingerprints) and identify structural features that characterize individual clusters. [For this task, the input data (X) is the molecular fingerprints for the compounds.]

Classification algorithms

Below are some commonly used machine learning algorithms for classification problems.

Naïve Bayes

Naïve Bayes classifiers [3,4] are a collection of supervised learning classifiers based on Bayes' theorem. It is called "Naïve" Bayes because it naively assumes that input features are conditionally independent of each other, which is *not* most likely true. Because the Naïve Bayes classifier is simple, yet effective, it has been commonly used in many applications, especially for text classification tasks (for example, spam mail detection). Read the following document about Naïve Bayes:

- Introduction to Naive Bayes Classification
  https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54

Decision tree

Decision Tree (DT) classifiers [5,6] are non-parametric supervised learning methods that predict the value of a target variable by learning simple decision rules inferred from the data features. While decision tree is easy to understand and interpret, it tends to result in overfitted models that do not generalize the data very well. Read the following web page about decision trees.

- Decision Trees in Machine Learning
  https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Random forest

Random forests [7-9] are an example of ensemble learning methods (which use multiple learning algorithms to achieve predictive performance better than the performance of any of the constituent learning algorithms alone). A random forest model consists of multiple decision tree models, each of which is built using a sample drawn with replacement (often called a bootstrap sample) from the training set. In addition, a random subset of the original set of features is considered when partitioning at each node during decision tree construction. The resulting random forest model predicts the output class of a new input by letting each classifier vote for a single class or by averaging their probabilistic prediction. Random forest classifiers alleviate data outfitting issues of decision trees. Learn more about random forests from this web page:

- Understanding Random Forest
  https://towardsdatascience.com/understanding-random-forest-58381e0602d2

k-Nearest neighbors

k-nearest neighbors classification [10,11] does not construct a "general" model that predicts an output value. Instead, it simply remembers all instances of the training data. When a new input is provided to the kNN classifier, it finds a pre-defined number (k) of training samples closest to the input and predicts its label (output) from the labels of these k-nearest neighbors (through plurality voting). [For this reason, kNN is an example of instance-based learning or non-generalizing learning. In addition, it is also called a lazy learning because all computations are deferred until a new input is provided for label prediction.

Finding nearest neighbors of an input involves computation of the distances between the input and the training data, which can be evaluated using several distance metrics, including those discussed in Chapter 6 (e.g., Euclidean distance, Manhattan distance, Jaccard/Tanimoto, and so on). Read this introductory material about kNN classifiers.

- Machine Learning Basics with the K-Nearest Neighbors Algorithm
  https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Support vector machine

A support vector machine (SVM) [12-14] is a classification algorithm widely used in many fields. SVMs attempts to find an optimal hyperplane that separates classes by the largest possible margin in the feature space. SVMs use kernel functions that project the data from a low-dimensional space into a higher-dimensional space. This projection makes the data more widely scattered in higher-dimensional spaces, and therefore often more easily separable. In addition, because almost all real-world data is not cleanly separable, the SVM algorithm uses the concept of the soft margin, which allows for some misclassifications. Read this web page to learn about SVM.

- SVM (Support Vector Machine) — Theory
  https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

Neural network

Artificial neural networks (ANNs) [15,16] are biologically inspired computer algorithms designed to simulate the way in which the human brain processes information. An artificial neural network is based on a collection of interconnected artificial neurons (nonlinear information processing units that loosely model the neurons in a biological brain). The interconnections between these neurons are called synapses or weights. These neurons are normally arranged in layers.

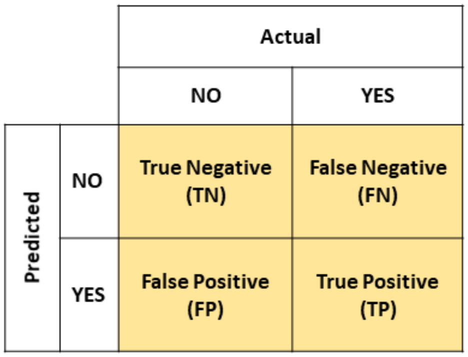
An example of supervised neural network algorithms is multi-layer perceptron (MLP) that trains using back-propagation. In this supervised training, the neural network processes the inputs and compares the expected outputs with its actual outputs. Errors are then propagated back through the network and the weights between the neurons are adjusted with respect to the errors. This process is repeated until the errors are minimized. Read the following web page to learn about ANNs.

- Everything You Need to Know About Artificial Neural Networks
  https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18245a1

Classification Performance Metrics

A confusion matrix is a tabular layout that summarizes prediction results from a classification model (often called classifier) and helps analyze the performance of the classifier. It shows the number of correct and incorrect predictions, broken down by each class. The information contained in a confusion matrix can be used to compute a wide variety of performance metrics as summarized in this Wikipedia document (https://en.Wikipedia.org/wiki/Confusion_matrix).

While a confusion matrix may be generated for multi-class problems, this chapter focuses on the one for a binary classification problem. Suppose that we want to identify active compounds against a given target protein, using a binary classifier that predicts whether a compound can change the activity of the target. The predictions may be “YES” (active) or “NO” (“inactive”). The confusion matrix for this classification problem is shown in this figure.



		Actual	
		NO	YES
Predicted	NO	True Negative (TN)	False Negative (FN)
	YES	False Positive (FP)	True Positive (TP)

Figure 8.1.1: Confusion Matrix

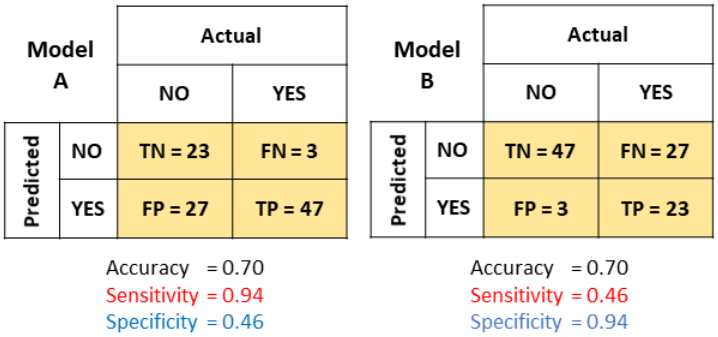
Let’s define the terms used in this confusion matrix.

- **True positive (TP)**:
  Cases in which active compounds are predicted to be active.
  (The model predicts “YES” for a compound and it is indeed “YES”)
- **True negative (TN)**:
  Cases in which inactive compounds are predicted to be inactive.
  (The model predicts “NO” for a compound and it is indeed “NO”).
- **False positive (FP)**:
  Cases in which inactive compounds are predicted to be active.
  (The model predicts “YES” for a compound but it is actually “NO”)
- **False negative (FN)**:
  Cases in which active compounds are predicted to be inactive.
  (The model predicted “NO” for a compound but it is actually “YES”)

Among the four cases presented in the confusion matrix, TP and TN are correct predictions and FP and FN are incorrect. Therefore, we can define the accuracy of the predictions using the following equation:

$$Accuracy(ACC) = \frac{\text{Correct Predictions}}{\text{All Predictions}} = \frac{TP+TN}{TP+TN+FP+FN} \tag{8.1.1}$$

This accuracy metric is commonly used as a performance measure for a classification model. However, in some cases, the accuracy alone is not enough to describe the performance of a classifier. For example, suppose that we have two models (A and B) that resulted in the prediction results summarized in the following confusion matrix.



Model A		Actual	
		NO	YES
Predicted	NO	TN = 23	FN = 3
	YES	FP = 27	TP = 47

Accuracy = 0.70
Sensitivity = 0.94
Specificity = 0.46

Model B		Actual	
		NO	YES
Predicted	NO	TN = 47	FN = 27
	YES	FP = 3	TP = 23

Accuracy = 0.70
Sensitivity = 0.46
Specificity = 0.94

Figure 8.1.2: Accuracy, Sensitivity and specificity for two different confusions matrices representing models A and B

While both models gave the same accuracy (0.70), Model A works better for predicting active compounds (47 out of 50 active compounds were correctly predicted to be active) and Model B works better for predicting inactive compounds (47 out of 50 inactives were correctly predicted to be inactive). This difference can be described using sensitivity and specificity, which are given in these equations:

$$Sensitivity = \frac{\text{Number of Correct YES Predictions}}{\text{Actual number of "YES"}} = \frac{TP}{TP+FN} \tag{8.1.2}$$

$$Specificity = \frac{\text{Number of Correct NO Predictions}}{\text{Actual number of "NO"}} = \frac{TN}{TN+FP} \tag{8.1.3}$$

Sensitivity is the ability of a model to correctly identify “YES”. A model with a high sensitivity (e.g., Model A) can correctly predict active compounds to be active. On the other hand, specificity is the model’s ability to correctly identify “NO”. If a model has a high specificity (e.g., Model B), it can correctly predict inactive compounds to be inactive.

Closely related to sensitivity and specificity are the true positive rate (TPR) and false positive rate (FPR).

$$\text{True Positive Rate} = \frac{\text{Number of Correct YES Predictions}}{\text{Actual number of "YES"}} = \frac{TP}{TP+FN} = Sensitivity \tag{8.1.4}$$

$$\begin{aligned} \text{False Positive Rate} &= \frac{\text{Number of Incorrect YES Predictions}}{\text{Actual number of "NO"}} \\ &= \frac{FP}{TN+FP} = \frac{FP+TN-TN}{TN+FP} = 1 - \frac{TN}{TN+FP} \\ &= 1 - \text{Specificity} \end{aligned} \tag{8.1.5}$$

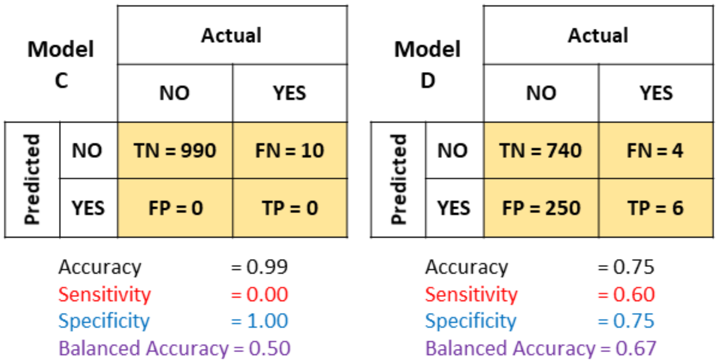
The TPR and FPR are used to generate the receiver-operating characteristic (ROC) curve [17-19]. While ROC curve was developed during World War II to detect a sonar signal from an enemy submarine, it is now used as a way to visualize the performance of any predictive model. It is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at *various threshold settings*. The area under the ROC curve (AUC), which ranges from 0 to 1, represents the degree of separability between two classes. A good model has an AUC value close to 1, which indicates a good degree of separability. An AUC value of 0.5 indicates that the model cannot separate two classes from each other. If a model has an AUC score close to 0, it means that the model does the opposite of what is supposed to do (that is, it predicts all active compounds to be inactive and all inactive compounds to be active).

Several web sites provide interactive tools that help understand how the ROC curve is generated and a nice example is the one created by Oleg Alenkin and Alex Rogozhnikov, which is available at:

http://arogozhnikov.github.io/RocCurve.html

More detailed instruction of this tool can be found at http://arogozhnikov.github.io/2015/10/05/roc-curve.html.

The accuracy score often gives an incomplete picture of a model's performance, especially when the model aims to predict rare events (e.g., finding an airline passenger who has illegal firearms in his/her baggage at an airport, or finding a hit compound from a large compound library). Suppose that we construct two models (Models C and D) for bioactivity prediction and test them against a compound set containing 10 active and 990 inactive compounds and the resulting confusion matrices look like the following:



Model C		Actual	
		NO	YES
Predicted	NO	TN = 990	FN = 10
	YES	FP = 0	TP = 0

Accuracy = 0.99
Sensitivity = 0.00
Specificity = 1.00
Balanced Accuracy = 0.50

Model D		Actual	
		NO	YES
Predicted	NO	TN = 740	FN = 4
	YES	FP = 250	TP = 6

Accuracy = 0.75
Sensitivity = 0.60
Specificity = 0.75
Balanced Accuracy = 0.67

Figure 8.1.3:

Model C has a greater accuracy score (0.99) than Model D (0.75). However, what Model C actually does is to predict *all* compounds to be inactive and fail to find any active compounds at all. If the goal is to identify active compounds for subsequent (in vitro or in vivo) experiments, Model C is not very helpful although its accuracy is close to 100%. Model D may be considered as a better one because it can identify at least some active compounds (although it results in more false positives). For this reason, the balanced accuracy [20] is often used when dealing with imbalanced data.

$$\begin{aligned}\text{Balanced Accuracy (BACC)} &= \frac{1}{2}\left(\frac{\text{Correct YES Predictions}}{\text{Actual YES Predictions}} + \frac{\text{Correct NO Predictions}}{\text{Actual NO Predictions}}\right) \\ &= \frac{1}{2}(\text{Sensitivity} + \text{Specificity})\end{aligned} \tag{8.1.6}$$

That is, the balanced accuracy for binary classification is the average of sensitivity and specificity. Note that the balanced accuracy of Model D (0.67) is greater than that of Model C (0.50).

8.5. Further Reading

- Concepts of Artificial Intelligence for Computer-Assisted Drug Discovery doi:10.1021/acs.chemrev.8b00728
- Applications of machine learning in drug discovery and development doi:10.1038/s41573-019-0024-5, PMCID: PMC6552674
- Introduction to Naive Bayes Classification https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54
- Decision Trees in Machine Learning https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052
- Understanding Random Forest https://towardsdatascience.com/understanding-random-forest-58381e0602d2
- Machine Learning Basics with the K-Nearest Neighbors Algorithm https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- Chapter 2 : SVM (Support Vector Machine) — Theory https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
- Everything You Need to Know About Artificial Neural Networks https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18245a1

- Performance Metrics for Classification problems in Machine Learning https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b
- Understanding AUC - ROC Curve https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

References

1. Yang X, Wang YF, Byrne R, Schneider G, Yang SY: **Concepts of Artificial Intelligence for Computer-Assisted Drug Discovery.** *Chem Rev* 2019, **119:**10520-10594.
2. Vamathevan J, Clark D, Czodrowski P, Dunham I, Ferran E, Lee G, Li B, Madabhushi A, Shah P, Spitzer M, Zhao SR: **Applications of machine learning in drug discovery and development.** *Nat Rev Drug Discov* 2019, **18:**463-477.
3. **Naive Bayes.** https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed Nov. 20, 2019.
4. **Introduction to Naive Bayes Classification.** https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54. Accessed Nov 20, 2019.
5. **Decision Trees.** https://scikit-learn.org/stable/modules/tree.html. Accessed Nov 19, 2019.
6. **Decision Trees in Machine Learning.** https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052. Accessed Nov. 20, 2019.
7. **Random Forests.** https://scikit-learn.org/stable/modules/ensemble.html#forest. Accessed Nov. 19, 2019.
8. **Understanding Random Forest.** https://towardsdatascience.com/understanding-random-forest-58381e0602d2. Accessed Nov. 20, 2019.
9. Breiman L: **Random forests.** *Mach Learn* 2001, **45:**5-32.
10. **Nearest Neighbors.** https://scikit-learn.org/stable/modules/neighbors.html. Accessed Nov. 20, 2019.
11. **Machine Learning Basics with the K-Nearest Neighbors Algorithm.** https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761. Accessed Nov. 20, 2019.
12. **Support Vector Machine.** https://scikit-learn.org/stable/modules/svm.html. Accessed Nov. 20, 2019.
13. **Chapter 2 : SVM (Support Vector Machine) — Theory.** https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. Accessed Nov. 20, 2019.
14. Boser BE, Guyon IM, Vapnik VN: **A training algorithm for optimal margin classifiers.** In *Book A training algorithm for optimal margin classifiers* (Editor ed.^eds.). pp. 144-152. City: ACM; 1992:144-152.
15. **Neural Network Models (Supervised).** https://scikit-learn.org/stable/modules/neural_networks_supervised.html. Accessed Nov 20, 2019.
16. **Everything You Need to Know About Artificial Neural Networks.** https://medium.com/technology-invention-and-more/everything-you-need-to-know-about-artificial-neural-networks-57fac18245a1. Accessed Nov 20, 2019.
17. Rao G: **What is an ROC curve?** *Journal of Family Practice* 2003, **52:**695-695.
18. Hoo ZH, Candlish J, Teare D: **What is an ROC curve?** *Emergency Medicine Journal* 2017, **34:**357-359.
19. Fawcett T: **An introduction to ROC analysis.** *Pattern Recognit Lett* 2006, **27:**861-874.
20. Brodersen KH, Ong CS, Stephan KE, Buhmann JM: **The Balanced Accuracy and Its Posterior Distribution.** In *2010 20th International Conference on Pattern Recognition*; *23-26 Aug. 2010*. 2010: 3121-3124.

8.2: Mathematica Assignment

8.3: Python Assignment

Machine Learning Basics

Downloadable Files

 lecture08_machine-learning.ipynb

- Download the ipynb file and run your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
  - This page is an html version of the above .ipynb file.
    - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

- **pandas**
- numpy
- time
- requests
- io
- rdkit
- sklearn

Objectives

- Build binary classification models that predict activity/inactivity of small molecules against human aromatase using supervised learning methods.
- Evaluate the performance of the developed models using performance measures.

Import bioactivity data from PubChem

In this notebook, we will develop a prediction model for small molecule's activity against human aromatase (https://pubchem.ncbi.nlm.nih.gov/protein/EAW77416), which is encoded by the CYP19A1 gene (https://pubchem.ncbi.nlm.nih.gov/gene/1588). The model will predict the activity of a molecule based on the structure of the molecule (represented with molecular fingerprints).

For model development, we will use the Tox21 bioassay data for human aromatase, archived in PubChem (https://pubchem.ncbi.nlm.nih.gov/bioassay/743139). The bioactivity data presented on this page can be downloaded by clicking the "Download" button available on this page and then read the data into a data frame. Alternatively, you can directly load the data into a data frame as shown in the cell below.

In [1]:

```
import pandas as pd
import numpy as np

url = 'https://pubchem.ncbi.nlm.nih.gov/assay/pcget.cgi?query=download&record_type=da
df_raw = pd.read_csv(url)
```

In [2]:

```
df_raw.head(7)
```

Out[2]:

	PUBCHEM_RESULT_TAG	PUBCHEM_SID	PUBCHEM_CID	PUBCHEM_ACTIVITY_OUTCOME	PUBCHEM_ACTIVITY_SCORE	PUBCHEM_ACTIVITY_URL	PUBCHEM_ASSAYDATA_COMMENT	Activity Summary	Antagonist Activity	Antagonist Potency (uM)	Antagonist Efficacy (%)	Viability Activity	Viability Potency (uM)	Viability Efficacy (%)	Sample Source
0	RESULT_TYPE	NaN	NaN	NaN	NaN	NaN	NaN	STRING	STRING	FLOAT	FLOAT	STRING	FLOAT	FLOAT	STRING
1	RESULT_DESCR	NaN	NaN	NaN	NaN	NaN	NaN	Type of compound activity based on both the ar...	Type of compound activity in the aromatase ant...	The concentration of sample yielding half-maxi...	Percent inhibition of aromatase.	Type of compound activity in the cell viabilit...	The concentration of sample yielding half-maxi...	Percent inhibition of cell viability.	Where sample was obtained.
2	RESULT_UNIT	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	MICROMOLAR	PERCENT	NaN	MICROMOLAR	PERCENT	NaN
3	1	144203552.0	12850184.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
4	2	144203553.0	89753.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
5	3	144203554.0	9403.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
6	4	144203555.0	13218779.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI

Note: Lines 0-2 provide the descriptions for each column (data type, descriptions, units, etc). These rows need be removed.

In [3]:

```
df_raw = df_raw[3:]
df_raw.head(5)
```

Out[3]:

	PUBCHEM_RESULT_TAG	PUBCHEM_SID	PUBCHEM_CID	PUBCHEM_ACTIVITY_OUTCOME	PUBCHEM_ACTIVITY_SCORE	PUBCHEM_ACTIVITY_URL	PUBCHEM_ASSAYDATA_COMMENT	Activity Summary	Antagonist Activity	Antagonist Potency (uM)	Antagonist Efficacy (%)	Viability Activity	Viability Potency (uM)	Viability Efficacy (%)	Sample Source
3	1	144203552.0	12850184.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
4	2	144203553.0	89753.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
5	3	144203554.0	9403.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
6	4	144203555.0	13218779.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
7	5	144203556.0	142766.0	Inconclusive	25.0	NaN	NaN	inconclusive antagonist (cytotoxic)	active antagonist	15.5454	-115.803	active antagonist	14.9601	-76.8218	NCI

The column names in this data frame contain white spaces and special characters. For simplicity, let's rename the columns (no spaces or special characters except for the "_" character.)

In [4]:

```
df_raw.columns
```

Out[4]:

```
Index(['PUBCHEM_RESULT_TAG', 'PUBCHEM_SID', 'PUBCHEM_CID',
       'PUBCHEM_ACTIVITY_OUTCOME', 'PUBCHEM_ACTIVITY_SCORE',
       'PUBCHEM_ACTIVITY_URL', 'PUBCHEM_ASSAYDATA_COMMENT', 'Activity Summary',
       'Antagonist Activity', 'Antagonist Potency (uM)',
       'Antagonist Efficacy (%)', 'Viability Activity',
       'Viability Potency (uM)', 'Viability Efficacy (%)', 'Sample Source'],
      dtype='object')
```

In [5]:

```
col_names_map = {'PUBCHEM_RESULT_TAG' : 'pc_result_tag',
                 'PUBCHEM_SID' : 'sid',
                 'PUBCHEM_CID' : 'cid',
                 'PUBCHEM_ACTIVITY_OUTCOME' : 'activity_outcome',
                 'PUBCHEM_ACTIVITY_SCORE' : 'activity_score',
                 'PUBCHEM_ACTIVITY_URL' : 'activity_url',
                 'PUBCHEM_ASSAYDATA_COMMENT' : 'assay_data_comment',
                 'Activity Summary' : 'activity_summary',
                 'Antagonist Activity' : 'antagonist_activity',
                 'Antagonist Potency (uM)' : 'antagonist_potency',
                 'Antagonist Efficacy (%)' : 'antagonist_efficacy',
                 'Viability Activity' : 'viability_activity',
                 'Viability Potency (uM)' : 'viability_potency',
                 'Viability Efficacy (%)' : 'viability_efficacy',
                 'Sample Source' : 'sample_source' }
```

In [6]:

```
df_raw = df_raw.rename(columns = col_names_map)
df_raw.columns
```

Out[6]:

```
Index(['pc_result_tag', 'sid', 'cid', 'activity_outcome', 'activity_score',
       'activity_url', 'assay_data_comment', 'activity_summary',
       'antagonist_activity', 'antagonist_potency', 'antagonist_efficacy',
       'viability_activity', 'viability_potency', 'viability_efficacy',
       'sample_source'],
      dtype='object')
```

Check the number of compounds for each activity group

First, we need to understand what our data look like. Especially, we are interested in the activity class of the tested compounds because we are developing a model that classifies small molecules according to their activities against the target. This information is available in the "**activity_outcome**" and "**activity_summary**" columns.

In [7]:

```
df_raw.groupby(['activity_outcome']).count()
```

Out[7]:

	pc_result_tag	sid	cid	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
activity_outcome														
Active	379	379	378	379	0	0	379	379	378	379	379	115	359	379
Inactive	7562	7562	7466	7562	0	0	7562	7562	0	7562	7562	324	7449	7562

	pc_result_tag	sid	cid	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
activity_outcome														
Inconclusive	2545	2545	2493	2545	0	0	2545	2545	2111	2136	2545	1206	2450	2545

Based on the data in the **activity_outcome** column, there are 379 actives, 7562 inactives, and 2545 inconclusives.

In [8]:

```
df_raw.groupby(['activity_outcome','activity_summary']).count()
```

Out[8]:

		pc_result_tag	sid	cid	activity_score	activity_url	assay_data_comment	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
activity_outcome	activity_summary													
Active	active antagonist	379	379	378	379	0	0	379	378	379	379	115	359	379
Inactive	inactive	7562	7562	7466	7562	0	0	7562	0	7562	7562	324	7449	7562
Inconclusive	active agonist	612	612	571	612	0	0	612	612	612	612	60	590	612
	inconclusive	44	44	44	44	0	0	44	0	0	44	19	42	44
	inconclusive agonist	414	414	409	414	0	0	414	212	223	414	12	397	414
	inconclusive agonist (cytotoxic)	59	59	59	59	0	0	59	41	45	59	59	59	59

		pc_result_tag	sid	cid	activity_score	activity_url	assay_data_comment	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
activity_outcome	activity_summary													
	inconclusive antagonist	367	367	364	367	0	0	367	227	230	367	8	313	367
	inconclusive antagonist (cytotoxic)	1049	1049	1046	1049	0	0	1049	1019	1026	1049	1048	1049	1049

Now, we can see that, in the **activity_summary** column, the inconclusive compounds are further classified into subclasses, which include:

- **active agonist**
- inconclusive
- inconclusive agonist
- inconclusive antagonist
- inconclusive agonist (cytotoxic)
- inconclusive antagonist (cytotoxic)

As implied in the title of this assay record (https://pubchem.ncbi.nlm.nih.gov/bioassay/743139), this assay aims to identify **aromatase inhibitors**. Therefore, all **active antagonists** (in the activity summary column) were declared to be **active** compounds (in the activity outcome column).

On the other hand, the assay also identified 612 **active agonists** (in the activity summary column), and they are declared to be **inconclusive** (in the activity outcome column).

With that said, "inactive" compounds in this assay means those which are neither active agonists nor active antagonist.

It is important to understand that the criteria used for determining whether a compound is active or not in a given assay are selected by the data source who submitted that assay data to PubChem. For the purpose of this assignment (which aims to develop a binary classifier that tells if a molecule is active or inactive against the target), we should clarify what we mean by "active" and "inactive".

- **active** : any compounds that can change (either increase or decrease) the activity of the target. This is equivalent to either **active antagonists** or **active agonists** in the activity summary column.
- **inactive** : any compounds that do not change the activity of the target. This is equivalent to **inactive** compounds in the activity summary column.

Select active/inactive compounds for model building

Now we want to select only the active and inactive compounds from the data frame (that is, active agonists, active antagonists, and inactives based on the "activity summary" column).

In [9]:

```
df = df_raw[ (df_raw['activity_summary'] == 'active agonist' ) |
             (df_raw['activity_summary'] == 'active antagonist' ) |
               (df_raw['activity_summary'] == 'inactive' ) ]

len(df)
```

Out[9]:

```
8553
```

In [10]:

```
print(len(df['sid'].unique()))
print(len(df['cid'].unique()))
```

```
8553
6864
```

Note that the number of CIDs is not the same as the number of SIDs. There are two important potential reasons for this observation.

First, not all substances (SIDs) in PubChem have associated compounds (CIDs) because some substances failed during structure standardization. [Remember that, in PubChem, substances are depositor-provided structures and compounds are unique structures extracted from substances through structure standardization.] Because our model will use structural information of molecules to predict their bioactivity, we need to remove substances without associated CIDs (i.e., no standardized structures).

Second, some compounds are associated with more than one substances. In the context of this assay, it means that a compound may be tested multiple times in different samples (which are designated as different substances). It is not uncommon that different samples of the same chemical may result in conflicting activities (e.g., active agonist in one sample but inactive in another sample). In this practice, we remove such compounds with conflicting activities.

Drop substances without associated CIDs.

First, check if there are subtances without associated CIDs.

In [11]:

```
df.isna().sum()
```

Out[11]:

```
pc_result_tag            0
sid                      0
cid                    138
activity_outcome         0
activity_score           0
activity_url          8553
assay_data_comment    8553
activity_summary         0
antagonist_activity      0
antagonist_potency    7563
antagonist_efficacy      0
viability_activity       0
viability_potency     8054
```

```
viability_efficacy      155
sample_source             0
dtype: int64
```

There are 138 records whose "cid" column is NULL, and we want to remove those records.

In [12]:

```
df = df.dropna( subset=['cid'] )
len(df)
```

Out[12]:

```
8415
```

In [13]:

```
print(len(df['sid'].unique()))
print(len(df['cid'].unique()))
```

```
8415
6863
```

In [14]:

```
df.isna().sum()   # Check if the NULL values disappeared in the "cid" column
```

Out[14]:

```
pc_result_tag             0
sid                       0
cid                       0
activity_outcome          0
activity_score            0
activity_url           8415
assay_data_comment     8415
activity_summary          0
antagonist_activity       0
antagonist_potency     7467
antagonist_efficacy       0
viability_activity        0
viability_potency      7919
viability_efficacy      154
sample_source             0
dtype: int64
```

Remove CIDs with conflicting activities

Now identify compounds with conflicting activities and remove them.

In [15]:

```
cid_conflict = []
idx_conflict = []

for mycid in df['cid'].unique() :

    outcomes = df[ df.cid == mycid ].activity_summary.unique()

    if len(outcomes) > 1 :

        idx_tmp = df.index[ df.cid == mycid ].tolist()
        idx_conflict.extend(idx_tmp)
        cid_conflict.append(mycid)

print("#", len(cid_conflict), "CIDs with conflicting activities [associated with", le
```

```
# 65 CIDs with conflicting activities [associated with 146 rows (SIDs).]
```

In [16]:

```
df.loc[idx_conflict,:].head(10)
```

Out[16]:

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
8	6	144203557.0	16043.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
5956	5954	144209507.0	16043.0	Active	43.0	NaN	NaN	active antagonist	active antagonist	54.4827	-73.4024	inconclusive antagonist	NaN	NaN	Sigma Aldrich
6850	6848	144210401.0	16043.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	SIGMA
52	50	144203601.0	443939.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI
6130	6128	144209681.0	443939.0	Active	61.0	NaN	NaN	active antagonist	active antagonist	1.65519	-115.932	active antagonist	12.1763	-120.598	Toronto Research

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
66	64	144203615.0	2170.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	BIOMOL
9118	9116	144212669.0	2170.0	Active	50.0	NaN	NaN	active antagonist	active antagonist	16.5803	-115.202	inconclusive antagonist	61.1306	-80.7706	SIGMA
106	104	144203655.0	2554.0	Inconclusive	20.0	NaN	NaN	active agonist	active agonist	2.87255	73.7025	inactive	NaN	0	SigmaAldrich
5920	5918	144209471.0	2554.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	SIGMA
6964	6962	144210515.0	2554.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	SIGMA

In [17]:

```
df = df.drop(idx_conflict)
```

In [18]:

```
df.groupby('activity_summary').count()
```

Out[18]:

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
activity_summary														
active agonist	537	537	537	537	537	0	0	537	537	537	537	58	517	537
active antagonist	343	343	343	343	343	0	0	343	342	343	343	108	326	343
inactive	7389	7389	7389	7389	7389	0	0	7389	0	7389	7389	318	7278	7389

In [19]:

```
print(len(df['sid'].unique()))
print(len(df['cid'].unique()))
```

```
8269
6798
```

Remove redundant data

The above code cells [in 3-(2)] do not remove compounds tested multiple times if the testing results are consistent [e.g., active agonist in all samples (substances)]. The rows corresponding to these compounds are redundant, so we want remove them except for only one row for each compound.

In [20]:

```
df = df.drop_duplicates(subset='cid')  # remove duplicate rows except for the first o
print(len(df['sid'].unique()))
print(len(df['cid'].unique()))
```

```
6798
6798
```

Adding "numeric" activity classes

In general, the inputs and outputs to machine learning algorithms need to have numerical forms.
In this practice, the input (molecular structure) will be represented with binary fingerprints, which already have numerical forms (0 or 1). However, the output (activity) is currently in a string format (e.g., 'active agonist', 'active antagonist'). Therefore, we want to add an additional, 'activity' column, which contains numeric codes representing the active and inactive compounds:

- 1 for actives (either active agonists or active antagonists)
- 0 for inactives

Note that we are merging the two classes "active agonist" and "active antagonist", because we are going to build a binary classifer that distinguish actives from inactives.

In [21]:

```
df['activity'] = [ 0 if x == 'inactive' else 1 for x in df['activity_summary'] ]
```

Check if the new column 'activity' is added to (the end of) the data frame.

In [22]:

```
df.head(3)
```

Out[22]:

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source	activity

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source	activity
3	1	144203552.0	12850184.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI	0
4	2	144203553.0	89753.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI	0
5	3	144203554.0	9403.0	Inactive	0.0	NaN	NaN	inactive	inactive	NaN	0	inactive	NaN	0	NCI	0

Double-check the count of active/inactive compounds.

In [23]:

```
df.groupby('activity_summary').count()
```

Out[23]:

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source	activity
activity_summary															
active agonist	451	451	451	451	451	0	0	451	451	451	451	44	432	451	451
active antagonist	291	291	291	291	291	0	0	291	290	291	291	88	275	291	291
inactive	6056	6056	6056	6056	6056	0	0	6056	0	6056	6056	269	5970	6056	6056

In [24]:

```
df.groupby('activity').count()
```

Out[24]:

	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source

activity	pc_result_tag	sid	cid	activity_outcome	activity_score	activity_url	assay_data_comment	activity_summary	antagonist_activity	antagonist_potency	antagonist_efficacy	viability_activity	viability_potency	viability_efficacy	sample_source
activity															
0	6056	6056	6056	6056	6056	0	0	6056	6056	0	6056	6056	269	5970	6056
1	742	742	742	742	742	0	0	742	742	741	742	742	132	707	742

Create a smaller data frame that only contains CIDs and activities.

Let's create a smaller data frame that only contains CIDs and activities. This data frame will be merged with a data frame containing molecular fingerprint information.

In [25]:

```
df_activity = df[['cid','activity']]
```

In [26]:

```
df_activity.head(5)
```

Out[26]:

	cid	activity
3	12850184.0	0
4	89753.0	0
5	9403.0	0
6	13218779.0	0
12	637566.0	0

Download structure information for each compound from PubChem

Now we want to get structure information of the compounds from PubChem (in isomeric SMILES).

In [27]:

```
cids = df.cid.astype(int).tolist()
```

In [28]:

```
chunk_size = 200
num_cids = len(cids)

if num_cids % chunk_size == 0 :
    num_chunks = int( num_cids / chunk_size )
else :
    num_chunks = int( num_cids / chunk_size ) + 1
```

```
print("# CIDs = ", num_cids)
print("# CID Chunks = ", num_chunks, "(chunked by ", chunk_size, ")")
```

```
# CIDs =  6798
# CID Chunks =  34 (chunked by  200 )
```

In [29]:

```
import time
import requests
from io import StringIO

df_smiles = pd.DataFrame()
list_dfs = []  # temporary list of data frames

for i in range(0, num_chunks) :

    idx1 = chunk_size * i
    idx2 = chunk_size * (i + 1)
    cidstr = ",".join( str(x) for x in cids[idx1:idx2] )

    url = ('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/' + cidstr + '/pro
    res = requests.get(url)
    data = pd.read_csv( StringIO(res.text), header=None, names=['smiles'] )
    list_dfs.append(data)

    time.sleep(0.2)

    if ( i % 5 == 0 ) :
        print("Processing Chunk ", i)

#    if ( i == 2 ) : break  #- for debugging

df_smiles = pd.concat(list_dfs,ignore_index=True)
df_smiles[ 'cid' ] = cids
df_smiles.head(5)
```

```
Processing Chunk  0
Processing Chunk  5
Processing Chunk  10
Processing Chunk  15
Processing Chunk  20
Processing Chunk  25
Processing Chunk  30
```

Out[29]:

	smiles	cid
0	C(C(=O)[C@H]([C@@H]([C@H](C(=O)[O-])O)O)O)O.C(...	12850184
1	C([C@H]([C@H]([C@@H]([C@H](C(=O)[O-])O)O)O)O)O...	89753
2	C[C@]12CC[C@H]3[C@H]([C@@H]1CC[C@@H]2OC(=O)CCC...	9403
3	C[C@@]12CC[C@@H](C1(C)C)C[C@H]2OC(=O)CSC#N	13218779
4	CC(=CCC/C(=C/CO)/C)C	637566

In [30]:

```
len(df_smiles)
```

Out[30]:

```
6798
```

In [31]:

```
df_smiles = df_smiles[['cid','smiles']]
df_smiles.head(5)
```

Out[31]:

	cid	smiles
0	12850184	C(C(=O)[C@H]([C@@H]([C@H](C(=O)[O-])O)O)O)O.C(...
1	89753	C([C@H]([C@H]([C@@H]([C@H](C(=O)[O-])O)O)O)O)O...
2	9403	C[C@]12CC[C@H]3[C@H]([C@@H]1CC[C@@H]2OC(=O)CCC...
3	13218779	C[C@@]12CC[C@@H](C1(C)C)C[C@H]2OC(=O)CSC#N
4	637566	CC(=CCC/C(=C/CO)/C)C

Generate MACCS keys from SMILES.

In [32]:

```
from rdkit import Chem
from rdkit.Chem import MACCSkeys
```

In [33]:

```
fps=dict()

for idx, row in df_smiles.iterrows() :

    mol = Chem.MolFromSmiles(row.smiles)

    if mol == None :
        print("Can't generate MOL object:", "CID", row.cid, row.smiles)
    else:
        fps[row.cid] = [row.cid] + list(MACCSkeys.GenMACCSKeys(mol).ToBitString())
```

```
Can't generate MOL object: CID 28145 [NH4+].[NH4+].F[Si-2](F)(F)(F)(F)F
Can't generate MOL object: CID 28127 F[Si-2](F)(F)(F)(F)F.[Na+].[Na+]
```

In [34]:

```
# Generate column names
fpbitnames = []

fpbitnames.append('cid')

for i in range(0,167):   # from MACCS000 to MACCS166
    fpbitnames.append( "maccs" + str(i).zfill(3) )

df_fps = pd.DataFrame.from_dict(fps, orient='index', columns=fpbitnames)
```

In [35]:

```
df_fps.head(5)
```

Out[35]:

	cid	maccs000	maccs001	maccs002	maccs003	maccs004	maccs005	maccs006	maccs007	maccs008	...	maccs157	maccs158	maccs159	maccs160	maccs161	maccs162	maccs163	maccs164	maccs165	maccs166
12850184	12850184	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
89753	89753	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
9403	9403	0	0	0	0	0	0	0	0	0	...	1	0	1	1	0	1	1	1	1	0
13218779	13218779	0	0	0	0	0	0	0	0	0	...	1	0	1	1	1	0	1	1	1	0

	cid	maccs000	maccs001	maccs002	maccs003	maccs004	maccs005	maccs006	maccs007	maccs008	...	maccs157	maccs158	maccs159	maccs160	maccs161	maccs162	maccs163	maccs164	maccs165	maccs166
637566	637566	0	0	0	0	0	0	0	0	0	...	1	0	0	1	0	0	0	1	0	0

5 rows × 168 columns

Merge activity data and fingerprint information

In [36]:

```
df_activity.head(3)
```

Out[36]:

	cid	activity
3	12850184.0	0
4	89753.0	0
5	9403.0	0

In [37]:

```
df_fps.head(3)
```

Out[37]:

	cid	maccs000	maccs001	maccs002	maccs003	maccs004	maccs005	maccs006	maccs007	maccs008	...	maccs157	maccs158	maccs159	maccs160	maccs161	maccs162	maccs163	maccs164	maccs165	maccs166
12850184	12850184	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
89753	89753	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
9403	9403	0	0	0	0	0	0	0	0	0	...	1	0	1	1	0	1	1	1	1	0

3 rows × 168 columns

In [38]:

```
df_data = df_activity.join(df_fps.set_index('cid'), on='cid')
```

In Section 5, there were two CIDs for which the MACCS keys could not be generated. They need to be removed from **df_data**.

In [39]:

```
df_data[df_data.isna().any(axis=1)]
```

Out[39]:

	cid	activity	maccs000	maccs001	maccs002	maccs003	maccs004	maccs005	maccs006	maccs007	...	maccs157	maccs158	maccs159	maccs160	maccs161	maccs162	maccs163	maccs164	maccs165	maccs166
2293	28145.0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9077	28127.0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2 rows × 169 columns

In [40]:

```
df_data = df_data.dropna()
len(df_data)
```

Out[40]:

```
6796
```

Save df_data in CSV for future use.

In [41]:

```
df_data.to_csv('df_data.csv')
```

Preparation for model building

Loading the data into X and y.

In [42]:

```
df_data.head(3)
```

Out[42]:

	cid	activity	maccs000	maccs001	maccs002	maccs003	maccs004	maccs005	maccs006	maccs007	...	maccs157	maccs158	maccs159	maccs160	maccs161	maccs162	maccs163	maccs164	maccs165	maccs166
3	12850184.0	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
4	89753.0	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
5	9403.0	0	0	0	0	0	0	0	0	0	...	1	0	1	1	0	1	1	1	1	0

3 rows × 169 columns

In [43]:

```
X = df_data.iloc[:,2:]
y = df_data['activity'].values
```

In [44]:

```
X.head(3)
```

Out[44]:

	maccs000	maccs001	maccs002	maccs003	maccs004	maccs005	maccs006	maccs007	maccs008	maccs009	...	maccs157	maccs158	maccs159	maccs160	maccs161	maccs162	maccs163	maccs164	maccs165	maccs166
3	0	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
4	0	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	1	0	1
5	0	0	0	0	0	0	0	0	0	0	...	1	0	1	1	0	1	1	1	1	0

3 rows × 167 columns

In [45]:

```
print(len(y))     # Number of all compounds
y.sum()           # Number of actives
```

```
6796
```

Out[45]:

```
742
```

Remove zero-variance features

Some features in X are not helpful in distinguishing actives from inactives, because they are set ON for all compounds or OFF for all compounds. Such features need to be removed because they would consume more computational resources without improving the model.

In [46]:

```
from sklearn.feature_selection import VarianceThreshold
```

In [47]:

```
X.shape  #- Before removal
```

Out[47]:

```
(6796, 167)
```

In [48]:

```
sel = VarianceThreshold()
X=sel.fit_transform(X)
X.shape  #- After removal
```

Out[48]:

```
(6796, 163)
```

In this case, four features had zero variances. Note that one of them is the first bit (maccs000) of the MACCS keys, which is added as a "dummy" to name each of bits 1~166 as maccs001, maccs002, ... maccs166.

Train-Test-Split (a 9:1 ratio)

Now split the data set into a training set (90%) and test set (10%). The training set will be used to train the model. The developed model will be tested against the test set.

In [49]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, shuffle=True, random_state=3100, stratify=y, test_size=0.1

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(y_train.sum(), y_test.sum())
```

```
(6116, 163) (680, 163) (6116,) (680,)
668 74
```

Balance the training set through downsampling

Check the dimension of the training data set.

In [50]:

```
print(len(y_train))
print(len(X_train))
print(len(X_train[0]))
```

```
6116
6116
163
```

Check the number of actives and inactives compound.

In [51]:

```
print("# inactives : ", len(y_train) - y_train.sum())
print("# actives   : ", y_train.sum())
```

```
# inactives :  5448
# actives   :  668
```

The data set is highly imbalanced [the inactive to active ratio is 8.16 (=5448 / 668)]. To address this issue, let's downsample the majority class (inactive compounds) to balance the data set.

In [52]:

```
# Indicies of each class' observations
idx_inactives = np.where( y_train == 0 )[0]
idx_actives   = np.where( y_train == 1 )[0]

# Number of observations in each class
num_inactives = len(idx_inactives)
num_actives   = len(idx_actives)

# Randomly sample from inactives without replacement
np.random.seed(0)
idx_inactives_downsampled = np.random.choice(idx_inactives, size=num_actives, replace

# Join together downsampled inactives with actives
X_train = np.vstack((X_train[idx_inactives_downsampled], X_train[idx_actives]))
y_train = np.hstack((y_train[idx_inactives_downsampled], y_train[idx_actives]))
```

It is noteworthy that **np.vstack** is used for X_train and **np.hstack** is used for Y_train. The direction of stacking is different because X_train is a 2-D array and y_train is a 1-D array.

Confirm that the downsampled data set has the correct dimension and active/inactive counts.

In [53]:

```
print("# inactives : ", len(y_train) - y_train.sum())
print("# actives   : ", y_train.sum())
```

```
# inactives :  668
# actives   :  668
```

In [54]:

```
print(len(y_train))
print(len(X_train))
print(len(X_train[0]))
```

```
1336
1336
163
```

Build a model using the training set.

Now we are ready to build predictive models using machine learning algorithms available in the scikit-learn library (https://scikit-learn.org/). This notebook will use Naive Bayes and decisiont tree, because they are relatively fast and simple.

In [55]:

```
from sklearn.naive_bayes import BernoulliNB        #-- Naive Bayes
from sklearn.tree import DecisionTreeClassifier    #-- Decision Tree
```

In [56]:

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
```

Naive Bayes

In [57]:

```
clf = BernoulliNB()             # set up the NB classification model
```

In [58]:

```
clf.fit( X_train ,y_train )    # Train the model by fitting it to the data.
```

Out[58]:

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

In [59]:

```
y_true, y_pred = y_train, clf.predict( X_train )    # Apply the model to predict the
```

In [60]:

```
CMat = confusion_matrix( y_true, y_pred )    #-- generate confusion matrix
print(CMat)     # [[TN, FP],
                #  [FN, TP]]
```

```
[[462 206]
 [199 469]]
```

In [61]:

```
acc  = accuracy_score( y_true, y_pred )

sens = CMat[ 1 ][ 1 ] / ( CMat[ 1 ][ 0 ] + CMat[ 1 ][ 1 ] )    # TP / (FN + TP)
spec = CMat[ 0 ][ 0 ] / ( CMat[ 0 ][ 0 ] + CMat[ 0 ][ 1 ] )    # TN / (TN + FP )
bacc = (sens + spec) / 2
```

```
y_score = clf.predict_proba( X_train )[:, 1]
auc = roc_auc_score( y_true, y_score )
```

In [62]:

```
print("#-- Accuracy          = ", acc)
print("#-- Balanced Accuracy = ", bacc)
print("#-- Sensitivity       = ", sens)
print("#-- Specificity       = ", spec)
print("#-- AUC-ROC           = ", auc)
```

```
#-- Accuracy          =  0.6968562874251497
#-- Balanced Accuracy =  0.6968562874251496
#-- Sensitivity       =  0.7020958083832335
#-- Specificity       =  0.6916167664670658
#-- AUC-ROC           =  0.7496985818781599
```

When applied to predict the activity of the training compounds, the NB classifier resulted in the accuracy of 0.70 and AUC-ROC of 0.75. However, the real performance of the model should be evaluated with the test set data, which are not used for model training.

In [63]:

```
y_true, y_pred = y_test, clf.predict(X_test)    #-- Apply the model to predict the te
```

In [64]:

```
CMat = confusion_matrix( y_true, y_pred )    #-- generate confusion matrix
print(CMat)    # [[TN, FP],
               #  [FN, TP]]
```

```
[[412 194]
 [ 28  46]]
```

In [65]:

```
acc  = accuracy_score( y_true, y_pred )

sens = CMat[ 1 ][ 1 ] / ( CMat[ 1 ][ 0 ] + CMat[ 1 ][ 1 ] )
spec = CMat[ 0 ][ 0 ] / ( CMat[ 0 ][ 0 ] + CMat[ 0 ][ 1 ] )
bacc = (sens + spec) / 2

y_score = clf.predict_proba( X_test )[:, 1]
auc = roc_auc_score( y_true, y_score )

print("#-- Accuracy          = ", acc)
print("#-- Balanced Accuracy = ", bacc)
print("#-- Sensitivity       = ", sens)
print("#-- Specificity       = ", spec)
print("#-- AUC-ROC           = ", auc)
```

```
#-- Accuracy          =  0.6735294117647059
#-- Balanced Accuracy =  0.6507448042101507
#-- Sensitivity       =  0.6216216216216216
#-- Specificity       =  0.6798679867986799
#-- AUC-ROC           =  0.724099099099099
```

For the test set, the accuracy is 0.67 and the AUC-ROC is 0.72. These values are somewhat smaller (by 0.03) than those for the training set. Also note that the accuracy is no longer the same as the balanced accruacy (which is the average of the sensitivity and specificity).

Some additional performance information may be obtained using **classification_report()**.

In [66]:

```
print( classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.68      0.79       606
           1       0.19      0.62      0.29        74

    accuracy                           0.67       680
   macro avg       0.56      0.65      0.54       680
weighted avg       0.86      0.67      0.73       680

```

Decision Tree

In [67]:

```
clf = DecisionTreeClassifier( random_state=0 )    # set up the DT classification mode
```

In [68]:

```
clf.fit( X_train ,y_train )    # Train the model by fitting it to the data (using the
```

Out[68]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [69]:

```
y_true, y_pred = y_train, clf.predict( X_train )    # Apply the model to predict the
```

In [70]:

```
CMat = confusion_matrix( y_true, y_pred )    #-- generate confusion matrix
print(CMat)    # [[TN, FP],
               #  [FN, TP]]
```

```
[[663   5]
 [  3 665]]
```

In [71]:

```
acc  = accuracy_score( y_true, y_pred )

sens = CMat[ 1 ][ 1 ] / ( CMat[ 1 ][ 0 ] + CMat[ 1 ][ 1 ] )    # TP / (FN + TP)
spec = CMat[ 0 ][ 0 ] / ( CMat[ 0 ][ 0 ] + CMat[ 0 ][ 1 ] )    # TN / (TN + FP )
bacc = (sens + spec) / 2

y_score = clf.predict_proba( X_train )[:, 1]
auc = roc_auc_score( y_true, y_score )
```

In [72]:

```
print("#-- Accuracy          = ", acc)
print("#-- Balanced Accuracy = ", bacc)
print("#-- Sensitivity       = ", sens)
print("#-- Specificity       = ", spec)
print("#-- AUC-ROC           = ", auc)
```

```
#-- Accuracy          =  0.9940119760479041
#-- Balanced Accuracy =  0.9940119760479043
#-- Sensitivity       =  0.9955089820359282
#-- Specificity       =  0.9925149700598802
#-- AUC-ROC           =  0.9998890691670551
```

When applied to predict the activity of the **training** compounds, the DT classifier resulted in very high scores (>0.99) for all five performance measures considered here. However, it does **not** necessarily mean that the model will perform very well for the **test** set compounds. Let's apply the model to the test set.

In [73]:

```
y_true, y_pred = y_test, clf.predict(X_test)    #-- Apply the model to predict the te
```

In [74]:

```
CMat = confusion_matrix( y_true, y_pred )    #-- generate confusion matrix
print(CMat)    # [[TN, FP],
                #  [FN, TP]]
```

```
[[422 184]
 [ 32  42]]
```

In [75]:

```
acc  = accuracy_score( y_true, y_pred )

sens = CMat[ 1 ][ 1 ] / ( CMat[ 1 ][ 0 ] + CMat[ 1 ][ 1 ] )
spec = CMat[ 0 ][ 0 ] / ( CMat[ 0 ][ 0 ] + CMat[ 0 ][ 1 ] )
bacc = (sens + spec) / 2

y_score = clf.predict_proba( X_test )[:, 1]
auc = roc_auc_score( y_true, y_score )

print("#-- Accuracy          = ", acc)
print("#-- Balanced Accuracy = ", bacc)
print("#-- Sensitivity       = ", sens)
print("#-- Specificity       = ", spec)
print("#-- AUC-ROC           = ", auc)
```

```
#-- Accuracy          =  0.6823529411764706
#-- Balanced Accuracy =  0.6319686022656319
#-- Sensitivity       =  0.5675675675675675
#-- Specificity       =  0.6963696369636964
#-- AUC-ROC           =  0.6298724467041299
```

When the DT model was applied to the test set, all performance measures were much worse than those for the training set. This is a typical example of **outfitting**.

Model building through cross-validation

In the above section, the models were developed using the default values for many optional hyperparamters, which cannot be learned by the training algorithm. For example, when building a decision tree model, one should specify how the tree should be deep, how many compounds should be allowed in a single leaf, what is the minimum number of compounds in a single leaf, etc.

The cells below demonstrate how to perform hyperparameter optimization through 10-fold cross-validation. In this example, five values for each of three hyperparameters used in decision tree are considered (max_depth, min_samples_split, and min_samples_leaf), resulting in a total of 125 combination of the parameter values (=5 x 5 x 5). For each combination, 10 models are generated (through 10-fold cross validation) and the average performance will be tracked. The goal is to find the parameter value combination that results in the highest average performance score (e.g., 'roc_auc') from the 10-fold cross validation.

In [76]:

```
from sklearn.model_selection import GridSearchCV
```

In [77]:

```
scores = [ 'roc_auc', 'balanced_accuracy' ]
```

In [78]:

```
ncvs = 10

max_depth_range         = np.linspace( 3, 7, num=5, dtype='int32' )
min_samples_split_range = np.linspace( 3, 7, num=5, dtype='int32' )
min_samples_leaf_range  = np.linspace( 2, 6, num=5, dtype='int32' )

param_grid = dict( max_depth=max_depth_range,
                   min_samples_split=min_samples_split_range,
                   min_samples_leaf=min_samples_leaf_range )

clf = GridSearchCV( DecisionTreeClassifier( random_state=0 ),
                    param_grid=param_grid, cv=ncvs, scoring=scores, refit='roc_auc',
                    return_train_score = True, iid=False)
```

In [79]:

```
clf.fit( X_train, y_train )
print("Best parameter set", clf.best_params_)
```

```
Best parameter set {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 3}
```

If necessary, it is possible to look into the performance data for each parameter value combination (stored in **clf.cv*results***), as shown in the following cell.

In [80]:

```
means_1a = clf.cv_results_['mean_train_roc_auc']
stds_1a  = clf.cv_results_['std_train_roc_auc']

means_1b = clf.cv_results_['mean_test_roc_auc']
stds_1b  = clf.cv_results_['std_test_roc_auc']

means_2a = clf.cv_results_['mean_train_balanced_accuracy']
stds_2a  = clf.cv_results_['std_train_balanced_accuracy']

means_2b = clf.cv_results_['mean_test_balanced_accuracy']
stds_2b  = clf.cv_results_['std_test_balanced_accuracy']

iterobjs = zip( means_1a, stds_1a, means_1b, stds_1b,
                means_2a, stds_2a, means_2b, stds_2b, clf.cv_results_['params'] )

for m1a, s1a, m1b, s1b, m2a, s2a, m2b, s2b, params in iterobjs :

    print( "Grid %r : %0.4f %0.04f %0.4f %0.04f %0.4f %0.04f %0.4f %0.04f"
           % ( params, m1a, s1a, m1b, s1b, m2a, s2a, m2b, s2b))
```

```
Grid {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 3} : 0.7714 0.0042 (
Grid {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 4} : 0.7714 0.0042 (
Grid {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 5} : 0.7714 0.0042 (
Grid {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 6} : 0.7714 0.0042 (
Grid {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 7} : 0.7714 0.0042 (
.
.
.

Grid {'max_depth': 7, 'min_samples_leaf': 6, 'min_samples_split': 4} : 0.8873 0.0072 (
Grid {'max_depth': 7, 'min_samples_leaf': 6, 'min_samples_split': 5} : 0.8873 0.0072 (
Grid {'max_depth': 7, 'min_samples_leaf': 6, 'min_samples_split': 6} : 0.8873 0.0072 (
Grid {'max_depth': 7, 'min_samples_leaf': 6, 'min_samples_split': 7} : 0.8873 0.0072 (
```

Uncomment the following cell to look into additional performance data stored in cv*result*.

In [81]:

```
#print(clf.cv_result_)
```

It is important to understand that each model built through 10-fold cross-validation during hyperparameter optimization uses only 90% of the compounds in the training set and the remaining 10% is used for testing that model. After all parameter value combinations are evaluated, the best parameter values are selected and used to rebuild a model from **all** compounds in the training set. **GridSearchCV()** takes care of this last step automatically. Therefore, there is no need to take an extra step to build a model using **cls.fit()** after hyperparameter optimization.

In [82]:

```
y_true, y_pred = y_train, clf.predict( X_train )    # Apply the model to predict the
```

In [83]:

```
CMat = confusion_matrix( y_true, y_pred )    #-- generate confusion matrix
print(CMat)    # [[TN, FP],
               #  [FN, TP]]
```

```
[[543 125]
 [179 489]]
```

In [84]:

```
acc  = accuracy_score( y_true, y_pred )

sens = CMat[ 1 ][ 1 ] / ( CMat[ 1 ][ 0 ] + CMat[ 1 ][ 1 ] )    # TP / (FN + TP)
spec = CMat[ 0 ][ 0 ] / ( CMat[ 0 ][ 0 ] + CMat[ 0 ][ 1 ] )    # TN / (TN + FP )
bacc = (sens + spec) / 2

y_score = clf.predict_proba( X_train )[:, 1]
auc = roc_auc_score( y_true, y_score )
```

In [85]:

```
print("#-- Accuracy          = ", acc)
print("#-- Balanced Accuracy = ", bacc)
print("#-- Sensitivity       = ", sens)
print("#-- Specificity       = ", spec)
print("#-- AUC-ROC           = ", auc)
```

```
#-- Accuracy          =  0.7724550898203593
#-- Balanced Accuracy =  0.7724550898203593
#-- Sensitivity       =  0.7320359281437125
#-- Specificity       =  0.812874251497006
#-- AUC-ROC           =  0.8336452095808383
```

Compare these performance data with those from section 8-(2) (for the training set). When the default values were used, the DT model gave >0.99 for all performance measures, but the current models (developed using hyperparameter optimization) have much lower values, ranging from 0.73 to 0.83. Again, however, what really matters is the performance against the test set, which contains the data not used for model training.

In [86]:

```
y_true, y_pred = y_test, clf.predict(X_test)    #-- Apply the model to predict the te
```

In [87]:

```
CMat = confusion_matrix( y_true, y_pred )    #-- generate confusion matrix
print(CMat)    # [[TN, FP],
               #  [FN, TP]]
```

```
[[457 149]
 [ 26  48]]
```

In [88]:

```
acc  = accuracy_score( y_true, y_pred )

sens = CMat[ 1 ][ 1 ] / ( CMat[ 1 ][ 0 ] + CMat[ 1 ][ 1 ] )
spec = CMat[ 0 ][ 0 ] / ( CMat[ 0 ][ 0 ] + CMat[ 0 ][ 1 ] )
bacc = (sens + spec) / 2

y_score = clf.predict_proba( X_test )[:, 1]
auc = roc_auc_score( y_true, y_score )

print("#-- Accuracy          = ", acc)
print("#-- Balanced Accuracy = ", bacc)
print("#-- Sensitivity       = ", sens)
print("#-- Specificity       = ", spec)
print("#-- AUC-ROC           = ", auc)
```

```
#-- Accuracy          =  0.7426470588235294
#-- Balanced Accuracy =  0.7013870305949514
#-- Sensitivity       =  0.6486486486486487
#-- Specificity       =  0.7541254125412541
#-- AUC-ROC           =  0.7496209080367496
```

Now we can see that the model from hyperparameter optimization gives better performance data against the test set, compared to the model developed using the default parameter values. Importantly, the model from hyperparameter optimization shows smaller differences in performance measures between the training and test sets, indicatiing that the issue of outffiting has been alleviated substantially.

Exercises

In this assignment, we will build predictive models using the same aromatase data.

step 1 Show the following information to make sure that the activity data in the **df_activity** data frame is still available.

- The first five lines of **df_activity**

In [89]:

```
# Write your code in this cell.
```

- The counts of active/inactive compounds in **df_activity**

In [90]:

```
# Write your code in this cell.
```

Step 2 Show the following information to make sure the structure data is still available.

- The first five lines of **df_smiles**

In [91]:

```
# Write your code in this cell.
```

- the number of rows of **df_smiles**

In [92]:

```
# Write your code in this cell.
```

Step 3 Generate the (ECFP-equivalent) circular fingerprints from the SMILES strings.

- Use RDKit to generate 1024-bit-long circular fingerprints.
- Set the radius of the circular fingerprint to 2.
- Store the fingerprints in a data_frame called **df_fps** (along with the CIDs).
- Print the dimension of **df_fps**.
- Show the first five lines of **df_fps**.

In [93]:

```
# Write your code in this cell
```

Step 4 Merge the **df_activity** and **df_fps** data frames into a data frame called **df_data**

- Join the two data frames using the CID column as keys.

- Remove the rows that have any NULL values (i.e., compounds for which the fingerprints couldn't be generated).
- Print the dimension of **df_data**.
- Show the first five lines of **df_data**.

In [94]:

```
# Write your code in this cell.
```

Step 5 Prepare input and output data for model building

- Load the fingerprint data into 2-D array (X) and the activity data into 1-D array (y).
- Show the dimension of X and y.

In [95]:

```
# Write your code in this cell.
```

- Remove zero-variance features from X (if any).

In [96]:

```
# Write your code in this cell.
```

- Split the data set into training and test sets (90% vs 10%) (using random_state=3100).
- Print the dimension of X and y for the training and test sets.

In [97]:

```
# Write your code in this cell.
```

- Balance the training data set through downsampling.
- Show the number of inactive/active compounds in the downsampled training set.

In [98]:

```
# Write your code in this cell.
```

Step 6 Building a Random Forest model using the balanced training data set.

- First read the followng documents about random forest (https://scikit-learn.org/stable/modules/ensemble.html#forest and https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier).
- Use 10-fold cross validation to select the best value for the "n_estimators" parameter that maximizes the **balanced accuracy**. Test 40 values from 5 to 200 with an increment of 5 (e.g., 5, 10, 15, 20, ..., 190, 195, 200).
- For parameters 'max_depth', 'min_samples_leaf', and 'min_samples_split', use the best values found in Section 9.
- For other parameters, use the default values.
- For each parameter value, print the mean balanced accuracies (for both training and test from cross validation).

In [99]:

```
# Write your code in this cell.
```

Step 7 Apply the developed RF model to predict the activity of the **training** set compounds.

- Report the confusion matrix.
- Report the accuracy, balanced accurayc, sensitivity, specificity, and auc-roc.

In [100]:

```
# Write your code in this cell.
```

Step 8 Apply the developed RF model to predict the activity of the **test** set compounds.

- Report the accuracy, balanced accurayc, sensitivity, specificity, and auc-roc.

In [101]:

```
# Write your code in this cell.
```

Step 9 Read a recent paper published in *Chem. Res. Toxicol.* (https://doi.org/10.1021/acs.chemrestox.7b00037) and answer the following questions (in no more than five sentences for each question).

- What different approaches did the paper take to develop prediction models (compared to those used in this notebook)?
- How different are the models reported in the paper from those constructed in this paper (in terms of the performance measures)?
- What would you do to develop models with improved performance?

Write your answers in this cell.

-
-
-

In [ ]:

```

```

Index

Glossary

Sample Word 1 | Sample Definition 1

Detailed Licensing

Overview

Title: Cheminformatics

Webpages: 92

Applicable Restrictions: Noncommercial

All licenses found:

- Undeclared: 53.3% (49 pages)
- CC BY-NC-SA 4.0: 46.7% (43 pages)

By Page

- Cheminformatics - *CC BY-NC-SA 4.0*
    - Front Matter - *Undeclared*
        - TitlePage - *Undeclared*
        - InfoPage - *Undeclared*
        - Table of Contents - *Undeclared*
        - Licensing - *Undeclared*
    - 1: Introduction - *CC BY-NC-SA 4.0*
        - 1.1: Introduction - *CC BY-NC-SA 4.0*
        - 1.2: Brief History of Cheminformatics - *CC BY-NC-SA 4.0*
        - 1.3: Introduction to Data and Databases - *CC BY-NC-SA 4.0*
        - 1.4: Installing Python - *CC BY-NC-SA 4.0*
        - 1.5: Installing R - *CC BY-NC-SA 4.0*
        - 1.6: Installing Mathematica - *Undeclared*
        - 1.7: Accessing PubChem through a Web Interface - *CC BY-NC-SA 4.0*
        - 1.8: Programmatic Access to the PubChem Database - *CC BY-NC-SA 4.0*
        - 1.9: Cheminformatics Resources - *CC BY-NC-SA 4.0*
        - 1.10: Python Assignment 1 - *CC BY-NC-SA 4.0*
        - 1.11: R Assignment 1 - *Undeclared*
        - 1.12: Mathematica Assignment 1 - *Undeclared*
    - 2: Representing Small Molecules on Computers - *CC BY-NC-SA 4.0*
        - 2.1: Introduction - *CC BY-NC-SA 4.0*
        - 2.2: Connection Tables - *CC BY-NC-SA 4.0*
        - 2.3: Molecular Graph Issues - *CC BY-NC-SA 4.0*
        - 2.4: Line Notation - *CC BY-NC-SA 4.0*
        - 2.5: Structural Data Files - *CC BY-NC-SA 4.0*
        - 2.6: Chemical Resolvers, Molecular Editors and Visualization - *CC BY-NC-SA 4.0*
        - 2.7: Python Assignment - *CC BY-NC-SA 4.0*
            - 2.7.1: Python Assignment 2A - *CC BY-NC-SA 4.0*
            - 2.7.2: Python Assignment 2B - *Undeclared*
        - 2.8: R Assignment - *Undeclared*
            - 2.8.1: R Assignment 2A - *Undeclared*
            - 2.8.2: R Assignment 2B - *Undeclared*
        - 2.9: Mathematica Assignment - *Undeclared*
            - 2.9.1: Mathematica Assignment 2A - *Undeclared*
            - 2.9.2: Mathematica Assignment 2B - *Undeclared*
    - 3: Database Resources in Cheminformatics - *CC BY-NC-SA 4.0*
        - 3.1: Database Basics - *CC BY-NC-SA 4.0*
        - 3.2: Database Management - *CC BY-NC-SA 4.0*
        - 3.3: Public Chemical Databases - *CC BY-NC-SA 4.0*
        - 3.4: Data Organization in PubChem as a Data Aggregator - *CC BY-NC-SA 4.0*
        - 3.5: Database Query Introduction - *CC BY-NC-SA 4.0*
        - 3.6: Special Notes on Using Public Chemical Databases - *CC BY-NC-SA 4.0*
        - 3.7: Mathematica Assignment - *Undeclared*
        - 3.8: Python Assignment - *CC BY-NC-SA 4.0*
        - 3.9: R Assignment - *Undeclared*
        - 3.10: R Assignment (binder test) - *Undeclared*
        - 3.11: Assignments - *CC BY-NC-SA 4.0*
        - Back Matter - *Undeclared*
    - 4: Searching Databases for Chemical Information - *CC BY-NC-SA 4.0*
        - 4.1: PubChem Web Interfaces for Text - *CC BY-NC-SA 4.0*
        - 4.2: Text Search in PubChem - *CC BY-NC-SA 4.0*
        - 4.3: Additional Data Retrieval Approaches in PubChem - *CC BY-NC-SA 4.0*
        - 4.4: Searching PubChem Using a Non-Textual Query - *CC BY-NC-SA 4.0*
        - 4.5: Programming Topics - *CC BY-NC-SA 4.0*
        - 4.6: Python Assignments - *CC BY-NC-SA 4.0*
        - 4.7: R Assignment - *Undeclared*
        - 4.8: Mathematica Assignment - *Undeclared*
    - 5: Quantitative Structure Property Relationships - *CC BY-NC-SA 4.0*
        - 5.1: Quantitative Structure-Property Relationships - *Undeclared*

    - 5.2: Similar-Structure, Similar-Property Principle - *Undeclared*
    - 5.3: Molecular Descriptors - *Undeclared*
        - 5.3.1: Exercise 5.1 solution - *Undeclared*
        - 5.3.2: Exercise 5.2 solution - *Undeclared*
    - 5.4: Mathematica Assignment - *Undeclared*
    - 5.5: Python Assignment - *Undeclared*
    - 5.6: R Assignment - *Undeclared*
- 6: Molecular Similarity - *CC BY-NC-SA 4.0*
    - 6.1: Molecular Descriptors - *Undeclared*
    - 6.2: Similarity Coefficients - *Undeclared*
    - 6.3: Discussion - *Undeclared*
    - 6.4: Python Assignment - *Undeclared*
    - 6.5: R Assignment - *Undeclared*
    - 6.6: Mathematica Assignment - *Undeclared*
- 7: Computer-Aided Drug Discovery and Design - *CC BY-NC-SA 4.0*
    - 7.1: Reading - *Undeclared*
    - 7.2: Mathematica Assignment - *Undeclared*
    - 7.3: Python Assignment-Virtual Screening - *Undeclared*
    - 7.4: R Assignment - *Undeclared*
    - 7.5: Molecular Docking Experiments - *Undeclared*
- 8: Machine-learning Basics - *CC BY-NC-SA 4.0*
    - 8.1: Machine Learning Basics - *Undeclared*
    - 8.2: Mathematica Assignment - *Undeclared*
    - 8.3: Python Assignment - *Undeclared*
- 9: 9. Appendix - *CC BY-NC-SA 4.0*
    - 9.1: Programming Operators - *CC BY-NC-SA 4.0*
    - 9.2: Jupyter Notebooks Tutorial - *CC BY-NC-SA 4.0*
    - 9.3: Introduction to Mathematica - *Undeclared*
    - 9.4: Python - *Undeclared*
- Back Matter - *Undeclared*
    - Index - *Undeclared*
    - Glossary - *Undeclared*
    - Detailed Licensing - *Undeclared*