

1.2: Introduction to Data Formations and Graphics

The 2012 General Social Survey

In this chapter, we will practice transforming survey data and making graphs using the transformed data. The survey data we will use is from the 2012 General Social Survey (GSS) conducted by the National Opinion Research Center (NORC). The GSS collects information from a nationally representative sample of the non-institutionalized US population (ages 18 years and older). Specifically, the NORC employs a stratified, multistage area probability sampling method to select households across the USA. The GSS monitors changes and constants in the American population's attitudes, behaviors, and attributes. The collected data covers various topics, including demographics, social attitudes, religion, politics, and the like. The dataset's variables we will focus on are related to perceptions of the police's use of force. Two of the survey questions asked in the 2012 GSS were "Are there any situations you can imagine in which you would approve of a policeman striking an adult male citizen?" and "Would you approve of a police officer striking a citizen who had said vulgar and obscene things to the policeman?" We will focus on these variables to see how people feel about police use of force.

Importing the Data Using the Haven Package

The haven package is a very useful tool within the R environment because it allows users to import data that has different file extensions created in popular statistical software packages such as SAS, SPSS, and Stata. The haven package enables users to import and export datasets effortlessly by bridging the gap between different data formats. As I mentioned in the first chapter, installing and loading the haven package is the first step.

```
install.packages('haven')  
library(haven)
```

"library(haven)" loads the haven package in R, providing access to functions like `read_sav()` and `write_sav()`. Now, we are ready to read the 2012 GSS data that will be manipulated for the purpose of practice. You will be able to download this file from [the shared Google Drive folder containing the 2012 GSS data](#). The downloaded 2012 GSS data is an SPSS data file, and SPSS data files are typically based on the .sav extension. I will read the data from the file located on the following path: "C:/Users/75JCHOI/OneDrive - West Chester University of PA/WCU Research/R/data/GSS. 2012.sav". You will have to use your own file location to read the data. Please watch the following video to learn how to set up a working directory in R.



Setting up Working Directories in R. [Watch "Setting up Working Directories in R" on YouTube](#) (closed captioned)

I will assign the read data to a new data frame labeled GSS.2012. Please note that we could read this SPSS file because we used the haven package, which allows users to import data from different software programs.

```
GSS.2012 <- read_sav("C:/Users/75JCHOI/OneDrive - West Chester University of PA/WCU  
Research/R/data/GSS. 2012.sav")
```

View Function

To check what the imported data looks like, use the `view()` function. The `view()` function opens a data viewer window in R, allowing you to interactively explore the contents of a data frame.

```
view(GSS.2012)
```

The syntax above displays the contents of the GSS.2012 data frame. If you check the data, you may recognize that each row represents an individual who participated in the survey, and columns represent different variables. For instance, the column labeled as RACE refers to a respondent's race. You will see five variables included in this dataset: RACE, SEX, POLHITOK, POLABUSE, and AGE. Although many more variables were included in the 2012 GSS, I retained only our variables of interest here to simplify the data management process.

You will see a number below each variable. For instance, respondent's race (RACE) was coded as 1 = White, 2 = Black, and 3 = Other. SEX represents a respondent's sex (1 = Male and 2 = Female). POLHITOK represents a respondent's response to the following question: 'Are there any situations you can imagine in which you would approve of a policeman striking an adult male citizen?' The response options for this item were 1 (yes) and 2 (no). POLABUSE represents a respondent's response to the following question: 'Would you approve of a police officer striking a citizen who had said vulgar and obscene things to the policeman?' The response options for this item were also 1 (yes) and 2 (no). Finally, AGE represents a respondent's age.

Summary Function

There may be too much information to review if you try to read the entire content displayed from the view function. So, reviewing summarized information from our data frame may be better. The `summary()` function gives a quick rundown of what's inside a data frame, showing both the layout and key statistics for each variable.

```
summary(object = GSS.2012)
```

The results show the minimum, 1st quartile, median, mean, 3rd quartile, and maximum values. What about NA's? NA's indicate the number of missing values in each variable. For example, the summary statistics provided for AGE are as follows:

- Minimum (Min.): 18.00
- 1st Quartile (1st Qu.): 33.00
- Median: 47.00
- Mean: 48.19
- 3rd Quartile (3rd Qu.): 61.00
- Maximum (Max.): 89.00
- Number of missing values (NA's): 5

These statistics give insights into the distribution of ages in the dataset. For instance, the youngest person in the dataset is 18 years old. A quarter of the people are 33 or younger. The median age is 47, meaning half are younger, and half are older than that. The average age is a bit higher than the median age at 48.19 years, which suggests there might be some older individuals raising the average. Three-quarters of the people are 61 or younger, and the oldest person is 89. Five missing age values that need to be handled based on what kind of analysis or modeling you're doing.

Categorical vs Numerical Variables

What are categorical and numerical variables? In statistical analysis, we use these two types of variables to handle different kinds of data. Categorical variables deal with qualitative data, meaning they sort observations into specific groups or categories. Think of things like gender, race, marital status, education level, and car type. On the other hand, numerical variables handle quantitative data. These are numbers that show amounts or quantities, like age, height, income, or temperature. Examples of numerical variables include age, height, weight, income, and temperature.

Our summary statistics from the summary function suggest that some issues arise with the way our variables are coded. For instance, the RACE variable has been given summary statistics like "Min.", "1st Qu.", "Median", "Mean", "3rd Qu.", "Max.", and "NA's". This is incorrect because RACE is being treated as a numerical variable instead of a categorical one.

There is no average for the categorical variable of race. Calculating the mean of race categories (e.g., "WHITE," "BLACK," "OTHER") would imply a numeric relationship between these categories. However, race categories are qualitative and do not have inherent numeric values. Treating them as numeric would lead to misinterpretation and potentially incorrect conclusions. On the

other hand, calculating the average age, which is 48.19 years, makes perfect sense. However, it's illogical to apply the mean to categorical variables like race. Race categories (e.g., White, Black, Asian) don't have a numerical order or value that can be averaged as numerical values do.

We need to recode RACE and SEX, so that they are classified as categorical variables instead of numerical variables. Even POLHITOK and POLABUSE are categorical because the responses "yes" and "no" represent distinct categories rather than numerical values with inherent order or magnitude. While the coded values 1 and 2 for POLHITOK and POLABUSE may appear to be numerical, they are used here to represent categories rather than quantities. So, these variables also need to be transformed properly.

Dplyr Package

We will use the dplyr package to recode some of our variables. The dplyr package provides a set of grammars of data manipulation, offering a consistent set of verbs (functions) that help us solve common data manipulation challenges. We do not need to install this R package. Do you remember that we installed the tidyverse package in the previous chapter? The tidyverse package is a collection of several packages, including dplyr and ggplot2. So dplyr was already installed when we installed the tidyverse package.

One of the dplyr verbs is mutate(), and this function can be used to add new variables (columns) that are functions of existing variables. First, we will recode RACE to a factor, ensuring that it is treated as a categorical variable. However, we want to create a copy of the dataset in case we make mistakes and need to restore the original file. Again, you remember that we can create a new dataset using the left-arrow operator. This new dataset will be named "GSS.2012.cleaned".

```
GSS.2012.cleaned<-GSS.2012 %>%  
mutate(RACE = as.factor(x = RACE))
```

The first line, `GSS.2012.cleaned <- GSS.2012 %>%`, creates a new dataset called "GSS.2012.cleaned". This new dataset will include all our data manipulations after the `%>%` operator. The `%>%` operator, pronounced "pipe," lets us string multiple functions together in a sequence. This makes the code easier to read and understand, especially when handling complex data tasks. The second line, `mutate(RACE = as.factor(x = RACE))`, uses the `mutate()` function from dplyr to modify the "RACE" variable in the "GSS.2012" dataset. It changes the "RACE" variable into a factor using the `as.factor()` function, ensuring it is treated as a categorical variable. The `x = RACE` argument specifies the variable to be transformed. The first RACE in the parenthesis indicates the name of the variable in the new dataset. You can keep it as it is or create a new variable with a new name. This line of code ensures that the "RACE" variable is treated as a categorical factor variable in the dataset.

We can examine whether our data transformation succeeded using the summary function we learned earlier.

```
summary(GSS.2012.cleaned$RACE)
```

We can specify the variable we want to see in the dataset using the `$` operator. You will notice that no information is presented regarding minimum, 1st quartile, median, mean, 3rd quartile, and maximum values. Instead, you can see categories 1, 2, and 3:

Category 1: This category has a frequency count of 1477.

Category 2: This category has a frequency count of 301.

Category 3: This category has a frequency count of 196.

However, because we do not know which racial category is associated with each category, we will need to recode the values in the RACE variable using the following syntax:

```
GSS.2012.cleaned<-GSS.2012.cleaned %>%  
mutate(RACE = recode(.x = RACE, "1" = "WHITE")) %>%  
mutate(RACE = recode(.x = RACE, "2" = "BLACK")) %>%  
mutate(RACE = recode(.x = RACE, "3" = "OTHER"))
```

"`mutate(RACE = recode(.x = RACE, "1" = "WHITE"))`" recoded the values in the "RACE" variable by replacing a value of "1" with "WHITE". Similarly, `mutate(RACE = recode(.x = RACE, "2" = "BLACK"))` recodes the "RACE" variable again. This time, it replaces any value of "2" with "BLACK". Finally, `mutate(RACE = recode(.x = RACE, "3" = "OTHER"))` replaced any value of "3" with "OTHER".

We can review our results using the summary function.

```
summary(GSS.2012.cleaned$RACE)
```

The output indicated that there were 1477 whites, 301 blacks, and 196 others.

Even though we broke down this data transformation process into several steps (converting the RACE variable to a factor and then recoding its values to more descriptive labels (“WHITE”), you do not need to create multiple syntaxes because the pipe operator in R can simplify and streamline our codes. For instance, the following syntax can perform a series of data manipulations at the same time.

```
GSS.2012.cleaned<-GSS.2012 %>%  
mutate(RACE = as.factor(x = RACE))%>%  
mutate(RACE = recode(.x = RACE, "1" = "WHITE")) %>%  
mutate(RACE = recode(.x = RACE, "2" = "BLACK")) %>%  
mutate(RACE = recode(.x = RACE, "3" = "OTHER"))  
summary(GSS.2012.cleaned$RACE)
```

In sum, we can chain multiple actions together, making our code more concise and readable.

Ggplot2 Package

You may wonder if we can visualize the information regarding the RACE variable in the 2012 GSS data in a graph. One of the most popular tools used to create such data visualizations is the ggplot2 package. As with the dplyr, the ggplot2 package is a part of tidyverse. Therefore, you do not need to install or load ggplot2 as long as the tidyverse package is installed and loaded.

The following is a reusable template for making graphs with ggplot2.

```
ggplot(data = <DATA> +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

To create a graph, substitute the section within the angle brackets in the provided code with either a dataset, a geom function, or a set of mappings. For instance, let's create a bar chart to see how the different racial categories are distributed. Bar charts are handy for showing how often each category appears in a variable. Here's the code using ggplot2 to make the chart:

```
ggplot(GSS.2012.cleaned, aes(x = RACE)) +  
geom_bar()
```

This code initializes a ggplot object with our dataset "GSS.2012.cleaned" and tells ggplot to use the "RACE" variable for the x-axis. Then, it adds bars to the plot with geom_bar(). By default, this function counts how many times each category appears in the "RACE" variable and makes a bar for each one. Now, let's make the chart a bit more informative by adding labels:

```
ggplot(GSS.2012.cleaned, aes(x = RACE)) +  
geom_bar() +  
labs(x = "Race", y = "Frequency", title = "Distribution of Race")
```

Here, labs() sets the labels for the x-axis ("Race"), y-axis ("Frequency"), and the plot title ("Distribution of Race"), which gives more context to the chart. With “+” operator, we can add more aesthetic options to the bar graph. You can achieve the same result using the pipe operator:

```
GSS.2012.cleaned %>%  
ggplot(aes(x = RACE)) +  
geom_bar() +  
labs(x = "Race", y = "Frequency", title = "Distribution of Race")
```

Using the pipe operator can give you additional flexibility when more complex tasks need to be added. For example, we can create a new store place using the pipe operator and ggplot.

```
race.bar <- GSS.2012.cleaned %>%  
  ggplot(aes(x = RACE, fill = RACE)) +  
  geom_bar()  
race.bar
```

race.bar stores the resulting bar chart plot object, allowing you to further customize or display the plot as needed.

Now, I want to add the colors to the bars in the chart. The bars will be filled with "beige," "black," and "gray" colors, respectively, corresponding to the different levels of the "RACE" variable.

```
race.bar <- GSS.2012.cleaned %>%  
  ggplot(aes(x = RACE, fill = RACE)) +  
  geom_bar() +  
  scale_fill_manual(values = c("beige", "black", "gray"),  
    guide = FALSE) +  
  labs(x = "Race", y = "Frequency", title = "Distribution of Race")  
race.bar
```

scale_fill_manual(values = c("beige," "black," "gray")) sets the fill colors of the bars manually. The values argument specifies a vector of colors to use for filling

the bars.

guide = FALSE specifies whether to include a legend or guide for the fill scale.

Setting it to FALSE removes the legend, so the colors will be applied directly to the bars without a corresponding legend in the plot.

Finally, you may want to remove the gray background that makes the graph look less professional. You can remove background grids, axis lines, and other non-essential elements, resulting in a clean and simple appearance in ggplot.

```
race.bar <- GSS.2012.cleaned %>%  
  ggplot(aes(x = RACE, fill = RACE)) +  
  geom_bar() +  
  scale_fill_manual(values = c("beige", "black", "grey"),  
    guide = FALSE) +  
  labs(x = "Race", y = "Frequency", title = "Distribution of Race")  
+  
  theme_minimal()  
race.bar
```

theme_minimal() applies the minimalistic theme to the plot created by ggplot2, resulting in a plot with a clean and uncluttered appearance.

In this chapter, I introduced several useful data transformation and visualization packages. Additionally, we reviewed some basic codes and functions that can help you recode the variables and visualize data. In the next chapter, we will learn how to produce descriptive statistics.

This page titled [1.2: Introduction to Data Formations and Graphics](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Jaeyong Choi](#) ([The Pennsylvania Alliance for Design of Open Textbooks \(PA-ADOPT\)](#)) via [source content](#) that was edited to the style and standards of the LibreTexts platform.