5.8: Handling Missing Values

There's one last topic that I want to discuss briefly in this chapter, and that's the issue of ***missing data***. Real data sets very frequently turn out to have missing values: perhaps someone forgot to fill in a particular survey question, for instance. Missing data can be the source of a lot of tricky issues, most of which I'm going to gloss over. However, at a minimum, you need to understand the basics of handling missing data in R.

5.8.1 single variable case

Let's start with the simplest case, in which you're trying to calculate descriptive statistics for a single variable which has missing data. In R, this means that there will be `NA` values in your data vector. Let's create a variable like that:

```
> partial <- c(10, 20, NA, 30)
```

Let's assume that you want to calculate the mean of this variable. By default, R assumes that you want to calculate the mean using all four elements of this vector, which is probably the safest thing for a dumb automaton to do, but it's rarely what you actually want. Why not? Well, remember that the basic interpretation of `NA` is "I don't know what this number is". This means that `1 + NA = NA` : if I add 1 to some number that I don't know (i.e., the `NA`) then the answer is *also* a number that I don't know. As a consequence, if you don't explicitly tell R to ignore the `NA` values, and the data set does have missing values, then the output will itself be a missing value. If I try to calculate the mean of the `partial` vector, without doing anything about the missing value, here's what happens:

```
> mean( x = partial )
[1] NA
```

Technically correct, but deeply unhelpful.

To fix this, all of the descriptive statistics functions that I've discussed in this chapter (with the exception of `cor()` which is a special case I'll discuss below) have an optional argument called `na.rm` , which is shorthand for "remove NA values". By default, `na.rm = FALSE` , so R does nothing about the missing data problem. Let's try setting `na.rm = TRUE` and see what happens:

When calculating sums and means when missing data are present (i.e., when there are `NA` values) there's actually an additional argument to the function that you should be aware of. This argument is called `na.rm` , and is a logical value indicating whether R should ignore (or "remove") the missing data for the purposes of doing the calculations. By default, R assumes that you want to keep the missing values, so unless you say otherwise it will set `na.rm = FALSE` . However, R assumes that `1 + NA = NA` : if I add 1 to some number that I don't know (i.e., the `NA`) then the answer is *also* a number that I don't know. As a consequence, if you don't explicitly tell R to ignore the `NA` values, and the data set does have missing values, then the output will itself be a missing value. This is illustrated in the following extract:

```
> mean( x = partial, na.rm = TRUE )
[1] 20
```

Notice that the mean is `20` (i.e., `60 / 3`) and *not* `15` . When R ignores a `NA` value, it genuinely ignores it. In effect, the calculation above is identical to what you'd get if you asked for the mean of the three-element vector `c(10, 20, 30)` .

As indicated above, this isn't unique to the `mean()` function. Pretty much all of the other functions that I've talked about in this chapter have an `na.rm` argument that indicates whether it should ignore missing values. However, its behaviour is the same for all these functions, so I won't waste everyone's time by demonstrating it separately for each one.

5.8.2 Missing values in pairwise calculations

I mentioned earlier that the `cor()` function is a special case. It doesn't have an `na.rm` argument, because the story becomes a lot more complicated when more than one variable is involved. What it does have is an argument called `use` which does roughly the same thing, but you need to think little more carefully about what you want this time. To illustrate the issues, let's open

up a data set that has missing values, `parenthood2.Rdata`. This file contains the same data as the original parenthood data, but with some values deleted. It contains a single data frame, `parenthood2`:

```
> load( "parenthood2.Rdata" )
> print( parenthood2 )
  dan.sleep baby.sleep dan.grump day
1      7.59         NA        56   1
2      7.91      11.66        60   2
3      5.14       7.92        82   3
4      7.71       9.61        55   4
5      6.68       9.75        NA   5
6      5.99       5.04        72   6
BLAH BLAH BLAH
```

If I calculate my descriptive statistics using the `describe()` function

```
> describe( parenthood2 )
           var   n  mean    sd median trimmed   mad   min    max    BLAH
dan.sleep    1  91  6.98  1.02   7.03    7.02  1.13  4.84   9.00    BLAH
baby.sleep   2  89  8.11  2.05   8.20    8.13  2.28  3.25  12.07    BLAH
dan.grump    3  92 63.15  9.85  61.00   62.66 10.38 41.00  89.00    BLAH
day          4 100 50.50 29.01  50.50   50.50 37.06  1.00 100.00    BLAH
```

we can see from the `n` column that there are 9 missing values for `dan.sleep`, 11 missing values for `baby.sleep` and 8 missing values for `dan.grump`.[84] Suppose what I would like is a correlation matrix. And let's also suppose that I don't bother to tell R how to handle those missing values. Here's what happens:

```
> cor( parenthood2 )
           dan.sleep baby.sleep dan.grump day
dan.sleep          1         NA        NA  NA
baby.sleep        NA          1        NA  NA
dan.grump         NA         NA         1  NA
day               NA         NA        NA   1
```

Annoying, but it kind of makes sense. If I don't *know* what some of the values of `dan.sleep` and `baby.sleep` actually are, then I can't possibly *know* what the correlation between these two variables is either, since the formula for the correlation coefficient makes use of every single observation in the data set. Once again, it makes sense: it's just not particularly *helpful*.

To make R behave more sensibly in this situation, you need to specify the `use` argument to the `cor()` function. There are several different values that you can specify for this, but the two that we care most about in practice tend to be `"complete.obs"` and `"pairwise.complete.obs"`. If we specify `use = "complete.obs"`, R will completely ignore all cases (i.e., all rows in our `parenthood2` data frame) that have any missing values at all. So, for instance, if you look back at the extract earlier when I used the `head()` function, notice that observation 1 (i.e., day 1) of the `parenthood2` data set is missing the value for `baby.sleep`, but is otherwise complete? Well, if you choose `use = "complete.obs"` R will ignore that row completely: that is, even when it's trying to calculate the correlation between `dan.sleep` and `dan.grump`, observation 1 will be ignored, because the value of `baby.sleep` is missing for that observation. Here's what we get:

```
> cor(parenthood2, use = "complete.obs")
             dan.sleep baby.sleep   dan.grump         day
dan.sleep   1.00000000  0.6394985 -0.89951468  0.06132891
baby.sleep  0.63949845  1.0000000 -0.58656066  0.14555814
dan.grump  -0.89951468 -0.5865607  1.00000000 -0.06816586
day         0.06132891  0.1455581 -0.06816586  1.00000000
```

The other possibility that we care about, and the one that tends to get used more often in practice, is to set `use = "pairwise.complete.obs"`. When we do that, R only looks at the variables that it's trying to correlate when determining what to drop. So, for instance, since the only missing value for observation 1 of `parenthood2` is for `baby.sleep` R will only drop observation 1 when `baby.sleep` is one of the variables involved: and so R keeps observation 1 when trying to correlate `dan.sleep` and `dan.grump`. When we do it this way, here's what we get:

```
> cor(parenthood2, use = "pairwise.complete.obs")
             dan.sleep  baby.sleep    dan.grump          day
dan.sleep   1.00000000  0.61472303 -0.903442442 -0.076796665
baby.sleep  0.61472303  1.00000000 -0.567802669  0.058309485
dan.grump  -0.90344244 -0.56780267  1.000000000  0.005833399
day        -0.07679667  0.05830949  0.005833399  1.000000000
```

Similar, but not quite the same. It's also worth noting that the `correlate()` function (in the `lsr` package) automatically uses the "pairwise complete" method:

```
> correlate(parenthood2)

CORRELATIONS
============
- correlation type:  pearson
- correlations shown only when both variables are numeric

           dan.sleep baby.sleep dan.grump    day
dan.sleep          .      0.615    -0.903 -0.077
baby.sleep     0.615          .    -0.568  0.058
dan.grump     -0.903     -0.568         .  0.006
day           -0.077      0.058     0.006      .
```

The two approaches have different strengths and weaknesses. The "pairwise complete" approach has the advantage that it keeps more observations, so you're making use of more of your data and (as we'll discuss in tedious detail in Chapter 10 and it improves the reliability of your estimated correlation. On the other hand, it means that every correlation in your correlation matrix is being computed from a slightly different set of observations, which can be awkward when you want to compare the different correlations that you've got.

So which method should you use? It depends a lot on *why* you think your values are missing, and probably depends a little on how paranoid you are. For instance, if you think that the missing values were "chosen" completely randomly[85] then you'll probably want to use the pairwise method. If you think that missing data are a cue to thinking that the whole observation might be rubbish (e.g., someone just selecting arbitrary responses in your questionnaire), but that there's no pattern to which observations are "rubbish" then it's probably safer to keep only those observations that are complete. If you think there's something systematic going on, in that some observations are more likely to be missing than others, then you have a much trickier problem to solve, and one that is beyond the scope of this book.

This page titled 5.8: Handling Missing Values is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by Danielle Navarro via source content that was edited to the style and standards of the LibreTexts platform.