Loading Data into R: Keeping Things Clean

Doug Koerber

10/26/2017

```r
suppress <- function(x) {
  suppressMessages(suppressWarnings(x))
}
suppress(library(readr))
```

Tip:

In the code chunk above, you can see the `suppress` function that I defined. What this function does is suppress the messages and warnings of any function that it is called upon. I did this to prevent `readr`'s output from taking up excessive space in this post.

Introduction

In class, we have learned that there exists multiple different ways to import and "clean" your data. We were introduced to two basic methods of importing data: using base R functions and using Hadley Wickham's `readr` package. In this post I am going to review the benefits of working with Hadley's packages and explore what the arguments in common functions do. The dataset we will be using is quite large (when compared to the NBA data we've used so far). In the code chunks below, we will run through examples that show how long it takes to import the data using different methods of importing, how large the dataset is, and various methods for importing subsets of the csv file.

Importing Data into R

Base R: Time To Import

```r
time_base <- suppress(system.time(loans <- read.csv('data/LoanStats3b.csv', header = TRUE)))
```

`readr`: Time To Import

```r
time_readr <- suppress(system.time(loans <- read_csv('data/LoanStats3b.csv', col_names = TRUE)))
```

Before I take a look at how long it took to import the data, first we must get a feel for the size of our data.

```r
variables <- ncol(loans)
obs <- nrow(loans)
```

By running the simple code above, we find that the number of variables in our dataframe is 137 and the number of observations is 188181. Furthermore, the csv file takes up about 157 MB on my hard drive.

Comparison of base R and `readr`

Now, having imported the data in 2 different ways, we can compare the amount of time it took to run the code. I assigned the output of the `system.time` function to save the results from our "experiment." Using the base R function `read.csv`, we can see that the time it took to import the data is 24.258 seconds, whereas `readr`'s `read_csv` function took only 3.167 seconds. In a separate test on my computer, I opened the file using Excel, which took just over a minute to load. As this simple example has shwon, R is clearly faster than Excel, and within R, `readr` is faster than base R.

The Arguments of `read_csv`

The `read_csv` function has many useful arguments, the most applicable of which will be covered in this post. These arguments were not covered in class, but do introduce new topics and methods for importing data that are useful in many scenarios.

Argument: `col_names`

In first twocode chunks that import data above, you can see that the `header` argument and the `col_names` argument do the same thing - they treat the first row of the data file as the names for variables. In base R, the default value for `header` is `FALSE`, whereas the default value for `col_names` is `TRUE` (I only included it in the code above to show that the two arguments are equivalent).

Argument: `col_types`

As we learned in class, `readr` functions will guess the data type of each column in a data set, but it will *not* automatically convert strings to factors. We only briefly covered the `col_types` argument in class, but I will quickly review it. To import data using the `read_csv` function and specify your own data types, we must use the `col_types` argument. It would be incredibly cumbersome to type out the individual column specifications for all 137 variables, so insted the example below only specifies that the annual_inc column be imported as a character and lets `readr` guess the type for all other columns. To expand column specification, simply type the name of the variables you want to specify and set them equal to one of `readr`'s types.

```r
l_col_types <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, col_types = cols(annual_inc = col_chara
cter())))
```

Argument: `skip`

The `skip` argument tells R to skip certain rows in the data set. If skip is set to a single number `n`, it will not import the first `n` rows in the data. Think of this as if you're opening up the file and deleting the first row of the data set, whether that row contains the variable names or just variables. If `col_names` is set to `TRUE`, then it will treat the new first row (the old second row) as the column headers, and if it is set to `FALSE`, then the data will be imported without column headers.

```
l_skip <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, skip = 1))
```

Argument: `n_max`

The `n_max` argument lets you control how many rows of data are imported. Setting `n_max` equal to some number `n` will keep only the first `n` rows of data. If the `col_names` argument is set to `TRUE`, then `readr` will import the header *and* the first `n` rows of data. If `col_names` is set to `FALSE`, then `readr` will simply import the first `n` rows of data. In the example below, the first row of the data will be set as the column headers and the next 10 rows of data will be imported as observations.

```
l_n_max <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, n_max = 10))
```

Argument: `na`

Not all data sets will have NA values as R interprets them. Some data sets may use the value 0 to represent missing information, others might use N/A, and perhaps some data sets may use both. Thankfully, `readr` supplies us with the `na` argument. This argument will change all values in the data set that match a value of vector that is passed into it. In the example below, all instances of 12000 and 11500 are converted to `na` values.

```
l_na <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, na = c("12000", "11500")))
```

Argument: `cols_only`

Perhaps the most useful of all arguments when working with large data sets is the `cols_only` argument. This argument allows you to directly select which columns are imported, rather than having to use the method we were shown in class of specifying `col_skip` for variables that we don't want to include. In the example below, only the annual_inc and loan_amnt variables are imported. I also specify their column types using the compact string representation of data types as outlined in the help documentation. This function can be incredibly useful when working with even larger data sets because it allows you to pick and choose the variables you wish to analyze. For example, a user might want to run a regression using only 8 of the 137 variables in our data set. Rather than `col_skip`-ing 129 columns, they can just type in the variables and column types into the `cols_only` argument.

```
l_cols_only <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, cols_only(annual_inc = "c", loan_amnt =
"i")))
```

Example

The classic estimation of *population* average is calculated by simply taking the average of a *sample* from the population. However, it can also be assumed that an estimator of population average is one that takes the average of only the first 3 values from the sample. To illustrate this, we will calculate the average of annual_inc using the entire data set, and compare it to the average of only the first 3 observations. I will use the `n_max` and `cols_only` arguments in this example to show how easy it is to apply them, and how minimizing the amount of data you import can affect import time.

Import annual_inc for the first 3 observations and all observations

```
annual_inc_3 <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, n_max = 3, cols_only(annual_inc = "d")
))

annual_inc_all <- suppress(read_csv('data/LoanStats3b.csv', col_names = TRUE, cols_only(annual_inc = "d")))
```

Calculate the average annual_inc of the first 3 observations and all observations

```
avg_3 <- as.integer(mean(annual_inc_3$annual_inc))
avg_all <- as.integer(mean(annual_inc_all$annual_inc))
```

As calculated above, the average of annual_inc for all observations is 72233, while the average for only the first 3 observations is 66920. The difference between the two, 5313, is relatively substantial, thus proving that the estimated average using only the first 3 observations is not an accurate measure of sample average.

Take Home Message

Using `readr` can drastically improve data import times and cut down the number of variables or observations that you need to sort through. When we were first introduced to `readr`, I remember feeling uncomfortable using it and unsure about its usefulness. Having completed this post, I now feel much more familiar with using `readr` and the various arguments, and I hope you do too.

References

1. https://cran.r-project.org/web/packages/readr/readr.pdf
2. https://cmitsolutions.com/blog/10-tips-tricks-and-add-ons-to-supercharge-your-use-of-adobe-acrobat-and-adobe-reader/
3. https://www.rstudio.com/resources/cheatsheets/
4. https://www.lendingclub.com/info/download-data.action
5. http://readr.tidyverse.org/articles/readr.html
6. http://rprogramming.net/read-csv-in-r/
7. http://yetanothermathprogrammingconsultant.blogspot.com/2016/12/reading-csv-files-in-r-readcsv-vs.html