Stats Lab 4

Jon Ashbrock

February 7, 2018

1 Introduction

Today students will learn how about:

1. Install packages into **R**
2. The Lock5Data package and data frames
3. Filtering Data

2 Installing Packages

While **R** has many built in functions, not everything we need is pre-installed into **R**. Therefore we may wish to download additional packages consisting of functions and other data into our program. An example of a package we will need for today is called "Lock5Data". You can import this package by typing the following into the console: After typing this, you will be asked to

```
install.packages("Lock5Data")
```

select the "CRAN" location you wish to use. We can select the one in "USA TN", for instance. This only ever needs to be done once for each package. IF you are writing a script that will use the data from the package, you should include: Note that you can replace "Lock5Data" with whatever package you

```
require("Lock5Data")
```

want to include in your script.

3 Data Frames and the Lock5Data package

Perhaps the most important type of data structure in data science or any statistical analysis is the data frame. A data frame is nothing more than a matrix with column labels. However these column labels will tell us what the column data contains, which is immensely helpful for performing statistical analysis on sets of data. To create a data frame, we first need a number of lists with the same length. Then the following syntax creates the frame: Then the matrix which is created is the matrix whose columns are the lists

```
my_frame <- data.frame(list1, list2, ..., listn)
```

we provided. The names of the columns are the names of the lists.

The Lock5Data package in **R** includes many built in data sets packaged as data frames. One such data set is called "SandP500", we can load this dataset into memory by typing (the first line below). We can also see the column titles of the data frame by using the second line below

```
data(SandP500)
names(SandP500)
```

The last important thing is to learn how to extract specific rows and columns from a data frame. The data frame is treated just like a matrix, and the commands for accessing sub-matrices of a matrix exist in the **R** Cheat sheet handout on the course web page.

You can access elements of a data frame just as you would a matrix. However, because of the way data frames store data, we will need to convert the entry to the desired data type. For example, the SandP500 data set contains the columns "Date", "Open", "High", "Low", "Close", "Volume". At this point the programmer will need to be able to guess what type of variables are contained in each column. The data types will almost always be numeric or characters. One could guess that each of the columns in our data frame are numeric except for the date column. Thus if we want to access a date and access the opening value of this stick on that date we could use:

```
the_date <- as.character(SandP500[1,1])
the_value <-as.numeric(SandP500[2,1])
```

A last, important command is to extract an entire column from a data frame. If we want to extract the list of dates in SandP500, we could use

```
dates <- as.character(SandP500[ , 1])
```

The syntax $vec[, 1]$ returns all rows of column 1. We could perform a similar command to extract all columns in a particular row, for example.

4 Cleaning and Filtering Data

Finally, there are times when we have faulty data or we only want to use certain subsets of data. There is nothing new in this section, just examples putting things we have previously learned together. These values will show up in data frames as "NA". Luckily for us, the plot function in **R** automatically omits rows which contain a "NA" value. However, it is always important to consider when data is missing when doing our analysis.

Sometimes, we may wish to filter out datapoints. For instance, if we only wish to look at days when the price of the stock opened higher than the previous day, we want to filter our list to contain only the desired values, Perhaps the simplest way to do this is to use the concatenate command to keep track of the indices for the data points we want. For example, the following code will contain only the indices (assuming we have the SandP500 dataset loaded already).

```
openings <- as.numeric(SandP500[ , 2])
len<- length(openings)# Return the number of rows
indices <- c() # Will hold the indices of the desired entries
for (i in 1:(len-1)){
if (openings[i+1]>openings[i]){
indices <- c(indices, i)
}}
```

Finally, we can access exactly these rows by writing

```
desired_rows <- openings[indices]
```