Chapter 1: The Basics

Tyson S. Barrett

Summer 2017

Introduction

and

This class will use R and RStudio to show how R can make several aspects of your research simpler, more likely to be reproducible, and more replicable.

Data Types, Objects and More

In this chapter we will discuss:

- data types and objects,

These aspects of R, although maybe mundane, are important for: 1) Data manipulation, 2) Modeling, and 3) Output.

Data Types, Objects and More

In this chapter we will discuss:

- data types and objects,
- importing data, and

These aspects of R, although maybe mundane, are important for: 1) Data manipulation, 2) Modeling, and 3) Output.

Data Types, Objects and More

In this chapter we will discuss:

- data types and objects,
- importing data, and
- saving data.

These aspects of R, although maybe mundane, are important for: 1) Data manipulation, 2) Modeling, and 3) Output.

Objects

Physical Objects

For Example:

- A table is great to eat on, write on, and somewhat good at sitting on.
- But it is horrible at taking you from Los Angeles to Toronto.

Virtual Objects

Likewise, objects in `R` are useful for some things and not for others. Objects are how we interact with the data, analyze it, and output it.

We will discuss the most important objects for working with data:

- Vectors

Virtual Objects

Likewise, objects in `R` are useful for some things and not for others. Objects are how we interact with the data, analyze it, and output it.

We will discuss the most important objects for working with data:

- Vectors
- Data Frames

Virtual Objects

Likewise, objects in `R` are useful for some things and not for others. Objects are how we interact with the data, analyze it, and output it.

We will discuss the most important objects for working with data:

- Vectors
- Data Frames
- Lists

Data Types: Vectors

numeric

numeric is a vector with numbers.[1] It can be whole numbers (i.e. an integer) or any real number (i.e. double). Below is a double.

```
x <- c(10.1, 2.1, 4.6, 2.3, 8.9)
x
```

```
[1] 10.1  2.1  4.6  2.3  8.9
```

Here, x, is the so-called “pointer” of this vector. So by typing the name of the object, we can see the contents.

[1]the c() is a function that glues the numbers (or whatever else is in the parenthases) together

character

A `character` vector is essentially just letters or words.

```r
ch <- c("I think this is great.",
        "I would suggest you learn R.",
        "You seem quite smart.")
ch
```

```
[1] "I think this is great."       "I would suggest you lea
[3] "You seem quite smart."
```

factor

A `factor` is a special vector in `R` for categorical variables. It is actually stored as numbers but we can give it labels.[2]

```
race <- c(1, 3, 2, 1, 1, 2, 1, 3, 4, 2)
race <- factor(race,
               labels = c("white", "black",
                          "hispanic", "asian"))
race
```

```
 [1] white    hispanic black    white    white    black
 [8] hispanic asian    black
Levels: white black hispanic asian
```

[2]Note that before we used the `factor()` function, the race object was just `numeric`. Once we told it was a “factor” `R` treats it as categorical.

Data Types: Data Frames and Lists

Data Frames

The `data.frame` is the most important data type for most projects.[3]

```r
df <- data.frame("A" = c(1,2,1,4,3),
                 "B" = c(1.4,2.1,4.6,2.0,8.2),
                 "C" = c(0,0,1,1,1))
df
```

```
  A   B C
1 1 1.4 0
2 2 2.1 0
3 1 4.6 1
4 4 2.0 1
5 3 8.2 1
```

[3]We can do quite a bit with the `data.frame` that we called `df`. (Once again, we could have called it anything, although I recommend short names.)

Data Frames

If “A” and “C” are factors we can tell R by:

```
df$A <- factor(df$A, labels = c("level1", "level2",
                                "level3", "level4"))
df$C <- factor(df$C, labels = c("Male", "Female"))
```

In the above code, the $ reaches into df to grab a variable (i.e. column).

Data Frames

The following code does the exact same thing:[4]

```
df[["A"]] <- factor(df$A, labels = c("level1", "level2",
                                     "level3", "level4"))
df[["C"]] <- factor(df$C, labels = c("Male", "Female"))
```

and so is the following:

```
df[, "A"] <- factor(df$A, labels = c("level1", "level2",
                                     "level3", "level4"))
df[, "C"] <- factor(df$C, labels = c("Male", "Female"))
```

[4] There are actually very small differences but its really not important here.

Data Frames

On the previous slide:

- `df[["A"]]` grabs the `A` variable just like `df$A`. The last example shows that we can grab both columns and rows.

```r
df[1:3, "A"]
df[1:3, 1]
```

Data Frames

On the previous slide:

- `df[["A"]]` grabs the `A` variable just like `df$A`. The last example shows that we can grab both columns and rows.
- In `df[, "C"]` we have a spot just a head of the comma. It works like this: `df[rows, columns]`.

```r
df[1:3, "A"]
df[1:3, 1]
```

Data Frames

On the previous slide:

- `df[["A"]]` grabs the `A` variable just like `df$A`. The last example shows that we can grab both columns and rows.
- In `df[, "C"]` we have a spot just a head of the comma. It works like this: `df[rows, columns]`.

```
df[1:3, "A"]
df[1:3, 1]
```

- Both lines of the above code grabs rows 1 thorugh 3 and column “A”.

Data Frames

Finally, we can combine the c() function to grab different rows and columns. To grab rows 1 and 5 and columns “B” and “C” you can do the following:

```
df[c(1,5), c("B", "C")]
```

Some Functions for Data Frames

Get the names of the variables:

```
names(df)
```

```
[1] "A" "B" "C"
```

Know what type of variable it is:

```
class(df$A)
```

```
[1] "factor"
```

Some Functions for Data Frames

Get quick summary statistics for each variable:

```r
summary(df)
```

```
       A            B              C    
 level1:2   Min.   :1.40   Male  :2  
 level2:1   1st Qu.:2.00   Female:3  
 level3:1   Median :2.10             
 level4:1   Mean   :3.66             
            3rd Qu.:4.60             
            Max.   :8.20             
```

Some Functions for Data Frames

Get the first 10 columns of your data:

```
head(df, n=10)
```

```
       A   B      C
1 level1 1.4   Male
2 level2 2.1   Male
3 level1 4.6 Female
4 level4 2.0 Female
5 level3 8.2 Female
```

Importing Data

Import, Don't Input

Most of the time you'll want to import data into `R` rather than manually entering it line by line, variable by variable.

There are some built in ways to import many delimited[5] data types (e.g. comma delimited–also called a CSV, tab delimited, space delimited). Other **packages**[6] have been developed to help with this as well.

[5]The delimiter is what separates the pieces of data.

[6]A package is an extension to `R` that gives you more functions–abilities–to work with data. Anyone can write a package, although to get it on the Comprehensive R Archive Network (CRAN) it needs to be vetted to a large degree. In fact, after some practice, you could write a package to help you more easily do your work.

Important Note about Importing

When you import data into R, it does not do anything to the data file (unless you ask it to). So, you can play around with it in R, change its shape, subset it, and whatever else you'd like without destroying or even modifying the original data.

Note that the slides that discuss **saving data** show you how you can override (not recommended) or save additional data files.

Importing Data

The first, if it is an R data file in the form .rda or .RData simply use:

```
load("file.rda")
```

Note that you don't assign this to a name such as df. Instead, it loads whatever R objects were saved to it.

Delimited Files

Most delimited files are saved as .csv, .txt, or .dat. As long as you know the delimiter, this process is easy.

```r
## for csv
df <- read.table("file.csv", sep = ",", header=TRUE)
## for tab delimited
df <- read.table("file.txt", sep = "\t", header=TRUE)
## for space delimited
df <- read.table("file.txt", sep = " ", header=TRUE)
```

The argument sep tells the function what kind of delimiter the data has and header tells R if the first row contains the variable names.

Note that at the end of the lines you see that I left a **comment** using #. Anything after a # is not read by the computer; it's just for us humans.

Other Data Formats

Data from other statistical software such as SAS, SPSS, or Stata are also easy to get into R. We will use two powerful packages:

1. `haven`
2. `foreign`

To install, simply run:

```r
install.packages("packagename")
```

This only needs to be run once on a computer. Then, to use it in a single R session (i.e. from when you open R to when you close it) run:

```r
library(packagename)
```

Other Data Formats

Using these packages, I will show you simple ways to bring your data in from other formats.

```
library(haven)
## for Stata data
df <- read_dta("file.dta")
## for SPSS data
df <- read_spss("file.sav")
## for this type of SAS file
df <- read_sas("file.sas7bdat")
```

```
library(foreign)
## for export SAS files
df <- read.xport("file.xpt")
```

Data and Questions

If you have another type of data file to import, online helps found on sites like www.stackoverflow.com and www.r-bloggers.com often have the solution.

Saving Data

Saving Data

Finally, there are many ways to save data. Most of the `read...` functions have a corresponding `write...` function.

```
## to create a CSV data file
write.table(df, file="file.csv", sep = ",")
```

Saving Data

`R` automatically saves missing data as `NA` since that is what it is in `R`. But often when we write a CSV file, we might want it as blank or some other value. If that's the case, we can add another argument `na = " "` after the sep argument.

Help Menu in R

If you ever have questions about the specific arguments that a certain function has, you can simply run:

```
?functionname
```

So, if you were curious about the different arguments in `write.table` simply run: `?write.table`. In the pane with the files, plots, packages, etc. a document will show up to give you more informaton.

Conclusions

R is built for you

- `R` is designed to be flexible and do just about anything with data that you'll need to do as a researcher.
- With this chapter under your belt, you can now read basic `R` code, import and save your data.
- The next chapter will introduce the "tidyverse" of methods that can help you join, reshape, summarize, group, and much more.