Week 1 R Tutorial

Statistics and Statistical Programming
Northwestern University
MTS 525

Aaron Shaw

September 15, 2020

Contents

Screencast #1

Very brief introduction to R (and R Studio and R Markdown)

If this is your first foray into R (or just your latest attempt), welcome! I hope this introduction helps you get started.

This file accompanies a screencast. The idea is that you can watch the screencast with RStudio and the file open on your own machine. You can read the compiled file as an HTML or PDF document and interact with it more directly by opening the accompanying .Rmd (RMarkdown) file in R Studio.

We'll begin with a few basics. What is R? What is R Studio? What is the R Console? What is R Markdown? After that (in the second screencast), we'll move on to performing some basic operations that you'll need to understand to actually use R to perform statistical programming.

What is R?

R is a free software environment and programming language for statistical computing. At it's core, it's a very flexible system that you can use to conduct any kind of statistical computing you (or anyone else) can imagine. People may say "R" to refer to the language and the software environment interchangeably. You can read more about R on the R project home page.

What is R Studio?

R Studio is an "integrated development environment" (IDE) that you can use with R. In other words, it's an application built to make it relatively easy to conduct statistical analysis, manage datasets, generate plots, and generally interact with R in a whole variety of ways.

R Studio has a bunch of options that you can use to adjust the look, feel, and organization of the interface. You can find them under the 'Tools' menu and 'Global Options'.

R Studio also has a number of very, very helpful keyboard shortcuts. Personally, I love keyboard shortcuts and I find that they vastly improve my experience using R Studio. If you want to learn the keyboard shortcuts, print out a copy of a cheatsheet like this and make sure it's handy any time you even think about using R Studio. You'll improve quickly.

The most important keyboard shortcut when you're using R Studio is probably 'CTRL-Enter.' It lets you send a command from the scripting window to the R console.

What is the R Console?

If you're have opened RStudio, you should see a window labeled "Console." This window allows you to enter direct commands to R. You can type these commands after the little sideways angle-bracket symbol ('>') or send them to the console from a script or notebook file. I will demonstrate how I do both of these things in the screencast.

The important thing to know about the R Console is that when you type anything (a 'statement') after the sideways angle-bracket and press 'Enter' R will try to evaluate the statement and "do" whatever it says. If the console cannot evaluate the statement successfully, it will generate an error.

What is RMarkdown?

RMarkdown is a markup language that you can use to create documents that combine snippets of text and (evaluated) R code. This file was created as an RMarkdown document. You can use RMarkdown to compile documents in many formats, including HTML, PDF, or MS Word documents. For more details on using RMarkdown see http://rmarkdown.rstudio.com where you can also access a series of dedicated tutorials.

You can think of RMarkdown files as "notebooks" where you can write, execute, and compile a combination of text and R code that can then be "knitted" together. Indeed, this is exactly what RStudio does when you click the **Knit** button above an open RMarkdown (.Rmd) file. Clicking **Knit** (or executing the corresponding keyboard shortcut) will produce a document that includes both the text content as well as the output of any embedded R code "chunks" within the document. An embedded R code chunk looks like this:

```r
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

That chunk calls a built-in function 'summary' to provide information about a built-in dataset called 'cars'. R has many built-in functions and a few built-in datasets. We'll come back to them later. For now, the point is that you can see how RMarkdown integrates a snippet of code and the results of executing that code right into the flow of the text.

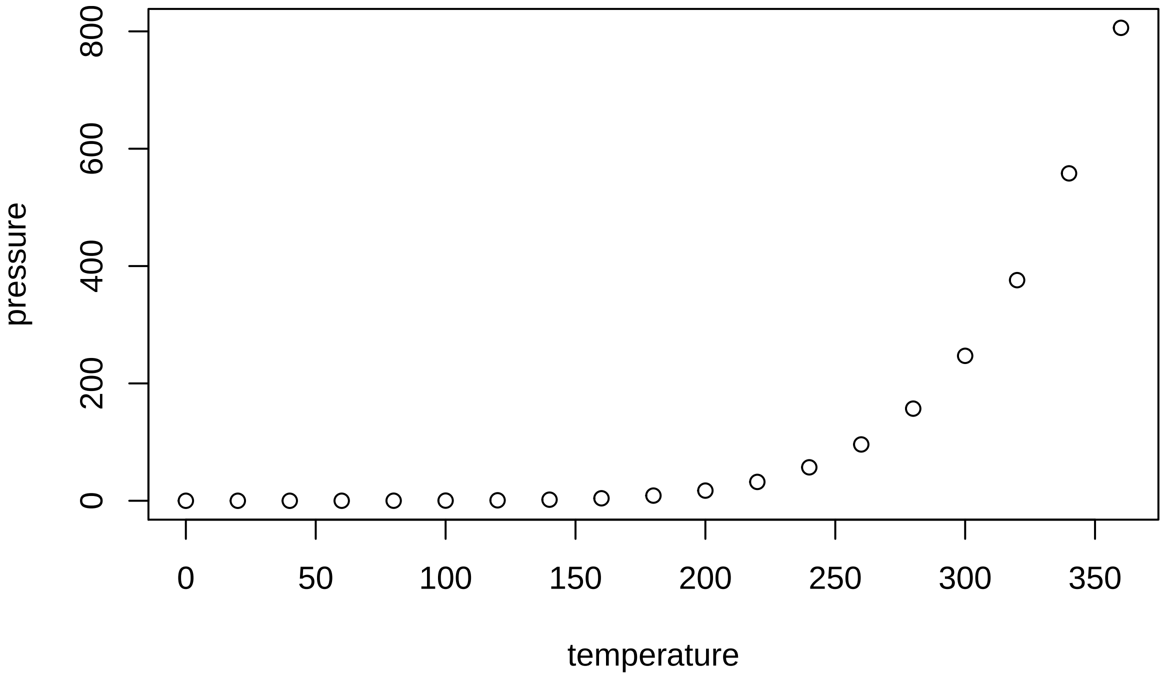
Embedding a new code chunk is easy. There is an 'Insert chunk' option in the 'Code' menu as well as an 'Insert' dropdown at the top of the .Rmd window. You can also use the keyboard shortcut (recommended! It's CTRL-ALT-i for Windows/Linux/Chrome users).

RMarkdown also lets you format text in a variety of ways including *italics* and **bold**.[1]

RMarkdown is not required for this course, but I strongly recommend that you invest the effort to learn to work in something like RMarkdown (other alternatives include Jupyter Notebooks or LaTeX) to complete your problem sets. The results will be clean, easy-to-read files that can integrate R code, analysis output, and graphics.

Including Plots in R Markdown

You can also embed plots, for example here's a plot of another one of R's built-in datasets containing data on the relation between temperature in degrees Celsius and vapor pressure of mercury in millimeters (of mercury).:



Note that the R code that generated the plot didn't show up this time. That's because I included an `echo = FALSE` parameter at the beginning of the code chunk (open up the .Rmd file to see what this looks like and try changing it the `echo = TRUE`). It is sometimes helpful to run code chunks without printing them.

Some other basics of R Studio

R Studio is being very actively developed and has many features that I don't know much/anything about. You can learn a **lot** more on the R Studio site and online (more on that in a moment). For now, I want to make sure you know how to do a few other things that will make it possible to complete your assignments for my class.

[1]It even does footnotes!

Setting preferences and options

The appearance and some features of R studio can be customized "globally" (across all projects) through the 'Global options' item in the 'Tools' menu. For example, I prefer a darker editor theme that feels more relaxing on my eyes.

Working with projects

An R Studio 'project' is a bundle of data, code, figures, output, and more that you want to keep bundled together. A project might contain multiple data files or notebooks. It also might contain other material such as a README file (documentation), supplementary materials, or even a finished paper. For the purposes of class, you may want to treat each problem set or assignment as an R Studio project.

R Studio projects are saved as '.Rproj' files accompanied by whatever else the project may entail. You can open them with R Studio and/or create new ones from the 'File' menu (select 'New Project'). Note that R studio can have multiple scripts and/or R Markdown files open, but seems to only be able to have one project open at a time.

Creating and saving a new R Markdown script

Creating new R Markdown scripts is also very straightforward in R Studio. From the 'File' menu, select 'New File' and 'R Markdown'. This will let you define some key attributes of the new file and automatically populate the .Rmd with some basic information.

Getting help

There are many, many ways to get help figuring out how to do things in R, R Studio, and R Markdown. I'll talk more about getting help with R functions in the second Screencast, but for now you should make sure you also have some idea of where/how to look things up when you have questions about R Studio or R Markdown. For example, try out the 'Help' menu items and identify some of the cheatsheets (like the one I mentioned earlier) that you think you might want to have around while you're learning to use these tools. The R Studio website links to several other resources and tutorials that you might find useful. StackOverflow also has extensive Q&A activity for questions about R, R Studio, Markdown, and related topics.

Screencast #2

Basics of R

This second screencast focuses on building basic skills with R. It can/will be far more interactive. The rest of the R Markdown script is intentionally short and is basically just an outline of the topics that will be covered. Please run these commands and experiment with R yourself in parallel as you watch/listen.

Using R as a calculator

R is a very fast calculator. You can enter simple arithmetic operations (addition, subtraction, multiplication, division, exponentiation) directly into the console or via your scripts, e.g.:

```
2 + 2
```

```
## [1] 4
```

```
6/3
```

```
## [1] 2
```

```
10^5
```

```
## [1] 1e+05
```

Try entering some others at the console yourself.

Variables

In R, you can use variables to do many things. The basic idea is that a variable allows you to 'assign' a value or set of values to a name. You indicate assignment by typing `<-` (keyboard shortcut: 'Alt–') or `=`. Here's an example:

```
x <- 2
x
```

```
## [1] 2
```

In the first line, I assigned a value of '2' to be called 'x'. In the second line, I just type 'x', which tells R to print the value for x. Surprise, surprise, it prints '2'. (More on why it also prints `[1]` in a moment…)

Try this out yourself at the R console. Then try assigning another value to 'x' and ask R to print x again.

For the most part, you can assign any value or set of values to any variable name and you can then use the variable name instead of the value(s):

```
cups.of.coffee <- 3
cups.of.coffee + 1
```

```
## [1] 4
```

```
cups.of.coffee*3
```

```
## [1] 9
```

Some variable names and words are 'special', however, in that R has pre-assigned values to them or pre-assigned functions. We will encounter many of these. For one example of a pre-assigned variable, try typing `pi` at the console and press 'Enter'.

One other special value a variable may take is `NA` (no quotes!) which means it is missing. If a value is missing, you may not be able to do mathematical operations with it:

```
cups.of.coffee <- NA
cups.of.coffee-1
```

```
## [1] NA
```

Types (also known as classes)

Every variable has a 'type' or 'class'. For example, we've already created a few variables which are 'numeric'. These can be whole integers or have decimals. If you ever want to know what a variable's type is, you can ask R to tell you using the `class()` function like this:

```r
class(x)
```

```
## [1] "numeric"
```

We'll come back to functions in a moment. In the meantime, other important types of variables are are 'characters' and 'logical':

```r
my.name <- "Aaron"
class(my.name)
```

```
## [1] "character"
```

```r
my.answer <- TRUE ## Note the capitalization!
class(my.answer)
```

```
## [1] "logical"
```

It is often important to know what class a variable is because R lets you perform some operations on certain kinds of variables, but not on others.

Functions

In R, you use functions to do just about everything (e.g., inquire about the class or type of a variable as we did above). Every function takes some input (called an argument) usually in parentheses and provides some output (sometimes called the return value). Some functions take multiple inputs and return multiple outputs. You can also write your own functions and edit existing functions. This is part of what makes R so powerful and flexible.

Arguably the most important function is `help()`. The help function will retrieve the documentation for any function. To learn more about help, try entering `help(help)` at the console.

Another useful function allows you to delete a variable: `rm()` or `remove()`. Try creating a variable and removing it.

There are many built in functions. Some are common mathematical operations like `sqrt()`, `log()`, or `log1p()`. Others help you manage your workspace like `ls()`.

Check your reference card for many, many more examples.

Vectors

You can think of a vector as a set of things that are all the same type. In R, all variables are vectors even though they may have just one thing in them! That's why the R Console prints out `[1]` next to the value of a variable with just one value:

```r
my.name
```

```
## [1] "Aaron"
```

You can make vectors with a special function `c()`:

```r
ages <- c(36, 50, 38)
ages
```

```
## [1] 36 50 38
```

Vectors can be of any type but they can have only one type:

```r
class(ages)
```

```
## [1] "numeric"
```

```r
painters <- c("frida", "diego", "daniel")
class(painters)
```

```
## [1] "character"
```

If you mix types vectors together, they will be "coerced" to a single type. The results be surprising (and sometimes annoying).

```r
class(c(ages, painters)) ## Notice that you can "nest" functions within each other.
```

```
## [1] "character"
```

Indexing

You can index the elements in a vector using square brackets and a number like this:

```r
painters[2]
```

```
## [1] "diego"
```

You can also use indexing to refer to multiple elements in a vector

```r
painters[2:3] ## A sequence of the second and third elements
```

```
## [1] "diego"  "daniel"
```

You can even assign new values to an item (or add items) in a vector using indexing:

```r
ages[2] <- 52
ages
```

```
## [1] 36 52 38
```

Recycling

Mathematical operations are "recycled" when applied to a vector. R just performs the same operation on each item in the vector and gives you the output:

```r
ages*2
```

```
## [1]  72 104  76
```

```r
ages/2
```

```
## [1] 18 26 19
```

Naming items

You can apply a name to any item in a vector

```r
names(ages)
```

```
## NULL
```

```r
names(ages) <- c("Wilma", "Fred", "Barney")
names(ages)
```

```
## [1] "Wilma"  "Fred"   "Barney"
```

Now you can index into 'ages' using the name of each item:

```r
ages["Barney"]
```

```
## Barney
##     38
```

Working with vectors with multiple elements

Some functions are very handy for working with vectors that have multiple elements:

```r
length(ages)
```

```
## [1] 3
```

```r
sum(ages)
```

```
## [1] 126
```

```r
mean(ages)
```

```
## [1] 42
```

```r
sd(ages) ## Standard deviation. More on that later.
```

```
## [1] 8.717798
```

```r
sort(ages)
```

```
##  Wilma Barney   Fred
##     36     38     52
```

```r
range(ages)
```

```
## [1] 36 52
```

```r
summary(ages)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      36      37      38      42      45      52
```

```r
table(ages)
```

```
## ages
## 36 38 52
##  1  1  1
```

You can also construct new vectors by performing logical comparisons on an existing vector:

```r
ages < 39
```

```
##  Wilma   Fred Barney
##   TRUE  FALSE   TRUE
```

```r
ages != 38
```

```
##  Wilma   Fred Barney
##   TRUE   TRUE  FALSE
```

```
painters == "Diego"
```

```
## [1] FALSE FALSE FALSE
```

```
painters == "diego"
```

```
## [1] FALSE  TRUE FALSE
```

```
painters != "frida"
```

```
## [1] FALSE  TRUE  TRUE
```

This is very useful for indexing and recoding a variable. In this case I'll use the built-in variable 'rivers' which is the lengths in miles of 141 major North American rivers (type `help(rivers)` to learn more) :

```
rivers
```

```
##   [1]  735  320  325  392  524  450 1459  135  465  600  330  336  280  315  870
##  [16]  906  202  329  290 1000  600  505 1450  840 1243  890  350  407  286  280
##  [31]  525  720  390  250  327  230  265  850  210  630  260  230  360  730  600
##  [46]  306  390  420  291  710  340  217  281  352  259  250  470  680  570  350
##  [61]  300  560  900  625  332 2348 1171 3710 2315 2533  780  280  410  460  260
##  [76]  255  431  350  760  618  338  981 1306  500  696  605  250  411 1054  735
##  [91]  233  435  490  310  460  383  375 1270  545  445 1885  380  300  380  377
## [106]  425  276  210  800  420  350  360  538 1100 1205  314  237  610  360  540
## [121] 1038  424  310  300  444  301  268  620  215  652  900  525  246  360  529
## [136]  500  720  270  430  671 1770
```

```
head(rivers) ## 'head()' shows you the first five values of a vector
```

```
## [1] 735 320 325 392 524 450
```

```
rivers < 300 ## Recycles the comparison and returns TRUE or FALSE for each river
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
##  [13]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE
##  [37]  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49]  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [73] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [121] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE
## [133]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
rivers[rivers < 300] ## A subset of the data
```

```
##  [1] 135 280 202 290 286 280 250 230 265 210 260 230 291 217 281 259 250 280 260
## [20] 255 250 233 276 210 237 268 215 246 270
```

```
little.rivers <- rivers[rivers < 300]
big.rivers <- rivers; big.rivers[big.rivers < 300] <- NA ## Two commands, one line. Recodes the short r
```

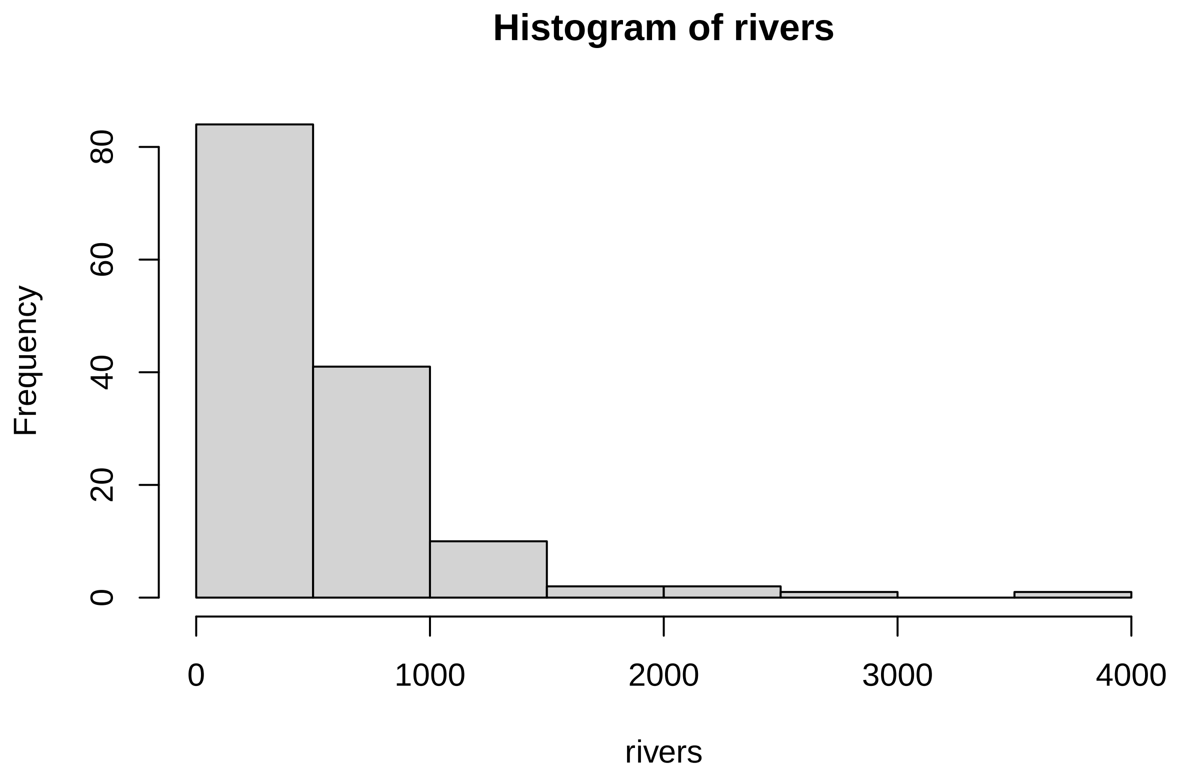
Basic plotting and visualizations

Visualizations can help you explore data and interpret results. Use them often!

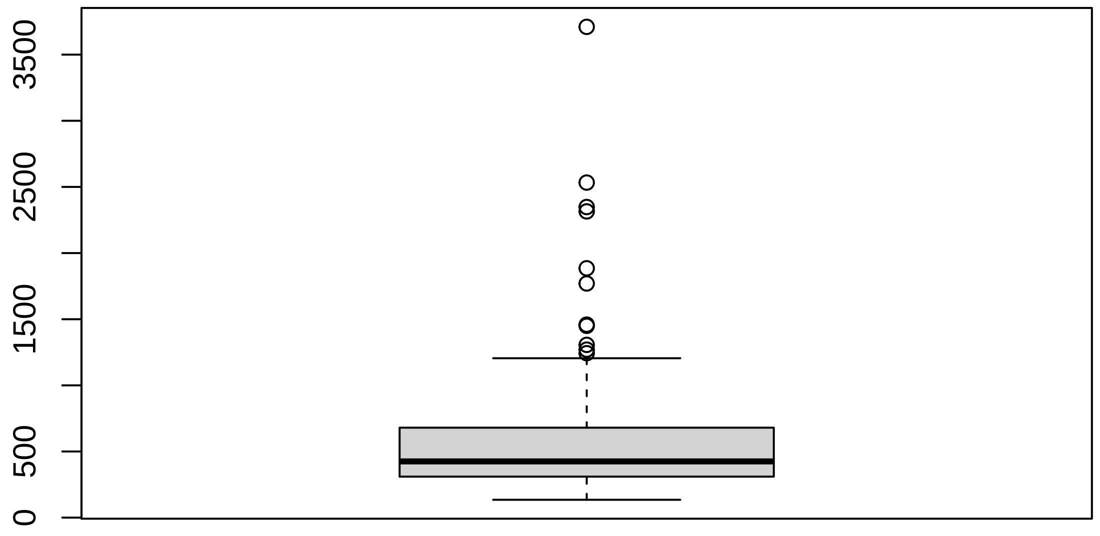
```
table(rivers>300)
```

```
##
## FALSE  TRUE
##    32   109
```

```
hist(rivers)
```



```
boxplot(rivers)
```



Packages

By default, R has many built-in functions and example datasets. However, many people have extended R by creating additional functions. Often these additional functions are collected together and distributed as "packages" or "libraries" that may also include additional datasets. Rstudio gives you a couple of ways to work with these. The traditional method is via the following commands

```r
install.packages("UsingR") ## note the quotation marks. This package accompanies the Verzani book.
install.packages("openintro") ## This package goes along with our textbook.

## Then you can load the package this way:
library(UsingR) ##  No quotes!
library(openintro)
```

Run these commands on your system. Use the 'Packages' tab to explore the documentation of the functions and datasets available through the `openintro` package.

Note that I have told R not to evaluate this last chunk of code because it generates a bunch of output and you only need to install any given package once. To skip evaluating a code chunk you can include the `eval=FALSE` in the header to that chunk in your R Markdown file. This tells R not to execute the code when it knits the file. Take a look at the .Rmd file to learn more.

Loading datasets

Often datasets will be located online or locally on your computer and you'll want to load them directly. For '.Rdata' files you can do this using the `load()` command. For others you may want to use commands like `read.csv()`, `read.table()`, or `read.foreign()` (that last one requires the 'foreign' package, so you'll need to load it first). RStudio also has a drop-down menu item ('File' → 'Import dataset') that can help you load a local file.

Environment and History

By default, R Studio allows you to see all the variables or 'objects' currently available to you in a particular session. Find the window/tab called "Environment" and take a look at what's there.

There's another tab (likely in the same window) called "History" that contains all the commands you have run in the current session. This can be super helpful when you're trying to piece together what you did a few moments ago or why that command you just ran worked and the one you tried a before did not.

Getting help

As mentioned earlier, the `help()` command is your friend. RStudio also has a 'Help' tab in one of the default windows. You can also use the RStudio cheatsheets, StackOverflow, the Verzani textbook, the Quick-R tutorials, and/or many, many other resources on the internet, including the rseek search engine (which just searches the web for R-related resources).