Introduction to R and RStudio

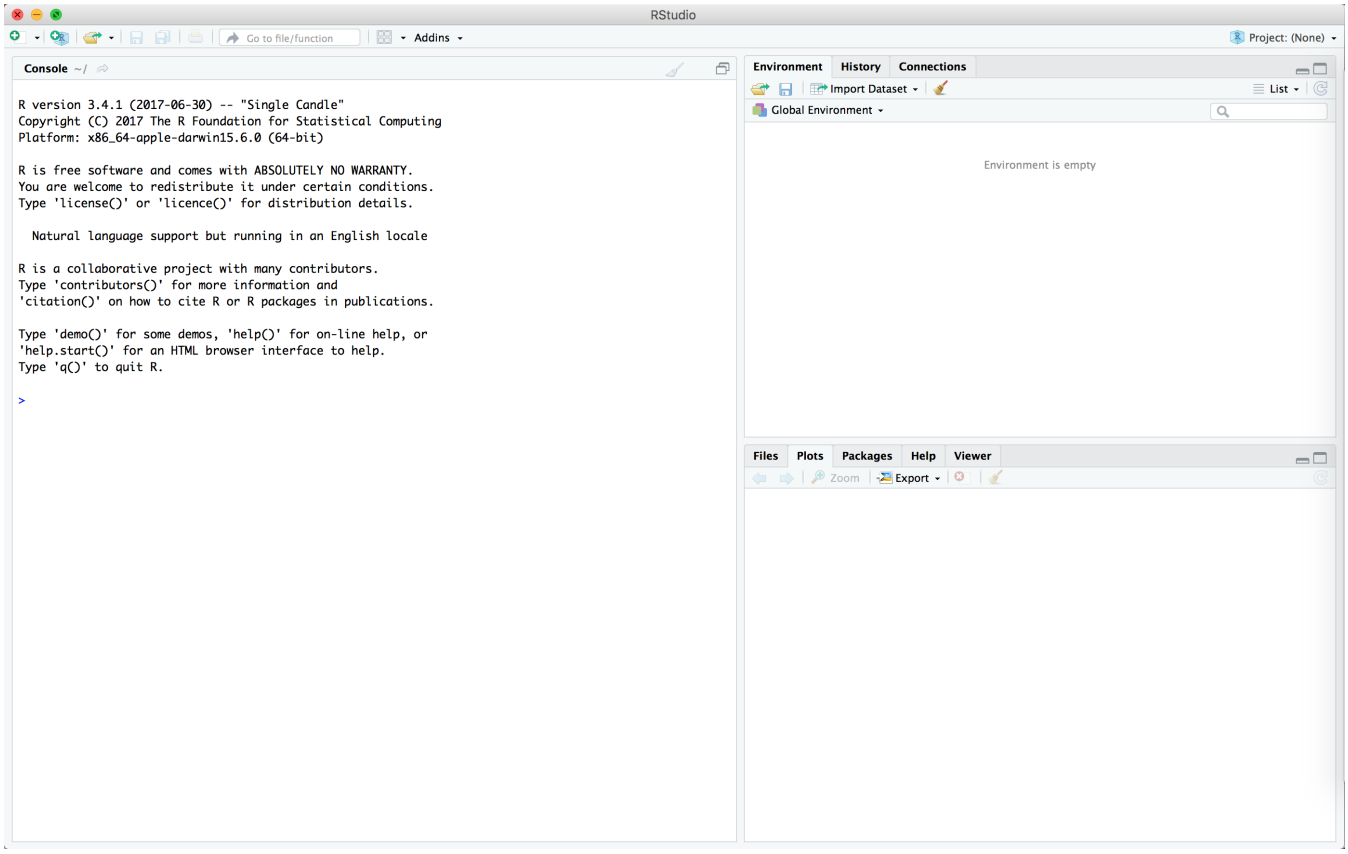
The RStudio Interface

The goal of this lab is to introduce you to R and RStudio, which you'll be using throughout the course both to learn the statistical concepts discussed in the course and to analyze real data and come to informed conclusions. To clarify which is which: R is the name of the programming language itself and RStudio is a convenient interface.

As the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. Before we get to that stage, however, you need to build some basic fluency in R. Today we begin with the fundamental building blocks of R and RStudio: the interface, reading in data, and basic commands.

To get started, you'll need to install R (https://cran.r-project.org/) and RStudio (https://rstudio.com/products/rstudio/download/#download). These should both be relatively painless whether you're using Windows, OSX, or a Unix-based operating system. If you're having trouble, please let me or your TA know and we will be very happy to help you install them.

When you've done that, go ahead and launch RStudio. You should see a window that looks something like this:



The panel on the lower left is where the action happens. It's called the *console*. Everytime you launch RStudio, it will have the same text at the top of the console telling you the version of R that you're running. Below that information is the *prompt*. As its name suggests, this prompt is really a request: a request for a command. Initially, interacting with R is all about typing commands and interpreting the output. These commands and their syntax have evolved over decades (literally) and now provide what many users feel is a fairly natural way to access data and organize, describe, and invoke statistical computations.

The panel in the upper right contains your *workspace* as well as a history of the commands that you've previously entered. Any plots that you generate will show up in the panel in the lower right corner. This is also where you can browse your files, access help, manage packages, etc.

R Packages

R is an open-source programming language, meaning that users can contribute packages that make our lives easier, and we can use them for free. For this lab, and many others in the future, we will use the following R package:

- *tidyverse*: an ecosystem of packages designed for working with and presenting data

If packages are not already available in your R environment, you can install them by typing the following line of code into the console of your RStudio session, pressing the enter/return key. Note that you can check to see which packages (and which versions) are installed by inspecting the *Packages* tab in the lower right panel of RStudio.

For this and other labs, *tidyverse* will already be pre-installed. But nothing bad will happen if you install it again.

```
install.packages("tidyverse")
```

In order to use the tools in a package, you need to load it into your working environment. We do this with the `library` function. Run the following lines of code in your console.

```
library(tidyverse)
```

You only need to *install* packages once, but you need to *load* them each time you relaunch RStudio.

You can find out more about the packages that make up the `tidyverse` at http://tidyverse.org/ (http://tidyverse.org/).

Creating a reproducible lab report

We will be using R Markdown to create reproducible lab reports. See the following videos describing why:

Why use R Markdown for Lab Reports? (https://youtu.be/lNWVQ2oxNho)

We will be using a markdown language, R Markdown, to type up the lab report. This allows you to complete your lab entirely in RStudio as well as ensuring reproducibility of your analysis and results. For every lab, we'll provide a template for you that you can find in the project. Use the following code to download this template:

```
download.file("https://dyurovsky.github.io/85309/post/rmd/lab1.Rmd",
              destfile = "lab1.Rmd")
```

You will see a new file called `lab1.Rmd` in the Files tab on the pane in the bottom right corner of your RStudio window. We will refer to this as your "R markdown file" or "your report." Click on the file name to open the file. All you need to do to complete the lab is to type up your **brief** answers and the `R` code (when necessary) in the spaces provided in the document.

Before you keep going, type your name in the "author:" field, and the date in the "date" field at the top. Then click on *Knit HTML* and you'll see your document in a new pop-up window.

Going forward you should refrain from typing your code directly in the console, and instead type any code (final correct answer, or anything you're just trying out) in the R Markdown file and run the chunk using either the Run button on the chunk(green sideways triangle) or by highlighting the code and clicking Run on the top right corner of the R Markdown editor. If at any point you need to start over, you can Run All Chunks above the chunk you're working in by clicking on the down arrow in the code chunk. *Note:* Work you do in the console won't transfer over to your R Markdown file.

Dr. Arbuthnot's Baptism Records

To get you started, run the following command to load the data.

```
arbuthnot <- read_csv("https://dyurovsky.github.io/85309/data/lab1/arbuthnot.csv")
```

You can do this by

- clicking on the green arrow at the top right of the code chunk in the R Markdown (Rmd) file, or
- putting your cursor on this line, and hit the *Run* button on the upper right corner of the pane, or
- hitting `Ctrl-Shift-Enter` , or
- typing the code in the console.

This command instructs R to load some data: the Arbuthnot baptism counts for boys and girls. You should see that the workspace area in the upper righthand corner of the RStudio window now lists a data set called `arbuthnot` that has 82 observations on 3 variables. As you interact with R, you will create a series of objects. Sometimes you load them as we have done here, and sometimes you create them yourself as the byproduct of a computation or some analysis you have performed.

The Arbuthnot data set refers to Dr. John Arbuthnot, an 18^{th} century physician, writer, and mathematician. He was interested in the ratio of newborn boys to newborn girls, so he gathered the baptism records for children born in London for every year from 1629 to 1710. We can view the data by typing its name into the console.

```
arbuthnot
```

However printing the whole dataset in the console is not that useful. One advantage of RStudio is that it comes with a built-in data viewer. Click on the name `arbuthnot` in the *Environment* pane (upper right window) that lists the objects in your workspace. This will bring up an alternative display of the data set in the *Data Viewer* (upper left window). You can close the data viewer by clicking on the `x` in the upper lefthand corner.

What you should see are four columns of numbers, each row representing a different year: the first entry in each row is simply the row number (an index we can use to access the data from individual years if we want), the second is the year, and the third and fourth are the numbers of boys and girls baptized that year, respectively. Use the scrollbar on the right side of the console window to examine the complete data set.

Note that the row numbers in the first column are not part of Arbuthnot's data. R adds them as part of its printout to help you make visual comparisons. You can think of them as the index that you see on the left side of a spreadsheet. In fact, the comparison to a spreadsheet will generally be helpful. R has stored Arbuthnot's data in a kind of spreadsheet or table called a *tibble*.

You can see the dimensions of this tibble as well as the names of the variables and the first few observations by typing:

```
glimpse(arbuthnot)
```

This command should output the following

```
## Rows: 82
## Columns: 3
## $ year  <dbl> 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639…
## $ boys  <dbl> 5218, 4858, 4422, 4994, 5158, 5035, 5106, 4917, 4703, 5359, 5366…
## $ girls <dbl> 4683, 4457, 4102, 4590, 4839, 4820, 4928, 4605, 4457, 4952, 4784…
```

We can see that there are 82 observations and 3 variables in this dataset. The variable names are `year`, `boys`, and `girls`. At this point, you might notice that many of the commands in R look a lot like functions from math class; that is, invoking R commands means supplying a function with some number of arguments. The `glimpse` command, for example, took a single argument, the name of a tibble.

Some Exploration

Let's start to examine the data a little more closely. We can access the data in a single column of a tibble separately using a command like

```
arbuthnot$boys
```

This command will only show the number of boys baptized each year. The dollar sign basically says "go to the tibble that comes before me, and find the variable that comes after me".

Exercise 1 What command would you use to extract just the counts of girls baptized? Try it! (Enter your answer in your R markdown document and run the entire report by hitting **Knit HTML**. Voila! The R output you need is already in your report.)

Notice that the way R has printed these data is different. When we looked at the complete tibble, we saw 82 rows, one on each line of the display. These data are no longer structured in a table with other variables, so they are displayed one right after another. Objects that print out in this way are called *vectors*; they represent a set of numbers. R has added numbers in [brackets] along the left side of the printout to indicate locations within the vector. For example, 5218 follows [1], indicating that 5218 is the first entry in the vector. And if [43] starts a line, then that would mean the first number on that line would represent the 43rd entry in the vector.

Data visualization

R has some powerful functions for making graphics. We can create a simple plot of the number of girls baptized per year with the command

```
ggplot(arbuthnot, aes(x = year, y = girls)) +
  geom_point()
```

The `ggplot()` function (meaning "grammar of graphics plot") is a function from the `ggplot2` package that lets you build plots by specifying them in a grammar of graphics. Here we made a scatterplot that should appear under the *Plots* tab of the lower right panel of RStudio.

Notice that the `ggplot` command above again looks like a function, this time two arguments separated by commas. The first argument is the name of the tibble that contains the data we want to plot, and the second argument is a list of aesthetics. This list says I want to plot a column of the tibble called `year` on the x-axis, and a column of the tibble called `girls` on the y-axis. I add to this a `geom_point()` which says that I want to represent the data geometrically as a set of points.

If we wanted to connect the data points with lines, we can add another `geom` to the plot:

```
ggplot(arbuthnot, aes(x = year, y = girls)) +
  geom_point() +
  geom_line()
```

You might wonder how you are supposed to know about all of these arguments and geoms. Thankfully, R documents all of its functions extensively. To read what a function does and learn the arguments that are available to you, just type in a question mark followed by the name of the function that you're interested in. Try the following.

```
?ggplot
```

Notice that the help file replaces the plot in the lower right panel. You can toggle between plots and help files using the tabs at the top of that panel.

Exercise 2 Is there an apparent trend in the number of girls baptized over the years? How would you describe it? (To ensure that your lab report is comprehensive, be sure to include the code needed to make the plot as well as your written interpretation.)

R as a big calculator

Now, suppose we want to plot the total number of baptisms. To compute this, we could use the fact that R is really just a big calculator. We can type in mathematical expressions like

```
5218 + 4683
```

to see the total number of baptisms in 1629. We could repeat this once for each year, but there is a faster way. If we add the vector for baptisms for boys to that of girls, R will compute all sums simultaneously.

```
arbuthnot$boys + arbuthnot$girls
```

What you will see are 82 numbers (in that packed display, because we aren't looking at a tibble here), each one representing the sum we're after. Take a look at a few of them and verify that they are right.

Adding a new variable to the tibble

We'll be using this new vector to generate some plots, so we'll want to save it as a permanent column in our tibble.

```
arbuthnot <- arbuthnot %>%
  mutate(total = boys + girls)
```

The `%>%` operator is called the **piping** operator. It takes the output of the previous expression and pipes it into the first argument of the function in the following one. If you're inclined to think about mathematical functions, `x %>% f %>% g` is equivalent to `g(f(x))`

> **A note on piping:** Note that we can read these three lines of code as the following:
>
> *"Take the `arbuthnot` dataset and **pipe** it into the `mutate` function. Mutate the `arbuthnot` data set by creating a new variable called `total` that is the sum of the variables called `boys` and `girls`. Then assign the resulting dataset to the object called `arbuthnot`, i.e. overwrite the old `arbuthnot` dataset with the new one containing the new variable."*
>
> This is equivalent to going through each row and adding up the `boys` and `girls` counts for that year and recording that value in a new column called `total`.

> **Where is the new variable?** When you make changes to variables in your dataset, click on the name of the dataset again to update it in the data viewer.

You'll see that there is now a new column called `total` that has been tacked on to the tibble. The special symbol `<-` performs an *assignment*, taking the output of one line of code and saving it into an object in your workspace. In this case, you already have an object called `arbuthnot`, so this command updates that data set with the new mutated column.

We can make a plot of the total number of baptisms per year with the command

```
ggplot(arbuthnot, aes(x = year, y = total)) +
  geom_line()
```

Similarly to how we computed the total number of births, we can compute the ratio of the number of boys to the number of girls baptized in 1629 with

```
5218 / 4683
```

or we can act on the complete columns with the expression

```
arbuthnot <- arbuthnot %>%
  mutate(boy_to_girl_ratio = boys / girls)
```

We can also compute the proportion of newborns that are boys in 1629

```
5218 / (5218 + 4683)
```

or this may also be computed for all years simultaneously and append it to the dataset:

```
arbuthnot <- arbuthnot %>%
  mutate(boy_ratio = boys / total)
```

Note that we are using the new `total` variable we created earlier in our calculations.

Exercise 3 Now, generate a plot of the proportion of boys born over time. What do you see?

Tip: If you use the up and down arrow keys, you can scroll through your previous commands, your so-called command history. You can also access it by clicking on the history tab in the upper right panel. This will save you a lot of typing in the future.

Finally, in addition to simple mathematical operators like subtraction and division, you can ask R to make comparisons like greater than, `>`, less than, `<`, and equality, `==`. For example, we can ask if boys outnumber girls in each year with the expression

```
arbuthnot <- arbuthnot %>%
  mutate(more_boys = boys > girls)
```

This command add a new variable to the `arbuthnot` dataframe containing the values of either `TRUE` if that year had more boys than girls, or `FALSE` if that year did not (the answer may surprise you). This variable contains a different kind of data than we have encountered so far. All other columns in the `arbuthnot` data frame have values that are numerical (the year, the number of boys and girls). Here, we've asked R to create *logical* data, data where the values are either `TRUE` or `FALSE`. In general, data analysis will involve many different kinds of data types, and one reason for using R is that it is able to represent and compute with many of them.

More Practice

In the previous few pages, you recreated some of the displays and preliminary analysis of Arbuthnot's baptism data. Your assignment involves repeating these steps, but for present day birth records in the United States. Load the present day data with the following command.

```
present <- read_csv("https://dyurovsky.github.io/85309/data/lab1/present.csv")
```

The data are stored in a tibble called `present`.

Exercise 4 What years are included in this data set? What are the dimensions of the tibble? What are the variable (column) names?

Exercise 5 How do these counts compare to Arbuthnot's? Are they of a similar magnitude?

Exercise 6 Make a plot that displays the proportion of boys born over time. What do you see? Does Arbuthnot's observation about boys being born in greater proportion than girls hold up in the U.S.? Include the plot in your response. *Hint:* You should be able to reuse your code from Ex 3 above, just replace the tibble name.

Exercise 7 In what year did we see the most total number of births in the U.S.? *Hint:* First calculate the totals and save it as a new variable. Then, sort your dataset in descending order based on the total column. You can do this interactively in the data viewer by clicking on the arrows next to the variable names. To include the sorted result in your report you will need to use two new functions: `arrange` (for sorting). We can arrange the data in a descending order with another function: `desc` (for descending order). Sample code provided below.

```
present %>%
  arrange(desc(total))
```

These data come from reports by the Centers for Disease Control listed in the references section. If you would like to read more about an analysis of sex ratios at birth in the United states, check out this page (http://www.cdc.gov/nchs/data_access/vitalstatsonline.htm).

Resources for R and RStudio

That was a short introduction to R and RStudio, but we will provide you with more functions and a more complete sense of the language as the course progresses.

In this course we will be using R packages called `dplyr` for data wrangling and `ggplot2` for data visualization–both part of the `tidyverse`. If you are googling for R code, make sure to also include these package names in your search query. For example, instead of googling "scatterplot in R", google "scatterplot in R with ggplot2".

These cheatsheets may come in handy throughout the semester:

- RStudio cheatsheet (https://github.com/rstudio/cheatsheets/raw/main/rstudio-ide.pdf)
- RMarkdown cheatsheet (https://github.com/rstudio/cheatsheets/raw/main/rmarkdown-2.0.pdf)
- Data transformation cheatsheet (https://github.com/rstudio/cheatsheets/raw/main/data-transformation.pdf)
- Data visualization cheatsheet (https://github.com/rstudio/cheatsheets/raw/main/data-visualization-2.1.pdf)

The book titled "Getting used to R, RStudio, and R Markdown" by Chester Ismay, which can be freely accessed here (https://ismayc.github.io/rbasics-book), is also a wonderful resource for for new users of R, RStudio, and R Markdown. It includes examples showing working with R Markdown files in RStudio recorded as GIFs.

Note that some of the code on these cheatsheets may be too advanced for this course, however majority of it will become useful throughout the semester.