Introduction to RStudio

Part One: Getting familiar with RStudio

In the class directory, double-click on the `.Rproj` file and it will open RStudio.

RStudio is your dashboard for writing R scripts. Script files will show up in the upper-left quadrant of RStudio.

The upper-right quadrant has several tabs, the first of which is **environment**: if you store any information in variables (such as a table of data), they will show up there.

The lower-right quadrant also has several tabs, the first of which is Files: it shows what files exist in your current working directory, which is the class folder which has the `.Rproj` file we clicked on. Go ahead and click once on the `scripts` folder and then the `first-r-notebook.Rmd` file, which is an R Notebook file. It will appear in the upper-left quadrant.

In the lower-left, you'll see the **Console**. This will reflect all the R code you run, but we don't do much with it when we're working in R Notebook files.

We encourage the use of **Notebook** files (.Rmd), which enable you to both write comments (like what I'm writing here) and to write code, in what we call a **code chunk**:

```
x <- "Hello, World!"
print(x)
```

```
## [1] "Hello, World!"
```

In notebook files, the results of the code in each code chunk will print below the chunk, allowing you to quickly see the results of your work. The utility of this will be apparent when we start asking questions of data.

You create a code chunk by typing `ctrl+alt+i` (`cmd+option+i` on a Mac) and you write your `code` in between the first and last lines of the code chunk. All your comments, questions, thoughts, and notes are written outside of the code chunks. The comments outside of code chunks use markdown for formatting. Markdown is similar to html, but if you don't know it, don't worry about it for now.

Part Two: Terminology

Some important terminology has come up already:

- **Notebook**: a certain type of R script that allows us to write whatever text we want, including `code` inside of code chunks
- **code chunk**: created by typing `ctrl+alt+i` (`cmd+option+i` on a Mac), type your code inside code chunks.
- **environment**: this is basically your workspace for every R session. It's empty until you start storing information in variables.
- **variable**: a variable is a container that holds something. For example, the following code stores the word "spaghetti" into a variable named "x", using the assignment operator `<-` , and then prints the contents of that variable to the console below the code chunk:

```
x <- "spaghetti"
print(x)
```

```
## [1] "spaghetti"
```

In the example above, “spaghetti” is a string of text, or characters. You can also store numbers in variables (which do not require double quotes):

```
y <- 3
```

Naming variables: you can use letters, numbers, underscores and dots

Variable Name	Validity	Reason
var_name2.	valid	has letters, numbers, underscore, dot
.var_name	valid	can start with a dot (but not a dot followed by a number)
var_name%	invalid	has the character ‘%’ (not allowed)
2var_name	invalid	starts with a number
.2var_name	invalid	cannot start with a dot and a number
_var_name	invalid	cannot start with underscore

- **data types**:
    - character: commonly referred to as a string; can be letters, numbers, punctuation, etc. Always enclosed in “double quotes”.
    - integer: a whole number, such as `1`, `5`, `10000`. No decimal places.
    - numeric: a number that can have decimal places, such as `5.2` or `100.37`
    - logical: either `TRUE` or `FALSE` (not quoted)
    - complex: e.g. `3 + 2i` (probably won’t use this)
    - raw: (won’t use this either)
- **vector**: this is a common feature of R that you will use regularly. A vector is a series of values. Vectors can only store elements of the same data type, for example all strings or all numbers, and are created with the `c()` **function**:

```
x <- c("spagetti sauce", "noodles", "parmesan")
y <- c(1,2,3)
```

- **function**: these are the backbones of all analysis, in any program (from spreadsheets to database managers to programming languages). Every function has a particular structure and does a particular thing. The structure is: `function_name(arguments)`. For example, the `sum()` function works in R the same way it works in other programs:

```
x <- 1
y <- 2
sum(x,y)
```

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

- **package**: a set of features and functions that are not a part of base R (what you initially installed), but that you can add to R to increase its functionality. You install packages once by using the `install.packages()` function. Then every time you want to use a package (such as tidyverse) in your script, you load it into your environment using the `library()` function. All the necessary packages for this class have already been installed.
- **pipe**: a pipe does what it sounds like: it pipes information from one function to the next. It is a part of the tidyverse (https://www.tidyverse.org/) **package**, which is the primary set of tools we use for data analysis in R. You will use the pipe a lot, and it looks like this: `%>%` . A shortcut for typing it is `ctrl+shift+m` (`cmd+option+m` on a Mac).

Part Three: The key to understanding programming…

The key to understanding a programming language like R is to understand how **information is passed around**. For the sake of this class, let's refer to information as **data**, although it won't always be tabular. Data can be stored in a variable or printed to the **console** (which is directly below a code chunk in R Notebooks). Data stored as a variable shows up in the environment, and can be referred to later in your script.

Additionally, data can be passed (or **piped**) through functions that filter, sort, aggregate, and/or mutate it in some way. But it will always end up either printing to the console or being stored in a variable.

Part Four: Time to write some code

To create Notebook files, use File > New File > R Notebook; or the first button in the button bar above (which looks like a white box with a green plus sign on it). Then you're ready to code!