# Intro to R, R Studio, and Markdown

## The RStudio Interface

The goal of this lab is to introduce you to R, RStudio, and R Markdown, 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, RStudio is a convenient interface, and R Markdown allows you to write reports that include writing, code, and output.

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.

Go ahead and launch RStudio (go to: https://posit.cloud/). You may also download R and R studio to your personal laptop if you wish, but save that as something to do on your own time.

The panel on the left is where the action happens. It's called the *console*. Every time 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. What is a package? Think of it as a neat collection of tools (that someone else built) that help us do common, repetitive tasks more efficiently. For this lab, and many others in the future, we will use the following R packages (we will cover more on each of these later):

- `dplyr`: for data wrangling
- `ggplot2`: for data visualization
- `tidyverse`: an umbrella package that encompasses the previous two and many more

If these packages are not already available in your R environment, you will need to install them. 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.

Next, you need to load these packages in your working environment. We do this with the `library` function. Run the following line in your console.

```
library(tidyverse)
```

Note that you need to *load* packages each time you relaunch RStudio.

### Creating a reproducible lab report

We will be using R Markdown to create reproducible lab reports. Remember, key pieces of ethical data science are (1) transparancy in your analyses, and (2) reproducibility.

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.

## Loading Data

In order to do much of anything interesting in R, we'll first need some data. Many R packages contain datasets that are available once the package is loaded. Sometimes, delivering these datasets is the main purpose of the package. Other times, these datasets are used in the examples that illustrate the functions that the package provides.

To see what data frames are available to your R session, use the `data()` command. This will cause a list of data objects organized by package to pop up in your RStudio window.

> Note: you should **NOT** include this command in an R Markdown document, since that window won't pop up!

```
data()
```

If you want to see only those data objects provided by a specific package, you can specify the `package` argument. For example, we might want to list all of the data objects provided by the `dplyr` package:

```
data(package = "dplyr")
```

> Note that not all data objects listed by `data()` are `data.frames`! Some of them are `lists` and some have class `matrix`. (Don't worry about the difference between `data.frame`, `list`, and `matrix` for now, we'll dive into that later!)

### Lazy-loading

Typically, data objects provided by R packages are loaded lazily. This means that you don't have to explicitly load the data with the `data()` function. Once the package is loaded, the data is made available to you. However, not **all** packages do this.

## Data from a CSV

Most frequently, you'll probably want to load your own data into R. The simplest and most common way to get data into R is to import it from a CSV file. CSV stands for **C**omma **S**eparated **V**alues, and it is a non-proprietary file format for sharing tabular data (data organized into rows of variables and columns of observations).

In your exploration of R, you may come across two differed "spellings" of commands that look very similar: `read.csv()` and `read_csv()`.

- The `read.csv()` command comes from `base` R; you don't need to load any packages to use it.
- However, this command has largely has been superseded by the `read_csv()` command from the `readr` package. `readr` is part of the `tidyverse` (which we loaded previously), and `read_csv()` is much faster than `read.csv()`. Additionally, this version will not automatically convert your character vectors to factors.

TL;DR: I can't think of any reason to use `read.csv()` when you have the option to use `read_csv()`. Using this function, you can read a CSV file from your computer, or straight from the Internet via a URL. CSVs

should have the file extension `.csv`. To get you started, run the following command to load the data from a comma-separated value (CSV) file hosted online:

```
magazines <- read_csv("http://gattonweb.uky.edu/sheather/book/docs/datasets/magazines.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 and hitting `Enter`.

This command instructs R to load some data (specifically, revenue and advertising data from *Advertising Age*'s 14th annual Magazine 300 Report, which was released in September 2003), and to save that data as an object with the name `magazines`. You should see that the workspace area in the upper righthand corner of the RStudio window now lists a dataset called `magazines` that has 204 observations (obs.) on 5 variables. The `<-` symbol you used in this command is the assignment operator in R. We read this symbol as "gets", for example if you were reading out loud the command we just ran you would say something like: "`magazines` gets *Advertising Age* data".

As you interact with R, you will create a series of objects. Sometimes you'll load them from a file as we have done here, and sometimes you'll create them yourself as the byproduct of a computation or some analysis you have performed.

We can view the contents of an object by typing its name into the console.

```
magazines
```

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 `magazines` in the *Environment* pane (upper right window) that lists the objects in your workspace. This will bring up an alternative display of the dataset in the *Data Viewer* (upper left window).

The colums of the dataset are as follows:

- `Magazine`: the name of the magazine
- `AdRevenue`: revenue from advertising (in thousands of $)
- `AdPages`: number of pages of paid advertising
- `SubRevenue`: revenue from paid subscriptions (in thousands of $)
- `NewsRevenue`: revenue from newsstand sales (in thousands of $)

Use the scrollbar on the right side of the console window to examine the complete dataset.

Note that the row numbers in the first column are not part of the dataset; 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 *Advertising Age*'s data in a kind of spreadsheet or table called a *data frame*.

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

```
glimpse(magazines)
```

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 data frame.

## Some Exploration

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

```
pull(magazines, AdRevenue)
```

This command will only show the `AdRevenue` for each magazine. In a sense, `pull()` extracts the column `AdRevenue` from the data frame `magazines`.

What command would you use to extract just the subscription revenue? Try it!

Notice that the way R has printed these data is different. When we looked at the complete data frame, we saw 204 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, 2280 follows [1], indicating that 2280 is the first entry in the vector. And if [45] starts a line, then that would mean the first number on that line would represent the 45th entry in the vector.

### Data visualization

R has some powerful functions for making graphics. We can create a simple plot of the amount of Ad Revenue per Ad Pages with the command

```
qplot(x = AdPages, y = AdRevenue, data = magazines)
```

The `qplot()` function (meaning "quick plot") considers the type of data you have provided it and makes the decision to visualize it with a scatterplot.

Notice that the command above again looks like a function, this time with three arguments separated by commas. The first two arguments in the `qplot()` function specify the variables for the x-axis and the y-axis and the third provides the name of the dataset where they can be found. If we wanted to connect the data points with lines, we could add a fourth argument to specify the geometry that we'd like.

```
qplot(x = AdPages, y = AdRevenue, data = magazines, geom = "line")
```

You might wonder how you are supposed to know that it was possible to add that fourth argument. 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.

```
?qplot
```

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.

Is there an apparent trend in the amount of advertising revenue relative to the number of ad pages? 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 revenue for advertising, subscriptions, and newsstand sales combined. To compute this, we could use the fact that R is really just a big calculator. We can type in mathematical expressions like

```
2280 + 854 + 16568
```

to see the total revenue for the *Weekly World News*. We could repeat this once for each magazine, but there is a faster way. If we add the vectors for `AdRevenue`, `SubRevenue`, and `NewsRevenue`, R will compute all sums simultaneously.

```
pull(magazines, AdRevenue) + pull(magazines, SubRevenue) + pull(magazines, NewsRevenue)
```

What you will see are 204 numbers (in that packed display, because we aren't looking at a data frame 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 data frame

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

```
magazines <- magazines %>%
  mutate(TotalRevenue = AdRevenue + SubRevenue + NewsRevenue)
```

The `%>%` operator is called the **pipe** operator. It takes the output of the previous expression and sends (or "pipes") it in as the first argument to the next function in the chain.

To continue our analogy with mathematical functions, `x %>% f(y)` is just another way of writing `f(x, y)`.

**A note on piping:** we can read these three lines of code as the following:

1. *Take the* `magazines` *dataset and* ***pipe*** *it into the* `mutate` *function.*
2. *Mutate the* `magazines` *dataset by creating a new variable called* `TotalRevenue` *that is the sum of the variables called* `AdRevenue`*,* `SubRevenue`*, and* `NewsRevenue`*.*
3. *Finally, assign the resulting dataset to the object called* `magazines`*, i.e. overwrite the old* `magazines` *dataset with the new one containing the new variable.*

This is equivalent to going through each row and adding up the `AdRevenue`, `SubRevenue`, and `NewsRevenue` values for that magazine and recording that value in a new column called `total`. We'll learn a bunch more about piping later.

**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 `TotalRevenue` that has been tacked on to the data frame. Remember, 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 `magazines`, so this command updates that dataset with the new mutated column.

Similarly to how we computed the total revenue, we can compute the average revenue per page for each magazine:

```
magazines <- magazines %>%
  mutate(PerPageRevenue = AdRevenue / AdPages)
```

**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 a magazine does better at newsstand (one-off) or subscription sales with the expression:

```
magazines <- magazines %>%
  mutate(OneOff = NewsRevenue > SubRevenue)
```

This command adds a new variable to the `magazines` dataframe containing the values of either `TRUE` if that magazine had higher newsstand revenue than subscription revenue, or `FALSE` otherwise (the answer may surprise you). For example, check out what happens if we sort the dataframe by descending `TotalRevenue`.

This variable contains a different kind of data than we have encountered so far. All other columns in the `magazines` data frame have values that are numerical. 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 sections, you recreated some of the displays and preliminary analysis of magazine revenue data. Your task for the rest of this session involves repeating these steps, but for **a dataset of your own choosing**.

Start a **new** R Markdown file, and do/answer the following:

1. Find or load a new dataset.
2. Explain where you found your dataset and who collected the data. Do you expect any biases or issues in the data?
3. Print a glimpse of the dataset.
4. What columns are included in the dataset?
5. How many records does it contain?
6. Choose two variables in the dataset you believe might be related. Explain how you think they are related.
7. Make a quick plot to confirm or disprove if/how those variable are related.
8. Add a new variable to your dataset.
9. Make a quick plot that shows the relationship between the new variable and one of the original variables. Describe the relationship.