CEMA 0907: Statistics in the Real World

Getting Started with Data and R

Anthony Scotina

Sarah the chimp

- In 1978, researchers Premack and Woodruff published a study in *Science* magazine, reporting an experiment where an adult chimpanzee named Sarah was shown videotapes of eight different scenarios of a human being faced with a problem.
- After each videotape showing, she was presented with two photographs, one of which depicted a possible solution to the problem.
- Sarah could pick the photograph with the correct solution for seven of the eight problems!

How?!

What are **two possible explanations** for Sarah getting 7 correct answers out of 8?

1. Sarah was just guessing and got lucky.

2. Sarah can do better than just guessing.

Which explanation do you think is better?

- I think explanation (1) is better. How can you convince me that (1) is *not* the better explanation?

Refuting Explanation (1)

Let's try to look at what Sarah's results would be, **if she just guessed**.

- What is a simple way to *model* guessing between two choices?

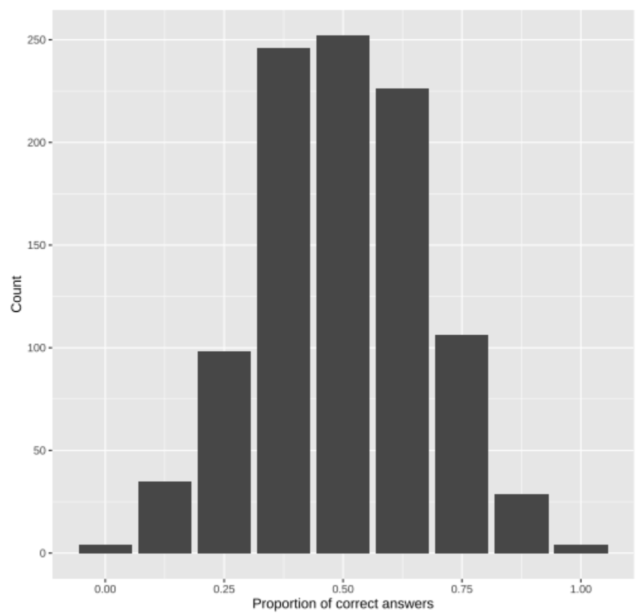
  Let's define "heads" as a *correct answer* and "tails" as an *incorrect answer*.

- If Sarah were just guessing ("flipping a coin"), what would be the **expected** number of correct guesses ("heads")?

Simulating Guessing

If Sarah were just guessing, we would *expect* the number of correct guesses to be 4.

- However, not every set of 8 coin tosses will result in 4 heads.
- Let's repeat the set of 8 coin tosses many times, to generate the pattern for correct answers that could happen in the long run, **under the assumption that Sarah is just guessing**.



What do you notice?

The **distribution** of the rate of correct answers, **under the assumption that Sarah was guessing**, is centered at 0.50 (50%, or 4 correct answers out of 8).

- The red line indicates the **observed proportion** of correct answers, 7 out of 8 (87.5%).

The majority of the distribution lies between 0.25 and 0.75.

- This means that, if Sarah were actually guessing, then it would be *highly unlikely* to observe 7 out of 8 correct answers.

- Thus, we are fairly convinced that Sarah is doing better than just guessing.

What if Sarah got 5 correct answers out of 8 instead? Would we still be convinced of Sarah's ability to do better than guessing?

SPOILER ALERT!!!

We just conducted a **statistical hypothesis test**, and this will be the last topic covered in Statistics in the Real World.

Let's start from the beginning...

Course Introductions

Who am I?

Anthony Scotina (he/him)

- Asst. Prof of Statistics at Simmons University
- Graduated with a Ph.D. in Biostatistics from Brown University in 2018.
- Website/blog: https://scotinastats.rbind.io/
- I used to have many hobbies, but all I do these days is use **R**.
- I have an 18-month old cat named **Moose**!

Who are you?

Where are you?

Statistics in the Real World!

Some information

Assignments:

- **Problem Sets and Participation**: Problem sets will be assigned *almost* daily. See the syllabus for a detailed schedule with due dates.
  - You will have access to the solution after submitting your problem set to Canvas, where you'll be able to self-assess your work.
- **Weekly Reflection**: After each week, you will be asked to write a short (1-2 paragraphs) reflection piece about your engagement and progress with the course content.
- **R Labs**: There will be R labs completed *in class* during roughly half of the class sessions. There are *not* to be turned in, and are primarily for your own practice.
- **Mini-Projects**: Groups of 4-5 students will be responsible for weekly *mini-projects*, using material covered in class each week. Short presentations will be given on Fridays.

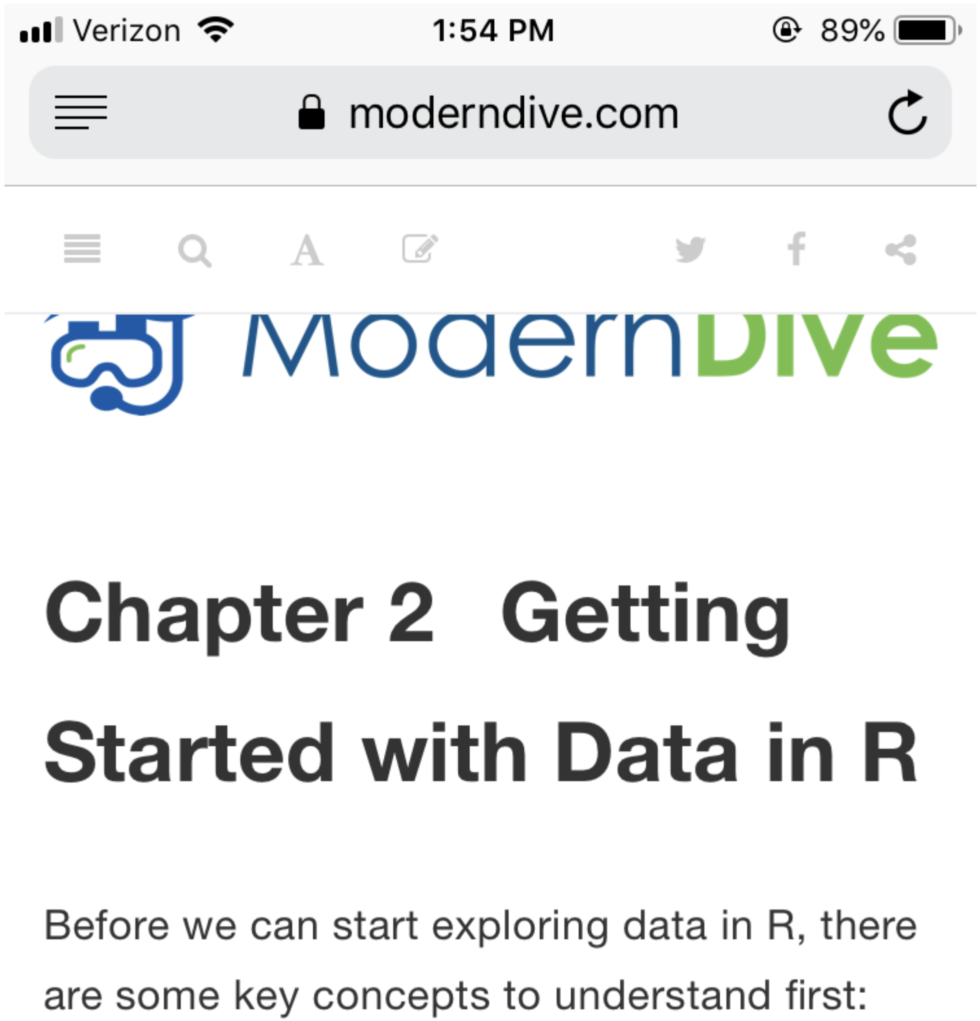
Where are you?

Statistics in the Real World!

Some information

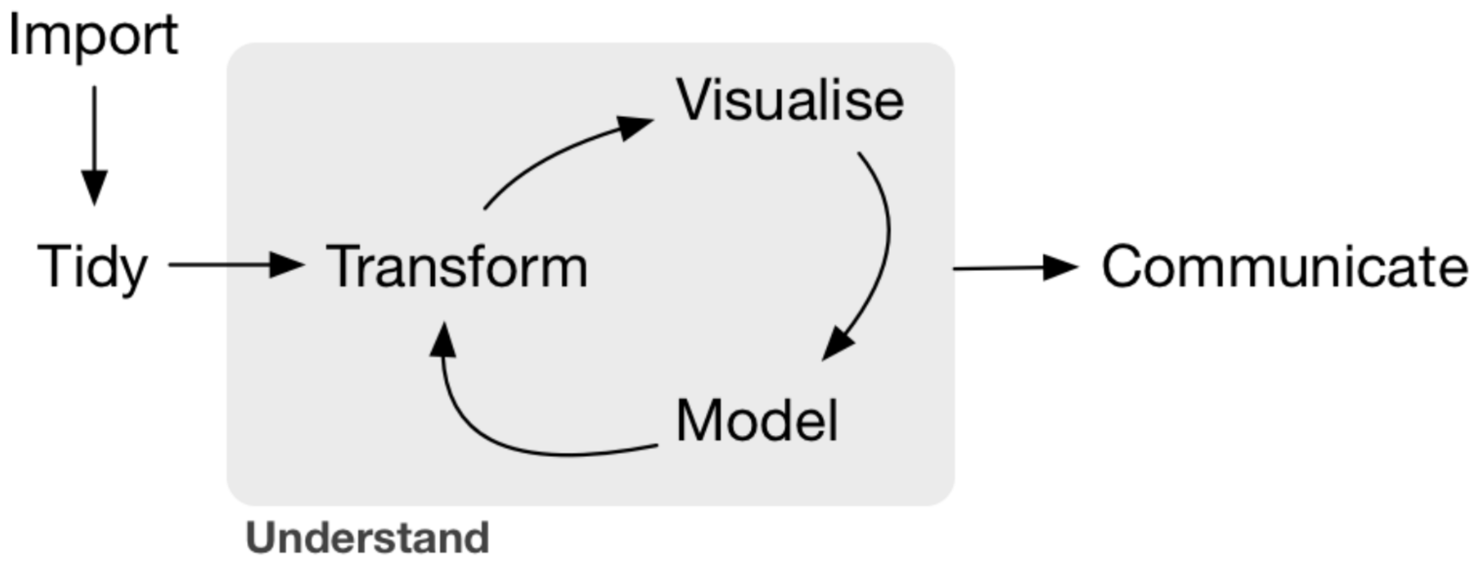
- Our textbook:
  - **ModernDive**: Statistical Inference via Data Science
  - Webpage: https://moderndive.com
  - Reading the assigned chapters in the textbook BEFORE EACH CLASS is **crucial**!!!

ModernDive


Verizon
1:54 PM
89%
moderndive.com
ModernDive
Chapter 2 Getting Started with Data in R
Before we can start exploring data in R, there are some key concepts to understand first:

Course Objectives

- Learn how to answer scientific questions with **data**.



- Statistics isn't just a bunch of numbers and math. We will aim to cover the entire **data science pipeline** in this course.

Course Objectives

In order to foster a conceptual understanding of statistics, use **real data** whenever possible.

How can we do this?

- Two engines:
    1. Mathematics: formulas, approximations, probability theory, etc.
    2. Computing: simulations, random number generating, etc.

- In *this* class:
    - Less of (1)
    - More of (2)

The "Engine"

RStudio

Getting Started with Data and R

First, let's install R and R Studio...

Installing R

- Click HERE to get started.

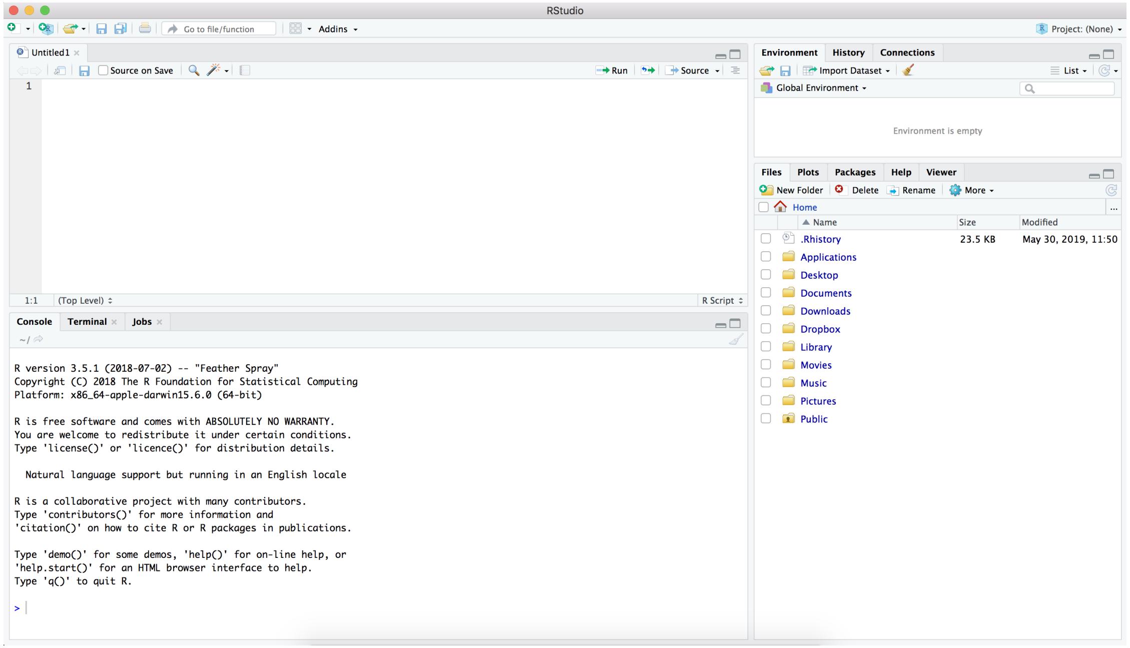
Installing R Studio

- Click HERE to get started.

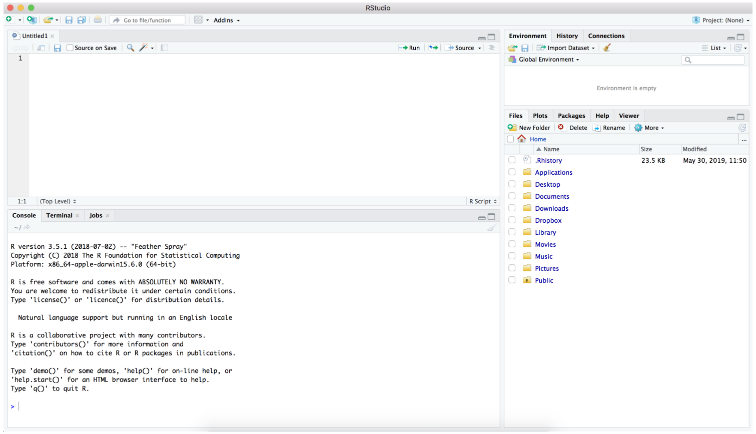
Using R Studio

1. Open **R Studio** (never open **R**).

2. In the menu bar at the top of your screen: **File** -- **New File** -- **R Markdown...**



The R Studio Window



The Four Panels:

1. **Console** (bottom-left): This is where you can crunch numbers or run/execute commands.
    - Either type code directly into the console, or run from a *script*…
2. **Editor** (top-left): This is where you can save and edit R code, text, etc.
    - *Save* all of your work in R Markdown (.Rmd) files!!!
3. **Files, Packages, Help, Plots** (bottom-right): See your files, packages, help screens, and plots (more in a few…).
4. **Environment** (top-right): Your current workspace (more in a few…).

Before we get started...

Don't worry. This class does not require you to have *any* experience with computer programming, nor is this a computer science class.

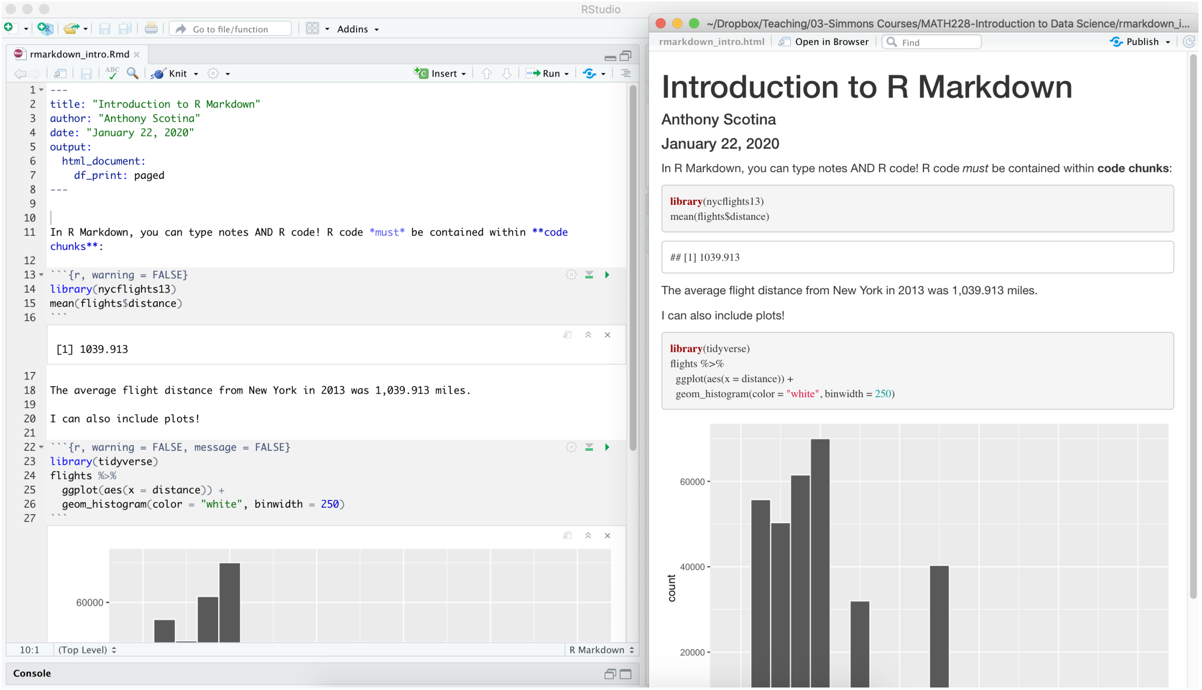
- "Should all statistics students be programmers? **No!**"
- "Should all statistics students program? **Yes!**"
    - Hadley Wickham, Chief Scientist at R Studio

Learning R is almost like learning a **new language**. It's difficult, but *incredibly rewarding*.

- You will learn tools that *actual* statisticians and data scientists use in the **real world**!

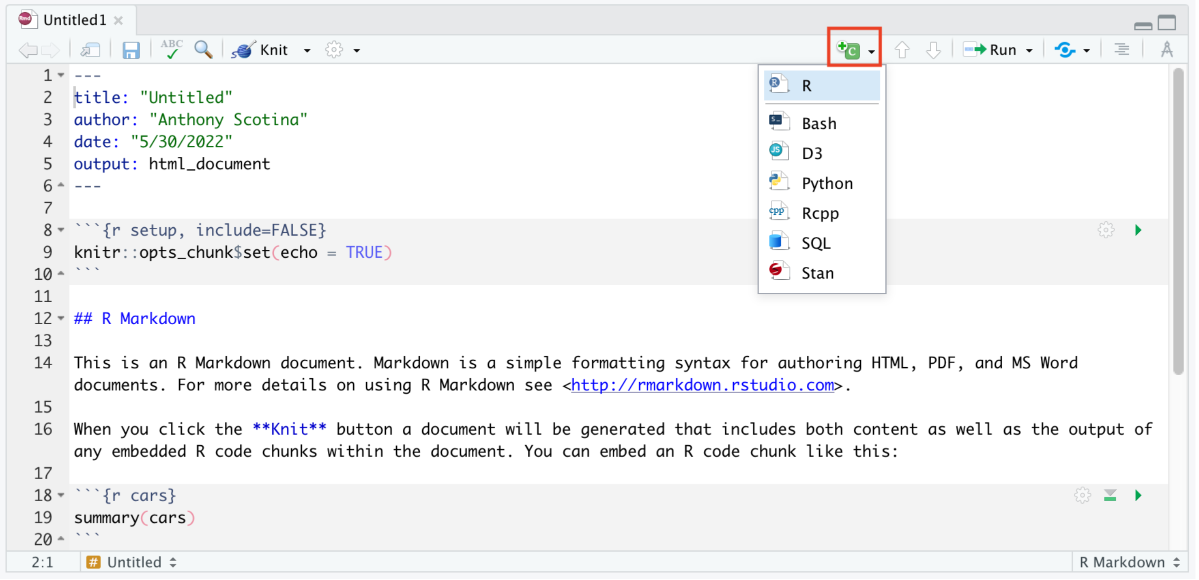
R Markdown Basics

R Markdown provides a way for R (and python/SQL) users to produce a single file containing code, output, and notes.



R Markdown Code Chunks

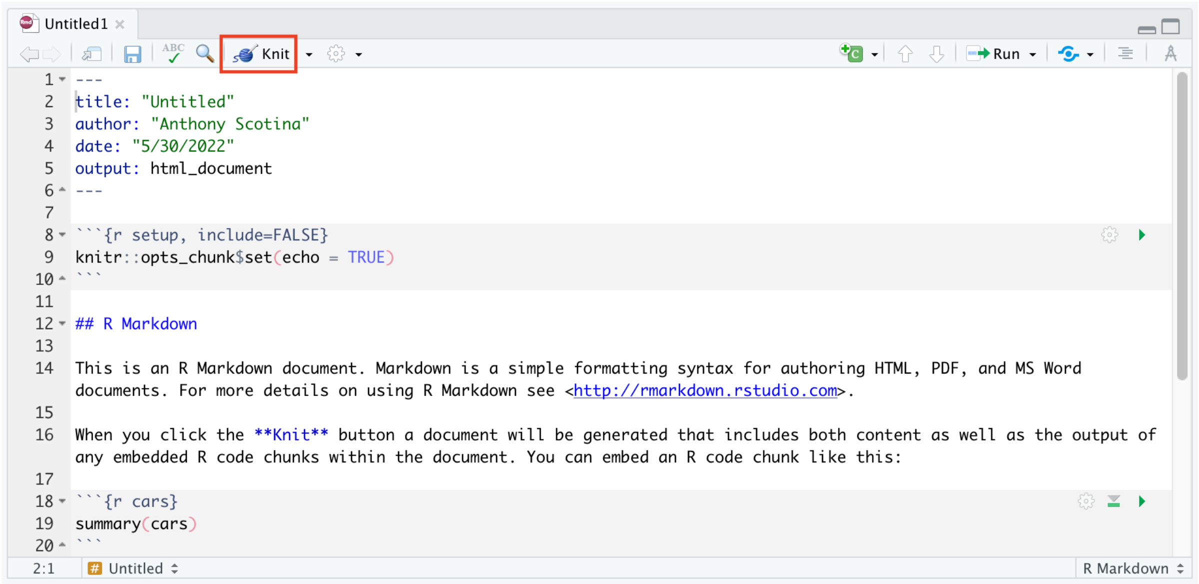
To enter and execute code in an R Markdown document, you'll need to create a **code chunk**.



- Or just use the **keyboard shortcut** for code chunks...
  - [command]+[option]+[i] for Macs
  - [ctrl]+[alt]+[i] for PC

R Markdown Knitting

To compile your R Markdown file into a finished .html (or PDF/Word doc) report, click the **Knit** button.



🚨 **Note** 🚨: This will *not work* if your code contains **errors**!

- Knit **early and often** in order to catch little errors *early*!

Code Chunk Options

- `echo = FALSE`: Don't *show* code
- `eval = FALSE`: Don't *evaluate* the code
- `include = FALSE`: Don't show the code or the results
- `message = FALSE`: Don't show the messages
  - This is usually relevant when you load a package but want to suppress the different "welcome" messages they might give.
- `warning = FALSE`: Don't show warning messages
- `out.width = "50%"` Makes a figure half the size (you can change the percentage to fit your needs).

In general, show your code and your results, but not your messages.

Some references

- R Markdown: The Definitive Guide, by Xie, Allaire, and Grolemund (here)
- R Markdown Cookbook, by Xie, Dervieux, and Riederer (here)

Vectors

R is built around **vectors**, which are probably the single-most important data structure you'll need to understand for this class.

Examples

```
## [1] 3 3 8 3 3 9
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
## [1] "I"     "have"  "a"     "cat"   "named" "Moose"
```

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

Vectors can take elements of *multiple types* (e.g., `numeric`, `character`, `logical`).

- But each vector's elements must *all* be the **same type**.

Creating Vectors

There are *many*, **many** ways to create vectors. One way is via the `c()` function:

```
c(3, 3, 8)

c("I", "have", "a", "cat", "named", "Moose")

c(TRUE, TRUE, FALSE, FALSE, TRUE)

c("Heads", "Tails")
```

- Each element is separated by a **comma**, and the *output* is a vector.

Creating Vectors: a : b

There are other ways to create vectors that can be *much* more useful than entering individual elements into `c()`.

- The `:` operator can be used to generate a sequence of *integers* from a **starting** value to an **end** value.

```r
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
0:1
```

```
## [1] 0 1
```

```r
-1:4
```

```
## [1] -1  0  1  2  3  4
```

Vector Operations

At its core, R is a big, *fancy* statistical calculator.

- While we can **add**, **subtract**, **multiply**, and **divide** numbers like you can in any calculator, we can also perform these operations (and more!) on vectors.

```
c(1, 2, 3, 4, 5) + c(6, 7, 8, 9, 10)
```

```
## [1]  7  9 11 13 15
```

```
c(1, 2, 3, 4, 5) * c(6, 7, 8, 9, 10)
```

```
## [1]  6 14 24 36 50
```

- 🚨 **Note**: Make sure vectors are the same *length* when doing this!

Assignment

We can store vectors under an **alias** so we don't have to keep typing out `c()`, `seq()`, etc.

```
my_vec = c(1, 2, 3, 4, 5)
```

```
my_vec^2
```

```
## [1]  1  4  9 16 25
```

```
my_vec + my_vec
```

```
## [1]  2  4  6  8 10
```

```
my_vec/2
```

```
## [1] 0.5 1.0 1.5 2.0 2.5
```

Logical Vectors

Logical vectors are made up of only *two unique "logical" elements*:

- `TRUE` or `FALSE`

"Behind the scenes", `TRUE` and `FALSE` have values of `1` and `0`, respectively.

```
c(TRUE, FALSE) + c(TRUE, FALSE)
```

```
## [1] 2 0
```

```
mean(c(TRUE, TRUE, TRUE, FALSE, FALSE))
```

```
## [1] 0.6
```

Logical Operators

Let's create two objects to use with some *logical tests*:

```
moose_age = 2 # rounding up
anthony_age = 32 # rounding down
```

Here are some commonly-used **logical operators**

- `==`: equal to
- `!=`: not equal to
- `>`: greater than
- `>=`: greater than or equal to
- `<`: less than
- `<=`: less than or equal to
- `%in%`: true if a value is **in** a vector

Logical Operators

The `==` operator asks whether two objects are **equal**.

The code below tests the following:

> Moose's age equals Anthony's age.

```
moose_age == anthony_age
```

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

> Moose's age does *not* equal Anthony's age.

```
moose_age != anthony_age
```

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

> "Moose's age is greater than Anthony's age.

```
moose_age > anthony_age
```

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

Combining Logicals

We can combine *several* logical operators to check multiple conditions!

- & for **and**, | for **or**

The code below tests the following:

> Moose's age is less than Anthony's age **and** Moose's age is less than 5.

```
(moose_age < anthony_age) &
  (moose_age < 5)
```

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

> Moose's age is less than Anthony's age **or** Anthony's age is less than 30.

```
(moose_age < anthony_age) |
  (anthony_age < 30)
```

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

Simulating Data

We won't do this *too much* in this class (we'll usually be working with **real data**), but we can *simulate data* using one of R's many built-in functions.

Let's randomly sample some data from a **normal distribution** (i.e., a *bell-shaped curve*).

```
set.seed(907) # Ro control R's random number generator
my_sample = rnorm(n = 1000, mean = 10, sd = 2)

my_sample
```

```
##    [1]  8.654466 10.216267 10.234048  7.330059 11.918426  9.724135 12.2421
##    [8]  8.455732 12.243928  7.455587  9.449277  5.438832 11.930787  9.1306
##   [15] 12.574922 10.727721  8.906333  6.638739  9.747085 12.065449  7.9471
##   [22]  7.642952  8.920384  9.021932  9.332082 10.206781  8.761498  7.4866
##   [29] 11.483212  6.561548  8.727129 12.197539  9.413502  5.790426  8.4162
##   [36] 10.835815  7.039804  7.331920 11.140476  7.372977 10.909648  8.3529
##   [43] 12.970961  9.393271  6.979053 10.013422  7.835197  6.817146 13.0925
##   [50] 11.382999  9.158072  9.819015  9.122685 10.685470 12.025734 12.6048
##   [57] 11.375301 10.042034 10.841477  6.788241  7.459775 10.970157 13.4808
##   [64]  7.552282 11.183549  9.627813 11.062646 10.113571 13.390081  9.8747
##   [71] 11.791086 12.844501  9.655099  7.571164 11.142211  8.961376  8.1313
```

Some Other Useful Functions

- `mean()`: calculates the **mean** of a vector

```
mean(my_sample)
```

```
## [1] 10.05411
```

- `sum()`: calculates the **sum** of a vector

```
sum(my_sample)
```

```
## [1] 10054.11
```

- `summary()`: calculates several **summary statistics** of a vector

```
summary(my_sample)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.288   8.659  10.108  10.054  11.444  15.672
```

- plus many more!

The `%in%` Operator

The `%in%` operator is *very useful* for checking whether *multiple* elements occur in a vector.

Recall `my_sample`. Let's check whether each element equals either `5`, `10`, or `15`:

```
my_sample == 5 | my_sample == 10 | my_sample == 15
```

- **Note**: The output is a **logical vector** that is the same length as `my_sample`.

Alternatively, we could use %in%:

```
my_sample %in% c(5, 10, 15)
```

R Packages

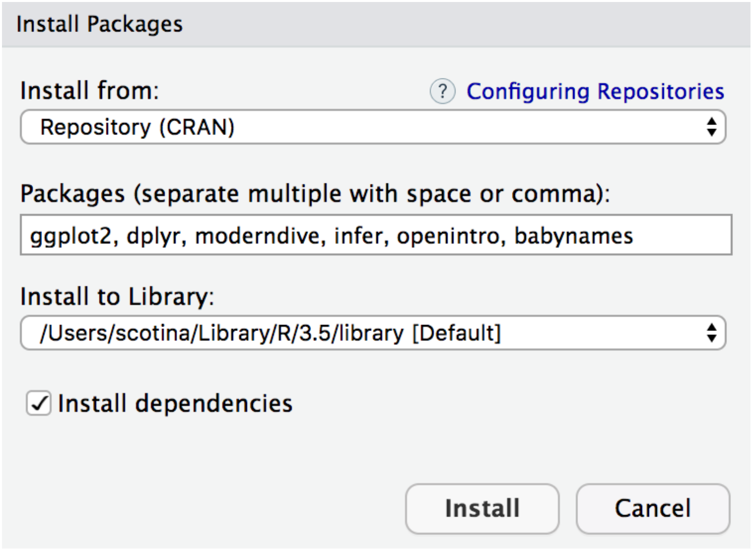
We will be using **R packages** extensively.

- R is *open-source*, which means that members in the community can provide additional functions, data, or documentation in a *package*.
- Packages are *free* and can be easily downloaded.

Downloading packages in R Studio

- **Packages** tab (bottom-right) -- **Install** -- Type package name and press *Install*
- **For now**, install the following packages (separate by a comma when typing the names):
  - `tidyverse`: suite of data science oriented packages
  - `moderndive`: package that accompanies the textbook
  - `infer`: package for statistical inference
  - `openintro`, `babynames`, `nycflights13`: packages with useful datasets

R Packages



Note: Once you install a package, *you never have to again*!

- But, you have to *load* them every time you open R Studio.
- To load a package, use the `library` function. Run the following:

```
library(tidyverse)
library(nycflights13)
```

`nycflights13` Package

This package contains five data sets saved in five separate **data frames** with information about all domestic flights departing from New York City in 2013:

1. `flights`: Information on all 336,776 flights
2. `airlines`: A table matching airline names and their two letter IATA airline codes (also known as carrier codes) for 16 airline companies
3. `planes`: Information about each of 3,322 physical aircrafts used.
4. `weather`: Hourly meteorological data for each of the three NYC airports.
5. `airports`: Airport names, codes, and locations for 1,458 destination airports.

- **Note**: *Data frames* and *tibbles* are analogous to rectangular spreadhseets you would see in Excel or Google Spreadsheets.

- Ideally, rows of a data frame correspond to unique *observations*, and columns correspond to *variables*.

`flights` Data Frame

Run the following:

```
flights
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_tim
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int
 1  2013     1     1      517            515         2      830            81
 2  2013     1     1      533            529         4      850            83
 3  2013     1     1      542            540         2      923            85
 4  2013     1     1      544            545        -1     1004           102
 5  2013     1     1      554            600        -6      812            83
 6  2013     1     1      554            558        -4      740            72
 7  2013     1     1      555            600        -5      913            85
 8  2013     1     1      557            600        -3      709            72
 9  2013     1     1      557            600        -3      838            84
10  2013     1     1      558            600        -2      753            74
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm
```

`flights` Data Frame

A few notes on this dataset...

- A "tibble" is a type of data frame in R. The `flights` data frame has:
  - 336,776 **rows**
  - 19 **columns**
- The 19 columns correspond to 19 different **variables**. Some of which are: *year*, *month*, *departure time*, *arrival time*, *carrier*, *origin*, etc.
- By default, we are shown the first 10 rows, since the rest can't fit on the screen.

Exploring Data Frames

There are many ways to explore a data frame besides what we just accomplished. One of which is through the `View` function.

- Run the following:

```
View(flights)
```

- **Note**: R is *case sensitive*. So make sure you use an uppercase "V" in `View`, rather than `view`.

Exploring Variables

The `$` operator allows us to explore a single variable within a data frame. For example, run the following in your console:

```
airlines
```

```
airlines$name
```

```
airlines$carrier
```

- The `$` extracts only the `name` variable from the `airlines` data frame and returns it as a **vector**.

Help Files

You can get help in R by entering a `?` before the name of a function or data frame, and a page will appear in the bottom-right panel.

- Try the following:

```
?flights
```

```
?mean
```

I use the help files **all the time**, and you should too, especially if you're stuck with a specific function!

What's to come?