CSSS 508, Lecture 1

Syllabus and Introduction to R, RStudio, and RMarkdown

Michael Pearce

(based on slides from Chuck Lanfear)

September 29, 2022

UW CS&SS

Welcome to CSSS 508: Introduction to R for Social Scientists

Today, we will do:

- Introductions
- Syllabus
- Lecture 1: Introduction to R, RStudio, and RMarkdown

Introductions

We'll go around the room and each share our:

- Name and preferred pronouns
- Program and year
- Experience with programming (in R or generally)
- Something fun you did over the summer

Syllabus

The syllabus (as well as lots of other information) can be found on our course website:

https://pearce790.github.io/CSSS508

Feel free to follow along online as I run through the syllabus!

Course Goals

This course is intended to give students a foundational understanding of programming in the statistical language R. General topics include:

- Exploring data with graphics and summaries
- Cleaning, preparing, and linking data for analyses
- Foundational programming skills such as functions and loops
- Organizing projects and creating reproducible research

We will cover almost no statistics here, but I hope you'll leave being able to focus on *statistics instead of coding* in future CSSS or STAT courses!

Logistics

Sessions:

- **Lecture:** Thursdays, 3:30-5:20 (Savery 117) -- Interactive sessions in which we'll learn key skills, concepts, and principles
- **Lab:** Tuesdays, 3:30-5:20 (Savery 117) -- Optional and mostly unstructured sessions to work on homework and review
- **Office Hours:** Wednesdays 10:30-11:30am and 3:30-4:30pm (on Zoom; link on Canvas)

Course Website: https://pearce790.github.io/CSSS508

Contact: Feel free to email me at [mpp790 at uw dot edu]

Schedule

- Week 0 (Th 9/29): Introduction to R, RStudio, and RMarkdown
- Week 1 (Th 10/6): Visualizing Data
- Week 2 (Th 10/13): Manipulating and Summarizing Data
- Week 3 (Th 10/20): Understanding R Data Structures
- Week 4 (Th 10/27): Importing, Exporting, and Cleaning Data
- Week 5 (Th 11/3): Using Loops
- Week 6 (Th 11/10): Writing Functions
- Week 7 (Th 11/17): Working with Text Data
- Week 8 (Th 11/24): Thanksgiving Holiday (No class; following lab will review past homeworks)
- Week 9 (Th 12/1): Working with Geographical Data
- Week 10 (Th 12/8): Reproducibility and Model Results

This course will have no meeting during final exam week.

Prereqs, Materials, and Texts

Prerequisites: None

Materials: All course materials are provided on the course website. This includes:

- These slides and the code used to generate them.
- An R script for the slides to follow along in class.
- Homework instructions and/or templates
- Useful links to other resources.

Laptops: It's help to bring a laptop to class. If you don't have one, you can use the lab computers or borrow one for free from the UW Student Technology Loan Program.

Textbooks: This course has no textbook. However, the website has links to a few texts which I have found useful!

Grading

Final grade: C/NC, 60% to get Credit.

- **Homework** (75%; assessed by peers): 9 total homeworks; assessed on a 0-3 point rubric. Assigned after Thursday lectures and due the following Thursday before lecture.
- **Peer Grading** (25%; assessed by the instructor): One per homework, assessed on a binary "good"/"not good" scale. Due the following Tuesday before lab.

Assignment/peer grading instructions and deadlines can be found on the Homework page of the course website. All homework will be turned in on Canvas.

Ugh, peer grading?

Yes, because:

- You will write your reports better knowing others will see them
- You learn alternate approaches to the same problem
- You will have more opportunities to practice and have the material sink in

How to peer review:

- Leave **constructive comments** -- more than just saying "good job!"
    - I expect at least 3-4 sentences saying what was good, interesting, could use improvement, etc.
    - Earlier assignments will need more feedback, later assignments less.
- **Email me** if you would like your assignment to be regraded or provide feedback if no peer review was given.

Academic Integrity

Academic integrity is essential to this course and to your learning. Violations of the academic integrity policy include but are not limited to:

- Copying from a peer
- Copying from an online resource
- Using resources from a previous iteration of the course.

I hope you will collaborate with peers on assignments and use Internet resources when questions arise to help solve issues. The key is that you **ultimately submit your own work**.

Anything found in violation of this policy will be automatically given a score of 0 with no exceptions. If the situation merits, it will also be reported to the UW Student Conduct Office, at which point it is out of my hands. If you have any questions about this policy, please do not hesitate to reach out and ask.

Classroom Environment

I am absolutely committed to fostering a friendly and inclusive classroom environment in which all students have an equal opportunity to learn and succeed.

- Names & Pronouns: Everyone should be addressed respectfully and correctly. Feel free to send me your preferred name/pronouns anytime.
- Covid: Covid creates unique circumstances for each of us, which may limit your ability to fully participate in this course. You never need to apologize to me for anything pandemic related. Let me know how I can help!
- Accessibility & Accomodations: See course website for information on health, disability, and religious accomodations.
- Feedback: I encourage feedback at any point in the quarter. I will also send out a mid-quarter evaluation around Week 5.
- Getting Help: If you ever find yourself struggling, know I'm here to help! Try Canvas discussion boards, chatting after class, email, or office hours.

Asking Questions

Don't ask like this:

> tried lm(y~x) but it iddn't work wat do

Instead, ask like this:

> ```
> y <- seq(1:10) + rnorm(10)
> x <- seq(0:10)
> model <- lm(y ~ x)
> ```
>
> Running the block above gives me the following error, anyone know why?
>
> ```
> Error in model.frame.default(formula = y ~ x,
> drop.unused.levels = TRUE) : variable lengths differ
> (found for 'x')
> ```

I may send out your question (anonymously) and my answer to the course mailing list!

Questions?

Lecture 1: Introduction to R, RStudio, and RMarkdown

A Note on Slide Formatting

Bold and *Italics* usually indicate emphasis or important terms!

`Code` represents R code you type into the editor or console or keystrokes used to perform actions, for example: "Press `Ctrl-P` to open the print dialogue."

Code chunks that span the page represent *actual R code embedded in the slides*.

```r
# Sometimes important stuff is highlighted!
7 * 49
```

```
## [1] 343
```

Why R?

R is a programming language built for statistical computing.

If one already knows Stata or similar software, why use R?

- R is *free*.
- R has a *very* large community.
- R can handle virtually any data format.
- R makes replication easy.
- R is a *language* so it can do *everything*.
- R skills transfer to other languages like Python and Julia.

R Studio

R Studio is a "front-end" or integrated development environment (IDE) for R that can make your life *easier*.

We'll show RStudio can...

- Organize your code, output, and plots
- Auto-complete code and highlight syntax
- Help view data and objects
- Enable easy integration of R code into documents with **R Markdown**

It can also...

- Manage `git` repositories
- Run interactive tutorials
- Handle other languages like C++, Python, SQL, HTML, and shell scripting

Selling You on R Markdown

The ability to create R Markdown files is a powerful advantage of R:

- Document analyses by combining text, code, and output
    - No copying and pasting into Word
    - Easy for collaborators to understand
    - Show as little or as much code as you want
- Produce many different document types as output
    - PDF documents
    - HTML webpages and reports
    - Word and PowerPoint documents
    - Presentations (like these slides)
- Works with LaTeX and HTML for math and more formatting control

We'll get back to this shortly!

Downloading R and RStudio

If you don't already have R and RStudio on your machine, now is the time to do so!

1. Go to the course homepage, https://pearce790.github.io/CSSS508
2. Click the *Download R* link and download R to your machine.
3. Afterwards, click the *Download RStudio* link and download RStudio to your machine.

We'll take a ~10 minute break now to stretch and solve any software issues!

Getting Started

Open up RStudio now and choose *File* > *New File* > *R Script*.

Then, let's get oriented with the interface:

- *Top Left*: Code **editor** pane, data viewer (browse with tabs)
- *Bottom Left*: **Console** for running code (`>` prompt)
- *Top Right*: List of objects in **environment**, code **history** tab.
- *Bottom Right*: Tabs for browsing files, viewing plots, managing packages, and viewing help files.

Editing and Running Code

There are several ways to run R code in RStudio:

- Highlight lines in the **editor** window and click *Run* at the top or hit `Ctrl+Enter` or `⌘+Enter` to run them all.
- With your **caret**[1] on a line you want to run, hit `Ctrl+Enter` or `⌘+Enter`. Note your caret moves to the next line, so you can run code sequentially with repeated presses.
- Type individual lines in the **console** and press `Enter`.
- In R Markdown documents, click within a code chunk and click the green arrow to run the chunk. The button beside that runs *all prior chunks*.

The console will show the lines you ran followed by any printed output.

[1] This thing is the caret: `|`

Incomplete Code

If you mess up (e.g. leave off a parenthesis), R might show a `+` sign prompting you to finish the command:

```
> (11-2
+
```

Finish the command or hit `Esc` to get out of this.

R as a Calculator

In the **console**, type `123 + 456 + 789` and hit `Enter`.

```r
123 + 456 + 789
```

```
## [1] 1368
```

The `[1]` in the output indicates the numeric **index** of the first element on that line.

Now in your blank R document in the **editor**, try typing the line `sqrt(400)` and either clicking *Run* or hitting `Ctrl+Enter` or `⌘+Enter`.

```r
sqrt(400)
```

```
## [1] 20
```

Functions and Help

`sqrt()` is an example of a **function** in R.

If we didn't have a good guess as to what `sqrt()` will do, we can type `?sqrt` in the console and look at the **Help** panel on the right.

```
?sqrt
```

Arguments are the *inputs* to a function. In this case, the only argument to `sqrt()` is `x` which can be a number or a vector of numbers.

Help files provide documentation on how to use functions and what functions produce.

Creating Objects

R stores everything as an **object**, including data, functions, models, and output.

Creating an object can be done using the **assignment operator**: `<-`

```
new.object <- 144
```

Operators like `<-` are functions that look like symbols but typically sit between their arguments (e.g. numbers or objects) instead of having them inside `()` like in `sqrt(x)`.

We do math with operators, e.g., `x + y`. `+` is the addition operator!

Calling Objects

You can display or "call" an object simply by using its name.

```r
new.object
```

```
## [1] 144
```

Object names can contain `_` and `.` in them but cannot *begin* with numbers. Try to be consistent in naming objects. RStudio auto-complete means *long names are better than vague ones*!

Good names save confusion later!

Using Objects

An object's **name** represents the information stored in that **object**, so you can treat the object's name as if it were the values stored inside.

```
new.object + 10
```

```
## [1] 154
```

```
new.object + new.object
```

```
## [1] 288
```

```
sqrt(new.object)
```

```
## [1] 12
```

Creating Vectors

A **vector** is a series of **elements**, such as numbers.

You can create a vector and store it as an object in the same way. To do this, use the function `c()` which stands for "combine" or "concatenate".

```r
new.object <- c(4, 9, 16, 25, 36)
new.object
```

```
## [1]  4  9 16 25 36
```

If you name an object the same name as an existing object, *it will overwrite it*.

You can provide a vector as an argument for many functions.

```r
sqrt(new.object)
```

```
## [1] 2 3 4 5 6
```

More Complex Objects

The same principles can be used to create more complex objects like **matrices**, **arrays**, **lists**, and **dataframes** (lists which look like matrices but can hold multiple data types at once).

Most data sets you will work with will be read into R and stored as a **dataframe**, so this course will mainly focus on manipulating and visualizing these objects.

Before we get into these, let's revisit R Markdown.

R Markdown

R Markdown Documents

Let's try making an R Markdown file:

1. Choose *File* > *New File* > *R Markdown...*
2. Make sure *HTML Output* is selected and click OK
3. Save the file somewhere, call it `my_first_rmd.Rmd`
4. Click the *Knit HTML* button
5. Watch the progress in the R Markdown pane, then gaze upon your result!

You may also open up the file in your computer's browser if you so desire, using the *Open in Browser* button at the top of the preview window.

If you ever have trouble knitting your file (especially if creating a PDF), try running the following code in the console:

```r
install.packages('rmarkdown')
install.packages('tinytex')
tinytex::install_tinytex()
```

R Markdown Headers

The header of an .Rmd file is a YAML (YAML Ain't Markup Language[1]) code block, and everything else is part of the main document.

```
---
title: "Untitled"
author: "Michael Pearce"
date: "September 29, 2022"
output: html_document
---
```

To mess with global formatting, you can modify the header[2].

```
output:
  html_document:
    theme: readable
```

[1] Nerds love recursive acronyms.
[2] Be careful though, YAML is space-sensitive; indents matter!

R Markdown Syntax

Output	Syntax
bold/strong emphasis	`**bold/strong emphasis**`
italic/normal emphasis	`*italic/normal emphasis*`
Header	`# Header`
Subheader	`## Subheader`
Subsubheader	`### Subsubheader`

Block quote from famous person	`> Block quote from`
	`> famous person`

More R Markdown Syntax

Output

1. Ordered lists
2. Are real easy
    1. Even with sublists
    2. Or when lazy with numbering

- Unordered lists
- Are also real easy
    - Also even with sublists

URLs are trivial

Syntax

```
1. Ordered lists
1. Are real easy
  1. Even with sublists
  1. Or when lazy with numbering


* Unordered lists
* Are also real easy
  + Also even with sublists


[URLs are trivial](http://www.uw.edu

![pictures too](http://depts.washing
```

Formulae and Syntax

Output

You can put some math $y = \left(\frac{2}{3}\right)^2$ right up in there.

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}_n$$

Or a sentence with `code-looking font`.

Or a block of code:

```
y <- 1:5
z <- y^2
```

Syntax

~~~
You can put some math $y= \left(
\frac{2}{3} \right)^2$ right up in
there

`$$\frac{1}{n} \sum_{i=1}^{n}
x_i = \bar{x}_n$$`

Or a sentence with
`code-looking font`.

Or a block of code:

    ```
    y <- 1:5
    z <- y^2
    ```
~~~

R Markdown Tinkering

R Markdown docs can be modified in many ways. Visit these links for more information.

- Ways to modify the overall document appearance
- Ways to format parts of your document
- R Markdown: The Definitive Guide

Formatting Caveats

To keep R Markdown dead-simple, it lacks some features you might occasionally want to use. Your options for fancier documents are:

- Templates
- Use HTML with CSS for custom formatting[1]
- Use LaTeX files instead of .Rmd[2]

For day-to-day use, plain vanilla R Markdown does the job.

For handouts, memos, and homeworks, default R Markdown PDFs look surprisingly good!

[1] These slides were created using Xaringan, a blend of RMarkdown and CSS.

[2] LaTeX is powerful but exacts a terrible price.

R Code in R Markdown

Inside RMarkdown, lines of R code are called **chunks**. Code is sandwiched between sets of three backticks and `{r}`. This chunk of code...

~~~
```{r}
summary(cars)
```
~~~

Produces this output in your document:

```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
```

Chunk Options

Chunks have options that control what happens with their code, such as:

- `echo=FALSE`: Keeps R code from being shown in the document
- `eval=FALSE`: Shows R code in the document without running it
- `include=FALSE`: Hides all output but still runs code (good for `setup` chunks where you load packages!)
- `results='hide'`: Hides R's (non-plot) output from the document
- `cache=TRUE`: Saves results of running that chunk so if it takes a while, you won't have to re-run it each time you re-knit the document
- `fig.height=5, fig.width=5`: modify the dimensions of any plots that are generated in the chunk (units are in inches)

Some of these can be modified using the gear-shaped *Modify Chunk Options* button in each chunk. There are a *lot* of other options, however.

Playing with Chunk Options

Try adding or changing the chunk options (separated by commas) for the two chunks in `my_first_Rmd.Rmd` and re-knitting to check what happens.

You can also name your chunks by putting something after the `r` before the chunk options.

~~~
```{r summarize_cars, echo=FALSE}
summary(cars)
```
~~~

After you name your chunks, look what happens in the dropdown on the bottom left of your editor pane.

Naming chunks allows you to browse through an RMarkdown document by named chunks.

You can also browse by sections named using headers and subheaders.

In-Line R code

Sometimes we want to insert a value directly into our text. We do that using code in single backticks starting off with `r`.

```
Four score and seven years ago is the same as `r 4*20 + 7` years.
```

Four score and seven years ago is the same as 87 years.

Maybe we've saved a variable in a chunk we want to reference in the text:

```r
x <- sqrt(77) # <- is how we assign objects
```

```
The value of `x` rounded to the nearest two decimals is `r round(x, 2)`.
```

The value of `x` rounded to the nearest two decimals is 8.77.

This is Amazing!

Having R dump values directly into your document protects you from silly mistakes:

- Never wonder "how did I come up with this quantity?" ever again: Just look at your formula in your .Rmd file!
- Consistency! No "find/replace" mishaps; reference a variable in-line throughout your document without manually updating if the calculation changes (e.g. reporting sample sizes).
- You are more likely to make a typo in a "hard-coded" number than you are to write R code that somehow runs but gives you the wrong thing.

Example: Keeping Dates

In your YAML header, make the date come from R's `Sys.time()` function by changing:

```
date: "March 30, 2016"
```

to:

```
date: "`r Sys.time()`"
```

Fancier option: Use this instead to take today's date and make it read nicely:[1]

```
date: "`r format(Sys.Date(), format='%B %d, %Y')`"
```

[1] `format(Sys.Date(), format='%B %d, %Y')` says "format system date as month name (`%B`), day-of-month (`%d`), and four-digit year (`%Y`): September 28, 2022. See `?strptime` for these format codes.

Data Frames

What's Up with `cars`?

In the sample R Markdown document you are working on, we can load the built-in data `cars`, which loads as a dataframe, a type of object mentioned earlier. Then, we can look at it in a couple different ways.

`data(cars)` loads this dataframe into the **Global Environment** (as a *promise*[1]).

`View(cars)` pops up a **Viewer** pane ("interactive" use only, don't put in R Markdown document!) or...

```
head(cars, 5) # prints first 5 rows, see tail() too
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
```

[1] Promises are *unevaluated arguments*. Read more about R's lazy evaluation here.

Tell Me More About `cars`

`str()` displays the structure of an object:

```
str(cars) # str[ucture]
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

`summary()` displays summary information[1]:

```
summary(cars)
```

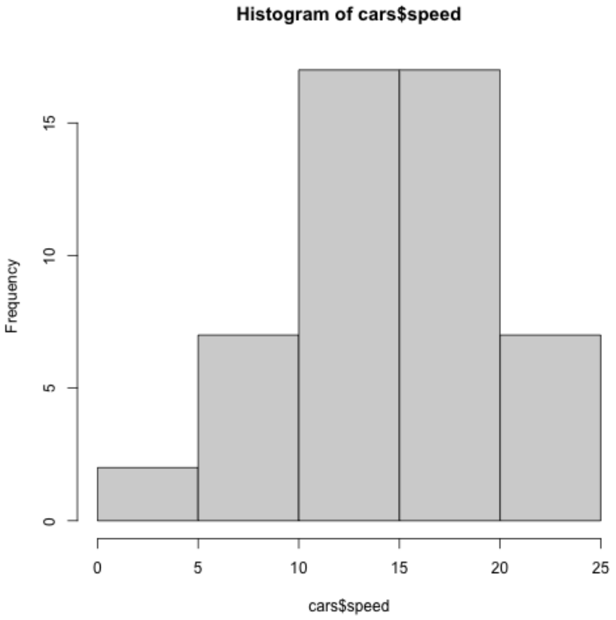
```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

[1] Note R is **object-oriented**: `summary()` provides different information for different types of objects!

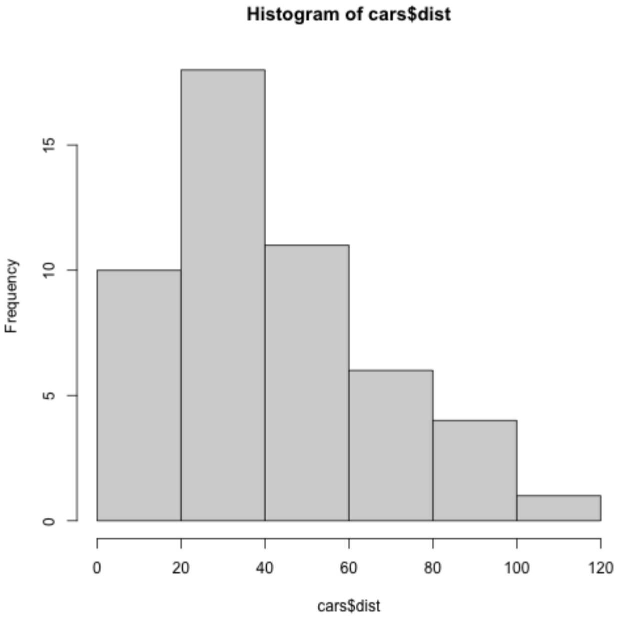
Ugly Pictures of `cars`

`hist()` generates a histogram of a vector. Note you can access a vector that is a column of a dataframe using `$`, the **extract operator**.

```
hist(cars$speed) # Histogram
```

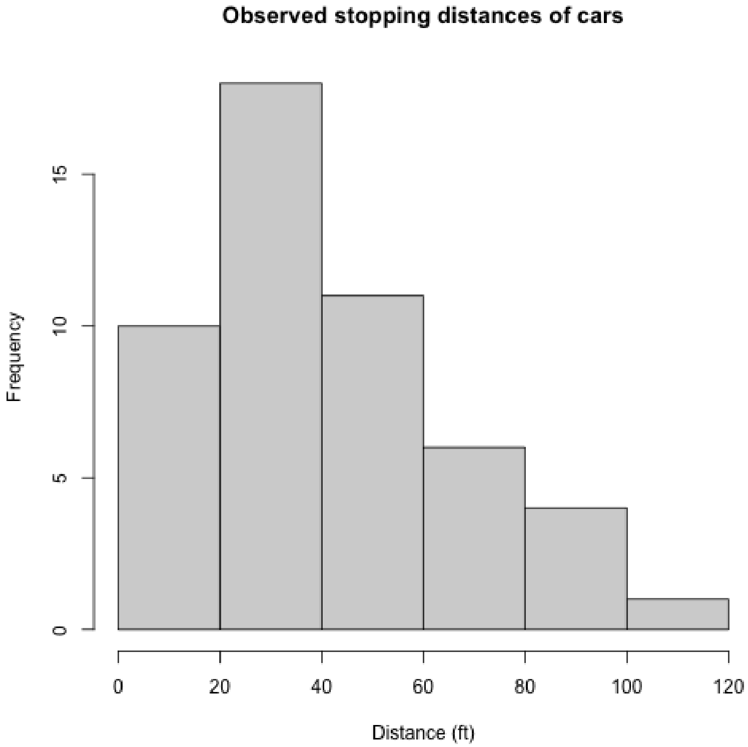


```
hist(cars$dist)
```



Drawing Slightly Less Ugly Pictures

```
hist(cars$dist,
     xlab = "Distance (ft)", # X axis label
     main = "Observed stopping distances of cars") # Title
```



Math with `cars`

If you put an assignment such as `x <- y` in parentheses `()`, R will print the output of the assignment out for you in your document. Otherwise, it won't show the value.

```
( dist_mean  <- mean(cars$dist) )
```

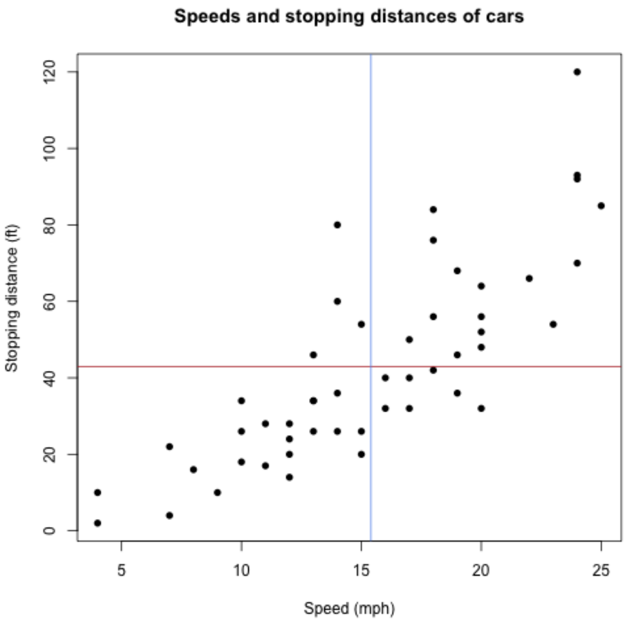
```
## [1] 42.98
```

```
( speed_mean <- mean(cars$speed) )
```

```
## [1] 15.4
```

Drawing Still Ugly Pictures

```r
plot(dist ~ speed, data = cars,
     xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     main = "Speeds and stopping distances of cars",
     pch = 16) # Point size
abline(h = dist_mean, col = "firebrick")
abline(v = speed_mean, col = "cornflowerblue")
```



Note that `dist ~ speed` is a **formula** of the type `y ~ x`. The first element (`dist`) gets plotted on the y-axis and the second (`speed`) goes on the x-axis. Regression formulae follow this convention as well!

`swiss` Time

Let's switch gears to the `swiss` data frame built in to R.

First, use `?swiss` to see what things mean.

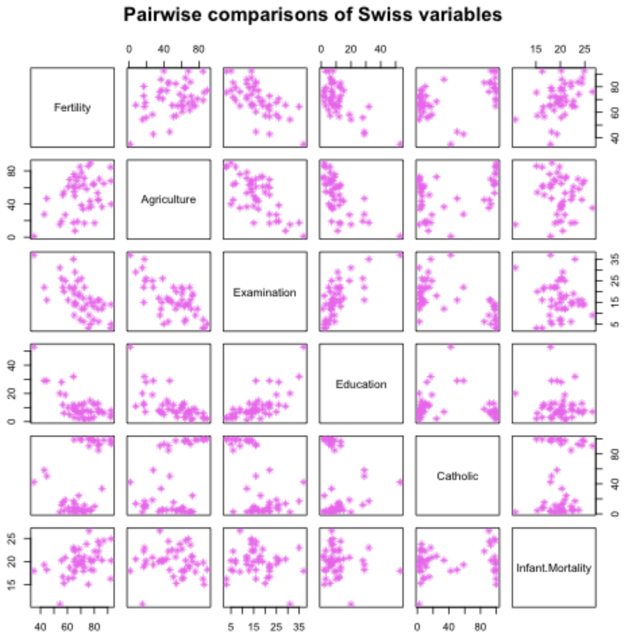
Then, load it using `data(swiss)`

Add chunks to your R Markdown document inspecting swiss, defining variables, doing some exploratory plots using hist or plot.

You might experiment with colors and shapes.

Looking at `swiss`

```
pairs(swiss, pch = 8, col = "violet",
      main = "Pairwise comparisons of Swiss variables")
```



`pairs()` is a pairwise scatterplot function. Good for a quick look at small datasets, but mostly useless for larger data.

Installing Packages

Let's make a table that looks a little less code-y in the output. To do this, we'll want to install a **package** called `pander`. Packages contain premade functions and/or data we can use. R's strength is its wide variety of packages!

In the console: `install.packages("pander")`.

- Note that unlike the `library()` command, *the name of a package to be installed must be in quotes*. This is because the name here is a search term (text, not an object!) while for `library()` it is an actual R object.

- Once you install a package, you don't need to re-install it until you update R. *Consequently, you should not include* `install.packages()` *in any markdown document or R script!*

Making Tables

```r
library(pander) # loads pander, do once in your session
pander(summary(swiss), style = "rmarkdown", split.tables = 120)
```

Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
Min. :35.00	Min. : 1.20	Min. : 3.00	Min. : 1.00	Min. : 2.150	Min. :10.80
1st Qu.:64.70	1st Qu.:35.90	1st Qu.:12.00	1st Qu.: 6.00	1st Qu.: 5.195	1st Qu.:18.15
Median :70.40	Median :54.10	Median :16.00	Median : 8.00	Median : 15.140	Median :20.00
Mean :70.14	Mean :50.66	Mean :16.49	Mean :10.98	Mean : 41.144	Mean :19.94
3rd Qu.:78.45	3rd Qu.:67.65	3rd Qu.:22.00	3rd Qu.:12.00	3rd Qu.: 93.125	3rd Qu.:21.70
Max. :92.50	Max. :89.70	Max. :37.00	Max. :53.00	Max. :100.000	Max. :26.60

Data Look a Little Nicer This Way

```r
pander(head(swiss, 5), style = "rmarkdown", split.tables = 120)
```

	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
Courtelary	80.2	17	15	12	9.96	22.2
Delemont	83.1	45.1	6	9	84.84	22.2
Franches-Mnt	92.5	39.7	5	5	93.4	20.2
Moutier	85.8	36.5	12	7	33.77	20.3
Neuveville	76.9	43.5	17	15	5.16	20.6

`split.tables = 120` tells `pander` to break a table into multiple tables if it will be over 120 characters wide. Adjust this to get widths *just right*.

Homework

Visit Homework Page for details.

Advice: Start with the provided template, then modify!

See you on Tuesday for lab!

(I'll stay for a bit after class to chat!)