Introduction to R

Simon Fraser University
ECON 483
Summer 2023

Disclaimer

I do not allow this content to be published without my consent.

Outline

- What is R?
- What is RStudio?
- Editing a script
- What is R Markdown?
- Working with data
  - Plots in R
  - Useful commands
- Loops in R

What is R?

- **R** is a computational language (Open source!)
- **RStudio** is a software that provides the interface for **R** commands
- Can be used to:
  - Analyze data (more tools than Excel)
  - Export data after making operations on them
  - Produce great figures, graphs and tables (nicer than Excel)
  - Perform mathematical operations (fancier than Excel)
  - Write pdf, html, presentation documents, and many more via **R Markdown**
- You need to download both! **R** here first and then **RStudio** here

How does it work?

- Write code (=instructions) in the **R** language
- Instructions are sent to the software (run/compile the code)
- **R** returns an output
- The output can be saved in various formats:
  - **.RData** (e.g. computations results, data sets)
  - **.csv** (e.g. data sets. What I recommend in practice)

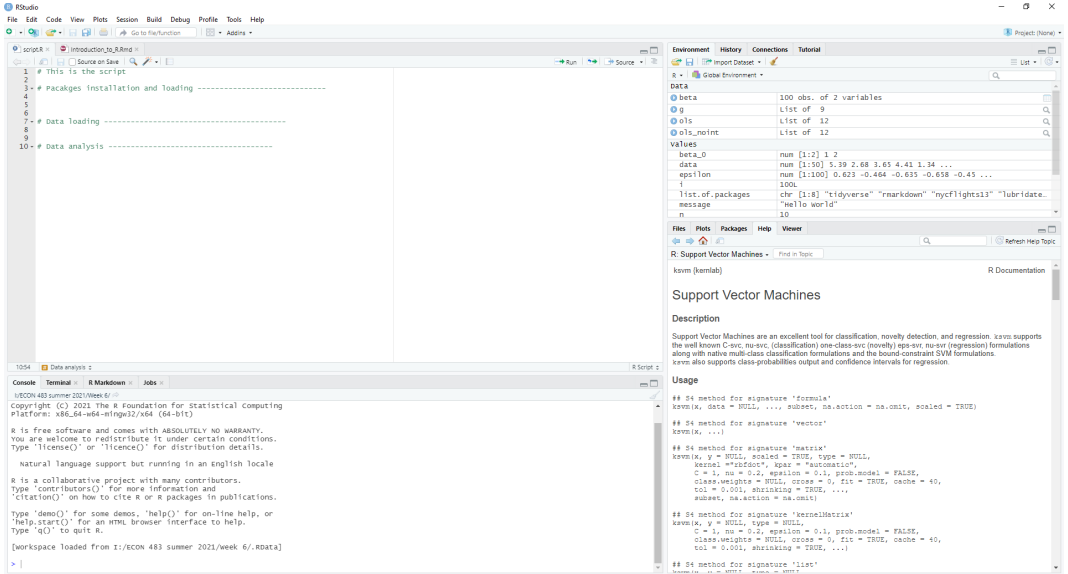
Learning R

- Many books and online resources
- **Swirl** (online step by step tutorial. Try the first couple of lessons!)
- **R cheat sheets**
- **R for data science** is a free online book that teaches data visualization, manipulation, etc... The author Hadley Wickham made huge contributions to the **R** community, including this book (the book itself was made in **R**!!)
- Like many languages have the same roots or structures (e.g. French/Spanish/Italian), the **R** language is very close to other languages like Matlab, Julia, Python. So it is a good investment no matter what
- Knowing at least one computational language is a big (if not the biggest) asset of a young economist these days

What is RStudio?

- **R** is the language, but the interface is not very user friendly...
- **RStudio** is an **Integrated Development Environment (IDE)** with a waaaay better interface
- It is composed of 4 panels:
  - Top left: This is your **script**, what you edit and save for later
  - Top right: This is the **working environment**. Saved variables, vectors, data sets appear on that panel
  - Bottom left: **The command window**. where the results appear. You can also type commands in it and run them directly (they are not saved in the script in this case)
  - Bottom right: Help files, plots, and working folders can be found among other things

RStudio interface



Workflow with **RStudio**

- Have a folder that contains the **.R** or **.Rmd** file, the data, the paper/assignment you are writing and anything related to the script (one project = one folder)
- Open the **.R** or **.Rmd** file by clicking on it every time, it sets the **working directory** directly in that folder (don't skip that step if you want to have a smooth experience. Trust me!!)
- This way, data can be loaded directly, without specifying the whole path of the file
- Believe me, it will avoid you a lot of struggles!

Workflow with **RStudio:** Editing a script

- **R** contains some functions (= commands), but many commands come in **packages**
- A package is a folder that contains commands for particular purposes
- First thing to do: Install the packages using `install.packages("package name")` (**RStudio** will download them from the internet and put them in the appropriate folder automatically)
- Installation can be done once per computer. Every time **RStudio** is open, you need to **load** the packages using the `library("package name")` command (**RStudio** will make the packages ready to use)
- Start editing your script. In general, it involves loading some data set, then performing computations on it and make some comments to keep track of your progress

Editing a script: Install and load packages

```
# This chunk of code takes a list of packages, checks if they are
# installed already and if not, installs them.
# Then it loads them and we are ready to go. Credit goes to Chris Muris
list.of.packages <- c("tidyverse","rmarkdown")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()
                                   [,"Package"])]
if(length(new.packages)) install.packages(new.packages)
library(tidyverse)
library(rmarkdown)
```

Editing a script: Basic commands

```
print("Hello, world!")     # Print() shows the output inside the brackets
```

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

```
"Ceci n' est pas une pipe" # Or we can make R show it by directly asking
```

```
## [1] "Ceci n' est pas une pipe"
```

```
# for it. Print() is more helpful in general
```

Editing a script: Basic commands

```
# You can save the message in the working directory
# (Top right-hand corner) using the <- or = symbol
message <- "Hello World"
message      # that will show what the variable "message" is
```

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

```
# Doing some basic statistical operations
data <- rnorm( 20, mean = 5 , sd = 2)  # Creates numbers "randomly",
# according to a normal distribution
head(data)  # shows the first rows and columns of the data
```

```
## [1] 6.4349540 5.1773072 6.7100623 5.6986034 0.8456626 4.6772494
```

```
mean(data)
```

```
## [1] 5.159391
```

Editing a script: Basic commands

```
mean(data)  # Computes the sample mean
```

```
## [1] 5.159391
```

```
mean(data^2) # Computes the sample mean of the data with each element squared
```

```
## [1] 28.91817
```

```
var(data)                              # variance derivation
```

```
## [1] 2.419847
```

```
sd(data)                               # standard deviation derivation formula
```

```
## [1] 1.555586
```

```
# standard deviation by hand, using the square root. Why are the results different???
sqrt(mean(data^2) - (mean(data))^2) # standard deviation by hand, using the square root
```

```
## [1] 1.516197
```

```
summary(data) # summary() shows a couple of statistics about the data
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8457  4.2622  5.0826  5.1594  6.5037  7.4653
```

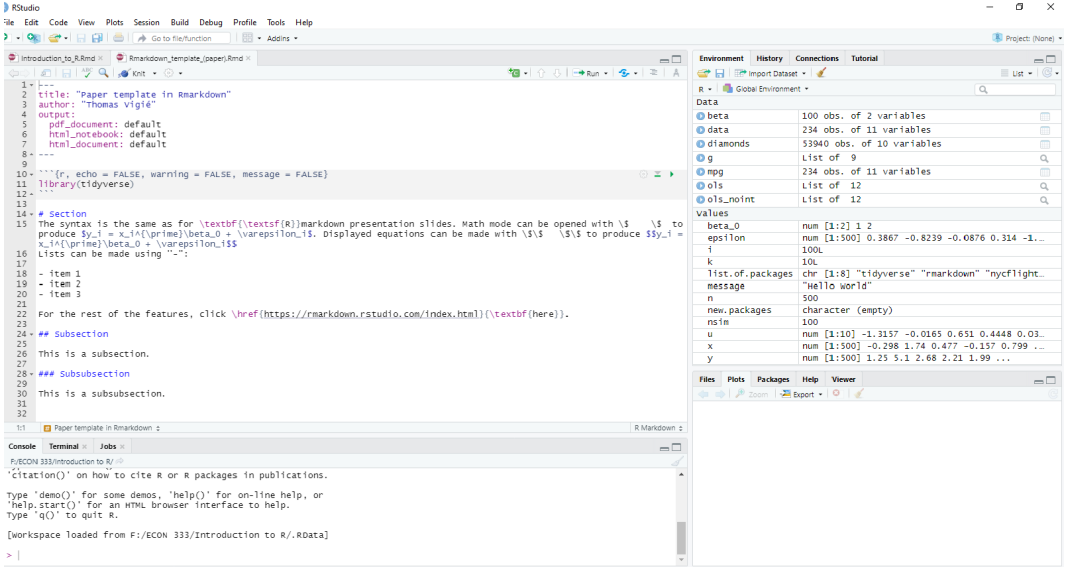
What is R Markdown?

- **R** is the language
- **RStudio** is the interface (the software you use to run everything)
- **R Markdown** is a file format for making dynamic documents with **R**
- We can write plain text, and add code chunks in the middle
- **RStudio** then compiles the code ("knits") and produces a pdf or html or Word document
- So you can write a paper, a presentation or your resume entirely with **R** using **R Markdown**
- The course lecture notes are made in **R Markdown**

Workflow with R Markdown

- Write plain text for sections, titles, bullet points etc
- Insert code chunks in between, using ***Ctrl + Alt + I*** (windows)
- Press on the **Knit** button to compile in the format of your choice
- You are **not required** to use **R Markdown** to produce your paper or presentation, but you are encouraged to try (templates will be made available on cnavas for you to play around)

R Markdown interface



Useful commands and shortcuts in **R Markdown**

- ***Ctrl + Shift + C*** (windows) or ***Cmd + Shift + C*** (MacOS) turns a line into a comment (same as **RStudio**, but the comment symbol is not # anymore, it is <!– commented text -->)
- A section starts with #, a subsection with ##, a subsubsection with ###
- Use the dash **-** to start an itemized (bullet points) list
- "**'' before and after a word produces bold face
- The dollar sign **$** is used to start the math mode (comes from Latex, an editing programming language). With it, you can type fancy mathematical expressions and symbols such as $\bar{\boldsymbol{X}}$, $\boldsymbol{X_i}$, $\sum_{i=1}^{n} (y_i - x_i'\beta)^2$
- See the **R Markdown** cheat sheet **here** online

Data analysis in R

```
data <- read_csv("vancouver_daily_crime.csv")   # Load the data, and call the data set "data"
```

```
## Rows: 5854 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (3): Theft, BreakAndEnter, OffenceAgainst
## date (1): date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
names(data)    # See the names of the variables in the data
```

```
## [1] "date"           "Theft"          "BreakAndEnter"  "OffenceAgainst"
```

Data analysis in R

```
head(data)      # See the first rows of the data
```

```
## # A tibble: 6 x 4
##   date       Theft BreakAndEnter OffenceAgainst
##   <date>     <dbl>          <dbl>           <dbl>
## 1 2003-01-01    89             32              31
## 2 2003-01-02    80             36              14
## 3 2003-01-03    80             40              15
## 4 2003-01-04    87             24              12
## 5 2003-01-05    69             21              12
## 6 2003-01-06    76             25              13
```

Data analysis in R

```
summary(data)  # Shows a summary of each variable
```

```
##       date                 Theft          BreakAndEnter    OffenceAgainst
##  Min.   :2003-01-01   Min.   : 12.0   Min.   :  2.00   Min.   : 0.00
##  1st Qu.:2007-01-03   1st Qu.: 42.0   1st Qu.: 12.00   1st Qu.: 7.00
##  Median :2011-01-05   Median : 54.0   Median : 16.00   Median :10.00
##  Mean   :2011-01-05   Mean   : 55.4   Mean   : 17.33   Mean   :10.07
##  3rd Qu.:2015-01-07   3rd Qu.: 67.0   3rd Qu.: 21.00   3rd Qu.:12.00
##  Max.   :2019-01-10   Max.   :131.0   Max.   :184.00   Max.   :43.00
```

Data analysis in R

```
nrow(data)  # Save the sample size
```

```
## [1] 5854
```

```
ncol(data)  # Save the number of variables in the data
```

```
## [1] 4
```

```
# data$Theft    # Looks at one variable in particular
data[6:8, 2]   # looks at the 6th column of the data, 6th to 8th rows
```

```
## # A tibble: 3 x 1
##   Theft
##   <dbl>
## 1    76
## 2    89
## 3    87
```

Plots in R: ggplot

- The **tidyverse** package contains a loooot of features for data analysis
- Highly recommended to always load **tidyverse**
- **ggplot** is one of them. It produces waaay better plots than the basic **plot** command
- It works by layers: First, you tell what data you are using to plot
- Then, you specify the mapping (what $\boldsymbol{x}$ and $\boldsymbol{y}$ are) using the **aes** command. If you want to assign a color to a third (qualitative) variable, you can use **color =**
- You can change the type of plot using **geom_** (**geom_point**, **geom_histogram**, etc)
- Add more layers to change the way the plot looks: Axis labels, legend style, etc

ggplot in R: Illustration

```
mpg <- ggplot2::mpg # This is a data set on cars,
# contained in the ggplo2 package
names(mpg)    # See the names of the variables in the data
```

```
##  [1] "manufacturer" "model"        "displ"        "year"         "cyl"
##  [6] "trans"        "drv"          "cty"          "hwy"          "fl"
## [11] "class"
```

ggplot in R: Illustration

```
head(mpg)      # See the first row of the data
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

ggplot in R: Illustration

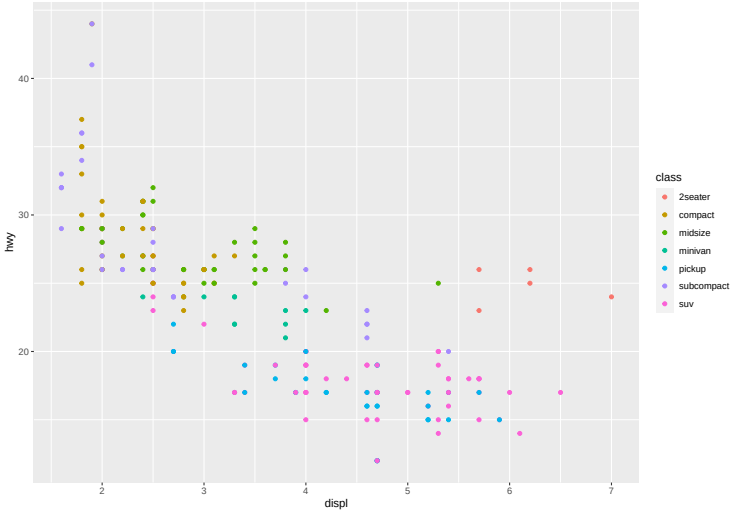
```
summary(mpg)  # Shows a summary of each variable
```

```
##  manufacturer          model               displ           year
##  Length:234         Length:234         Min.   :1.600   Min.   :1999
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999
##  Mode  :character   Mode  :character   Median :3.300   Median :2004
##                                        Mean   :3.472   Mean   :2004
##                                        3rd Qu.:4.600   3rd Qu.:2008
##                                        Max.   :7.000   Max.   :2008
##       cyl           trans               drv                cty
##  Min.   :4.000   Length:234         Length:234         Min.   : 9.00
##  1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00
##  Median :6.000   Mode  :character   Mode  :character   Median :17.00
##  Mean   :5.889                                         Mean   :16.86
##  3rd Qu.:8.000                                         3rd Qu.:19.00
##  Max.   :8.000                                         Max.   :35.00
##       hwy             fl               class
##  Min.   :12.00   Length:234         Length:234
##  1st Qu.:18.00   Class :character   Class :character
##  Median :24.00   Mode  :character   Mode  :character
##  Mean   :23.44
##  3rd Qu.:27.00
##  Max.   :44.00
```

ggplot in R: Illustration

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

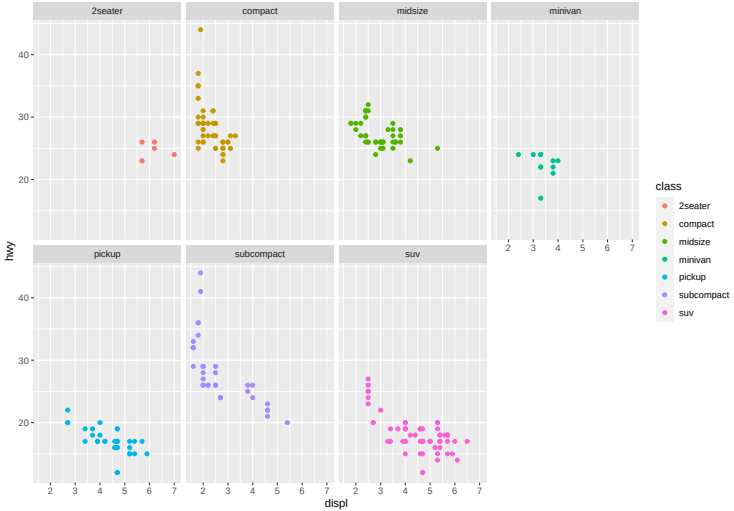
ggplot in R: Illustration



ggplot in R: Illustration

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy, color = class))+
facet_wrap(~ class, nrow = 2)
```

ggplot in R: Illustration



ggplot in R: Illustration

```
diamonds <- ggplot2::diamonds
names(diamonds)
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

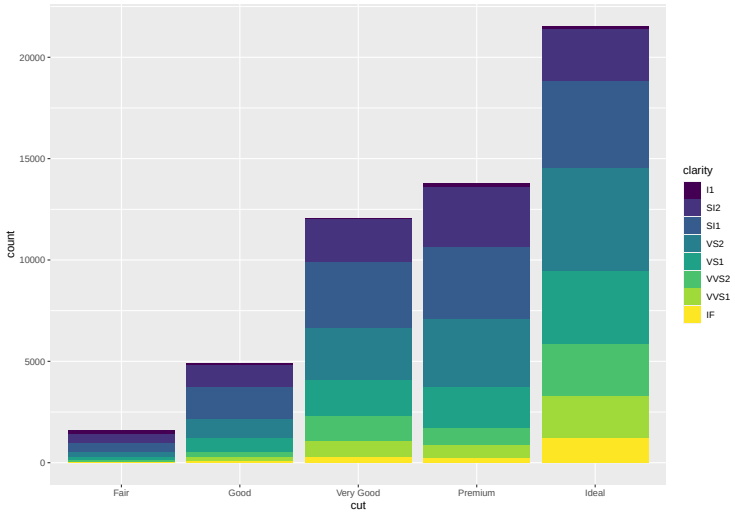
```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

ggplot in R: Illustration

```
ggplot(data = diamonds) +
geom_bar(mapping = aes(x = cut, fill = clarity))
```

ggplot in R: Illustration



Useful commands in **R**

- I will show the implementation of every method covered in the course
- Here are some basic but useful commands:
  - `head()` shows the first lines of a data set
  - `summary()` shows a summary of the data (good to check everything is ok)
  - `ggplot()` for graphs of all kinds (from the `ggplot2` or `tidyverse` packages)
  - `nrow()` and `nrow()` return the number of rows and columns of a data set
  - `ifelse()` creates a 1/0 variable (or other values) based on a condition we specify (1 if the condition is met, 0 otherwise)
  - `lm()` computes the OLS estimator (built-in **R**. See chapter on linear regression)
- Run `??function` to see help on how to use the command (look for the exact function among the ones proposed on the bottom right panel)
- Example: `??lm`

Loops in R

- A **loop** is a sequence of instructions that the software will execute in an iterated fashion
- 3 types of loops:
  - `for` loops repeat some instructions a predetermined number of times. Generally, an update is made at the beginning of each iteration
  - `while` loops execute a sequence of instructions as long as a criterion is satisfied. The criterion is checked at each iteration (so without an update, the loop may never stop!)
  - `if` "loops" check if a condition is satisfied, execute a command if the condition is **TRUE**, and another command if it is **FALSE**

Loops in R: Basic for loop

```
rounds <- 10
vec <- matrix(0, nrow = 1, ncol = rounds) # Making a matrix
#of 0 that the loop will replace with what I want

for (i in 1:rounds){    # i is the index: It will change at each iteration
vec[1, i] <- i
}
vec
```

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

Loops in R: Basic `while` loop

```
vec <- c() # Making an empty vector without a specified size
index <- 1
limit <- 15

while (index < limit){    # i is the index: It will change at each iteration
vec[index] <- index*3
index <- index + 1
}
vec
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30 33 36 39 42
```

```
length(vec)   # gives the number of elements in vec
```

```
## [1] 14
```

Loops in R: Basic if statements

```
x <- 5
if (x < 8){
  print("x is smaller than 8!!")
  }  else{
  print("x is bigger than 8!!")
  }
```

```
## [1] "x is smaller than 8!!"
```

```
# The ifelse function creates an object from  condition to check
y <- ifelse(x >= 5, "x is bigger or equal to 5", "x is strictly smaller tha
z <- ifelse(x > 2, x^2, 1)
y
```

```
## [1] "x is bigger or equal to 5"
```

```
z
```

Loops in R: A simulation

- Let us make a simulation
- In simulation, we are God: We can generate variables according to the assumptions we want to satisfy
- Simulations are used to see how an estimator does over multiple samples of different size
- We can check the **bias**, **variance**, **consistency** and **distribution** of estimators
- Let us use a loop to generate many samples and compute the mean of each

Loops in R: A simulation

```
rounds <- 10
vec <- matrix(0, nrow = 1, ncol = rounds) # Making a matrix
#of 0 that the loop will replace with what I want

for (i in 1:rounds){# i is the index: It will change at each iteration
x <- rnorm(100)
vec[i] <- mean(x)
}
vec
```

```
##              [,1]      [,2]         [,3]      [,4]        [,5]        [,6]
## [1,] -0.06348676 0.113284 -0.03552159 0.1314116 0.07989706 0.04693865 0
##            [,8]        [,9]        [,10]
## [1,] 0.1701509 0.09036108 -0.03161447
```

More data to look at

- Many data sets are available through packages. I recommend you check
  - `nycflights13`
  - `lubridate`
  - `Lock5Data`
  - `crimedata`
  - `fivethirtyeight`
- Be patient, use online help to learn. You will get a lot of errors, and some of them will be hard to decipher at first
- It opens the door to many other computer languages
- Have fun!