Intro to R and R Markdown

07/03/2020

Overview: Goals & exercises

- Learn the basics of the R programming language

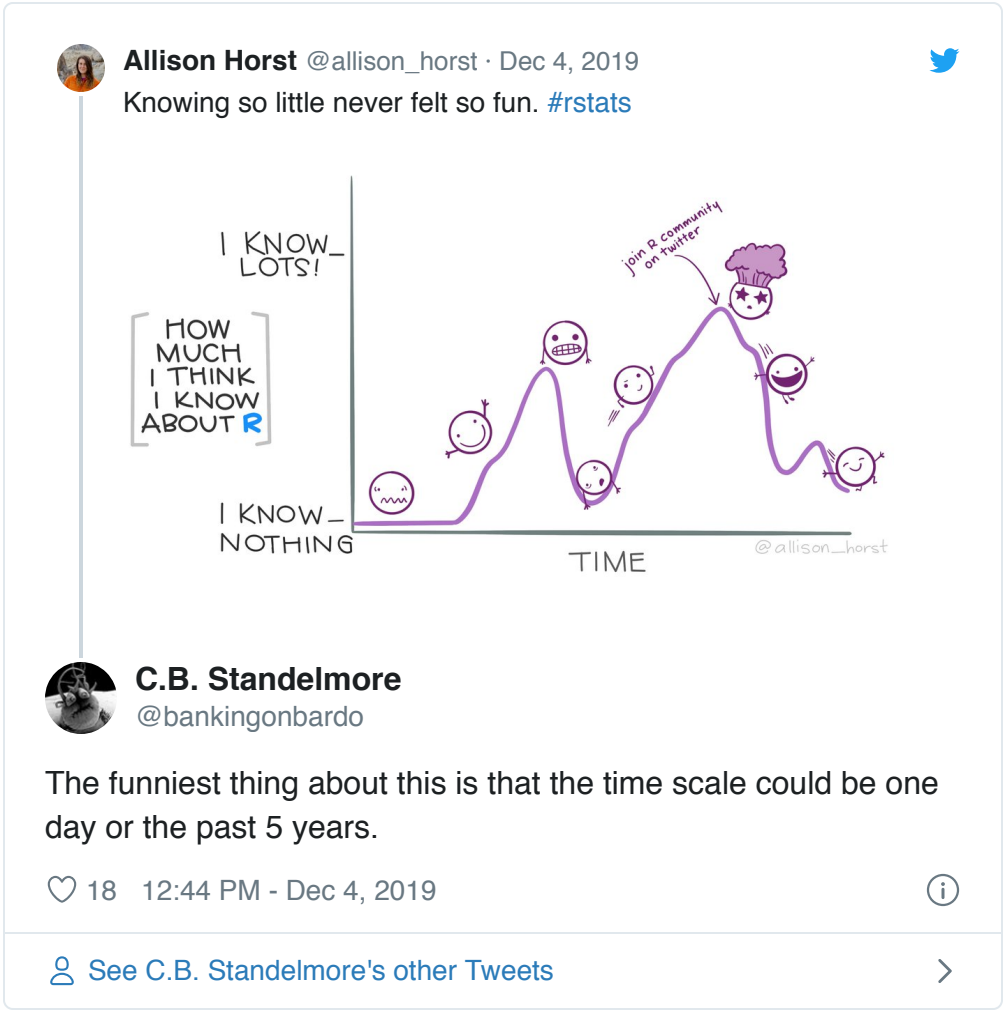
 1. Create new R project

 2-3. Create & edit helper.R

- Learn the basics of Markdown writing
- Learn how to combine R code and markdown text to create RMarkdown documents

 4-6. Create, edit, and compile .Rmd file

Don’t worry



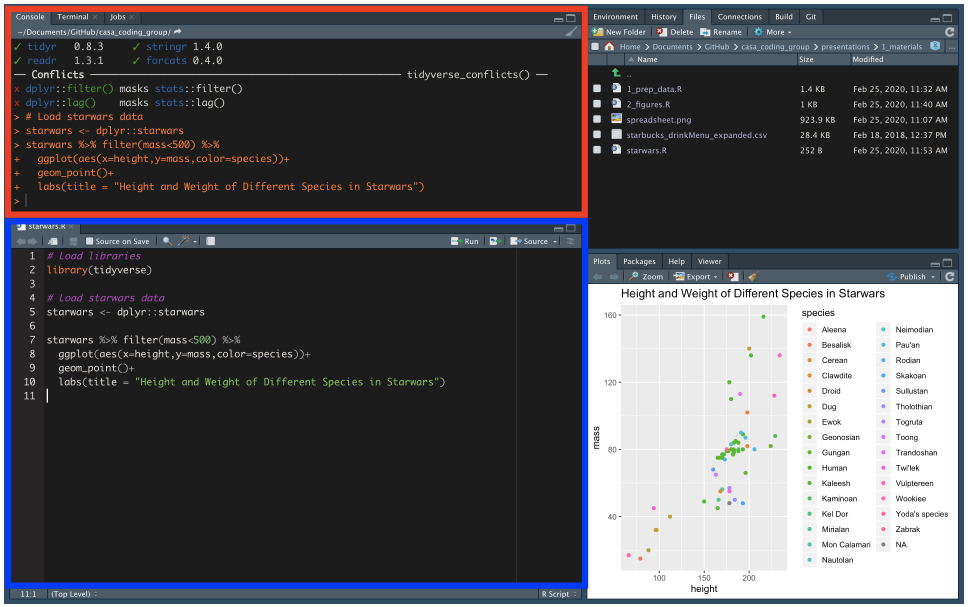
p.s. Allison Horst’s R illustrations are amaaaazing and I will be using them throughout.

Tool/interface to make it easy to work with R.

Programming language

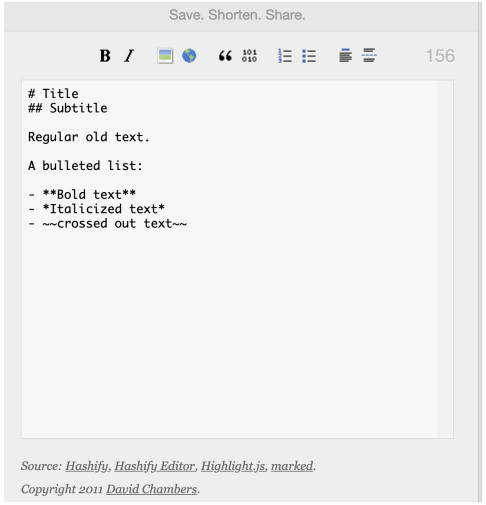
Package designed to integrate text & R code

Recap: Console and script panes in RStudio



- Console: Run code, see print outs, see warnings, messages, and errors
- Source: Run code from a script. Multiple scripts can be open at once.

What is “Markdown?”



Title

Subtitle

Regular old text.

A bulleted list:

- **Bold text**
- *Italicized text*
- ~~crossed out text~~

Hashify.me

- Simple syntax that allows you to add tags to plain text to format it
- Originally designed to be HTML replacement
  - Easier to learn and easier to read
- “Minimalist writing system”

Lifehacker: What is markdown and why is it better for my to do lists and notes?

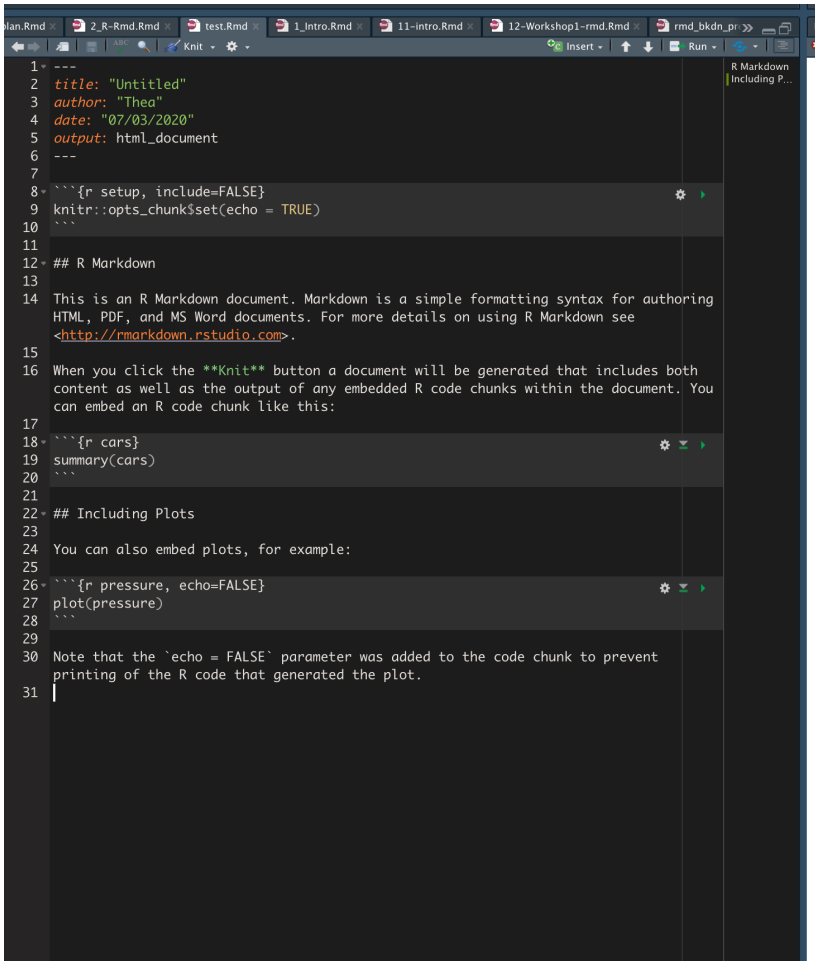
What is R Markdown?

https://rmarkdown.rstudio.com/

- Integrate R code directly into your writing using basic Markdown syntax
- Reference management integration
- Reproducibility
- Accessible learning curve

✏️ Very useful for writing summary reports, articles, etc.

R Markdown



```
---
title: "Untitled"
author: "Thea"
date: "07/03/2020"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring
HTML, PDF, and MS Word documents. For more details on using R Markdown see
<http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both
content as well as the output of any embedded R code chunks within the document. You
can embed an R code chunk like this:

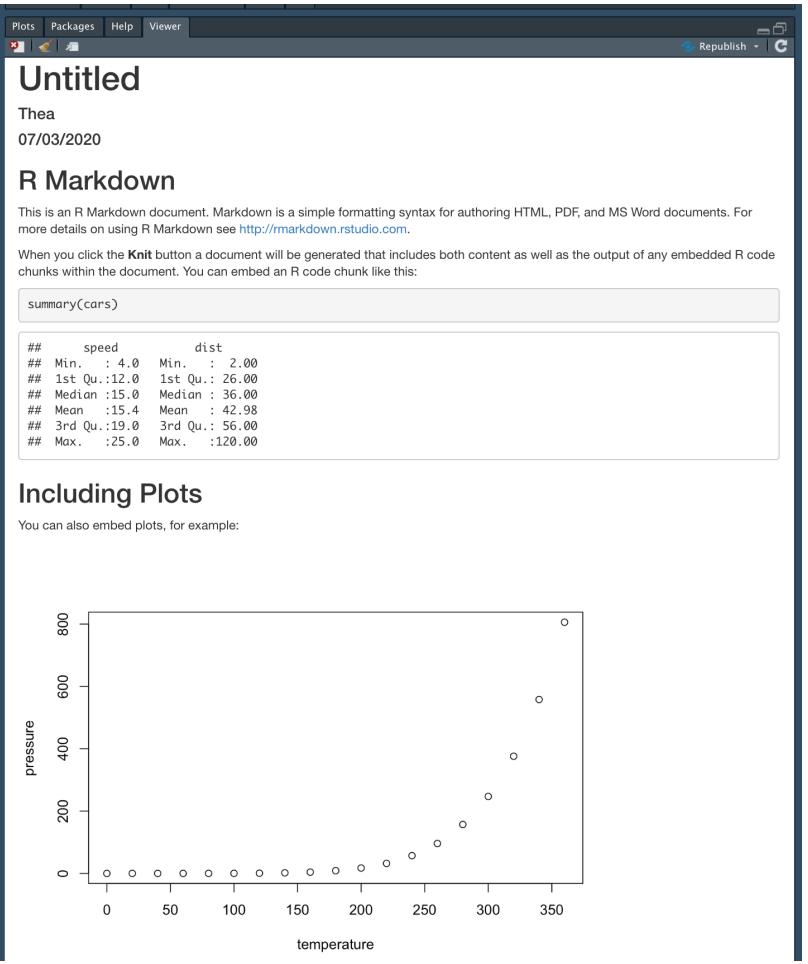
```{r cars}
summary(cars)
```

Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent
printing of the R code that generated the plot.
```



Untitled

Thea

07/03/2020

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
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
```

Including Plots

You can also embed plots, for example:

We’re going to make this today!

Intro to 

R Packages & Libraries

Packages are bundles of code written to do (typically) specific sets of functions

- Some packages are automatically downloaded and loaded into your workspace when you install R (`base`, `utils`, etc…) or R Studio (`rmarkdown`)
- Others you have to explicitly download
- Many packages are hosted on CRAN - this is the official "home" of peer-approved packages
    - These can be installed using the function `install.packages()`.

```
install.packages("tidyverse")
```

R Packages & Libraries

Non-CRAN packages

- Other packages are not hosted on CRAN - many of these are excellent, but some may be less reliable.
  - Many of these are hosted on GitHub.com
  - These usually have to be installed using a function `install_github()` which is part of the `remotes` package.

```
# install.packages("remotes")
# This installs the "emo" package for easy emoji use
remotes::install_github("hadley/emo")
```

Libraries

- "Library": where your R packages live after install (automatic location)
- "Load packages" with `library()` function to *use* a package.

```
# load the tidyverse package
# expect a bunch of output messages (this is normal)
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 -

## ✓ ggplot2 3.2.1     ✓ purrr   0.3.3
## ✓ tibble  2.1.3     ✓ dplyr   0.8.3
## ✓ tidyr   0.8.3     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.4.0

## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

What can you do with R code?

Use it like a calculator

```
3 + 2
```

```
## [1] 5
```

```
10^4
```

```
## [1] 10000
```

```
4*100
```

```
## [1] 400
```

What can you do with R code?

Print information to the console

```
print("Hello!")
```

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

```
print(3*5)
```

```
## [1] 15
```

What can you do with R code?

Create new objects

Read the following as "x is equal to 5"

```
x <- 5
```

And we can now type x in the console to see its value

```
x
```

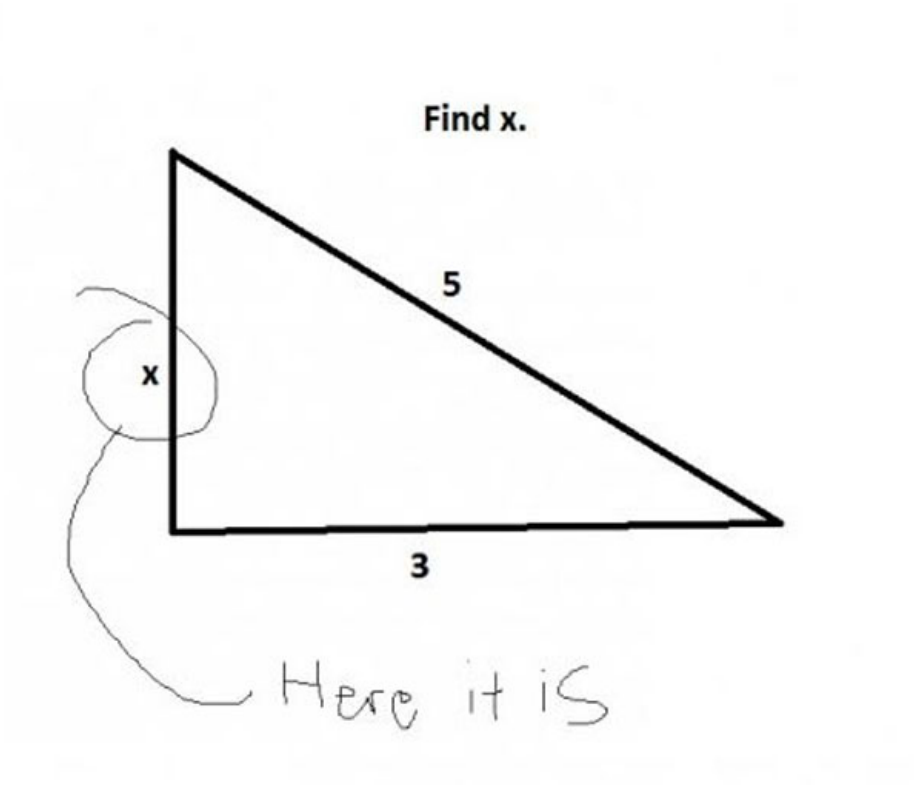
```
## [1] 5
```

x is now a **variable** that is set to the value of the number 5.

Variables

Variable: a symbol that stores/represents some other *value* or set of values.

Think of variables as containers.



Variable assignment

Variables get **assigned** their value in R with a left arrow `<-`

Numeric variables represent numbers

```
x <- 5
```

This means the value of the variable we have named `x` is equal to 5.

String variables represent "strings" of text. Text has to be enclosed in quotes `""`

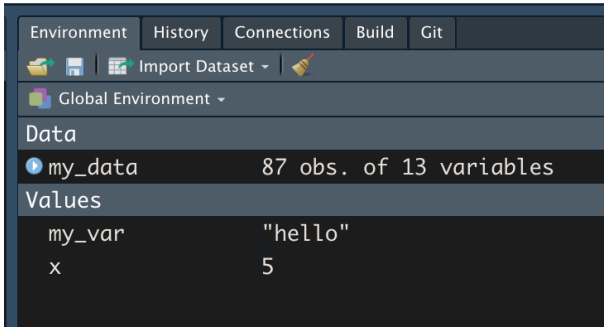
```
my_var <- "hello"
```

This means the value of the variable `my_var` is equal to the text "hello".

Variable assignment

Variables don't just hold numbers and text - they can hold sets of numbers/text, lists of various kinds of information, and whole data sets!

When you create a new variable in R, you will see it in your Environment pane.



What do I name my variables?

Best practices in naming your variables

In R, it's pretty flexible. Not as flexible in other languages (**cough** *Praat* **cough**)

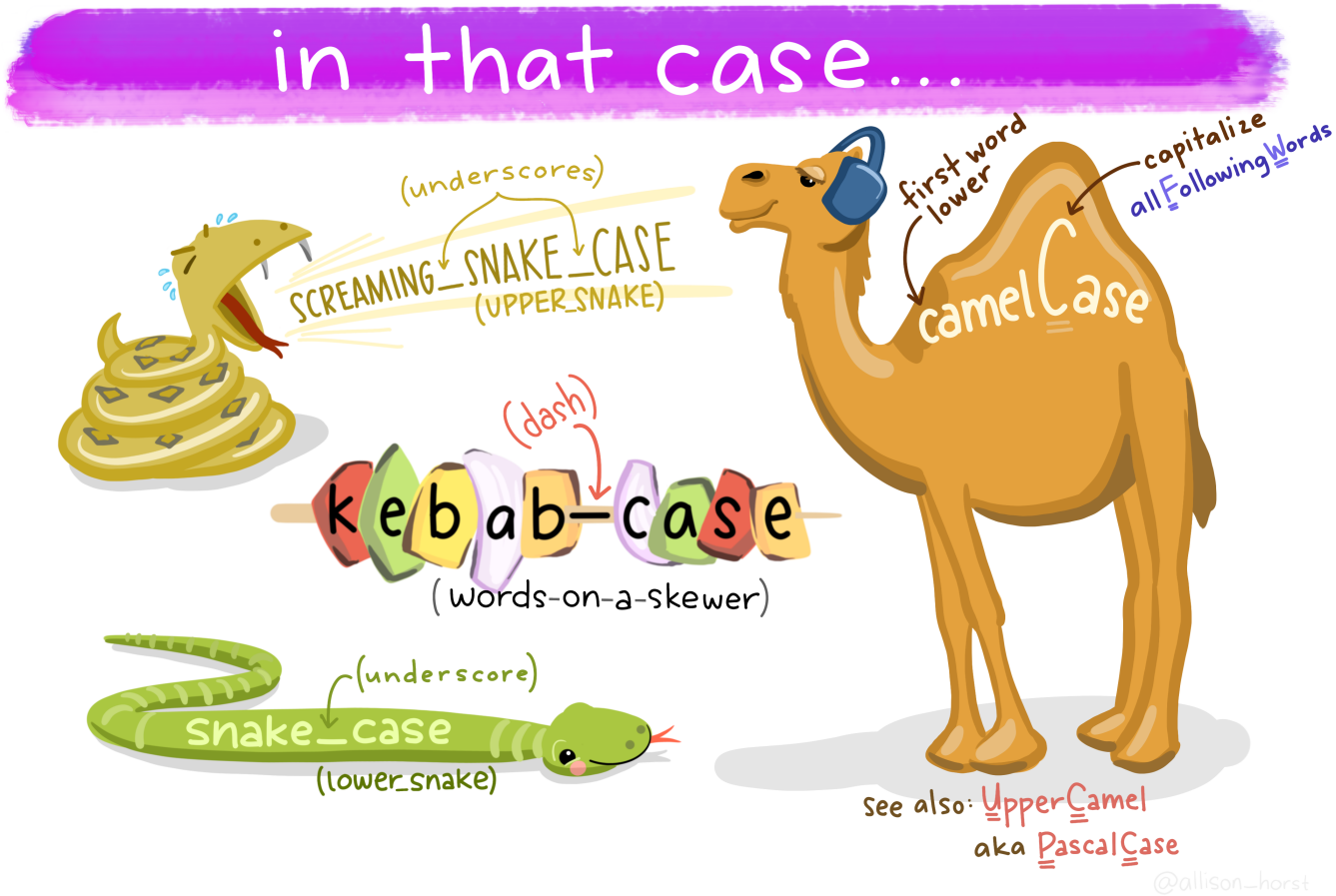
Some suggestions

Try your best to…

- Be descriptive
- Be consistent in your content
- Be consistent in your case (camel/snake/etc)

i_like_snake_case but some people *likeCamelCase* and others may *like-kabob-case*,

Variable names: Choose your case



https://github.com/allisonhorst/stats-illustrations

Variable names: Not okay

General rules:

Not allowed	Why
1abc	Starts with a number
intelligibility%	Contains a special character like !@#$%^&*, etc.
.1my_var	Starts with a period followed by a number (.1
_my_var	Starts with an underscore (_)
special characters	Some words have special meaning to R and you shouldn't overwrite them (e.g., "mean")

Variable names: Better

Allowed	Good because	Bad because
x	short, lowercase	Not meaningful
intelligibility_scores	meaningful, easy to remember	Takes a long time to type
int_mean	meaningful, short	🤷‍♀️
int_mean_pd, int_mean_hc	meaningful, shortish, descriptive	A bit long

Beware of typos!

```
my_var <- 5

My_var
```

```
## Error in eval(expr, envir, enclos): object 'My_var' not found
```

```
myVar
```

```
## Error in eval(expr, envir, enclos): object 'myVar' not found
```

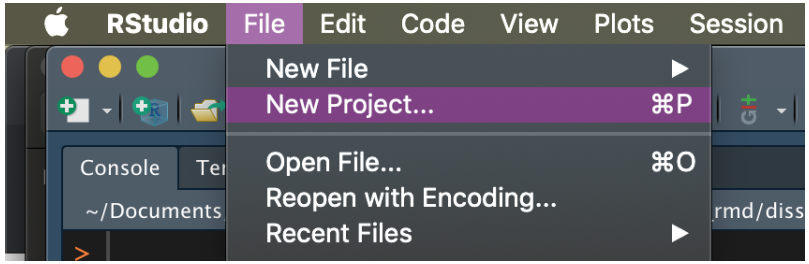
```
my_var
```

```
## [1] 5
```

Getting started: R Projects

R Projects make project management really simple. For every new project you embark on, creating a new .RProj file. Open that .RProj file whenver you're ready to work on that project, and it will:

- Provide easy access to the directory
- Restore your last RStudio session from that project
- Provide access to your R history from your last session



 Exercise 1

1. Create a new R Project for this group
2. Create a new R script: helper.R
3. Include the following lines:

```
# Load libraries ----
# install.packages("tidyverse")
library(tidyverse)

x <- 5
y <- x*10
my_var <- "hello"

y
```

```
## [1] 50
```

Exercise 2

Try running some code…

- From the script (Cmd or Ctrl + Enter to run the line your cursor's on)
- From the console (type code + hit enter)

Create some new variables

- Run them in the script
- Call them in the console

Types of objects in R

1. Numeric

```
1, 198758, 10^6, pi, abs(-6)
```

1. Strings: collection of characters, e.g…

```
"abc", "the rainbow is a division of white light", "12345", "$%^&"
```

1. Factors: categorical variables. Make up a finite set.

```
"Blue" "Green" "Red"
```

1. Logicals: special kind of factor that only has two values

```
TRUE vs FALSE, 0 vs 1
```

Types of data structures

Good explanation here

1. **Vectors**: collection of similar elements (numbers, characters, factors, etc..). 1 dimensional

```
# This vector contains 3 numeric components
c(1, 100, 1000)
```

1. Matrices: 2D arrangement of vectors
2. Lists: Can hold lots of different element types
3. Arrays: Vectors with 1 or more dimensions
4. **Data frames**: collections of vectors

We will mostly use data frames. More on those in a minute…

Comments

- In R scripts, any line of code preceded by a hashmark (#) is not *evaluated* (i.e., it's ignored)
- You can write notes to your future self this way
- To comment out a block of code, select it and type Cmd (or Ctrl) + Shift + C

```
# This line is a comment to myself.
# Set x equal to 0
x <- 0
# Now add 1 to the value of x
y <- x+1
```

Functions

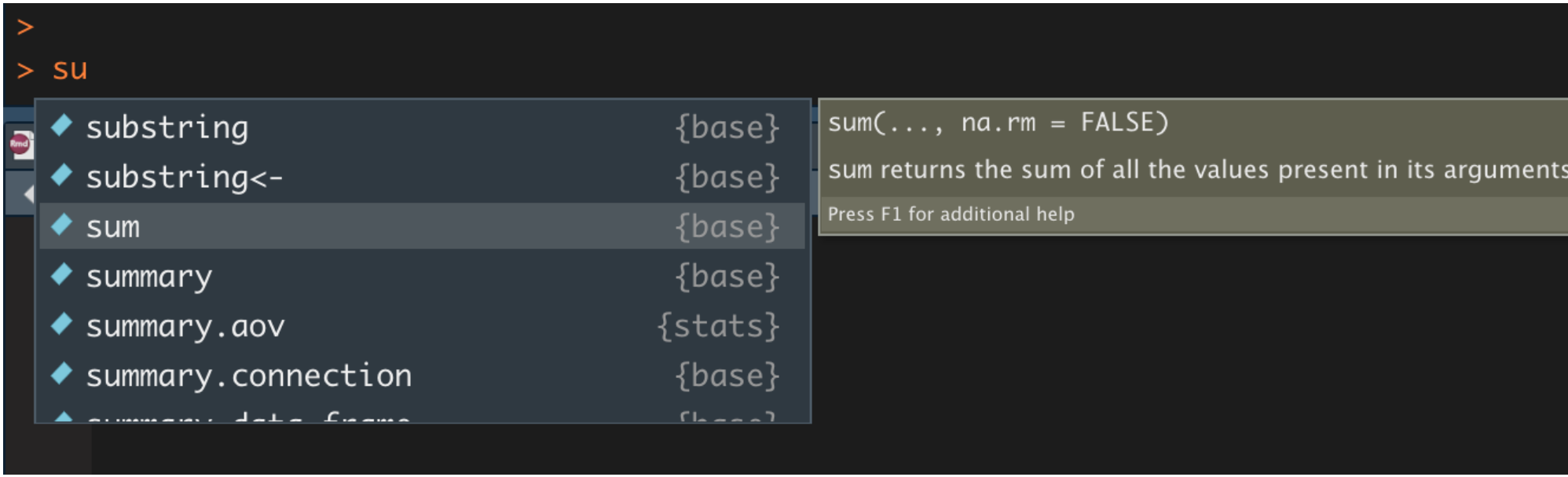
Functions: A certain named format of code that outlines a procedure. Often this allows several lines of code to be executed with a single line of code (by using the name of the function)

- In other words, functions are **actions**.
- Most functions take **arguments**
    - What do you want to act on?

Calling functions

function_name(argument1, argument2, …)

- In R Studio, if you start typing the name of a function and hit tab, you will get a pop up with possible options.
- Within the () of the function, pressing tab will help you add the arguments.



Looking at data: A spreadsheet perspective

Most of us are used to looking at data in the form of a spreadsheet.

- Named columns with many rows
- Each row usually represents one observation

See simulated_vot_data.csv

Looking at data: A spreadsheet perspective

My usual workflow:

- Collect data in spreadsheet.
- Load spreadsheet data into R as a **data frame**.
- Do all other data manipulation and analyses in R.
- Never touch the raw data again.

Looking at data

Tidy data = data that makes data analysis easy

Storage	Meaning
Column	Variable
Row	Observation
Data frame	Data set

Looking at data

To access individual columns, use the `$` operator

```
my_data$Participant
```

To peak at the first 6 rows of the data, use the `head()` function

```
head(my_data)
```

```
##        VOT Participant  Item Condition
## 1 43.37020       subj1 item1      Slow
## 2 39.11875       subj1 item2      Fast
## 3 34.18235       subj1 item3      Slow
## 4 30.46931       subj1 item4      Fast
## 5 19.98014       subj1 item5      Slow
## 6 37.17107       subj1 item6      Fast
```

Looking at data

We can also look at subsets of data

```
subset(my_data, Condition=="Fast")
```

```
##              VOT Participant   Item Condition
## 2     39.1187515        subj1  item2      Fast
## 4     30.4693097        subj1  item4      Fast
## 6     37.1710733        subj1  item6      Fast
## 8     46.9807138        subj1  item8      Fast
## 10    28.4736995        subj1 item10      Fast
## 12    38.3584361        subj1 item12      Fast
## 14    26.1054615        subj1 item14      Fast
## 16    31.6125367        subj1 item16      Fast
## 18    47.1704913        subj1 item18      Fast
## 20    41.2946820        subj1 item20      Fast
## 22    31.3528478        subj1 item22      Fast
## 24    34.6233935        subj1 item24      Fast
## 26    26.7056101        subj1 item26      Fast
## 28    38.3367957        subj1 item28      Fast
## 30    21.5010941        subj1 item30      Fast
## 32    17.7997123        subj1 item32      Fast
## 34    48.2864751        subj1 item34      Fast
```

Looking at data

Using the pipe %>% to combine functions

```
my_data %>%
  first_do(args) %>%
  then_do(args)
```

```
my_data %>%
  subset(Condition=="Fast") %>%
  head()
```

```
##         VOT Participant   Item Condition
## 2  39.11875       subj1  item2      Fast
## 4  30.46931       subj1  item4      Fast
## 6  37.17107       subj1  item6      Fast
## 8  46.98071       subj1  item8      Fast
## 10 28.47370       subj1 item10      Fast
## 12 38.35844       subj1 item12      Fast
```

Tidyverse: A word

"The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures." www.tidyverse.org

- Changed the game of R coding
- Much more intuitive syntax (IMHO); more "english-like"
- BUT not everyone likes it. Sometimes makes things easier, sometimes makes things more complicated.
- We'll be using both
- Core packages of the tidyverse: ggplot2, dplyr, tidyr, readr, stringr…
- "The pipe is syntactic sugar" - Hadley Wickham

📝 Exercise 3

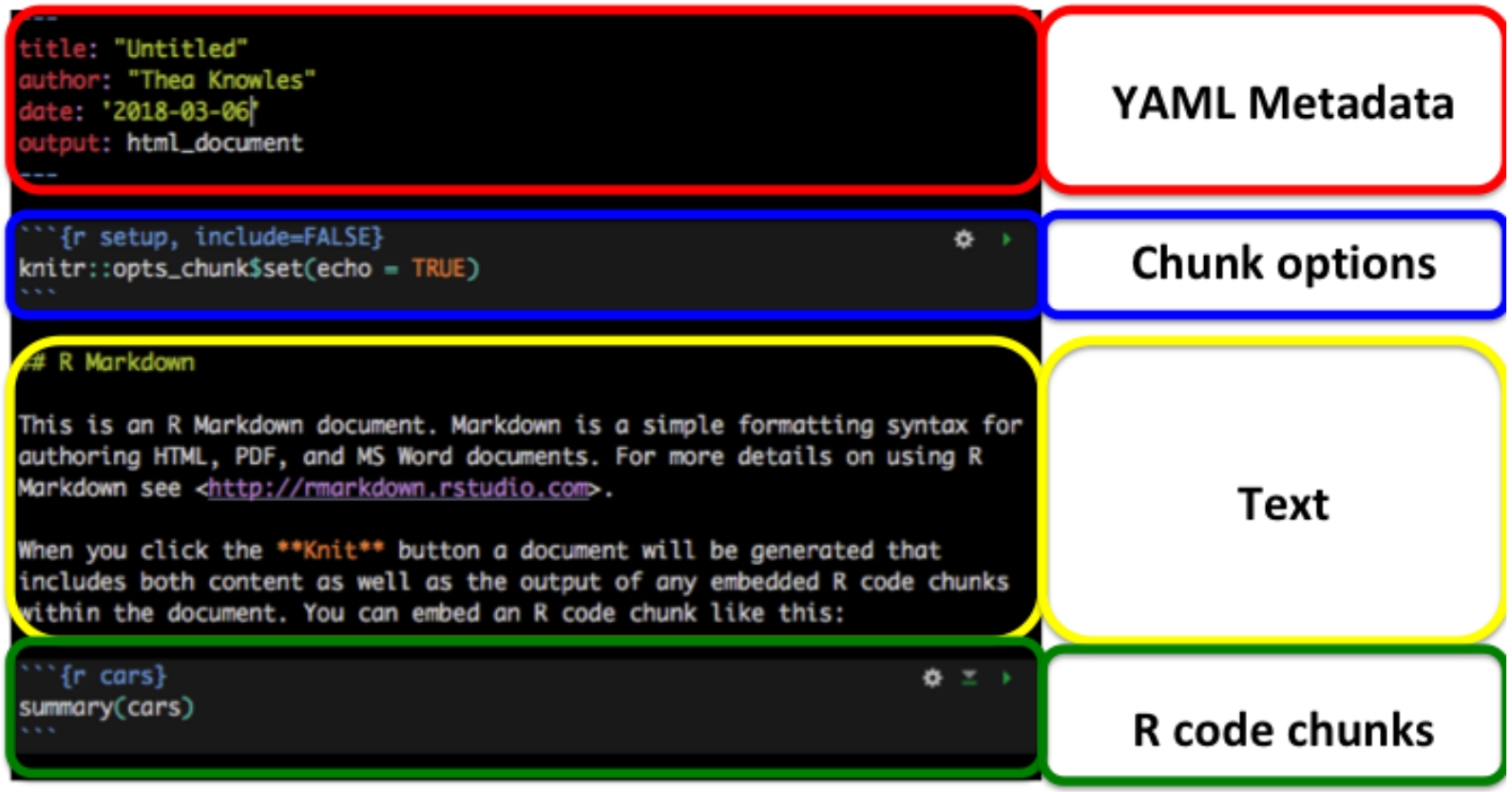
1. Ensure `simulated_vot_data.csv` is saved in your R Project folder
2. Add and run the following code to your R script:

```
# Load data ----
vot <- read.csv("simulated_vot_data.csv")

# Explore data ----
head(vot)
tail(vot)
str(vot)
names(vot)
summary(vot)
nrow(vot)
View(vot)
```

Rmarkdown

Essential parts of any R Markdown document



Essential parts of any R Markdown document

For today, we will only bother learning to edit text and code chunks.

We'll learn about editing the YAML and chunk options another day.

R Markdown: Text in Markdown syntax

Markdown

Markdown: set of conventions for editing plain text.

Write as you normally would in a text editor or word processor, but you signal text formatting with certain characters (next slide).

Markdown (which is distinguished from *markUP* language) is designed to be

- **easy to read**
- **easy to write**
- **easy to learn**

Markdown syntax

syntax

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
superscript^2^
~~strikethrough~~
[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi*r^{2}$
image: ![](path/to/smallorb.png)

horizontal rule (or slide break):

***
```

becomes

Plain text
End a line with two spaces to start a new paragraph.
italics and *italics*
bold and **bold**
superscript^2
~~strikethrough~~
link

Header 1

Header 2

Header 3

Header 4

Header 5

Header 6

endash: –
emdash: —
ellipsis: …
inline equation: $A = \pi * r^2$
image: 
horizontal rule (or slide break):

Markdown syntax

Tables written like this:

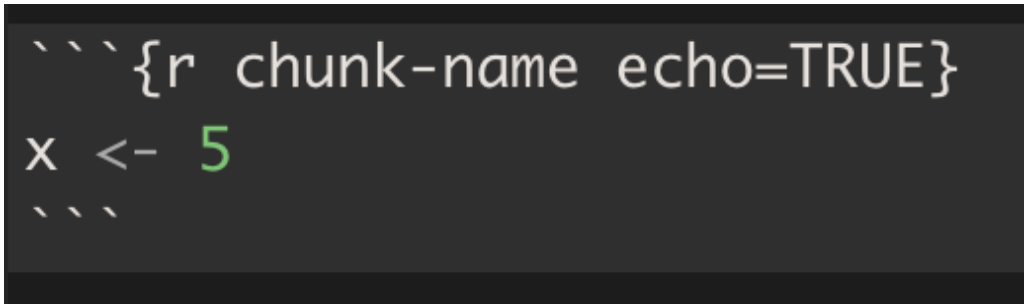
```
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

- See Tables Generator website
- *See `remedy` and `beautifyR` R packages/RStudio addins for dealing with markdown tables*
- Data frames can be turned into tables without manual modification
    - *See `kable` and `flextable` R packages*

R Markdown: R code in code chunks

Code chunks

- *Chunks* are sections that will include R code. By setting defaults at the beginning of your document, you can specify what you want most of your chunks to do.
- In each chunk, you can specify options in the form `tag=value` in the chunk header.
  - Example: `echo=TRUE`: code will print verbatim in the output.
  - **For now, don't worry about editing the header!**
- Within a chunk: regular R code!



Code chunks

Insert a new R chunk by

- Typing `Alt + Cmd/Ctrl + i` OR
- Clicking `Code >> Insert Chunk` from the R Studio menu

Inline R code

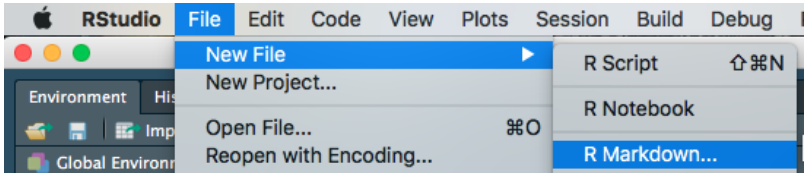
You can include R code as *inline R code*, embedded in to the text.

```
The mean VOT is `r mean(vot$VOT)`
```

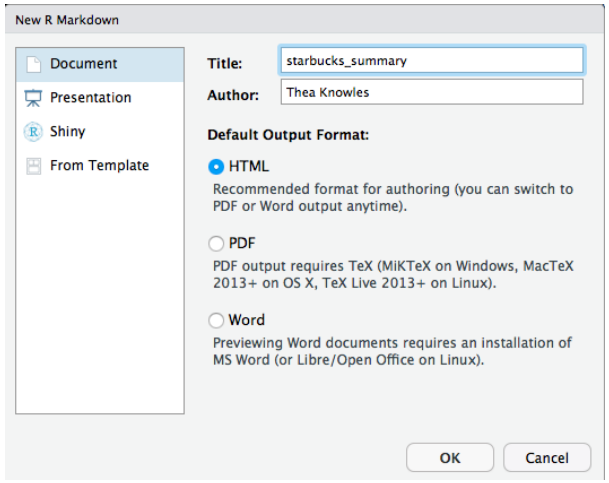
The value of `mean(vot$VOT)` will print in the output.

Exercise 4

Make a new R Markdown document!



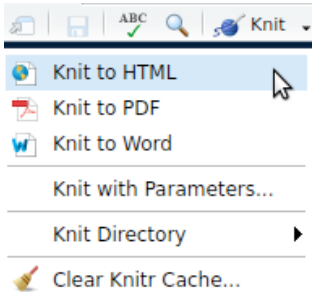
Title it something useful



Save it in your project folder (01_c4s_rmd.Rmd)

Exercise 5

1. Delete everything after Line 12.
2. Add a new header
3. Add some plain text
4. Add some **bold** text
5. Add a code chunk that includes 1x <- 5`and`x`
6. Compile! (“Knit”): Cmd/Ctrl + K or with Knit button



Exercise 6

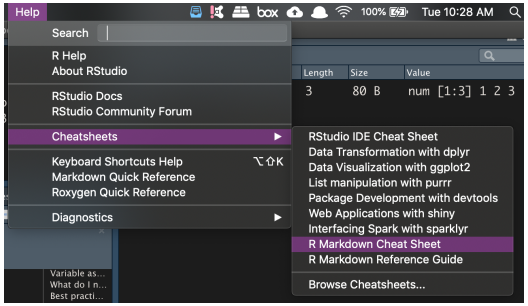
- Copy the code from your `helper.R` script to your R Markdown document.
- Compile again!
- Let's diagnose our errors!

Note: It is also possible to `source` an R script into an R Markdown document. This is much more helpful for manuscript writing… more on this another day!

Getting unstuck

Googling errors as art form

- Google your errors!
- Google “how to use XX package in r”
- Search Twitter (#rstats)



Read the documentation

- Type `??function` in the console
- Google the package name online
- R Studio cheat sheets

Schedule for the rest of the term

Date	Time	Location	Topic
2/25	4pm	Cary Hall 42	Intro to group + RStudio
3/10	4pm	Cary Hall 135	Intro to R + R Markdown
3/24	4pm	Cary Hall 135	Cleaning + manipulating data
4/7	4pm	Cary Hall 135	Data visualizations
4/21	4pm	Cary Hall 135	Praat?
5/5	4pm	Cary Hall 135	Praat?

Resources: Intro to R

- RStudio's recommended beginner R links: Lots of tutorials, suggestions, and cheat sheets!
- R for Data Science: THE book on learning R.
- Teacup Giraffes: Intro modules to statistics with R
- "R you with me?": A series of online learning resources for beginner R users designed by R-Ladies Sydney

Resources: RMarkdown

- Intro to RMarkdown: Video (12 minutes)
- RMarkdown: The definitive guide
- R for data science: R Markdown
- RMarkdown reference guide from RStudio
- RMarkdown Cheatsheet from RStudio