

NC STATE UNIVERSITY

R Basics, RMarkdown, & the Tidyverse

Justin Post & Matt Beckman

Workshop Intention

- Introduce data wrangling and visualization using packages from the tidyverse
 - `readr`, `dplyr`, `ggplot2`
- Create reproducible work using RMarkdown
- Be introduced to RShiny
 - Create interactive documents that run R on the backend
- Only 4 total hours! Not going in depth... but we will provide resources for continued learning!

Schedule

- RStudio IDE (Integrated Development Environment)
- Vectors and Data frames/Tibbles
- Importing CSV files with `readr`
- Markdown basics
- Common data manipulation with `dplyr`
- Plotting with `ggplot2`
- Shiny basics

RStudio IDE

In RStudio, four main 'areas'

- Console (& Terminal)
- Scripting and Viewing Window
- Plots/Help (& Files/Packages)
- Environment (& Connections/Git)

Console

- Type code directly into the **console** for evaluation

```
#simple math operations  
# <-- is a comment - code not evaluated  
3 + 7  
## [1] 10
```

```
mean(cars$speed)  
## [1] 15.4
```

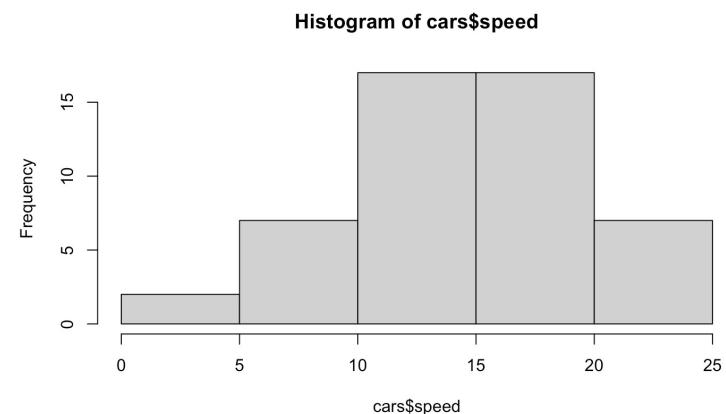
```
10 * exp(3) #exp is exponential function
```

```
## [1] 200.8554
```

```
log(pi^2) #log is natural log by default
```

```
## [1] 2.28946
```

```
hist(cars$speed)
```



Scripting and Viewing Window

- Usually want to keep code for later use!
- Write code in a ‘script’ and save script (or use markdown - covered later)
- From script can send code to console via:
 - “Run” button (runs current line)
 - CTRL+Enter (PC) or Command+Enter (MAC)
 - Highlight section and do above

Plots/Help

- Created plots stored in `Plots` tab
 - Cycle through past plots
 - Easily save
- Type `help(...)` into the console for documentation
 - `help(seq)`
 - `help(data.frame)`

Environment

- Store **data/info/function/etc.** in R objects
- Create an R object via `<-` (recommended) or `=`

```
#save for later
avg <- (5 + 7 + 6) / 3
#call avg object
avg
```

```
## [1] 6
```

```
#strings (text) can be saved as well
words <- c("Hello there!", "How are you?")
words
```

```
## [1] "Hello there!" "How are you?"
```

Environment

- Look at all current objects with `ls()`

```
ls()
```

```
## [1] "avg"    "words"
```

- `rm()` to remove

```
rm(avg)
```

```
ls()
```

```
## [1] "words"
```

```
#rm(list = ls()) cleans up environment
```

Environment

- Built-in objects exist like letters and cars

letters

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"  
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
head(cars, n = 3)
```

```
##   speed dist  
## 1     4    2  
## 2     4   10  
## 3     7    4
```

- `data()` shows available built-in datasets

RStudio IDE

Four main 'areas'

- Console (& Terminal)
- Scripting and Viewing Window
- Plots/Help (& Files/Packages)
- Environment (& Connections/Git)

Quick Examples

- Go to file -> New File -> R Script
- Type `View(cars)` (note capital V)
- Type `plot(cars)`
- Type `hist(iris$Sepal.Length)`
- Submit these to console using button or hot key
- Use the Help tab to learn about R functions
 - Type `help(hist)` in the console

R Objects and Structure

- Create an R object via `<-` (recommended) or `=`
 - allocates memory to object

```
vec <- c(1, 4, 10)  
vec
```

```
## [1] 1 4 10
```

R Objects and Structure

- Create an R object via `<-` (recommended) or `=`
 - allocates memory to object

```
fit <- lm(dist ~ speed, data = cars)
fit

## 
## Call:
## lm(formula = dist ~ speed, data = cars)
## 
## Coefficients:
## (Intercept)      speed
##       -17.579     3.932
```

Investigating Objects

Many functions to help understand an R Object

- `str()`
- compactly displays the internal structure of an R object

```
str(cars)
```

```
## '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 ...
```

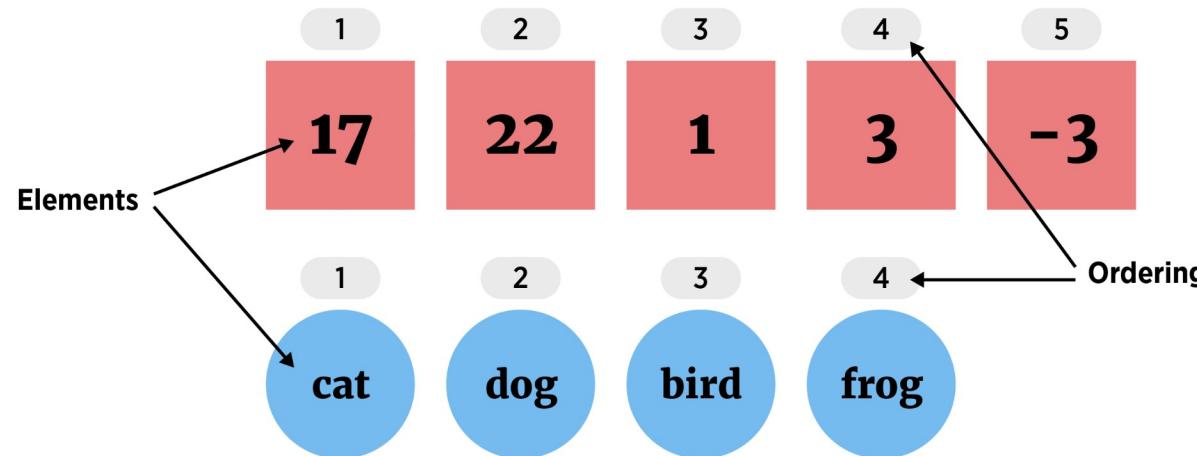
Data Objects

- Understand data structures first: Five major types
 - Atomic Vector (1d)
 - Matrix (2d)
 - Array (nd)
 - Data Frame (2d)
 - List (1d)

Dimension	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data Frame

Vector

1. Atomic Vector (1D group of elements with an ordering)



- Elements must be same 'type'
 - numeric (integer or double), character, or logical

Vector

1. Atomic Vector (1D group of elements with an ordering)

- Create with `c()` function ('combine')

```
#vectors (1 dimensional) objects
x <- c(17, 22, 1, 3, -3)
y <- c("cat", "dog", "bird", "frog")
x
```

```
## [1] 17 22  1   3  -3
```

```
y
```

```
## [1] "cat"  "dog"  "bird" "frog"
```

Vector

- Many functions output a numeric vector

```
v <- seq(from = 1, to = 10, by = 2)
```

```
v
```

```
## [1] 1 3 5 7 9
```

```
1:10
```

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

Vector

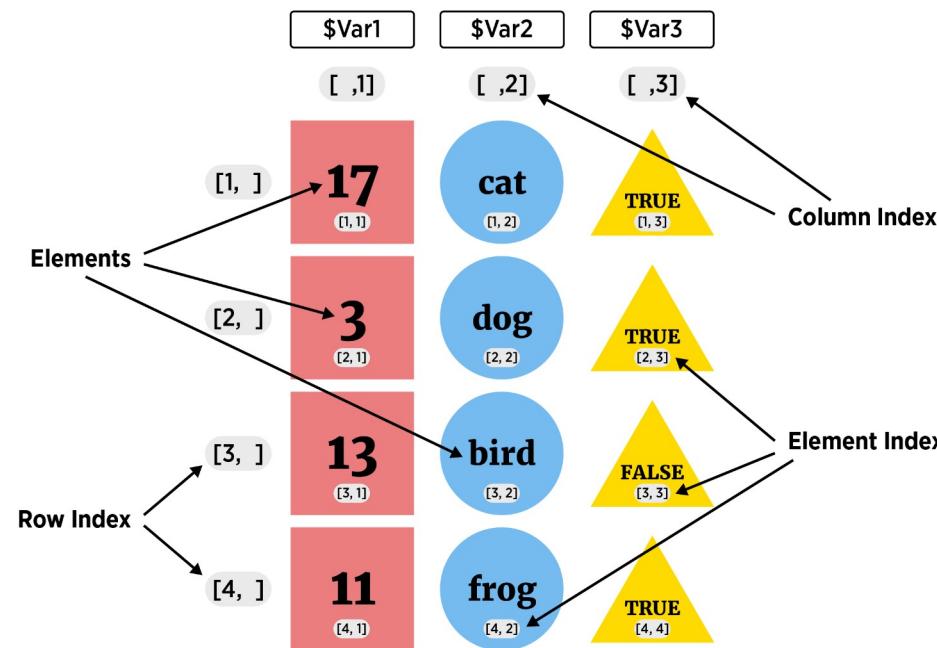
1. Atomic Vector (1D group of elements with an ordering)

- Vectors useful to know about
- Not usually useful for a dataset
- Often consider as 'building blocks' for other data types

Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same length



Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**
- Create with `data.frame()` function

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
data.frame(x, y, z)
```

```
##   x   y   z
## 1 a   1 10
## 2 b   3 11
## 3 c   4 12
## 4 d  -1 13
## 5 e   5 14
## 6 f   6 15
```

Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**
- Create with `data.frame()` function

```
data.frame(char = x, data1 = y, data2 = z)
```

```
##   char data1 data2
## 1   a     1    10
## 2   b     3    11
## 3   c     4    12
## 4   d    -1    13
## 5   e     5    14
## 6   f     6    15
```

- `char`, `data1`, and `data2` become the variable names for the data frame

Data Frame

4. Data Frame (2D data structure)

- collection (list) of *vectors* of the same **length**
- Create with `data.frame()` function
- Perfect for most data sets!
- Most functions that read 2D data store it as a data frame
- Tibbles are special data frames used in the `tidyverse`

Exercises

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_1-2_Exercises.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_1-2_Exercises.html)

Schedule

- RStudio IDE (Integrated Development Environment)
- Vectors and Data Frames/Tibbles
- Importing CSV files with `readr`
- Markdown basics
- Common data manipulation with `dplyr`
- Plotting with `ggplot2`
- Shiny basics

Importing Data

How to read in data depends on raw/external data type!

- Delimited data
 - Delimiter - Character (such as a ,) that separates data entries

Treatment,Sex,Age,Duration,Pain
P,F,68,1,No
B,M,74,16,No
P,F,67,30,No
P,M,66,26,Yes
B,F,67,28,No
B,F,77,16,No
A,F,71,12,No
B,F,72,50,No
B,F,76,9,Yes
A M 71 17 Vac

Comma: usually .csv

temp conc time percent
-1 -1 -1 45.9
1 -1 -1 60.6
-1 1 -1 57.5
1 1 -1 58.6
-1 -1 1 53.3
1 -1 1 58
-1 1 1 58.8

Space: usually .txt or .dat

"color"	"spine"	"width"	"satell"	"weight"	"y"
3	3	28.3	8	3050	1
4	3	22.5	0	1550	0
2	1	26	9	2300	1
4	3	24.8	0	2100	0
4	3	26	4	2600	1
3	3	23.8	0	2100	0
2	1	26.5	0	2350	0
4	2	24.7	0	1900	0
3	1	23.7	0	1950	0
4	3	25.6	0	2150	0
4	3	24.3	0	2150	0
3	3	25.8	0	2650	0
3	3	28.2	11	3050	1
5	2	21	0	1850	0
3	1	26	14	2300	1
2	1	27.1	8	2950	1
3	3	25.2	1	2000	1
3	3	29	1	3000	1
5	2	21.7	0	2200	0

Tab: usually .tsv or .txt

2012>4>12>MIN>LAA>D.J. Reyburn
2012>4>12>SD>ARI>Marty Foster
2012>4>12>WSH>CIN>Mike Everitt
2012>4>12>PHI>MIA>Jeff Nelson
2012>4>12>CHC>MIL>Fieldin Culbreth
2012>4>12>LAD>PIT>Wally Bell
2012>4>12>TEX>SEA>Doug Eddings
2012>4>12>COL>SF>Ron Kulpa
2012>4>12>DET>TB>Mark Carlson
2012>4>13>NYY>LAA>Mike DiMuro
2012>4>13>COL>ART>Mark Wagner

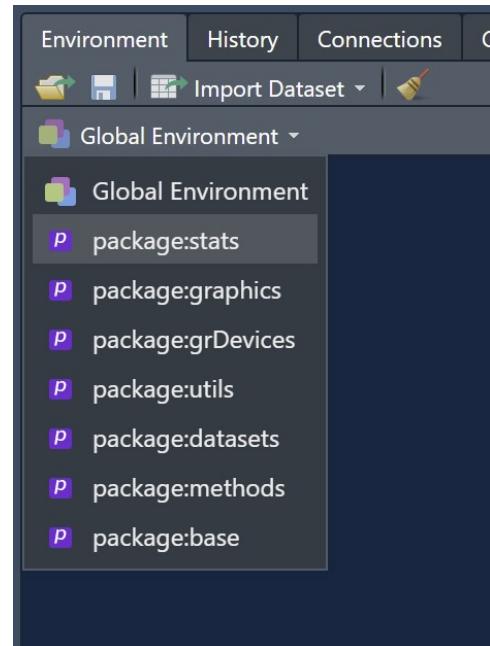
General: usually .txt or .dat

Importing Delimited Data

- When you open R a few packages are loaded
- R package
 - Collection of functions/datasets/etc. in one place
 - Packages exist to do almost anything
 - [List of CRAN](#) approved packages on R's website
 - Plenty of other packages on places like GitHub

Importing Delimited Data

- When you open R a few packages are loaded



- utils package has *family* of `read.` functions ready for use!

Reading Delimited Data

- Functions from `read.` family work well
- Concerns:
 - poor default function behavior
 - (formerly, prior to R 4.0) strings are read as factors
 - row & column names can be troublesome
 - (Slightly) different behavior on different computers
 - Want to have most of our functions we use 'feel' the same...

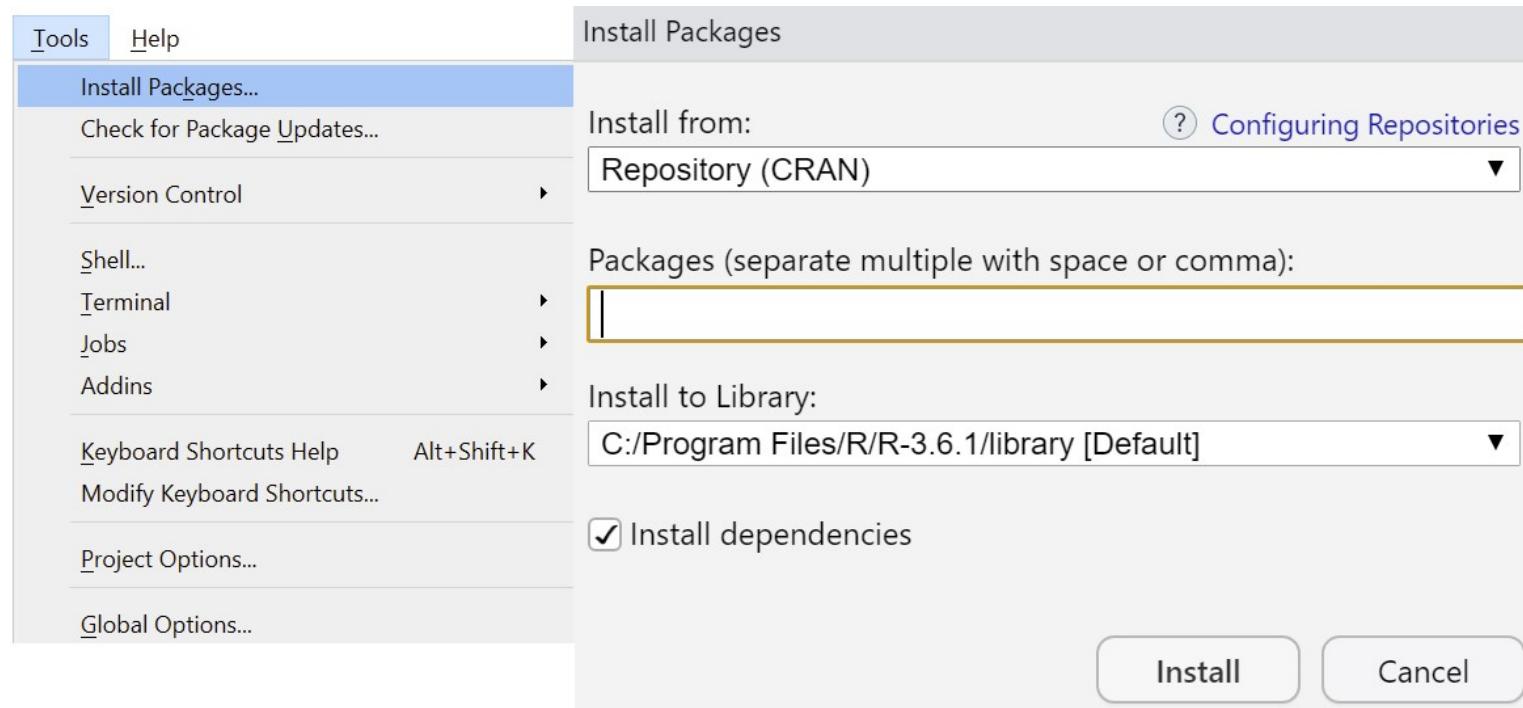
Aside: R Packages

- R package
 - Collection of functions in one place
 - Packages exist to do almost anything
 - [List of CRAN](#) approved packages on R's website
 - Plenty of other packages on places like GitHub
- “[TidyVerse](#)” - collection of R packages that share common philosophies and are designed to work together!

Aside: R Packages

- First time using a package
 - Must install package (download files)
 - Can use code or menus

```
install.packages("readr")
```



Aside: R Packages

- Only install once!
- **Each session:** read in package using `library()` or `require()`

```
library("readr")
```

Aside: R Packages

- Hopefully you've already installed the `tidyverse` package via:

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

Aside: R Packages

- Hopefully you've already installed the `tidyverse` package via:

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

- Now you can load the library

```
library(tidyverse)
```

- Once library loaded, check `help(filter)`

Aside: R Packages

- Can call functions without loading full library with `::`
- If not specified, most recently loaded package takes precedent

```
#stats::filter(...) calls time-series function from stats package
dplyr::filter(iris, Species == "virginica")
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3        3.3         6.0        2.5 virginica
## 2          5.8        2.7         5.1        1.9 virginica
## 3          7.1        3.0         5.9        2.1 virginica
## 4          6.3        2.9         5.6        1.8 virginica
## 5          6.5        3.0         5.8        2.2 virginica
## 6          7.6        3.0         6.6        2.1 virginica
## 7          4.9        2.5         4.5        1.7 virginica
## 8          7.3        2.9         6.3        1.8 virginica
## 9          6.7        2.5         5.8        1.8 virginica
## 10         7.2        3.6         6.1        2.5 virginica
## 11         6.5        3.2         5.1        2.0 virginica
## 12         6.4        2.7         5.3        1.9 virginica
## 13         6.8        3.0         5.5        2.1 virginica
## 14         5.7        2.5         5.0        2.0 virginica
## 15         5.8        2.8         5.1        2.4 virginica
## 16         6.4        3.2         5.3        2.3 virginica
```

Reading Delimited Data

baseR and tidyverse (readr package does the heavy lifting) function and purpose:

Type of Delimiter	utils Function	readr Function
Comma	read.csv()	read_csv()
Semicolon (, for decimal)	read.csv2()	read_csv2()
Tab	read.delim()	read_tsv()
General	read.table(sep = "")	read_delim()
White Space	read.table(sep = "")	read_table() read_table2()

Reading a .csv File

- Let's read in the '[neuralgia.csv](#)' file
- By default, R looks in the working directory for the file

```
getwd()
```

```
## [1] "/Users/mattbeckman/Documents/GitHub/presentations/2021/TeachingWithR/Presentation"
```

```
#change with setwd()  
#better to use R projects!
```

Reading a .csv File

```
neuralgiaData <- read_csv("neuralgia.csv")  
neuralgiaData
```

Reading a .csv File

- Let's read in the '[neuralgia.csv](#)' file
- R can pull from URLs as well!

```
neuralgiaData <- read_csv("https://www4.stat.ncsu.edu/~online/datasets/neuralgia.csv")
neuralgiaData

## # A tibble: 60 x 5
##   Treatment Sex     Age Duration Pain
##   <chr>     <chr> <dbl>    <dbl> <chr>
## 1 P          F        68        1  No
## 2 B          M        74       16  No
## 3 P          F        67       30  No
## 4 P          M        66       26  Yes
## 5 B          F        67       28  No
## 6 B          F        77       16  No
## 7 A          F        71       12  No
## 8 B          F        72       50  No
## 9 B          F        76        9  Yes
## 10 A         M        71       17  Yes
## # ... with 50 more rows
```

Reading a .csv File

`read_csv()` function

- Other useful inputs:
 - `skip = 0`
 - `col_names = TRUE`
 - `na = c("", "NA")`

tibbles

- Behavior slightly different than a standard data frame
- Use either `pull()` or `$` to return a column as a vector

```
pull(neuralgiaData, 1) #or pull(neuralgiaData, Treatment)
```

```
## [1] "P" "B" "P" "P" "B" "B" "A" "B" "B" "A" "A" "A" "B" "A" "P" "A" "P" "A" "P"  
## [20] "B" "B" "A" "A" "B" "P" "B" "B" "P" "P" "A" "A" "B" "B" "B" "B" "A" "P" "B"  
## [39] "B" "P" "P" "A" "B" "A" "P" "P" "A" "B" "P" "P" "P" "B" "A" "P" "A" "P"  
## [58] "A" "B" "A"
```

```
neuralgiaData$Treatment
```

```
## [1] "P" "B" "P" "P" "B" "B" "A" "B" "B" "A" "A" "A" "B" "A" "P" "A" "P" "A" "P"  
## [20] "B" "B" "A" "A" "B" "P" "B" "B" "P" "P" "A" "A" "B" "B" "B" "B" "A" "P" "B"  
## [39] "B" "P" "P" "A" "B" "A" "P" "P" "A" "B" "P" "P" "P" "B" "A" "P" "A" "P"  
## [58] "A" "B" "A"
```

Exercises

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_1-2_Exercises.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_1-2_Exercises.html)

Partial Solution

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_1-2_Solutions.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_1-2_Solutions.html)

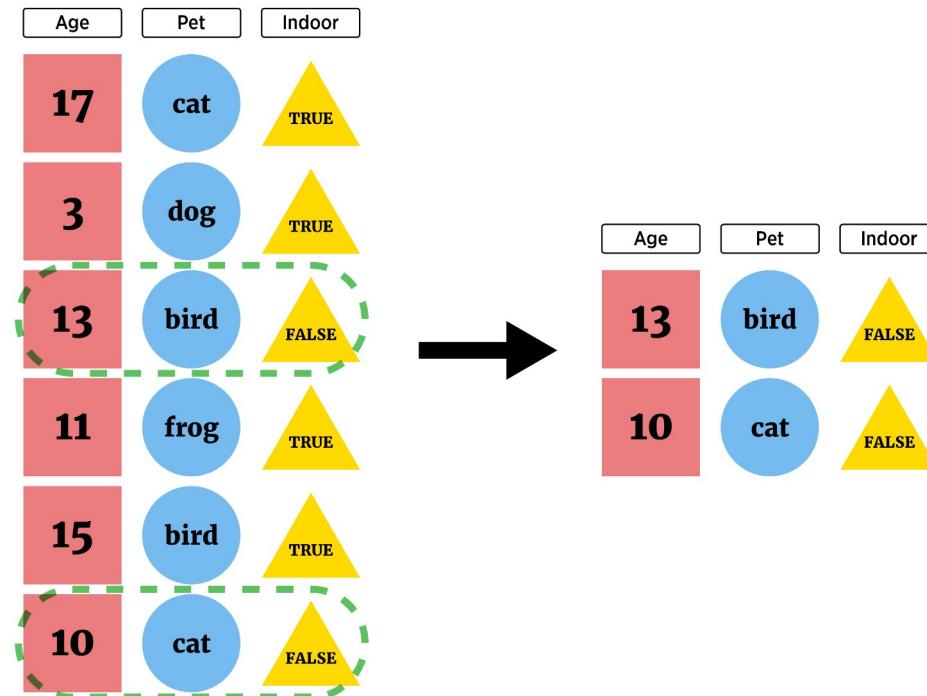
Schedule

- RStudio IDE (Integrated Development Environment)
- Vectors and Data Frames/Tibbles
- Importing CSV files with `readr`
- Markdown basics
- Common data manipulation with `dplyr`
- Plotting with `ggplot2`
- Shiny basics

Data manipulation idea

We may want to subset our full data set or create new data

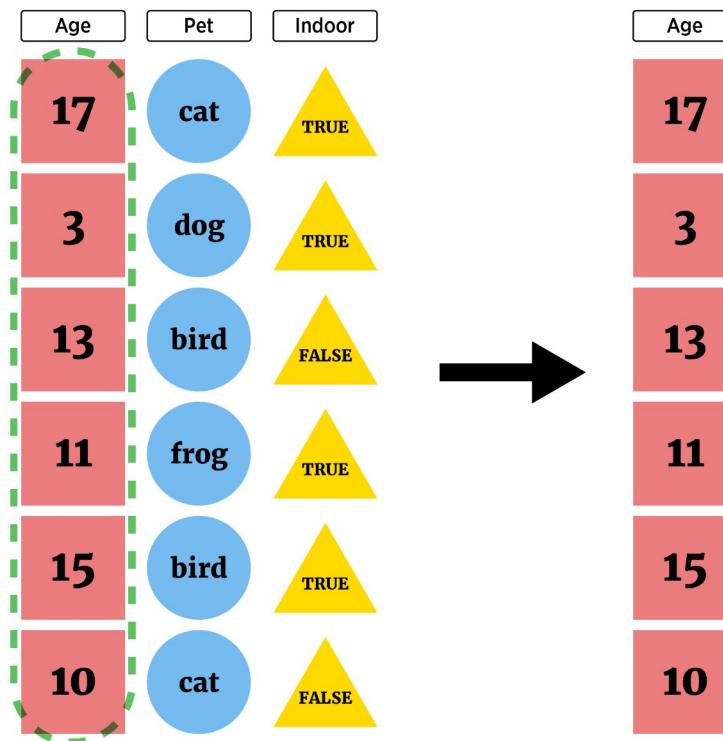
- Grab only certain types of observations (**filter rows**)



Data manipulation idea

We may want to subset our full data set or create new data

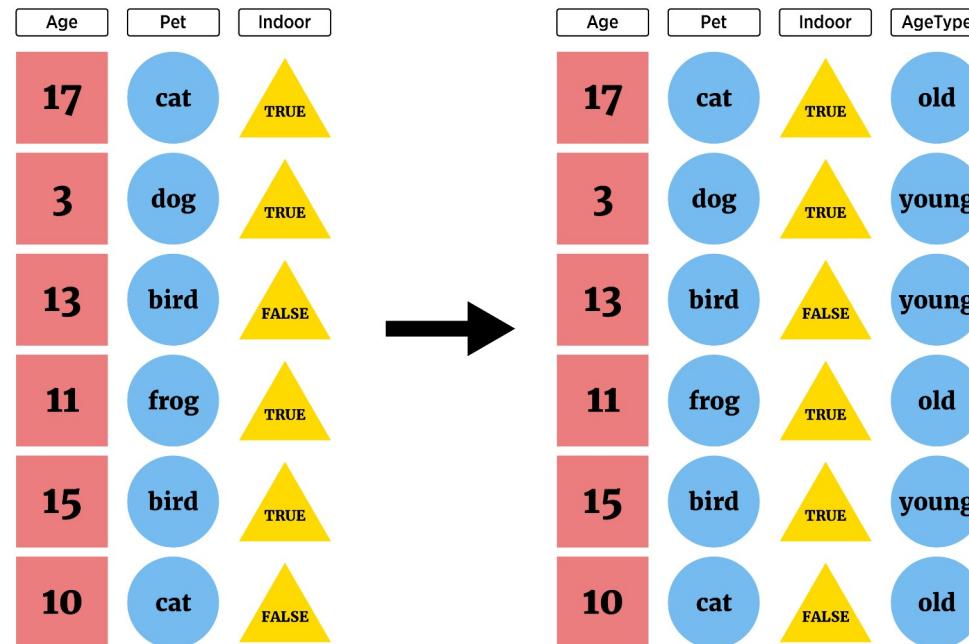
- Look at only certain variables (**select columns**)



Data manipulation idea

We may want to subset our full data set or create new data

- Create new variables



Data manipulation idea

We may want to subset our full data set or create new data

- Communication and reproducibility vital!
- Traditional documentation through comments (# in R) in script
- May have heard of [JUPYTER](#) notebooks
- R Markdown - built in notebook for RStudio

Documenting with Markdown

- R Markdown = Digital “Notebook”: Program that weaves word processing and code.
- Designed to be used in three ways (R for Data Science)

Documenting with Markdown

- R Markdown = Digital “Notebook”: Program that weaves word processing and code.
- Designed to be used in three ways (R for Data Science)
 - Communicating to decision makers (focus on conclusions not code)
 - Collaborating with other data scientists (including future you!)
 - As environment to do data science (documents what you did and what you were thinking)

Documenting with Markdown

Verbage

- May have heard of HTML (HyperText Mark-up Language)
 - Write plain text that the browser interprets and renders

Documenting with Markdown

Verbage

- May have heard of HTML (HyperText Mark-up Language)
 - Write plain text that the browser interprets and renders
- Markdown is a specific markup language
 - Easier syntax
 - Not as powerful
- White space is important! (May want to show white space characters.)
- Any plain text file can be used (.Rmd extension associates it with RStudio)

Documenting with Markdown

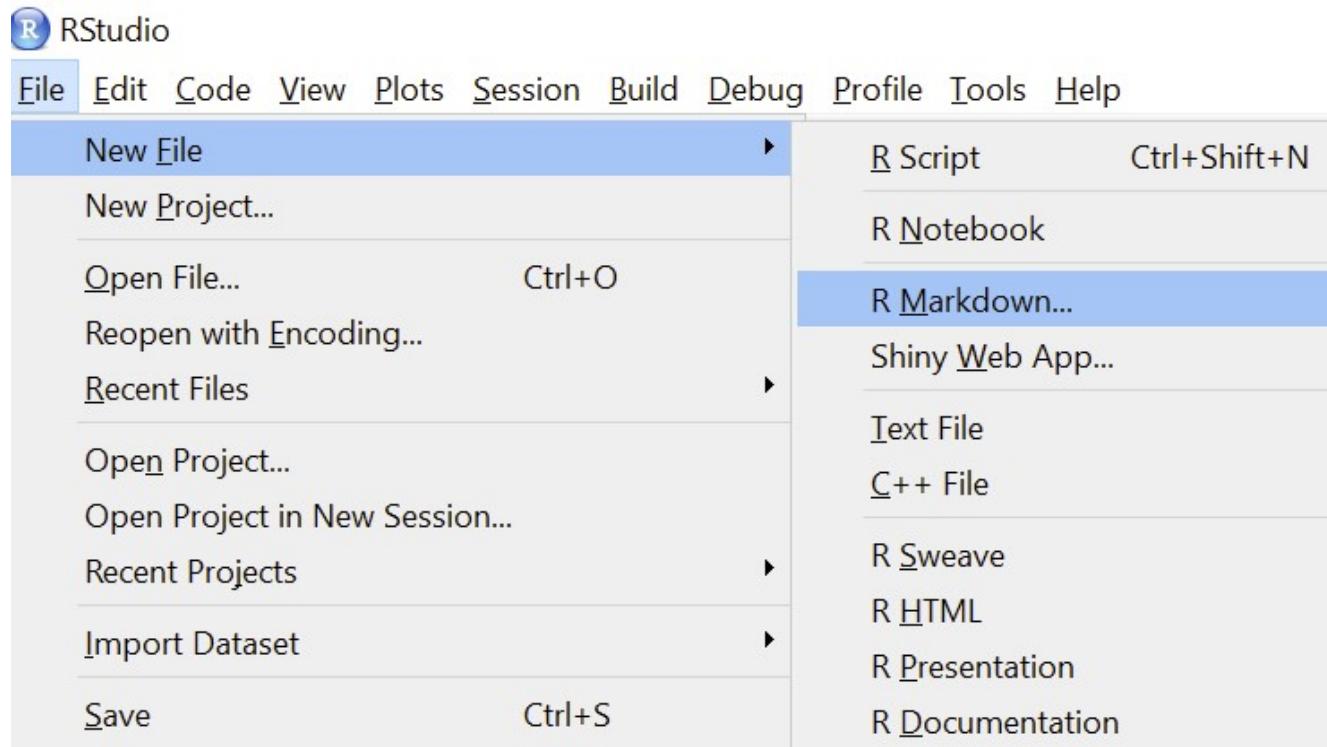
R Markdown file contains three important types of content:

1. (Optional) YAML header surrounded by ---s
2. Chunks of R code
3. Text mixed with simple text formatting instructions

Documenting with Markdown

Creating an R Markdown Document

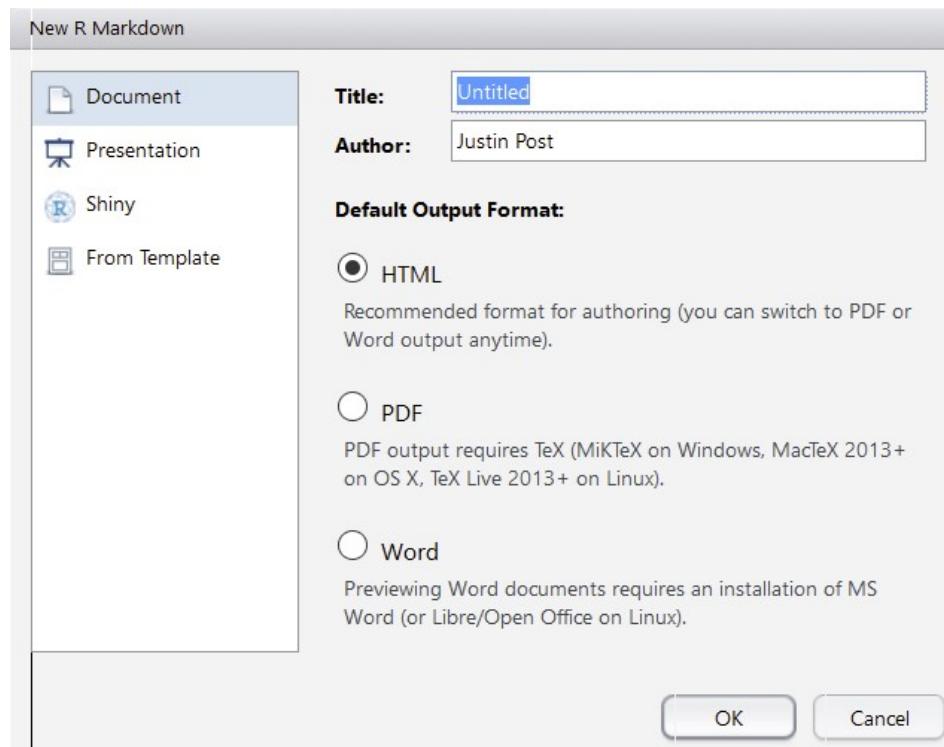
- RStudio makes it easy!



Documenting with Markdown

Creating an R Markdown Document

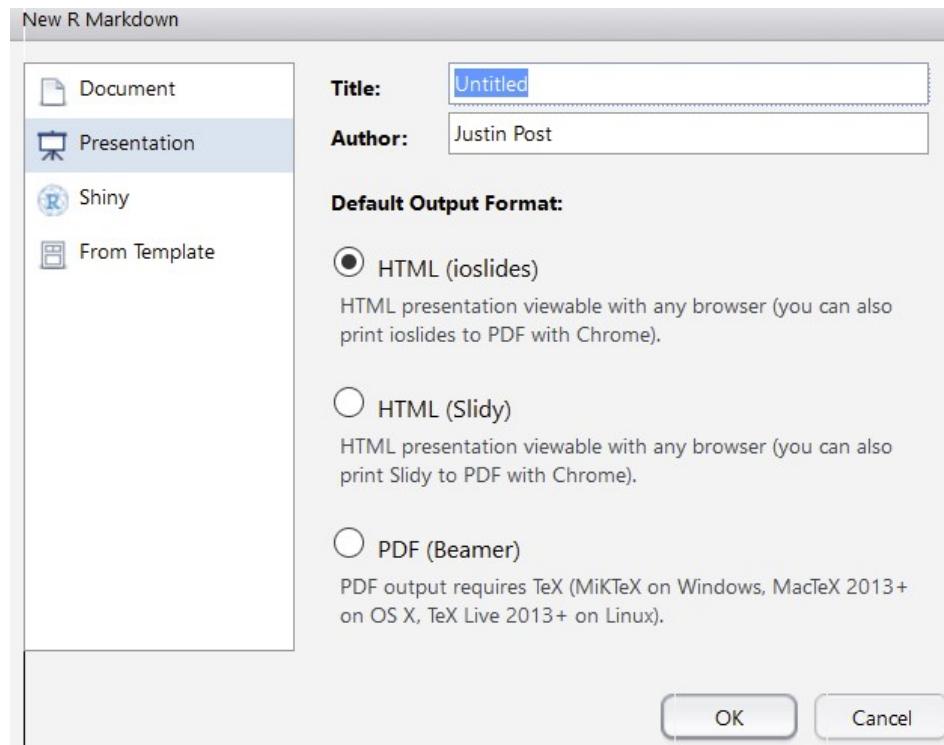
- Commonly used document types can be created



Documenting with Markdown

Creating an R Markdown Document

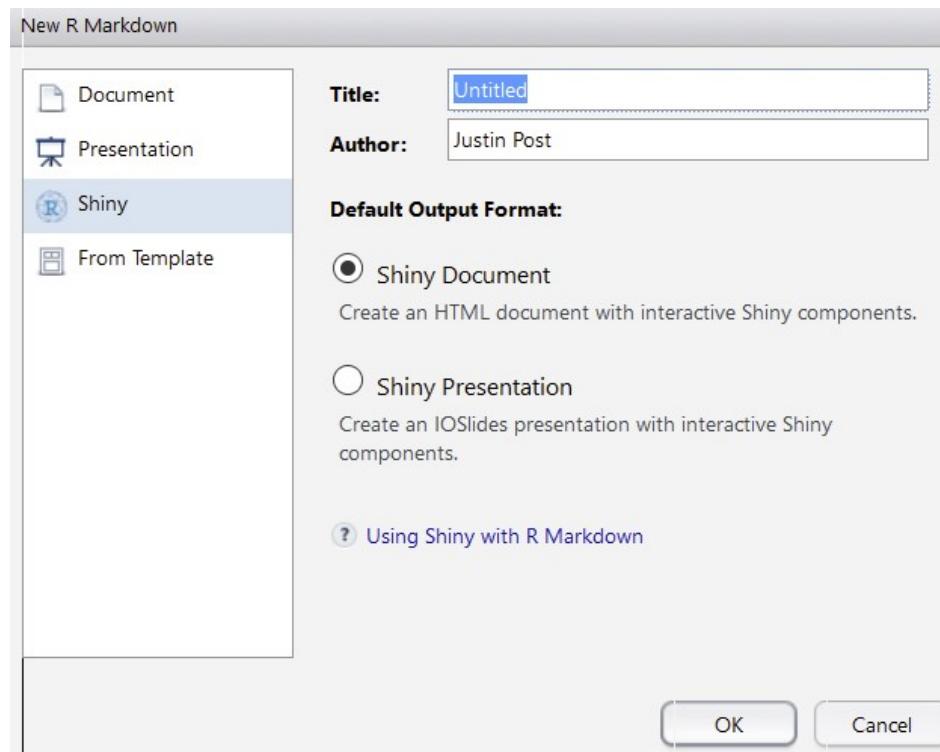
- Slide presentations



Documenting with Markdown

Creating an R Markdown Document

- Truly Interactive Documents/Pages (require R backend)



Documenting with Markdown

YAML Header

- Define settings for document

```
title: "Untitled"
author: "First Last"
date: "xxxx"
output: html_document
```

- CTRL/CMD + Shift + k knits via this info

Documenting with Markdown

Code Chunks

- Below YAML header: 'r chunk'

```
```{r ggplot, eval=FALSE}
select(iris, Sepal.Width)
ggplot(iris, aes(x = Sepal.Width, y = Sepal.Length)) +
geom_point()
```
```

- Start code chunk by typing ```{r} out or with CTRL/CMD + Alt/Option + I
- Code will be executed when document is created
- Can specify options on individual code chunks

Documenting with Markdown

Markdown Syntax

- Below code chunk is plain text with markdown syntax

```
## 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.

- When file created, “##” becomes a header, “<...>” a link, and **Knit** bold font

Documenting with Markdown

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.

Where do we go from here?

Briefly investigate:

- Markdown syntax
- Code chunks and their options
- Changing type of output

Documenting with Markdown

Markdown syntax

- [Cheat Sheet link] (<https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>) becomes [Cheat Sheet link](#)

Documenting with Markdown

Markdown syntax

- [Cheat Sheet link] (<https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>) becomes [Cheat Sheet link](#)
- # Header 1 becomes a large font header
- ## Header 2 becomes a slightly smaller font header
- Goes to 6 headers
 - Use of headers can automatically create a Table of Contents!

Documenting with Markdown

Markdown syntax

- [Cheat Sheet link] (<https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>) becomes [Cheat Sheet link](#)
- # Header 1 becomes a large font header
- ## Header 2 becomes a slightly smaller font header
- Goes to 6 headers
 - Use of headers can automatically create a Table of Contents!
- **bold** and __bold__
- `code` becomes code

Documenting with Markdown

Markdown syntax

- Can do lists:
 - Indent sub lists four spaces
 - * unordered list
 - * item 2
 - + sub-item 1
 - + sub-item 2
 - item 2
 - sub-item 1
 - sub-item 2
 - 1. ordered list
 - 2. item 2
 - + sub-item 1
 - + sub-item 2

Documenting with Markdown

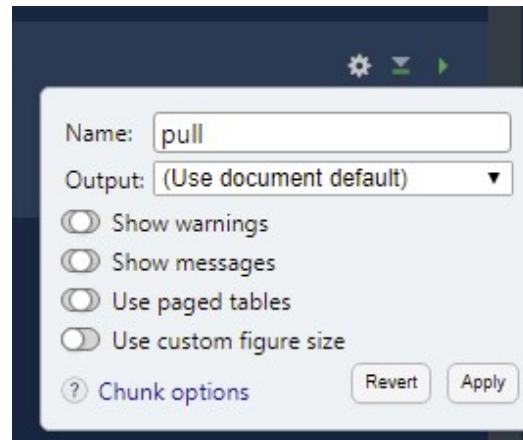
Code chunks and their options

- Any R code can go into the chunk
- Chunks evaluate sequentially (can use output from prior chunk)
- Code can be added in line: Ex: The Iris dataset has 150 observations
- Added by beginning with back-tick `r` and ending with a back-tick: Iris has `r
`length(iris$Sepal.Length)``

Documenting with Markdown

Code chunks and their options

- Can hide/show code with `echo = FALSE/TRUE`
- Can choose if code is evaluated with `eval = TRUE/FALSE`
- `message = TRUE/FALSE` and `warning = TRUE/FALSE` can turn on/off displaying messages/warnings



Documenting with Markdown

Code chunks and their options

- Many options depending on chunk purpose!
- Can hide/show code with `echo = FALSE/TRUE`
- Can choose if code is evaluated with `eval = TRUE/FALSE`
- `message = TRUE/FALSE` and `warning = TRUE/FALSE` can turn on/off displaying messages/warnings
- Can set global options for all chunks

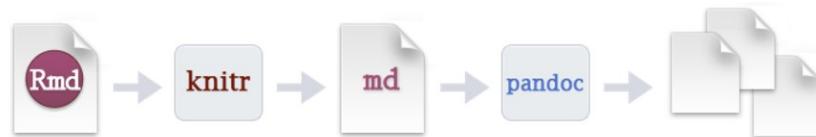
```
opts_chunk$set(echo = FALSE, eval = TRUE, warning = FALSE)
```

- Allows for easy change of audience!

Documenting with Markdown

Changing type of output

R Markdown really flexible!

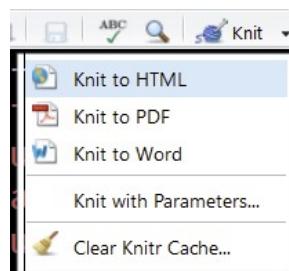


Documenting with Markdown

Changing type of output

Change output type in the YAML header:

- Use CTRL/CMD + Shift + k or the Knit menu:



- Use code explicity:

```
rmarkdown::render("file.Rmd", output_format = "word_document")
```

Documenting with Markdown

Changing type of output

For HTML & PDF can include Table of Contents with options

```
output:  
  html_document:  
    toc: true  
    toc_float: true
```

Documenting with Markdown

Changing type of output

For HTML & PDF can include Table of Contents with options

```
output:  
  html_document:  
    toc: true  
    toc_float: true
```

For HTML another option is to make the code chunks hidden by default, but visible with a click:

```
output:  
  html_document:  
    code_folding: hide
```

Documenting with Markdown

Changing type of output

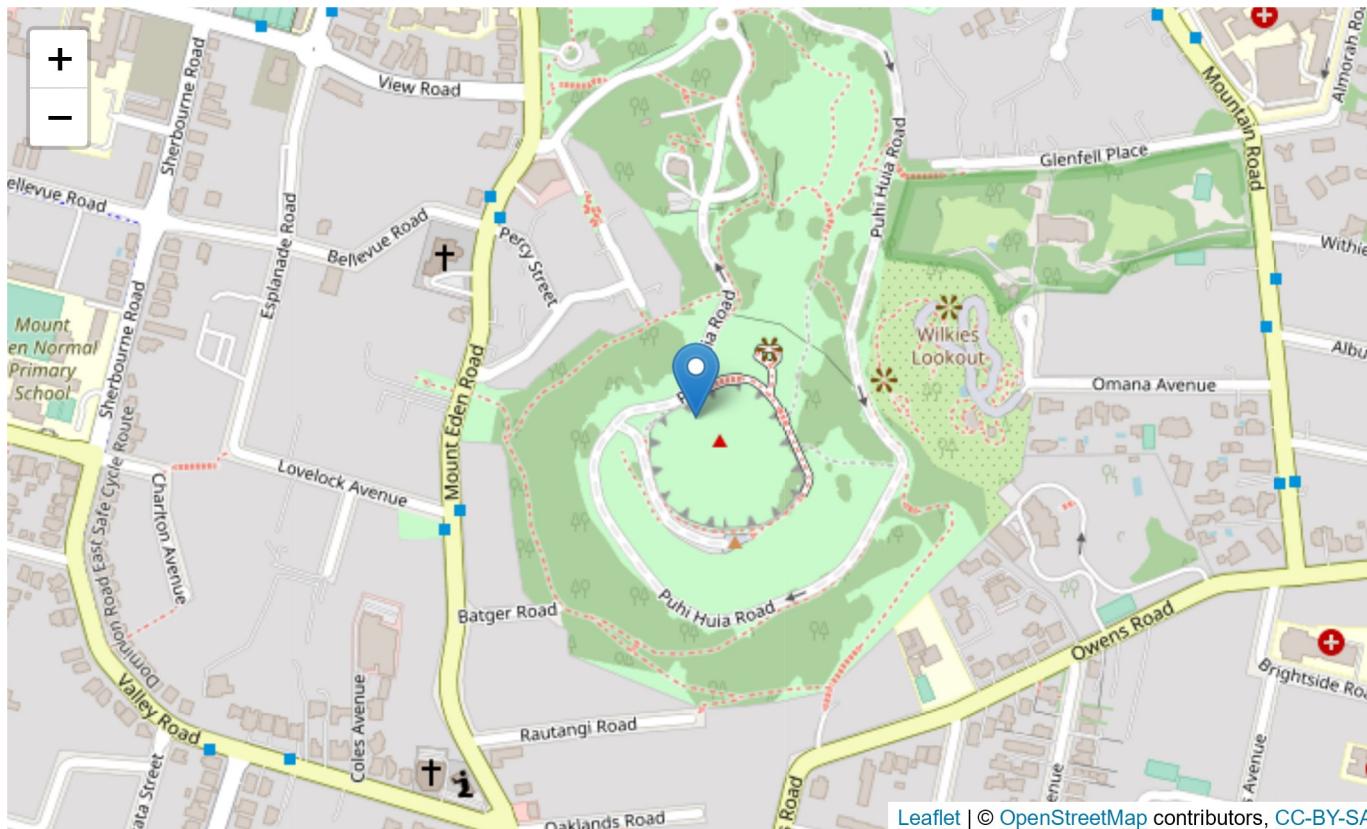
HTML documents inherently interactive

- Widgets can be included with appropriate R package

```
library(leaflet)
leaflet() %>%
  setView(174.764, -36.877, zoom = 16) %>%
  addTiles() %>%
  addMarkers(174.764, -36.877, popup = "Maungawhau")
```

Documenting with Markdown

Changing type of output



Documenting with Markdown

Changing type of output

- PDF
 - Install MikTex and update its packages or install a smaller version using the `tinytex` package: `tinytex::install_tinytex()`

```
output: pdf_document
```

Documenting with Markdown

Changing type of output

- PDF
 - Install MikTex and update its packages or install a smaller version using the `tinytex` package: `tinytex::install_tinytex()`

```
output: pdf_document
```

- Word

```
output: word_document
```

Documenting with Markdown

Changing type of output

- PDF
 - Install MikTex and update its packages or install a smaller version using the `tinytex` package: `tinytex::install_tinytex()`

```
output: pdf_document
```

- Word

```
output: word_document
```

- Slides (## for new slide)

```
output: ioslidespresentation
```

Exercises

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_3_Exercises.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_3_Exercises.html)

Partial Solution

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_3_Solutions.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_3_Solutions.html)

tidyverse for data manipulations

Now we can document everything: let's manipulate some data!

Overview of dplyr and tidyr packages

- `dplyr` package made for most standard data manipulation tasks
- `tidyr` package reshapes data
- Both part of `tidyverse`
- Make sure `library(tidyverse)` has been run!

Tidyverse Syntax

- Reason to prefer `dplyr` and packages from the `tidyverse`
- Good defaults
- All packages have similar syntax! All work on `tibbles` (data frames)
- Syntax: `function(tibble, actions, ...)`

dplyr

- Basic commands
 - `as_tibble()` - convert data frame to one with better printing
 - `filter()` - subset **rows**
 - `arrange()` - reorder **rows**
 - `select()` - subset **columns**
 - `rename()` - rename **columns**
 - `mutate()` - add newly created **column**
 - `transmute()` - create new variable
 - `group_by()` - group **rows** by a variable
 - `summarise()` - apply basic function to data

as_tibble() - tidy data frame

as_tibble() - convert data frame to one with better printing and no simplification

```
#install.packages("Lahman")
library(Lahman)
head(Batting, n = 4) #look at just first 4 observations

##      playerID yearID stint teamID lgID   G   AB   R   H  X2B  X3B  HR RBI SB CS BB SO
## 1 abercda01  1871     1    TRO  NA  1   4   0   0   0   0   0   0   0   0   0   0   0   0
## 2 addybo01  1871     1    RC1  NA 25 118  30  32   6   0   0  13   8   1   4   0
## 3 allisar01  1871     1    CL1  NA 29 137  28  40   4   5   0  19   3   1   2   5
## 4 allisdo01  1871     1    WS3  NA 27 133  28  44  10   2   2  27   1   1   0   2
##      IBB HBP SH SF GIDP
## 1   NA  NA NA NA     0
## 2   NA  NA NA NA     0
## 3   NA  NA NA NA     1
## 4   NA  NA NA NA     0
```

`as_tibble()` - tidy data frame

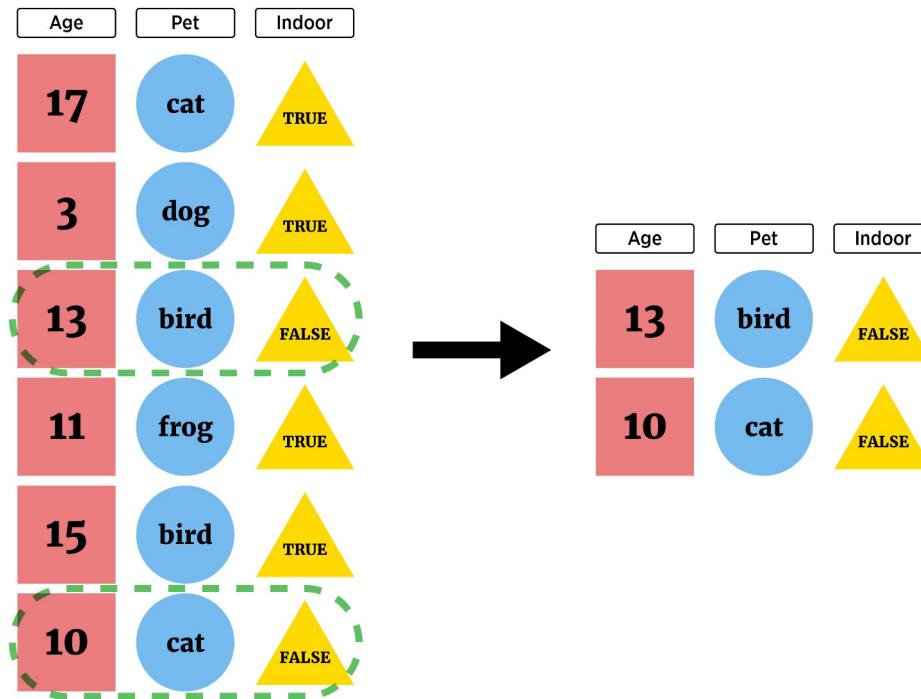
- Just 'wrap' a standard R data frame

```
myBatting <- as_tibble(Batting); myBatting
```

```
## # A tibble: 107,429 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>     <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 abercda... 1871     1 TRO    NA     1     4     0     0     0     0     0     0
## 2 addybo01  1871     1 RC1    NA    25    118    30    32     6     0     0
## 3 allisar... 1871     1 CL1    NA    29    137    28    40     4     5     0
## 4 allisdo... 1871     1 WS3    NA    27    133    28    44    10     2     2
## 5 ansonca... 1871     1 RC1    NA    25    120    29    39    11     3     0
## 6 armstbo... 1871     1 FW1    NA    12    49     9    11     2     1     0
## 7 barkeal... 1871     1 RC1    NA     1     4     0     1     0     0     0
## 8 barnero... 1871     1 BS1    NA    31    157    66    63    10     9     0
## 9 barrebi... 1871     1 FW1    NA     1     5     1     1     1     0     0
## 10 barrofr... 1871    1 BS1    NA    18    86    13    13     2     1     0
## # ... with 107,419 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

filter() - subset rows or columns

- Grab only certain types of observations (**filter rows**)



Logical statements

Goal: Subset rows or columns

- **logical statement** - comparison that resolves as TRUE or FALSE

```
"hi" == " hi" #== is comparison          4 != 1  
  
## [1] FALSE                                ## [1] TRUE  
  
"hi" == "hi"                                sqrt(3)^2 == 3  
  
## [1] TRUE                                 ## [1] FALSE  
  
4 >= 1                                      dplyr::near(sqrt(3)^2, 3)  
  
## [1] TRUE                                 ## [1] TRUE
```

Logical statements

Goal: Subset rows or columns

- **logical statement** - comparison that resolves as TRUE or FALSE

```
#use of is. functions
is.character("10")                                is.character("10")
is.numeric("Word")                                ## [1] TRUE
## [1] FALSE

is.na(c(1:2, NA, 3))                            is.na(c(1:2, NA, 3))
is.numeric(10)                                    ## [1] FALSE FALSE TRUE FALSE
## [1] TRUE

is.matrix(c("hello", "world"))                   is.matrix(c("hello", "world"))
## [1] FALSE
```

Logical statements

Goal: Subset rows or columns

- **logical statement** - useful for indexing an R object
- Concept:
 - Feed index a vector of TRUE/FALSE
 - R returns elements where TRUE

```
myBatting$G > 20 #vector indicating Games > 20
```

```
##      [1] FALSE  TRUE   TRUE   TRUE   TRUE  FALSE  FALSE   TRUE  FALSE  FALSE   TRUE  FALSE
##     [13] FALSE  FALSE  FALSE   TRUE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE   TRUE  FALSE
##     [25] FALSE  FALSE   TRUE  FALSE  FALSE   TRUE   TRUE  FALSE  FALSE   TRUE   TRUE  FALSE
##     [37] TRUE   TRUE   TRUE  FALSE   TRUE   TRUE  FALSE  FALSE   TRUE  FALSE   TRUE  FALSE
##     [49] TRUE   TRUE  FALSE   TRUE   TRUE  FALSE  TRUE   TRUE  FALSE  FALSE   TRUE  FALSE
##     [61] TRUE  FALSE   TRUE  FALSE  FALSE   TRUE   TRUE   TRUE  FALSE   TRUE   TRUE  FALSE
##     [73] TRUE   TRUE   TRUE   TRUE   TRUE  FALSE  FALSE   TRUE   TRUE   TRUE  FALSE
##     [85] TRUE   TRUE   TRUE  FALSE  FALSE  FALSE   TRUE   TRUE  FALSE   TRUE  FALSE   TRUE
##     [97] TRUE  FALSE   TRUE   TRUE  FALSE   TRUE   TRUE  FALSE   TRUE   TRUE   TRUE  FALSE
##    [109] FALSE  TRUE   TRUE  FALSE   TRUE   TRUE   TRUE  FALSE   TRUE  FALSE  FALSE   TRUE
##    [121] FALSE  TRUE  FALSE   TRUE   TRUE  FALSE  FALSE  FALSE  FALSE   TRUE   TRUE   TRUE
##    [133] TRUE  FALSE  FALSE  FALSE   TRUE  FALSE  TRUE  FALSE  FALSE   TRUE  FALSE  TRUE
##    [145] FALSE  FALSE   TRUE  FALSE  FALSE  FALSE   TRUE  FALSE  TRUE   TRUE   TRUE  TRUE
```

filter() - subset rows or columns

- **logical statement** - useful for indexing an R object

```
filter(myBatting, G > 20)
```

```
## # A tibble: 70,417 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 addybo01  1871     1 RC1     NA     25   118    30    32     6     0     0
## 2 allisar... 1871     1 CL1     NA     29   137    28    40     4     5     0
## 3 allisdo... 1871     1 WS3     NA     27   133    28    44    10     2     2
## 4 ansonca... 1871     1 RC1     NA     25   120    29    39    11     3     0
## 5 barnero... 1871     1 BS1     NA     31   157    66    63    10     9     0
## 6 bassjo01  1871     1 CL1     NA     22   89     18    27     1    10     3
## 7 bellast... 1871     1 TRO     NA     29   128    26    32     3     3     0
## 8 birdge01  1871     1 RC1     NA     25   106    19    28     2     5     0
## 9 birdsda... 1871     1 BS1     NA     29   152    51    46     3     3     0
## 10 brainas... 1871     1 WS3    NA     30   134    24    30     4     0     0
## # ... with 70,407 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

Logical statements

Compound logicals via Logical Operators

- & 'and'
- | 'or'

| Operator | A,B true | A true, B false | A,B false |
|----------|--------------|-----------------|---------------|
| & | A & B = TRUE | A & B = FALSE | A & B = FALSE |
| | A B = TRUE | A B = TRUE | A B = FALSE |

Logical statements

- Pull out those that played more than 20 games and played in 2015

```
(myBatting$G > 20) & (myBatting$yearID == 2015)
```

```
## [1] FALSE  
## [13] FALSE  
## [25] FALSE  
## [37] FALSE  
## [49] FALSE  
## [61] FALSE  
## [73] FALSE  
## [85] FALSE  
## [97] FALSE  
## [109] FALSE  
## [121] FALSE  
## [133] FALSE  
## [145] FALSE  
## [157] FALSE  
## [169] FALSE  
## [181] FALSE  
## [193] FALSE  
## [205] FALSE  
## [217] FALSE  
## [229] FALSE  
## [241] FALSE FALSE
```

filter() - subset rows or columns

- Pull out those that played more than 20 games and played in 2015

```
filter(myBatting, (G > 20) & (yearID == 2015))
```

```
## # A tibble: 949 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>     <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 aardsda...  2015     1 ATL    NL     33     1     0     0     0     0     0     0
## 2 abadfe01   2015     1 OAK    AL     62     0     0     0     0     0     0     0
## 3 abreuj...  2015     1 CHA    AL    154    613    88   178    34     3    30
## 4 ackledu...  2015     1 SEA    AL     85    186    22    40     8     1     6
## 5 ackledu...  2015     2 NYA    AL     23     52     6    15     3     2     4
## 6 adamecr...  2015     1 COL    NL     26     53     4    13     1     1     0
## 7 adamsau...  2015     1 CLE    AL     28     1     0     0     0     0     0
## 8 adamsma...  2015     1 SLN    NL     60    175    14    42     9     0     5
## 9 adriaeh...  2015     1 SFN    NL     52   113    11    21     7     1     0
## 10 affelje... 2015     1 SFN    NL     52     2     0     0     0     0     0
## # ... with 939 more rows, and 10 more variables: RBI <int>, SB <int>, CS <int>,
## #   BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>, GIDP <int>
```

arrange() - reorder rows

```
#reorder by teamID
arrange(myBatting, teamID)

## # A tibble: 107,429 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 berrych...  1884     1 ALT    UA     7    25     2     6     0     0     0     0
## 2 brownji...  1884     1 ALT    UA    21    88    12    22     2     2     1
## 3 carropa...  1884     1 ALT    UA    11    49     4    13     1     0     0
## 4 connojo...  1884     1 ALT    UA     3    11     0     1     0     0     0
## 5 crosscl...  1884     1 ALT    UA     2     7     1     4     1     0     0
## 6 daisege...  1884     1 ALT    UA     1     4     0     0     0     0     0
## 7 doughch...  1884     1 ALT    UA    23    85     6    22     5     0     0
## 8 gradyjo...  1884     1 ALT    UA     9    36     5    11     3     0     0
## 9 harrifr...  1884     1 ALT    UA    24    95    10    25     2     1     0
## 10 koonsha... 1884     1 ALT    UA    21    78     8    18     2     1     0
## # ... with 107,419 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

arrange() - reorder rows

```
#get secondary arrangement as well
arrange(myBatting, teamID, G)

## # A tibble: 107,429 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int>
## 1 daisege...  1884     1 ALT    UA      1     4     0     0     0     0     0
## 2 crosscl...  1884     1 ALT    UA      2     7     1     4     1     0     0
## 3 manloch...  1884     1 ALT    UA      2     7     1     3     0     0     0
## 4 connojo...  1884     1 ALT    UA      3    11     0     1     0     0     0
## 5 shafff01   1884     1 ALT    UA      6    19     1     3     0     0     0
## 6 berrych...  1884     1 ALT    UA      7    25     2     6     0     0     0
## 7 noftsge...  1884     1 ALT    UA      7    25     0     1     0     0     0
## 8 learyja...  1884     1 ALT    UA      8    33     1     3     0     0     0
## 9 gradyjo...  1884     1 ALT    UA      9    36     5    11     3     0     0
## 10 carropa... 1884     1 ALT   UA     11    49     4    13     1     0     0
## # ... with 107,419 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

arrange() - reorder rows

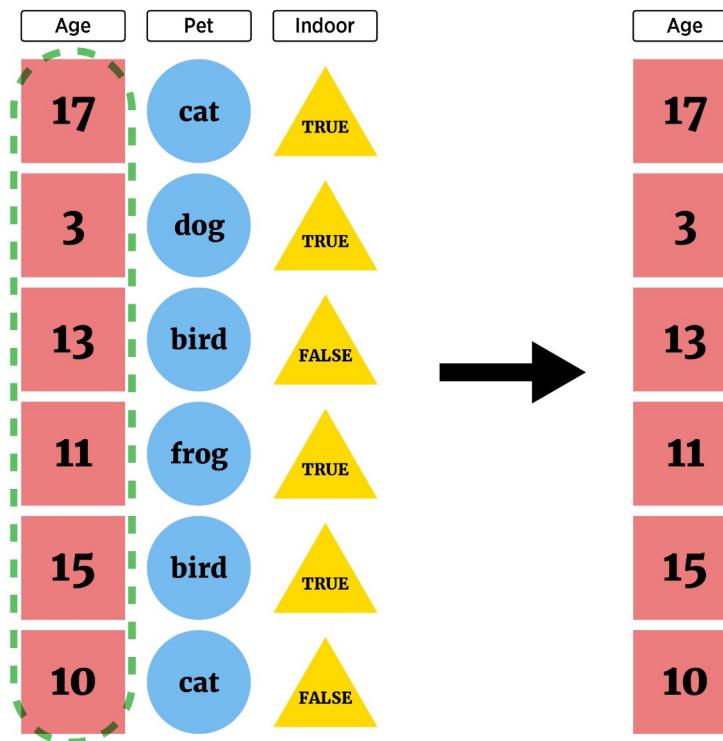
```
#descending instead
arrange(myBatting, teamID, desc(G))

## # A tibble: 107,429 x 22
##   playerID yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int>
## 1 smithge...  1884     1 ALT    UA     25  108     9    34     8     1     0
## 2 harrifr...  1884     1 ALT    UA     24   95    10    25     2     1     0
## 3 doughch...  1884     1 ALT    UA     23   85     6    22     5     0     0
## 4 murphjo...  1884     1 ALT    UA     23   94    10    14     1     0     0
## 5 brownji...  1884     1 ALT    UA     21   88    12    22     2     2     1
## 6 koonsha...  1884     1 ALT    UA     21   78     8    18     2     1     0
## 7 mooreje...  1884     1 ALT    UA     20   80    10    25     3     1     1
## 8 shaffta...  1884     1 ALT    UA     13   55    10    18     2     0     0
## 9 carropa...  1884     1 ALT    UA     11   49     4    13     1     0     0
## 10 gradyjo... 1884     1 ALT    UA      9   36     5    11     3     0     0
## # ... with 107,419 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

Data manipulation idea

We may want to subset our full data set or create new data

- Look at only certain variables (**select columns**)



select() - subset columns

- May only want certain variables
- `select()` function has same syntax as other `dplyr` functions!

```
#Choose a single column by name  
select(myBatting, X2B)
```

```
## # A tibble: 107,429 x 1  
##       X2B  
##   <int>  
## 1     0  
## 2     6  
## 3     4  
## 4    10  
## 5    11  
## 6     2  
## 7     0  
## 8    10  
## 9     1  
## 10    2  
## # ... with 107,419 more rows
```

select() - subset columns

- Often only want select variables
- `select()` function has same syntax as other `dplyr` functions!

```
#Choose a single column by name  
select(myBatting, playerID, X2B)
```

```
## # A tibble: 107,429 x 2  
##   playerID     X2B  
##   <chr>      <int>  
## 1 abercda01     0  
## 2 addybo01      6  
## 3 allisar01     4  
## 4 allisdo01    10  
## 5 ansonca01    11  
## 6 armstbo01     2  
## 7 barkeal01     0  
## 8 barnero01    10  
## 9 barrebi01     1  
## 10 barrofr01    2  
## # ... with 107,419 more rows
```

Aside: Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with `%>%` operator helps

```
arrange(select(filter(myBatting, teamID == "PIT"), playerID, G, X2B), desc(X2B))
```

```
## # A tibble: 4,871 x 3
##   playerID      G   X2B
##   <chr>     <int> <int>
## 1 wanerpa01    154    62
## 2 wanerpa01    148    53
## 3 sanchfr01    157    53
## 4 wanerpa01    152    50
## 5 comorad01    152    47
## 6 mclouna01    152    46
## 7 wagneho01    135    45
## 8 parkeda01    158    45
## 9 vanslan01    154    45
## 10 wagneho01   132    44
## # ... with 4,861 more rows
```

Aside: Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with `%>%` operator helps

```
myBatting %>%
  filter(teamID == "PIT") %>%
  select(playerID, G, X2B) %>%
  arrange(desc(X2B))
```

```
## # A tibble: 4,871 x 3
##   playerID      G   X2B
##   <chr>     <int> <int>
## 1 wanerpa01    154    62
## 2 wanerpa01    148    53
## 3 sanchfr01    157    53
## 4 wanerpa01    152    50
## 5 comorad01    152    47
## 6 mclouna01    152    46
## 7 wagneho01    135    45
## 8 parkeda01    158    45
## 9 vanslan01    154    45
## 10 wagneho01   132    44
## # ... with 4,861 more rows
```

Aside: Piping or Chaining

- Generically, pipe does the following
 - x `%>% f(y)` turns into `f(x, y)`
 - x `%>% f(y) %>% g(z)` turns into `g(f(x, y), z)`
- Can be used with functions outside the tidyverse if this structure works!

select() - subset columns

- Great functionality for choosing variables

```
#all columns between
myBatting %>% select(X2B:HR)

## # A tibble: 107,429 x 3
##       X2B     X3B     HR
##   <int> <int> <int>
## 1     0     0     0
## 2     6     0     0
## 3     4     5     0
## 4    10     2     2
## 5    11     3     0
## 6     2     1     0
## 7     0     0     0
## 8    10     9     0
## 9     1     0     0
## 10    2     1     0
## # ... with 107,419 more rows
```

select() - subset columns

- Great functionality for choosing variables

```
#all columns containing
myBatting %>% select(contains("X"))

## # A tibble: 107,429 x 2
##       X2B     X3B
##   <int> <int>
## 1     0     0
## 2     6     0
## 3     4     5
## 4    10     2
## 5    11     3
## 6     2     1
## 7     0     0
## 8    10     9
## 9     1     0
## 10    2     1
## # ... with 107,419 more rows
```

select() - subset columns

- Great functionality for choosing variables

```
#all columns starting with
myBatting %>% select(starts_with("X"))

## # A tibble: 107,429 x 2
##       X2B     X3B
##   <int> <int>
## 1     0     0
## 2     6     0
## 3     4     5
## 4    10     2
## 5    11     3
## 6     2     1
## 7     0     0
## 8    10     9
## 9     1     0
## 10    2     1
## # ... with 107,419 more rows
```

select() - subset columns

- Great functionality for choosing variables

```
#multiple selections
myBatting %>% select(starts_with("X"), ends_with("ID"), G)

## # A tibble: 107,429 x 7
##       X2B     X3B playerID   yearID teamID lgID      G
##   <int> <int> <chr>     <int> <fct> <fct> <int>
## 1     0     0 abercda01  1871  TRO    NA     1
## 2     6     0 addybo01  1871  RC1    NA    25
## 3     4     5 allisar01  1871  CL1    NA    29
## 4    10     2 allisdo01  1871  WS3    NA    27
## 5    11     3 ansonca01  1871  RC1    NA    25
## 6     2     1 armstbo01  1871  FW1    NA    12
## 7     0     0 barkeal01  1871  RC1    NA     1
## 8    10     9 barnero01  1871  BS1    NA    31
## 9     1     0 barrebi01  1871  FW1    NA     1
## 10    2     1 barrofr01  1871  BS1    NA    18
## # ... with 107,419 more rows
```

select() - subset columns

- Can reorder variables

```
#reorder  
myBatting %>% select(playerID, HR, everything())  
  
## # A tibble: 107,429 x 22  
##   playerID     HR yearID stint teamID lgID      G    AB     R     H    X2B    X3B  
##   <chr>     <int> <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>  
## 1 abercda...     0    1871     1 TRO     NA     1     4     0     0     0     0  
## 2 addybo01      0    1871     1 RC1     NA    25    118    30    32     6     0  
## 3 allisar...     0    1871     1 CL1     NA    29    137    28    40     4     5  
## 4 allisdo...     2    1871     1 WS3     NA    27    133    28    44    10     2  
## 5 ansonca...     0    1871     1 RC1     NA    25    120    29    39    11     3  
## 6 armstbo...     0    1871     1 FW1     NA    12     49     9    11     2     1  
## 7 barkeal...     0    1871     1 RC1     NA     1     4     0     1     0     0  
## 8 barnero...     0    1871     1 BS1     NA    31    157    66    63    10     9  
## 9 barrebi...     0    1871     1 FW1     NA     1     5     1     1     1     0  
## 10 barrofr...    0    1871     1 BS1    NA    18     86    13    13     2     1  
## # ... with 107,419 more rows, and 10 more variables: RBI <int>, SB <int>,  
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,  
## #   GIDP <int>
```

rename() - rename variables

```
#rename our previous
myBatting %>%
  select(starts_with("X"), ends_with("ID"), G) %>%
  rename("Doubles" = X2B, "Triples" = X3B)

## # A tibble: 107,429 x 7
##   Doubles Triples playerID yearID teamID lgID     G
##   <int>   <int> <chr>     <int> <fct> <fct> <int>
## 1     0       0 abercda01  1871  TRO    NA      1
## 2     6       0 addybo01  1871  RC1    NA     25
## 3     4       5 allisar01  1871  CL1    NA     29
## 4    10       2 allisdo01  1871  WS3    NA     27
## 5    11       3 ansonca01  1871  RC1    NA     25
## 6     2       1 armstbo01  1871  FW1    NA     12
## 7     0       0 barkeal01  1871  RC1    NA      1
## 8    10       9 barnero01  1871  BS1    NA     31
## 9     1       0 barrebi01  1871  FW1    NA      1
## 10    2       1 barrofr01  1871  BS1    NA     18
## # ... with 107,419 more rows
```

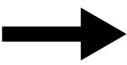
dplyr

[Cheat sheet](#)

- Basic commands
 - `as_tibble()` - convert data frame to one with better printing
 - `filter()` - subset **rows**
 - `arrange()` - reorder **rows**
 - `select()` - subset **columns**
- Many joins to combine tibbles too! (Similar to SQL)

Data manipulation idea

- Create new variables



| Age | Pet | Indoor | |
|-----|------|--------|--|
| 17 | cat | TRUE | |
| 3 | dog | TRUE | |
| 13 | bird | FALSE | |
| 11 | frog | TRUE | |
| 15 | bird | TRUE | |
| 10 | cat | FALSE | |

| Age | Pet | Indoor | AgeType |
|-----|------|--------|---------|
| 17 | cat | TRUE | old |
| 3 | dog | TRUE | young |
| 13 | bird | FALSE | young |
| 11 | frog | TRUE | old |
| 15 | bird | TRUE | young |
| 10 | cat | FALSE | old |

Creating New Variables

- `mutate()` - add newly created **column(s)** to current data frame (doesn't overwrite the data frame)
- `transmute()` - create new data frame with created variable(s) only
- Syntax:

```
mutate(data, newVarName = functionOfData, newVarName2 =  
functionOfData, ...)
```

Creating New Variables

- Consider a data set on movie ratings

```
library(fivethirtyeight)
fandango

## # A tibble: 146 x 23
##   film    year rottentomatoes rottentomatoes_norm metacritic metacritic_user  imdb
##   <chr>   <dbl>        <int>                <int>      <int>      <dbl> <dbl>
## 1 Aven...  2015         74                  86       66       7.1    7.8
## 2 Cind...  2015         85                  80       67       7.5    7.1
## 3 Ant-...  2015         80                  90       64       8.1    7.8
## 4 Do Y...  2015         18                  84       22       4.7    5.4
## 5 Hot ... 2015         14                  28       29       3.4    5.1
## 6 The ... 2015         63                  62       50       6.8    7.2
## 7 Irra...  2015         42                  53       53       7.6    6.9
## 8 Top ... 2014         86                  64       81       6.8    6.5
## 9 Shau...  2015         99                  82       81       8.8    7.4
## 10 Love... 2015        89                  87       80       8.5    7.8
## # ... with 136 more rows, and 16 more variables: fandango_stars <dbl>,
## #   fandango_ratingvalue <dbl>, rt_norm <dbl>, rt_user_norm <dbl>,
## #   metacritic_norm <dbl>, metacritic_user_norm <dbl>, imdb_norm <dbl>,
## #   rt_norm_round <dbl>, rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #   metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>,
## #   ... and 10 more variables: fandango_norm <dbl>, fandango_norm_round <dbl>
```

mutate() - create new column(s)

```
##Create an average rottentomatoes score variable
fandango %>%
  mutate(avgRotten = (rottentomatoes + rottentomatoes_user)/2)

## # A tibble: 146 x 24
##   film      year rottentomatoes rottentomatoes_... metacritic metacritic_user    imdb
##   <chr>     <dbl>        <int>            <int>       <int>        <dbl> <dbl>
## 1 Aven...    2015         74             86         66        7.1  7.8
## 2 Cind...    2015         85             80         67        7.5  7.1
## 3 Ant-...    2015         80             90         64        8.1  7.8
## 4 Do Y...    2015         18             84         22        4.7  5.4
## 5 Hot ...   2015         14             28         29        3.4  5.1
## 6 The ...   2015         63             62         50        6.8  7.2
## 7 Irra...    2015         42             53         53        7.6  6.9
## 8 Top ...   2014         86             64         81        6.8  6.5
## 9 Shau...    2015         99             82         81        8.8  7.4
## 10 Love...   2015         89             87         80        8.5  7.8
## # ... with 136 more rows, and 17 more variables: fandango_stars <dbl>,
## #   fandango_ratingvalue <dbl>, rt_norm <dbl>, rt_user_norm <dbl>,
## #   metacritic_norm <dbl>, metacritic_user_nom <dbl>, imdb_norm <dbl>,
## #   rt_norm_round <dbl>, rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #   metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>,
## #   fandango_votes <int>, fandango_difference <dbl>, avgRotten <dbl>
```

mutate () - create new column(s)

```
#can't see it!
fandango %>%
  mutate(avgRotten = (rottentomatoes + rottentomatoes_user)/2) %>%
  select(film, year, avgRotten, everything())

## # A tibble: 146 x 24
##   film    year avgRotten rottentomatoes rottentomatoes_... metacritic
##   <chr>  <dbl>      <dbl>          <int>          <int>          <int>
## 1 Aven...  2015        80            74            86            66
## 2 Cind...  2015       82.5           85            80            67
## 3 Ant-...  2015        85            80            90            64
## 4 Do Y...  2015        51            18            84            22
## 5 Hot ...  2015        21            14            28            29
## 6 The ...  2015       62.5           63            62            50
## 7 Irra...  2015       47.5           42            53            53
## 8 Top ...  2014        75            86            64            81
## 9 Shau...  2015       90.5           99            82            81
## 10 Love... 2015       88            89            87            80
## # ... with 136 more rows, and 18 more variables: metacritic_user <dbl>,
## #     imdb <dbl>, fandango_stars <dbl>, fandango_ratingvalue <dbl>,
## #     rt_norm <dbl>, rt_user_norm <dbl>, metacritic_norm <dbl>,
## #     metacritic_user_nom <dbl>, imdb_norm <dbl>, rt_norm_round <dbl>,
## #     rt_user_norm_round <dbl>, metacritic_norm_round <dbl>,
## #     metacritic_user_norm_round <dbl>, imdb_norm_round <dbl>,
```

mutate() - create new column(s)

- Add more than one variable

```
fandango %>%
  mutate(avgRotten = (rottentomatoes + rottentomatoes_user)/2,
        avgMeta = (metacritic_norm + metacritic_user_nom)/2) %>%
  select(film, year, avgRotten, avgMeta, everything())

## # A tibble: 146 x 25
##   film    year avgRotten avgMeta rottentomatoes rottentomatoes_... metacritic
##   <chr> <dbl>     <dbl>     <dbl>       <int>           <int>       <int>
## 1 Aven...  2015      80     3.42         74            86          66
## 2 Cind...  2015     82.5    3.55         85            80          67
## 3 Ant-...  2015      85     3.62         80            90          64
## 4 Do Y...  2015      51     1.72         18            84          22
## 5 Hot ...  2015      21     1.58         14            28          29
## 6 The ...  2015     62.5    2.95         63            62          50
## 7 Irra...  2015     47.5    3.22         42            53          53
## 8 Top ...  2014      75     3.72         86            64          81
## 9 Shau...  2015     90.5    4.22         99            82          81
## 10 Love... 2015     88     4.12         89            87          80
## # ... with 136 more rows, and 18 more variables: metacritic_user <dbl>,
## #   imdb <dbl>, fandango_stars <dbl>, fandango_ratingvalue <dbl>,
## #   rt_norm <dbl>, rt_user_norm <dbl>, metacritic_norm <dbl>,
## #   metacritic_user_nom <dbl>, imdb_norm <dbl>, rt_norm_round <dbl>,
```

Creating New Variables

`mutate()` and `transmute()` can also use some statistical functions

```
fandango %>%
  select(rottentomatoes) %>%
  mutate(avg = mean(rottentomatoes), sd = sd(rottentomatoes))

## # A tibble: 146 x 3
##   rottentomatoes     avg      sd
##       <int>    <dbl>   <dbl>
## 1          74    60.8   30.2
## 2          85    60.8   30.2
## 3          80    60.8   30.2
## 4          18    60.8   30.2
## 5          14    60.8   30.2
## 6          63    60.8   30.2
## 7          42    60.8   30.2
## 8          86    60.8   30.2
## 9          99    60.8   30.2
## 10         89    60.8   30.2
## # ... with 136 more rows
```

Creating New Variables

`mutate()` and `transmute()` can also use some statistical functions

- `group_by` to create summaries for groups

```
fandango %>%
  select(year, rottentomatoes) %>%
  group_by(year) %>%
  mutate(avg = mean(rottentomatoes), sd = sd(rottentomatoes))
```

```
## # A tibble: 146 x 4
## # Groups:   year [2]
##       year rottentomatoes     avg      sd
##   <dbl>          <int> <dbl> <dbl>
## 1 2015              74  58.4  30.3
## 2 2015              85  58.4  30.3
## 3 2015              80  58.4  30.3
## 4 2015              18  58.4  30.3
## 5 2015              14  58.4  30.3
## 6 2015              63  58.4  30.3
## 7 2015              42  58.4  30.3
## 8 2014              86  79.5  21.6
## 9 2015              99  58.4  30.3
## 10 2015             89  58.4  30.3
## # ... with 136 more rows
```

Other notes: Summarizing with `dplyr`

- `summarize()` is often used with `group_by()` to obtain summaries without adding to the dataset

```
fandango %>%
  select(year, rottentomatoes) %>%
  group_by(year) %>%
  summarize(avg = mean(rottentomatoes), sd = sd(rottentomatoes))

## # A tibble: 2 x 3
##   year     avg     sd
##   <dbl> <dbl> <dbl>
## 1 2014    79.5  21.6
## 2 2015    58.4  30.3
```

if_else() - conditional execution

- Often want to execute statements conditionally to create a variable
- `if_else()` syntax:
 - `if_else(condition, true, false)`
 - `condition` is a vector of TRUE/FALSE
 - `true` is what to do when TRUE occurs
 - `false` is what to do when FALSE occurs

Creating New Variables

Conditional Execution with If then, If then else

- Consider built-in data set `airquality`
 - daily air quality measurements in New York
 - from May (Day 1) to September (Day 153) in 1973

```
myAirquality <- as_tibble(airquality)
myAirquality

## # A tibble: 153 x 6
##   Ozone Solar.R  Wind  Temp Month Day
##   <int>    <int> <dbl> <int> <int> <int>
## 1     41      190    7.4    67     5     1
## 2     36      118     8     72     5     2
## 3     12      149   12.6    74     5     3
## 4     18      313   11.5    62     5     4
## 5     NA       NA  14.3    56     5     5
## 6     28       NA  14.9    66     5     6
## 7     23      299    8.6    65     5     7
## 8     19       99  13.8    59     5     8
## 9      8       19  20.1    61     5     9
## 10    NA      194    8.6    69     5    10
```

if_else() with mutate()

```
myAirquality <- myAirquality %>%
  mutate(Status = if_else(Wind >= 15, "HighWind",
                         if_else(Wind >= 10, "Windy",
                                if_else(Wind >= 6, "LightWind", "Calm"))))

myAirquality
```



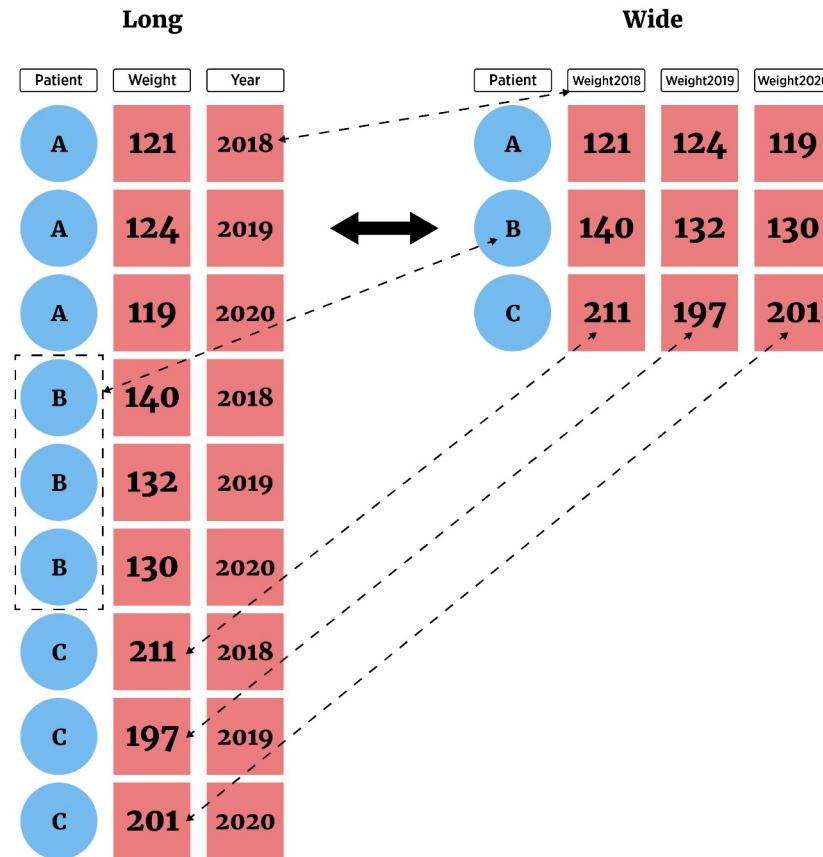
```
## # A tibble: 153 x 7
##   Ozone Solar.R  Wind  Temp Month Day Status
##   <int>    <int> <dbl> <int> <int> <int> <chr>
## 1     41      190   7.4    67     5     1 LightWind
## 2     36      118     8    72     5     2 LightWind
## 3     12      149  12.6    74     5     3 Windy
## 4     18      313  11.5    62     5     4 Windy
## 5     NA       NA  14.3    56     5     5 Windy
## 6     28       NA  14.9    66     5     6 Windy
## 7     23      299   8.6    65     5     7 LightWind
## 8     19       99  13.8    59     5     8 Windy
## 9      8       19  20.1    61     5     9 HighWind
## 10    NA      194   8.6    69     5    10 LightWind
## # ... with 143 more rows
```

Creating New Variables Recap!

- `mutate()` - add newly created **column(s)** to current data frame
- `transmute()` - create new data frame with created variable(s)
 - Use `if_else()` to do conditional creation
 - Note: `cut()` can be used to categorize a numeric variable!

Other notes: Reshaping Data

Long vs Wide format data



Other notes: Reshaping Data with the `tidyverse` package

Easily allows for two very important actions

- `pivot_longer()` - lengthens data by increasing the number of rows and decreasing the number of columns
 - Most important as analysis methods often prefer this form
- `pivot_wider()` - widens data by increasing the number of columns and decreasing the number of rows

Exercises

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_4_Exercises.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_4_Exercises.html)

Partial Solution

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_4_Solutions.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_4_Solutions.html)

Schedule

- RStudio IDE (Integrated Development Environment)
- Vectors and Data Frames/Tibbles
- Importing CSV files with `readr`
- Markdown basics
- Common data manipulation with `dplyr`
- Plotting with `ggplot2`
- Shiny basics

Graphical Summaries

Three major systems for plotting:

- Base R (built-in functions)
- Lattice
- ggplot2 (sort of part of the tidyverse - [Cheat Sheet](#))
 - `ggplot(data = data_frame)` creates a plot instance
 - Add “layers” to the system (geoms or stats)

Great [reference book here!](#)

ggplot2 Plotting

ggplot2 basics ([Cheat Sheet](#))

- `ggplot(data = data_frame)` creates a plot instance
- Add “layers” to the system (geoms or stats)
 - Creates a visualization of the data

ggplot2 Plotting

ggplot2 basics ([Cheat Sheet](#))

- `ggplot(data = data_frame)` creates a plot instance
- Add “layers” to the system (geoms or stats)
 - Creates a visualization of the data
- Modify layer “mapping” args (aes)
 - Ex: size, color, and x, y location(s)
- Coordinate system (mostly use Cartesian plane)
- Optional: Titles, etc.

factors

- factor - special class of vector with a `levels` attribute
- Levels define all possible values for that variable
 - Great for variable like `Day` (Monday, Tuesday, ...)
 - Not great for variable like `Name` where new values may come up
- Quite useful with plotting
 - Allows for easy labeling of subgroups

factors

- Consider data on titanic passengers in `titanic.csv`

```
titanicData <- read_csv("titanic.csv")
#convert survival status to a factor
titanicData$survived <- as.factor(titanicData$survived)
levels(titanicData$survived) #R knows it isn't numeric now

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

- Can't add value unless it is a level

```
titanicData$survived[1] <- "5"

## Warning in `[<-.factor`(`*tmp*`, 1, value = structure(c(NA, 2L, 1L, 1L, :
## invalid factor level, NA generated
```

factor levels

- Useful if you want to create better labels (or change the ordering)

```
levels(titanicData$survived) <- c("Died", "Survived")
```

```
levels(titanicData$survived)
```

```
## [1] "Died"      "Survived"
```

ggplot2 Plotting: Categorical variables

Categorical variable - entries are a label or attribute

Generally, describe distribution using a barplot!

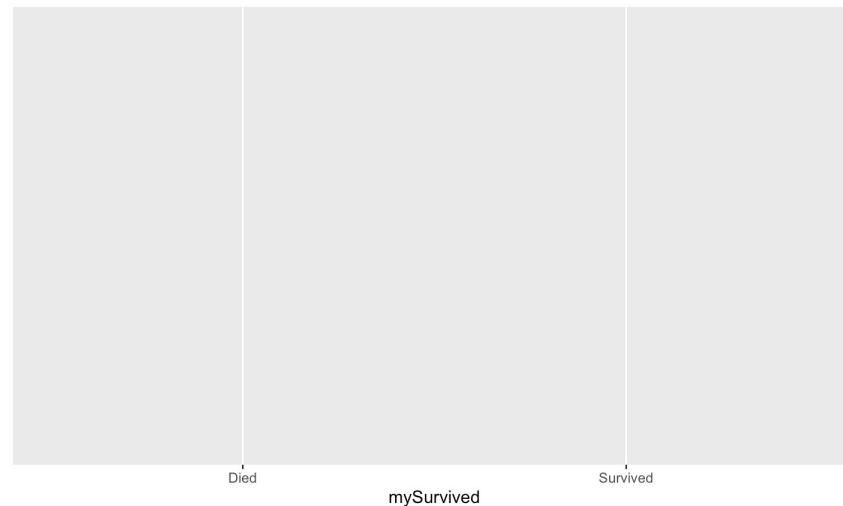
- Barplots via ggplot + geom_bar

```
titanicData <- read_csv("titanic.csv")
titanicData$mySurvived <- as.factor(titanicData$survived)
levels(titanicData$mySurvived) <- c("Died", "Survived")
titanicData$myEmbarked <- as.factor(titanicData$embarked)
levels(titanicData$myEmbarked) <- c("Cherbourg", "Queenstown", "Southampton")
titanicData <- titanicData %>% drop_na(mySurvived, sex, myEmbarked)
```

ggplot2 barplots

- Barplots via `ggplot + geom_bar`
- Across x-axis we want our categories - specify with `aes(x = ...)`

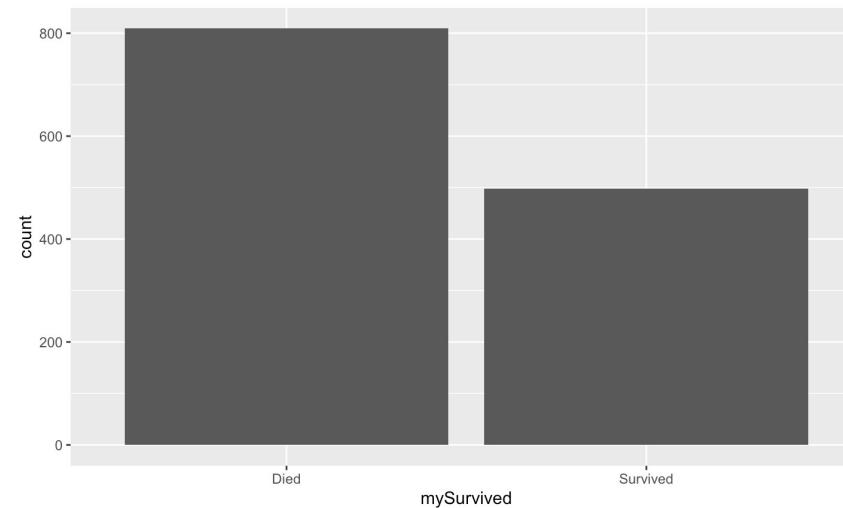
```
ggplot(data = titanicData, aes(x = mySurvived) )
```



ggplot2 barplots

- Barplots via `ggplot + geom_bar`
- Must add `geom` (or `stat`) layer!

```
ggplot(data = titanicData, aes(x = mySurvived)) + geom_bar()
```



ggplot2 barplots

- Generally: Save base object, then “add layers”

```
g <- ggplot(data = titanicData, aes(x = mySurvived))  
g + geom_bar()
```

ggplot2 barplots

- Generally: Save base object, then “add layers”

```
g <- ggplot(data = titanicData, aes(x = mySurvived))  
g + geom_bar()
```

- `aes()` defines visual properties of objects in the plot

`x = , y = , size = , shape = , color = , alpha = , ...`

- [Cheat Sheet](#) gives most common properties for a given `geom`

ggplot2 barplots

- Generally: Save base object, then “add layers”

```
g <- ggplot(data = titanicData, aes(x = mySurvived))  
g + geom_bar()
```

- `aes()` defines visual properties of objects in the plot

`x = , y = , size = , shape = , color = , alpha = , ...`

- Cheat Sheet gives most common properties for a given `geom`

```
d + geom_bar()
```

`x, alpha, color, fill, linetype, size, weight`

ggplot2 global and local aesthetics

data and aes can be set in two ways;

- 'globally' (for all layers) via the `ggplot` statement
- 'locally' (for just that layer) via the `geom`, `stat`, etc. layer

ggplot2 global and local aesthetics

data and aes can be set in two ways;

- 'globally' (for all layers) via the `ggplot` statement
- 'locally' (for just that layer) via the `geom`, `stat`, etc. layer

```
#global  
ggplot(data = titanicData, aes(x = mySurvived)) + geom_bar()  
#local  
ggplot() + geom_bar(data = titanicData, aes(x = mySurvived))
```

ggplot2 global and local aesthetics

data and aes can be set in two ways;

- 'globally' (for all layers) via the `ggplot` statement
- 'locally' (for just that layer) via the `geom`, `stat`, etc. layer

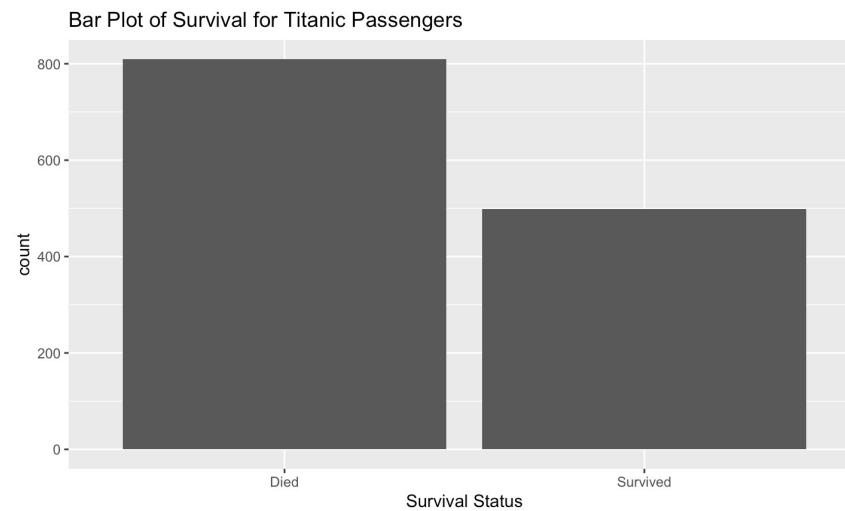
```
#global  
ggplot(data = titanicData, aes(x = mySurvived)) + geom_bar()  
#local  
ggplot() + geom_bar(data = titanicData, aes(x = mySurvived))
```

- To set an attribute that doesn't depend on the data (i.e. `color = 'blue'`), generally place these outside of the `aes`

ggplot2 barplots

- Add better labels and a title (new layers, see cheat sheet!)

```
ggplot(data = titanicData, aes(x = mySurvived)) +  
  geom_bar() +  
  labs(x = "Survival Status", title = "Bar Plot of Survival for Titanic Passengers")
```



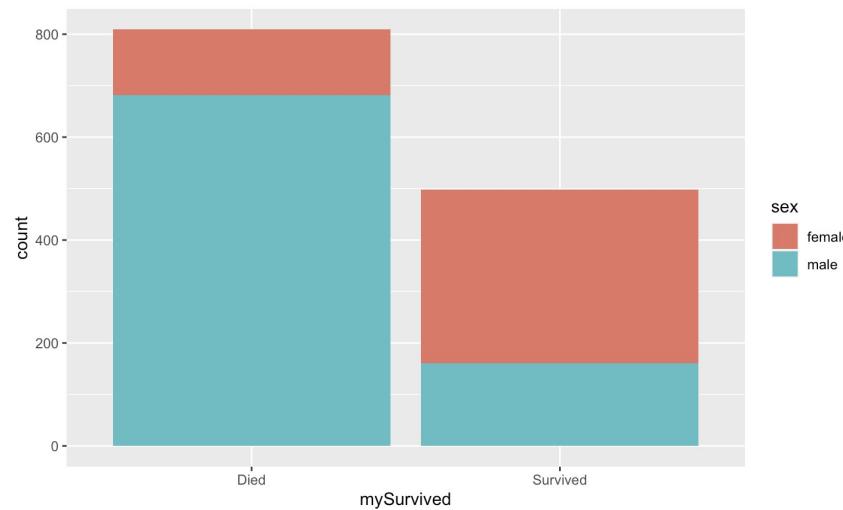
ggplot2 stacked barplots

- Stacked barplot created by via `fill` aesthetic and same process
 - Create base object
 - Add geoms
 - Use `aes` to specify aspects of the plot

ggplot2 stacked barplots

- Stacked barplot created by via `fill` aesthetic
- Automatic assignment of colors, creation of legends, etc. for `aes` elements (except with `group`)

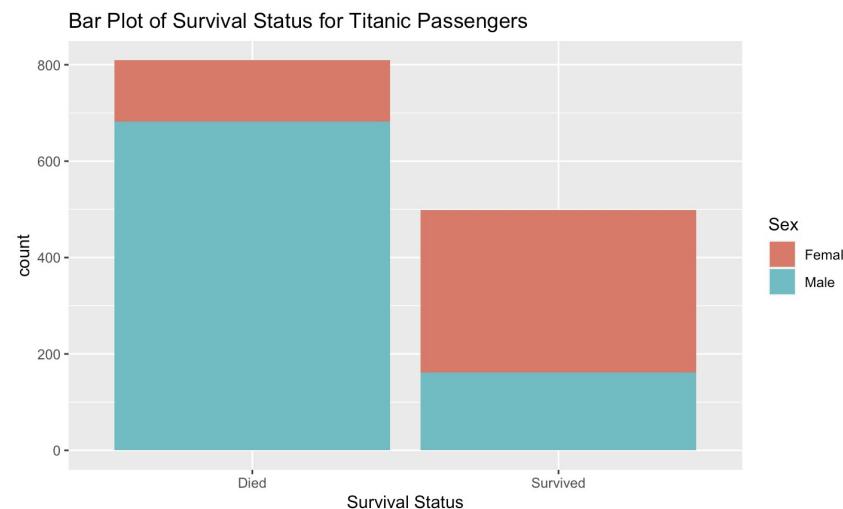
```
ggplot(data = titanicData, aes(x = mySurvived, fill = sex)) + geom_bar()
```



ggplot2 labeling

- Add custom labels by adding more layers

```
ggplot(data = titanicData, aes(x = mySurvived, fill = sex)) +  
  geom_bar() +  
  labs(x = "Survival Status",  
       title = "Bar Plot of Survival Status for Titanic Passengers") +  
  scale_fill_discrete(name = "Sex", labels = c("Female", "Male"))
```



ggplot2 labeling

- Adjusting appropriate labeling via `scale_*_discrete`

```
aes(x = survived, fill = sex)

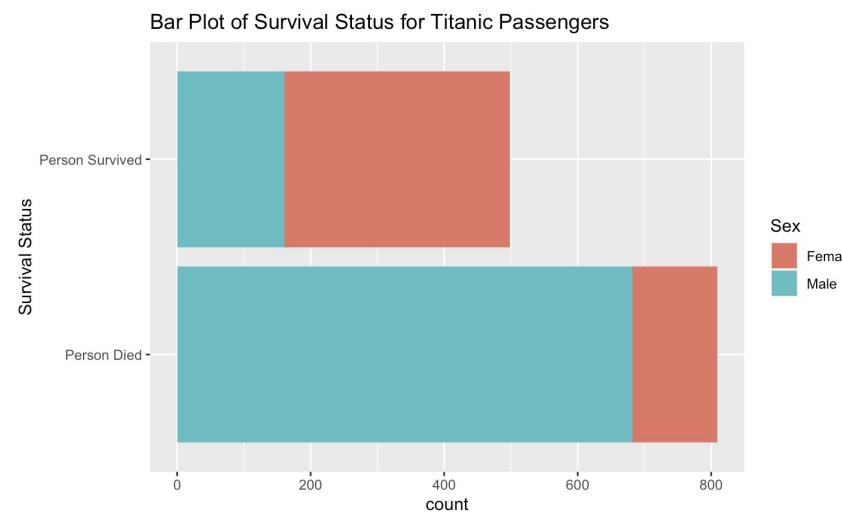
scale_x_discrete(labels = c("Person Died", "Person Survived"))

scale_fill_discrete(name = "Sex", labels = c("Female", "Male"))
```

ggplot2 horizontal barplots

- Easy to rotate a plot with `coord_flip`

```
ggplot(data = titanicData, aes(x = mySurvived, fill = sex)) + geom_bar() +  
  labs(x = "Survival Status",  
       title = "Bar Plot of Survival Status for Titanic Passengers") +  
  scale_x_discrete(labels = c("Person Died", "Person Survived")) +  
  scale_fill_discrete(name = "Sex", labels = c("Female", "Male")) +  
  coord_flip()
```



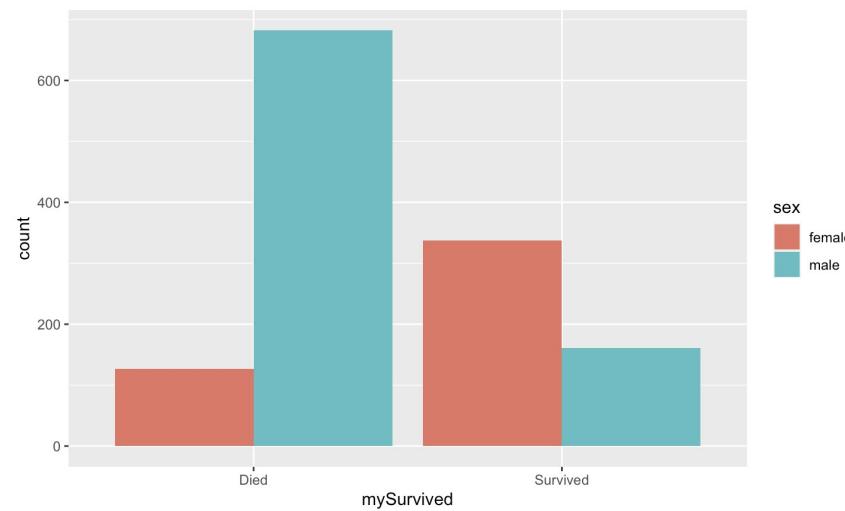
ggplot2 side-by-side barplots

- **Side-by-side barplot** created by via position aesthetic
 - `dodge` for side-by-side bar plot
 - `jitter` for continuous data with many points at same values
 - `fill` stacks bars and standardises each stack to have constant height
 - `stack` stacks bars on top of each other

ggplot2 side-by-side barplots

- Side-by-side barplot created by via position aesthetic

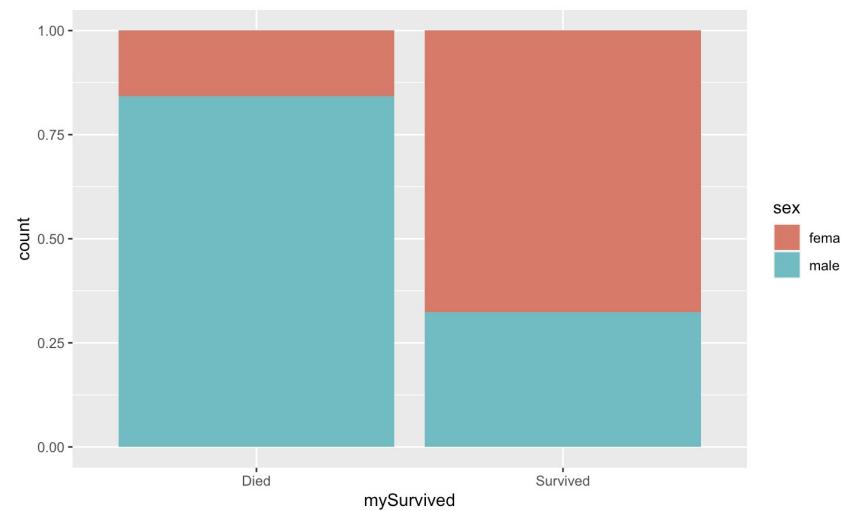
```
ggplot(data = titanicData, aes(x = mySurvived, fill = sex)) +  
  geom_bar(position = "dodge")
```



ggplot2 filled barplots

- `position = "fill"` stacks bars and standardises each stack to have constant height (especially useful with equal group sizes)

```
ggplot(data = titanicData, aes(x = mySurvived, fill = sex)) +  
  geom_bar(position = "fill")
```



ggplot2 faceting

How to create same plot for each `myEmbarked` value? Use **faceting!**

ggplot2 faceting

How to create same plot for each `myEmbarked` value? Use **faceting!**

`facet_wrap(~ var)` - creates a plot for each setting of `var`

- Can specify `nrow` and `ncol` or let R figure it out

ggplot2 faceting

How to create same plot for each `myEmbarked` value? Use **faceting!**

`facet_wrap(~ var)` - creates a plot for each setting of `var`

- Can specify `nrow` and `ncol` or let R figure it out

`facet_grid(var1 ~ var2)` - creates a plot for each combination of `var1` and `var2`

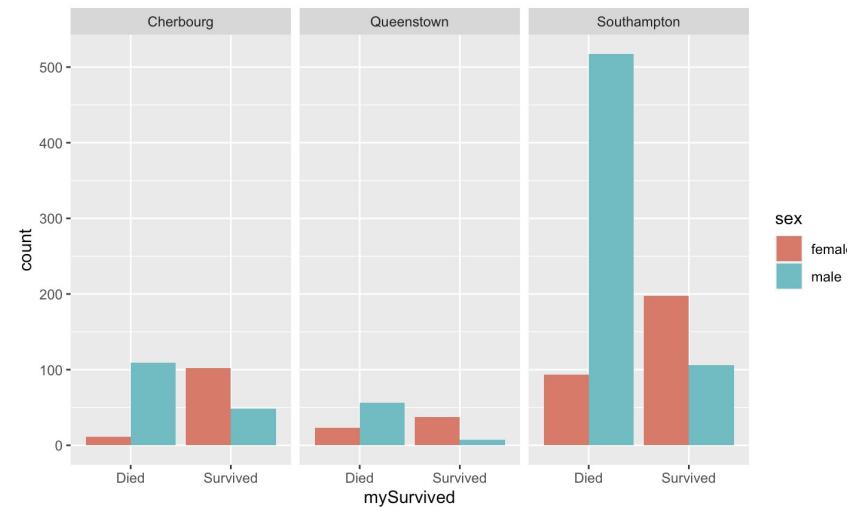
- `var1` values across rows
- `var2` values across columns
- Use `. ~ var2` or `var1 ~ .` to have only one row or column

ggplot2 faceting

How to create same plot for each `myEmbarked` value? Use **faceting!**

- `facet_wrap(~ var)` - creates a plot for each setting of `var`

```
ggplot(data = titanicData, aes(x = mySurvived)) +  
  geom_bar(aes(fill = sex), position = "dodge") +  
  facet_wrap(~ myEmbarked)
```



ggplot2 Plotting Recap

General `ggplot` things:

- Can set local or global `aes`
- Modify titles/labels by adding more layers
- Faceting (multiple plots) via `facet_grid` or `facet_wrap`
- Only need `aes` if setting a mapping value that is dependent on the data (or you want to create a custom legend!)

ggplot2 Plotting: Numeric Variables

Numeric variables - generally, describe distribution via a histogram or boxplot!

Same process:

- Create base object
- Add geoms
- Use aes to specify aspects of the plot

ggplot2 smoothed histogram

- **Kernel Smoother** - Smoothed version of a histogram
- Common `aes` values (from cheat sheet):

```
c + geom_density(kernel = "gaussian")
```

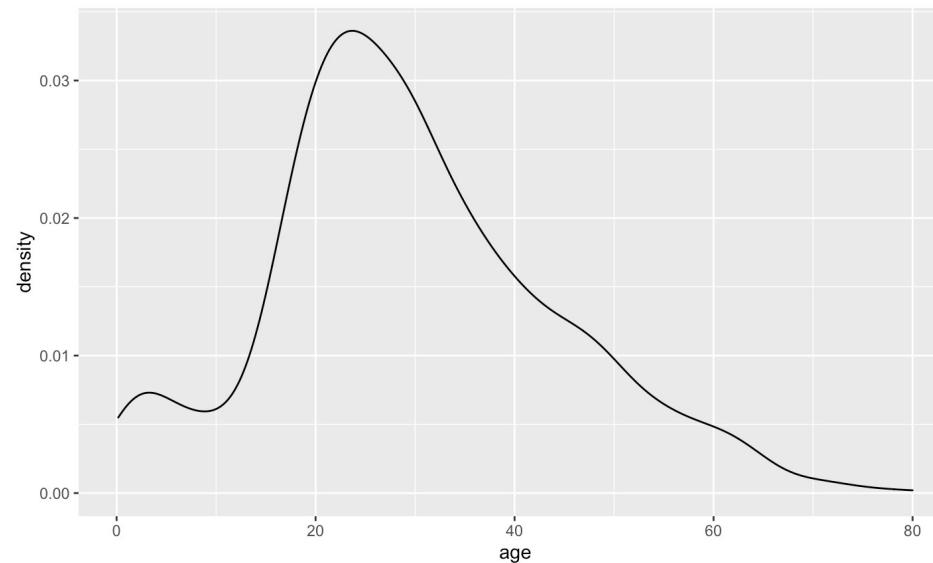
```
x, y, alpha, color, fill, group, linetype, size, weight
```

- Only `x` = is really needed

ggplot2 smoothed histogram

- Kernel Smoother - Smoothed version of a histogram

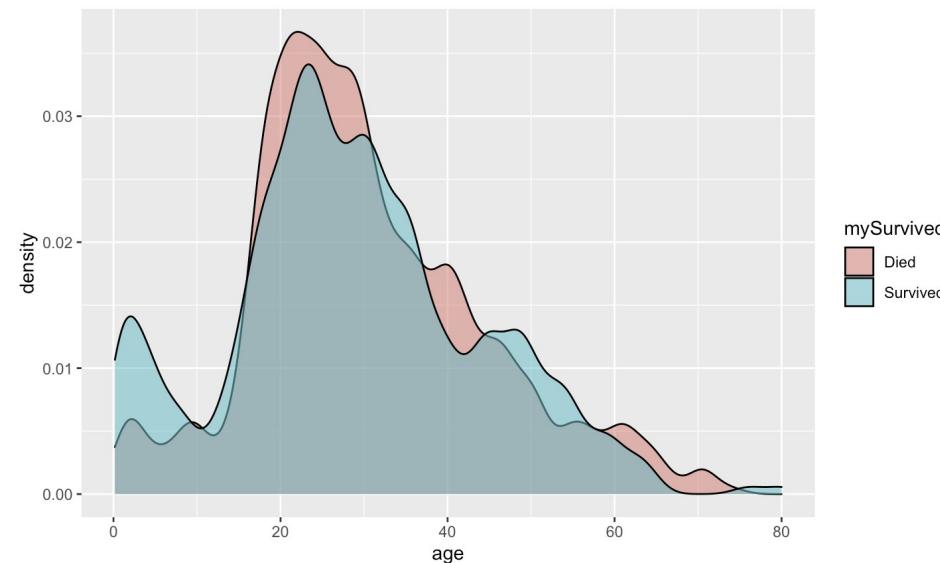
```
g <- ggplot(titanicData, aes(x = age))  
g + geom_density()
```



ggplot2 smoothed histogram

- Kernel Smoother - Smoothed version of a histogram
- fill a useful aesthetic!

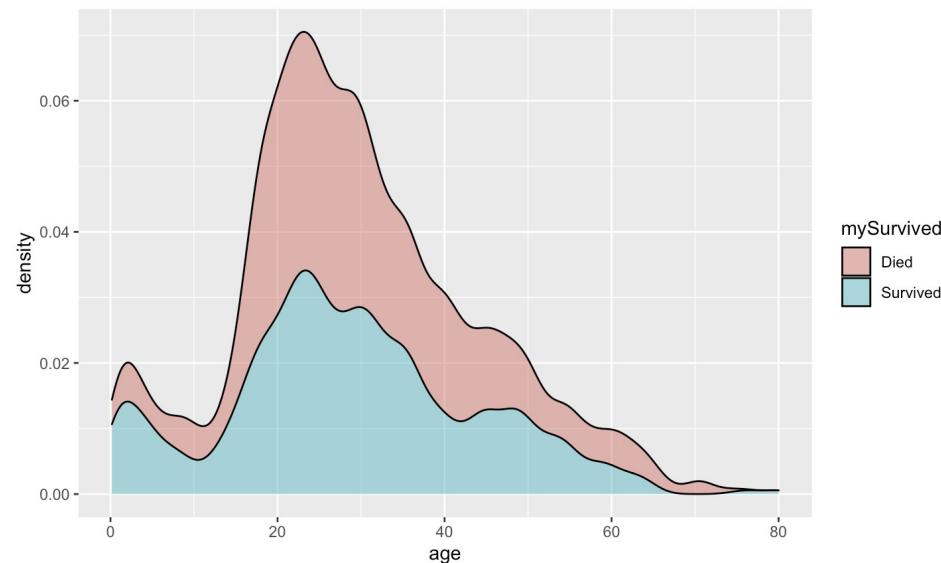
```
g + geom_density(adjust = 0.5, alpha = 0.5, aes(fill = mySurvived))
```



ggplot2 smoothed histogram

- Kernel Smoother - Smoothed version of a histogram
- recall position choices of dodge, jitter, fill, and stack

```
g + geom_density(adjust = 0.5, alpha = 0.5, position = "stack", aes(fill = mySurvived))
```



ggplot2 boxplots

- **Boxplot** - Provides the five number summary in a graph
- Common `aes` values (from cheat sheet):

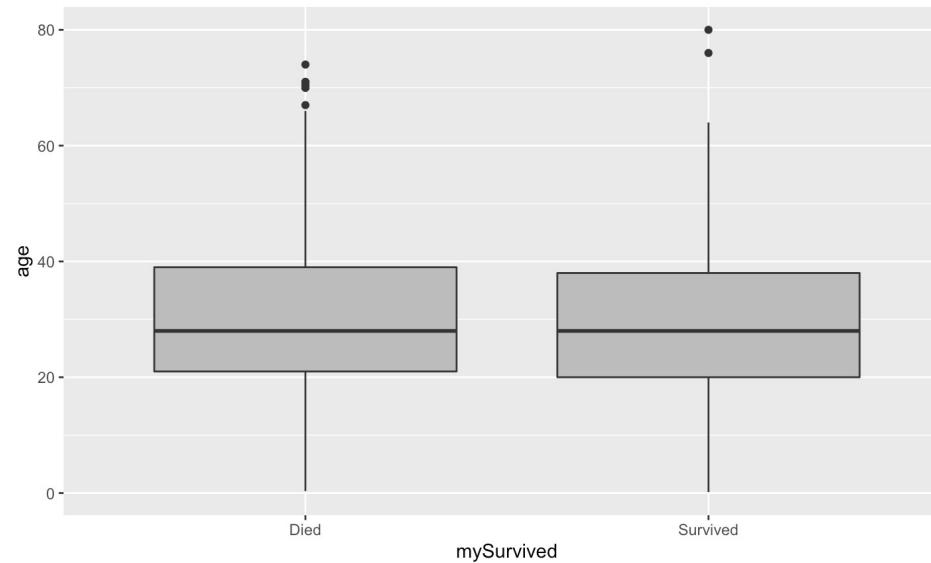
```
f + geom_boxplot()
```

```
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill,  
group, linetype, shape, size, weight
```

- Only `x =`, `y =` are really needed

ggplot2 boxplots

```
g <- ggplot(titanicData, aes(x = mySurvived, y = age))  
g + geom_boxplot(fill = "grey")
```

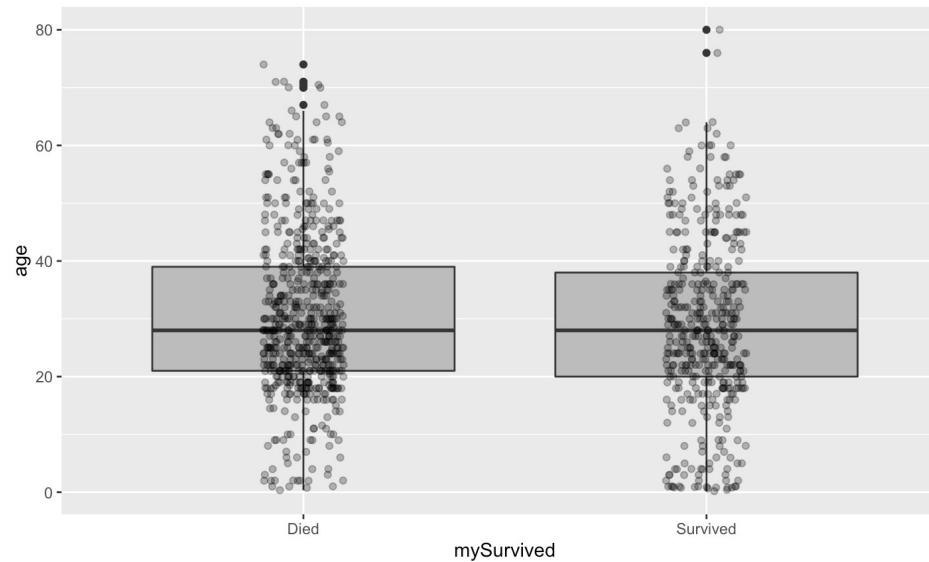


ggplot2 boxplots with points

- Can add data points (jittered) to see shape of data better (or use violin plot)

```
g +
```

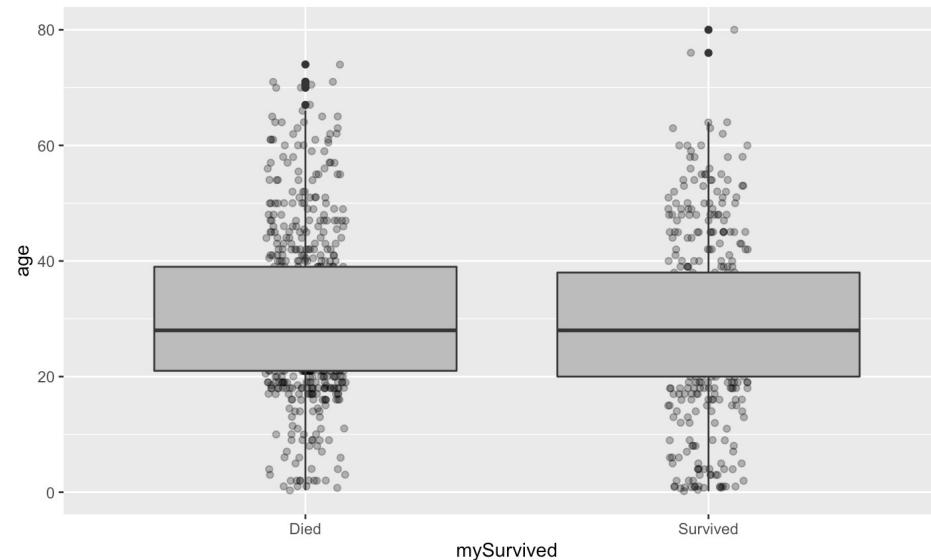
```
geom_boxplot(fill = "grey") +  
geom_jitter(width = 0.1, alpha = 0.3)
```



ggplot2 boxplots with points

- Order of layers important!

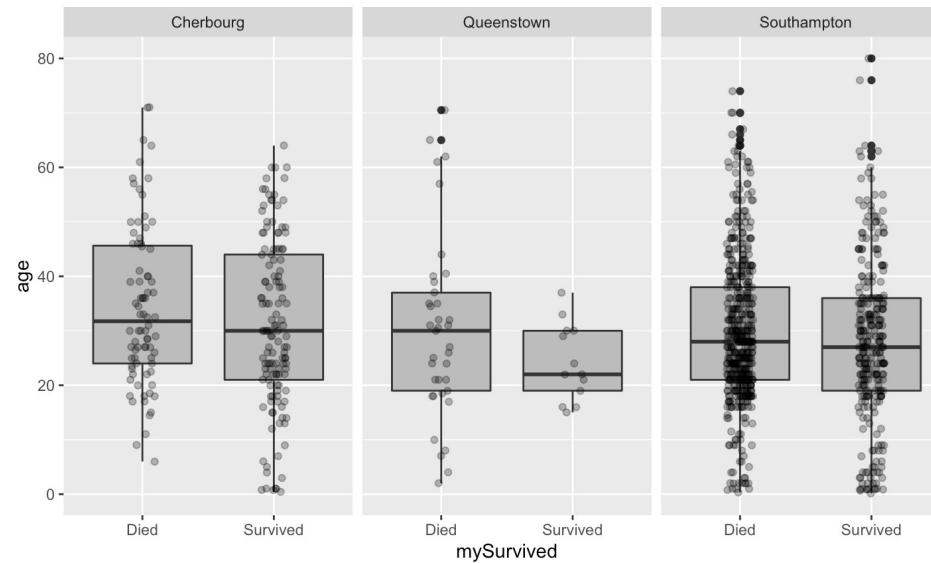
```
g +  
  geom_jitter(width = 0.1, alpha = 0.3) +  
  geom_boxplot(fill = "grey")
```



ggplot2 faceting

- Can facet easily!

```
g + geom_boxplot(fill = "grey") +
  geom_jitter(width = 0.1, alpha = 0.3) +
  facet_wrap(~ myEmbarked)
```



ggplot2 Plotting: Numeric variables

Recap!

Numeric variable - entries are a numerical value where math can be performed

Most common plots:

- Histogram (`geom_hist`), Density (`geom_density`)
- Boxplot (`geom_boxplot`), Violin plot (`geom_violin`)
- Scatter plot (`geom_point`), Smoothers (`geom_smooth`)
- Jittered points (`geom_jitter`)
- Text on plot (`geom_text`)

ggplot2 Plotting Recap

General `ggplot` things:

- Can set local or global `aes`
- Modify titles/labels by adding more layers
- Use either `stat` or `geom` layer
- Faceting (multiple plots) via `facet_grid` or `facet_wrap`
- Only need `aes` if setting a mapping value that is dependent on the data (or you want to create a custom legend!)
- `esquisse` is a [great package for exploring ggplot2!](#)

Exercises

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_5_Exercises.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_5_Exercises.html)

Partial Solution

[https://jbpost2.github.io/TeachingWithR/exercises
/TeachingR_Part_5_Solutions.html](https://jbpost2.github.io/TeachingWithR/exercises/TeachingR_Part_5_Solutions.html)