



Using R Markdown & the Tidyverse to Create Reproducible Research

Justin Post

Overview & R Markdown

Justin Post

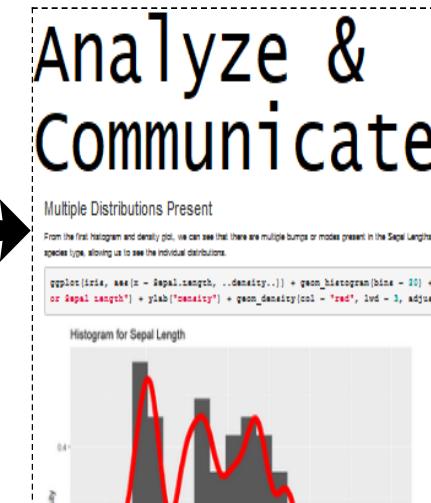
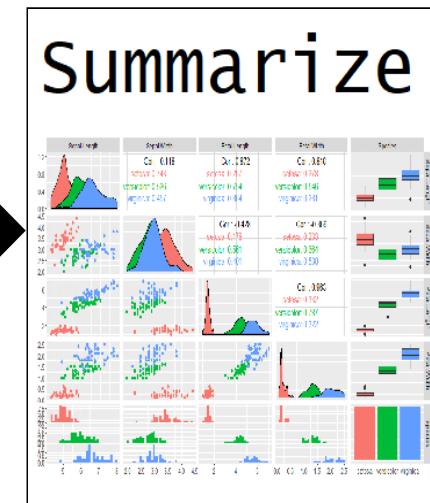
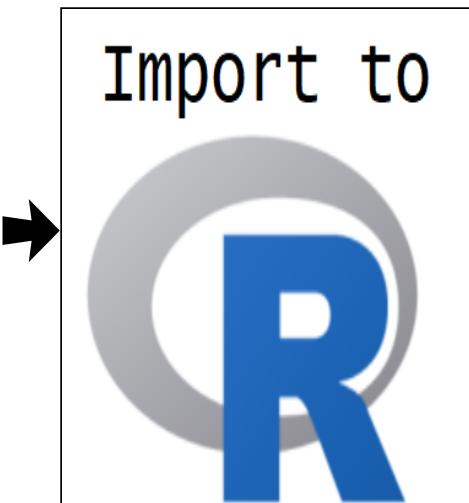
What is this course about?

Basic use of R for reading, manipulating, and summarizing data. With a focus on reproducibility!

What is this course about?

Basic use of R for reading, manipulating, and summarizing data. With a focus on reproducibility!

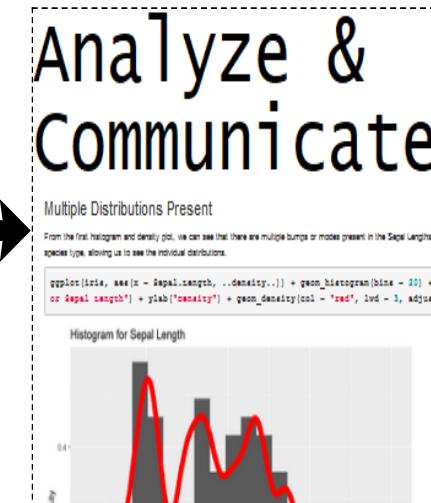
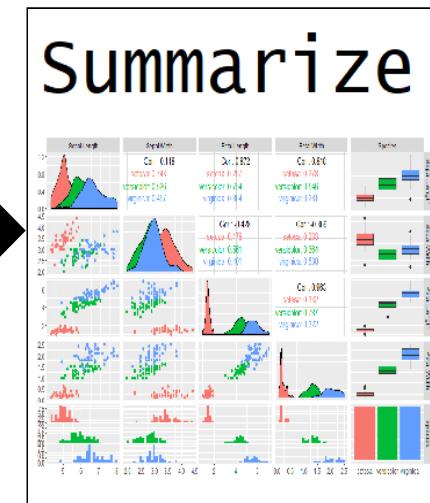
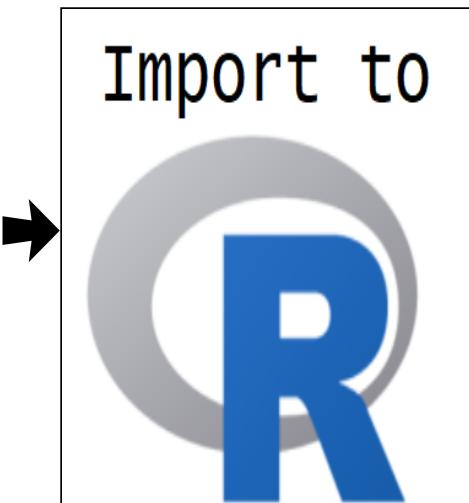
```
temp conc time percent
-1 -1 -1 45.9
1 -1 -1 60.6
-1 1 -1 57.5
1 1 1 Raw Data
-1
1 -1 1 58
-1 1 1 58.8
1 1 1 52.4
```



What is this course about?

Basic use of R for reading, manipulating, and summarizing data. With a focus on reproducibility!

```
temp conc time percent
-1 -1 -1 45.9
1 -1 -1 60.6
-1 1 -1 57.5
1 1 1 Raw Data
-1
1 -1 1 58
-1 1 1 58.8
1 1 1 52.4
```



Start with an example!

First Part of Course

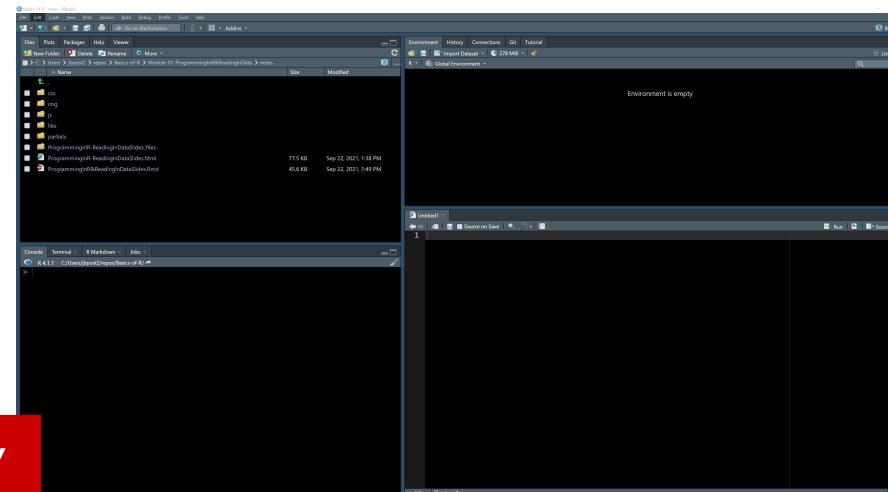


- Learn basics of R Markdown for literate programming
- Understand how R stores data
- Read external data into R

RStudio IDE

In RStudio, four main locations (easy to customize!)

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



Console

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

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

## [1] 10

10 * exp(3) #exp is exponential function

## [1] 200.8554

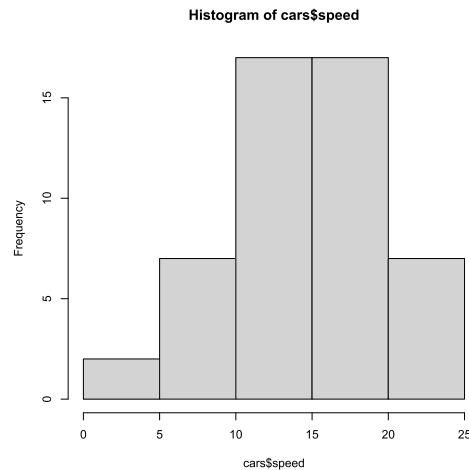
log(pi^2) #log is natural log by default

## [1] 2.28946
```

```
mean(cars$speed)
```

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

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

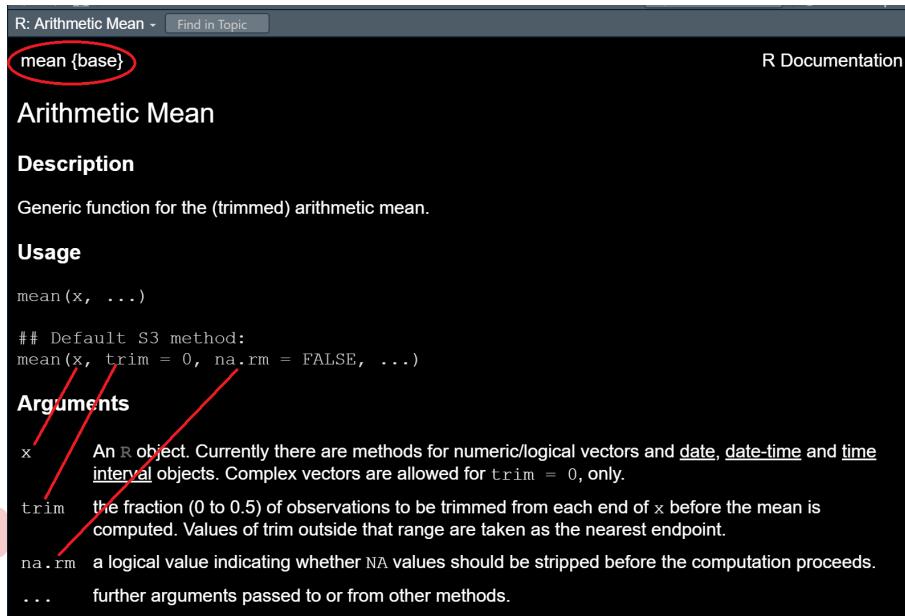


Files/Plots/Packages/Help

- Files (navigate through files)
- Created plots stored in `Plots` tab
 - Cycle through past plots
 - Easily save
- Packages (update and install)
- Documentation within RStudio via `help(...)`
 - Ex: `help(seq)`

Files/Plots/Packages/Help

- Files (navigate through files)
- Created plots stored in **Plots** tab
 - Cycle through past plots
 - Easily save
- Packages (update and install)
- Documentation within RStudio via **help(...)**
 - Ex: **help(seq)**



Files/Plots/Packages/Help

- Files (navigate through files)
- Created plots stored in **Plots** tab
 - Cycle through past plots
 - Easily save
- Packages (update and install)
- Documentation within RStudio via **help(...)**
 - Ex: **help(seq)**

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed ~~as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.~~

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

[Package `base` version 4.1.2 Index]

Environment

- We 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

- Built-in objects exist like `letters` and `cars` don't show automatically

```
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"

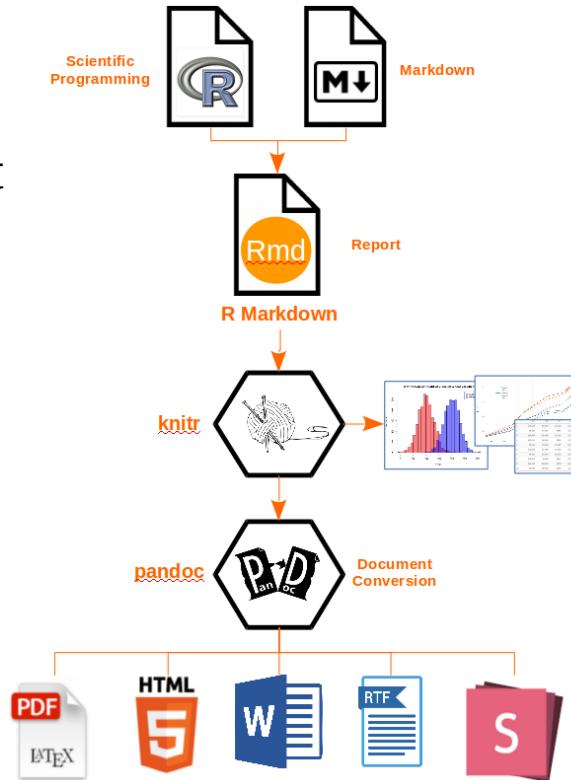
head(cars, n = 3)

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

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

Scripting and Viewing Window

- Usually want to keep code for later use!
- Traditionally code in a 'script' and save script
 - Instead we'll use R Markdown
 - A file with extension `.Rmd`
- Let's start with R Markdown!



<http://applied-r.com/>

Documenting with Markdown

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)

Markdown Languages

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

Markdown Languages

- May have heard of HTML (HyperText Mark-up Language)
 - Write plain text with *tags* that the browser interprets and renders
- R Markdown is a specific markup language
 - Easier syntax
 - Not as powerful
- Any plain text file can be used (.Rmd extension associates it with R Studio)

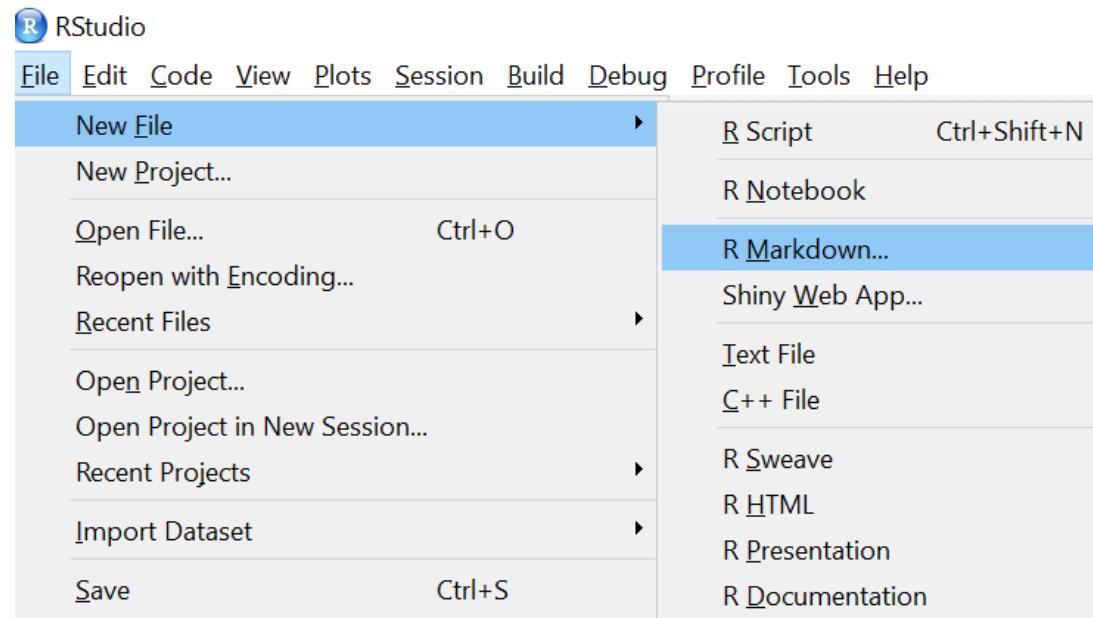
Markdown Languages

- May have heard of HTML (HyperText Mark-up Language)
 - Write plain text with *tags* that the browser interprets and renders
- R Markdown is a specific markup language
 - Easier syntax
 - Not as powerful
- Any plain text file can be used (`.Rmd` extension associates it with R Studio)
- Easy to create many types of documents in R Markdown!



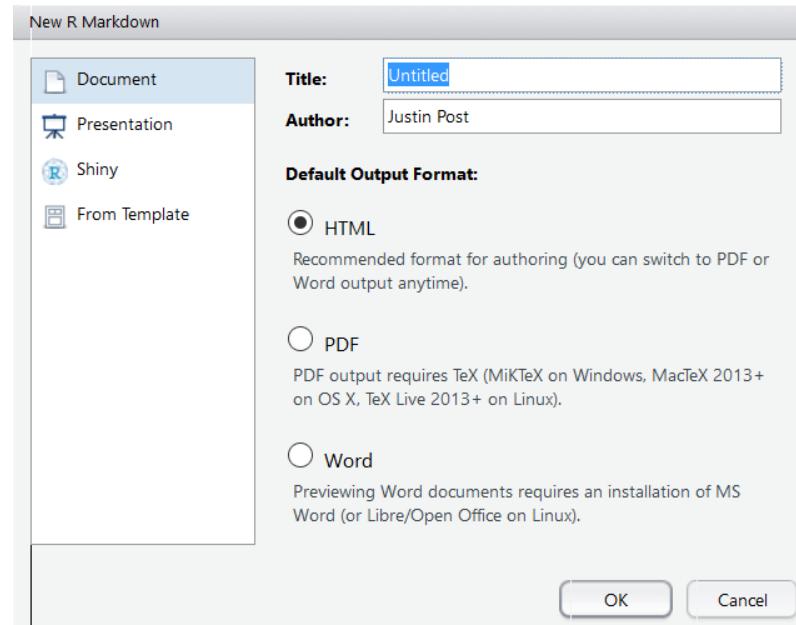
Creating an R Markdown Document

- R Studio makes it easy!



Choosing the Output Type

- Many commonly used document types can be created



.Rmd Files

R Markdown files (.Rmd) contain three important types of content:

1. (Optional) YAML header surrounded by ---s
 - Defines meta data about the document
2. Chunks of R code
 - Code that may evaluate and produce output when *knitting* the document
3. Text mixed with simple text formatting instructions (R Markdown syntax)

YAML Header

- Defines settings for the creation process

```
---
```

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

- CTRL/CMD + Shift + k or the **Knit** button creates a document via this info
- [Great examples of options here](#)

Creating PDF Output

Change the `output` to `pdf_document`

- If you have a `Tex` engine on their computer (such as `MikTex`), good to go
- If not, easiest to do the following first:
 1. Install the `tinytex` package
 2. Run `library(tinytex)`
 3. Run `install_tinytex()`
 4. Restart R

Markdown Syntax

- `# R Markdown` → First level header
- `## Next` → Second level header

Markdown Syntax

- `# R Markdown` → First level header
- `## Next` → Second level header
- `<http://rmarkdown.rstudio.com>` → A hyperlink: <http://rmarkdown.rstudio.com>
- `[Cheat Sheet link](https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf)` → [Cheat Sheet link](https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf)

Markdown Syntax

- `# R Markdown` → First level header
- `## Next` → Second level header
- `<http://rmarkdown.rstudio.com>` → A hyperlink: <http://rmarkdown.rstudio.com>
- `[Cheat Sheet link](https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf)` → Cheat Sheet link
- `**Knit**` or `__Knit__` → Bold font (**Knit**)
- `*italic*` or `_italic_` → Italic font (*italic*)
- `*__both__*` → Bold and italic (***both***)

Markdown Syntax

- Can do lists (be careful with spaces & returns)
 - Indent sub lists four spaces

```
- unordered list
- item 2
  + sub-item 1
  + sub-item 2
```

```
1. ordered list
2. item 2
  a. sub-item 1
  b. sub-item 2
```

- See the cheatsheet [here](#) or [here](#)

```
• unordered list
• item 2
  ◦ sub-item 1
  ◦ sub-item 2
```

```
1. ordered list
2. item 2
  a. sub-item 1
  b. sub-item 2
```

Code Chunks

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

# Code Chunks

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

- Start code chunk by typing out the syntax or with CTRL/CMD + Alt/Option + I
- Can execute code in RStudio as you are writing
- Code is executed sequentially when document is created

Code Chunks

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

Can specify behavior of each code chunks via **global** or **local** chunk options:

- Hide/show code: `echo = FALSE/TRUE`
- `eval = TRUE/FALSE`
- Eval, not show code or output: `include = TRUE/FALSE`
- `message = TRUE/FALSE` and `warning = TRUE/FALSE`

# Code Chunks

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

Global chunk options:

- Usually have a **setup** code chunk (with `include = FALSE`)
- Defines global behavior for all chunks

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

Code Chunks

```
knitr::include_graphics("img/chunk.png")
```

Using chunks is generally the best way to include an image.

Code chunk options often used

- `fig.align = 'center'`
- `out.width = "500px"`
- `echo = FALSE`

Inline R Code

R code can be evaluated inline!

- Begin with a single back-tick followed by `r`
- End with another back-tick

Ex: Iris has `r length(iris[,1])` observations → Iris has 150 observations

Recap!

- R Markdown allows for easily reproducible analyses and documenting of thought processes
 - YAML header
 - Plain text with R Markdown syntax
 - Code chunks
- [Cheat Sheet link](#) great for getting started
- Everything you could want to know about R Markdown in [R Markdown: The Definitive Guide](#)

Let's Practice

- Take the raw text [here](#)
- Create the HTML output [here](#)

Guidance:

- Create a new .Rmd file and replace its test with the raw text (copy/paste)
- Knit the document as is to start
- Add headers (the `dplyr`, `ggplot2`, `readr`, and `tidyverse` sections are second level headers), reknit
- [Update YAML header](#) to add code folding, reknit
- Update the global chunk options (`echo`, `eval` should be `TRUE`, `message` should be `FALSE`), reknit
- Modify local chunk options on the images to suppress code, reknit
- [Add in markdown syntax](#) (links, code font, etc.), reknit
- [Add tabssets](#), reknit

Data Storage in R

Justin Post

R Objects and Classes

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

```
vec <- c(1, 4, 10)
vec
## [1] 1 4 10
```

R Objects and Classes

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

```
vec <- c(1, 4, 10)
vec
## [1] 1 4 10

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 ...

str(vec)

## num [1:3] 1 4 10
```

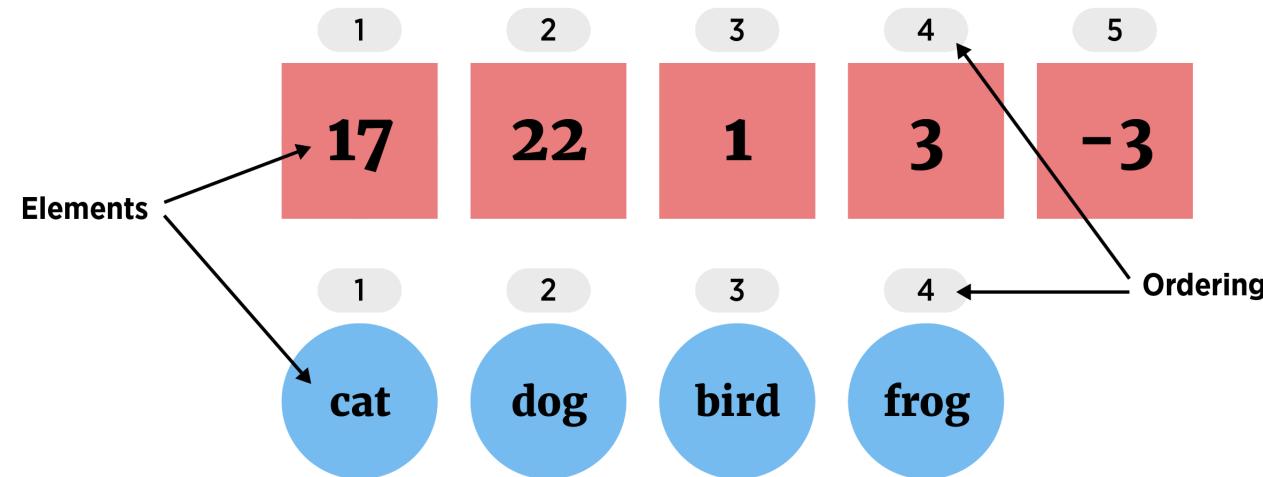
Data Objects

- 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
nd	Array	

Vector

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



- Elements must be same 'type'
 - numeric, character, or logical

Vector

(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 = 5, length = 5)
#same result with different inputs:
v <- seq(from = 1, to = 5, by = 1)
```

```
v
```

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

```
str(v)
```

```
## num [1:5] 1 2 3 4 5
```

:

to Create a Sequence

1:20

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

- R does element-wise math on vectors

1:20/20

```
## [1] 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75  
## [16] 0.80 0.85 0.90 0.95 1.00
```

1:20 + 1

```
## [1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
```

Slicing Vectors

- Return elements using square brackets `[]`

```
letters #built-in vector  
## [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"  
letters[1] #R starts counting at 1!  
## [1] "a"  
letters[26]  
## [1] "z"
```

Slicing Vectors

- Return elements using square brackets `[]`
- Can 'feed' in a vector of indices to `[]`

```
letters[1:4]
## [1] "a" "b" "c" "d"

letters[c(5, 10, 15, 20, 25)]
## [1] "e" "j" "o" "t" "y"

x <- c(1, 2, 5)
letters[x]

## [1] "a" "b" "e"
```

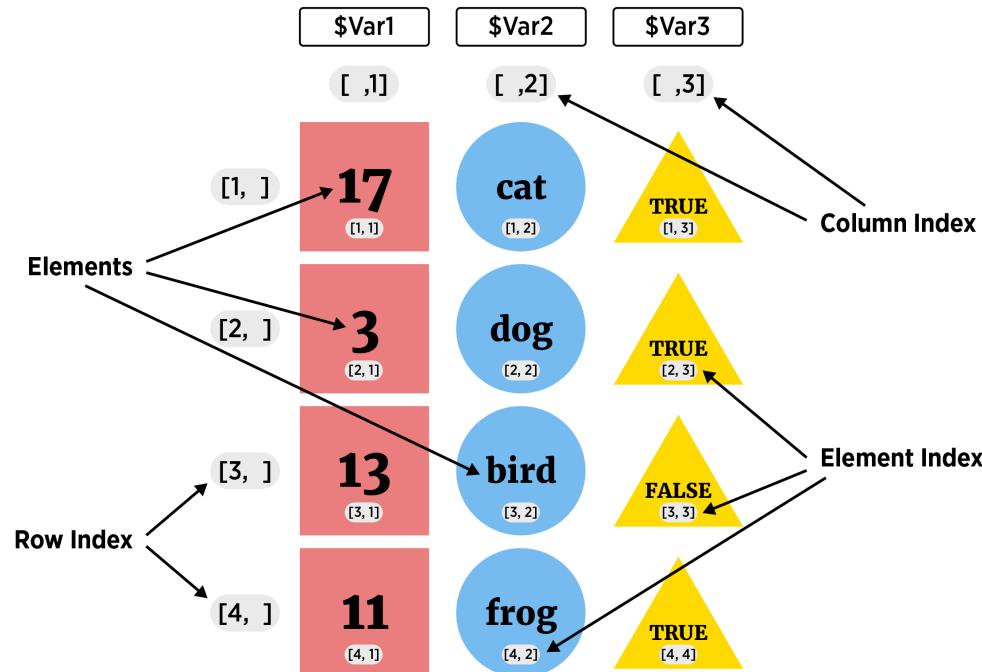
Slicing Vectors

- Return elements using square brackets `[]`
- Can 'feed' in a vector of indices to `[]`
- Use negative indices to return without

```
letters[-(1:4)]  
## [1] "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w"  
## [20] "x" "y" "z"  
  
x <- c(1, 2, 5)  
letters[-x]  
## [1] "c" "d" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v"  
## [20] "w" "x" "y" "z"
```

Building Blocks for Data Frames

- Columns of a data frame are vectors



Data Frame

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
myDF <- data.frame(x, y, z)
myDF
```

```
##   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

Data Frame (2D data structure)

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

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

```
##   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

Slicing a Data Frame

- Use square brackets with a comma [,]
 - Index rows (prior to the comma) then columns (after the comma)

```
myDF
```

```
##   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
```

```
myDF[c(2, 4), ]
```

```
##   char data1 data2
## 2   b     3    11
## 4   d    -1    13
```

```
myDF[, 1]
```

```
## [1] "a" "b" "c" "d" "e" "f"
```

```
myDF[2, ]
```

```
##   char data1 data2
## 2   b     3    11
```

```
myDF[2, 1]
```

```
## [1] "b"
```

Slicing a Data Frame

- Can use columns names to subset

```
myDF[ , c("char", "data1")]
```

```
##   char data1
## 1     a     1
## 2     b     3
## 3     c     4
## 4     d    -1
## 5     e     5
## 6     f     6
```

Slicing a Data Frame

- Dollar sign allows easy access to a single column!

```
myDF$char  
## [1] "a" "b" "c" "d" "e" "f"  
  
myDF$data1  
## [1] 1 3 4 -1 5 6
```

Slicing a Data Frame

- Dollar sign allows easy access to a single column!
- Most used method for accessing a single variable
- RStudio fills in options.
 - Type `mydf$`
 - If no choices - hit tab
 - Hit tab again to choose

Recap!

Data Frame (2D data structure)

- Collection (list) of **vectors** of the same *length*
- Create with `data.frame()` function
- Access with `[,]` or `$`
- Perfect for most data sets!
- Most functions that read 2D data store it as a `data frame` (or `tibble` - a special data frame covered shortly)

Let's Practice

We'll add to our `.Rmd` file from the previous activity

- Download the prompts to add to our markdown document [here](#)

Guidance:

- Copy and paste the text from above into the bottom of the document, reknit
- Add to the code chunks, evaluating in the notebook
- Reknit occasionally to check the output

The tidyverse

Justin Post

Where Do Our Objects & Functions Come From?

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

Basic access via

- Atomic vectors - `x[]`
- Data Frames - `x[,]` or `x$name`
- Lists - `x[]`, `x[[]]`, or `x$name`

Where Do Our Objects & Functions Come From?

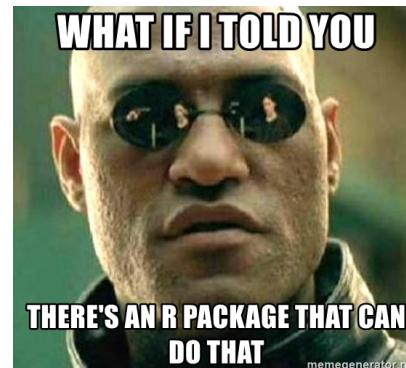
When you open R a few packages are loaded

- R package:
 - Collection of functions/objects/datasets/etc.

Where Do Our Objects & Functions Come From?

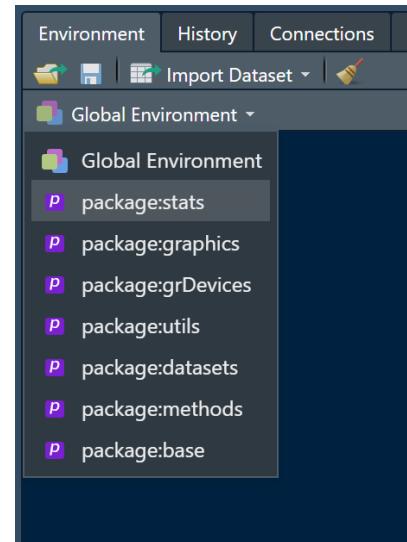
When you open R a few packages are loaded

- R package:
 - Collection of functions/objects/datasets/etc.
- Packages exist to do almost anything
 - List of CRAN approved packages
 - Plenty of other packages on places like GitHub



Where Do Our Objects & Functions Come From?

- Packages loaded automatically



- `base` package has `c()`, `data.frame()`, `list()`, ...

Base R vs Tidyverse

- Everything done so far uses Base R

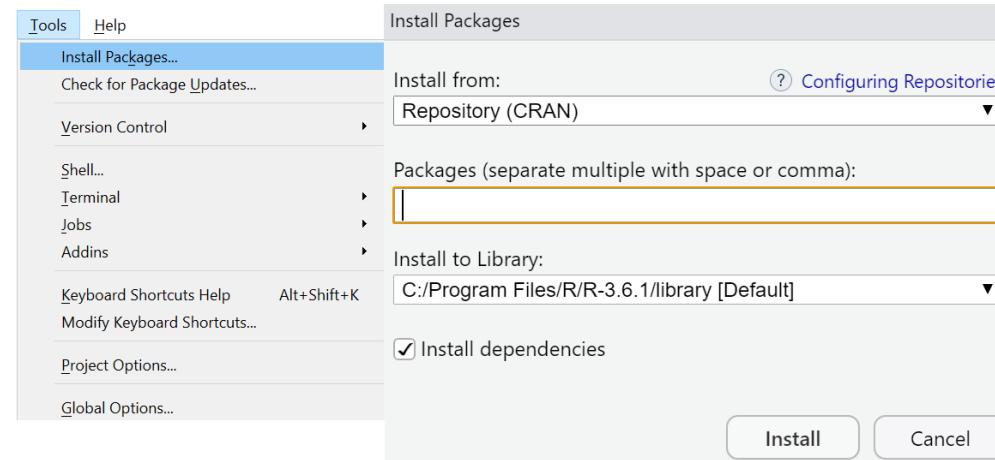
Coherent and opinionated framework for common data tasks:

- TidyVerse
 - data importing (`readr`, `readxl`, `haven`, `DBI`)
 - data manipulation (`dplyr`, `tidyverse`)
 - plotting (sort of) (`ggplot2`)
 - ...

Installing an R Package

- First time using a package
 - Must **install package** (download files to your computer)
 - Can use code, menus, or Packages tab

```
install.packages("dplyr")
```



Accessing a Package in Your R Session

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

```
library("dplyr")
```

Accessing a Package in Your R Session

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

```
library("dplyr")
```

- See everything from that package:

```
ls("package:dplyr")
```

## [1] "%>%"	"across"	"add_count"
## [4] "add_count_"	"add_row"	"add_rownames"
## [7] "add_tally"	"add_tally_"	"all_equal"
## [10] "all_of"	"all_vars"	"anti_join"
## [13] "any_of"	"any_vars"	"arrange"
## [16] "arrange_"	"arrange_all"	"arrange_at"
## [19] "arrange_if"	"as.tbl"	"as_data_frame"
## [22] "as_label"	"as_tibble"	"auto_copy"
## [25] "band_instruments"	"band_instruments2"	"band_members"
## [28] "bench_tbls"	"between"	"bind_cols"
## [31] "bind_rows"	"c_across"	"case_when"
## [34] "bind_tf"	"check_dbplyr"	"coalesce"
## [37] "bind_tf2"	"collect"	"combine"
## [40] "bind_tf3"	"compare_tbls"	"compare_tbls2"
## [43] "bind_tf4"	"contains"	"copy_to"

Calling From a Library

- 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         6.9         3.1          5.1          2.3 virginica
## 17         5.8         2.7          5.4          1.9 virginica
## 18         6.5         3.0          5.5          2.0 virginica
## 19         6.5         3.0          5.6          2.2 virginica
## 20         6.5         3.0          5.5          2.4 virginica
## 21         6.7         3.0          5.4          2.1 virginica
## 22         6.9         3.1          5.1          2.3 virginica
```

Reading Raw Data Into R

- Read in **raw** data using the `tidyverse` via `readr`, `readxl`, `haven`, and `DBI`

Reading Raw Data Into R

- Read in **raw** data using the `tidyverse` via `readr`, `readxl`, `haven`, and `DBI`

Plan:

- Read 'clean' delimited data
- Read Excel data
- See an example of connecting to a database

Reading Delimited Data

- 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
Δ 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	24.7	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 Wegner

General: usually .txt or .dat

Reading Delimited Data

`baseR utils` package and `tidyverse readr` package function and purpose:

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

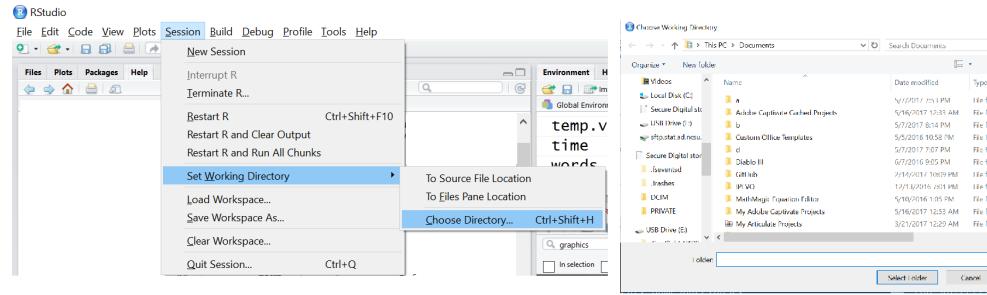
Working Directory

- Let's read in the '**neuralgia.csv**' file
 - *csv* implies comma separated value (sometimes semi-colon)
- By default, R looks in the **working directory** for the file

```
getwd()  
## [1] "C:/Users/jbpost2/repos/R4Reproducibility/slides"
```

Working Directory

- Can change *working directory* via code or menus



```
setwd("C:/Users/jbpost2/repos/R4Reproducibility/datasets")
#or
setwd("C:\\\\Users\\\\jbpost2\\\\repos\\\\R4Reproducibility\\\\datasets")
```

Reading a .csv File

Common arguments to `read_csv()`:

```
read_csv(  
  file,  
  col_names = TRUE,  
  na = c("", "NA"),  
  skip = 0,  
  col_types = NULL,  
  guess_max = min(1000, n_max),  
)
```

Reading a .csv File

With `neuralgia.csv` file in the working directory:

```
neuralgiaData <- read_csv("neuralgia.csv")  
  
neuralgiaData  
  
## # A tibble: 60 × 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  
## # ... with 55 more rows
```

Reading a .csv File

- Use full local path

```
neuralgiaData <- read_csv("C:/Users/jbpost2/repos/R4Reproducibility/slides/data/neuralgia.csv")
```

Reading a .csv File

- Use full local path

```
neuralgiaData <- read_csv("C:/Users/jbpost2/repos/R4Reproducibility/slides/data/neuralgia.csv")
```

- 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  
## # ... with 55 more rows
```

tibbles

What kind of object does `read_csv()` create?

```
class(neuralgiaData)
## [1] "spec_tbl_df" "tbl_df"       "tbl"          "data.frame"

str(neuralgiaData)

## #> spec_tbl_df [60 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## #>   $ Treatment: chr [1:60] "P" "B" "P" "P" ...
## #>   $ Sex      : chr [1:60] "F" "M" "F" "M" ...
## #>   $ Age      : num [1:60] 68 74 67 66 67 77 71 72 76 71 ...
## #>   $ Duration : num [1:60] 1 16 30 26 28 16 12 50 9 17 ...
## #>   $ Pain     : chr [1:60] "No" "No" "No" "Yes" ...
## #> - attr(*, "spec")=
## #>   .. cols(
## #>     ..   Treatment = col_character(),
## #>     ..   Sex = col_character(),
## #>     ..   Age = col_double(),
## #>     ..   Duration = col_double(),
## #>     ..   Pain = col_character()
## #>     .. )
## #> - attr(*, "problems")=<externalptr>
```

tibbles

- `tibbles` are the main object the tidyverse works with

tibbles

- `tibbles` are the main object the tidyverse works with
 - Fancy printing!
 - Checking column type is a basic data validation step

```
neuralgiaData  
## # A tibble: 60 × 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  
## # ... with 55 more rows
```

- Behavior slightly different than a standard data frame. No simplification!

tibbles

- tibbles do not simplify

```
neuralgiaData[ ,1]  
  
## # A tibble: 60 x 1  
##   Treatment  
##   <chr>  
## 1 P  
## 2 B  
## 3 P  
## 4 P  
## 5 B  
## # ... with 55 more rows
```

```
neuralgiaData2 <- as.data.frame(neuralgiaData)  
neuralgiaData2[ ,1]  
  
## [1] "P"  "B"  "P"  "P"  "B"  "B"  "A"  "B"  "B"  "A"  "A"  "A"  "B"  "A"  "P"  "A"  "P"  
## [20] "B"  "B"  "A"  "A"  "B"  "P"  "B"  "B"  "P"  "P"  "A"  "A"  "B"  "B"  "B"  "A"  "P"  
## [39] "B"  "P"  "P"  "P"  "A"  "B"  "A"  "P"  "P"  "A"  "B"  "P"  "P"  "P"  "B"  "A"  "P"  
## [58] "A"  "B"  "A"
```

tibbles

- Use either `dplyr::pull()` or `$` to return a vector

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

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

Reading Delimited Data with `readr`

- Reading *clean* delimited data pretty easy with the tidyverse!

Type of Delimiter	<code>readr</code> Function
Comma	<code>read_csv()</code>
Semicolon (, for decimal)	<code>read_csv2()</code>
Tab	<code>read_tsv()</code>
General	<code>read_delim()</code>
White Space	<code>read_table()</code>

- Let's read in the 'chemical.txt' file (space delimited) with `read_table()`
- Common arguments to `read_table()` are the same as `read_csv()`

Reading Space Delimited Data

- Let's read in the 'chemical.txt' file (space delimited) with `read_table()`

```
read_table("https://www4.stat.ncsu.edu/~online/datasets/chemical.txt")  
  
## # A tibble: 19 x 4  
##   temp conc time percent  
##   <dbl> <dbl> <dbl>    <dbl>  
## 1 -1    -1    -1     45.9  
## 2 1     -1    -1     60.6  
## 3 -1    1     -1     57.5  
## 4 1     1     -1     58.6  
## 5 -1    -1     1     53.3  
## 6 1     -1     1     58  
## 7 -1    1     1     58.8  
## 8 1     1     1     52.4  
## 9 -2    0     0     46.9  
## 10 2    0     0     55.4  
## 11 0    -2    0     55  
## 12 0    2     0     57.5  
## 13 0    0    -2     56.3  
## 14 0    0     2     58.9  
## 15 0    0     0     56.9  
## 16 2    -3    0     61.1  
## 17 2    -3    0     62.9
```

Reading Tab Delimited Data

- Let's read in the 'crabs.txt' file (tab delimited) with `read_tsv()`

```
read_tsv("https://www4.stat.ncsu.edu/~online/datasets/crabs.txt")  
  
## # A tibble: 173 x 6  
##   color spine width satell weight     y  
##   <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>  
## 1     3     3  28.3      8    3050     1  
## 2     4     3  22.5      0    1550     0  
## 3     2     1  26.0      9    2300     1  
## 4     4     3  24.8      0    2100     0  
## 5     4     3  26.0      4    2600     1  
## # ... with 168 more rows
```

Reading Generic Delimited Data

- Let's read in the '**umps2012.txt**' file ('>' delimited) with `read_delim()`
 - `read_delim()` requires a `delim` argument

Reading Generic Delimited Data

- Let's read in the '**umps2012.txt**' file ('>' delimited) with `read_delim()`
 - `read_delim()` requires a `delim` argument
- In **umps2012.txt** data file, no column names provided!
 - Use `col_names` argument: Either TRUE, FALSE or a character vector of column names.
 - When specifying a character vector, `read_delim()` automatically starts reading first row of data

Reading Generic Delimited Data

- Let's read in the 'umps2012.txt' file ('>' delimited) with `read_delim()`

```
read_delim("https://www4.stat.ncsu.edu/~online/datasets/umps2012.txt",
            delim = ">",
            col_names = c("Year", "Month", "Day", "Home", "Away", "HPUmpire"))

## # A tibble: 2,359 x 6
##   Year Month Day Home Away HPUmpire
##   <dbl> <dbl> <dbl> <chr> <chr> <chr>
## 1 2012     4    12 MIN   LAA D.J. Reyburn
## 2 2012     4    12 SD    ARI Marty Foster
## 3 2012     4    12 WSH   CIN Mike Everitt
## 4 2012     4    12 PHI   MIA Jeff Nelson
## 5 2012     4    12 CHC   MIL Fieldin Culbreth
## # ... with 2,354 more rows
```

Reading Fixed Field & Tricky Non-Standard Data

- `read_fwf()`
 - reads in raw data where entries are very structured
- `read_file()`
 - reads an entire file into a single string
- `read_lines()`
 - reads a file into a character vector with one element per line
- Usually parse the last two with `regular expressions` :(

Recap!

Type of Delimiter	readr Function
Comma	read_csv()
Semicolon (, for decimal)	read_csv2()
Tab	read_tsv()
General	read_delim()
White Space	read_table()

Common arguments:

```
file,  
col_names = TRUE,  
na = c("", "NA"),  
skip = 0,  
col_types = NULL,  
guess_max = min(1000, n_max),
```

Excel Data



Excel data refers to a `.xls` or `.xlsx` file

- `readxl` package does not load with `tidyverse` but is part of it!

Excel Data



Excel data refers to a `.xls` or `.xlsx` file

- `readxl` package does not load with `tidyverse` but is part of it!
- `read_excel()` function can read both types of excel data files
 - Can't pull from web though!
- Read in `censusEd.xlsx`

read_excel()

```
#install package if necessary  
library(readxl)
```

read_excel()

```
#install package if necessary
library(readxl)
#reads first sheet by default
edData <- read_excel("censusEd.xlsx")
```

edData

```
## # A tibble: 3,198 x 42
##   Area_name    STCOU EDU010187F EDU010187D EDU010187N1 EDU010187N2 EDU010188F
##   <chr>        <chr>     <dbl>      <dbl>      <chr>      <chr>      <dbl>
## 1 UNITED STATES 00000         0  40024299  0000       0000          0
## 2 ALABAMA        01000         0   733735  0000       0000          0
## 3 Autauga, AL    01001         0    6829  0000       0000          0
## 4 Baldwin, AL    01003         0   16417  0000       0000          0
## 5 Barbour, AL    01005         0   5071  0000       0000          0
## # ... with 3,193 more rows, and 35 more variables: EDU010188D <dbl>,
## #   EDU010188N1 <chr>, EDU010188N2 <chr>, EDU010189F <dbl>, EDU010189D <dbl>,
## #   EDU010189N1 <chr>, EDU010189N2 <chr>, EDU010190F <dbl>, EDU010190D <dbl>,
## #   EDU010190N1 <chr>, EDU010190N2 <chr>, EDU010191F <dbl>, EDU010191D <dbl>,
## #   EDU010191N1 <chr>, EDU010191N2 <chr>, EDU010192F <dbl>, EDU010192D <dbl>,
## #   EDU010192N1 <chr>, EDU010192N2 <chr>, EDU010193F <dbl>, EDU010193D <dbl>,
## #   EDU010193N1 <chr>, EDU010193N2 <chr>, EDU010194F <dbl>, ...
```

Dealing with Excel Sheets

- Can look at sheets available with `excel_sheets()`

```
excel_sheets("censusEd.xlsx")  
## [1] "EDU01A" "EDU01B" "EDU01C" "EDU01D" "EDU01E" "EDU01F" "EDU01G" "EDU01H"  
## [9] "EDU01I" "EDU01J"
```

- Specify sheet with name or integers (or `NULL` for 1st) using `sheet =`

```
read_excel("censusEd.xlsx", sheet = "EDU01D")  
## # A tibble: 3,198 x 42  
##   Area_name      STCOU EDU264190F EDU264190D EDU264190N1 EDU264190N2 EDU280190F  
##   <chr>        <chr>    <dbl>     <dbl>     <chr>        <chr>        <dbl>  
## 1 UNITED STATES 00000         0     4187099 0000       0000          0  
## 2 ALABAMA       01000         0      57284 0000       0000          0  
## 3 Autauga, AL  01001         0       604 0000       0000          0  
## 4 Baldwin, AL  01003         0      1761 0000       0000          0  
## 5 Barbour, AL  01005         0       466 0000       0000          0  
## # ... with 3,193 more rows, and 35 more variables: EDU280190D <dbl>,  
## #   EDU280190N1 <chr>, EDU280190N2 <chr>, EDU282190F <dbl>, EDU282190D <dbl>,  
## #   EDU282190N1 <chr>, EDU282190N2 <chr>, EDU284190F <dbl>, EDU284190D <dbl>,  
## #   EDU284190N1 <chr>, EDU284190N2 <chr>, EDU299190F <dbl>, EDU299190D <dbl>,  
## #   90N2 <chr>, EDU300200F <dbl>, EDU300200D <dbl>, 00N2 <chr>, EDU310200F <dbl>, EDU310200D <dbl>,  
## #   00N2 <chr>, EDU312200F <dbl>, ...
```

SAS Data



SAS data refers to a `.sas7bdat` file

- `haven` package does not load with `tidyverse` but is part of it!

SAS Data



SAS data refers to a `.sas7bdat` file

- `haven` package does not load with `tidyverse` but is part of it!
- `read_sas()` basically just needs the path to the SAS data set
- Read in `smoke2003.sas7bdat`

read_sas()

```
#install if necessary  
library(haven)
```

read_sas()

```
#install if necessary
library(haven)
smokeData <- read_sas("https://www4.stat.ncsu.edu/~online/datasets/smoke2003.sas7bdat")
smokeData

## # A tibble: 443 x 54
##   SEQN SDDSRVYR RIDSTATR RIDEXMON RIAGENDR RIDAGEYR RIDAGEMN RIDAGEEX RIDRETH1
##   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 21010      3       2       2       2      52     633     634      3
## 2 21012      3       2       2       1      63     765     766      4
## 3 21048      3       2       1       2      42     504     504      1
## 4 21084      3       2       1       2      57     692     693      3
## 5 21093      3       2       1       2      64     778     778      2
## # ... with 438 more rows, and 45 more variables: RIDRETH2 <dbl>,
## #   DMQMILIT <dbl>, DMDBORN <dbl>, DMDCITZN <dbl>, DMDYRSUS <dbl>,
## #   DMDEDUC3 <dbl>, DMDEDUC2 <dbl>, DMDEDUC <dbl>, DMDSCHOL <dbl>,
## #   DMDMARTL <dbl>, DMDHHSIZ <dbl>, INDHHINC <dbl>, INDFMINC <dbl>,
## #   INDFMPIR <dbl>, RIDEXPRG <dbl>, DMDHRGND <dbl>, DMDHRAGE <dbl>,
## #   DMDHRBRN <dbl>, DMDHREDU <dbl>, DMDHRMAR <dbl>, DMDHSEDU <dbl>,
## #   SIALANG <dbl>, SIAPROXY <dbl>, SIAINTRP <dbl>, FIALANG <dbl>, ...
```

- `haven` can also read SPSS data and others

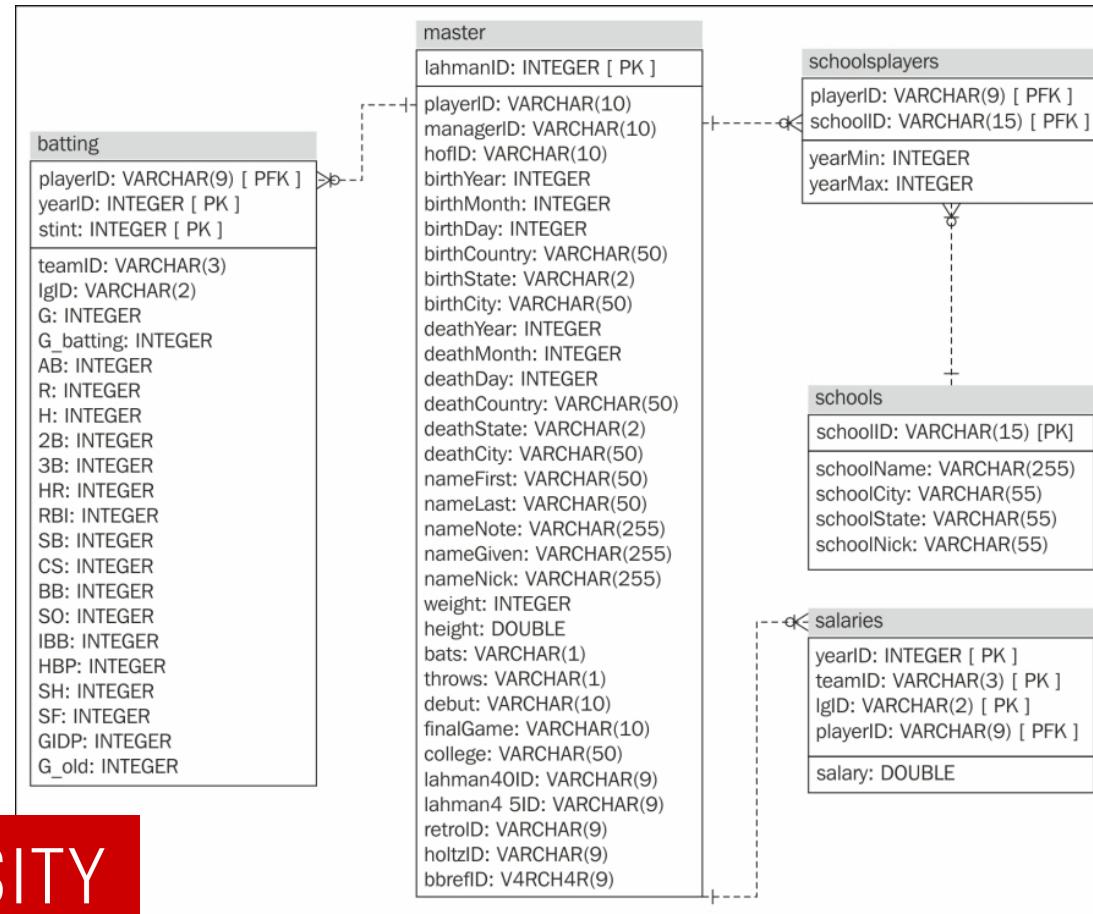
Other Data Sources

JSON - JavaScript Object Notation

- Used widely across the internet and databases
- Can represent usual 2D data or heirarchical data
- `tidyjson` package

Other Data Sources

Databases - Collection of data, often many related 2D tables



Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`

- Need appropriate R package for database backend. Ex:
 - `RSQLite::SQLite()` for RSQLite
 - `RMySQL::MySQL()` for RMySQL

```
con <- DBI::dbConnect(  
  RMySQL::MySQL(),  
  host = "hostname.website",  
  user = "username",  
  password = rstudioapi::askForPassword("DB password"))  
)
```

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`
 - Need appropriate R package for database backend
2. Use `tbl()` to reference a table in the database

```
tbl(con, "name_of_table")
```

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`
 - Need appropriate R package for database backend
2. Use `tbl()` to reference a table in the database
3. Query the database with `SQL` or `dplyr/dbplyr`

Databases - Common flow in R

1. Connect to the database with `DBI::dbConnect()`
 - Need appropriate R package for database backend
2. Use `tbl()` to reference a table in the database
3. Query the database with `SQL` or `dplyr/dbplyr`
4. Disconnect from database with `dbDisconnect()`

More about [R Studio and Databases](#)

Connect to chinook Database

- chinook database is a commonly used intro database
 - sqlite backend

```
library(DBI)
con <- dbConnect(
  RSQLite::SQLite(),
  "data/chinook.db"
)
dbListTables(con)

## [1] "albums"          "artists"         "customers"        "employees"
## [5] "genres"          "invoice_items"   "invoices"         "media_types"
## [9] "playlist_track" "playlists"        "sqlite_sequence" "sqlite_stat1"
## [13] "tracks"
```

Connect to chinook Database

- chinook database is a commonly used intro database

```
dbGetQuery(con, "SELECT * FROM invoices") %>%
  collect() %>%
  as_tibble()

## # A tibble: 412 x 9
##   InvoiceId CustomerId InvoiceDate      BillingAddress BillingCity BillingState
##       <int>      <int> <chr>          <chr>        <chr>        <chr>
## 1         1          1 2009-01-01 00:00~ Theodor-Heuss~ Stuttgart <NA>
## 2         2          2 2009-01-02 00:00~ Ullevålsveien~ Oslo      <NA>
## 3         3          3 2009-01-03 00:00~ Grétrystraat ~ Brussels <NA>
## 4         4          4 2009-01-06 00:00~ 8210 111 ST NW Edmonton AB
## 5         5          5 2009-01-11 00:00~ 69 Salem Stre~ Boston    MA
## # ... with 407 more rows, and 3 more variables: BillingCountry <chr>,
## #   BillingPostalCode <chr>, Total <dbl>
```

Big Recap!

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

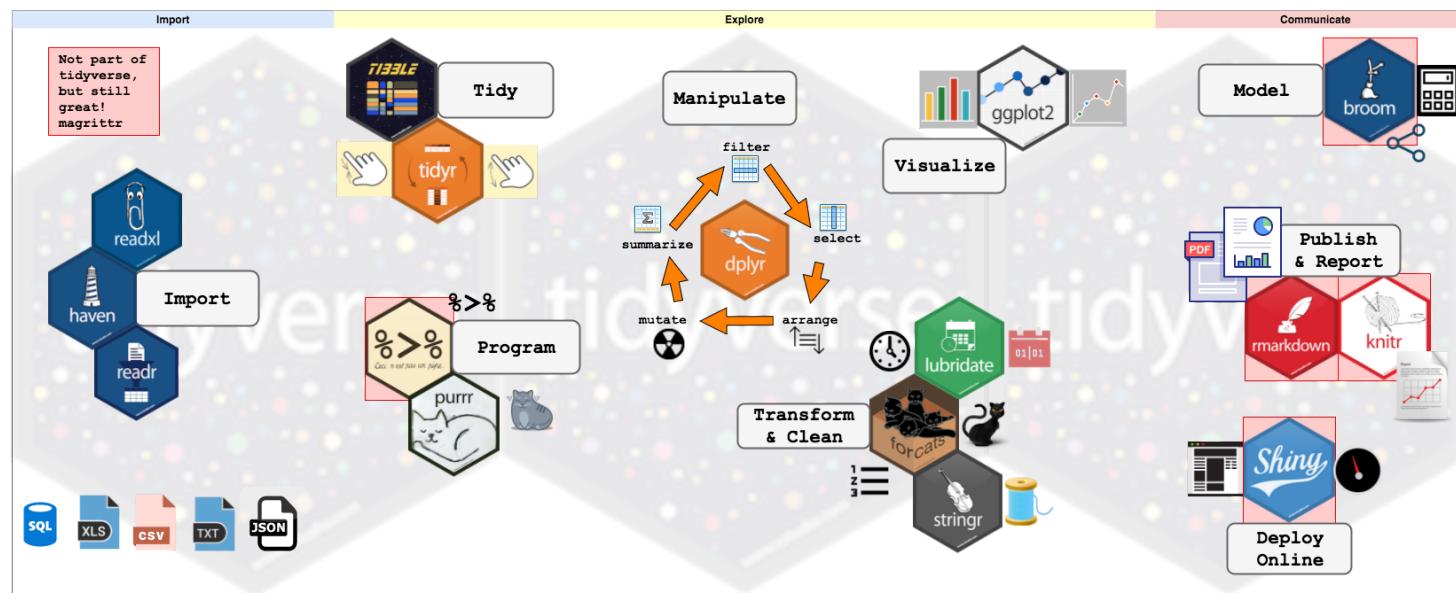
Basic access via

- Atomic vectors - `x[]`
- Data Frames - `x[,]` or `x$name`
- Lists - `x[]`, `x[[]]`, or `x$name`

Big Recap!

`tidyverse` - nice ecosystem of packages with similar behavior and syntax

- `readr`, `haven`, `readxl` all read the data into a tibble
- Good defaults that do the work for you



Let's Practice

We'll add to our `.Rmd` file from the previous activity

- Download the prompts to add to our markdown document [here](#)

Guidance:

- Copy and paste the text from above into the bottom of the document, reknit
- Add to the code chunks, evaluating in the notebook
- Reknit occasionally to check the output

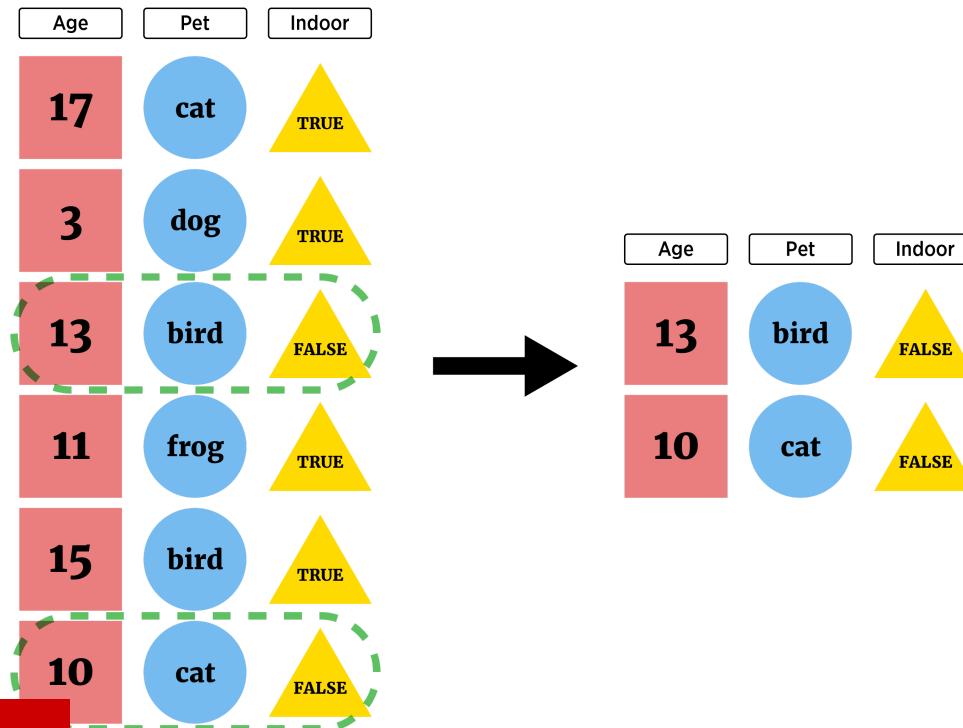
Data Manipulations with dplyr

Justin Post

Data Manipulation Ideas

We may want to subset our full data set or create new variables (columns)

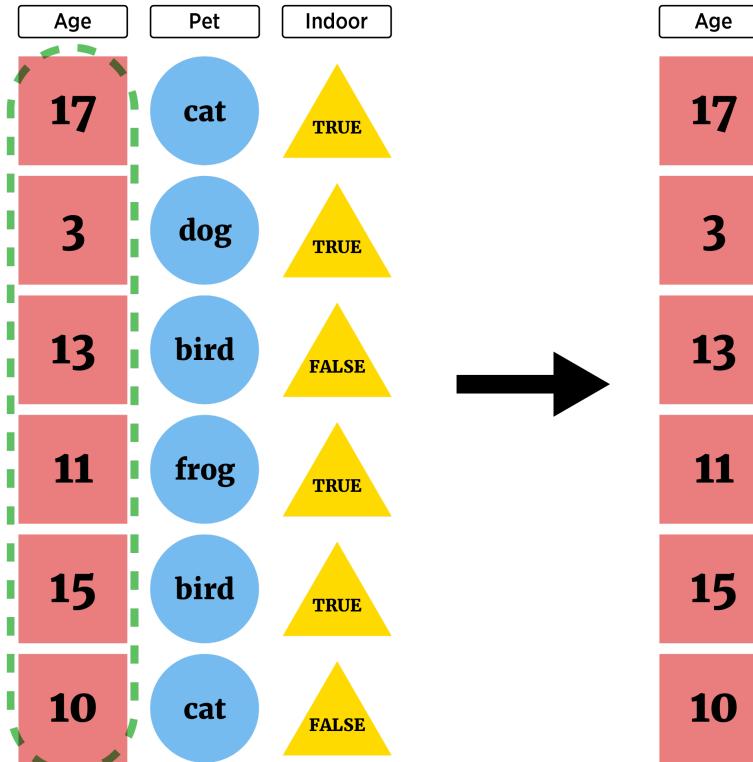
- Grab only certain types of observations (**filter** rows)



Data Manipulation Ideas

We may want to subset our full data set or create new variables (columns)

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



Data Manipulation Ideas

We may want to subset our full data set or create new variables (columns)

- Create new variables (**mutate** columns)

The diagram illustrates a data manipulation process. On the left, there is a 6-row dataset with three columns: Age (red boxes), Pet (blue circles), and Indoor (yellow triangles). The data is as follows:

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

A large black arrow points from the left dataset to the right, indicating the transformation. On the right, the dataset has been expanded to include a new column, AgeType (blue circles). The new data is as follows:

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

tidyverse

`tidyverse` provides a coherent ecosystem for these tasks via the `dplyr` package! **Cheat Sheet**

- Functions take in a `tibble` (special data frames)
- Functions output a `tibble`
- All functions have similar syntax!
`function(tibble, actions, ...)`
- Chaining makes for readable code: `tibble %>% function(actions)`

dplyr

- Commonly used functions:
 - `as_tibble()` - convert data frame to one with better printing

dplyr

- Commonly used functions:
 - `as_tibble()` - convert data frame to one with better printing
 - `filter()` - subset **rows**
 - `arrange()` - reorder **rows**

dplyr

- Commonly used functions:
 - `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**

dplyr

- Commonly used functions:
 - `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**
 - `group_by()` - group **rows** by a variable or variables
 - `if_else()` - conditional execution of code

as_tibble() - A Tidy Data Frame

Want to work on `tibbles`, not just `data frames`

- `as_tibble()` - converts a 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    TR0   NA  1   4   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() - A Tidy Data Frame

- Can 'wrap' a standard R data frame to convert it to a `tibble`

```
myBatting <- as_tibble(Batting)
myBatting

## # A tibble: 108,789 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 abercda01  1871     1 TRO     NA     1     4     0     0     0     0     0
## 2 addybo01   1871     1 RC1     NA    25    118    30    32     6     0     0
## 3 allisar01  1871     1 CL1     NA    29    137    28    40     4     5     0
## 4 allisdo01  1871     1 WS3     NA    27    133    28    44    10     2     2
## 5 ansonca01  1871     1 RC1     NA    25    120    29    39    11     3     0
## # ... with 108,784 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>
```

Filtering Rows Requires Logical Conditions

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

```
"hi" == " hi" #== is comparison
```

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

```
"hi" == "hi"
```

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

```
4 >= 1
```

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

```
4 != 1
```

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

```
sqrt(3)^2 == 3
```

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

```
dplyr::near(sqrt(3)^2, 3)
```

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

Filtering Rows Requires Logical Conditions

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

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

is.numeric(10)
## [1] TRUE
```

```
is.character("10")
## [1] TRUE

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

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

Filtering Rows

- Concept:
 - Feed an *indexing* vector of TRUE/FALSE values
 - 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 TRUE FALSE TRUE TRUE FALSE FALSE FALSE
## [61] TRUE FALSE TRUE FALSE FALSE TRUE TRUE TRUE FALSE TRUE TRUE FALSE
## [73] TRUE 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
## [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
## [157] FALSE FALSE TRUE TRUE FALSE TRUE TRUE FALSE FALSE FALSE FALSE TRUE
## [169] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE TRUE
## [181] FALSE TRUE FALSE TRUE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE
## [193] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE TRUE
## [241] FALSE TRUE TRUE FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE
## [253] FALSE TRUE TRUE FALSE TRUE FALSE FALSE TRUE FALSE TRUE FALSE TRUE
```

filter() - Subset Rows

- Return observations where myBatting\$G is greater than 20

```
filter(myBatting, G > 20)

## # A tibble: 70,926 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 addybo01    1871     1 RC1    NA     25   118    30    32     6     0     0
## 2 allisar01    1871     1 CL1    NA     29   137    28    40     4     5     0
## 3 allisdo01    1871     1 WS3    NA     27   133    28    44    10     2     2
## 4 ansonca01    1871     1 RC1    NA     25   120    29    39    11     3     0
## 5 barnero01    1871     1 BS1    NA     31   157    66    63    10     9     0
## # ... with 70,921 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>
```

Compound Logical Operators

- `&` 'and'
- `|` 'or'

Operator	A,B both true	A true, B false	A,B both false
and	TRUE	FALSE	FALSE
or	TRUE	TRUE	FALSE

Logical statements

- Condition on 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  
## [253] FALSE  
## [265] FALSE  
## [277] FALSE FALSE
```

filter() - Subset Rows

- 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>
## 1 aardsda01  2015     1 ATL    NL     33     1     0     0     0     0     0
## 2 abadfe01  2015     1 OAK    AL     62     0     0     0     0     0     0
## 3 abreuj02  2015     1 CHA    AL    154    613    88    178    34     3    30
## 4 ackledu01 2015     1 SEA    AL     85    186    22    40     8     1     6
## 5 ackledu01 2015     2 NYA    AL     23     52     6    15     3     2     4
## # ... with 944 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

- `%in%` to choose any observations matching a vector

```
filter(myBatting, teamID %in% c("ATL", "PIT", "WSH"))

## # A tibble: 7,236 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 barklsa01  1887     1 PIT    NL     89    340    44    76    10     4     1
## 2 beeched01  1887     1 PIT    NL     41    169    15    41     8     0     2
## 3 bishobi01  1887     1 PIT    NL      3     9     0     0     0     0     0
## 4 brownto01  1887     1 PIT    NL     47    192    30    47     3     4     0
## 5 carrofr01  1887     1 PIT    NL    102    421    71   138     24    15     6
## # ... with 7,231 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

- Other major observation (row) manipulation is to reorder the observations (rows)

```
arrange(myBatting, teamID)

## # A tibble: 108,789 x 22
##   playerID  yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>      <int> <fct> <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 berrych01  1884    1 ALT   UA     7    25     2     6     0     0     0
## 2 brownji01  1884    1 ALT   UA    21    88    12    22     2     2     1
## 3 carropa01  1884    1 ALT   UA    11    49     4    13     1     0     0
## 4 connojo01  1884    1 ALT   UA     3    11     0     1     0     0     0
## 5 crosscl01  1884    1 ALT   UA     2     7     1     4     1     0     0
## # ... with 108,784 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

- Can obtain a secondary arrangement

```
arrange(myBatting, teamID, G)

## # A tibble: 108,789 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 daisege01  1884     1 ALT    UA     1     4     0     0     0     0     0
## 2 crosscl01  1884     1 ALT    UA     2     7     1     4     1     0     0
## 3 manloch01  1884     1 ALT    UA     2     7     1     3     0     0     0
## 4 connojo01  1884     1 ALT    UA     3    11     0     1     0     0     0
## 5 shafff01   1884     1 ALT    UA     6    19     1     3     0     0     0
## # ... with 108,784 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

- Can reorder descending on a variable

```
arrange(myBatting, teamID, desc(G))

## # A tibble: 108,789 x 22
##   playerID  yearID stint teamID lgID     G    AB     R     H    X2B    X3B    HR
##   <chr>      <int> <fct> <fct> <int> <int> <int> <int> <int> <int> <int>
## 1 smithge01  1884    1 ALT   UA     25  108    9   34     8     1     0
## 2 harrifr01  1884    1 ALT   UA     24   95   10   25     2     1     0
## 3 doughch01  1884    1 ALT   UA     23   85     6   22     5     0     0
## 4 murphjo01  1884    1 ALT   UA     23   94   10   14     1     0     0
## 5 brownji01  1884    1 ALT   UA     21   88   12   22     2     2     1
## # ... with 108,784 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>
```

Recap!

- `dplyr` allows for easy row manipulations
 - `as_tibble()`
 - `filter()`
 - `arrange()`
- `dplyr` and `tidyverse` Cheat Sheet

select() - Subset Columns

- To return a single (probably simplified) column:

- `dplyr::pull()`
- `$`
- `[,]`

select() - Subset Columns

- To return a single (probably simplified) column:

- `dplyr::pull()`
- `$`
- `[,]`

```
library(Lahman)
library(dplyr)
myBatting <- as_tibble(Batting)
pull(Batting, X2B)
```

```
## [1] 0 6 4 10 11 2 0 10 1 2 1 0 0 0 9 3 0 0 1 0 2 3 4 0
## [25] 2 2 8 3 0 8 7 0 1 5 7 0 6 3 8 0 5 6 3 1 9 1 3 1 3 1
## [49] 9 3 0 4 6 3 4 3 4 5 1 1 1 1 10 1 3 8 7 7 3 8 3 0
## [73] 4 0 9 9 4 6 0 0 2 2 5 0 10 5 6 0 1 0 7 7 0 7 3 5
## [97] 6 2 10 5 0 4 3 1 7 7 6 2 0 6 10 7 5 5 3 3 2 4 0 4
## [121] 2 10 0 1 28 1 0 1 1 2 11 4 3 0 0 3 5 4 6 3 0 7 0 3
## [145] 0 0 7 1 7 0 4 1 3 9 10 4 0 1 20 3 3 10 13 1 0 0 1 11
## [169] 2 1 0 3 1 0 0 2 9 17 0 4 3 15 0 12 0 10 1 3 5 1 1 1
## [193] 0 0 0 1 0 0 8 1 7 0 0 9 5 0 0 2 1 2 3 6 5 3 0 0
## [217] 3 9 6 10 10 0 0 14 0 0 1 0 5 0 5 0 1 2 15 0 13 0 0 3
## [241] 0 7 10 2 0 5 2 2 12 0 5 1 0 6 1 0 2 4 7 1 2 0 1 11
## [337] 0 0 0 13 0 5 11 4 1 3 7 0 8 7 5 14 20 2 4 1 21 9 6 0
## [338] 0 2 18 10 2 0 9 5 5 5 4 2 0 13 0 1 3
## [339] 9 0 0 6 4 5 1 5 6 4 2 1 0 0 0 0 0
## [340] 1 3 7 0 8 7 5 14 20 2 4 1 21 9 6 0
```

select() - Subset Columns

- `select()` function has same syntax as other `dplyr` functions:

```
function(tibble, actions, ...)
```

select() - Subset Columns

- `select()` function has same syntax as other `dplyr` functions:

```
function(tibble, actions, ...)
```

```
  select(myBatting, X2B)

## # A tibble: 108,789 x 1
##       X2B
##   <int>
## 1     0
## 2     6
## 3     4
## 4    10
## 5    11
## # ... with 108,784 more rows
```

select() - Subset Columns

- `select()` function has same syntax as other `dplyr` functions:

```
function(tibble, actions, ...)
```

```
  select(myBatting, playerID, X2B)

## # A tibble: 108,789 x 2
##   playerID    X2B
##   <chr>     <int>
## 1 abercda01     0
## 2 addybo01      6
## 3 allisar01      4
## 4 allisdo01     10
## 5 ansonca01     11
## # ... with 108,784 more rows
```

Piping or Chaining

- When applying multiple functions, reading the code can be difficult!

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

## # A tibble: 4,920 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
## # ... with 4,915 more rows
```

Piping or Chaining

- Piping or Chaining with `%>%` operator helps make code more readable

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

## # A tibble: 4,920 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
## # ... with 4,915 more rows
```

- Read `%>%` as 'then'

Piping or Chaining

- Generically, `%>%` does the following

`x %>% f(y)` turns into `f(x, y)`

`x %>% f(y) %>% g(z)` turns into `g(f(x, y), z)`

Piping or Chaining

- Generically, `%>%` does the following

`x %>% f(y)` turns into `f(x, y)`

`x %>% f(y) %>% g(z)` turns into `g(f(x, y), z)`

- As `tidyverse` function generally have the same syntax:

```
function(tibble, actions, ...)
```

and they usually return a tibble, they all work great together!

- Can be used with functions outside the tidyverse if this structure works!

select() - Subset Columns

- Great functionality for choosing variables
 - All columns between

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

## # A tibble: 108,789 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
## # ... with 108,784 more rows
```

select() - Subset Columns

- Great functionality for choosing variables
 - All columns containing

```
myBatting %>%  
  select(contains("X"))  
  
## # A tibble: 108,789 x 2  
##   X2B   X3B  
##   <int> <int>  
## 1     0     0  
## 2     6     0  
## 3     4     5  
## 4    10     2  
## 5    11     3  
## # ... with 108,784 more rows
```

select() - Subset Columns

- Great functionality for choosing variables
 - All columns starting with

```
myBatting %>%  
  select(starts_with("X"))
```

```
## # A tibble: 108,789 x 2  
##   X2B   X3B  
##   <int> <int>  
## 1     0     0  
## 2     6     0  
## 3     4     5  
## 4    10     2  
## 5    11     3  
## # ... with 108,784 more rows
```

select() - Subset Columns

- Great functionality for choosing variables
 - Combinations of operators

```
myBatting %>%  
  select(starts_with("X"), ends_with("ID"), G)  
  
## # A tibble: 108,789 x 7  
##   X2B    X3B playerID  yearID teamID lgID      G  
##   <int> <int> <chr>     <int> <fct>  <fct> <int>  
## 1     0     0 abercda01  1871  TR0    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  
## # ... with 108,784 more rows
```

select() - Subset Columns

- Can reorder variables with `everything()`

```
myBatting %>%  
  select(playerID, HR, everything())  
  
## # A tibble: 108,789 x 22  
##   playerID     HR yearID stint teamID lgID      G    AB     R     H    X2B    X3B  
##   <chr>     <int> <int> <fct> <fct> <int> <int> <int> <int> <int> <int>  
## 1 abercda01     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 allisar01     0    1871     1 CL1    NA    29    137    28    40     4     5  
## 4 allisdo01     2    1871     1 WS3    NA    27    133    28    44    10     2  
## 5 ansonca01     0    1871     1 RC1    NA    25    120    29    39    11     3  
## # ... with 108,784 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 Columns

- Easy to rename multiple columns (variables) at once

```
myBatting %>%  
  select(starts_with("X"), ends_with("ID"), G) %>%  
  rename("Doubles" = X2B, "Triples" = X3B)  
  
## # A tibble: 108,789 x 7  
##   Doubles Triples playerID yearID teamID lgID      G  
##   <int>    <int> <chr>     <int> <fct> <fct> <int>  
## 1       0       0 abercda01  1871  TR0    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  
## # ... with 108,784 more rows
```

Recap!

- `dplyr` allows for easy column manipulations
 - `select()`
 - `rename()`
- Pipe or Chain with `%>%`

Creating New Variables

- Consider a data set on movie ratings

```
library(fivethirtyeight)
fandango

## # A tibble: 146 x 23
##   film      year rottentomatoes rottentomatoes_~ metacritic metacritic_user    imdb
##   <chr>     <dbl>        <int>           <int>       <int>       <dbl> <dbl>
## 1 Avenge~  2015         74            86          66        7.1    7.8
## 2 Cinder~  2015         85            80          67        7.5    7.1
## 3 Ant-Man  2015         80            90          64        8.1    7.8
## 4 Do You~  2015         18            84          22        4.7    5.4
## 5 Hot Tu~  2015         14            28          29        3.4    5.1
## # ... with 141 more rows, and 16 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>
```

mutate() - Create New Column(s)

- Add newly created column(s) to current data frame (doesn't overwrite the data frame)

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

```
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 Avenge~  2015         74            86         66        7.1    7.8
## 2 Cinder~  2015         85            80         67        7.5    7.1
## 3 Ant-Man  2015         80            90         64        8.1    7.8
## 4 Do You~ 2015         18            84         22        4.7    5.4
## 5 Hot Tu~  2015         14            28         29        3.4    5.1
## # ... with 141 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)

- Reorder columns so we can 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 Avengers: Age of U~  2015       80           74            86         66  
## 2 Cinderella             2015     82.5          85            80         67  
## 3 Ant-Man                2015       85           80            90         64  
## 4 Do You Believe?       2015       51           18            84         22  
## 5 Hot Tub Time Machi~  2015       21           14            28         29  
## # ... with 141 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>,  
## #     metacritic_user_vote_count <int>, imdb_user_vote_count <int>, ...
```

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 Avengers: ~ 2015      80     3.42         74          86          66  
## 2 Cinderella 2015     82.5    3.55         85          80          67  
## 3 Ant-Man    2015      85     3.62         80          90          64  
## 4 Do You Bel~ 2015      51     1.72         18          84          22  
## 5 Hot Tub Ti~ 2015      21     1.58         14          28          29  
## # ... with 141 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>,  
## #   metacritic_user_vote_count <int>, imdb_user_vote_count <int>, ...
```

mutate() - Create New Column(s)

mutate() can 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  
## # ... with 141 more rows
```

mutate() & group_by() - Create New Column(s)

mutate() can 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  
## # ... with 141 more rows
```

mutate() & group_by() - Create New Column(s)

- `across(.cols, .funs)` for multiple columns/summaries at once

```
fandango %>%  
  select(year, rottentomatoes, metacritic) %>%  
  group_by(year) %>%  
  mutate(across(c(rottentomatoes, metacritic), list(avg = mean, SD = sd)))
```

```
## # A tibble: 146 x 7  
## # Groups:   year [2]  
##   year  rottentomatoes  metacritic  rottentomatoes_avg  rottentomatoes_SD  
##   <dbl>      <int>      <int>          <dbl>            <dbl>  
## 1 2015        74         66          58.4            30.3  
## 2 2015        85         67          58.4            30.3  
## 3 2015        80         64          58.4            30.3  
## 4 2015        18         22          58.4            30.3  
## 5 2015        14         29          58.4            30.3  
## # ... with 141 more rows, and 2 more variables: metacritic_avg <dbl>,  
## #   metacritic_SD <dbl>
```

mutate() & group_by() - Create New Column(s)

- `across(.cols, .funs)` for multiple columns/summaries at once

```
fandango %>%  
  select(year, ends_with("user")) %>%  
  group_by(year) %>%  
  mutate(across(ends_with("user"), list(trim_mean = mean), trim = 0.2))  
  
## # A tibble: 146 x 5  
## # Groups: year [2]  
##   year rottentomatoes_user metacritic_user rottentomatoes_use~ metacritic_user~  
##   <dbl>        <int>       <dbl>           <dbl>          <dbl>  
## 1 2015         86          7.1            64.9          6.63  
## 2 2015         80          7.5            64.9          6.63  
## 3 2015         90          8.1            64.9          6.63  
## 4 2015         84          4.7            64.9          6.63  
## 5 2015         28          3.4            64.9          6.63  
## # ... with 141 more rows
```

Conditional Execution

- Often want to execute statements conditionally to create a variable

`dplyr::if_else()` - *vectorized* conditional execution. 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

Returns a vector

Conditional Execution

- 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
## # ... with 148 more rows
```

Conditional Execution

Want to code a wind category variable

- high wind days ($15\text{mph} \leq \text{wind}$)
- windy days ($10\text{mph} \leq \text{wind} < 15\text{mph}$)
- lightwind days ($6\text{mph} \leq \text{wind} < 10\text{mph}$)
- calm days ($\text{wind} \leq 6\text{mph}$)

if_else()

```
if_else(myAirquality$Wind >= 15, "HighWind",
       if_else(myAirquality$Wind >= 10, "Windy",
              if_else(myAirquality$Wind >= 6, "LightWind", "Calm")))

## [1] "LightWind" "LightWind" "Windy"      "Windy"      "Windy"      "Windy"
## [7] "LightWind" "Windy"      "HighWind"    "LightWind"   "LightWind"   "LightWind"
## [13] "LightWind" "Windy"      "Windy"      "Windy"      "Windy"      "HighWind"
## [19] "Windy"     "LightWind"  "LightWind"   "HighWind"   "LightWind"   "Windy"
## [25] "HighWind"  "Windy"      "LightWind"   "Windy"      "Windy"      "Calm"
## [31] "LightWind" "LightWind"  "LightWind"   "HighWind"   "LightWind"   "LightWind"
## [37] "Windy"     "LightWind"  "LightWind"   "Windy"      "Windy"      "Windy"
## [43] "LightWind" "LightWind"  "Windy"      "Windy"      "Windy"      "HighWind"
## [49] "LightWind" "Windy"      "Windy"      "LightWind"  "Calm"       "Calm"
## [55] "LightWind" "LightWind"  "LightWind"   "Windy"      "Windy"      "Windy"
## [61] "LightWind" "Calm"       "LightWind"   "LightWind"  "Windy"      "Calm"
## [67] "Windy"     "Calm"       "LightWind"   "Calm"       "LightWind"   "LightWind"
## [73] "Windy"     "Windy"      "Windy"      "Windy"      "LightWind"  "Windy"
## [79] "LightWind" "Calm"       "Windy"      "LightWind"  "LightWind"  "Windy"
## [85] "LightWind" "LightWind"  "LightWind"   "Windy"      "LightWind"  "LightWind"
## [91] "LightWind" "LightWind"  "LightWind"   "Windy"      "LightWind"  "LightWind"
## [97] "LightWind" "Calm"       "Calm"       "Windy"      "LightWind"  "LightWind"
## [103] "Windy"     "Windy"      "Windy"      "LightWind"  "Windy"      "Windy"
## [109] "LightWind" "LightWind"  "Windy"      "Windy"      "HighWind"   "Windy"
## [115] "Windy"     "LightWind"  "Calm"       "LightWind"  "Calm"       "LightWind"
## [121] "Calm"      "LightWind"  "LightWind"   "LightWind"  "LightWind"  "Calm"
## [127] "HighWind"  "Windy"      "Windy"      "Windy"      "Windy"      "Windy"
## [133] "HighWind"  "LightWind"  "Windy"      "Windy"      "Windy"      "Windy"
## [139] "Windy"     "Windy"      "LightWind"  "Windy"      "LightWind"  "Windy"
```

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> <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
## # ... with 148 more rows
```

Recap!

- `mutate()` - add newly created **column(s)** to current data frame
- Can use `group_by()` with `mutate()` to add common summary statistics
- Use `if_else()` to do conditional creation
- `dplyr` and `tidyverse` Cheat Sheet

Let's Practice

We'll add to our `.Rmd` file from the previous activity

- Download the prompts to add to our markdown document [here](#)

Guidance:

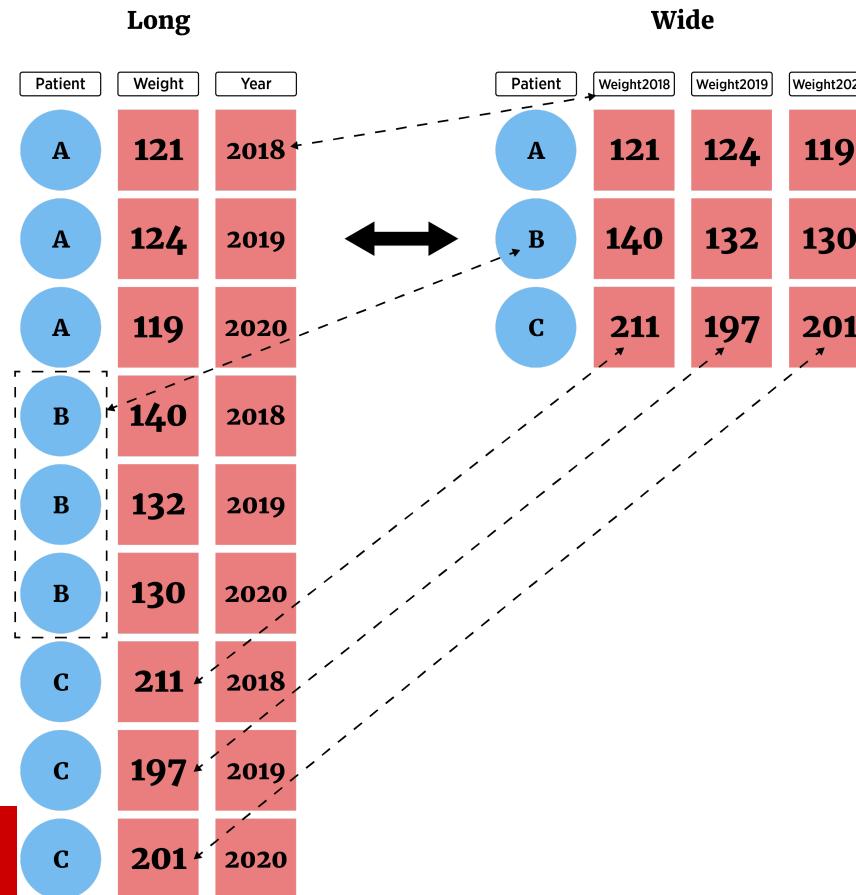
- Copy and paste the text from above into the bottom of the document, reknit
- Add to the code chunks, evaluating in the notebook
- Reknit occasionally to check the output

Reshaping Data with `tidyverse`

Justin Post

Reshaping Data

Long vs Wide format data



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

tidyverse Package

- Data in 'Wide' form

```
tempsData <- read_table(file = "https://www4.stat.ncsu.edu/~online/datasets/cityTemps.txt")
tempsData

## # A tibble: 6 x 8
##   city      sun    mon    tue    wed    thr    fri    sat
##   <chr>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 atlanta    81    87    83    79    88    91    94
## 2 baltimore   73    75    70    78    73    75    79
## 3 charlotte   82    80    75    82    83    88    93
## 4 denver      72    71    67    68    72    71    58
## 5 ellington   51    42    47    52    55    56    59
## 6 frankfort   70    70    72    70    74    74    79
```

Reshaping Data

```
## # A tibble: 6 x 8
##   city      sun    mon    tue    wed    thr    fri    sat
##   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 atlanta    81     87     83     79     88     91     94
## 2 baltimore   73     75     70     78     73     75     79
## 3 charlotte   82     80     75     82     83     88     93
## 4 denver      72     71     67     68     72     71     58
## 5 ellington   51     42     47     52     55     56     59
## 6 frankfort   70     70     72     70     74     74     79
```

- Switch to 'Long' form with `pivot_longer()`
 - `cols` = columns to pivot to longer format (`cols = 2:8`)
 - `names_to` = new name(s) for columns created (`names_to = "day"`)
 - `values_to` = new name(s) for data values (`values_to = "temp"`)

Reshaping Data

- Switch to 'Long' form with `pivot_longer()`
 - `cols` = columns to pivot to longer format (`cols = 2:8`)
 - `names_to` = new name(s) for columns created (`names_to = "day"`)
 - `values_to` = new name(s) for data values (`values_to = "temp"`)

```
tempsData %>% pivot_longer(cols = 2:8, names_to = "day", values_to = "temp")
```

```
## # A tibble: 42 x 3
##   city    day    temp
##   <chr>   <chr> <dbl>
## 1 atlanta sun     81
## 2 atlanta mon     87
## 3 atlanta tue     83
## 4 atlanta wed     79
## 5 atlanta thr     88
## # ... with 37 more rows
```

Reshaping Data

- Switch to 'Long' form with `pivot_longer()`
- Can provide columns in many ways!

```
newTempsData <- tempsData %>%
  pivot_longer(cols = sun:sat, names_to = "day", values_to = "temp")
newTempsData

## # A tibble: 42 x 3
##   city    day    temp
##   <chr>   <chr> <dbl>
## 1 atlanta sun     81
## 2 atlanta mon     87
## 3 atlanta tue     83
## 4 atlanta wed     79
## 5 atlanta thr     88
## # ... with 37 more rows
```

Reshaping Data

- Switch to 'Wide' form with `pivot_wider()`
 - `names_from` = column(s) to get the names used in the output columns (`names_from = "day"`)
 - `values_from` = column(s) to get the cell values from (`values_from = "temp"`)

```
newTempsData %>%
  pivot_wider(names_from = "day", values_from = "temp")

## # A tibble: 6 x 8
##   city      sun    mon    tue    wed    thr    fri    sat
##   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 atlanta    81    87    83    79    88    91    94
## 2 baltimore   73    75    70    78    73    75    79
## 3 charlotte   82    80    75    82    83    88    93
## 4 denver      72    71    67    68    72    71    58
## 5 ellington   51    42    47    52    55    56    59
## 6 frankfort   70    70    72    70    74    74    79
```

Big Recap!

- `dplyr` and `tidyverse` packages
 - Convert to tibble: `as_tibble()`
 - Row manipulations: `arrange()`, `filter()`
 - Column manipulations: `select()`, `rename()`, `mutate()`, `group_by()`, `if_else()`
 - Reshape data: `pivot_wider()`, `pivot_longer()`

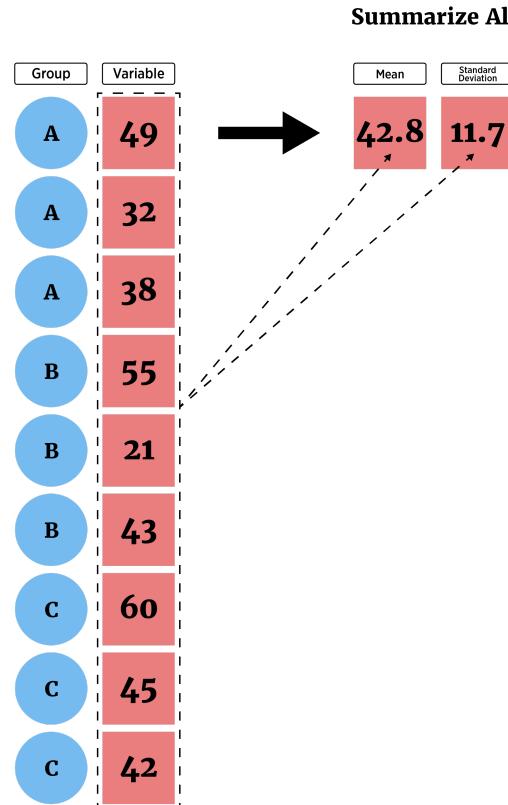
EDA: Numeric Summaries

Justin Post

Making Sense of Data

Goal: Understand types of data and their distributions

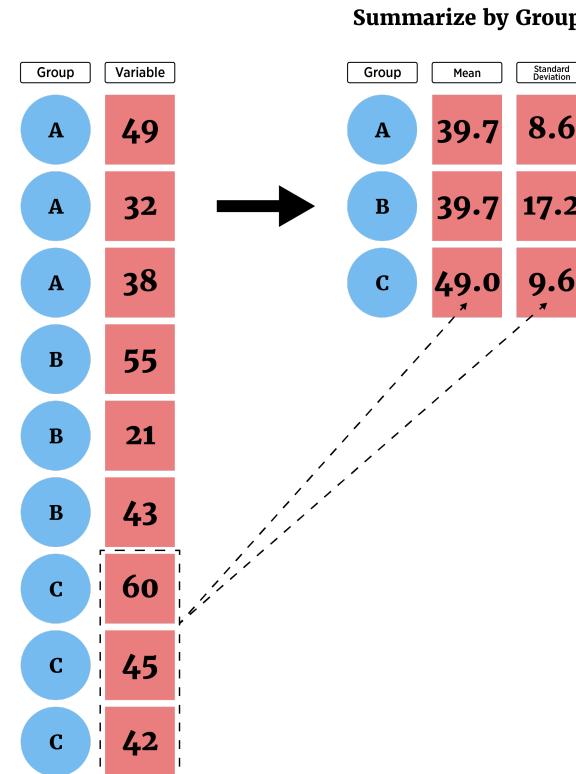
- Numerical summaries



Making Sense of Data

Goal: Understand types of data and their distributions

- Numerical summaries (across subgroups)



Making Sense of Data

Goal: Understand types of data and their distributions

- Numerical summaries (across subgroups)
 - Contingency Tables
 - Mean/Median
 - Standard Deviation/Variance/IQR
 - Quantiles/Percentiles

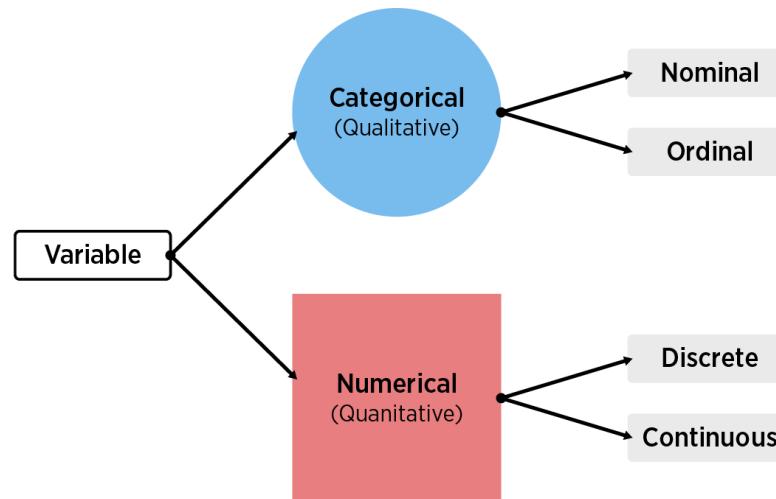
Making Sense of Data

Goal: Understand types of data and their distributions

- Numerical summaries (across subgroups)
 - Contingency Tables
 - Mean/Median
 - Standard Deviation/Variance/IQR
 - Quantiles/Percentiles
- Graphical summaries (across subgroups)
 - Bar plots
 - Histograms
 - Box plots
 - Scatter plots

Types of Data

- How to summarize data depends on the type of data
 - Categorical (Qualitative) variable - entries are a label or attribute
 - Numeric (Quantitative) variable - entries are a numerical value where math can be performed



Categorical Data

Goal: Describe the **distribution** of the variable

- Distribution = pattern and frequency with which you observe a variable
- Categorical variable - entries are a label or attribute

Categorical Data

Goal: Describe the **distribution** of the variable

- Distribution = pattern and frequency with which you observe a variable
- Categorical variable - entries are a label or attribute
 - Describe the relative frequency (or count) for each category
 - Can be done with `group_by()` and `summarize()` from `dplyr` (easier with base R `table()`)

Contingency tables

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

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

## # A tibble: 1,310 x 14
##   pclass survived name      sex    age sibsp parch ticket  fare cabin embarked
##   <dbl>     <dbl> <chr>    <chr> <dbl> <dbl> <dbl> <chr>  <dbl> <chr> <chr>
## 1     1       1 Allen, M~ fema~ 29        0      0 24160  211. B5      S
## 2     1       1 Allison,~ male   0.917     1      2 113781 152. C22     ~ S
## 3     1       0 Allison,~ fema~ 2          1      2 113781 152. C22     ~ S
## 4     1       0 Allison,~ male   30         1      2 113781 152. C22     ~ S
## 5     1       0 Allison,~ fema~ 25         1      2 113781 152. C22     ~ S
## # ... with 1,305 more rows, and 3 more variables: boat <chr>, body <dbl>,
## #   home.dest <chr>
```

Contingency tables

- Create a **one-way contingency table** for the `embarked` variable and for the `survived` variable

```
titanicData %>%  
  group_by(embarked) %>%  
  summarize(counts = n())  
  
## # A tibble: 4 x 2  
##   embarked counts  
##   <chr>     <int>  
## 1 C          270  
## 2 Q          123  
## 3 S          914  
## 4 <NA>        3
```

```
titanicData %>%  
  group_by(survived) %>%  
  summarize(counts = n())  
  
## # A tibble: 3 x 2  
##   survived counts  
##   <dbl>    <int>  
## 1 0          809  
## 2 1          500  
## 3 NA         1
```

Two-way contingency tables

- Create **two-way contingency tables** for pairs of categorical variables

```
titanicData %>%
  group_by(embarked, survived) %>%
  summarize(counts = n())

## # A tibble: 8 x 3
## # Groups:   embarked [4]
##   embarked survived counts
##   <chr>     <dbl>  <int>
## 1 C           0     120
## 2 C           1     150
## 3 Q           0      79
## 4 Q           1      44
## 5 S           0     610
## 6 S           1     304
## 7 <NA>        1      2
## 8 <NA>       NA      1
```

Two-way contingency tables

- Create **two-way contingency tables** for pairs of categorical variables

```
titanicData %>%  
  group_by(embarked, survived) %>%  
  summarize(counts = n())  
  
## # A tibble: 8 x 3  
## # Groups:   embarked [4]  
##   embarked survived counts  
##   <chr>     <dbl>   <int>  
## 1 C           0     120  
## 2 C           1     150  
## 3 Q           0      79  
## 4 Q           1      44  
## 5 S           0     610  
## 6 S           1     304  
## 7 <NA>        1      2  
## 8 <NA>        NA     1
```

```
titanicData %>%  
  group_by(embarked, survived) %>%  
  summarize(counts = n()) %>%  
  pivot_wider(values_from = counts, names_from = embarked)  
  
## # A tibble: 3 x 5  
##   survived     C     Q     S `NA`  
##   <dbl>   <int> <int> <int> <int>  
## 1       0     120     79    610     NA  
## 2       1     150     44    304      2  
## 3     NA     NA     NA     NA      1
```

Two-way contingency tables

- Let's drop the NA values first

```
titanicData %>%  
  drop_na(embarked, survived) %>%  
  group_by(embarked, survived) %>%  
  summarize(counts = n())  
  
## # A tibble: 6 x 3  
## # Groups:   embarked [3]  
##   embarked survived counts  
##   <chr>     <dbl>  <int>  
## 1 C          0     120  
## 2 C          1     150  
## 3 Q          0      79  
## 4 Q          1      44  
## 5 S          0     610  
## 6 S          1     304
```

```
titanicData %>%  
  drop_na(embarked, survived) %>%  
  group_by(embarked, survived) %>%  
  summarize(counts = n()) %>%  
  pivot_wider(values_from = counts, names_from = embarked)  
  
## # A tibble: 2 x 4  
##   survived     C     Q     S  
##   <dbl>    <int> <int> <int>  
## 1       0     120     79    610  
## 2       1     150     44    304
```

Numeric Data

Goal: Describe the **distribution** of the variable

- Distribution = pattern and frequency with which you observe a variable
- Numeric variable - entries are a numerical value where math can be performed

Numeric Data

Goal: Describe the **distribution** of the variable

- Distribution = pattern and frequency with which you observe a variable
- Numeric variable - entries are a numerical value where math can be performed

For a single numeric variable, describe the distribution via

- Shape: Histogram, Density plot, ...
- Measures of center: Mean, Median, ...
- Measures of spread: Variance, Standard Deviation, Quartiles, IQR, ...

Measures of Center

Mean & Median

```
mean(titanicData$fare, na.rm = TRUE)  
## [1] 33.29548  
  
median(titanicData$fare, na.rm = TRUE)  
## [1] 14.4542  
  
titanicData %>%  
  summarize(fareMean = mean(fare, na.rm = TRUE),  
            fareMedian = median(fare, na.rm = TRUE))  
  
## # A tibble: 1 x 2  
##   fareMean fareMedian  
##     <dbl>      <dbl>  
## 1     33.3      14.5
```

```
mean(titanicData$age, na.rm = TRUE)  
## [1] 29.88113  
  
median(titanicData$age, na.rm = TRUE)  
## [1] 28  
  
titanicData %>%  
  summarize(ageMean = mean(age, na.rm = TRUE),  
            ageMedian = median(age, na.rm = TRUE))  
  
## # A tibble: 1 x 2  
##   ageMean ageMedian  
##     <dbl>      <dbl>  
## 1     29.9      28
```

Measures of Spread

Standard Deviation, Quantiles, & IQR

```
titanicData %>%  
  summarize(fareMean = mean(fare, na.rm = TRUE),  
            fareMedian = median(fare, na.rm = TRUE),  
            fareSD = sd(fare, na.rm = TRUE),  
            fareIQR = IQR(fare, na.rm = TRUE),  
            fareQ1 = quantile(fare, probs = c(0.25), na.rm = TRUE))  
  
## # A tibble: 1 x 5  
##   fareMean fareMedian fareSD fareIQR fareQ1  
##     <dbl>      <dbl>    <dbl>    <dbl>    <dbl>  
## 1     33.3      14.5    51.8    23.4     7.90
```

Measures of Spread

Standard Deviation, Quantiles, & IQR

```
titanicData %>%  
  summarize(fareQuantiles = quantile(fare, probs = c(0.1, 0.25, 0.5, 0.75, 0.9), na.rm = TRUE),  
            q = c(0.1, 0.25, 0.5, 0.75, 0.9))  
  
## # A tibble: 5 x 2  
##   fareQuantiles     q  
##       <dbl> <dbl>  
## 1        7.57  0.1  
## 2        7.90  0.25  
## 3       14.5   0.5  
## 4      31.3   0.75  
## 5       78.1   0.9
```

Measures of Linear Relationship

For two numeric variables we can find Covariance & Correlation

```
titanicData %>%  
  summarize(covar = cov(fare, age, use = "complete.obs"),  
            corr = cor(fare, age, use = "complete.obs"))  
  
## # A tibble: 1 x 2  
##   covar    corr  
##   <dbl> <dbl>  
## 1 143. 0.179
```

Summaries Across Groups

Usually want summaries for different **subgroups of data**

- Ex: Get similar fare summaries for each *survival status*

Idea:

- Use `dplyr::group_by()` to associate groups with the tibble
- Use `dplyr::summarize()` to create basic summaries for each subgroup

Summaries Across Groups

- Ex: Get similar fare summaries for each *survival status*

```
titanicData %>%
  group_by(survived) %>%
  summarise(avg = mean(fare, na.rm = TRUE),
            med = median(fare, na.rm = TRUE),
            var = var(fare, na.rm = TRUE))

## # A tibble: 3 x 4
##   survived     avg     med     var
##       <dbl>    <dbl>    <dbl>    <dbl>
## 1        0    23.4    10.5 1166.
## 2        1    49.4    26.0 4713.
## 3      NA    NaN     NA     NA
```

Summaries Across Groups

- Remove NA class for survived

```
titanicData %>%
  drop_na(survived) %>%
  group_by(survived) %>%
    summarise(avg = mean(fare, na.rm = TRUE),
              med = median(fare, na.rm = TRUE),
              var = var(fare, na.rm = TRUE))

## # A tibble: 2 x 4
##   survived   avg   med   var
##       <dbl> <dbl> <dbl> <dbl>
## 1        0  23.4  10.5 1166.
## 2        1  49.4  26   4713.
```

Summaries Across Groups

- Ex: Get similar fare summaries for each *survival status* and *embarked value*

```
titanicData %>%  
  drop_na(survived, embarked) %>%  
  group_by(survived, embarked) %>%  
    summarise(avg = mean(fare, na.rm = TRUE),  
              med = median(fare, na.rm = TRUE),  
              var = var(fare, na.rm = TRUE))  
  
## # A tibble: 6 x 5  
## # Groups:   survived [2]  
##   survived   embarked     avg     med     var  
##   <dbl> <chr>     <dbl> <dbl> <dbl>  
## 1       0   C         40.3  15.0  3198.  
## 2       0   Q         11.6  7.75  119.  
## 3       0   S         21.5  11.5  829.  
## 4       1   C         80.0  55.4  9534.  
## 5       1   Q         13.8  7.75  306.  
## 6       1   S         39.2  25.9  2271.
```

Summarizing across groups

`dplyr::across()` allows for applying a summarization to multiple columns easily!

```
titanicData %>%  
  drop_na(survived) %>%  
  group_by(survived) %>%  
    summarise(across(.fns = mean, .cols = c(age, fare), na.rm = TRUE))  
  
## # A tibble: 2 x 3  
##   survived     age   fare  
##       <dbl> <dbl> <dbl>  
## 1        0  30.5  23.4  
## 2        1  28.9  49.4
```

Summarizing across groups

`dplyr::across()` allows for applying a summarization to multiple columns easily!

```
titanicData %>%  
  drop_na(survived) %>%  
  group_by(survived) %>%  
  summarise(across(.fns = mean, .cols = where(is.double), na.rm = TRUE))  
  
## # A tibble: 2 x 7  
##   survived  pclass    age  sibsp  parch    fare    body  
##   <dbl>     <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl>  
## 1       0     2.50  30.5  0.522  0.329  23.4  161.  
## 2       1     1.96  28.9  0.462  0.476  49.4  NaN
```

Recap!

- Must understand the type of data you have
- Goal: Describe the distribution
- Numerical summaries: use `summarize()`
 - Contingency Tables: `n()`
 - Mean/Median: `mean()`, `median()`
 - Standard Deviation/Variance/IQR: `sd()`, `var()`, `IQR()`
 - Quantiles/Percentiles: `quantile()`
- Across subgroups with `dplyr::group_by()`,
- Multiple columns/functions with `dplyr::across()`

Let's Practice

We'll add to our `.Rmd` file from the previous activity

- Download the prompts to add to our markdown document [here](#)

Guidance:

- Copy and paste the text from above into the bottom of the document, reknit
- Add to the code chunks, evaluating in the notebook
- Reknit occasionally to check the output

ggplot2

Justin Post

Graphical Summaries in R

Three major systems for plotting:

- Base R (built-in functions)
- `lattice`
- `ggplot2` (sort of part of the tidyverse)

Great `ggplot2` reference book here!

ggplot2 Plotting

ggplot2 basics ([Cheat Sheet](#))

- `ggplot(data = data_frame)` creates a plot instance
- Add "layers" to the plot (`geom` or `stat` layers)
 - Creates a visualization of the data

ggplot2 Plotting

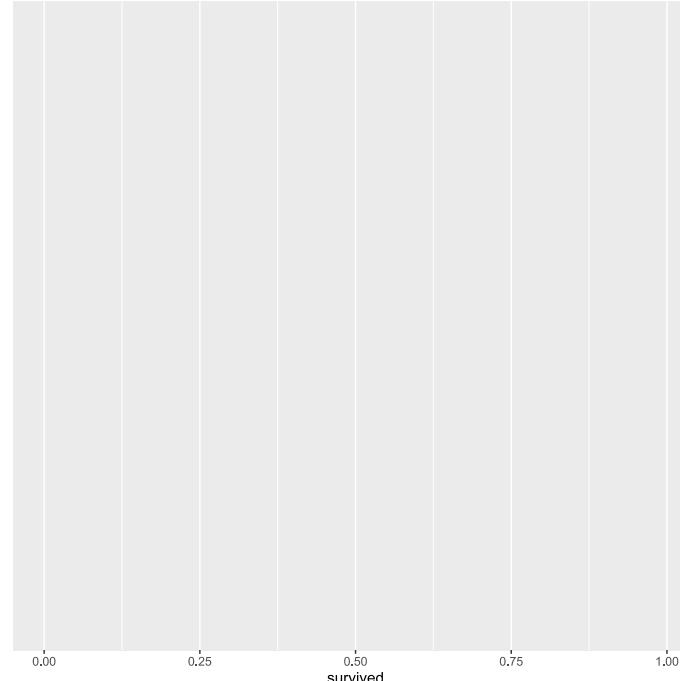
ggplot2 basics ([Cheat Sheet](#))

- `ggplot(data = data_frame)` creates a plot instance
- Add "layers" to the plot (`geom` or `stat` layers)
 - Creates a visualization of the data
- Modify layer "mapping" args (usually with `aes()`)
 - Map variables to attributes of the plot
 - Ex: size, color, x variable, y variable
- Improve by adding title layers, faceting, etc.

ggplot2 Barplots

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

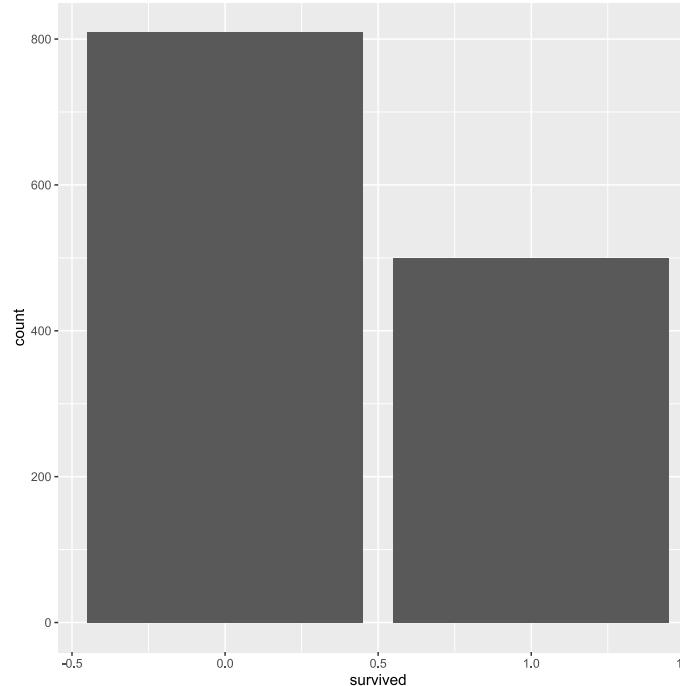
```
ggplot(data = titanicData, aes(x = survived))
```



ggplot2 Barplots

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

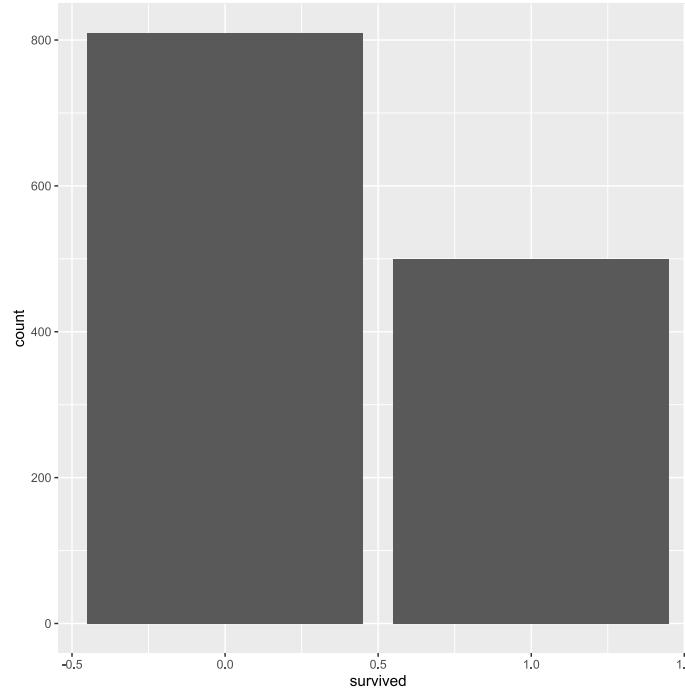
```
ggplot(data = titanicData, aes(x = survived)) + geom_bar()
```



ggplot2 Barplots

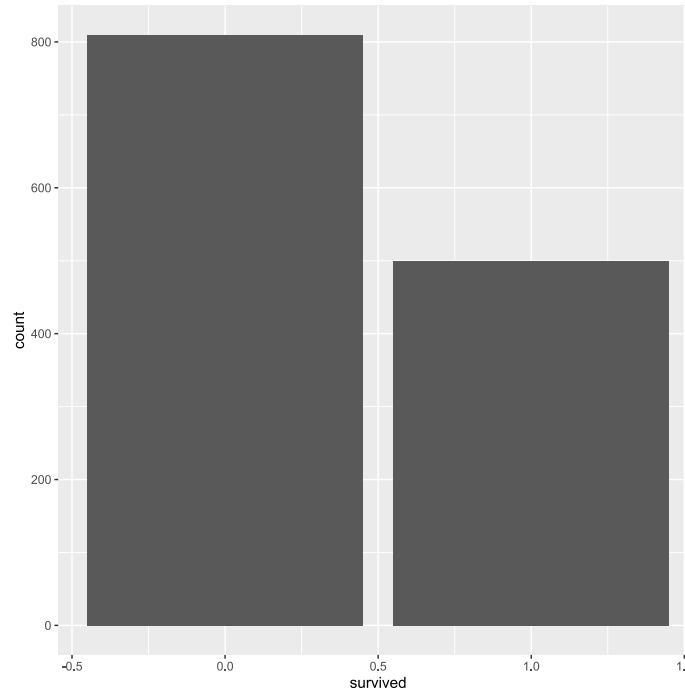
- Generally: Save base object with **global** `aes()` assignments, then add layers

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



Better Labeling Needed

- `survived` data values create a sub-optimal plot!



- Can fix with additional layers but it is easier to change the variable used for plotting!

Factors

- `survived` is read as a numeric variable (truly it is categorical)

```
titanicData
```

```
## # A tibble: 1,310 x 14
##   pclass survived name      sex    age sibsp parch ticket fare cabin embarked
##   <dbl>     <dbl> <chr>    <chr> <dbl> <dbl> <dbl> <chr> <dbl> <chr> <chr>
## 1     1       1 Allen, M~ fema~ 29      0     0 24160  211. B5    S
## 2     1       1 Allison,~ male   0.917    1     2 113781 152. C22   ~ S
## 3     1       0 Allison,~ fema~ 2        1     2 113781 152. C22   ~ S
## 4     1       0 Allison,~ male   30      1     2 113781 152. C22   ~ S
## 5     1       0 Allison,~ fema~ 25      1     2 113781 152. C22   ~ S
## # ... with 1,305 more rows, and 3 more variables: boat <chr>, body <dbl>,
## #   home.dest <chr>
```

- Can create a new version that is a `factor` - works well with `ggplot()`

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, ..., Sunday)
 - Not great for variable like `Name` where new values may come up

Factors

- Create a new factor version of survived

```
titanicData <- titanicData %>% mutate(mySurvived = as.factor(survived))
str(titanicData$mySurvived)

## Factor w/ 2 levels "0","1": 2 2 1 1 1 2 2 1 2 1 ...
levels(titanicData$mySurvived)

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

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

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

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

Prepare our Data

- Let's convert another categorical variable to a factor for better plotting

```
titanicData <- titanicData %>% mutate(myEmbarked = as.factor(embarked))  
levels(titanicData$myEmbarked) <- c("Cherbourg", "Queenstown", "Southampton")
```

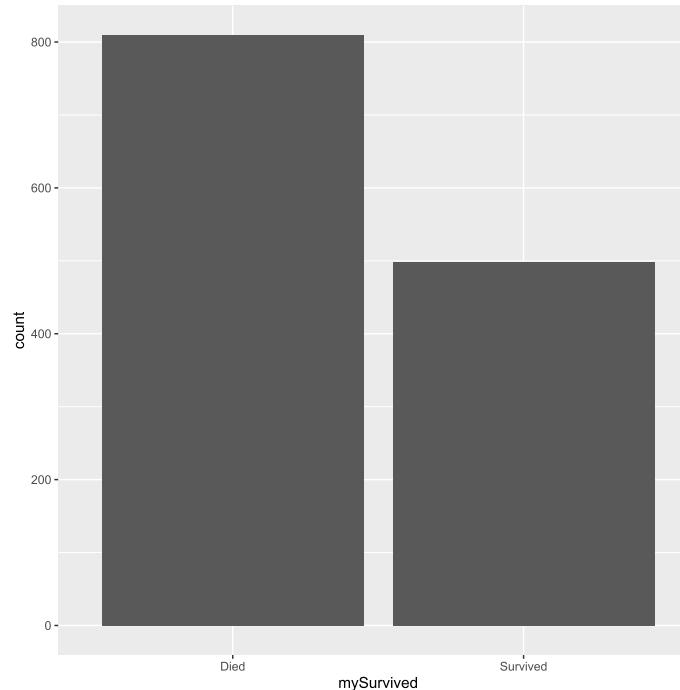
- Let's drop any rows with missing values for any of these variables

```
titanicData <- titanicData %>% drop_na(mySurvived, sex, myEmbarked)
```

Better Labels!

- Same barplot using the factor version of the variable: `mySurvived`

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



aes() Arguments

- `aes()` defines visual properties of objects in the plot
- Map variables in the data frame to plot elements
 - `x = , y = , size = , shape = , color = , alpha = , ...`
- **Cheat Sheet** gives most common properties for a given `geom`

aes() Arguments for Barplots

- `aes()` defines visual properties of objects in the plot
- Map variables in the data frame to plot elements

`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`

aes() Arguments for Barplots

- **Stacked barplot** created by via `fill` aesthetic
- Automatic assignment of colors and creation of legends for `aes` elements (except group)

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

ggplot2 Global vs Local Aesthetics

`data` and `aes` can be set in two ways;

- 'globally' (for all layers) via the `aes()` function in the `ggplot()` call
- 'locally' (for just that layer) via the `geom` or `stat` layer's `aes()`

ggplot2 Global vs Local Aesthetics

`data` and `aes` can be set in two ways;

- 'globally' (for all layers) via the `aes()` function in the `ggplot()` call
- 'locally' (for just that layer) via the `geom` or `stat` layer's `aes()`

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

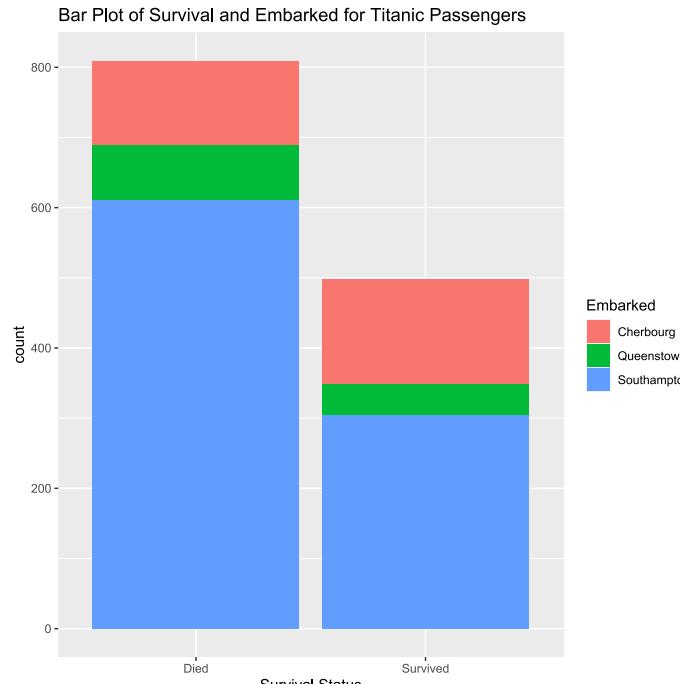
```
#some local, some global  
ggplot(data = titanicData, aes(fill = myEmbarked)) + geom_bar(aes(x = mySurvived))
```

```
#all local  
ggplot() + geom_bar(data = titanicData, aes(x = mySurvived, fill = myEmbarked))
```

ggplot2 Barplots

- Improve our plot by adding a `labs` layer

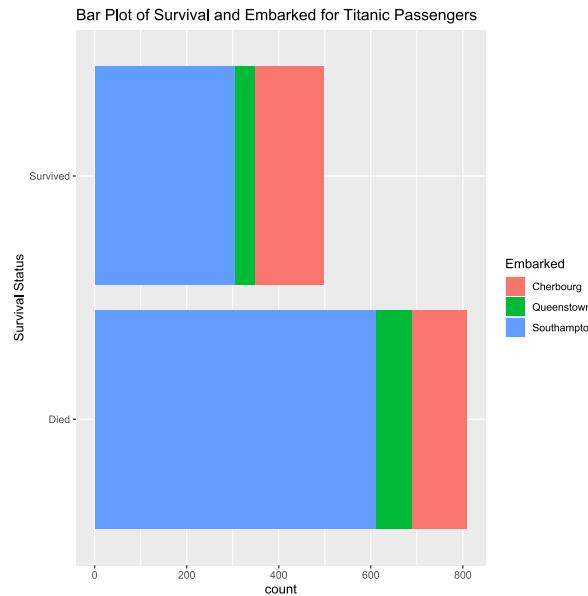
```
g <- ggplot(titanicData)
g + geom_bar(aes(x = mySurvived, fill = myEmbarked)) +
  labs(x = "Survival Status", title = "Bar Plot of Survival and Embarked for Titanic Passengers",
       fill = "Embarked")
```



ggplot2 Horizontal Barplots

- Easy to rotate a plot with `coord_flip()`

```
g <- ggplot(titanicData)
g + geom_bar(aes(x = mySurvived, fill = myEmbarked)) +
  labs(x = "Survival Status", title = "Bar Plot of Survival and Embarked for Titanic Passengers",
       fill = "Embarked") +
  coord_flip()
```



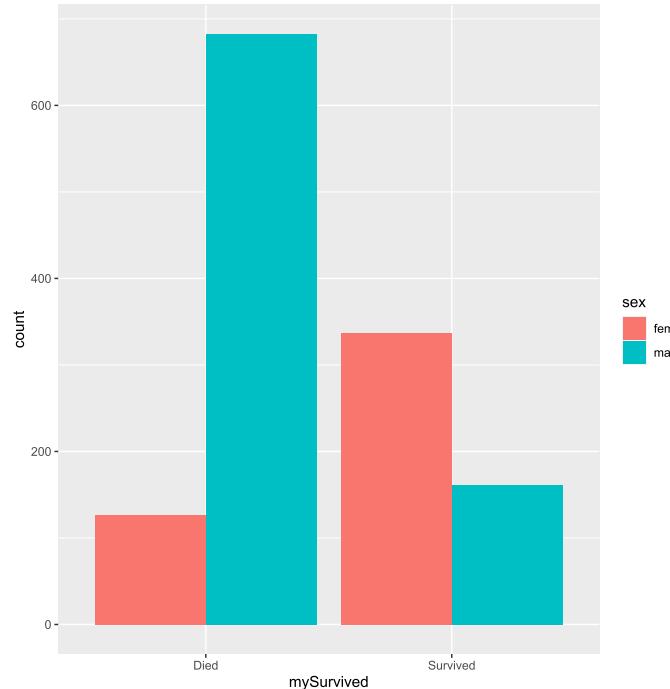
ggplot2 Side-By-Side Barplots

- **Side-by-side barplot** created via the `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

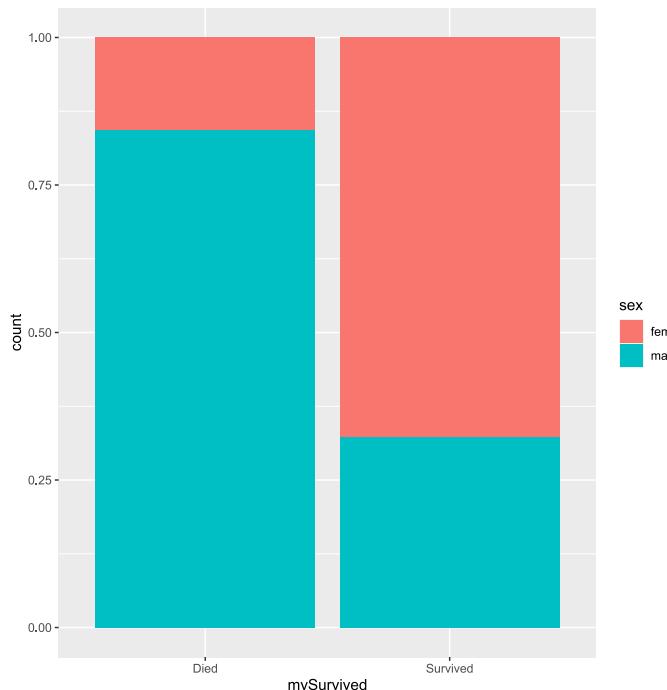
```
g <- ggplot(data = titanicData, aes(x = mySurvived, fill = sex))  
g + geom_bar(position = "dodge")
```



ggplot2 Filled Barplots

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

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



Recap!

General `ggplot2` things:

- Create base plot with `ggplot()`
- Add `geom` layer
- Can set local or global `aes()` (mappings of variables to attributes of the plot)
- Modify titles, labels, etc. by adding more layers
- `position` argument can change style of plot

ggplot2 Smoothed Histogram

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

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

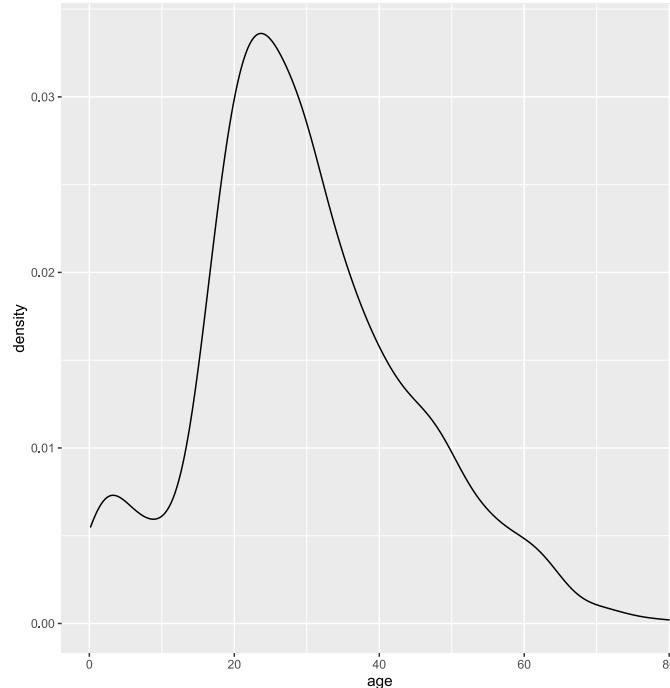
```
x, y, alpha, color, fill, group, linetype, size, weight
```

- Only `x =` is really needed

ggplot2 Smoothed Histogram

- **Kernel Smoother** - Smoothed version of a histogram

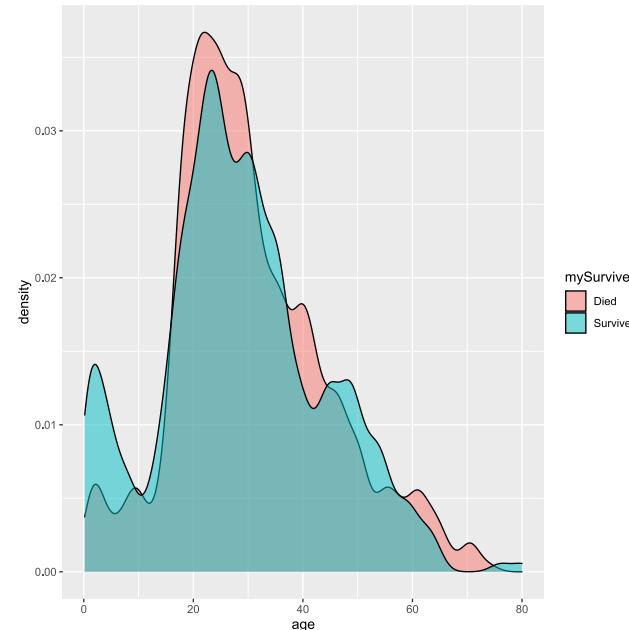
```
g <- ggplot(titanicData, aes(x = age))  
g + geom_density()
```



ggplot2 Smoothed Histogram

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

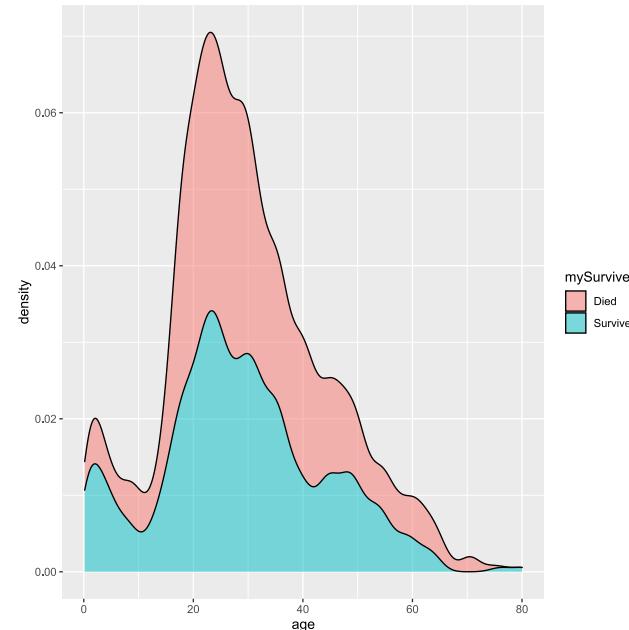
```
g <- ggplot(titanicData, aes(x = age))
g + geom_density(adjust = 0.5, alpha = 0.5, aes(fill = mySurvived))
```



ggplot2 Smoothed Histogram

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

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

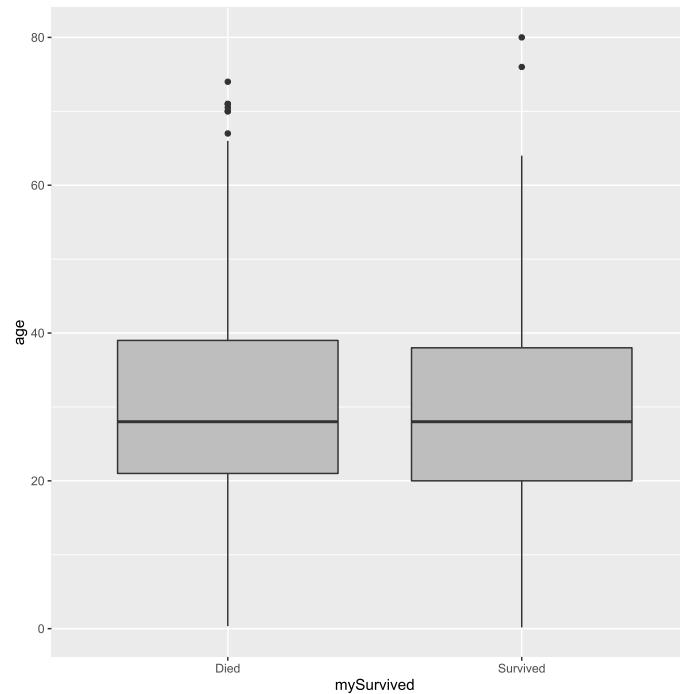
```
f + geom_boxplot()
```

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

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

ggplot2 Boxplots

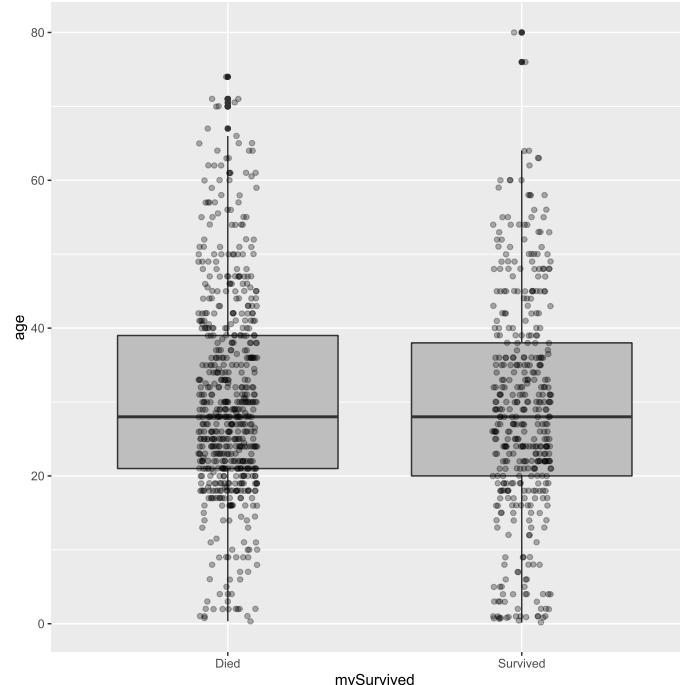
```
g <- ggplot(titanicData, aes(x = mySurvived, y = age))
g + geom_boxplot(fill = "grey")
```



ggplot2 Boxplots with Points

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

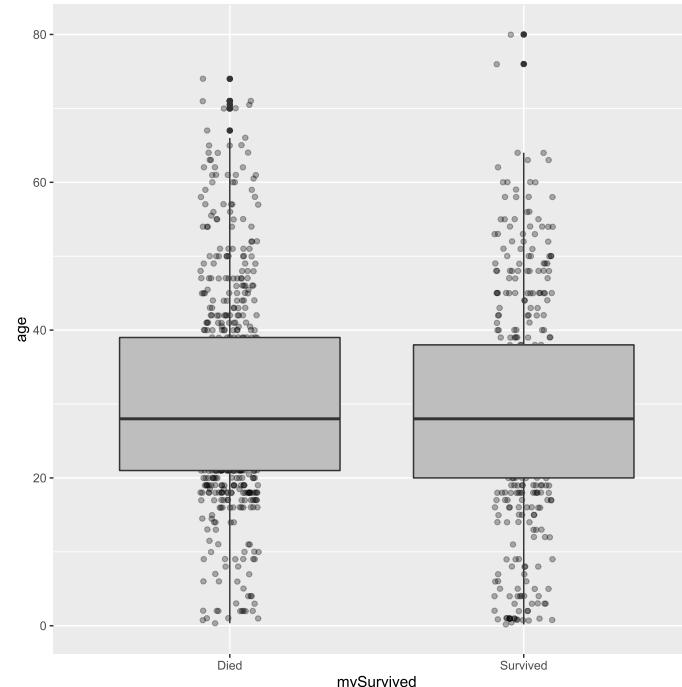
```
g <- ggplot(titanicData, aes(x = mySurvived, y = age))
g + geom_boxplot(fill = "grey") +
  geom_jitter(width = 0.1, alpha = 0.3)
```



ggplot2 Boxplots with Points

- Order of layers important!

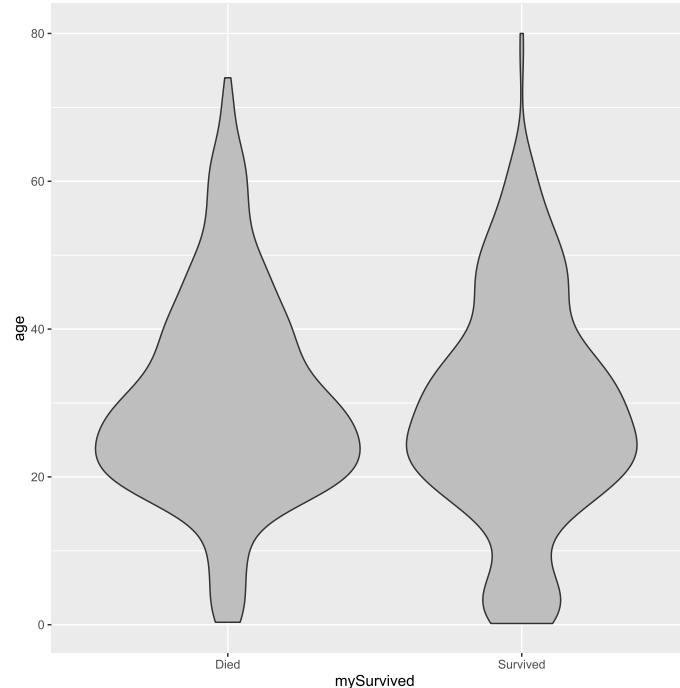
```
g <- ggplot(titanicData, aes(x = mySurvived, y = age))
g + geom_jitter(width = 0.1, alpha = 0.3) +
  geom_boxplot(fill = "grey")
```



ggplot2 Violin Plots

- Violin plot similar to boxplot

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



ggplot2 Scatter Plots

Two numerical variables

- **Scatter Plot** - graphs points corresponding to each observation
- Common `aes` values (from cheat sheet):

`e + geom_point()`

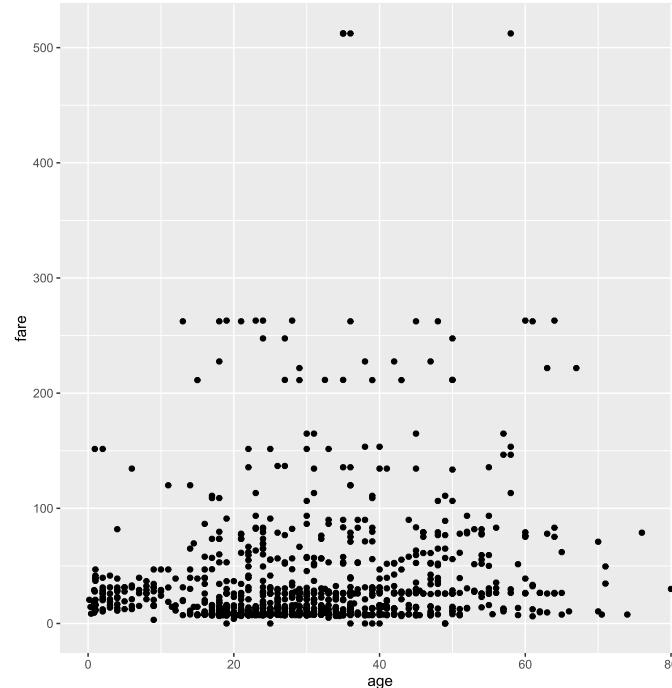
`x, y, alpha, color, fill, shape, size, stroke`

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

ggplot2 Scatter Plots

- **Scatter Plot** - graphs points corresponding to each observation

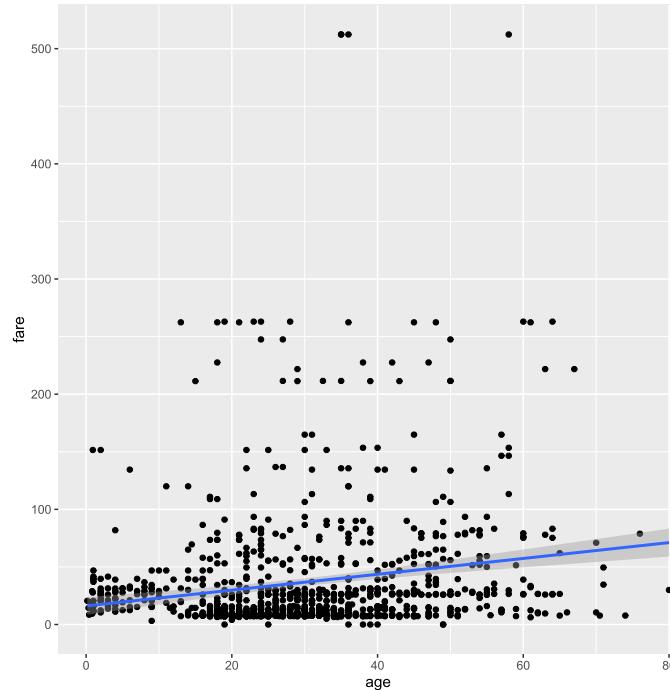
```
g <- ggplot(titanicData, aes(x = age, y = fare))  
g + geom_point()
```



ggplot2 Scatter Plots with Trend Line

- Add trend lines easily with `geom_smooth()`

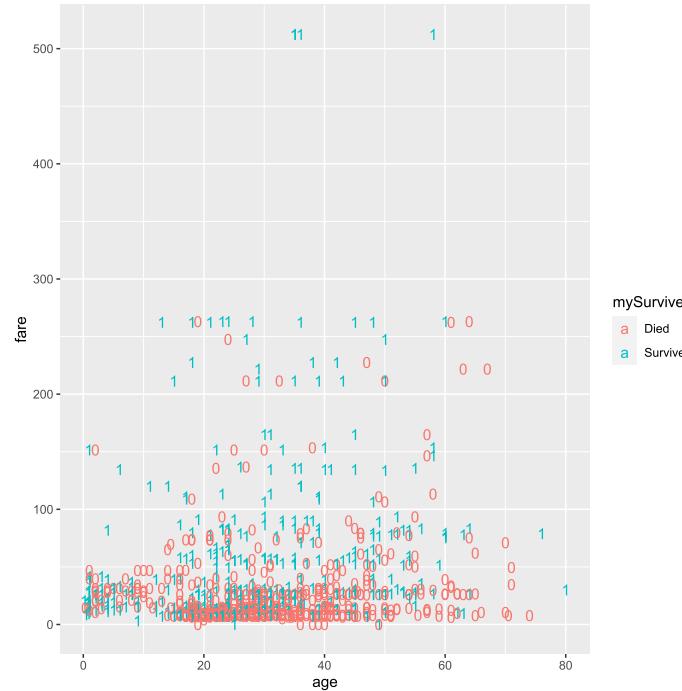
```
g <- ggplot(titanicData, aes(x = age, y = fare))  
g + geom_point() +  
  geom_smooth(method = lm)
```



ggplot2 Scatter Plots with Text Points

- Text for points with `geom_text`

```
g <- ggplot(titanicData, aes(x = age, y = fare))
g + geom_text(aes(label = survived, color = mySurvived))
```

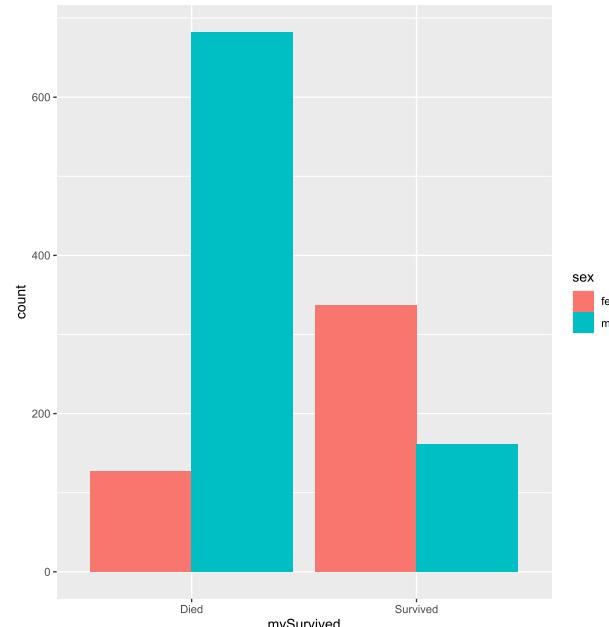


ggplot2 Faceting

Suppose we want to take one of our plots and produce similar plots across another variable!

How to create this plot across each `myEmbarked` category? Use **faceting**!

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



ggplot2 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

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

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

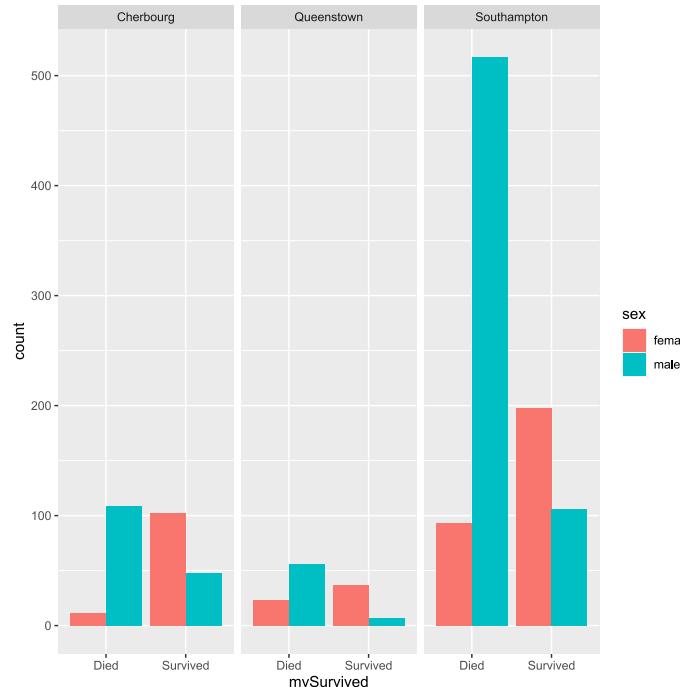
`facet_grid(var1 ~ var2)` - creates a plot for each combination of `var1` and `var2`

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

ggplot2 Faceting

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

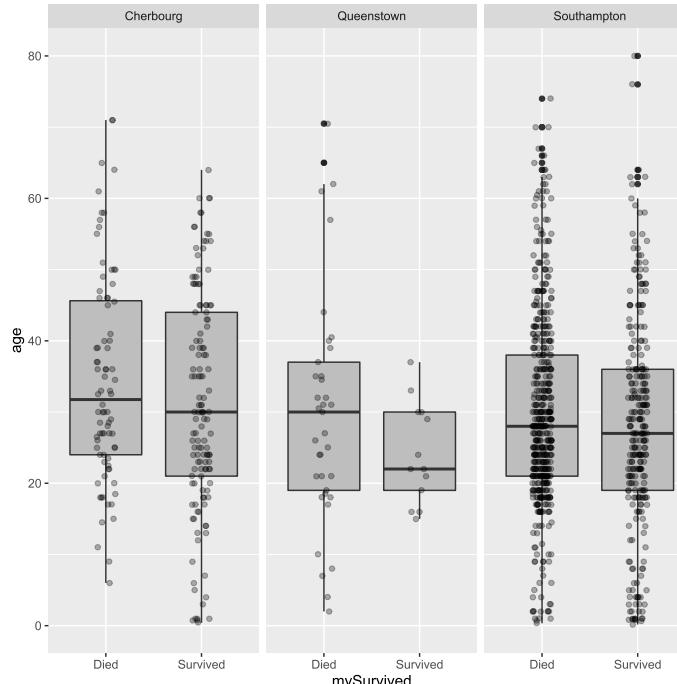
```
g <- ggplot(data = titanicData, aes(x = mySurvived, fill = sex))  
g + geom_bar(position = "dodge") +  
  facet_wrap(~ myEmbarked)
```



ggplot2 Faceting

- Faceting can be used with any `ggplot`

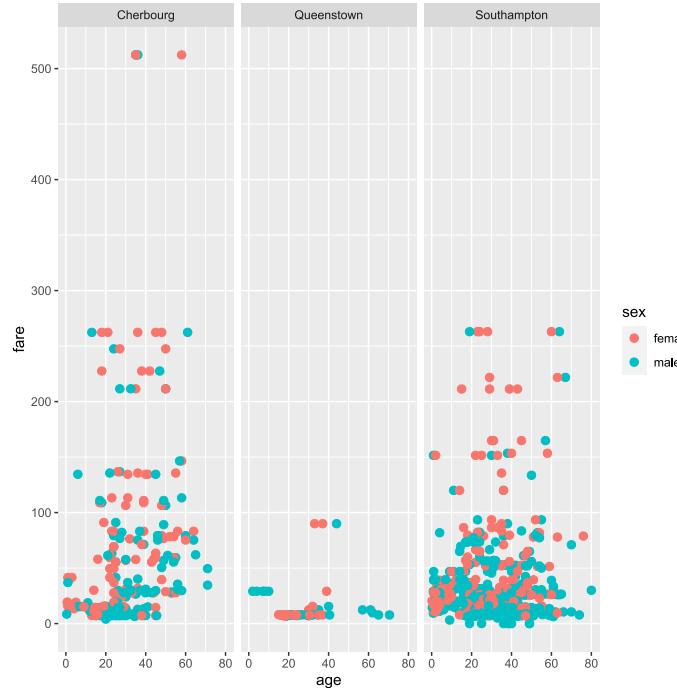
```
g <- ggplot(titanicData, aes(x = mySurvived, y = age))  
g + geom_boxplot(fill = "grey") +  
  geom_jitter(width = 0.1, alpha = 0.3) +  
  facet_wrap(~ myEmbarked)
```



ggplot2 Faceting

- Faceting can be used with any `ggplot`

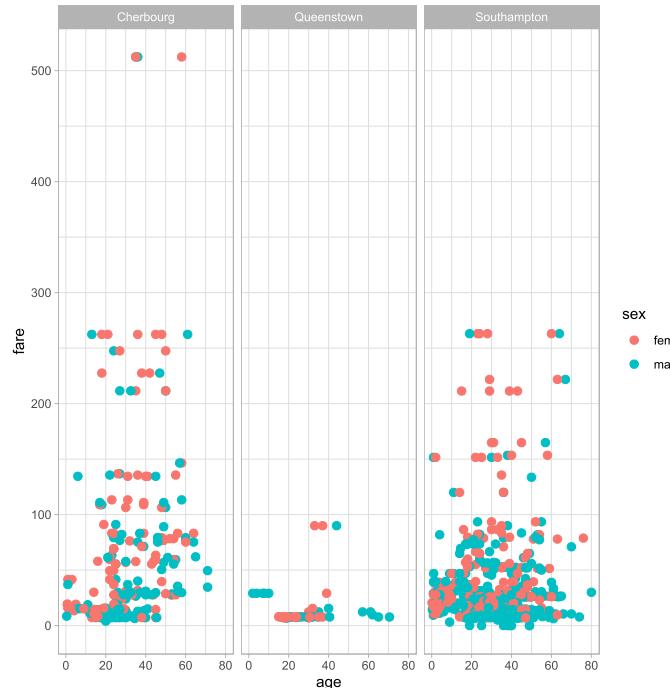
```
g <- ggplot(titanicData, aes(x = age, y = fare))  
g + geom_point(aes(color = sex), size = 2.5) +  
  facet_wrap(~ myEmbarked)
```



ggplot2 Themes

- Can easily change the general look of plots using themes

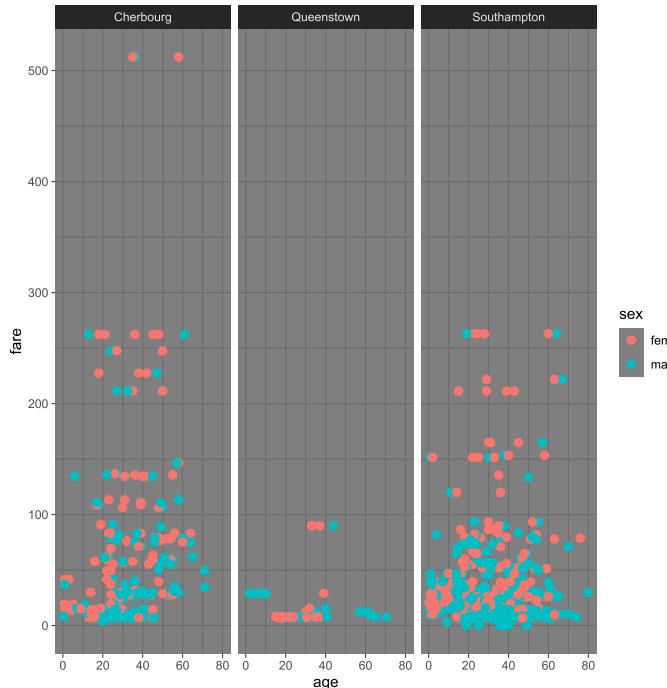
```
g <- ggplot(titanicData, aes(x = age, y = fare))  
g + geom_point(aes(color = sex), size = 2.5) +  
  facet_wrap(~ myEmbarked) +  
  theme_light()
```



ggplot2 Themes

- Can easily change the general look of plots using themes

```
g <- ggplot(titanicData, aes(x = age, y = fare))  
g + geom_point(aes(color = sex), size = 2.5) +  
  facet_wrap(~ myEmbarked) +  
  theme_dark()
```

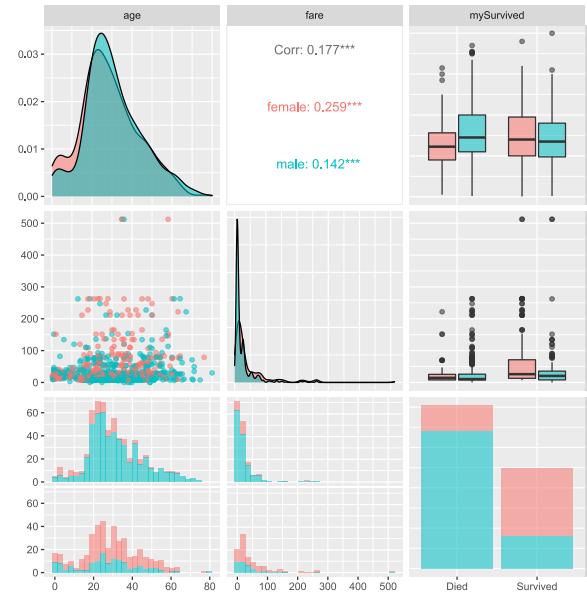


ggplot2 Extensions

Many extension packages that do nice things!

- `GGally` package has the `ggpairs()` function

```
library(GGally) #install if needed  
ggpairs(titanicData, aes(colour = sex, alpha = 0.4), columns = c("age", "fare", "mySurvived"))
```



ggplot2 Extensions

Over 100 registered extensions at <https://exts.ggplot2.tidyverse.org/>!

- `ggridge` package allows for the creation of gifs

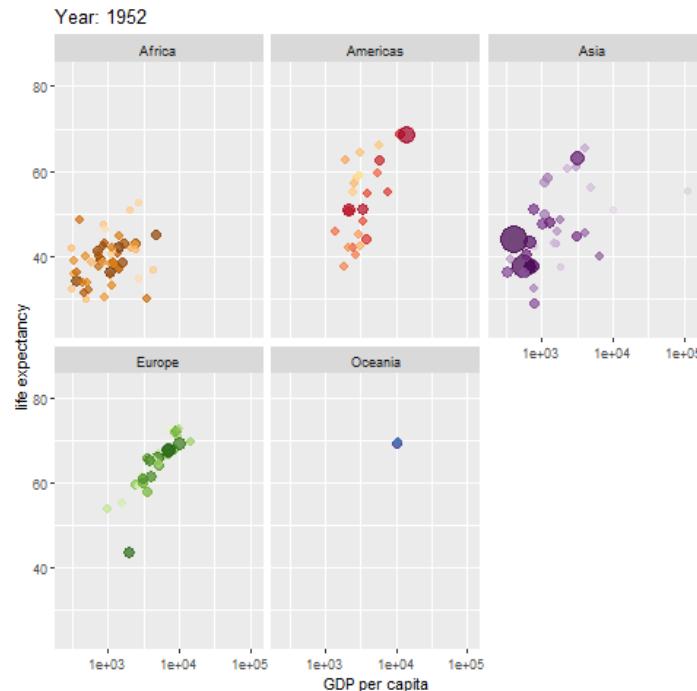
```
#install each if needed
library(gapminder)
library(ggridge)
library(gifski)

gif <- ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, colour = country)) +
  geom_point(alpha = 0.7, show.legend = FALSE) +
  scale_colour_manual(values = country_colors) +
  scale_size(range = c(2, 12)) +
  scale_x_log10() +
  facet_wrap(~continent) +
  # Here comes the ggridge specific bits
  labs(title = 'Year: {frame_time}', x = 'GDP per capita', y = 'life expectancy') +
  transition_time(year) +
  ease_aes('linear')
anim_save(filename = "img/myGif.gif", animation = gif, renderer = gifski_renderer())
```

ggplot2 Extensions

Over 100 registered extensions at [https://exts.ggplot2.tidyverse.org/!](https://exts.ggplot2.tidyverse.org/)

- `ggridge` package allows for the creation of gifs



Recap!

General `ggplot2` things:

- Can set local or global `aes()`
 - Generally, only need `aes()` if setting a mapping value that is dependent on the data
- Modify titles/labels by adding more layers
- Use either `stat` or `geom` layer
- Faceting (multiple plots) via `facet_grid()` or `facet_wrap()`
- `esquisse` is a great package for exploring `ggplot2`!

Let's Practice

We'll add to our `.Rmd` file from the previous activity

- Download the prompts to add to our markdown document [here](#)

Guidance:

- Copy and paste the text from above into the bottom of the document, reknit
- Add to the code chunks, evaluating in the notebook
- Reknit occasionally to check the output