Intro To R

AKA Why I Love R And You Should Too

Laura Albrecht

What is R?

R is a language and environment for statistical computing and graphics

- Developed by Ross Ihaka and Robert Gentleman at University of Auckland in 1993
- Based off of an older programming language called S

Why Use R?

- R is free!
- R is open-source
- Designed specifically for statistical analysis and data reconfiguration
- Extensive package system
- Easy to get started but also powerful
- Amazing support community!

Today's Goal

What we will cover

- Navigating RStudio
- RMarkdown
- Basics of R Programming
- Data Manipulation
- Plotting

RMarkdown

RMarkdown allows you to both save and execute code and generate high quality reports that can be shared.

This entire presentation was made in RMarkdown!

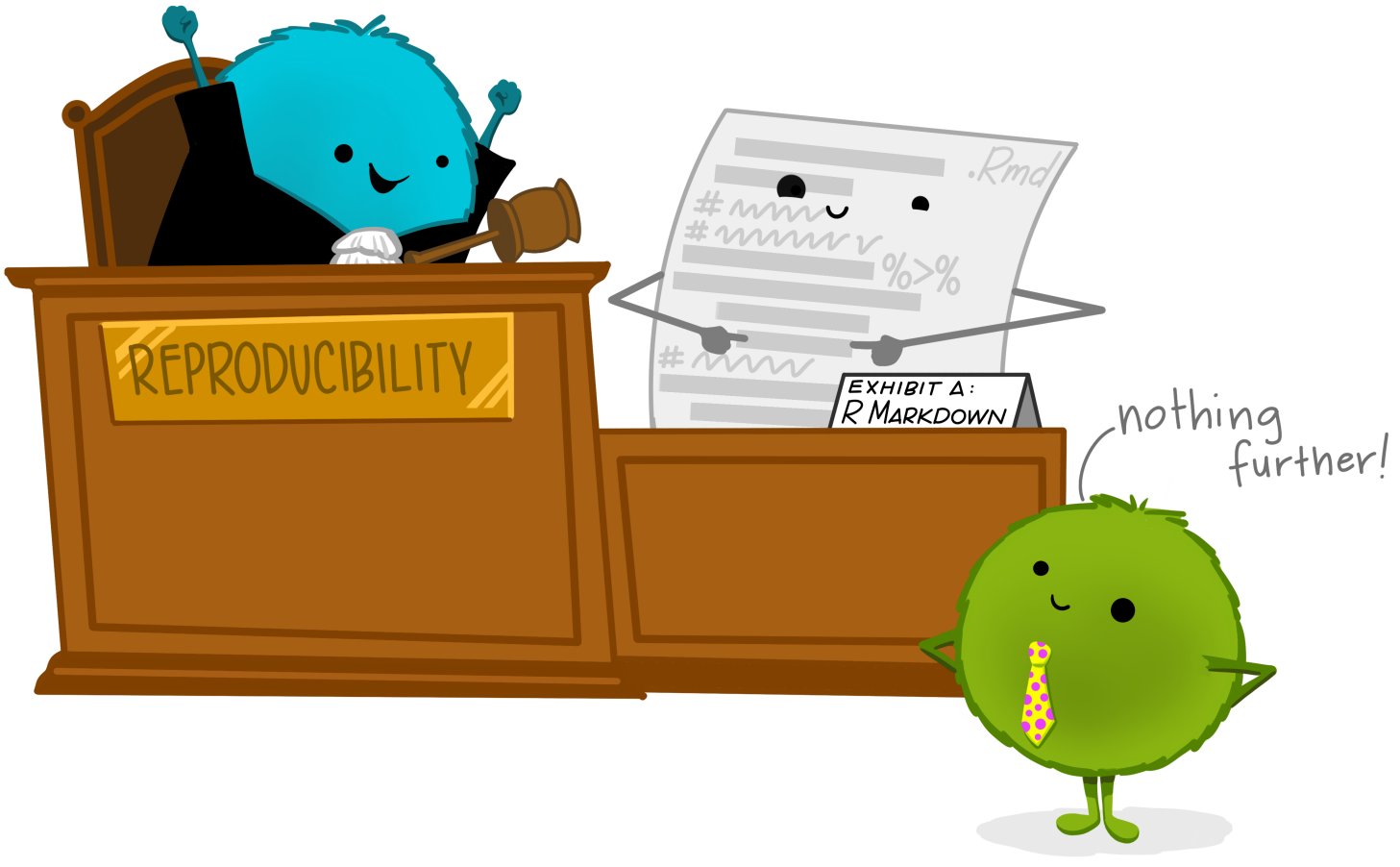
Document Types:

- HTML
- Word
- PDF
- Beamer Presentation
- Power Point Presentation
- Shiny Web App
- Many More!

RMarkdown


Rmarkdown
TEXT. CODE. OUTPUT.
(GET IT TOGETHER, PEOPLE.)
#HEADERS
CODE
italics
TEXT
PANDOC
KNITR
FAIRY DUST
RECIPE
1. ADD TEXT
2. ADD CODE
3. KNIT
4. (magic)
5. CELEBRATE PERCEIVED WIZARDRY
YAML
rmarkdown
.Rmd
knitr
pandoc
.md
Horst '19

RMarkdown


REPRODUCIBILITY
.Rmd
#
#
%>%
#
EXHIBIT A:
R MARKDOWN
nothing further!
@allison_horst

Useful RMarkdown Code Chunk Options

Option	Description
echo	Display code in output document
eval	Run code in chunk
include	Include chunk in doc after running
message	Display code messages in document
warning	Display code warnings in document

Useful shortcuts

Task	Windows	Mac
Insert code chunk	Ctrl+Alt+I	Command+Option+I
Run current line/selection	Ctrl+Enter	Command+Return
Run current chunk	Ctrl+Alt+C	Command+Option+C
Comment/Uncomment lines	Ctrl+Shift+C	Command+Shift+C
Insert pipe (%>%)	Ctrl+Shift+M	Command+Shift+M

R Basics

Assignment

```
# Pound signs indicate a line is a comment
# and will not be run.

x <- 10 #assigns the variable x to be equal to 10

x #prints x (output should be 10)
```

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

```
print(x) #also prints the value of x
```

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

Common Data Types

Type	Description
Numeric	Integers or floating point numbers
Logical	Boolean Values (TRUE or FALSE)
Character	Character strings
Factor	Character strings with preset levels

Vectors

To create a vector, we use the c() function (c = combine/concatenate)

```
y <- c(1, 2, 3, 4, 5) #creates a vector called y of numbers

#Alternatively, we can use ":" to get a sequence of numbers

y <- 1:5
y <- c(1:5)

#Vectors don't have to be numeric either

fruit <- c("apple", "orange", "banana")

fruit
```

```
## [1] "apple"  "orange" "banana"
```

Vectors

Note that if we mix characters and numbers, everything will be converted to the least restrictive data type (character)

```
fruit3 <- c("apple", "orange", "banana", 3)

fruit3
```

```
## [1] "apple"  "orange" "banana" "3"
```

Vectors

We can also generate vectors automatically using a few functions

```
#Vector of all 1's of length 10
ones <- rep(1, 10)
ones
```

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

```
#sequence from 1 to 5 by 0.5
x <- seq(1,5, by = 0.5)
x
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
#10 equally spaced points between 1 amd 5
y <- seq(1,5, length.out = 10)
y
```

```
##  [1] 1.000000 1.444444 1.888889 2.333333 2.777778
##  [6] 3.222222 3.666667 4.111111 4.555556 5.000000
```

Selecting Vector Elements By Position

- **Note indexing in R begins at 1, not 0**

```
z <- 1:10
z[2] #Returns the second element in z
```

```
## [1] 2
```

```
z[c(1,5)] #Returns elements 1 and 5
```

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

```
z[2:4] #Returns elements 2 through 4
```

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

```
z[-3] #Returns all but the 3rd element of z
```

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

Selecting Vector Elements By Value

```
z[z<5] #returns all elements of z less than 10
```

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

```
z[z %in% c(9,10,11)]
```

```
## [1]  9 10
```

Generate Random Numbers

```r
#sample 8 values from z
sample(z, 8) #note default is without replacement
```

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

```r
#sample with replacement
sample(z, 8, replace = TRUE)
```

```
## [1]  4 10  2  1  7  6  8  8
```

```r
#generate 10 values with a Normal(0, 1) distribution
rnorm(10, 0, 1)
```

```
##  [1]  1.2629649 -0.5268869 -0.5256283 -1.2756169
##  [5] -1.3282613 -1.2926088 -0.2792940  1.0784745
##  [9]  1.0556807 -0.2519930
```

Different Distributions

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Vector Calculations

R is vectorized

```
x <- 1:10

x*2 #multiply each element by 2
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x*x #element wise multiplication
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
x/x #element wise division
```

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

Matrix

```
m <- matrix(1:9, nrow = 3, ncol = 3)
m
```

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

The default is to fill the matrix by columns. Set "byrow = TRUE"" to fill by rows.

```
matrix(1:9, nrow = 3, ncol = 3, byrow= TRUE)
```

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

Matrix Operations

```
#Transpose
t(m)
```

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

```
#Matrix multiplication
m %*% m
```

```
##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

Matrix Operations

```r
#Inverse
m2 <- matrix(rnorm(9), 3, 3)
solve(m2)
```

```
##             [,1]       [,2]       [,3]
## [1,] -0.3837198 0.4792639 -0.2060457
## [2,]  0.7145676 0.2122502 -0.2489354
## [3,] -0.2443474 0.6316648  0.6943656
```

```r
#Determinant
det(m2)
```

```
## [1] -2.329791
```

Selecting Elements of Matrix

Select a row:

```
m[2,] #Returns the second row
```

```
## [1] 2 5 8
```

Select a column:

```
m[,1] #Returns the first column
```

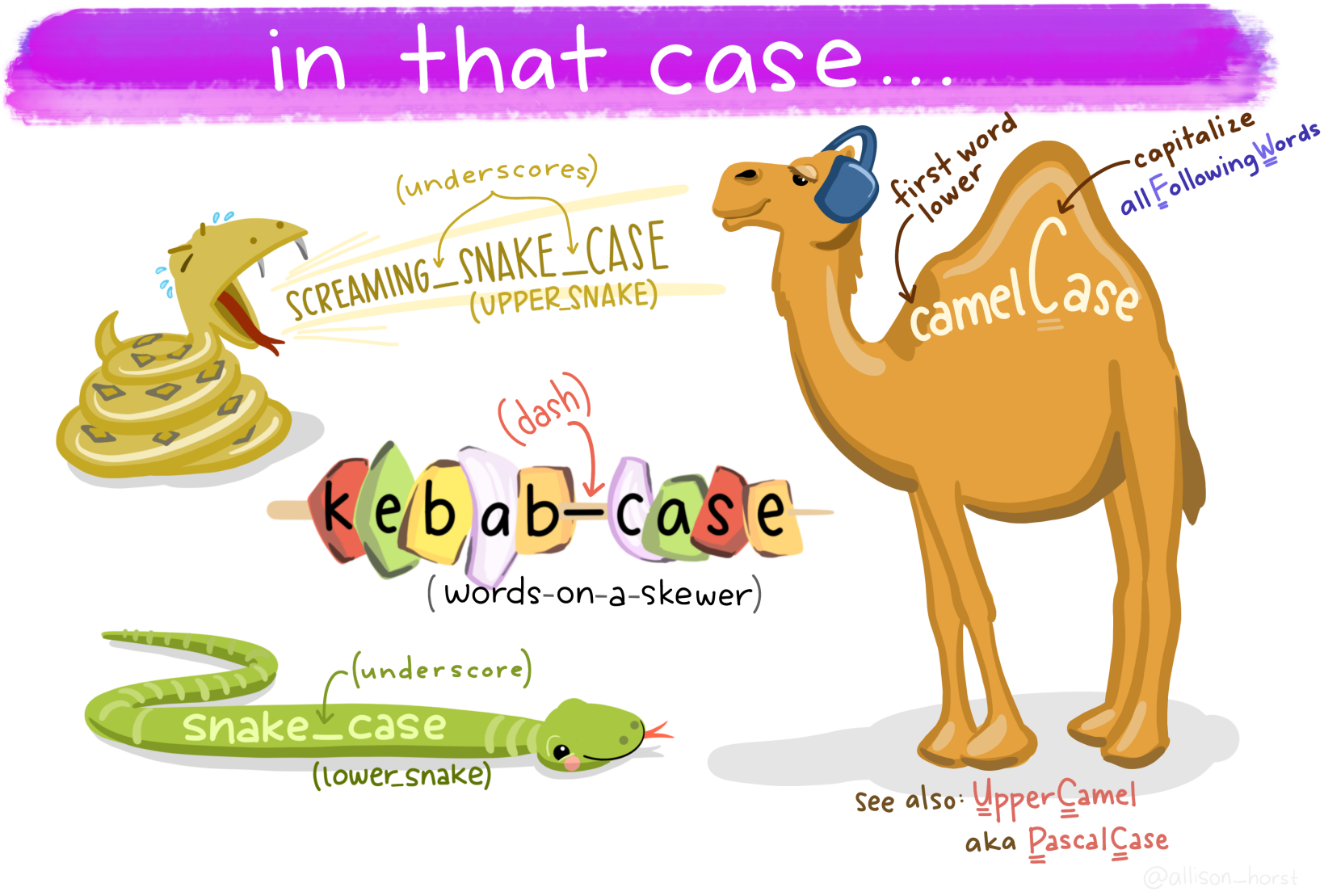
```
## [1] 1 2 3
```

Select an element:

```
m[2,3] #Returns the element in the second row, thrid colum
```

```
## [1] 8
```

Naming Conventions



Lists

A list is a collection of elements which can be of different types

```
my_list <- list(x = 1:5, y = c("a", "b", "c"))
my_list
```

```
## $x
## [1] 1 2 3 4 5
##
## $y
## [1] "a" "b" "c"
```

Selecting Elements of Lists

New list with only second element:

```
my_list[2]
```

```
## $y
## [1] "a" "b" "c"
```

```
my_list["y"]
```

```
## $y
## [1] "a" "b" "c"
```

Selecting Elements of Lists

Second element of my_list

```
my_list[[2]] # Second element of my_list
```

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

```
my_list$y #Element named y
```

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

Data Frames

A data frame is a special case of a list where all elements are the same length

```
fruit <- c("apple", "orange", "banana")
y <- 1:3
df <- data.frame(fruit, y)
df
```

```
##    fruit y
## 1  apple 1
## 2 orange 2
## 3 banana 3
```

Data Frames

We could also name the columns explicitly

```
df1 <- data.frame(fruit = fruit, number = y)
df1
```

```
##     fruit number
## 1   apple      1
## 2  orange      2
## 3  banana      3
```

Data Frames

Convert a matrix to a data frame:

```
df2 <- as.data.frame(m)
df2
```

```
##   V1 V2 V3
## 1  1  4  7
## 2  2  5  8
## 3  3  6  9
```

We can rename the columns in our new data frame using "names"

```
names(df2)<- c("x", "y", "z")
df2
```

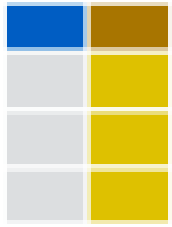
```
##   x y z
## 1 1 4 7
## 2 2 5 8
## 3 3 6 9
```

Selecting Elements of Data Frames

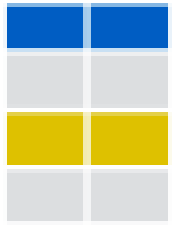
Subsets of data frames can be done in two ways:

Matrix subsetting

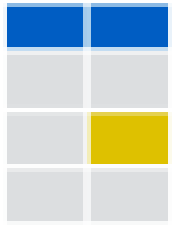
`df[ , 2]`



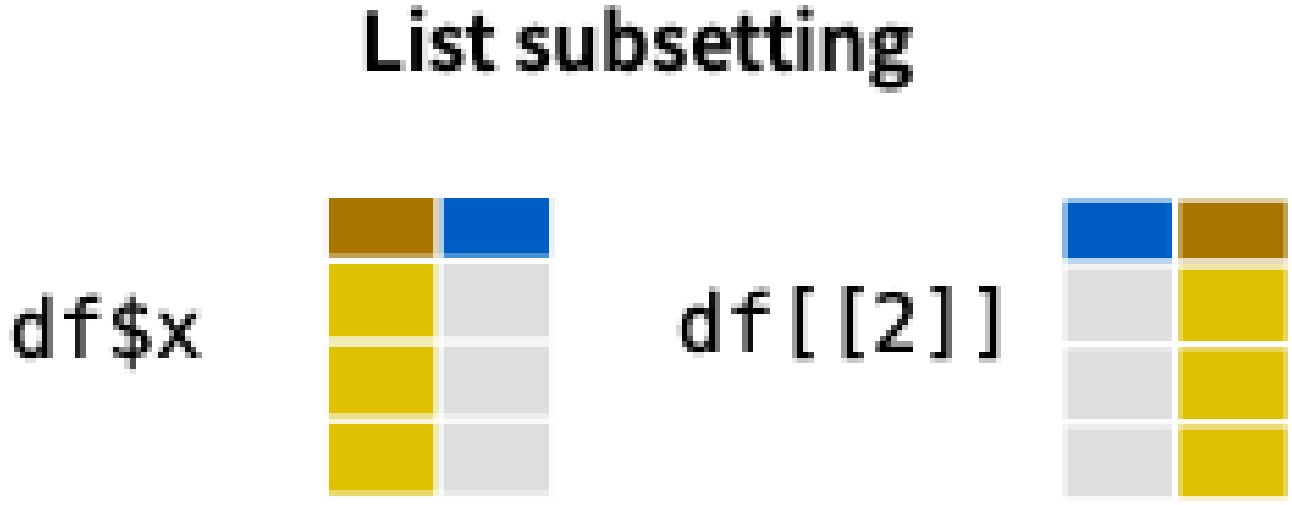
`df[2, ]`



`df[2, 2]`



Selecting Elements of Data Frames



Logical Operations

Operation	Definition
a == b	Equal
a < b	Less than
a <= b	Less than or equal
a > b	Greater than
a >= b	Greater than or equal
!a	Not a
a & b	a and b
a | b	a or b
is.na(a)	Is missing in a
a %in% b	a in the set b

If statements

```
if (condition){
  Do something
} else{
  Do something different
}
```

Example:

```
x <- 5
if(x > 7){
  print("Yes")
} else {
  print("No")
}
```

```
## [1] "No"
```

For Loop

```
for(variable in sequence){
  Do something
}
```

Example:

```
for(i in 1:5){
  j <- i + 1
  print(j)
}
```

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

Writing Functions

```
function_name <- function(var1, var2, ...){
  Do something
  return(new_variable)
}
```

Example:

```
squared <- function(x){
  squared <- x*x
  return(squared)
}
nth_power <- function(x, n){
  return(x^n)
}
```

Reading in Data

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Loading Packages

There are many functions that come pre-loaded into R but there are thousands of R packages you can install that give you access to many other useful functions

Loading Packages

The **first** time you use a package in R you will need to install it.

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

After it is installed, you will need to load in that package in every session when you want to use it:

```
library(dplyr)
```

Help Files

Help files can be useful if you aren't sure how to use a particular function in R or want information about a particular package.

```
#?mean (run without the # sign)
```

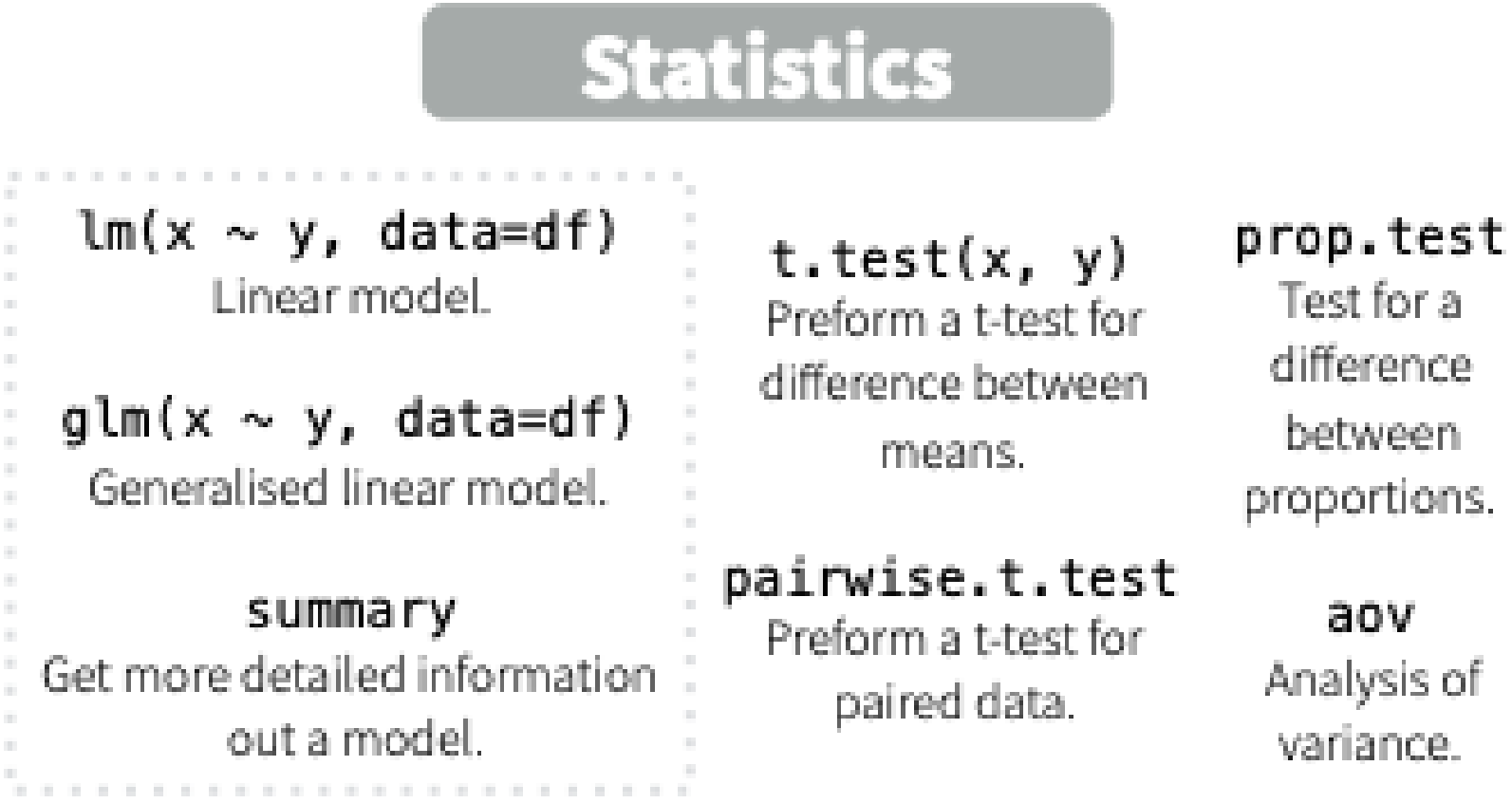
A question mark followed by the name of the function you want to look up will bring up the help file in the "Help" window

Basic Mathematical Functions

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Basic Statistical Functions



Viewing Data

The starwars data set is included in the tidyverse package. Let's start by just viewing the data:

```
#View(starwars) # opens a tab to view the whole dataset
head(starwars) # default prints the first 6 rows
```

```
## # A tibble: 6 x 14
##   name  height  mass hair_color skin_color
##   <chr>  <int> <dbl> <chr>      <chr>
## 1 Luke~    172    77 blond      fair
## 2 C-3PO    167    75 <NA>       gold
## 3 R2-D2     96    32 <NA>       white, bl~
## 4 Dart~    202   136 none       white
## 5 Leia~    150    49 brown      light
## 6 Owen~    178   120 brown, gr~ light
## # ... with 9 more variables: eye_color <chr>,
## #   birth_year <dbl>, sex <chr>, gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>,
```

Viewing Data

```
tail(starwars)
```

```
## # A tibble: 6 x 14
##   name  height  mass hair_color skin_color
##   <chr>  <int> <dbl> <chr>      <chr>
## 1 Finn      NA    NA black      dark
## 2 Rey       NA    NA brown      light
## 3 Poe ~     NA    NA brown      light
## 4 BB8       NA    NA none       none
## 5 Capt~     NA    NA unknown    unknown
## 6 Padm~    165    45 brown      light
## # ... with 9 more variables: eye_color <chr>,
## #   birth_year <dbl>, sex <chr>, gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Viewing Data

```r
str(starwars[,1:10])  # gives a description of each column
```

```
## tibble [87 x 10] (S3: tbl_df/tbl/data.frame)
##  $ name      : chr [1:87] "Luke Skywalker" "C-3PO" "R2-D
##  $ height    : int [1:87] 172 167 96 202 150 178 165 97
##  $ mass      : num [1:87] 77 75 32 136 49 120 75 32 84 7
##  $ hair_color: chr [1:87] "blond" NA NA "none" ...
##  $ skin_color: chr [1:87] "fair" "gold" "white, blue" "w
##  $ eye_color : chr [1:87] "blue" "yellow" "red" "yellow"
##  $ birth_year: num [1:87] 19 112 33 41.9 19 52 47 NA 24
##  $ sex       : chr [1:87] "male" "none" "none" "male" ..
##  $ gender    : chr [1:87] "masculine" "masculine" "mascu
##  $ homeworld : chr [1:87] "Tatooine" "Tatooine" "Naboo"
```

Exploring Data

```
dim(starwars) # returns dimensions
```

```
## [1] 87 14
```

```
ncol(starwars) # number of columns
```

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

```
nrow(starwars) # number of rows
```

```
## [1] 87
```

Exploring Data

For vectors you can use all the same calls except length() should be used instead of dim()

```
head(z)
```

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

```
str(z)
```

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

```
length(z) # use length instead of dim for vectors
```

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

```
summary(z)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

Exploring Data

Now let's explore the starwars data a little closer

```
summary(starwars[,1:10])
```

```
##      name                height
##  Length:87          Min.   : 66.0
##  Class :character   1st Qu.:167.0
##  Mode  :character   Median :180.0
##                     Mean   :174.4
##                     3rd Qu.:191.0
##                     Max.   :264.0
##                     NA's   :6
##       mass          hair_color
##  Min.   :  15.00   Length:87
##  1st Qu.:  55.60   Class :character
##  Median :  79.00   Mode  :character
##  Mean   :  97.31
##  3rd Qu.:  84.50
##  Max.   :1358.00
```

Exploring Data

Here are two ways of learning more about the "character" data types:

```
summary(as.factor(starwars$eye_color))
```

```
##          black           blue      blue-gray
##             10             19              1
##          brown           dark           gold
##             21              1              1
## green, yellow         hazel         orange
##              1              3              8
##           pink            red      red, blue
##              1              5              1
##        unknown          white         yellow
##              3              1             11
```

Exploring Data

Two ways of learning more about the "character" data types:

```
table(starwars$eye_color)
```

```
##
##          black           blue      blue-gray
##             10             19              1
##          brown           dark           gold
##             21              1              1
## green, yellow          hazel         orange
##              1              3              8
##           pink            red      red, blue
##              1              5              1
##        unknown          white         yellow
##              3              1             11
```

Exploring Data

How can we find out which character has pink eyes?

```
which(starwars$eye_color == "pink")
```

```
## [1] 43
```

The character in row 43 has pink eyes

```
starwars[43, 1]
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Bib Fortuna
```

Exploring Data

There are a lot of ways we could extract this information though:

```
starwars[starwars$eye_color == "pink",]
```

```
## # A tibble: 1 x 14
##   name  height  mass hair_color skin_color
##   <chr>  <int> <dbl> <chr>      <chr>
## 1 Bib ~    180    NA none       pale
## # ... with 9 more variables: eye_color <chr>,
## #   birth_year <dbl>, sex <chr>, gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

This way is a little cleaner:

```
ind <- starwars$eye_color == "pink"
starwars[ind,]
```

Exploring Data

The filter function from the dplyr package also provides a nice clean way of doing this:

```
filter(starwars, eye_color == "pink")
```

```
## # A tibble: 1 x 14
##   name  height  mass hair_color skin_color
##   <chr>  <int> <dbl> <chr>      <chr>
## 1 Bib ~    180    NA none       pale
## # ... with 9 more variables: eye_color <chr>,
## #   birth_year <dbl>, sex <chr>, gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Adding Columns or Rows

Combine vectors, matrices, or data frames

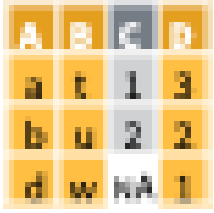
```r
# Combine columns of x and y
cbind(x, y) # x and y need to have same number of rows

# Combine rows of x and z
rbind(x, z) # x and z need to have the same number of cols

df$letters = c("a", "b", "c") # add a column by name
```

Mutating Joins

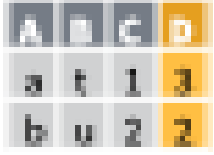
Join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

Mutating Joins

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.



right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.



inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Select Function

Only keep specific columns by name

```
starwars_small <- select(starwars, name, height, mass)
```

Mutate Functions

The mutate function will create a new variable from a data set

```
starwars_small <- mutate(starwars_small,
                         bmi = mass/(height^2))
```

```
## # A tibble: 87 x 4
##    name                height  mass     bmi
##    <chr>                <int> <dbl>   <dbl>
##  1 Luke Skywalker         172    77 0.00260
##  2 C-3PO                  167    75 0.00269
##  3 R2-D2                   96    32 0.00347
##  4 Darth Vader            202   136 0.00333
##  5 Leia Organa            150    49 0.00218
##  6 Owen Lars              178   120 0.00379
##  7 Beru Whitesun lars     165    75 0.00275
##  8 R5-D4                   97    32 0.00340
##  9 Biggs Darklighter      183    84 0.00251
## 10 Obi-Wan Kenobi         182    77 0.00232
## # ... with 77 more rows
```

Separate Function

The separate function breaks a character column into multiple columns at a regular expression or numeric location

```
people <- data.frame(Names = c("Peyton Manning", "Tom Brady
                               "John Elway", "Joe Montana",
                               "Brett Favre"))
people
```

```
##             Names
## 1 Peyton Manning
## 2       Tom Brady
## 3      John Elway
## 4     Joe Montana
## 5     Brett Favre
```

Separate Function

We want to separate the Names column into two columns, "First" and "Last"

```
separate(data = people,
         col = Names, # column to separate
         into = c("First", "Last"), # new col names
         sep = " ") # separate at the blank space
```

```
##    First    Last
## 1 Peyton Manning
## 2    Tom   Brady
## 3   John   Elway
## 4    Joe Montana
## 5  Brett   Favre
```

Reformatting Data

Long versus wide format:

Wide data has a column for each variable

- Wide data is often useful for data entry so observations are commonly imported this way

Long data has a column for possible variable types and column for the values of those variables

- Long form data is sometimes needed for statistical modeling and plotting

Wide Data

```
library(readr)
italy_covid <- read_csv("italy_covid.csv")
head(italy_covid)
```

```
## # A tibble: 6 x 26
##   date       Napoli Piacenza Parma Reggio_Emilia
##   <date>      <dbl>    <dbl> <dbl>          <dbl>
## 1 2020-02-24      0        0     0              0
## 2 2020-02-25      0       18     4              0
## 3 2020-02-26      0       10     4              0
## 4 2020-02-27      3       35     2              0
## 5 2020-02-28      1       26    17              1
## 6 2020-02-29      9       49     8              3
## # ... with 21 more variables: Modena <dbl>,
## #   Bologna <dbl>, Rimini <dbl>, Roma <dbl>,
## #   Varese <dbl>, Como <dbl>, Milano <dbl>,
## #   Bergamo <dbl>, Brescia <dbl>, Pavia <dbl>,
## #   Cremona <dbl>, Lecco <dbl>, Lodi <dbl>,
```

Long Data

```
italy_long <- pivot_longer(italy_covid, 2:26, names_to = "C

head(italy_long)
```

```
## # A tibble: 6 x 3
##   date       City          Cases
##   <date>     <chr>         <dbl>
## 1 2020-02-24 Napoli            0
## 2 2020-02-24 Piacenza          0
## 3 2020-02-24 Parma             0
## 4 2020-02-24 Reggio_Emilia     0
## 5 2020-02-24 Modena            0
## 6 2020-02-24 Bologna           0
```

Back to Wide

```
italy_wide <- pivot_wider(italy_long, names_from = City, va
head(italy_wide)
```

```
## # A tibble: 6 x 26
##   date       Napoli Piacenza Parma Reggio_Emilia
##   <date>      <dbl>    <dbl> <dbl>          <dbl>
## 1 2020-02-24      0        0     0              0
## 2 2020-02-25      0       18     4              0
## 3 2020-02-26      0       10     4              0
## 4 2020-02-27      3       35     2              0
## 5 2020-02-28      1       26    17              1
## 6 2020-02-29      9       49     8              3
## # ... with 21 more variables: Modena <dbl>,
## #   Bologna <dbl>, Rimini <dbl>, Roma <dbl>,
## #   Varese <dbl>, Como <dbl>, Milano <dbl>,
## #   Bergamo <dbl>, Brescia <dbl>, Pavia <dbl>,
## #   Cremona <dbl>, Lecco <dbl>, Lodi <dbl>,
## #   Monza <dbl>, Bolzano <dbl>, Torino <dbl>,
```

Pipes

Pipes allow you to chain multiple functions together to be run in sequence, with each function operating on the preceding function's output.

Pipes

Pipes

```
starwars %>%
  group_by(species) %>%
  summarise(avg_height = mean(height, na.rm = TRUE)) %>%
  arrange(avg_height)
```

```
## # A tibble: 38 x 2
##    species         avg_height
##    <chr>                <dbl>
##  1 Yoda's species          66
##  2 Aleena                  79
##  3 Ewok                    88
##  4 Vulptereen              94
##  5 Dug                    112
##  6 Xexto                  122
##  7 Droid                  131.
##  8 Toydarian              137
##  9 Sullustan              160
## 10 Toong                  163
```

Pipes

Without pipes you'd have to do something like this:

```
arrange(
  summarise(
    group_by(starwars, species),
    avg_height = mean(height, na.rm = TRUE)
  ),
  avg_height
)
```

Or like this:

```
x1 <- starwars
x2 <- group_by(x1, species)
x3 <- summarise(x3, avg_height = mean(height, na.rm = TRUE)
arrange(x3, avg_height)
```

Pipes

By default, pipes pass the left hand side to the first argument in the next function.

Pipe a result to a specific argument:

```
starwars %>%
  lm(mass ~ height, data = .)
```

```
##
## Call:
## lm(formula = mass ~ height, data = .)
##
## Coefficients:
## (Intercept)       height
##    -13.8103       0.6386
```

Plotting

Visualizing data is an important part of exploring and analyzing your data. We will cover the basics of both "base" R plotting and ggplot2.

Base R Plots

Type of Plot	Function
Scatter plot	plot(x,. . .)
Line charts	plot(x, type = “l”, . . .)
Histogram	hist(x,. . .)
Box plots	boxplot(x, . . .)
Bar chars	barplot(height, . . .)

Optional Arguments for Plotting

Customize these arguments in the "..." parts of plot functions

Option	Description
xlab = ""	x-axis label
ylab = ""	y-axis label
main = ""	Title label
col = ""	Set color
cex =	Change size of all elements
cex.lab =	Change label size
cex.axis =	Change tick mark label size
xlim = c(,)	Set x-axis limits
ylim = c(,)	Set y-axis limits
lty =	Change line type in line chart
lwd =	Change line width in line chart
pch =	Change point type in scatter plot

Add to existing plot

Option	Description
func(. . . , add = TRUE)	Add any plot function to existing plot
lines(x,. . .)	Add line
points(x,. . .)	Add points
title(main = , xlab = , ylab =)	Add labels
abline(h =)	Horizontal line at y=h
abline(v =)	Vertical line at x=v
abline(a = , b =)	Line with intercept a and slope b
abline(my_reg)	Best fit line from regression object

Multiple Plots at a time

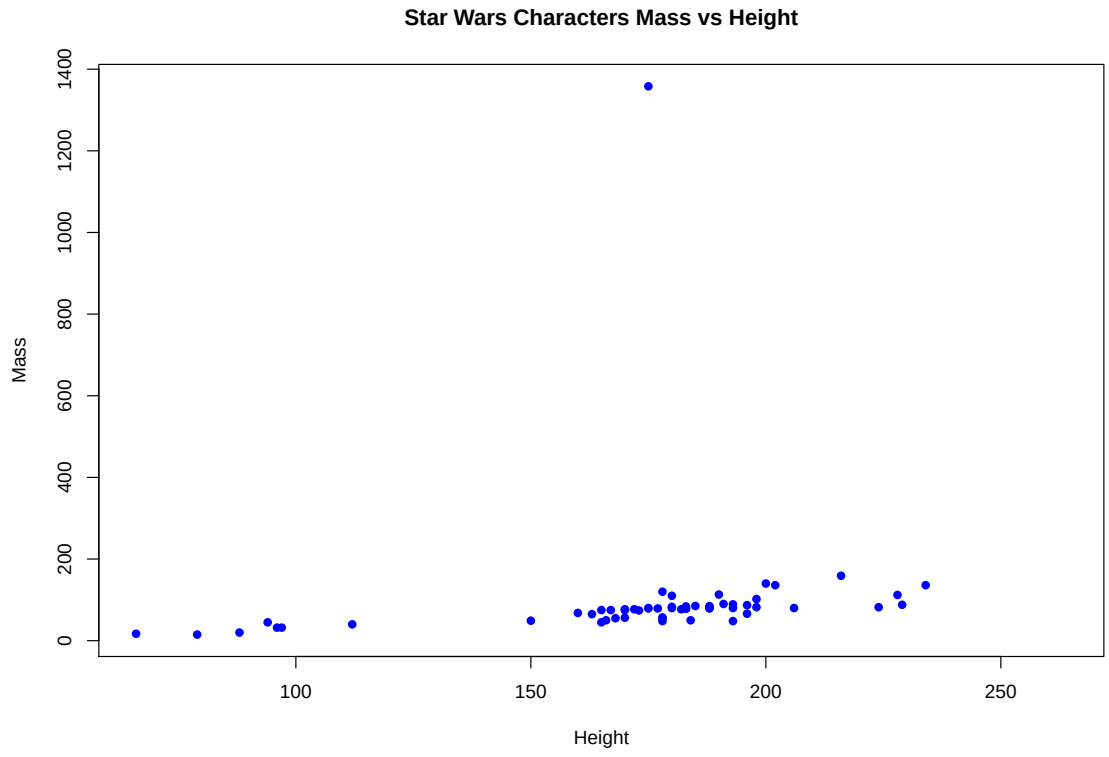
To include multiple plots in one image, you can change the number of plots using the "par" function

```
par(mfrow = c(nrow, ncol))
```

Plots called after this will be plotted on a grid of nrow and ncol

Example Plot

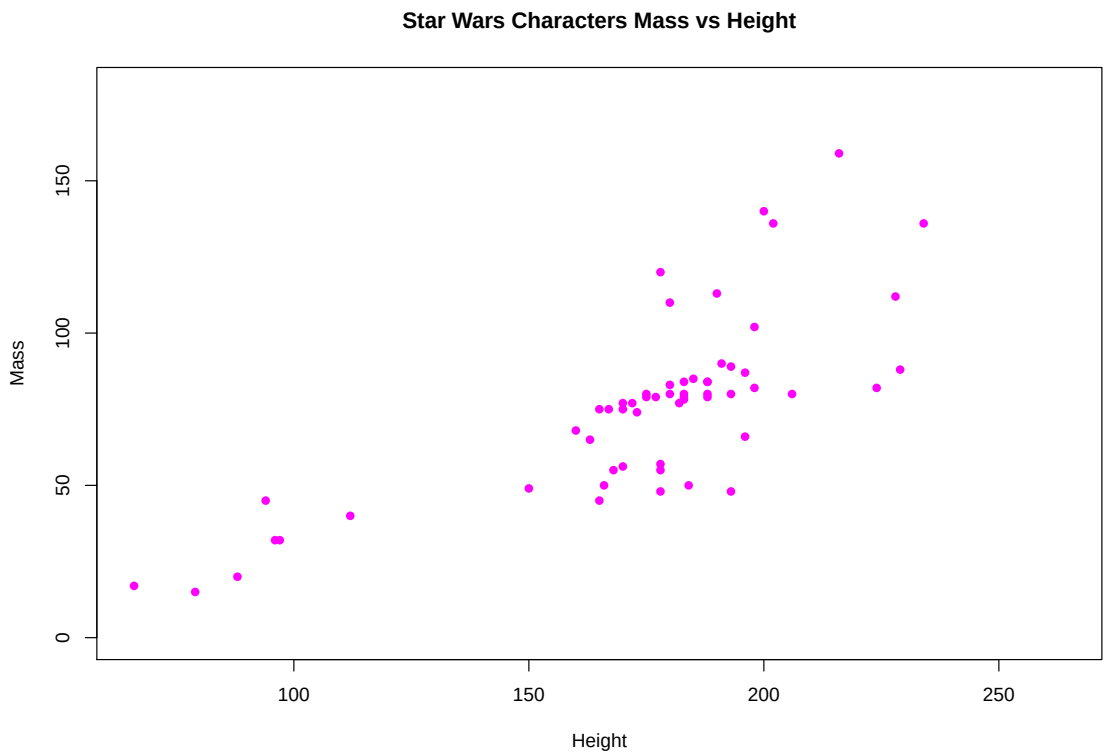
```
plot(starwars$height, starwars$mass,
     xlab = "Height", ylab = "Mass",
     main = "Star Wars Characters Mass vs Height",
     pch = 16, col = "blue")
```



Example Plot

Let's zoom in so we don't include the outlier in the plot

```
plot(starwars$height, starwars$mass,
     xlab = "Height", ylab = "Mass",
     main = "Star Wars Characters Mass vs Height",
     pch = 16, col = "magenta", ylim = c(0, 180))
```



ggplot2


ggplot2:
Build a data
MASTERPIECE
THEMES
GEOMS
aes
fill
color
ggplot
gganimate
plotly
HORST '18

ggplot2

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same components:

- A Data Set
- A Coordinate System
- Geoms (visual marks that represent data points)

Plots made in ggplot2 can be saved as objects and different layers can be added when plotting

ggplot2

You can use this same code template to make any plot:

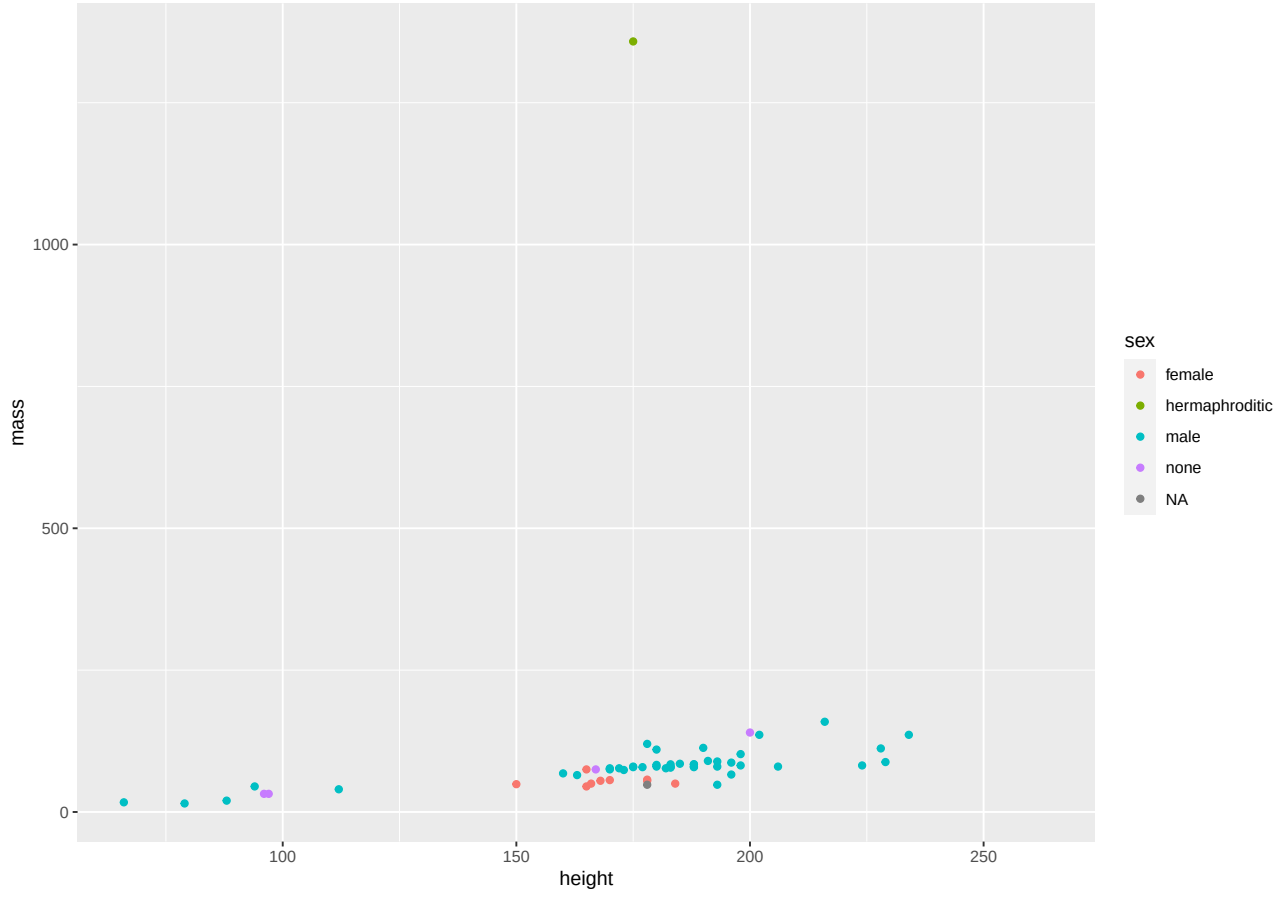
```
ggplot(data = <DATA>,
       aes(x = <X_VARIABLE>, y = <Y_VARIABLE>)) +
  <GEOM_FUNCTION>()
```

ggplot2

Component	Examples
aes	x, y, color, fill, size
geom	geom_line(), geom_point(), geom_col(), geom_boxplot(), geom_abline(), geom_hline(), geom_vline(), geom_histogram(), geom_density

ggplot2

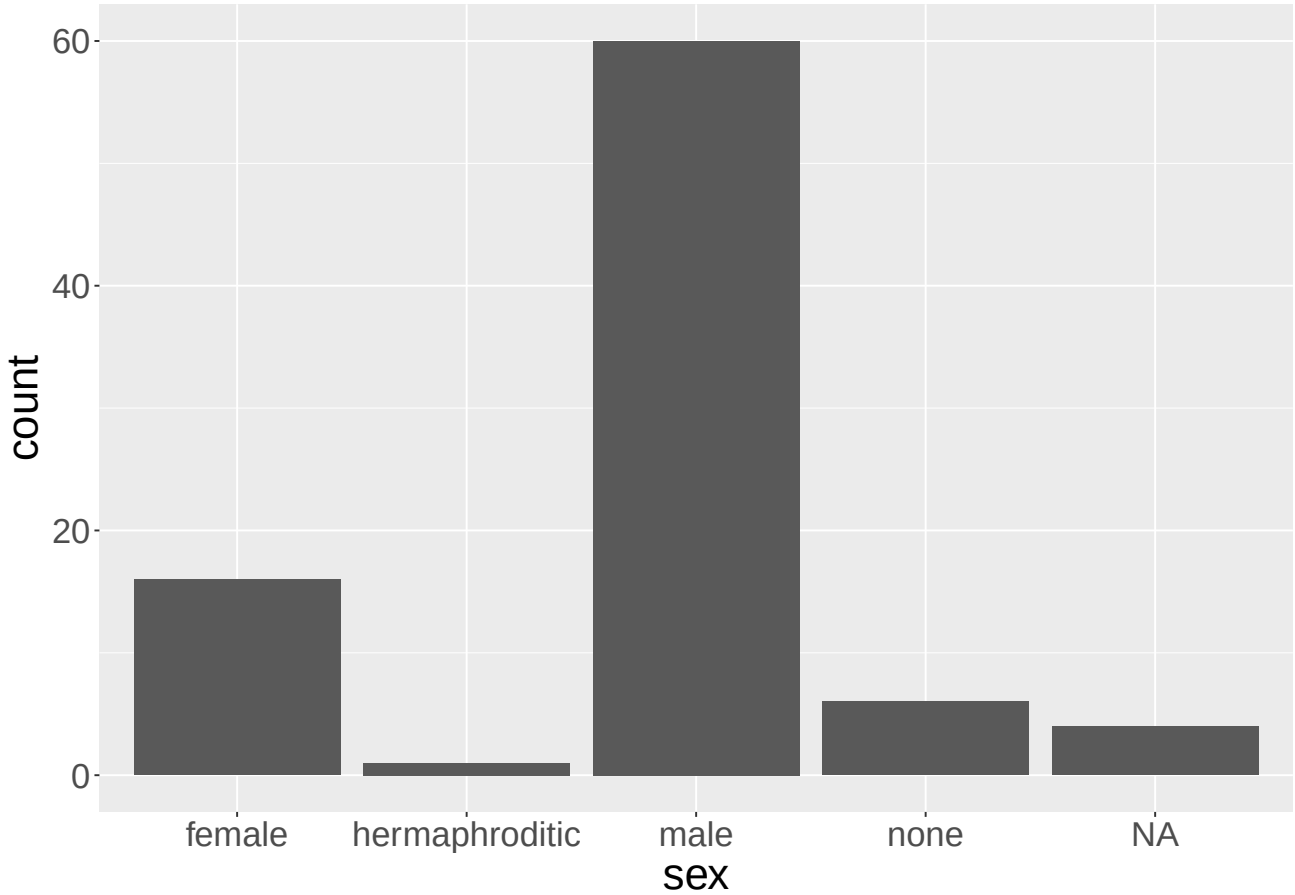
```
ggplot(starwars, aes(height, mass, color = sex)) +
  geom_point()
```



ggplot2

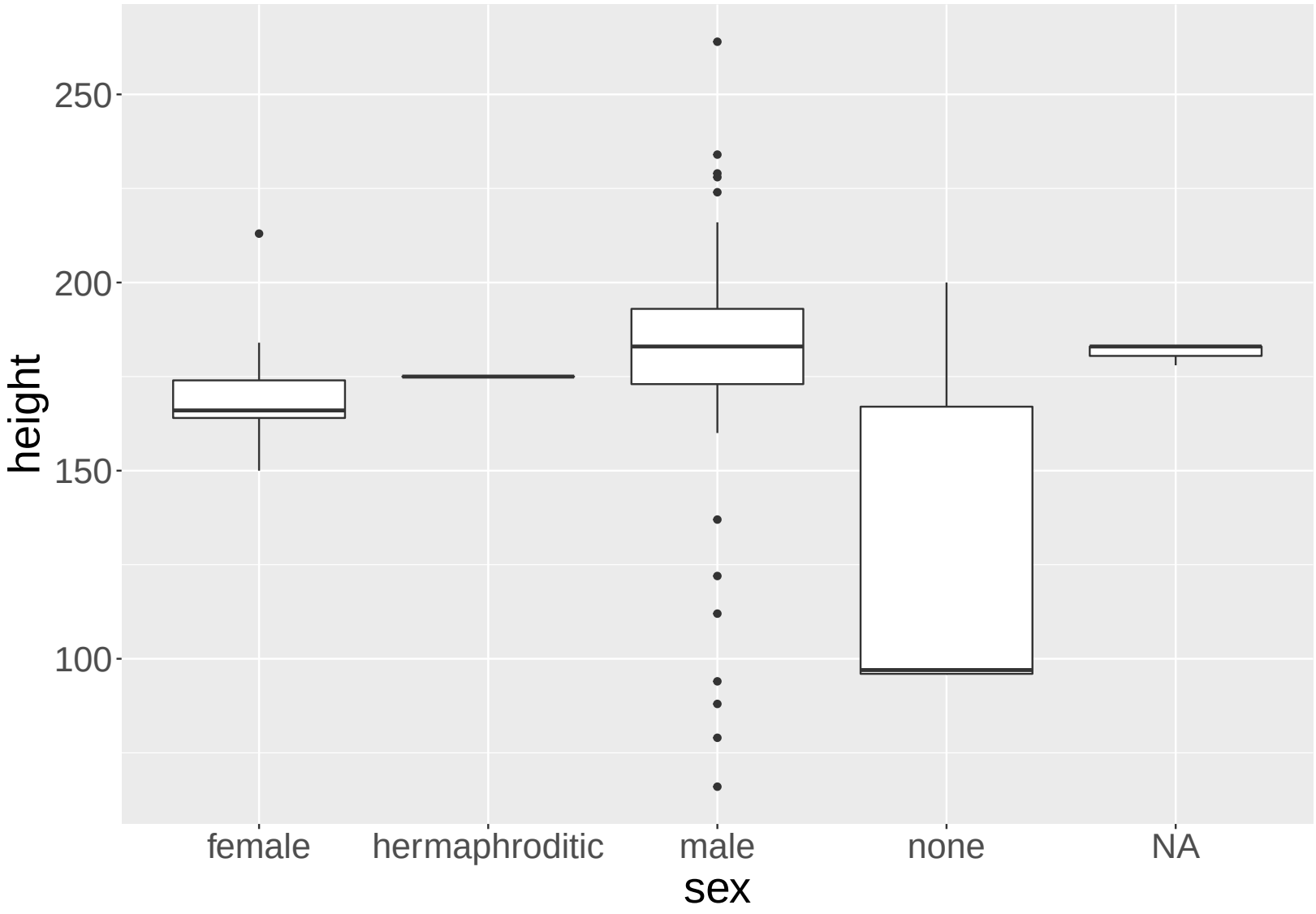
Make labels bigger using "theme"

```
ggplot(starwars, aes(sex)) +
  geom_bar() +
  theme(text = element_text(size = 24))
```



ggplot2

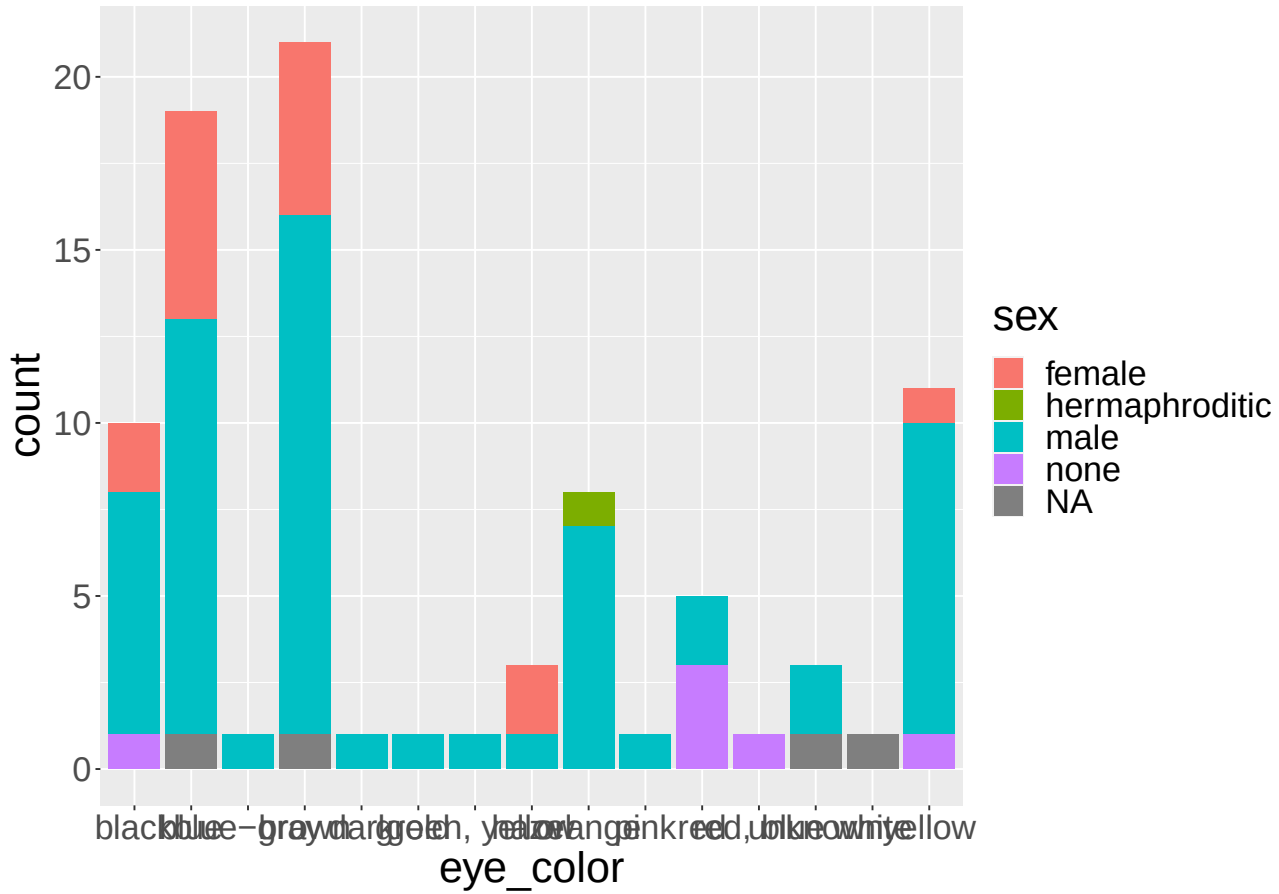
```
ggplot(starwars, aes(x = sex, y =height)) +
  geom_boxplot() + theme(text = element_text(size = 24))
```



ggplot2

The aesthetic can also be passed in the geom layer

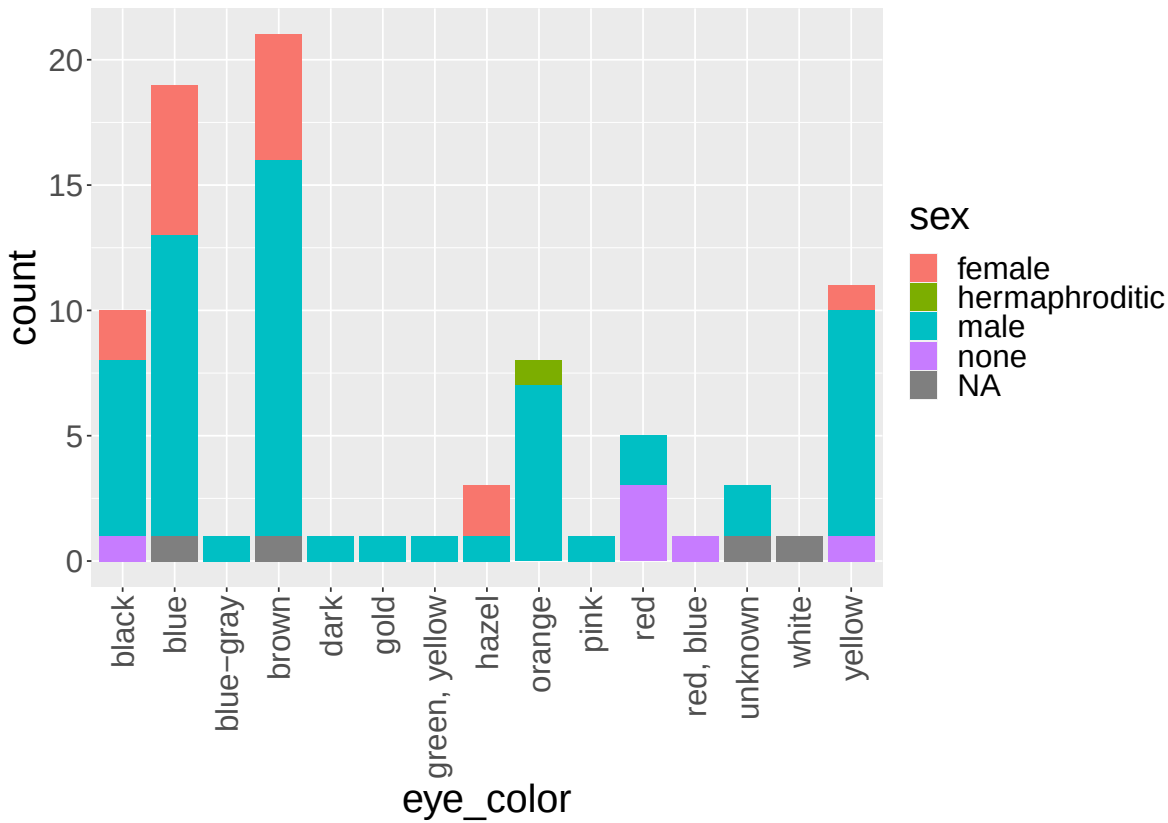
```
ggplot(starwars) + geom_bar(aes(eye_color, fill = sex)) +
  theme(text = element_text(size = 24))
```



ggplot2

Rotate x-axis labels

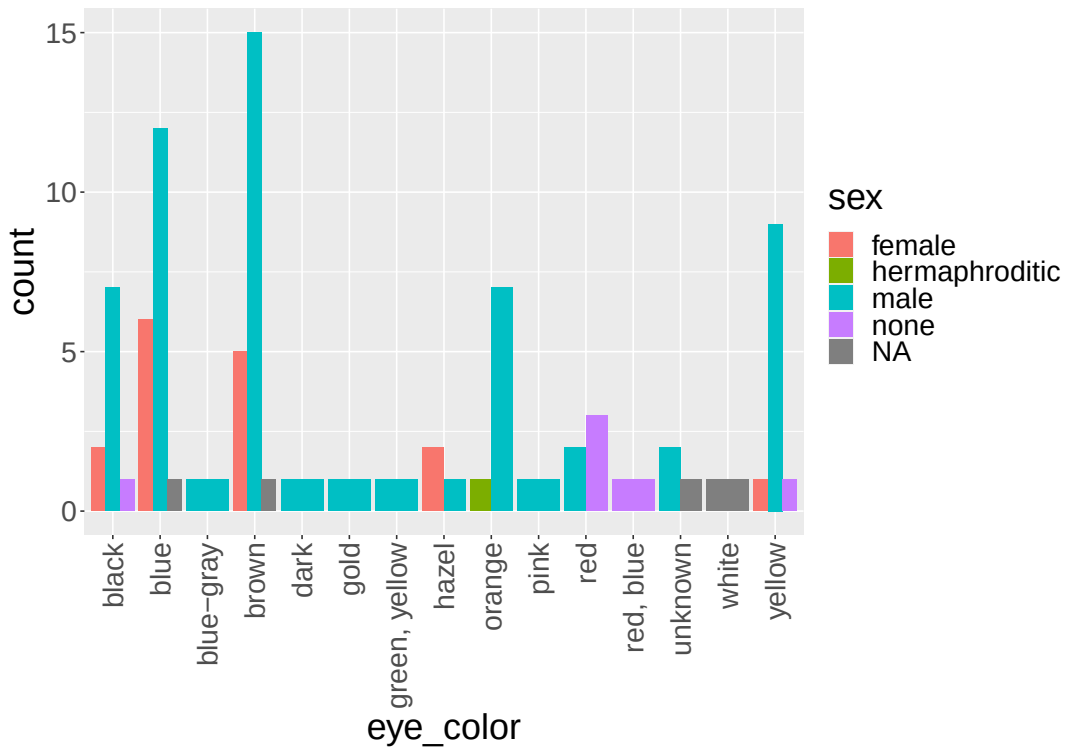
```
ggplot(starwars)+ geom_bar(aes(eye_color, fill = sex)) +
  theme(axis.text.x =
          element_text(angle = 90, vjust = 0.5, hjust=1),
        text = element_text(size = 24))
```



ggplot2

In bar or col plots, use position = "dodge" if you want side by side bars instead of stacked

```
ggplot(starwars) +
  geom_bar(aes(eye_color, fill = sex), position = "dodge")
  theme(text = element_text(size = 24), axis.text.x =
          element_text(angle = 90, vjust = 0.5, hjust=1))
```

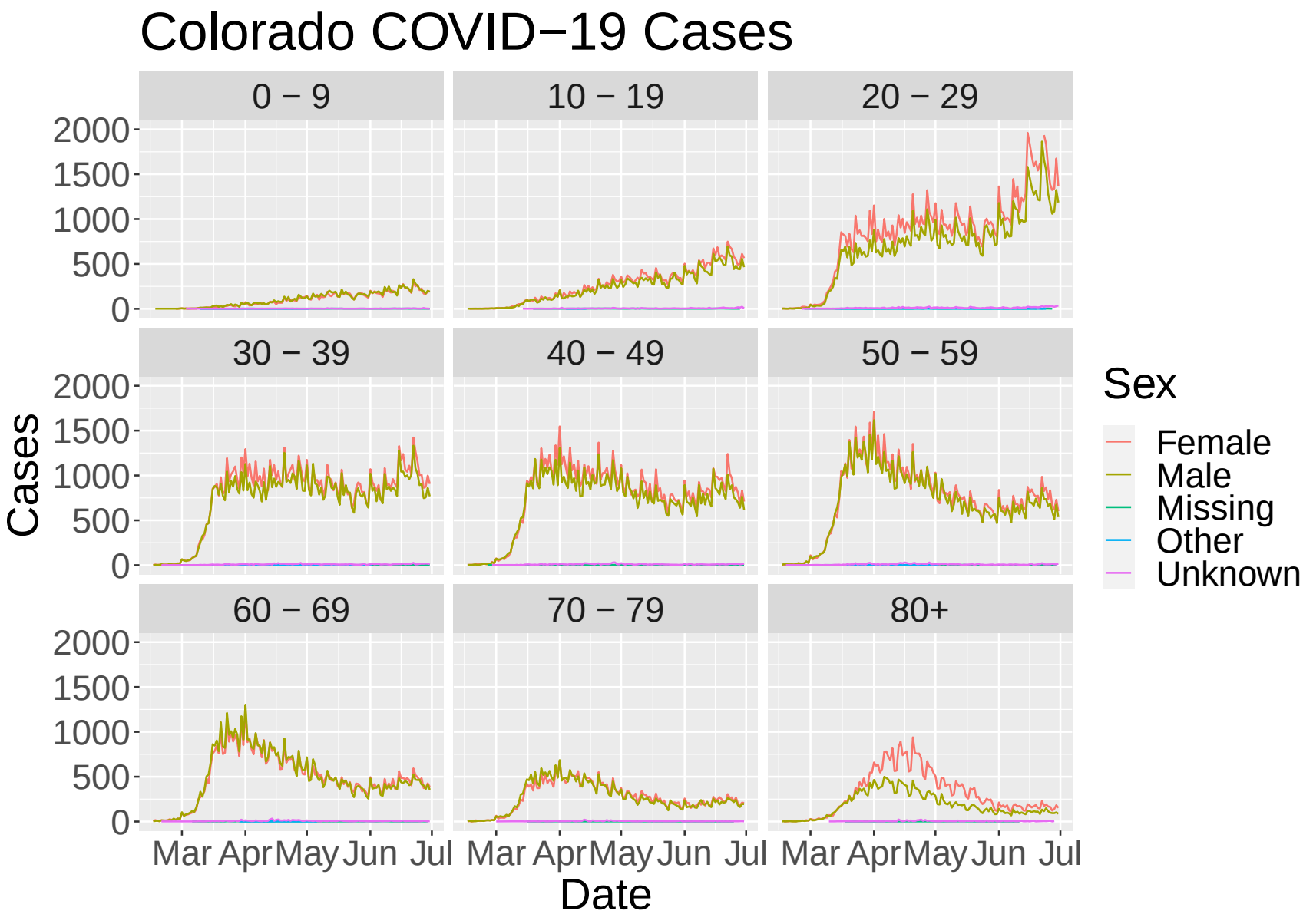


ggplot2

To make a grid of plots based on a factor, use facet_wrap or facet_grid

```
colorado_clean %>%
  ggplot(aes(date, cases, color = sex)) +
  geom_line() +
  facet_wrap(~age_group)+
  scale_y_continuous(limits = c(0,2000)) +
  labs(x = "Date", y = "Cases", color = "Sex",
       title = "Colorado COVID-19 Cases")
```

ggplot2


Colorado COVID−19 Cases
0 − 9
10 − 19
20 − 29
30 − 39
40 − 49
50 − 59
60 − 69
70 − 79
80+
2000
1500
1000
500
0
Cases
Mar
Apr
May
Jun
Jul
Date
Sex
Female
Male
Missing
Other
Unknown

Some Other Useful/Fun Packages

Package	Description
broom	Turns messy statistical model outputs into clean tibbles
fields	Curve, surface, and function fitting
beepr	Plays fun noises when your code finishes running
praise	Gives you praise when you feel like you need it
janitor	Quickly reformat all column names to be the same (snake_case, camelCase, etc)

R Community

* All artwork is courtesy of Allison Horst, Resident Artist at RStudio

R Community

The R community is incredibly supportive, active, and fun!

Virtually:

- Follow #rstats on twitter!
- Participate in Tidy Tuesday

In person meetups (currently on hold. . .):

- Denver R User Group
- R Ladies Denver

Cheat Sheets

Basics of R: https://rstudio.com/wp-content/uploads/2016/10/r-cheat-sheet-3.pdf

RMarkdown: https://github.com/rstudio/cheatsheets/raw/master/rmarkdown-2.0.pdf

ggplot2: https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf

dplyr: https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

Many Others: https://rstudio.com/resources/cheatsheets/