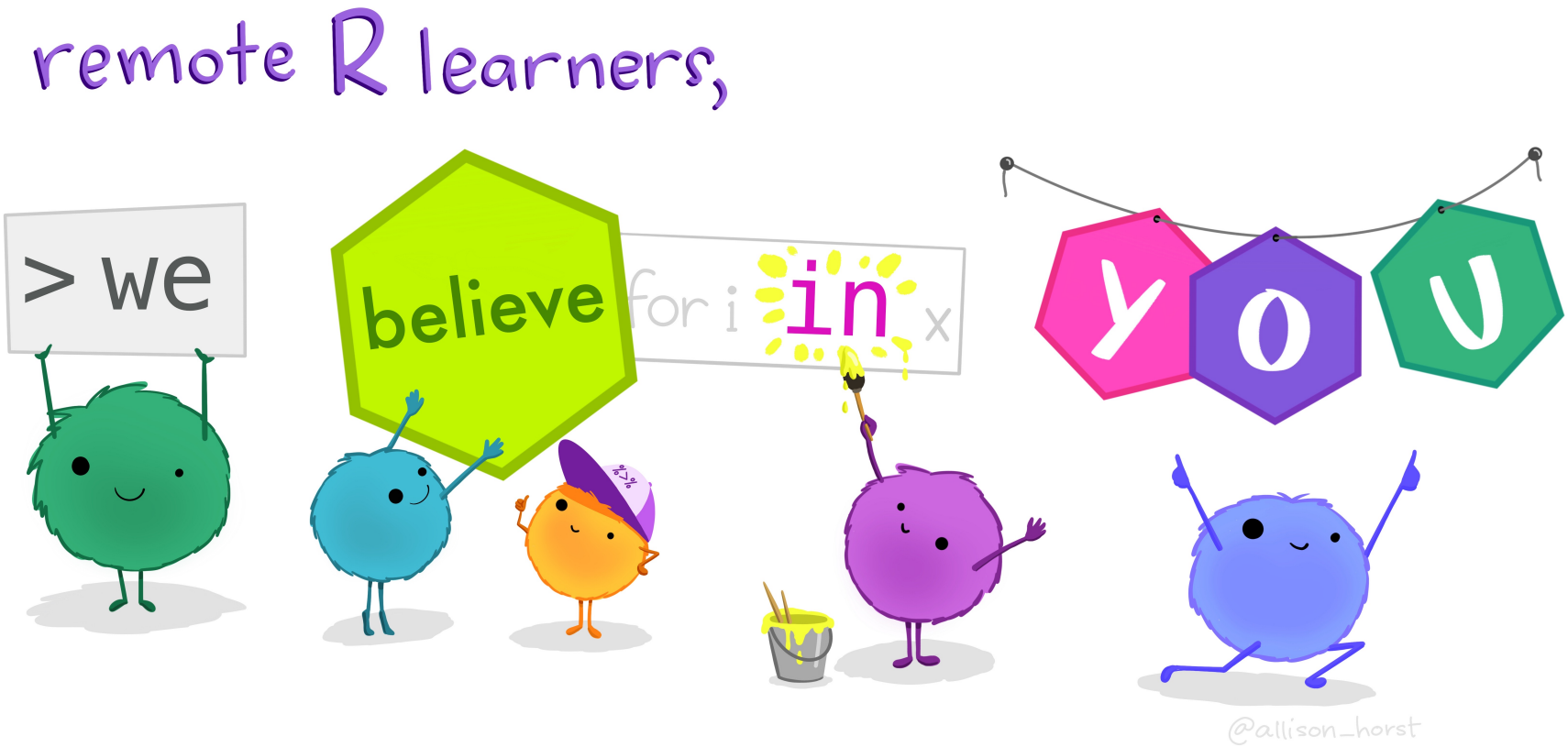
UNIVERSITY OF CAMBRIDGE
Bioinformatics Training

Introduction to for Biologists

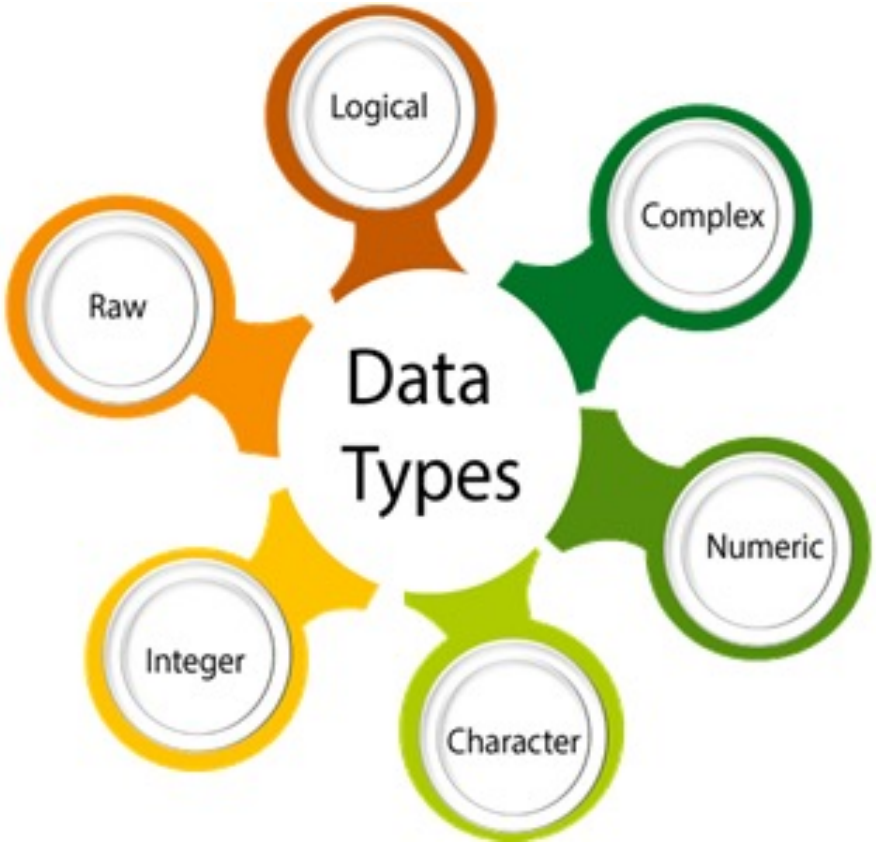
WORKFLOW & VISUALISATION

Author: Gita Yadav

Indebted to Alison Horst!


remote R learners,
> we
believe
for i in x
y
o
u
@allison_horst

Data Type (Recap)



Recap: Reading in Data from a file

- We've **looked** at the **raw** format of the file (CSV format)
- Let us **Load** the data into R
- Use function read.csv() to load the data into an object of class data.frame

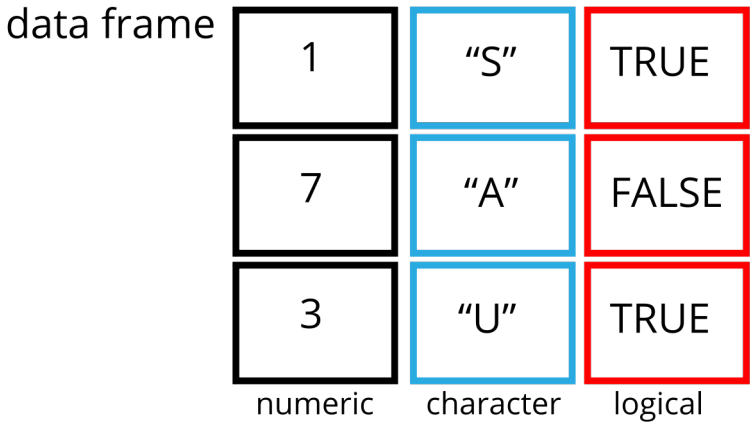
```
surveys <- read.csv("data/portal_data_joined.csv")
```

- Look at the data

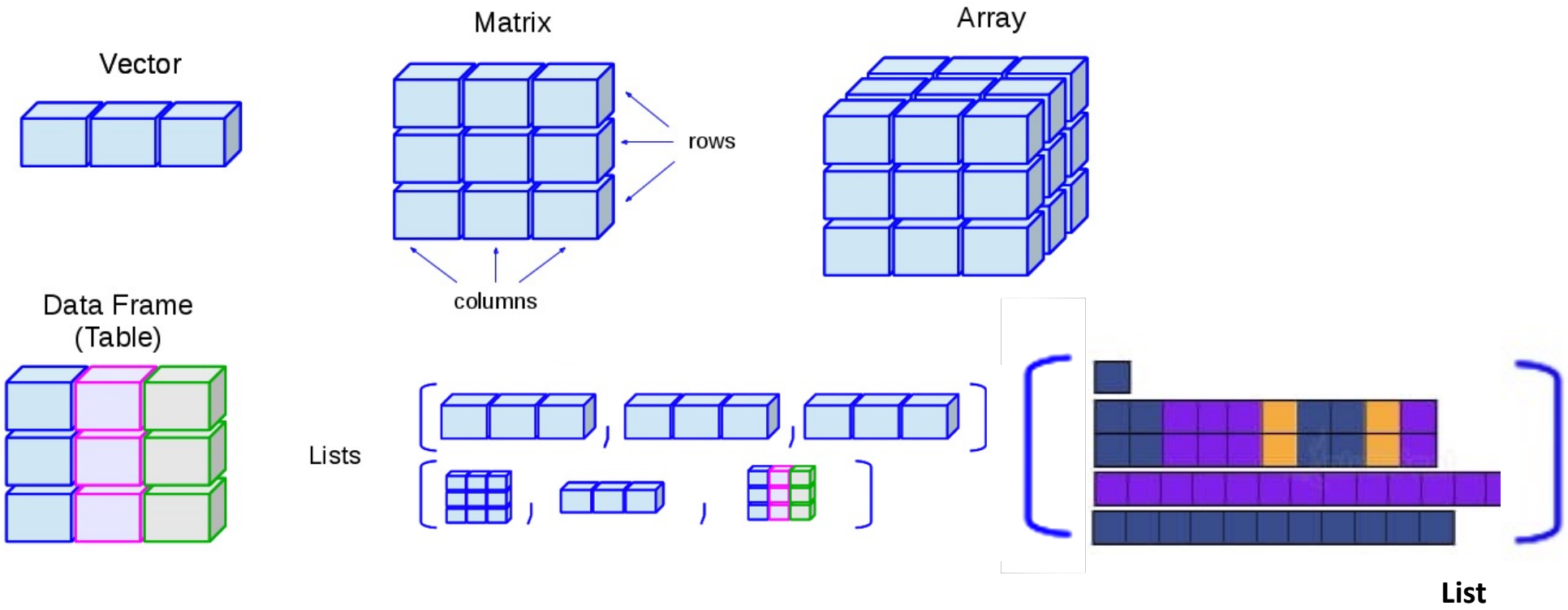
```
head(surveys)
```

```
## Try also
View(surveys)
```

Recap Data frame



Data Structures (Recap)



STRUCTURE of a data.frame Object

```
str(surveys)
```

```
#> 'data.frame':    34786 obs. of  13 variables:
#>  $ record_id      : int  1 72 224 266 349 363 435 506 588 661 ...
#>  $ month          : int  7 8 9 10 11 11 12 1 2 3 ...
#>  $ day            : int  16 19 13 16 12 12 10 8 18 11 ...
#>  $ year           : int  1977 1977 1977 1977 1977 1977 1977 1978 1978 1978 ...
#>  $ plot_id        : int  2 2 2 2 2 2 2 2 2 2 ...
#>  $ species_id     : chr  "NL" "NL" "NL" "NL" ...
#>  $ sex            : chr  "M" "M" "" "" ...
#>  $ hindfoot_length: int  32 31 NA NA NA NA NA NA NA NA ...
#>  $ weight         : int  NA NA NA NA NA NA NA NA 218 NA ...
#>  $ genus          : chr  "Neotoma" "Neotoma" "Neotoma" "Neotoma" ...
#>  $ species        : chr  "albigula" "albigula" "albigula" "albigula" ...
#>  $ taxa           : chr  "Rodent" "Rodent" "Rodent" "Rodent" ...
#>  $ plot_type      : chr  "Control" "Control" "Control" "Control" ...
```

Inspecting a data.frame object

- Size:
  - `dim(surveys)` - returns a vector with the number of rows in the first element, and the number of columns as the second element (the **dim**ensions of the object)
  - `nrow(surveys)` - returns the number of rows
  - `ncol(surveys)` - returns the number of columns
- Content:
  - `head(surveys)` - shows the first 6 rows
  - `tail(surveys)` - shows the last 6 rows
- Names:
  - `names(surveys)` - returns the column names (synonym of `colnames()` for `data.frame` objects)
  - `rownames(surveys)` - returns the row names
- Summary:
  - `str(surveys)` - structure of the object and information about the class, length and c content of each column
  - `summary(surveys)` - summary statistics for each column

Indexing and sub-setting data frames

```
> head(surveys)
  record_id month day year plot_id species_id sex hindfoot_length weight   genus  species   taxa plot_type
1         1     7  16 1977       2         NL   M              32     NA Neotoma albigula Rodent   Control
2        72     8  19 1977       2         NL   M              31     NA Neotoma albigula Rodent   Control
3       224     9  13 1977       2         NL                  NA     NA Neotoma albigula Rodent   Control
4                     077       2         NL                  NA     NA Neotoma albigula Rodent   Control
5                             2         NL                  NA     NA Neotoma albigula Rodent   Control
6       363                   2         NL                  NA     NA Neotoma albigula Rodent   Control
>
```

	record_id	month	day
	[1,1]	[1,2]	[1,3]
1	1	7	16
	[2,1]	[2,2]	[2,3]
2	72	8	19
	[3,1]	[3,2]	[3,3]
3	224	9	13

colnames(surveys)

index [R,C]

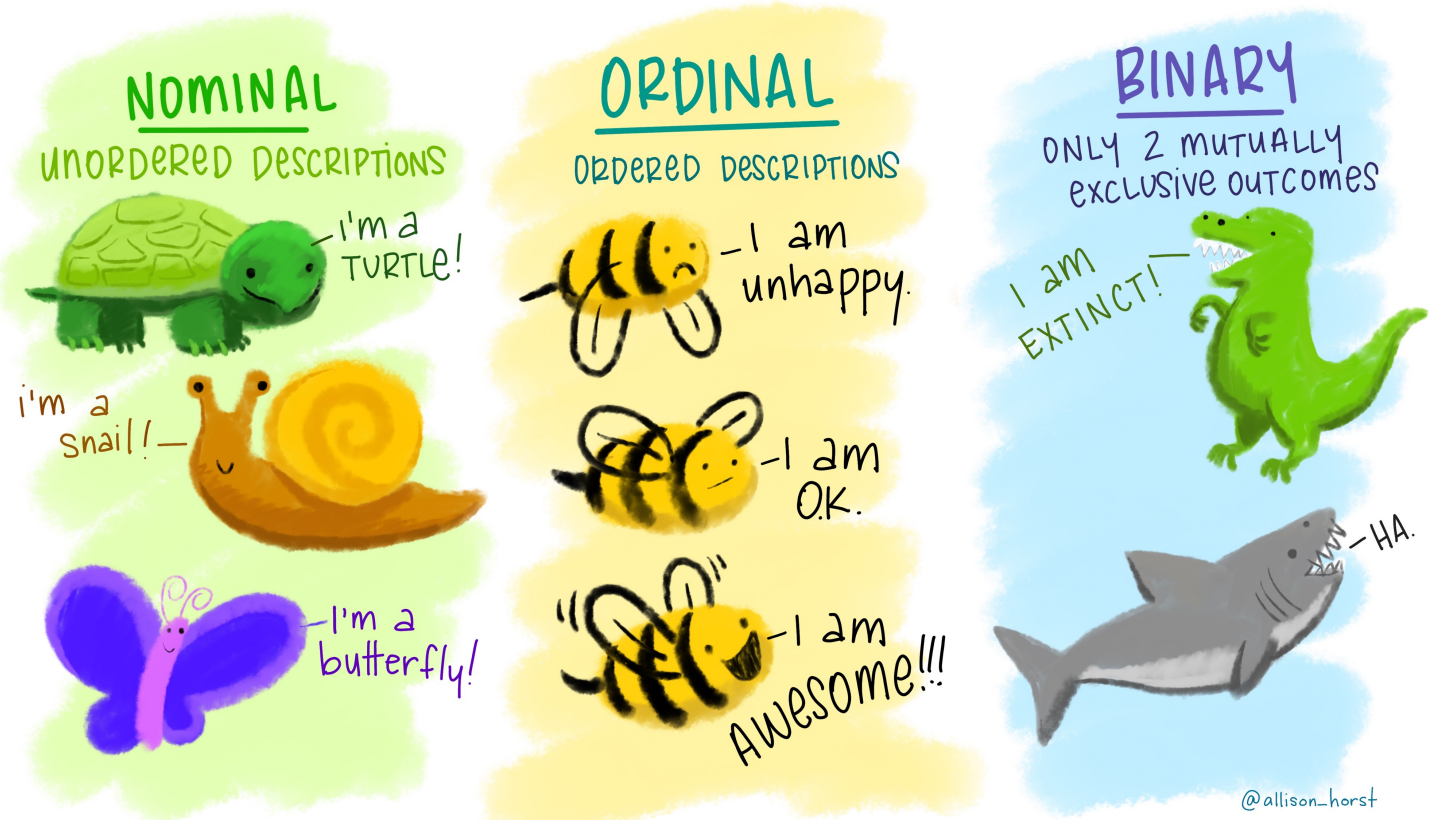
rownames(surveys)

Numeric Indexing [1,2]

Name indexing $

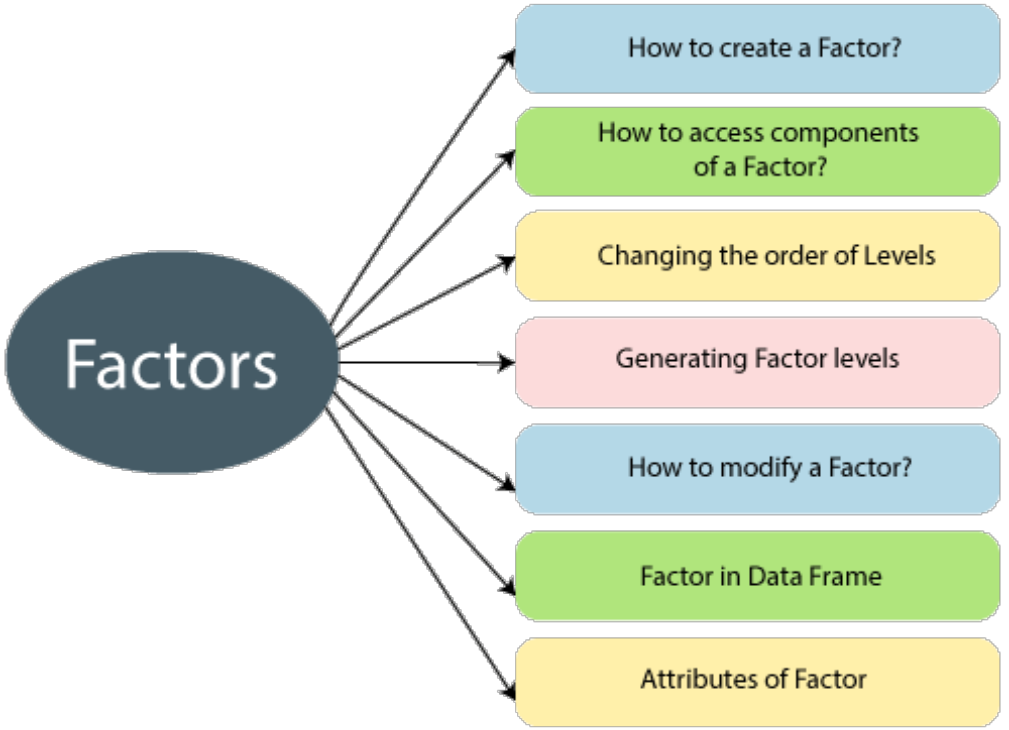
Logical indexing == | >=

The Strange Case of "Categorical Variables"!

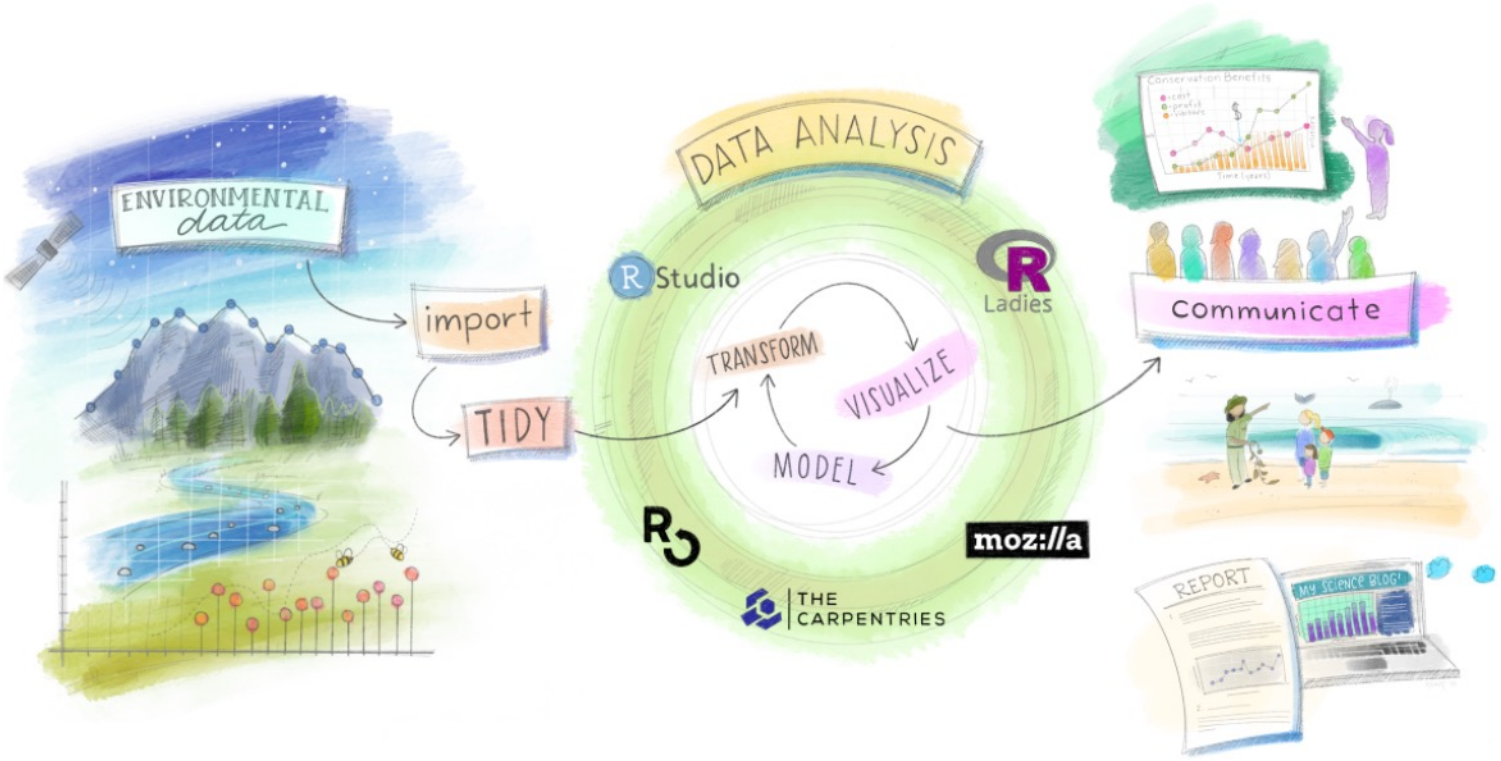


Factors

- **Categorical** Variables in Statistics
- Can take Limited set of Values (levels)
  - Example:
    - "Gender" = {Male, Female}
    - "Meal" = {Breakfast, Lunch, Dinner}
- No Intrinsic Ordering (alphabetical)
- Order can be changed
  - Why/When would you wish this!?

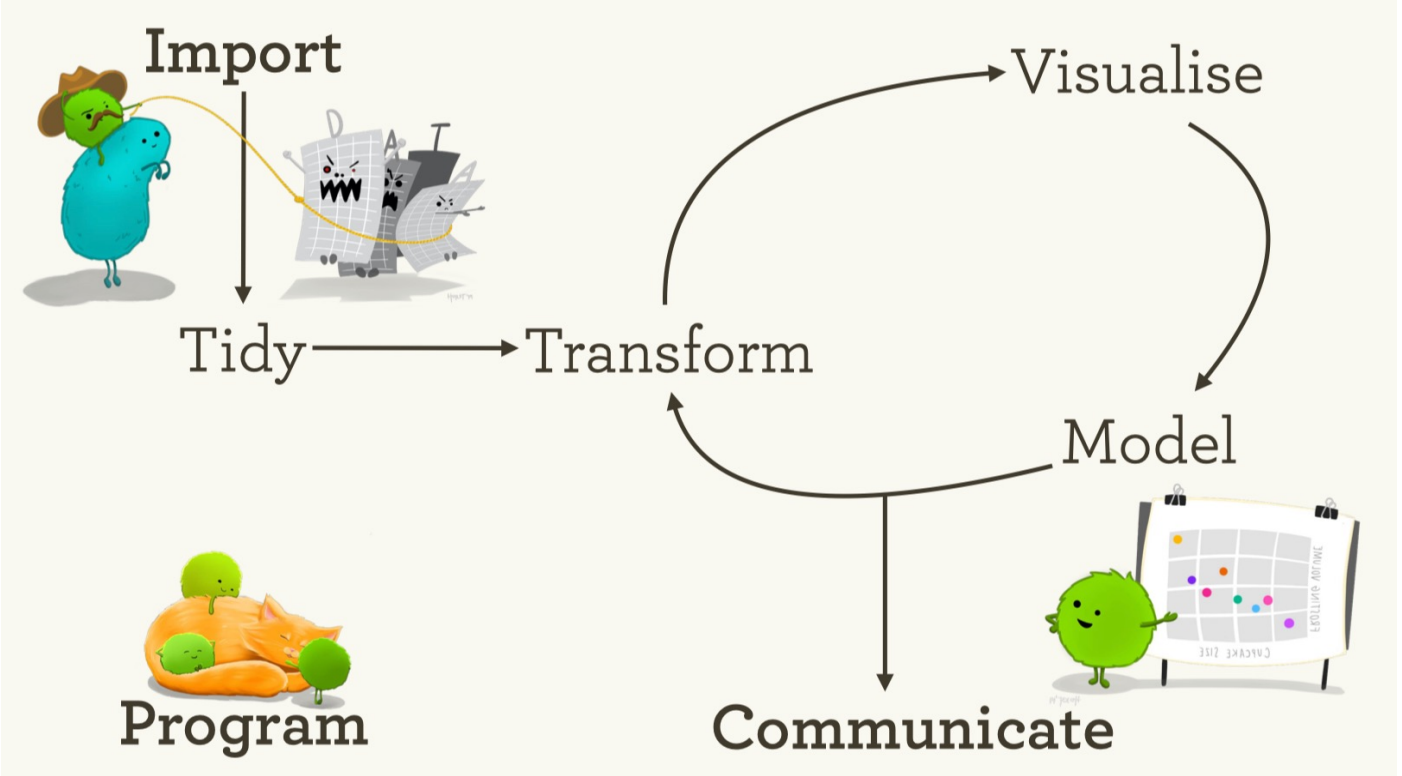


The Data Exploration Workflow



art: @allison_horst; updated from Wickham & Grolemund

Learning Data Science!

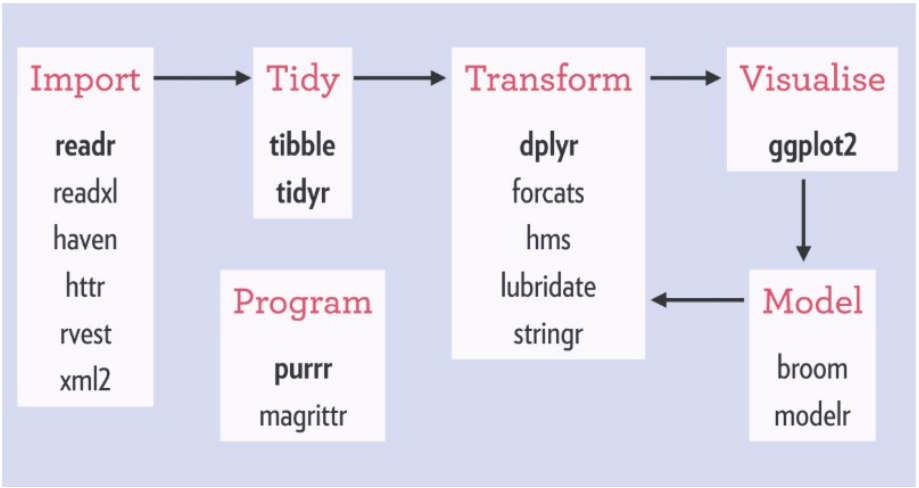


Art: Alison Horst (GitHub)

Packages

- In-built R Functions: R's **R base**.
  Eg. What we've been using till now: str() and head() and so on
- External functions are enabled through **Packages**
- **Packages in R:** Additional functions that let you do more stuff!

- **Tidyverse (2016)** is an "Umbrella Package"
  Lets you perform the entire
  Data exploration Workflow!



TIDYVERSE

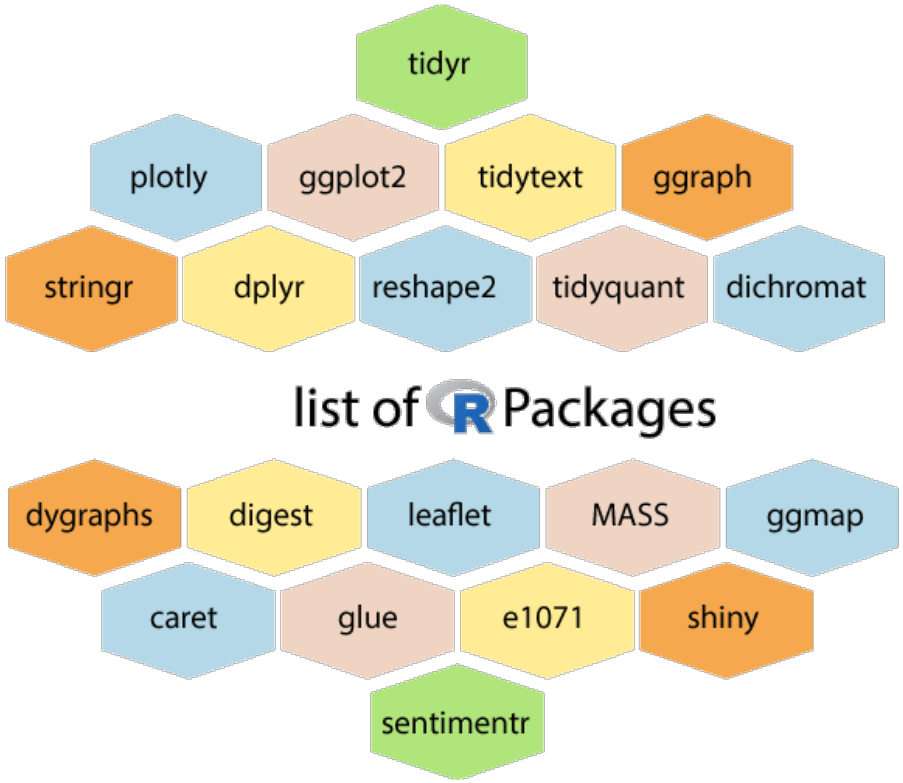
Tidyverse

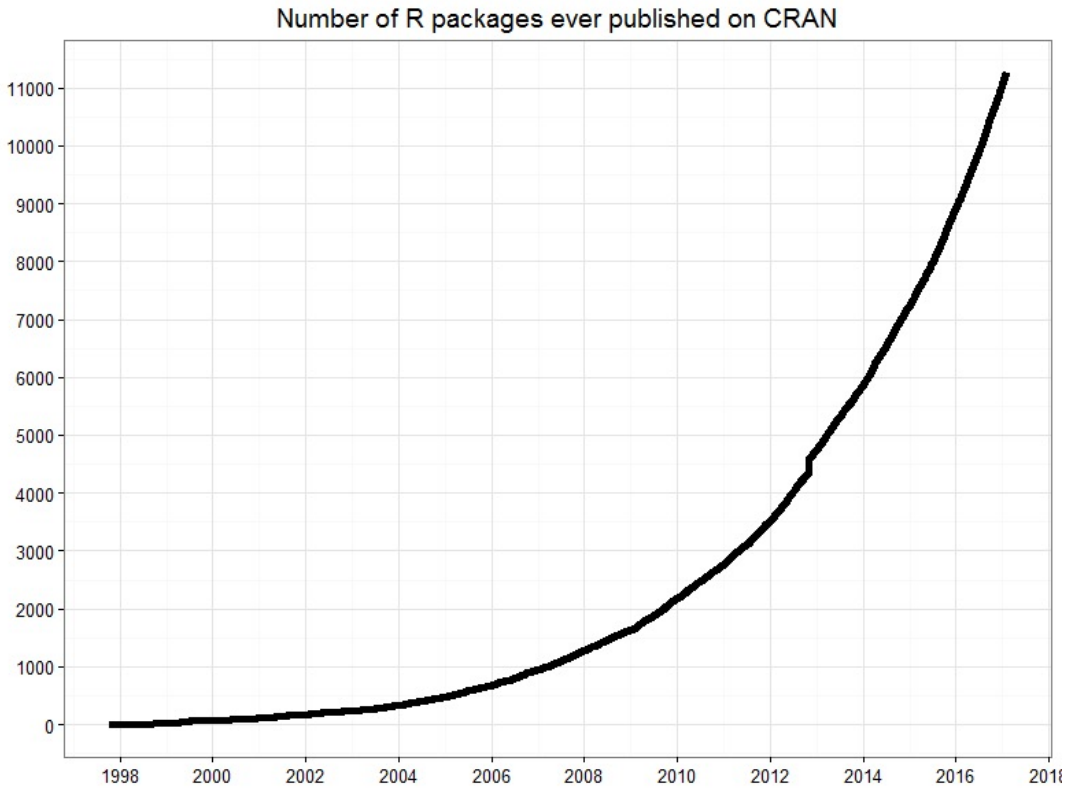
- Easier to read
- 10x Faster

```r
# R Base
surveys[surveys$species=="albigula" &
        surveys$year==1977, ]

# tidyverse
filter(surveys, species=="albigula" & year==1977)
```

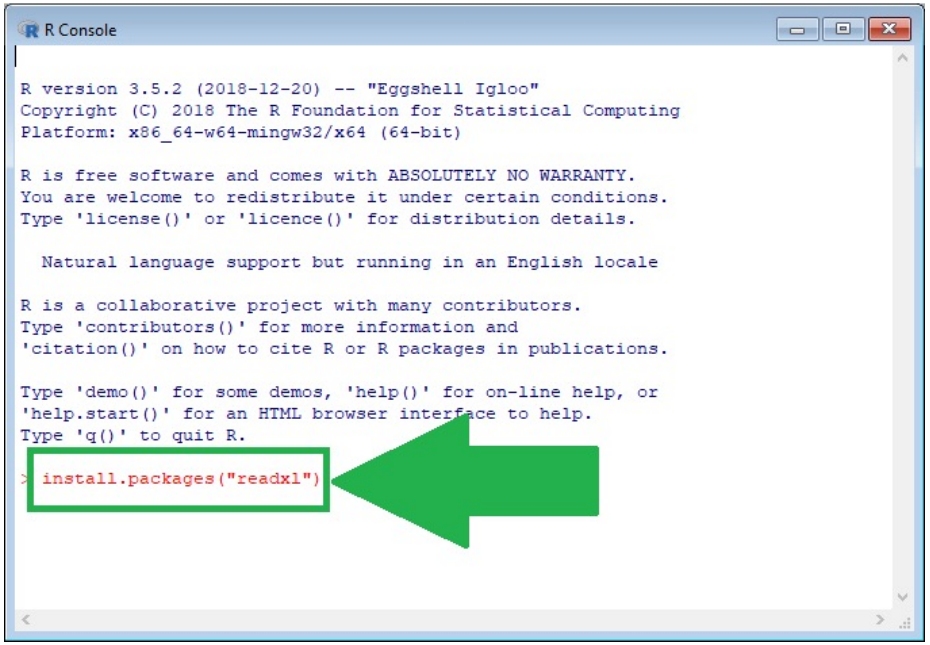
R Packages


tidyr
plotly
ggplot2
tidytext
ggraph
stringr
dplyr
reshape2
tidyquant
dichromat
list of R Packages
dygraphs
digest
leaflet
MASS
ggmap
caret
glue
e1071
shiny
sentimentr


Number of R packages ever published on CRAN
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016

Installing Packages

- Directly from R console
- No need to Repeat or Re-Install

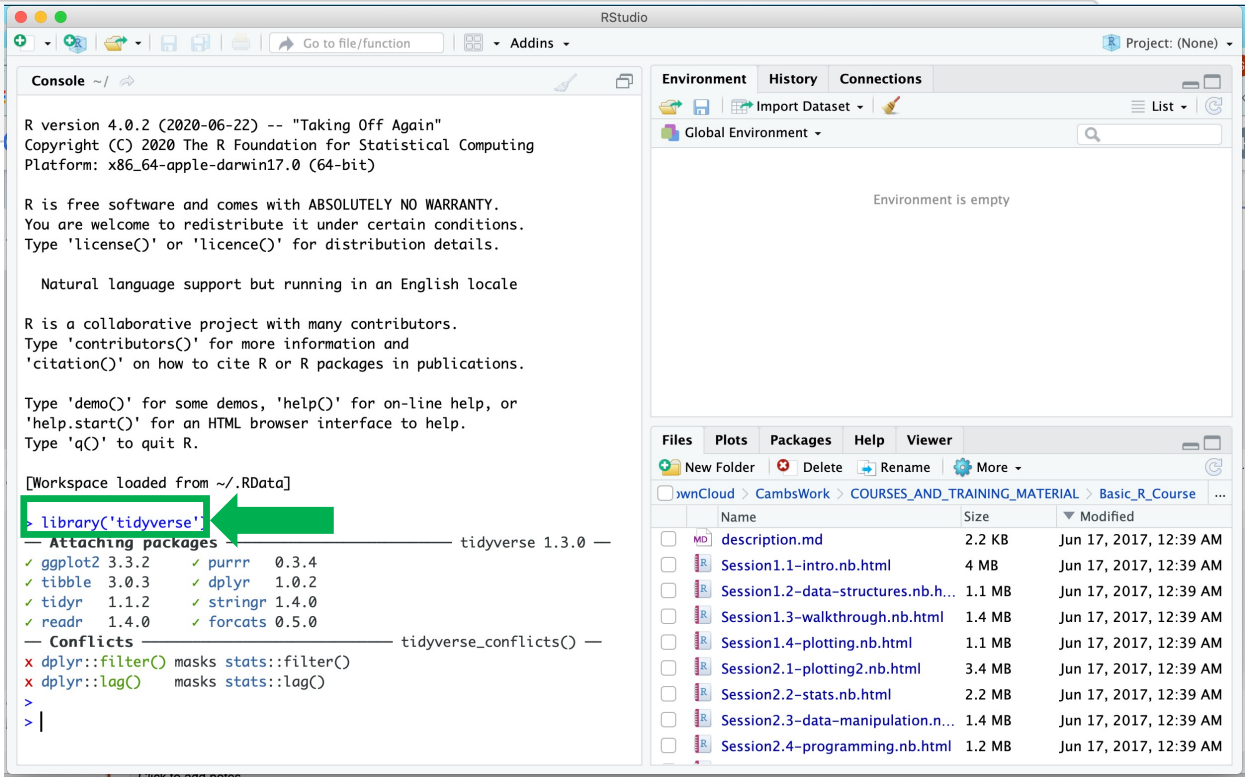



```
#install the tidyverse package
install.packages("tidyverse")
```

Loading Packages

```
## load the tidyverse package
library(tidyverse)
```

Try it !



Importing / Reading Data from Files

```
surveys <- read_csv("data/portal_data_joined.csv")
```

- The **read_csv** function is from tidyverse package **readr**
- Instead of **read.csv (**R base)

What's a tibble

- Object of class tbl
- A reimagining of the data.frame
- Looks and acts like a data.frame
- ..but its even better
- Retains useful features
- drops frustrating features
- tidyverse works with tibbles

```
Console ~/ownCloud/CambsWork/COURSES_AND_TRAINING_MATERIAL/Basic_R_Course/TESTR/
>
> surveys <- read_csv("portal_data_joined.csv")

── Column specification ────────────────────────────────────────────────────────
cols(
  record_id = col_double(),
  month = col_double(),
  day = col_double(),
  year = col_double(),
  plot_id = col_double(),
  species_id = col_character(),
  sex = col_character(),
  hindfoot_length = col_double(),
  weight = col_double(),
  genus = col_character(),
  species = col_character(),
  taxa = col_character(),
  plot_type = col_character()
)

> View(surveys)
```

tibble

Display first 6 rows
head(surveys)

To print the first 15 rows
print(surveys, n=15)

- Provides Information on
  - data structure
  - data types of each column
  - Size of the dataset
- Str() function redundant!

```
> 
> head(surveys)
# A tibble: 6 x 13
  record_id month   day  year plot_id species_id sex   hindfoot_length weight genus species taxa
      <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl> <chr> <chr>   <chr>
1         1     7    16  1977       2 NL         M                  32     NA Neot… albigu… Rode…
2        72     8    19  1977       2 NL         M                  31     NA Neot… albigu… Rode…
3       224     9    13  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
4       266    10    16  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
5       349    11    12  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
6       363    11    12  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
# … with 1 more variable: plot_type <chr>
> 
> 
> print(surveys)
# A tibble: 34,786 x 13
   record_id month   day  year plot_id species_id sex   hindfoot_length weight genus species taxa
       <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl> <chr> <chr>   <chr>
 1         1     7    16  1977       2 NL         M                  32     NA Neot… albigu… Rode…
 2        72     8    19  1977       2 NL         M                  31     NA Neot… albigu… Rode…
 3       224     9    13  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
 4       266    10    16  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
 5       349    11    12  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
 6       363    11    12  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
 7       435    12    10  1977       2 NL         NA                 NA     NA Neot… albigu… Rode…
 8       506     1     8  1978       2 NL         NA                 NA     NA Neot… albigu… Rode…
 9       588     2    18  1978       2 NL         M                  NA    218 Neot… albigu… Rode…
10       661     3    11  1978       2 NL         NA                 NA     NA Neot… albigu… Rode…
# … with 34,776 more rows, and 1 more variable: plot_type <chr>
> 
```

DATA VISUALIZATION IN R

Visualizing data in R (ggplot)

- THREE main elements:

```
ggplot (data = <DATA>, mapping = aes(<MAPPINGS>)) + <GEOM_FUNCTION>()
```

- The function takes TWO arguments:

  data: Data frame with variables to plot (columns and rows)
  mapping: Aesthetics (How variables are mapped to visual properties of the plot)

- Using **ggplot** function on its own will not plot anything!

- Add a **geom_function** as a layer (By using +)
  geom_function specifies the type of plot would you like to plot

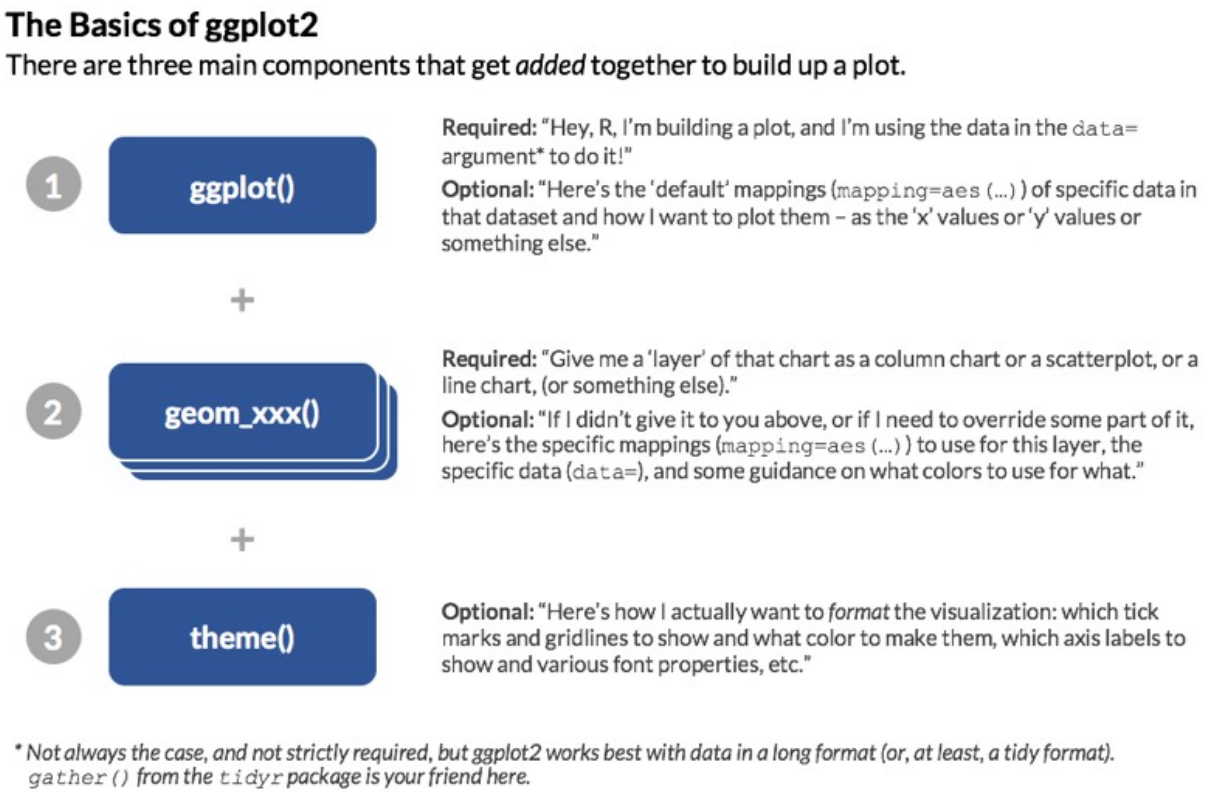
Visualizing data in R

Greatest advantage of ggplot:

- Easy to change plot types!

(New **geom_function** all else same)

[Order of Steps 2 & 3 does not matter]



Visualizing data in R

Main aspects:

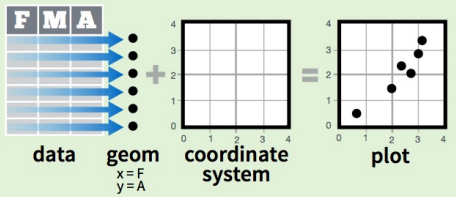
Data source - A tidy data source in long format
Geoms - geometric objects, the type of plot to produce. Line charts, bar charts, tiled plots etc.
Aesthetics - this specifies which variables in your data will vary and be plotted. (Misnomer – you actually control the look and feel of plots using *themes!)*

Other aspects (Not discussed today)
Themes - Like styles/CSS in HTML; Tweak / refine font, colors, spacing etc
Coordinate systems - Not just the usual x-y, but polar and more exotic systems
Scales - How your variables map onto the coordinate system (e.g. a log scale)
Statistics - Applied to data before plotting [eg. Binning for histograms]

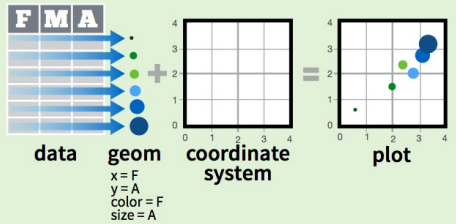
Thinking about what you want to produce via the components above will get you to your desired plot quicker

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**

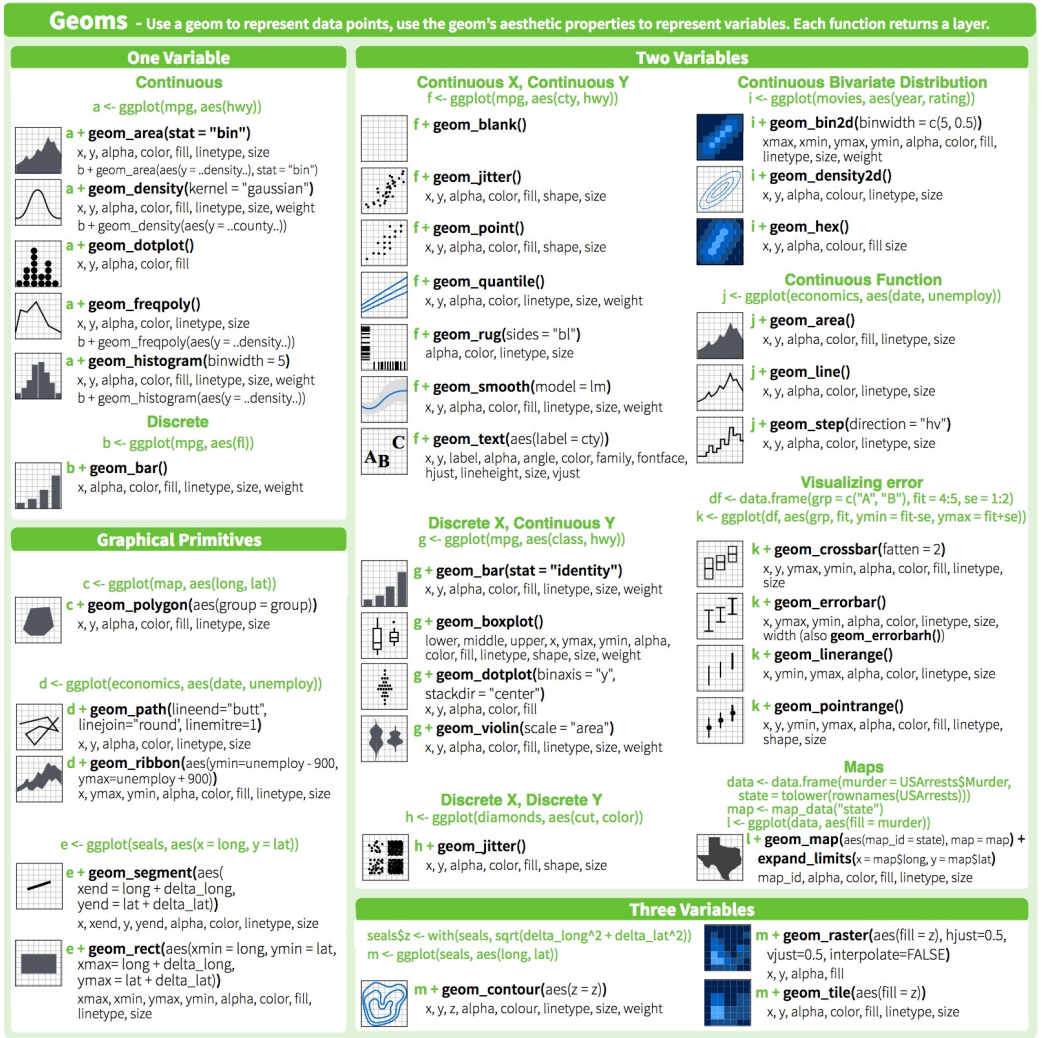


To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Visualizing data in R

Think of GEOMS as Layers!



Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + **geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + **geom_dotplot()**
x, y, alpha, color, fill

a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + **geom_histogram(**binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + **geom_polygon(**aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + **geom_path(**lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

d + **geom_ribbon(**aes(ymin=unemploy - 900, ymax=unemploy + 900**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + **geom_segment(**aes(xend = long + delta_long, yend = lat + delta_lat**)**
x, xend, y, yend, alpha, color, linetype, size

e + **geom_rect(**aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + **geom_blank()**

f + **geom_jitter()**
x, y, alpha, color, fill, shape, size

f + **geom_point()**
x, y, alpha, color, fill, shape, size

f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight

f + **geom_rug(**sides = "bl"**)**
alpha, color, linetype, size

f + **geom_smooth(**model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + **geom_text(**aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + **geom_violin(**scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + **geom_jitter()**
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + **geom_bin2d(**binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + **geom_area()**
x, y, alpha, color, fill, linetype, size

j + **geom_line()**
x, y, alpha, color, linetype, size

j + **geom_step(**direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + **geom_crossbar(**fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + **geom_errorbar()**
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + **geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size

k + **geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + **geom_map(**aes(map_id = state), map = map**) +**
expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

m + **geom_contour(**aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

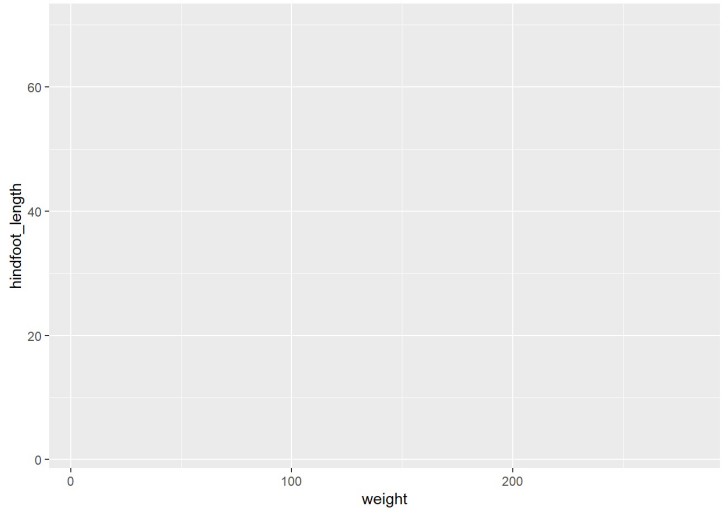
m + **geom_raster(**aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

m + **geom_tile(**aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

Visualizing data in R

- Let's Try it on the **surveys** data

```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length))
```



Visualizing data in R

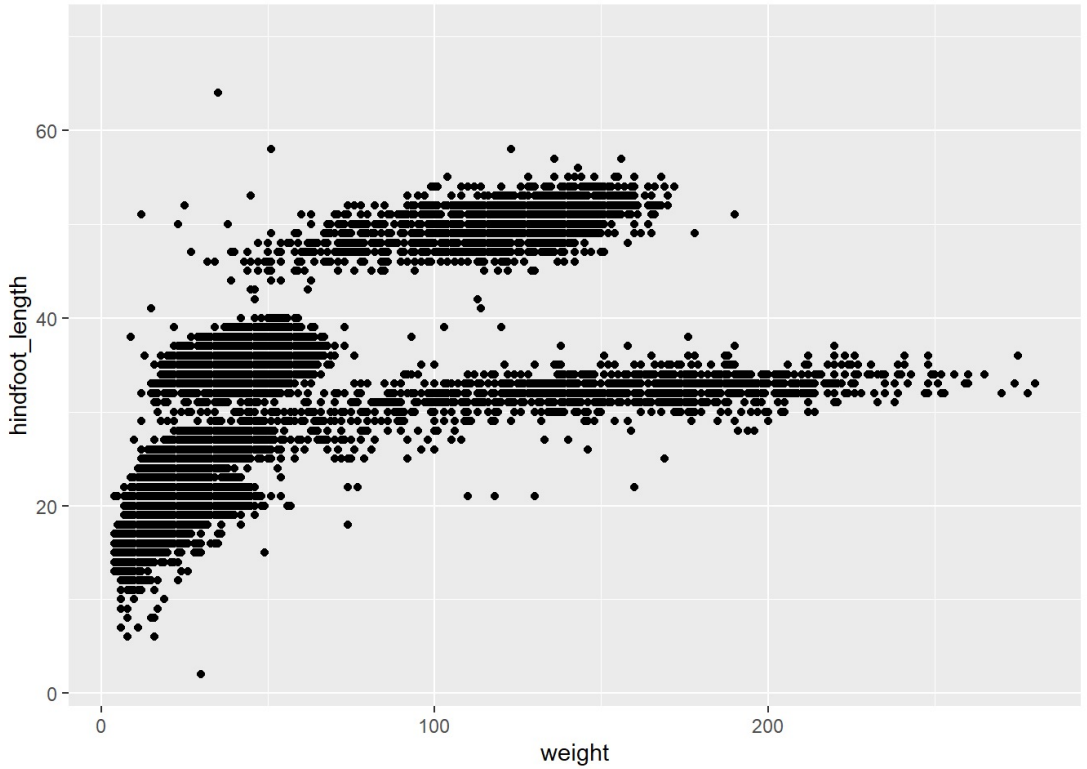
- Let's Try it on the **surveys** data

```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length))
```

- Add Layers

```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length)) + geom_point()
```

Visualizing data in R



- Add Layers

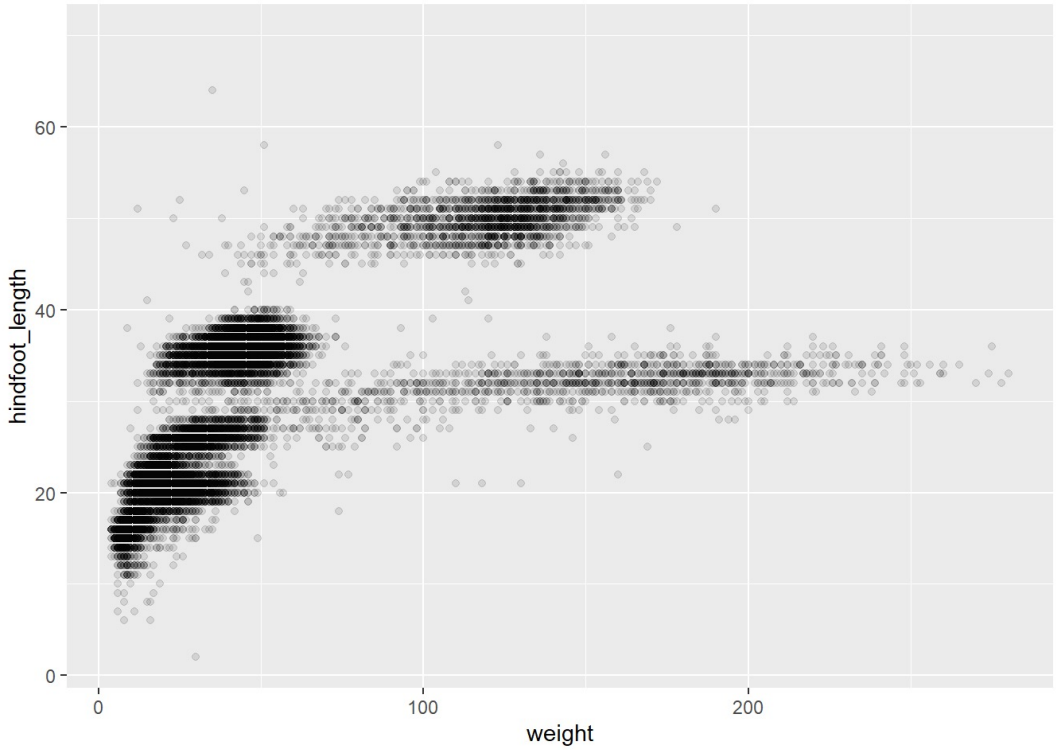
```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length)) + geom_point()
```

Visualizing data in R

- Customize: Add Transparency

```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length)) + geom_point (alpha = 0.1)
```

Visualizing data in R



➢ Customize: Add Transparency

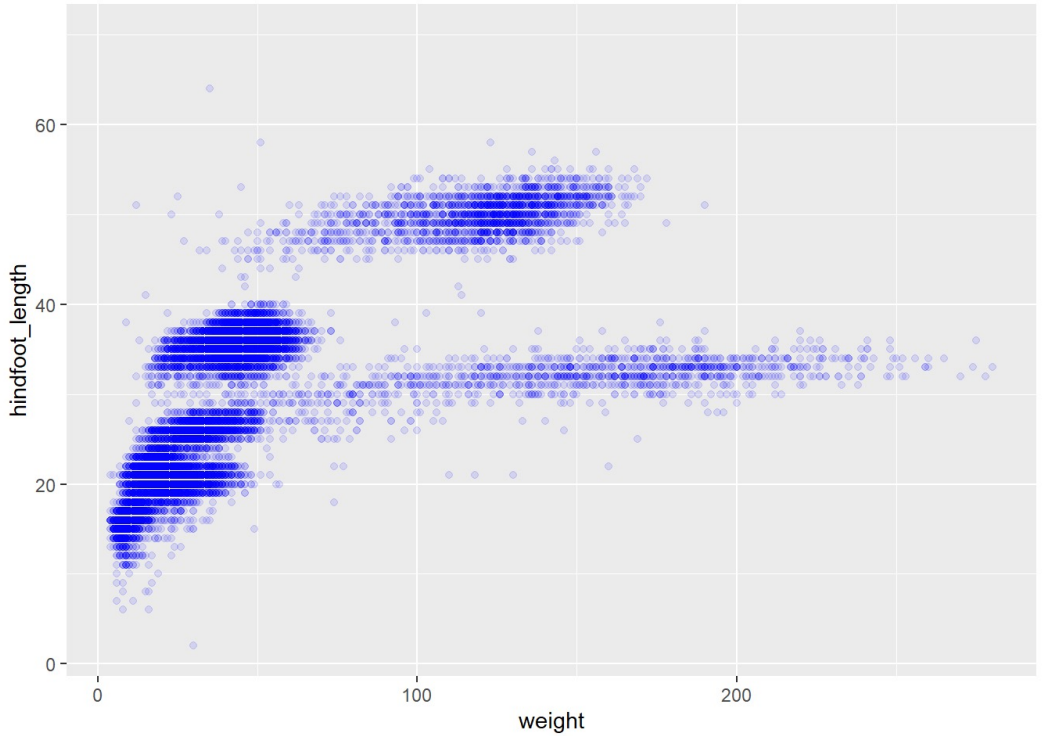
```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length)) + geom_point (alpha = 0.1)
```

Visualizing data in R

- Add color

```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length)) + geom_point (alpha = 0.1, color = 'blue')
```

Visualizing data in R



- Add color

```
ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length)) + geom_point (alpha = 0.1, color = 'blue')
```

Visualizing data in R

Let's Try other types of plots!

- *Save* the ggplot into a variable 'surveys_plot'

```
surveys_plot <- ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length))
```

- *Draw a scatter plot*

```
surveys_plot + geom_point()
```

- Now draw a geom_smooth plot

```
surveys_plot + geom_smooth()
```

Visualizing data in R

Try other types of plots!

- *Save* the ggplot into a variable 'surveys_plot'

```
surveys_plot <- ggplot (data = surveys , mapping = aes ( x = weight,  y = hindfoot_length))
```

- *Draw a scatter plot*

```
surveys_plot + geom_point()
```

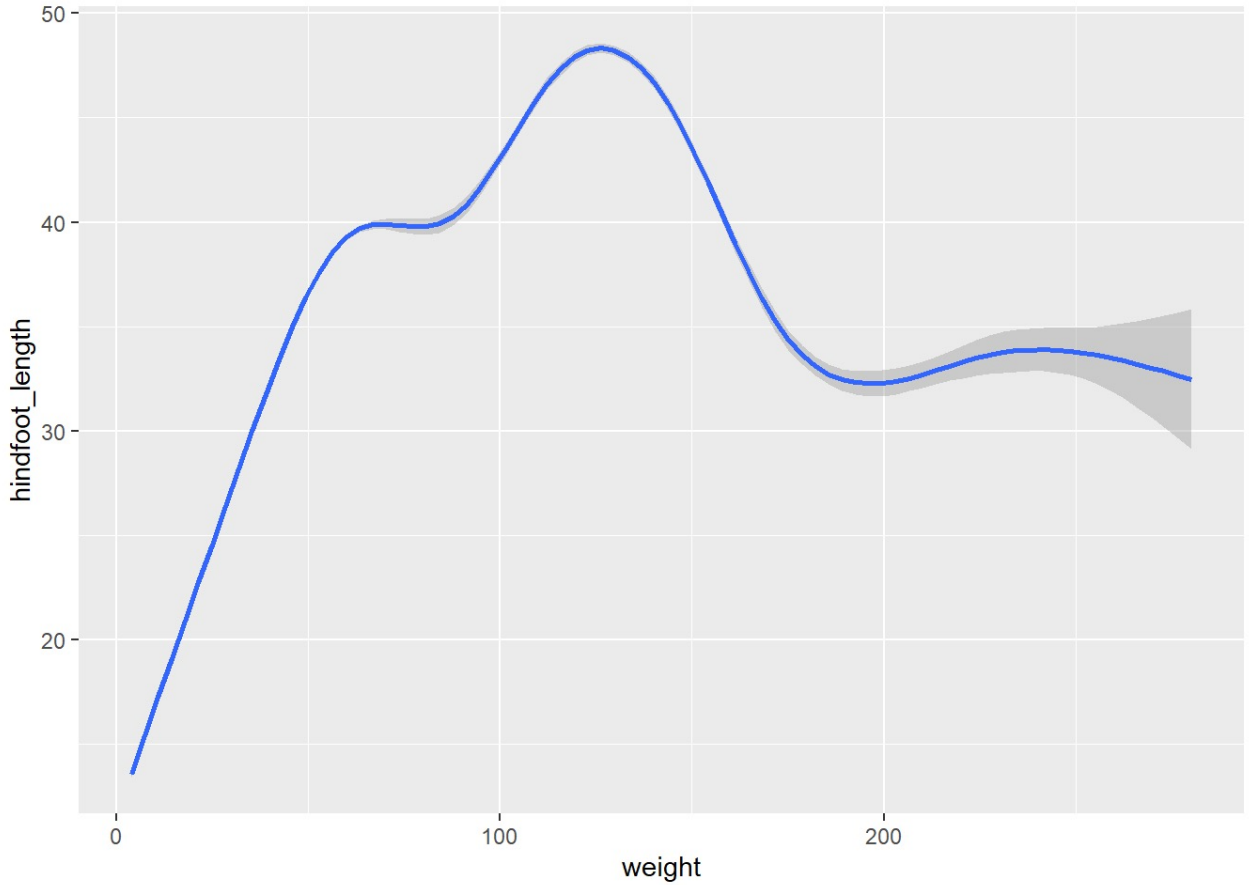
- Now draw a geom_smooth plot

```
surveys_plot +
        geom_smooth()
```

Visualizing data in R

A geom-smooth plot

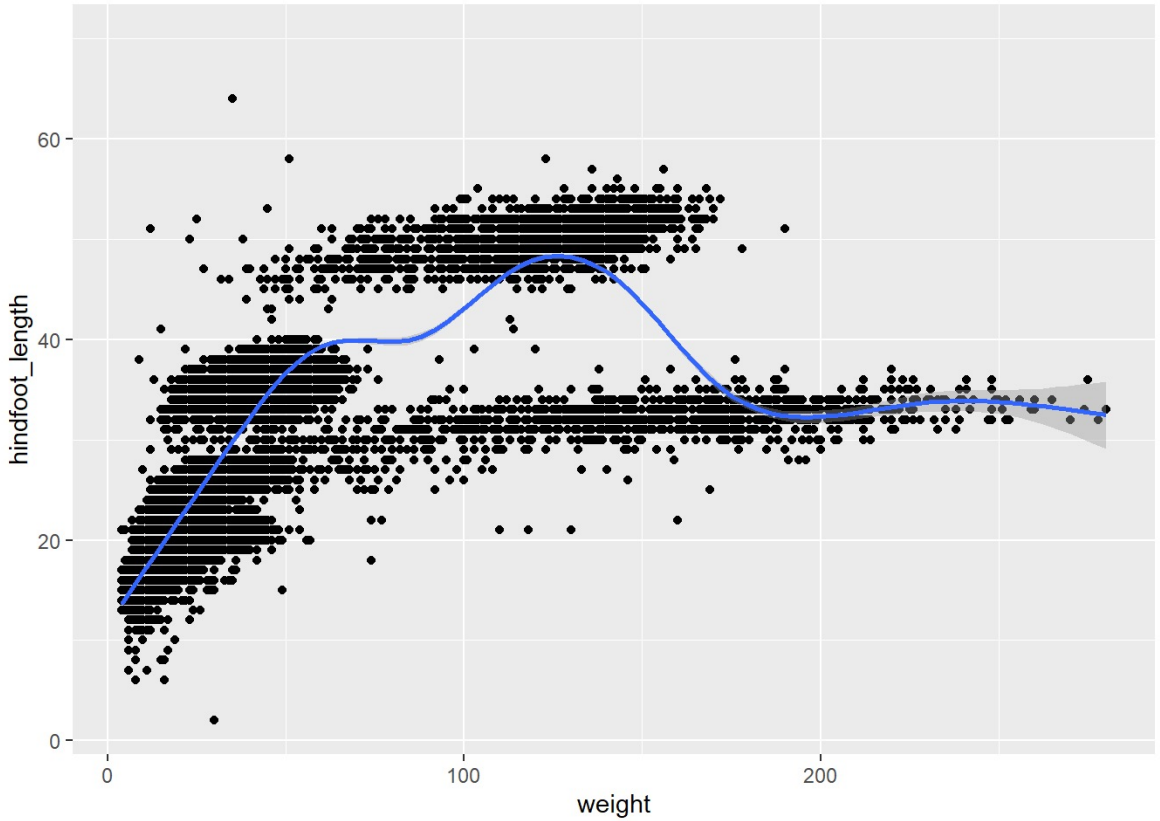
```
surveys_plot +
        geom_smooth()
```



Visualizing data in R

Multiple Plots as Layers:

```
surveys_plot
+ geom_point()
+ geom_smooth()
```



What happens if you change the order of layers added!?

Visualizing data in R

Syntax is Important!

+ sign MUST come at the END of line of previous layer

```
# This is the correct syntax for adding layers
surveys_plot +
 geom_point()

# This will not add the new layer and will return an error message
surveys_plot
 + geom_point()
```

Mappings for a given geom can be independent of global mappings defined in the ggplot() function

Exercise : Data Visualisation

- Scatter plots can be useful exploratory tools for **small** datasets.
- For data sets with **large** numbers of observations, such as the surveys data set, **overplotting** of points can be a limitation of scatter plots.
- We have already seen how we can visualise data better when we have overplotting with the **geom_smooth** plot.
- Another way for handling overplotting is to display the **density** of the data through **contours**.

- **As this challenge's task, do the following!**
  - Create a script called **plot_density2d.R** which :
    1. Loads the file **data/portal_data_joined.csv** into the variable **surveys**.
    2. Uses this dataset to plot **weight** on x-axis and **hindfoot_length** on y-axis in a geom_density2d plot

Saving a plot to a file

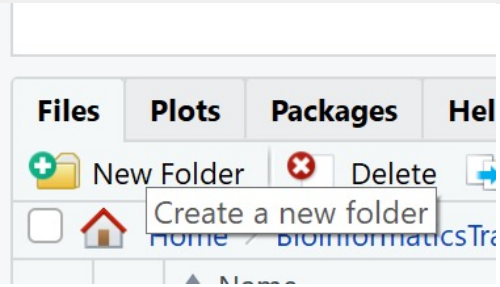
```
#save plot that you would like to save into a variable
out_plot <- surveys_plot + geom_density2d()
#save plot to file
ggsave(filename="img_output/plot_weight_hindfoot_density2d.png", plot=out_plot)
```

[Not necessary to save plot first; **ggsave** will auto save the LAST plot plotted by R]

```
#save plot to file
ggsave(filename="img_output/plot_weight_hindfoot_density2d.png")
```

- Make the folder 'img_output' beforehand!

Create from within RStudio



DATA TRANSFORMATION