

Introduction:

R & RStudio

Bianca Peterson (PhD)

Conquest Analytics and Training

07 May 2021

Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

What is R and RStudio?

- R —> both the programming language and the software that interprets the scripts
- RStudio —> a front end (interface) to R, which makes using R a lot nicer

Why learn R?

- R doesn't involve lots of pointing and clicking - easy to redo analysis if you collected more data
- R code is great for reproducibility - obtain same results from same dataset using same analysis
- R is extensible and interdisciplinary - 17,000+ packages to extend its capabilities, and statistical approaches from many scientific disciplines can be combined
- R works on data of all shapes and sizes - it is designed for data analysis
- R produces high-quality graphics - can adjust any aspect of your graph
- R has a large community - mailing lists and websites (Stack Overflow)
- R is open-source and cross-platform - Anyone can inspect the source code
- Less chance for mistakes

Objectives

- What is R and RStudio
 - RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Layout of RStudio

The screenshot shows the RStudio interface with the following components and callouts:

- Source Editor (Top Left):** Contains R code for a 'Yummy pasta recipe'.

```
1  
2 - # Yummy pasta recipe -----  
3  
4 # get out the equipment we need (load the packages)  
5 library(saucepan)  
6 library(colander)  
7  
8 # get the ingredients out on to the counter (load data)  
9 pasta <- read_csv("pasta.csv")  
10 cheese <- read_csv("cheese.csv")  
11 sauce <- read_csv("yummy_sauce.csv")  
12 water <- read_csv("tap_water.csv")  
13  
14 # cook pasta then drain it and then add the cheese and the sauce  
15 cooked_pasta <- saucepan(pasta + water)  
16 drained_pasta <- colander(cooked_pasta)  
17 yummy_pasta <- c(drained_pasta, cheese, sauce)  
18
```

Scripts are recipes (records) of how to do things.
Write and save your recipes here so that R knows what to cook.
- Console (Bottom Left):** Shows the execution of the code from the source editor.

```
> # get out the equipment we need (load the packages)  
library(saucepan)  
library(colander)  
  
# get the ingredients  
pasta <- read_csv("pasta.csv")  
cheese <- read_csv("cheese.csv")  
sauce <- read_csv("yummy_sauce.csv")  
water <- read_csv("tap_water.csv")  
  
# cook pasta then  
cooked_pasta <- saucepan(pasta + water)  
drained_pasta <- colander(cooked_pasta)  
yummy_pasta <- c(drained_pasta, cheese, sauce)
```

The console is where the cooking happens - send recipes here (run code) to cook them.
You can cook here without using a recipe, but you'll struggle to remember exactly how to recreate the dish in the future.
It's better to use a recipe!
- Environment (Top Right):** Shows the global environment.

The environment is like the kitchen counter - you can put ingredients (data) and finished dishes (model outputs) here to use while you cook.
- Files (Bottom Right):** Shows the file explorer.

Files are like ingredients in your cupboard - you need to get them out on to the kitchen counter (the environment) to use them.
The files that you need can be specified in the recipe so you know exactly what you need to get out.
- Packages (Bottom Right):** Shows the installed and available packages.

Packages are like tools - when you need to use a saucepan, you go out and buy one that someone has already designed and made [install.packages()].
Each time you want to use that pan, you just take it out of the cupboard [library()].

Objectives

- What is R and RStudio
- RStudio layout
 - Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

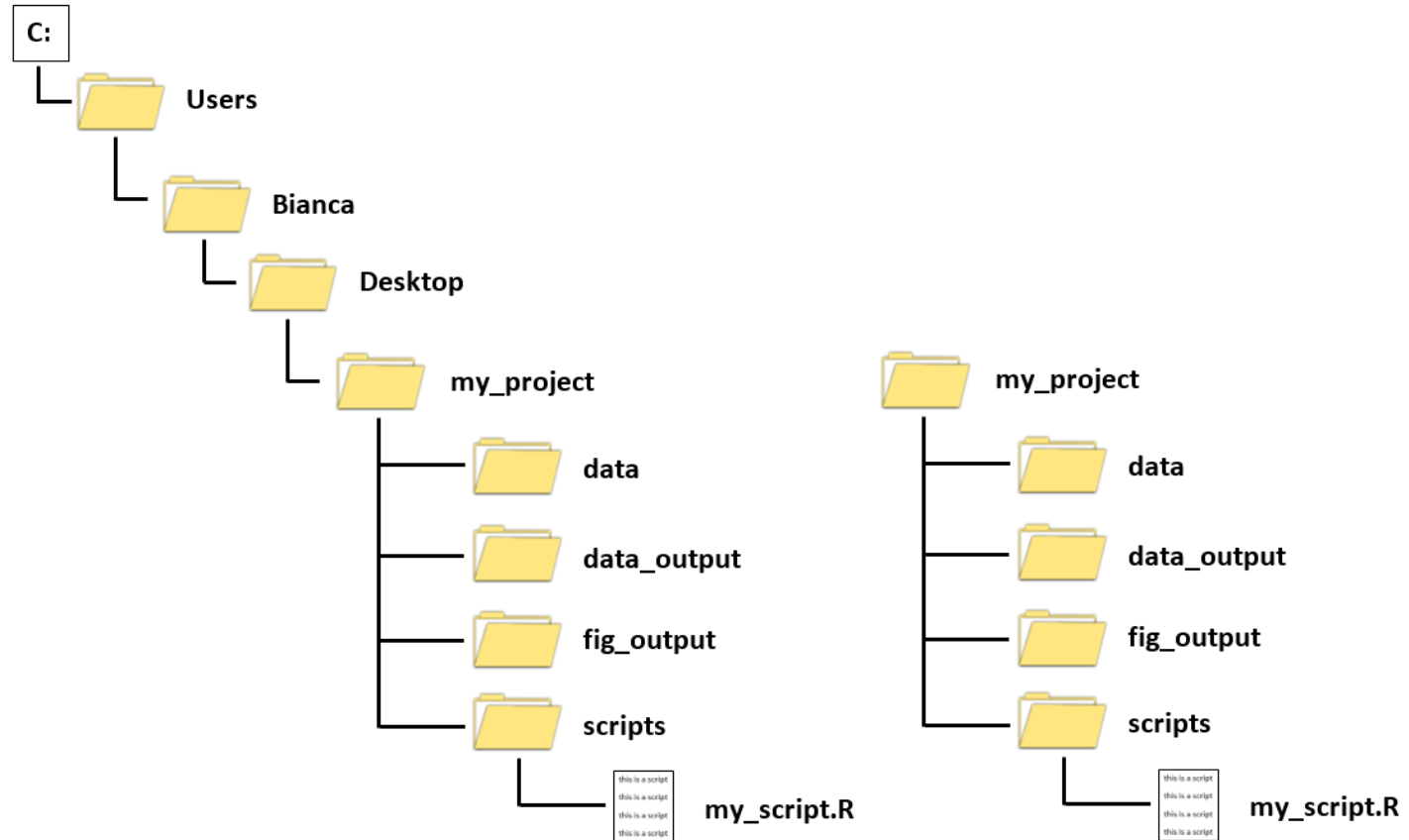
Create a new project

- It is good practice to keep a set of related data, analyses and text in single folder called the "working directory"
- You can easily share it with others without worrying about whether or not underlying scripts will still work
- Only ever use relative paths and not absolute paths

Absolute vs relative paths

ABSOLUTE PATH

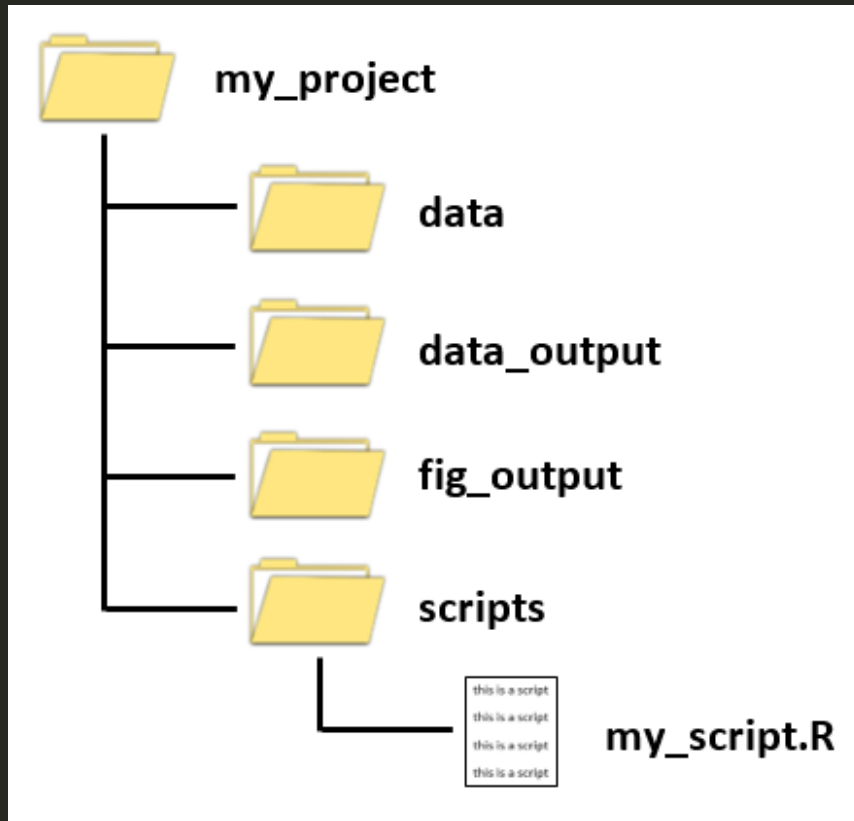
RELATIVE PATH



Time for...



Your working directory should look like this



- **Working directory** = where R looks for files that you ask it to load, and where it will put any files that you ask it to save

Interacting with R

- We write code/instructions = commands
- The computer executes/runs those commands
- Two ways to interact with R:
 - Type in the console and press Enter to execute -> forgotten when closed
 - Type in script editor and press Ctrl + Enter -> save code and workflow
- Prompt > is displayed when R is ready to accept commands
- Receives commands (typing, copy-pasting or sent from script editor) -> execute -> show results -> new prompt
- If R waits for more data -> will show a + prompt (continuation symbol), which means you haven't finished entering a complete command
- Either enter the missing letter/value/symbol or click inside the console and press Esc

Time for...



Seeking help (1)

- Use built-in RStudio help interface (bottom-right panel)
- Search function documentation with ? and ??
 - ?function
 - ??task
- help(function)
- Automatic code completion
 - Reminder of a function's name or arguments
 - Avoid spelling errors
- Package vignettes and cheat sheets
 - Instructions for how to use the package
 - browseVignettes()
 - Cheat sheets, keyboard shortcuts, and more, available in the Help menu (at the top of the RStudio window)

Seeking help (2)

- Finding more functions and packages
 - Help only searches installed packages
 - To search all available packages, use rdocumentation.org website
 - Google "R "
 - Many packages also have websites with additional help, tutorials, etc.
- Dealing with error messages
 - Errors are common when programming
 - The problem is often a typo
 - Fixing errors is part of any programmers daily work
 - Watch for red x's next to your code

Seeking help (3)

- Where to ask for help:
 - Friendly colleagues
 - Stack Overflow: <http://stackoverflow.com/questions/tagged/r>
 - R-help mailing list: <https://stat.ethz.ch/mailman/listinfo/r-help>
 - Use correct vocabulary
 - Check for package-specific mailing lists
 - List of topic-specific mailing lists: <http://www.r-project.org/mail.html>

Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
 - Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Time for...



Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
 - Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Structure of R expressions

```
object <- function(argument1, argument2, ...)
```

- object = can be any word you like, but avoid dots (and no spaces!)
 - also watch out for existing function names
- <- = assignment operator (Shortcut: Alt -)
- function = name of the function followed directly by ()
 - Example: `x <- round(3.14159)`
- arguments = specified within the () of the function, separated by commas
 - Example: `x <- round(3.14159, digits = 2)`

Time for...



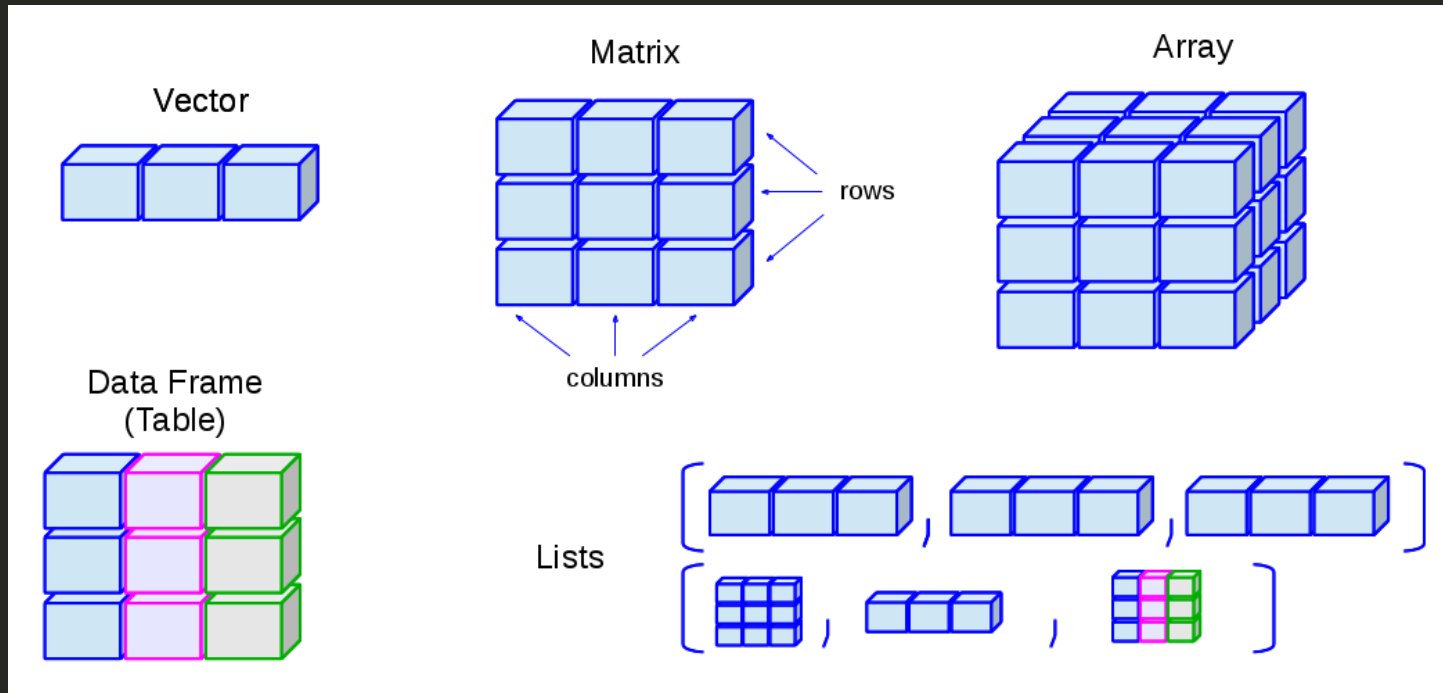
Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
 - **Vectors**
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Data structures vs Data types

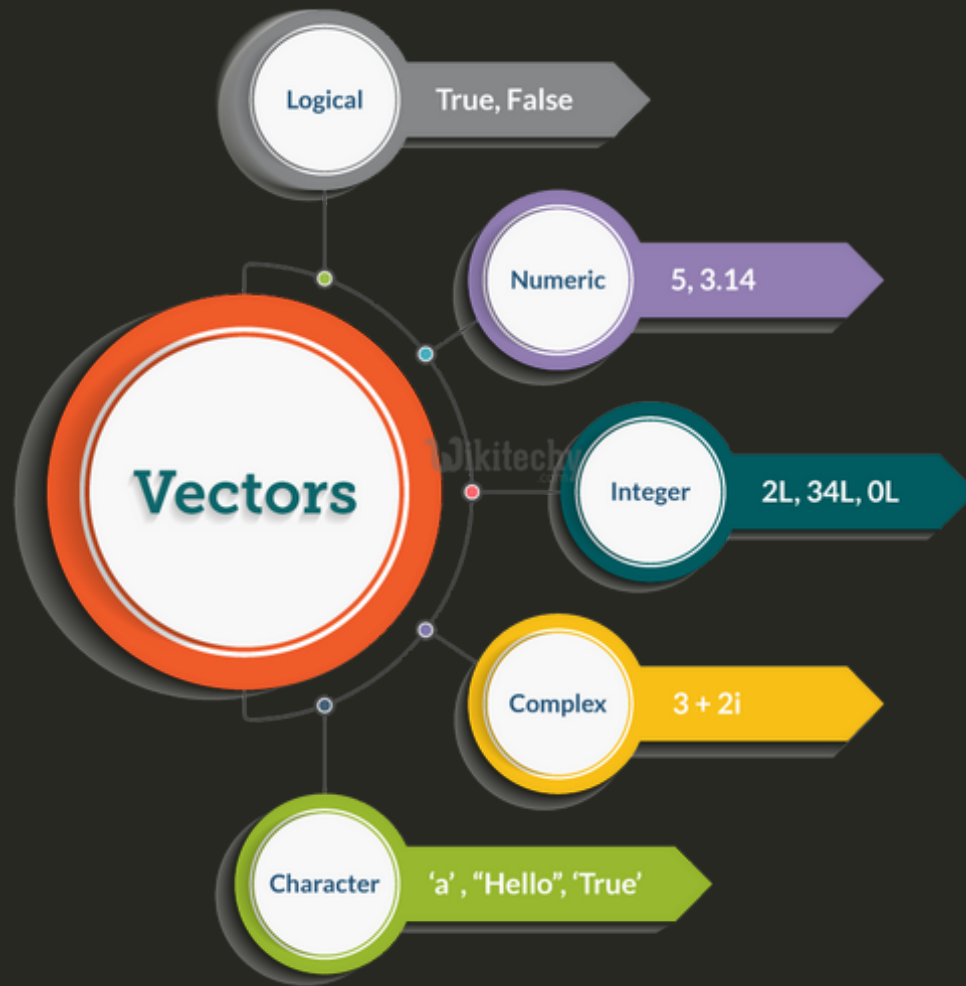
- Data structures:
 - Vector
 - Matrix
 - Array
 - Data Frame
 - List
 - Factor
- Data types:
 - Numeric
 - Integer
 - Complex
 - Logical
 - Character

Data structures



Credit: Maite Ceballos (IFCA) & Nicolas Cardiel (UCM) <http://venus.ifca.unican.es/Rintro/dataStruct.html>

Data types



Credit: Venkatesan Prabu .J (Wikitechy) <https://www.wikitechy.com/tutorials/r-programming/r-datatypes-vectors>

Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
 - Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Time for...



Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
 - Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

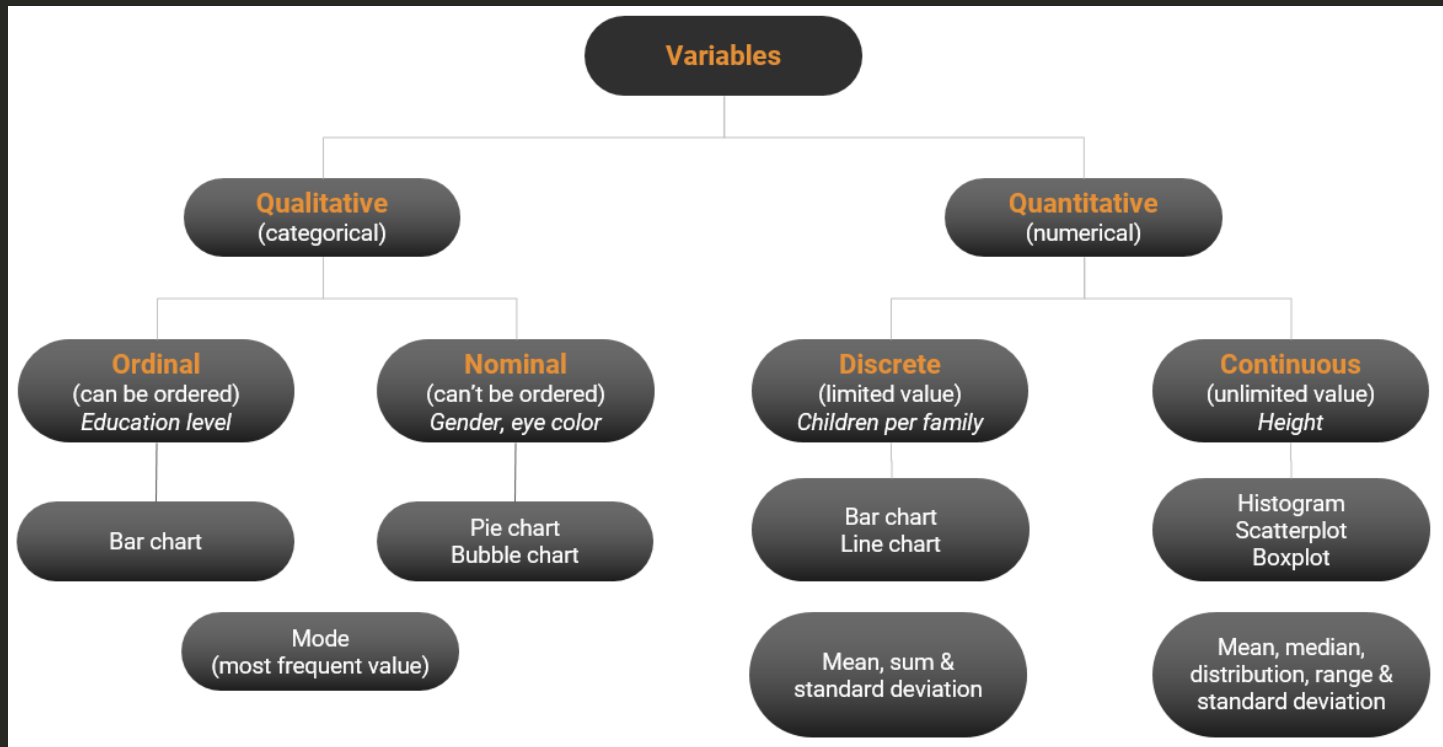
Importing data with `read_csv()`

- Sometimes there are a few lines of metadata at the top of the file:
 - You can use skip the first "n" lines: `read_csv("data.csv", skip = 5)`
 - Or drop all lines that start with a certain character, e.g. "#": `read_csv("data.csv", comment = "#")`
- The data might not have column names:
 - You can use `col_names = FALSE` to tell `read_csv()` not to treat the first row as headings, and instead label them sequentially from X1 to Xn: `read_csv("data.csv", col_names = FALSE)`
 - Alternatively, pass `header` a character vector with column names: `read_csv("data.csv", col_names = c("name", "surname", "age", "weight"))`
- Specify the value(s) that are used to represent missing values in your file: `read_csv("data.csv", na = ".")`
- Trim whitespace: `read_csv("data.csv", trim_ws = TRUE)`

The survey data

- The data consists of a survey of animals where the following variables were recorded:
 - record_id, month, day, year, plot_id, species_id, sex, hindfoot_length, weight, genus, species, taxon, plot_type
 - a description of each variable is available [here](#)
- The data set is stored as a comma separated value (CSV) file
- Each row contains information about a single animal
- Each column represents a single variable

Data types



Time for...



Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
 - Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Factors

```
x <- factor(c("wallet", "gum", "lotion", "sanitiser", "lip-gloss"))  
levels(x)
```

```
[1] "gum"      "lip-gloss" "lotion"    "sanitiser" "wallet"
```



Time for...



Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- **Formatting dates**
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Formatting dates

| Order of date elements | Parse function |
|-----------------------------------|----------------|
| year month day | ymd() |
| year day month | ydm() |
| month day year | mdy() |
| hour minute | hm() |
| hour minute second | hms() |
| year month day hour minute second | ymd_hms() |

Time for...



Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
 - Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

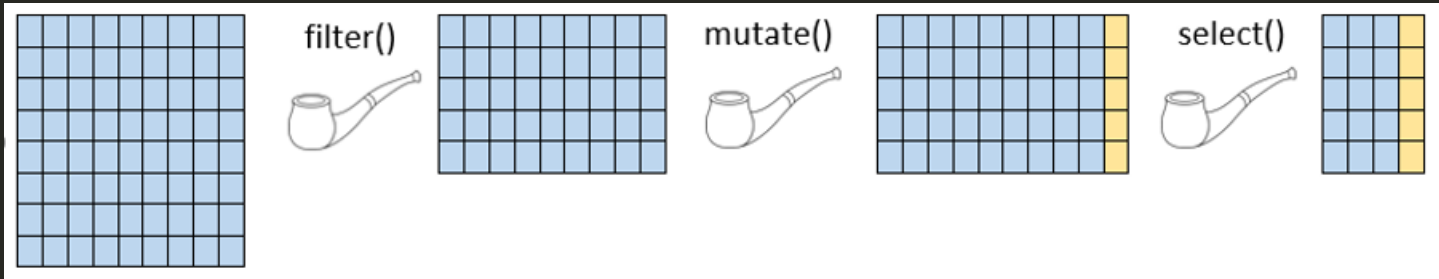
Data manipulation using dplyr and tidyr

- dplyr package makes tabular data manipulation easier
- tidyr package enables you to convert between different data formats for plotting and analysis
- New functions/analyses are developed and made available via packages
- Packages = additional functions that let you do more stuff
- Install package once - `install.packages()`
- Load package with every R session when you need it
- tidyverse = umbrella package that installs several packages

Time for...



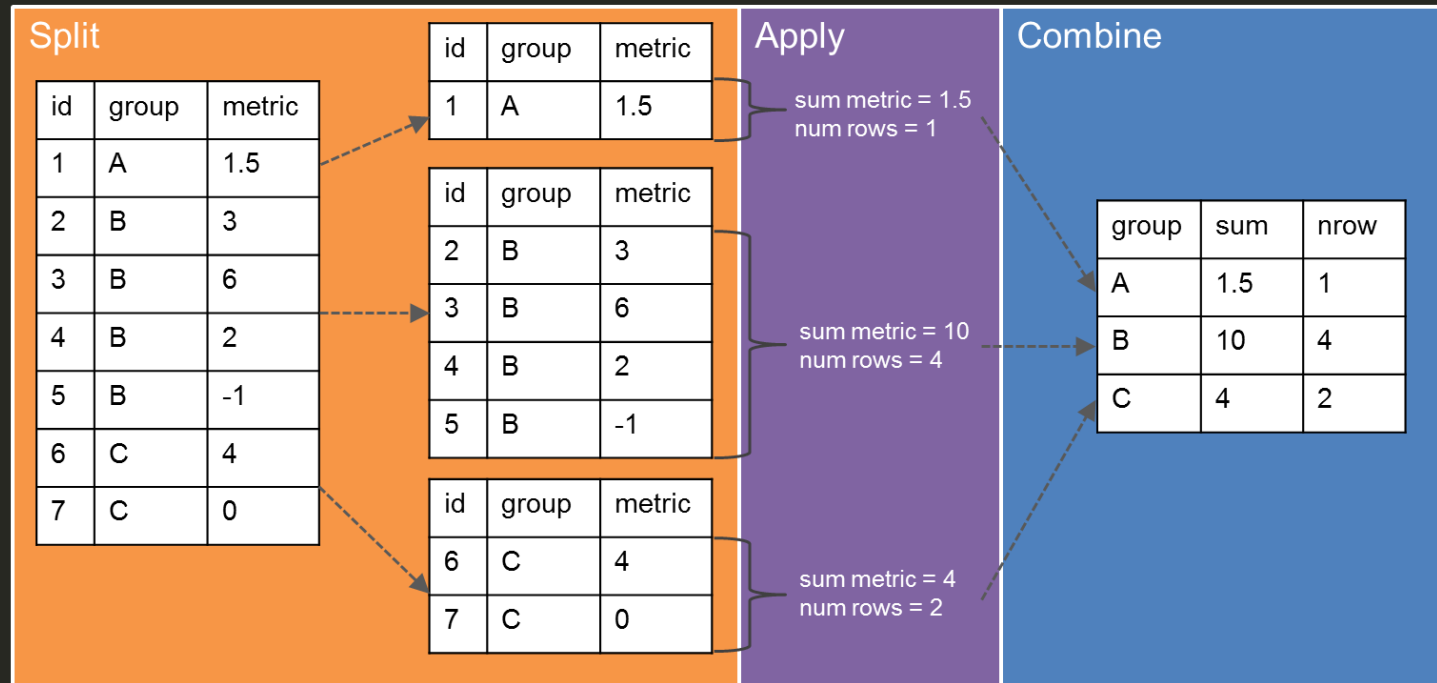
Pipes



Time for...



Split-apply-combine analysis



It collapses each group into a single-row summary of that group

Credit: Modified from Ranae Dietzel & Andee Kaplan, https://agron590-isu.github.io/materials/week_08/dplyr/slides_dplyr.html

Time for...



Reshaping your data: spread / pivot_wider

Long

| | key | value |
|---------|-------------|-------------|
| plot_id | genus | mean_weight |
| 1 | Baiomys | 7.00 |
| 2 | Baiomys | 6.00 |
| 3 | Baiomys | 8.61 |
| 1 | Chaetodipus | 22.20 |
| 2 | Chaetodipus | 25.11 |
| 3 | Chaetodipus | 24.64 |
| 1 | Dipodomys | 60.23 |
| 2 | Dipodomys | 55.68 |
| 3 | Dipodomys | 52.05 |

Wide

| plot_id | Baiomys | Chaetodipus | Dipodomys |
|---------|---------|-------------|-----------|
| 1 | 7.00 | 22.20 | 60.23 |
| 2 | 6.00 | 25.11 | 55.68 |
| 3 | 8.61 | 24.64 | 52.05 |

data.frame

column with new
variable names

column of values
for new
variables

```
surveys_gw %>% spread(key = genus, value = mean_weight)
```

Time for...



Reshaping your data: gather / pivot_longer

Long

| plot_id | genus | mean_weight |
|---------|-------------|-------------|
| 1 | Baiomys | 7.00 |
| 2 | Baiomys | 6.00 |
| 3 | Baiomys | 8.61 |
| 1 | Chaetodipus | 22.20 |
| 2 | Chaetodipus | 25.11 |
| 3 | Chaetodipus | 24.64 |
| 1 | Dipodomys | 60.23 |
| 2 | Dipodomys | 55.68 |
| 3 | Dipodomys | 52.05 |

Wide

| plot_id | Baiomys | Chaetodipus | Dipodomys |
|---------|---------|-------------|-----------|
| 1 | 7.00 | 22.20 | 60.23 |
| 2 | 6.00 | 25.11 | 55.68 |
| 3 | 8.61 | 24.64 | 52.05 |

`data.frame`

variable whose
values are
column names

variable whose
values are
spread over
columns

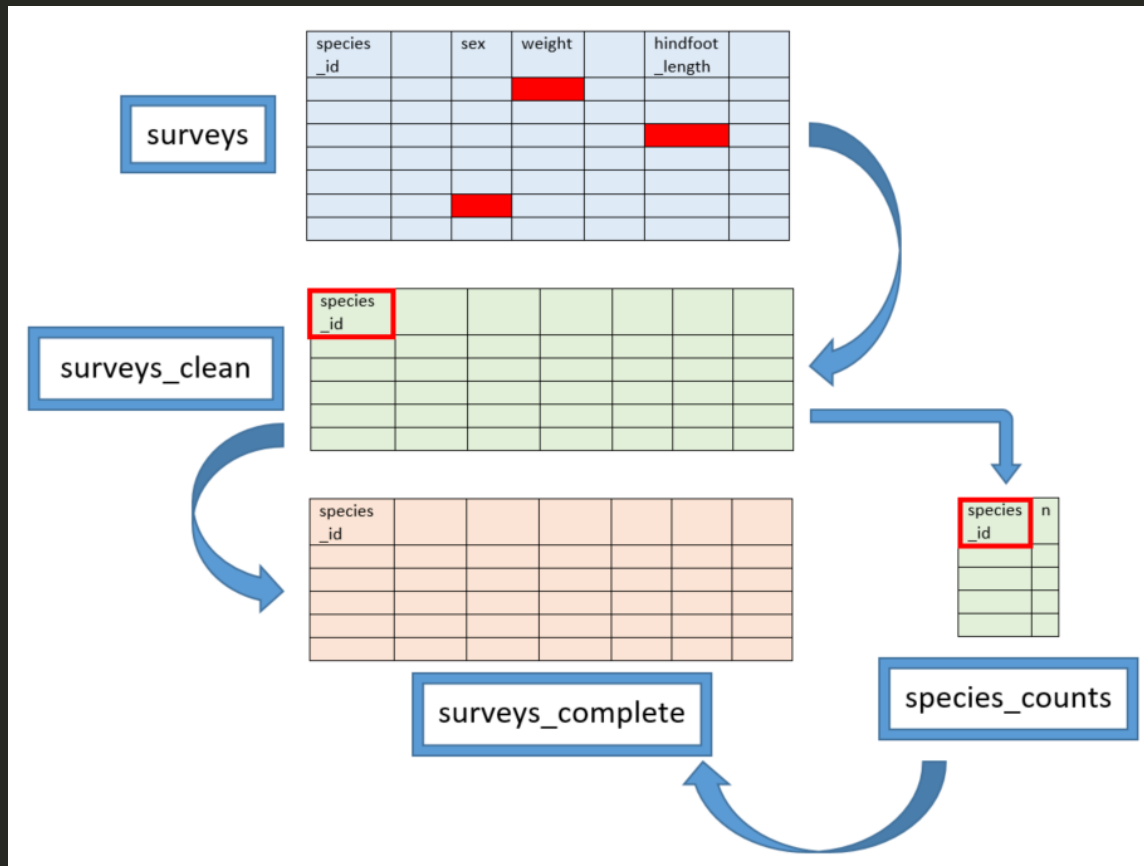
don't use this
values of this
variable

```
surveys_spread %>% gather(key = genus, value = mean_weight, -plot_id)
```

Time for...



Cleaning and subsetting data



Time for...



Objectives

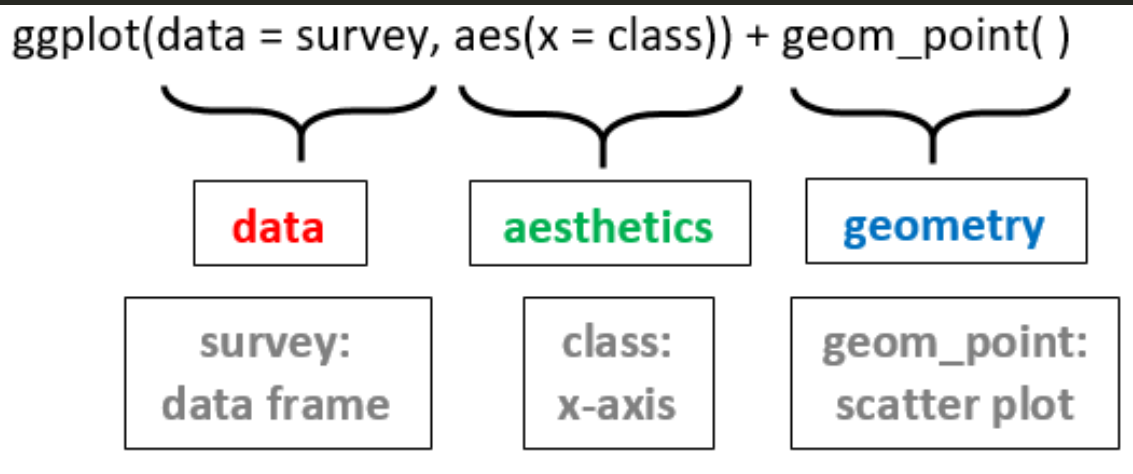
- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
 - Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Data visualisation: grammar of graphics

Grammar of graphics = a framework which follows a layered approach to describe and construct visualisations or graphics in a structured manner

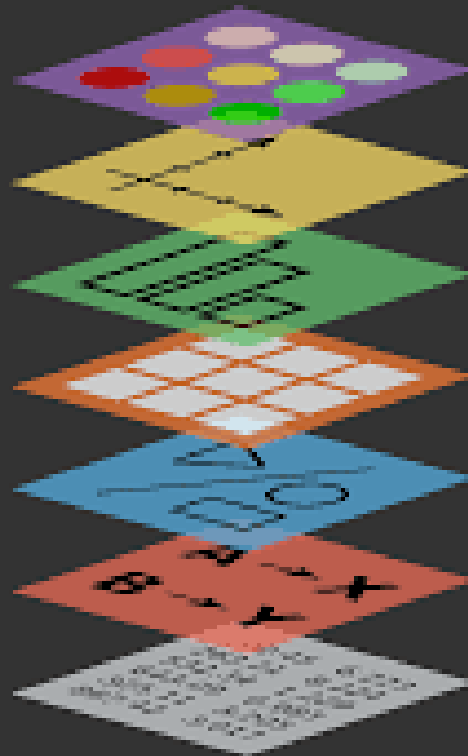
$$\text{plot} = \text{data} + \text{aesthetics} + \text{geometry}$$

- **<data>** = refers to a data frame (data set)
- **<aesthetics>** = indicates x and y variables & how data should be displayed in the plot (e.g. color, size, shape)
- **<geometry>** = refers to the type of graphics (bar chart, histogram, box plot, line plot, density plot, etc.)



Data visualisation: grammar of graphics

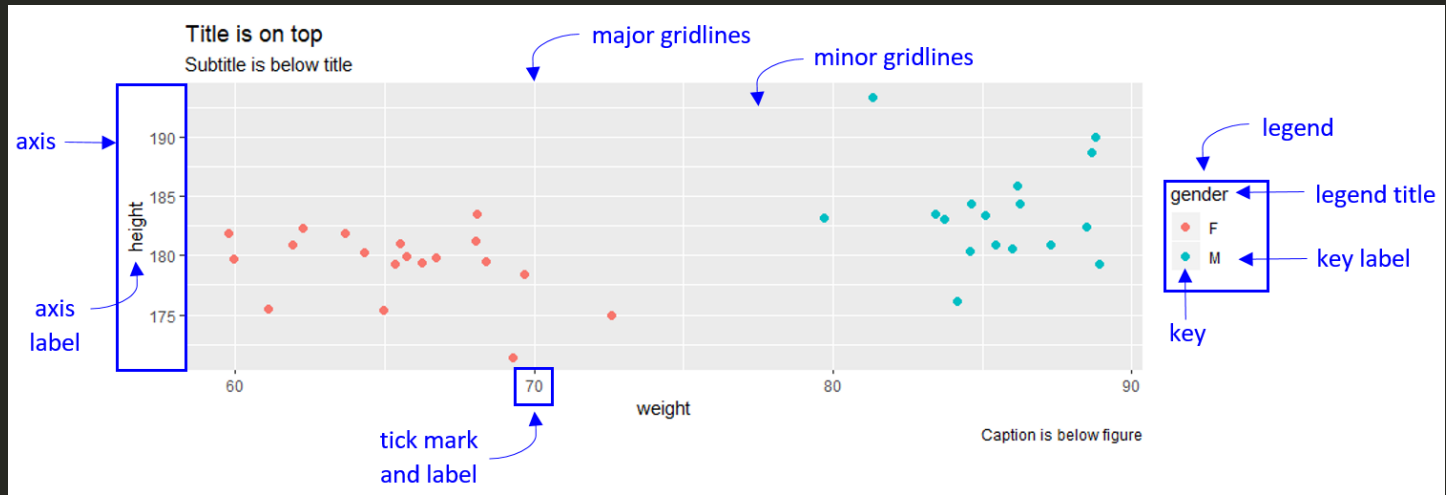
Theme
Coordinates
Statistics
Facets
Geometries
Aesthetics
Data



Time for...



Plot components



Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
- Summary

Some other useful functions

```
# Import data by browsing for file  
my_data <- read.csv(file.choose(new = TRUE))  
  
# Import Excel file (per sheet)  
library(readxl)  
  
read_excel("data/my_data.xlsx", sheet = "Sheet1")
```

Some other useful functions

```
# Update a package  
tidyverse_update()  
  
# Helper function to use within filter():  
between(weight, 30, 50)  
  
# Helper functions to use within select():  
ends_with("xyz")  
contains("cde")  
select(df, col5, col6, everything())  
  
# Create and keep only the new column:  
transmute(df, new_column = old_column*2)  
  
# Global substitutions (i.e. find and replace):  
gsub(pattern, replacement, data)
```

Some other useful functions

```
# Remove an object from the environment  
rm(object_name)
```

```
# Detach (i.e. "unload") a package  
detach(tidyverse)
```

```
# Pick the top one (from arranged data)  
dataframe %>%  
  filter(!is.na(var1)) %>%  
  arrange(var1, var2, var3) %>%  
  slice(1)
```

```
dataframe %>%  
  filter(!is.na(var1)) %>%  
  arrange(var1, var2, var3) %>%  
  top_n(1)
```

Some other useful functions

```
# Divide data into categories:
mutate(category = case_when(weight <= 100 ~ "small",
                             weight > 100 & weight < 200 ~ "medium",
                             weight > 200 ~ "large"))

# Rename specific column headers:
rename(df, new_name = old_name)

# Top 10 heaviest animals:
top_n(df, 10, weight)

# Separate (i.e. split) one column into multiple:
separate(column, into = c("col1", "col2"))

# Bind tables on columns or rows
cbind(df1, df2, ...)
rbind(df1, df2, ...)
```

Some other useful functions

```
# Mutating joins: combine variables from two data.frames
inner_join(x, y, by = "key") #keeps all observations present in both x and y.
left_join(x, y, by = "key")  #keeps all observations in x.
right_join(x, y, by = "key") #keeps all observations in y.
full_join(x, y, by = "key")  #keeps all observations in x and y.

# Filtering joins: keep cases from the left-hand data.frame
semi_join(x, y) #keeps all observations in x that have a match in y.
anti_join(x, y) #drops all observations in x that have a match in y.

# Nesting joins: create a list column of data.frames
nest_join(x, y, by = "key") #keeps all rows and all columns from x and adds a list column of tibbles

# Set operations: compare values of every variable
intersect(x, y): #return only observations in both x and y.
union(x, y):     #return unique observations in x and y.
setdiff(x, y):   #return observations in x that are not in y.
```

Plot components

Combine Data Sets

| a | | | b | | |
|----|----|---|----|----|---|
| x1 | x2 | | x1 | x3 | |
| A | 1 | + | A | T | = |
| B | 2 | | B | F | |
| C | 3 | | D | T | |

Mutating Joins

| x1 | x2 | x3 |
|----|----|----|
| A | 1 | T |
| B | 2 | F |
| C | 3 | NA |

dplyr::left_join(a, b, by = "x1")

Join matching rows from b to a.

| x1 | x3 | x2 |
|----|----|----|
| A | T | 1 |
| B | F | 2 |
| D | T | NA |

dplyr::right_join(a, b, by = "x1")

Join matching rows from a to b.

| x1 | x2 | x3 |
|----|----|----|
| A | 1 | T |
| B | 2 | F |

dplyr::inner_join(a, b, by = "x1")

Join data. Retain only rows in both sets.

| x1 | x2 | x3 |
|----|----|----|
| A | 1 | T |
| B | 2 | F |
| C | 3 | NA |
| D | NA | T |

dplyr::full_join(a, b, by = "x1")

Join data. Retain all values, all rows.

Filtering Joins

| x1 | x2 |
|----|----|
| A | 1 |
| B | 2 |

dplyr::semi_join(a, b, by = "x1")

All rows in a that have a match in b.

| x1 | x2 |
|----|----|
| C | 3 |

dplyr::anti_join(a, b, by = "x1")

All rows in a that do not have a match in b.

IF statements

```
if (condition) {  
    # code executed when condition is TRUE  
} else {  
    # code executed when condition is FALSE  
}
```

```
if (this) {  
    # do this  
} else if (that) {  
    # do that  
} else {  
    # do something else  
}
```

IF statements

```
temp <- 15

if (temp <= 0) {
  "freezing"
} else if (temp <= 10) {
  "cold"
} else if (temp <= 20) {
  "cool"
} else if (temp <= 30) {
  "warm"
} else {
  "hot"
}
```

Writing your own function

- Functions allow you to automate common tasks instead of copy-and-pasting
- Writing a function has 3 main advantages over using copy-and-paste:
 - You can give a function an evocative name that makes your code easier to understand.
 - As requirements change, you only need to update code in one place, instead of many.
 - You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

Writing your own function

```
function_name <- function(x) {  
  y <- read.csv(x)  
  
  y %>%  
    count(variable)  
}  
  
# Example  
  
get_species <- function(x) {  
  y <- read.csv(x)  
  
  y %>%  
    count(species)  
}
```

Useful shortcuts

- Ctrl + Shift + N = New script
- Alt + "-" = Assignment operator <—
- Ctrl + Enter = Run selected command(s)
- Ctrl + Shift + S = Run the whole script
- Ctrl + Shift + R = Create script headers
- Alt + Shift + K = Keyboard shortcut quick reference
- F1 (when function is highlighted in pop-up tooltip) = activates help window
- Ctrl + Shift + M = Insert pipe character %>%

Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
 - **Additional resources**
- Summary

Additional Resources

- Interactive learning
 - `install.packages("swirl")`
 - `library(swirl)`
 - Follow the on-screen instructions
- RStudio Resources
 - R Graph Gallery
 - RStudio Primers
 - Webinars & Videos
 - Cheat Sheets
 - Educational Material for Learners or Teachers
 - A Gentle Introduction to Tidy Statistics in R
- Books
- Introduction to R
- R FAQ
- Follow #rstats on twitter and ask questions there (also check out @RLangTips)
- MANY free online tutorials for learning R - just Google!

Objectives

- What is R and RStudio
- RStudio layout
- Creating a reproducible project
- Object assignment
- Functions and arguments
- Vectors
- Missing data
- Data frames
- Factors
- Formatting dates
- Data manipulation
- Data visualisation with ggplot2
 - Scatter plot
 - Boxplot
 - Line graph
 - Bar graph
 - Interactive plots
- Additional functions
- Additional resources
 - Summary

Summary

- RStudio projects give you a solid workflow that will serve you well in the future:
 - Create an RStudio project for each data analysis project
 - Create a file structure that is logical and reusable - helps to automate future work
 - Keep raw data files there
 - Keep scripts there
 - Save your outputs (plots and cleaned data) there
 - Only ever use relative paths, not absolute paths

Summary

