[PSY202A] Statistical Modeling in Psychological Sciences

Ihnwhi Heo, M.Sc.

Fall 2024

Contents

1 Introduction **5**

2 Introduction to R: Part 1 **7**
2.1 Word of wisdom 8
2.2 How R you? 8
2.3 Interacting with R 11
2.4 Computation 12
2.5 Types of objects 13
2.6 Workspace 18
2.7 Working directory 19
2.8 Importing external data 19
2.9 Exporting 20
2.10 Final exercise 20
2.11 R Markdown 21

3 Introduction to R: Part 2 **23**
3.1 What will we do? 23
3.2 Packages 24
3.3 Getting help 25
3.4 Tidyverse 25
3.5 Final exercise 29
3.6 Bonus: R Markdown 30

4 Summarizing Data **31**
4.1 Are you ready? 31
4.2 Packages 32
4.3 Numerical methods 32
4.4 Visual methods 35
4.5 Final exercise 46

5 Regression and ANOVA **49**
5.1 To be updated. 49

Chapter 1

Introduction

Hi everyone! I'm Ihnwhi.

It is my great pleasure to be your guest lecturer for PSY202A.

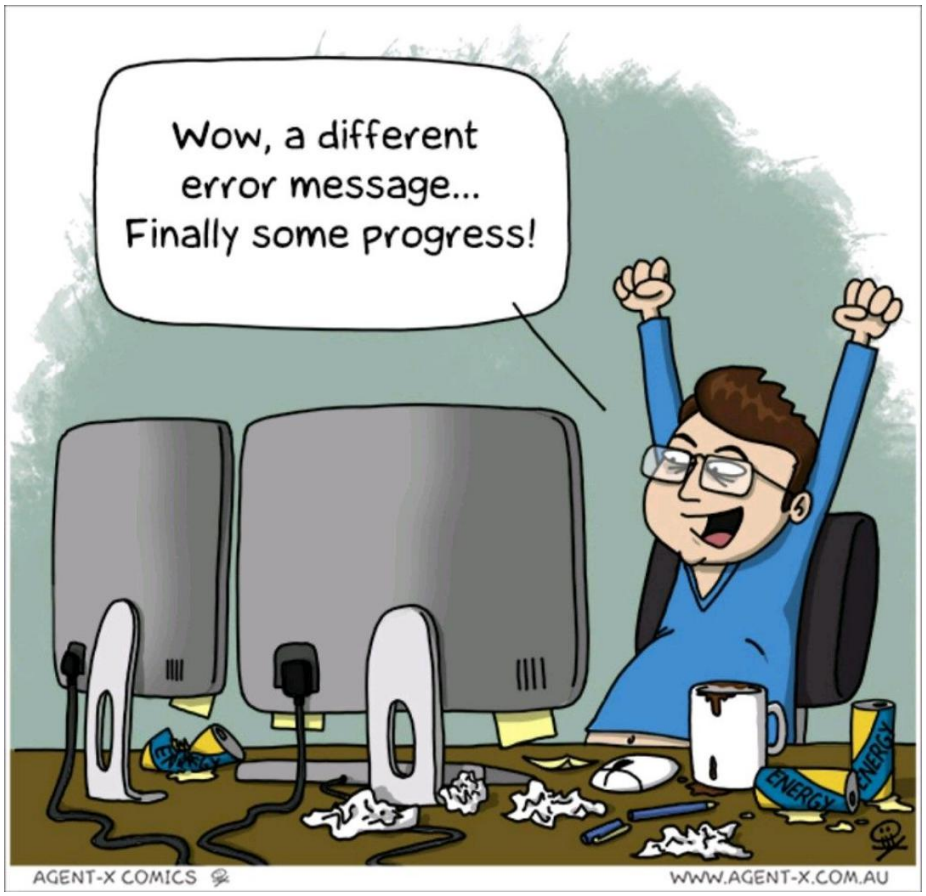
Statistical modeling is a key component in conducting research in the psychological sciences. While many statistical toolkits are available to researchers, R is arguably one of the most useful free and open-source statistical software programs. It offers a dizzying array of analytic options to answer your important and exciting research questions. In the upcoming four lab sessions, you will be introduced to R and become familiar with its capabilities. These sessions are designed to help you get acquainted with the fundamentals of R and learn how to use it wisely as the next generation of psychologists. You will then be guided through summarizing and analyzing data using R.

Are you ready? Let's get it on!

Chapter 2

Introduction to R: Part 1

2.1 Word of wisdom

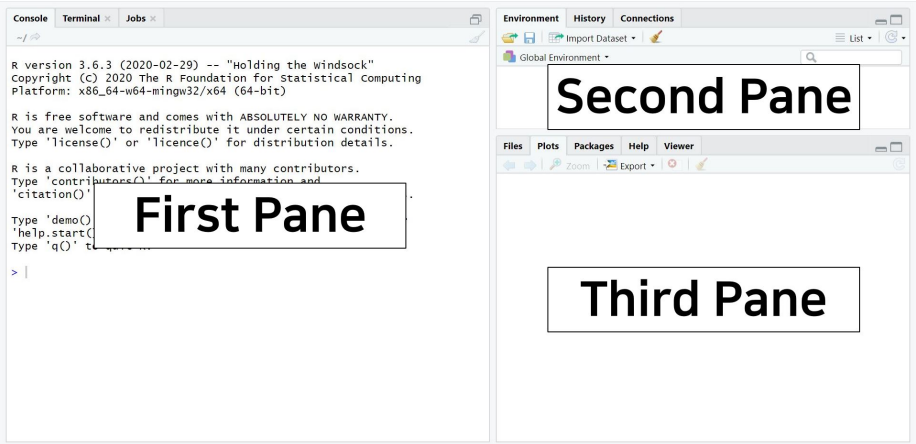


- Don't worry about making mistakes; it's part of the process.
- Feel free to ask questions to me or your peers.
- Remember, there isn't just one way to solve a problem.
- Be a wise user of resources like Google, YouTube, or AI.
- Keep in mind that you're not the only one struggling.

2.2 How R you?

2.2.1 Open RStudio

- Can you find three panes?

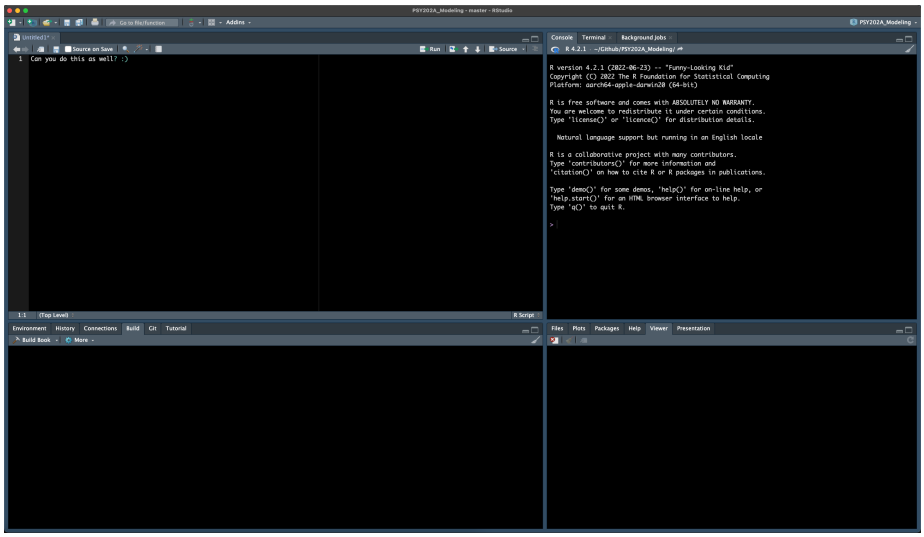


2.2.2 Open a new R script

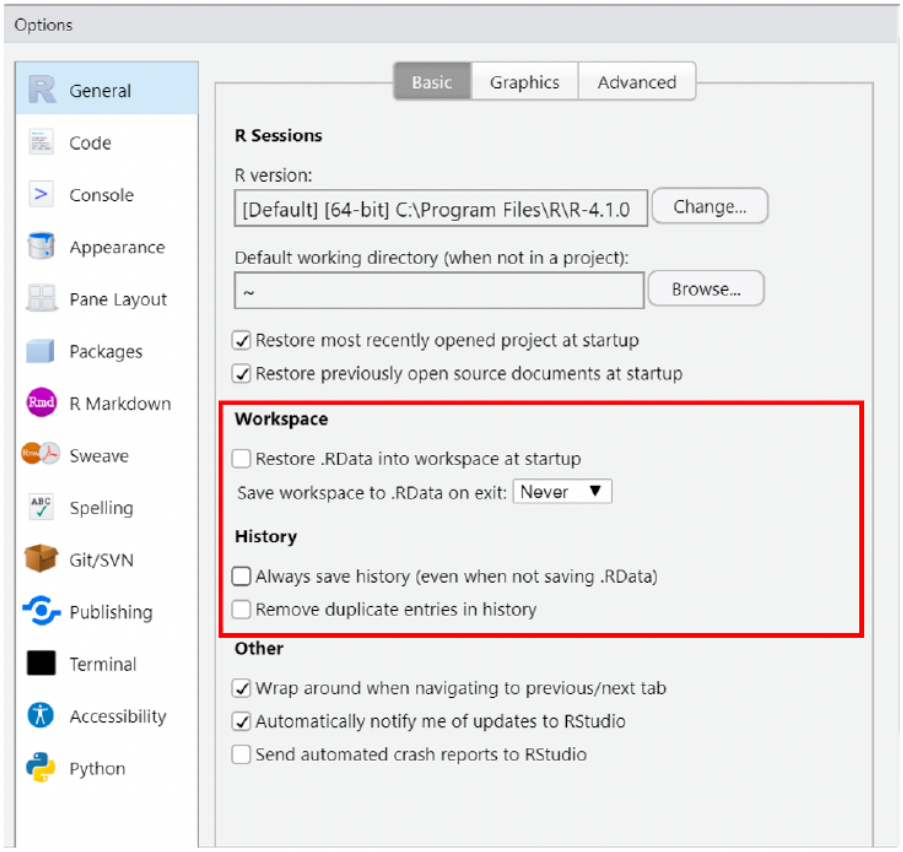
- Can you find four panes?

2.2.3 Individualize R

- Can you find your own R style?


PSY202A_Modeling - master - RStudio
Go to file/function
Addins
PSY202A_Modeling
Untitled1*
Source on Save
Run
Source
1 Can you do this as well? :)
1:1 (Top Level)
R Script
Console
Terminal
Background Jobs
R 4.2.1 · ~/Github/PSY202A_Modeling/
R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
History
Connections
Build
Git
Tutorial
Build Book
More
Files
Plots
Packages
Help
Viewer
Presentation

2.2.4 Some useful settings



- Under "Workspace":
  - Uncheck restore .RData into workspace at startup.
  - Save workshop to .Rdata on exit: "Never".
- Under "History":
  - Uncheck "Always save history (even when not saving .RData).
  - Uncheck "Remove duplicate entries in history".

2.3 Interacting with R

- How to run R commands?
  - Put your cursor on a line, or select the line(s), hit the Run button.
  - The hot key for Run the current line'selection is Ctrl + Enter (Windoes) or Command + Return (Mac).
- Can you run the below code?

```
greetings <- "Hello PSY202A"
print(greetings)
```

- Everything starting with a pound sign (#) is considered as a comment. R will skep all the characters after # until it finds a new line of command.
- Can you run the below code?

```
Comment

# Comment
```

2.4 Computation

2.4.1 Using R as a numerical calculator

- Can you run the code below?

```
2 + 4 # Addition
2 - 4 # Subtraction
2 * 4 # Multiplication
2 / 4 # Division
2 ** 4 # Power
sqrt(144) # Square root
log(144) # Natural logarithm
exp(5) # Exponential, i.e., power of e
(10 + 2*log(8) - (exp(8) - 4)/3) * 7
```

- Here, `sqrt()`, `log()`, and `exp()` are numerical functions.

2.4.1.1 Exercise: Numerical calculator

- Can you give me the answer to the mathematical formula below?

$$\frac{26}{\log(5)} \times e^3 + (-0.2)$$

- How about this?

$$-\sqrt{(\ln 7)}$$

2.4.2 Using R as a logical calculator

- You can also use R for logical statements.

```
1 == 1 # equal
1 != 2 # unequal
```

```r
10 > 1 # greater than
10 >= 1 # greater than or equal to
```

2.4.2.1 Exercise: Logical calculator

- Then... can you program your code to evaluate the below statements?
  - Try this first: 5 is lower than or equal to 10. What is the result? Does this make sense?
  - Then, try this second: 10 is not equal to 10. What is the result? Does this make sense?

2.5 Types of objects

- In R, anything with a name is an object.
  - You can give an object with any name you want insofar as that name is not taken in R already.
- An object can contain more than one value and have more complex data strutures, such as:
  - vector: a series of values of the same data type
  - matrix: a two-dimensional table of values with the same data type
  - data frame: a two-dimensional table of values, may not be the same data type
  - list: an object made of objects

2.5.1 Vector: Intro

- The concatenate function `c()` is usually used to put a set of values together.
  - An object `iq` is a vector of 5 IQ scores.

```r
# A vector of IQ scores
iq <- c(90, 100, 105, 110, 95)

# Here, 'iq' is an object

# Print the IQ object
print(iq)
iq
```

- Out of curiosity, what should we do if we want to know the sum of all the five iq scores? You can use the `sum()` function. Can you try?

```r
# Hint: put an object name within the sum()
```

- Many other functions also return vectors as results.
  - For instance, the sequence operator (:) generates consecutive numbers.

```
numbers <- 1:100
numbers
```

2.5.1.1 Exercise: Vector intro

- There are other specific functions. Can you guess what each of the functions does?

```
# Describe what the below function does:
values_1 <- seq(1, 5, by = 1)

# Describe what the below function does:
values_2 <- seq(1, 5, by = 0.5)

# Describe what the below function does:
values_3 <- rep(1, 5)

# Describe what the below function does:
values_4 <- rep(1:5, 3)
```

2.5.2 Vector: Advanced

- There are a couple of distribution functions that can be used to generate a vector of random numbers from a specific distribution.
- For example, you can generate 5 random numbers from a normal distribution with a mean of 0 and a standard deviation of 1:

```
random_normal <- rnorm(5, 0, 1)
random_normal
```

- As another example, you can generate 5 random numbers from a uniform distribution between 0 and 1:

```
random_uniform <- runif(5, 0, 1)
random_uniform
```

- The standard arithmetic operators and functions apply to vectors on a element-wise basis.

```
A <- c(1, 2, 3, 4) - 4
A

B <- c(1, 2, 3, 4)/4
B

C <- c(1, 2, 3, 4)/c(4, 3, 2, 1)
C
```

```
D <- log(c(1, 2, 3, 4))
D
```

- All elements in a vector must be the same type
  - Numeric, character (aka. string), logic

```
numeric_vector <- c(1, 2, 3, 4)

character_vector <- c("developmental", "health", "quantitative")

logical_vector <- c(FALSE, TRUE, FALSE, FALSE)
```

2.5.2.1 Exercise: Vector advanced

Can you run the code below? What do you see?

```
numeric_vector^2

1/numeric_vector

numeric_vector + character_vector

character_vector + logical_vector

numeric_vector + logical_vector

logical_vector + logical_vector

logical_vector+5
```

2.5.3 Matrix: intro

- We can create a matrix by combining multiple vectors (e.g., perhaps associated with multiple scales)

```
scale1 <- c(1, 4, 7, 4, 5)
scale2 <- c(5, 6, 8, 3, 2)
scale3 <- c(0, 9, 8, 9, 3)
```

- We could arrange these into a table, using `cbind()` function (bind by column).

```
my_data <- cbind(scale1, scale2, scale3)
my_data
```

- Or, if our data were in one long vector:

```
long_vec <- c(scale1, scale2, scale3)
my_matrix <- matrix(long_vec, ncol = 3, byrow = FALSE)
my_matrix
```

2.5.4 Matrix: advanced

- Find the size of the matrix (i.e., the number of rows and the number of columns) using `dim()`:

```
dim(my_matrix)
```

- ,where this parallels `length()` for vectors:

```
length(scale1)
```

2.5.4.1 Exercise: matrix advanced

- Can you create an arbitrary matrix with 5 rows and 3 columns? You can do anything to achieve this goal.
- If you are successful, can you transpose the matrix using the `t()` function?
- What is the size of the transposed matrix?

2.5.5 Data frame

- Similar to matrices, data frames can be created by combining multiple vectors, but data frames allow for different vector types (e.g., number, character, logical), but preserves the characteristics of each type.

```
v1 <- 1001:1006
v2 <- c(407.56, 442.20, 385.85, 295.31, 408.10, 280.52)
v3 <- c("A", "A", "A", "B", "B", "B")

my_dataframe <- data.frame(id = v1, reactiontime= v2, condition = v3)
my_dataframe
```

2.5.5.1 Exercise: data frame

- Can you describe the differences between a matrix and a data frame in R?

2.5.6 List

- Data frames must be rectangular, what if our data are non-rectangular?

```
mod_A <- c(4.25, 2.36, 2.37)
mod_B <- c(4.26, 2.45, 2.31, 7.5)
mod_C <- c(4.21, 2.44, 2.29, 7.7, 4.1)
flag <- c(FALSE, FALSE, TRUE)
```

- A list will allow any set of R objects to be combined into a single object

```
my_list <- list(parA = mod_A, parB = mod_C, parC = mod_C, flag = flag)
my_list
```

2.5.7 Exercise: Selecting elements in objects

- We can select elements of vectors, matrices, data frames, and lists using brackets (i.e., []).
- For each chunk of the code below, think of the expected output before running the code, and see if your guess was correct after running the code:

2.5.7.1 For vectors

- Given the vector below:

```
scale1 <- c(1, 4, 7, 4, 5)
```

- Can you expect the outcome?

```
scale1[3]
```

2.5.7.2 For matrices

- Recall that this is our predefined matrix object:

```
my_matrix
```

- Can you expect the outcome?

```
my_matrix[ , 2]
my_matrix[1,  ]
my_matrix[1, 2]
my_matrix[1:2,  ]
my_matrix[ , 1:2]
```

2.5.7.3 For data frames

- Recall that this is our predefined data frame object:

```
my_dataframe
```

- Can you expect the outcome?

```
my_dataframe[1, 1]
my_dataframe[1:2,  ]
my_dataframe[ , 1:2]
my_dataframe$id
```

2.5.7.4 For lists

- Recall that this is our predefined list object:

```
my_list
```

- Can you expect the outcome?
  - Note that, for list objects, we sometimes need to use the double brackets to extract an element of a part of a list

```
my_list[1]
my_list[[1]]
my_list[2]
my_list[[2]]
my_list$parB
```

2.5.7.5 Another bonus exercise for you

- Create a new data frame using the following vectors:

```
id <- c(1001:1040)
x <- runif(40, 250, 500)
y <- x * 0.2 + runif(40, 200, 450)
gender <- sample(c("M", "F"), 40, replace = TRUE)
```

- Get the mean of y in the data frame you just created. (Hint: use $ to extract the variable; use function mean() to compute the mean.)

2.6 Workspace

- Recall that everything that has a name in R is called an object.
- The workspace is your current R working environment and includes every user-defined objects, such as vectors, matrices, data frames, lists, etc.
- The ls() function will list all the objects in a workspace.
- The object(s) can be removed from the workspace by the `rm()` function.

2.6.1 Functions to manage workspace

- Can you run the code below and see what happens? Can you understand what is happening?

```
ls()
```

- What about this? What happened?

```
rm(v1)
ls()
```

- Shall we remove all the objects?

```
# Remove all the objects in the workspace
rm(list=ls(all = TRUE))
```

- If you haven't unchecked the box about Workspace in the Global Options, at the end of an R session, you will be asked whether you want to save an image of the current workspace that is automatically reloaded the next time R is started. Choose NO.
- Note that the workspace consists only of R objects, not of any of the output that you have generated during a session (e.g., images). If you want to save your output, just copy it from the console.
- Also, there are ways to export your data and save them in some specific formats for further uses.

2.7 Working directory

- The working directory is the location (file path) on your computer where R will look for files and where it will save any files.
  - For more details, see https://www.rensvandeschoot.com/tutorials/r-for-beginners/
- To see your current working directory, use `getwd`:

```
getwd()
```

- You can also check the current working directory by looking at bar at the top of the Console pane.
- To change your working directory, use `setwd()`:
  - For Windows users, ## use / instead of
- You can also manually set the working directory in the Files tab.
- If you start an R session from a .R file, the default working directory will be set to where the .R file is located on your computer.

2.8 Importing external data

- The most common way is to use the function `read.table()` to read in .txt files or .dat files; or `read.csv()` for .csv files.

```
mydata1 <- read.table(file = "SAT.dat", header = TRUE)
class(mydata1)
write.csv(mydata1, file = "SAT.csv", row.names = FALSE)
```

- You can try this as well:

```
mydata2 <- read.csv(file = "SAT.csv", header = TRUE, sep = ",")
class(mydata2)
```

- You can also manually choose a file to import
  - But this will not work for all data formats, so be careful

```
mydata3 <- read.table(file.choose(), header = TRUE)
class(mydata3)
```

- To look at the whole dataset, directly type the name of it.

```
mydata1

mydata2

mydata3
```

- To obtain a single variable from the dataset, use a dollar sign ($)

```
mydata1$Math
```

- Can you get the sum and the mean of the math scores using the `sum()` and `mean()` functions?

2.9 Exporting

- Similar to `read.table()`, the most-used expoert function is `write.table()`.
- You can export your data into some different formats than it was originally imported.
- To export the mydata1 into a tab-delimited file without row names:

```
write.table(mydata1, file = "SAT.txt", sep = "\t", row.names = FALSE, col.names = TRUE)
## sep="\t" tells R to use tab as separator in the text file. You can also try sep=" "
```

- Similar to `read.csv()`, there is a function `write.csv()`.

2.10 Final exercise

1. Create a folder named `PSY202A_Lab 1`, move the data file `SAT.csv` in it. Then create a new R script file using the File menu -> New File -> R Script.
2. Change the working directory to `PSY202A_Lab 1`.
3. In the new R script, write R code to examine the data:

- Read in `SAT.csv` file, name it as datSAT;
- Check the first 6 rows of the data set;
- Find the total number of observations in the data set using the `nrow()` function;

- List all the values in the State variable;
- Compute the mean of Verbal scores;
- Create a new variable `Combined`, which equals the sum of Verbal and Math.
  - Hint: assign the sum of two variables to `datSAT$Combined`.

4. Save the data set datSAT (now with a new variable Combined) to your working directory `PSY202A_Lab 1`, and name it as `SATCombined.csv`.
5. Save your R script as `Exercise 1.R` in your working directory `PSY202A_Lab 1` (File menu -> Save).

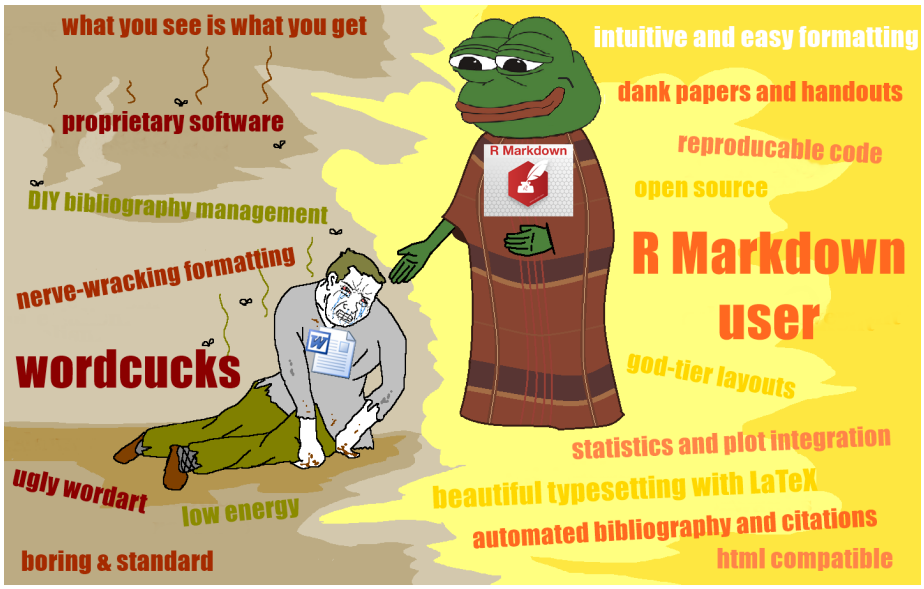
2.11 R Markdown

- Reproducibility is a key philosophical principle in the psychological sciences.
- An easy yet elegant way to ensure reproducibility in R programming is by using R Markdown for documentation.
- I strongly recommend downloading R Markdown before our next meeting. You can find resources at:
  - https://rmarkdown.rstudio.com/lesson-1.html
  - https://vimeo.com/178485416
  - https://yihui.org/tinytex/
- This is a great opportunity to learn how to use R Markdown, which you can then apply when submitting your upcoming homework assignments.

Chapter 3

Introduction to R: Part 2

3.1 What will we do?



3.2 Packages

- A package is a collection of previously programmed functions and data sets, often including functions for specific tasks.
- Some packages come with the base installation of R.
- There are thousands of user-contributed packages that you must manually download and install.
- To see which packages you have, click View -> Show Package.
- To see all available packages: https://cran.r-project.org/web/packages/
- Some packages can be very handy when working with psychological data.
  - The `psych` package is a general-purpose toolbox for analyzing psychological data.
  - The `haven` package can be used to import and export SPSS, Stata, and SAS files.

3.2.1 Installing packages

- The easiest way to install packages and add packages to the base version of R is to use the Tools -> Install Packages...
- Or use `insstall.packages()` function as follows:

```
install.packages("psych")
install.packages("haven")
```

3.2.2 Loading packages

- To access the package you have already installed, load it to the current R session using the function `library()`.

```
library(psych)
library(haven)
```

3.3 Getting help

- There are many ways to getting help for R programming.

3.3.1 Modern way of getting help

- Google is your friend.
- Stack Overflow is your cool friend.
- YouTube is your another wonderful friend.
- AI is your badass friend.

3.3.2 Traditional way of getting help

- When R was installed, HTML format help files were copied on your hard drive.
- To access these files, you can click Help -> R Help.
- Or just type:

```
help.start()
```

- To request an R document for a special function, use "?". To illustrate:

```
?log
# This will pull up an Internet window with everything about the function log()
```

- To request help by keywords, use "??". To demonstrate:

```
??logarithm
# This will give you an information window
# listing all the functions that contain the term "logarithm".
```

- To request help for a specific package, use `help(package = " ")`. That said:

```
help(package = "psych")
```

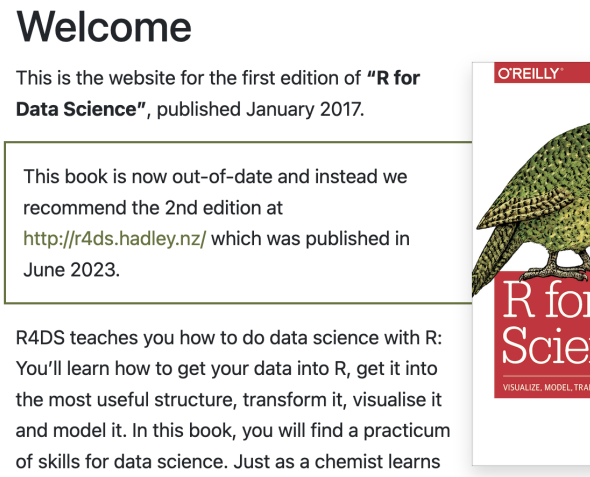
3.4 Tidyverse

- The tidyverse is a collection of R packages for data analysis that are developed with common ideas and norms - the tidyverse style.

- More and more popular in recent years.
- Website: https://www.tidyverse.org/packages
- To install and load the tidyverse packages:

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

- Purpose is to make R code easier to work with:
  - Data manipulation
  - Data visualization
  - Programming
  - Integration with other packages
  - . . . and more
- Book recommended: R for Data Science



  - Freely available at https://r4ds.had.co.nz/

3.4.1 Tidyverse basics

- As it is difficult to change how fundamental base R structures/functions work, the tidyverse suite of packages creates and uses data structures, functions, and operators to make working with data more intuitive.
- The two most basic changes are in the use of pipes and tibbles.

```
# Packages for tidyverse demo
library(datasets)
library(tidyverse)
```

3.4.2 Pipes

- Stringing together commands in R can be quite daunting. Also, trying to understand code that has many nested functions can be confusing.
- To make R code more human readable, the Tidyverse tools use the pipe, `%>%`, which was acquired from the magrittr package and comes installed

automatically with Tidyverse.
- The pipe allows the output of a previous command to be used as input to another command instead of using nested functions.
- Hint: The shortcut to write pipe is `shift + command + M`.

3.4.2.1 Exercise: pipes

- Base R method of running more than one command

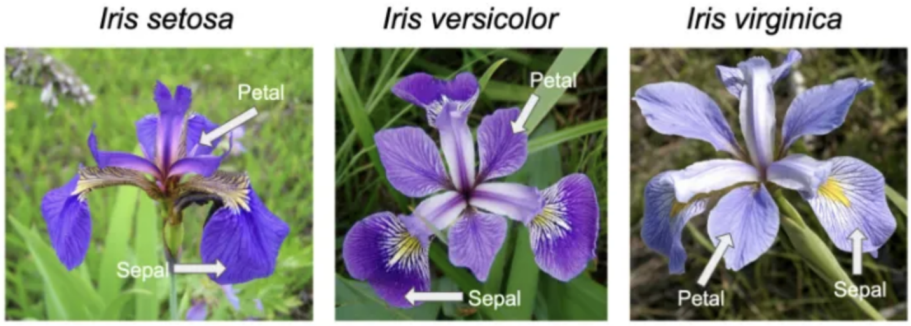
```
sqrt(83)
round(sqrt(83), digit = 2)
```

- Running more than one command with piping

```
sqrt(83) %>% round(digit = 2)
```

3.4.3 Tibbles

- A core component of the tidyverse is the tibble.
- Tibbles are a modern rework of the standard `data.frame`, with some internal improvements to make code more reliable.
- They are data frames but do not follow all of the same rules. For example, tibbles can have column names that are not normally allowed, such as numbers/symbols.
- Tibbles can be created directly using the `tibble()` function or data frames can be converted into tibbles using `as_tibble(name_of_df)`.
- In this section of code, the iris data frame is converted into a tibble. The iris dataset consists of five variables: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`, and `Species`.




```
# Loading the iris dataset from the datasets package
data("iris")

# Converting the iris data.frame into a tibble object
tibble_iris <- as_tibble(iris)
```

```
# Using a data.frame as a tibble with a pipe operator
iris %>% as_tibble()
```

3.4.4 Differences between tibbles and data.frames

- The main differences between `tibbles` and `data.frames` relate to printing and subsetting.

3.4.4.1 Printing

- A nice feature of a tibble is that when printing a variable to screen, it will show only the first 10 rows and the columns that fit to the screen by default.

```
# Default printing of data.frame
iris # Prints 150 rows

# Default printing of tibble
iris %>%
  as_tibble() # Prints 10 row
```

- This is nice since you don't have to specify head() to take a quick look at your dataset.
- If it is desirable to view more of the dataset, the print() function can change the number of rows or columns displayed.

```
# Printing of tibble with print() - change defaults
iris %>%
  as_tibble() %>%
  print(n = 20, width = Inf)
```

3.4.4.2 Subsetting

- When subsetting base R data.frames the default behavior is to simplify the output to the simplest data structure (i.e., a vector).
- If you use piping to subset a data frame, then the notation is slightly different from base R, requiring a placeholder `.` prior to the `[ ]` or `$`.

```
# Subsetting the Species variable in base R
iris$Species
iris[ ,"Species"]

# Subsetting the Species variable in output using a pipe
iris %>% .$Species
iris %>% .[ ,"Species"]
```

- Note that some older functions do not work with tibbles, so if you need to convert a tibble to a data.frame, the function

as.data.frame(name_of_tibble) will easily convert it.

3.4.5 Tidyverse tools

- Tidyverse has many tools for data wrangling, cleaning, and visualization.

3.4.5.1 dplyr

- Perhaps the most useful tool in the tidyverse is dplyr. It's a Swiss-army knife for data wrangling.
- dplyr has many handy functions:
  - `select()` extracts columns and returns a tibble.
  - `arrange()` changes the ordering of the rows.
  - `filter()` picks cases based on their values.
  - `mutate()` adds new variables that are functions of existing variables.
  - `rename()` easily changes the name of a column(s).
  - `summarise()` reduces multiple values down to a single summary.
  - `pull()` extracts a single column as a vector.
  - `_join()` group of functions that merge two data frames together (e.g., `inner_join()`, `left_join()`, `right_join()`, and `full_join()`).
- Here is an example of the `select()`, `filter()`, and `summarise()` functions using the iris dataset.

```
# Only select the columns related to the Sepal of the iris
iris %>%
  select(Sepal.Length,Sepal.Width) %>%
  head()
```

```
#F ilter for irises with a Sepal.Length greater than 5 and Sepal.Width greater than 4
iris %>%
  filter(Sepal.Length > 5, Sepal.Width > 4)
```

```
# Calculate the average Sepal.Length and Sepal.Width for each iris Species
iris %>%
  group_by(Species) %>%
  summarise(mean.Sepal.Length = mean(Sepal.Length), mean.Sepal.width = mean(Sepal.Width))
```

3.5 Final exercise

1. Subset the `Species` column from the iris dataset using a pipe.

- Hint: Use a `.` before the `[ ]` or `$`.

2. Select the `Petal.Length` and `Petal.Width` columns from the iris dataset.
3. Find the average `Petal.Length` and `Petal.Width` for each iris Species.

3.6 Bonus: R Markdown

- If you have not downloaded the R Markdown yet, please go to https://rmarkdown.rstudio.com/lesson-1.html and download the `rmarkdown` package.
- R Markdown is a tool for reproducible documentation in statistics. R markdown generates a new file that contains selected text, code, and results from the `.Rmd` file.
- The newly created file can be a finished website, PDF document, Word document, slide (beamer, powerpoint, xaringan) show, notebook, handout, book, dashboard, Shiny Apps, package vignette, or more.

3.6.1 How does the R Markdown work?

- According to https://rmarkdown.rstudio.com/lesson-2.html,
  - When you run render, R Markdown feeds the `.Rmd` file to `knitr`, which executes all of the code chunks and creates a new markdown (.md) document which includes the code and its output.
  - The markdown file generated by `knitr` is then processed by `pandoc` which is responsible for creating the finished format.
  - This may sound complicated, but R Markdown makes it extremely simple by encapsulating all of the above processing into a single render function.

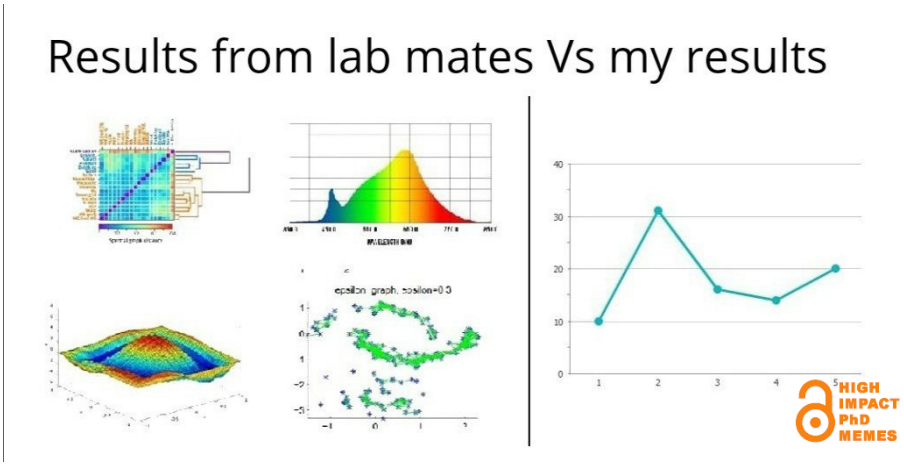
3.6.1.1 Practice

- Generate any PDF document from R Markdown by knitting your markdown file.
- You may need to visit:
  - https://yihui.org/tinytex/
  - https://yihui.org/tinytex/r/#debugging

Chapter 4

Summarizing Data

4.1 Are you ready?



4.1.1 Wait... you guys are getting results???!

4.2 Packages

- Here is a list of packages that we will use today:
  - `modeest` has the `mfv()` function for the mode.
  - `moments` has the `skewness()` and `kurtosis()` functions.
  - `tidyverse` contains the `ggplot2` package for data visualization.
- For those who hope to dive into the fancier data visualization tools, consider:
  - RShiny at https://shiny.posit.co/r/gallery/.
  - Extensions of the `ggplot2` package at https://exts.ggplot2.tidyverse.org/gallery/.

4.3 Numerical methods

4.3.1 Peabody data

- For the purpose of today's demonstration, we will use the following data from 40 children who completed the Peabody Developmental Motor Scales (PDMS).
- The PDMS is a popular standardized test used by physical and occupational therapists for children less than six.
- The variables include peabody scores, gender, and age (in months).

- The dataset is as follows:

```
# Peabody data
peabody <- c(123,99,80,81,90,104,69,86,107,94,82,99,83,98,86,74,67,90,106,102,
             68,80,76,65,69,81,76,68,84,88,68,92,92,97,78,77,94,74,89,68)

gender <- c("M","F","F","F","F","M","F","F","F","F","M","M","M","M","M","M","M",
            "M","M","M","F","F","F","M","F","F","F","M","M","F","M","M","M","M",
            "M","M","M","M","M","M")

age <- c(35,45,38,40,38,42,36,45,34,36,36,51,34,35,37,37,36,36,37,32,22,28,18,20,
         25,18,14,26,14,24,14,26,11,12,26,17,16,23,12,13)
```

- These variables can be stored in a data frame with the following code:

```
# Make a data frame
peabody_df <- data.frame(peabody = peabody, gender = gender, age = age)
```

4.3.2 Central tendency

- Mean

```
mean(peabody)
```

```
## [1] 85.1
```

```
mean(peabody_df$peabody)
```

```
## [1] 85.1
```

- Median

```
median(peabody)
```

```
## [1] 83.5
```

```
median(peabody_df$peabody)
```

```
## [1] 83.5
```

- Mode

```
library(modeest)
mfv(peabody)
```

```
## [1] 68
```

4.3.3 Variability

- Variance

```
var(peabody)
```

```
## [1] 181.1692
```

- Standard deviation

```
sd(peabody)
```

```
## [1] 13.45991
```

4.3.4 Skewness

- There are multiple methods that can be used to calculate skewness in R:
- You could use the formula for skewness.

```
# Define sample size
n <- length(peabody)

# Compute the skewness
(sum((peabody-mean(peabody))^3)/n)/(sum((peabody-mean(peabody))^2)/n)^(3/2)
```

```
## [1] 0.524655
```

- But the `skewness()` function in the `moments` package makes it even easier.

```
library(moments)
```

```
##
## Attaching package: 'moments'
```

```
## The following object is masked from 'package:modeest':
##
##     skewness
```

```
skewness(peabody)
```

```
## [1] 0.524655
```

4.3.5 Kurtosis

- There are multiple methods for calculating kurtosis in R.
- You can use the formula:

```
n <- length(peabody)
(sum((peabody-mean(peabody))^4)/n)/(sum((peabody-mean(peabody))^2)/n)^2
```

```
## [1] 2.911796
```

- Or you can use the `kurtosis()` function in the `moments` package.

```
kurtosis(peabody)
```

```
## [1] 2.911796
```

4.3.6 Using the tidyverse package

- tidyverse allows us to use the `summarise()` function.
  - which allows us to get various descriptive statistics such as minimum, maximum, mean, median, variance, standard deviation, skewness, kurtosis, etc.
- As a small exercise, try to run the code below. What do you see?

```
summarise(peabody_df,
          mean.peabody = mean(peabody),
          sd.peabody = sd(peabody),
          skew.peabody = moments::skewness(peabody),
          kurt.peabody = moments::kurtosis(peabody))
```

4.4 Visual methods

- Be sure to refer to the cheatsheet I have attached on CatCourses!
- Every useful piece of information is there!

4.4.1 Using the tidyverse package

- The `ggplot2` package is a core component of the tidyverse that allows us to build data visualizations.
- ChatGPT is so good at programming `ggplot2` code for you. But of course, sometimes it itself does produce wrong code, particularly when you want to do complicated visualizations.
  - It is suggested to know the fundamental grammar to be a smart user of the AI!
- The idea behind `ggplot2` is that every new concept we introduce will be layered on top of the information we've already learned.
- In this way, `ggplot2` uses layers of information added on top of each other to help build the graph.
  - This is most evidenced by how each line is separated by a plus sign (+). Each line of code is a different layer of the graph.
- You'll see what I mean as we go through several example graphs.
- Remember to work with a tibble instead of a data frame. Tibbles work much better in tidyverse.

```
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
## Warning: package 'tidyr' was built under R version 4.2.3
```

```
## Warning: package 'readr' was built under R version 4.2.3
```

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
## Warning: package 'stringr' was built under R version 4.2.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
```

```
peabody_tib <- as_tibble(peabody_df)
peabody_tib
```

```
## # A tibble: 40 x 3
##    peabody gender   age
##      <dbl> <chr>  <dbl>
##  1     123 M         35
##  2      99 F         45
##  3      80 F         38
##  4      81 F         40
##  5      90 F         38
##  6     104 M         42
##  7      69 F         36
##  8      86 F         45
##  9     107 F         34
## 10      94 F         36
## # i 30 more rows
```

- The type of plot you want to make is referred to as a `geom`. There are dozens of `geom` options available in `ggplot2`. For todays' lesson, we will focus on four specific `geoms`:
  - `geom_boxplot()`
  - `geom_histogram()`
  - `geom_bar()`
  - `geom_point()`
- When using `ggplot2`, plots will take the following general form:

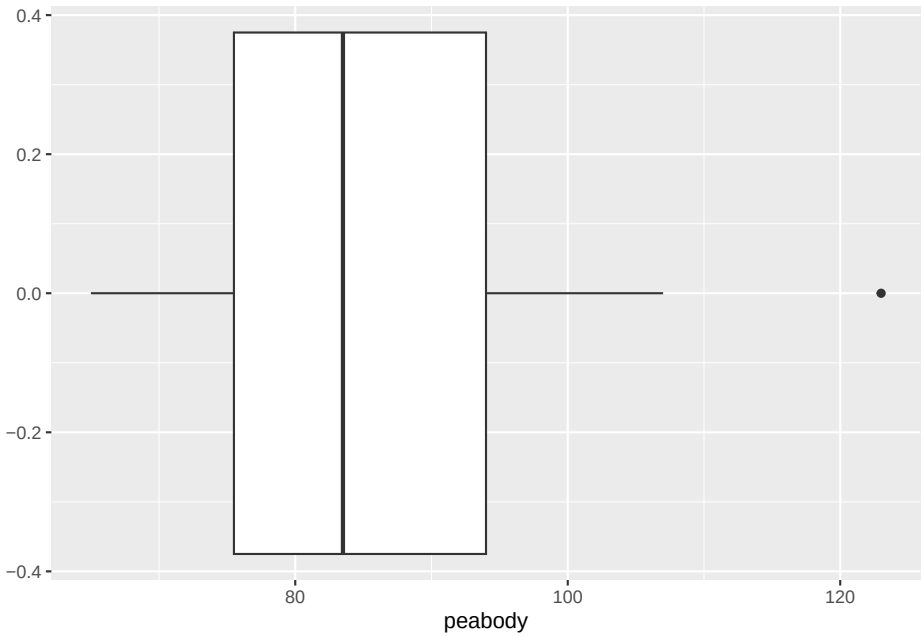
```
ggplot(data = DATASET, aes(VARIABLE(S))) +
  geom_PLOT_TYPE()
```

- First, you start with the `ggplot()` function, where you will specify the data.
- Second, you will include the `aes` argument, which is where you specify the variable(s) you want to plot.
- Third, you select the geom type (e.g., `geom_boxplot()`) you are interested in plotting. This is also where you can begin to customize your plots.

4.4.2 Boxplot

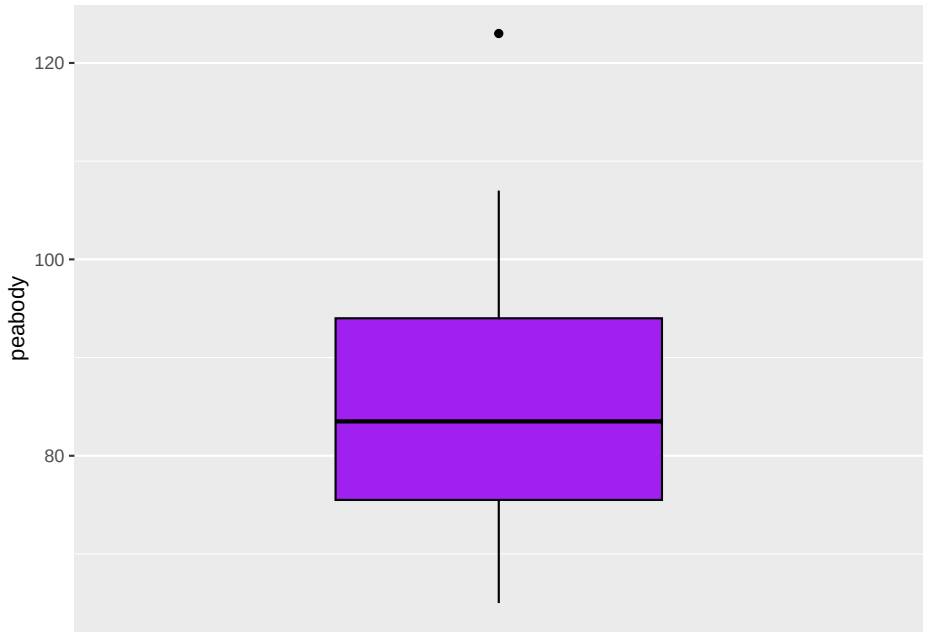
- Boxplots are a useful method for presenting the dispersion of the data, the symmetry of the distribution, and the potential outliers (or extreme scores).
- The following code provides a basic `ggplot2` setup for a boxplot of the peabody scorees:

```
ggplot(data=peabody_tib, aes(peabody)) +
  geom_boxplot()
```



- By adding another layer, you can further customize your plot. In the code below, I illustrate how you can change the coloring of the boxplot and remove the tick marks on the x-axis.

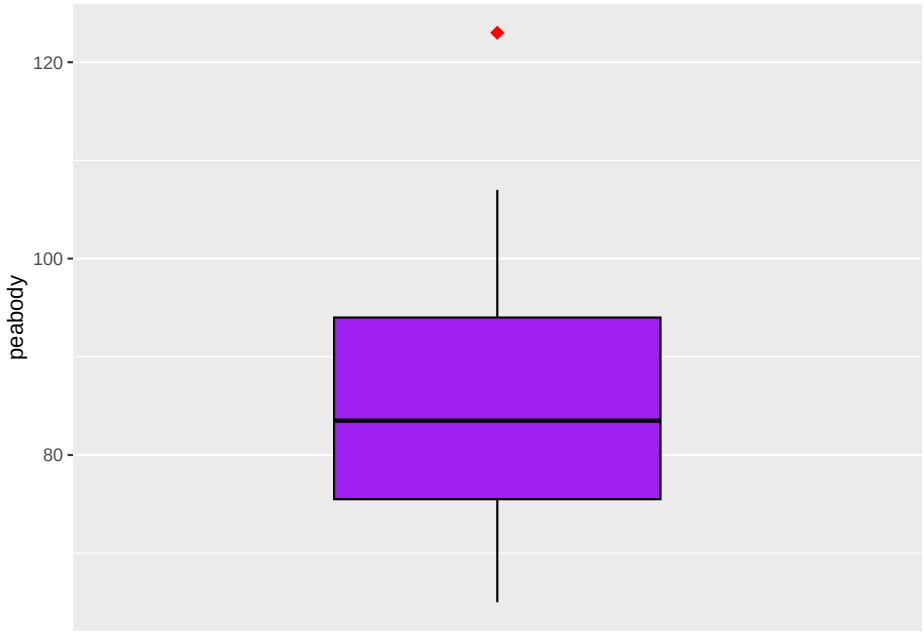
```
ggplot(data = peabody_tib, aes(y=peabody)) +
  geom_boxplot(fill = "purple", colour = "black") +
  scale_x_discrete( ) ## removes x-axis ticks
```



- You can also change the shape and color of the outlier.

- All you have to do is add another line of code that specifies the outlier color, shape, and size.

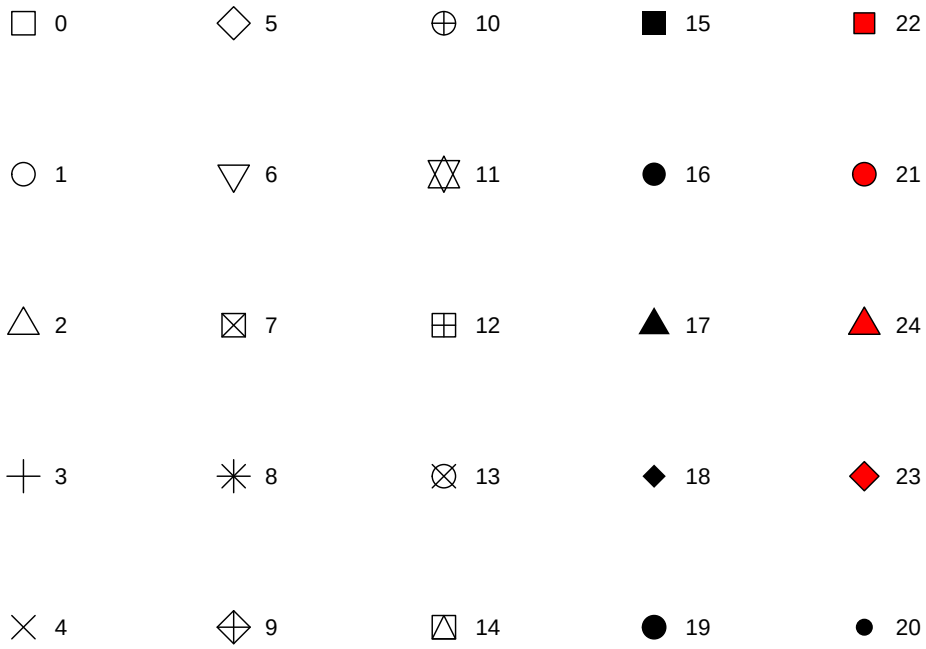
```
ggplot(data = peabody_tib, aes(y=peabody)) +
  geom_boxplot(fill = "purple", color = "black",
               outlier.color = "red", outlier.shape = 18, outlier.size = 3) +
  scale_x_discrete( ) ## removes x-axis ticks
```



- There are several available shapes in `ggplot2`.
  - I provide some examples below, but you can see more here: https://ggplot2.tidyverse.org/articles/ggplot2-specs.html

```
shapes <- data.frame(
  shape = c(0:19, 22, 21, 24, 23, 20),
  x = 0:24 %/% 5,
  y = -(0:24 %% 5)
)

ggplot(shapes, aes(x, y)) +
  geom_point(aes(shape = shape), size = 5, fill = "red") +
  geom_text(aes(label = shape), hjust = 0, nudge_x = 0.15) +
  scale_shape_identity() +
  expand_limits(x = 4.1) +
  theme_void()
```



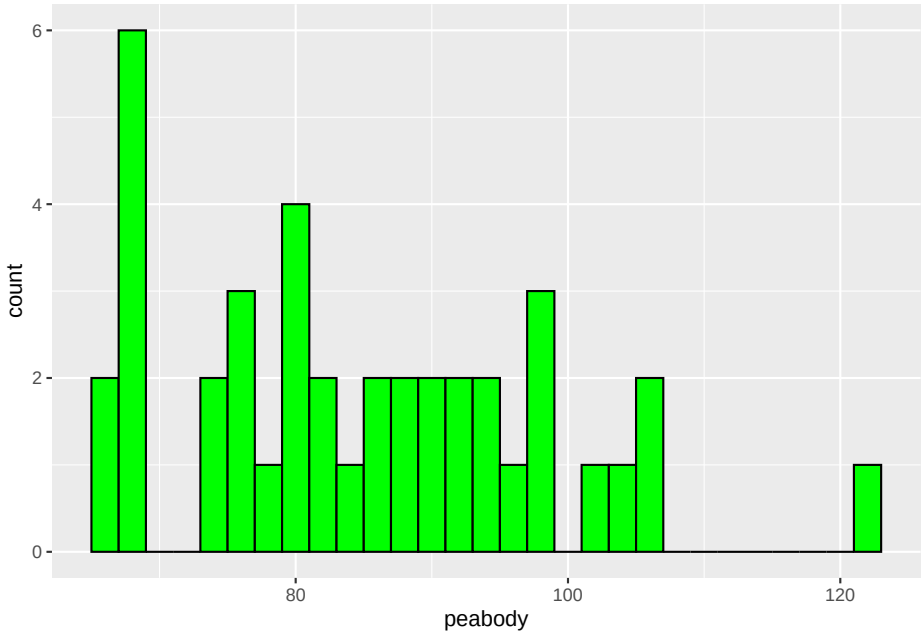
4.4.3 Histograms

- Histograms are a useful method for presenting the dispersion of the data and the symmetry of the distribution.

- The following code provides the setup for a histogram of the peabody scores:

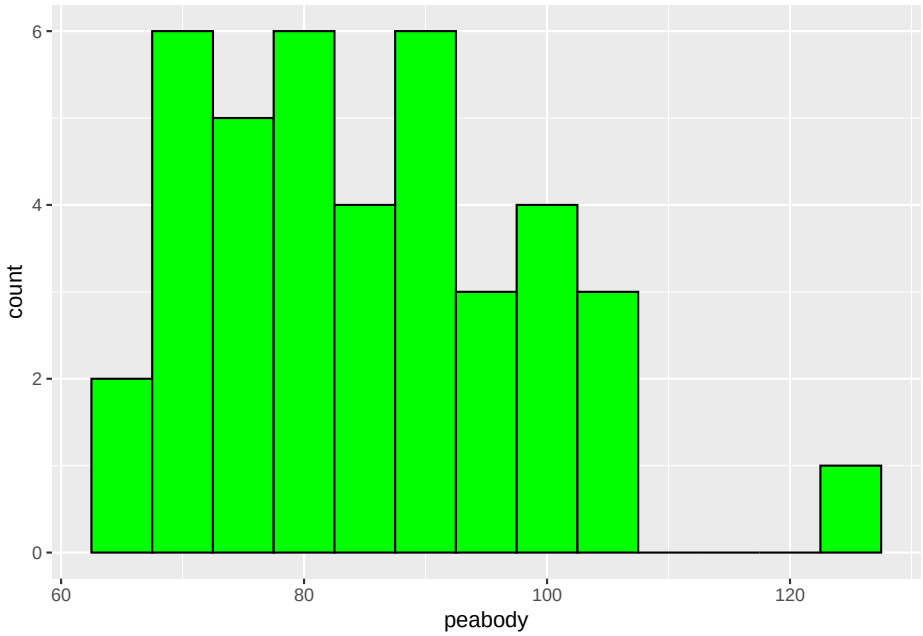
```
ggplot(data = peabody_tib, aes(peabody)) +
  geom_histogram(color = "black", fill = "green")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



- You can change the bindwidth of the histogram using the following code:

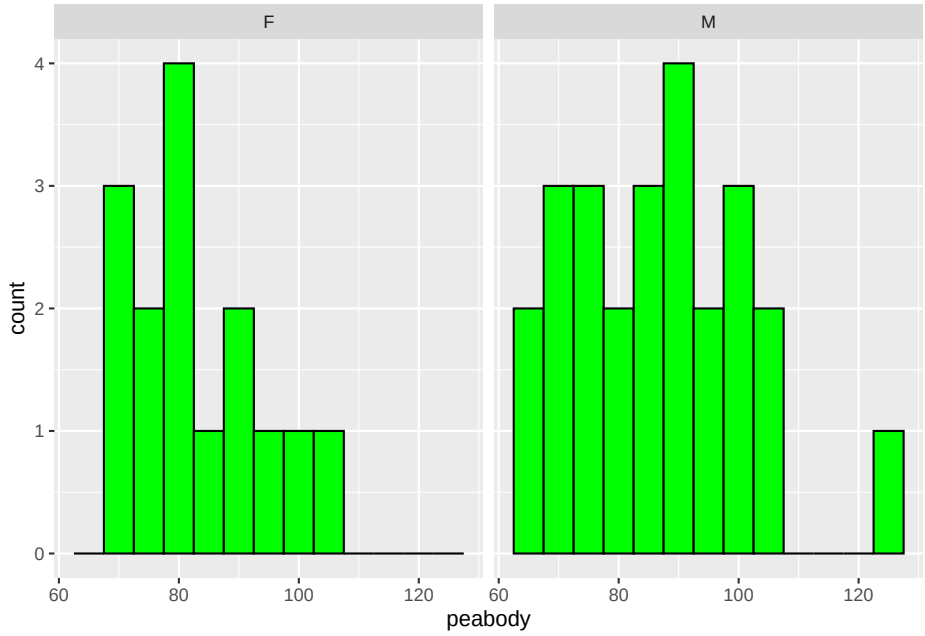
```
ggplot(data = peabody_tib, aes(peabody)) +
  geom_histogram(color = "black", fill = "green",
                 binwidth = 5)
```



- Alternatively, you could set the total number of bins with the `bins` argument.

- Sometimes you want to display plots for a subset of your data. This can be accomplished using the `facet_wrap()` function.

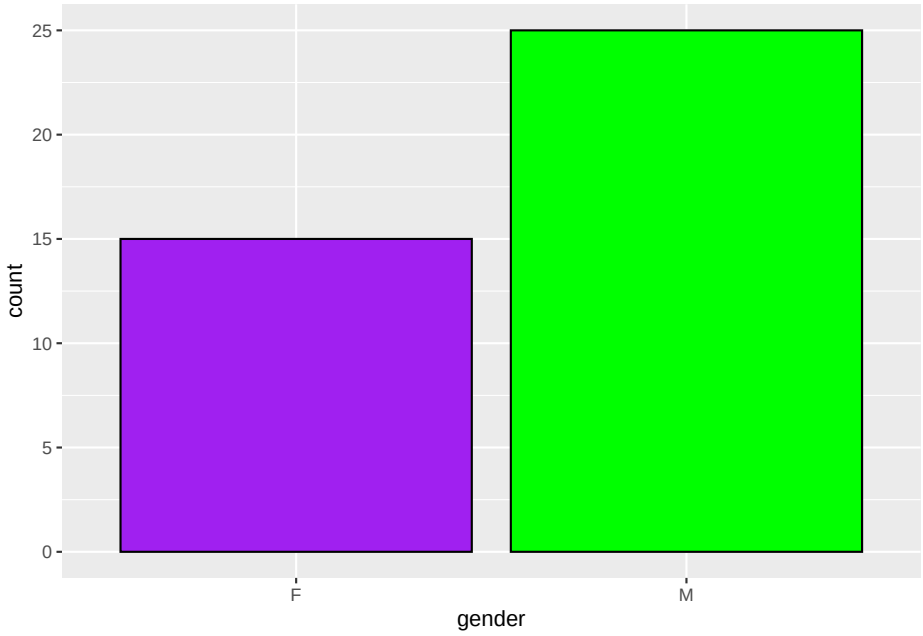
```
ggplot(data = peabody_tib, aes(peabody)) +
  geom_histogram(color = "black", fill = "green",
                 binwidth = 5) +
  facet_wrap(~gender, nrow = 1)
```



4.4.4 Bar charts

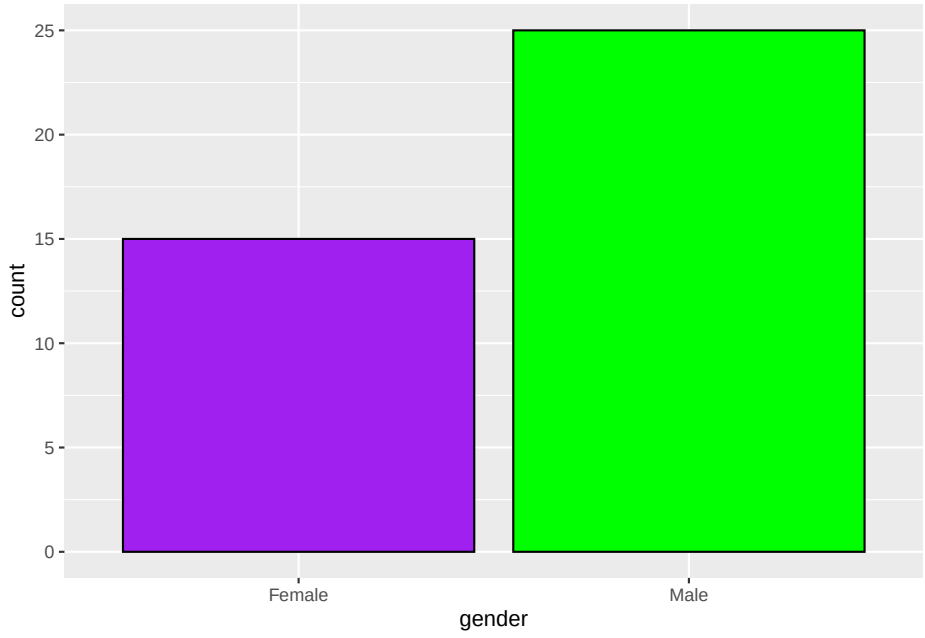
- Bar charts are a useful method for displaying categorical data. They can show the number of cases in each of the categories.

```
ggplot(data = peabody_tib, aes(gender)) +
  geom_bar(color = "black", fill = c("purple","green"))
```



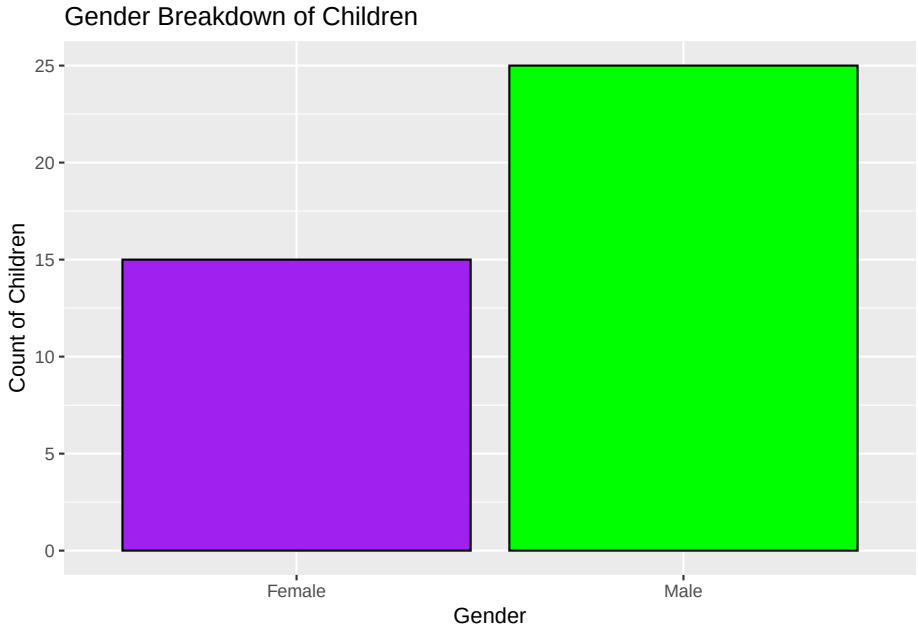
- Perhaps you would like to change the `F` to Female and the `M` to Male. The x-axis ticks can be changed using the following trick:

```
ggplot(data = peabody_tib, aes(gender)) +
  geom_bar(color = "black", fill = c("purple","green")) +
  scale_x_discrete(labels = c("Female", "Male")) ## Changes the labels on the x-axis ticks
```



- Sometimes it can be helpful to change the plot title, x-axis and y-axis.

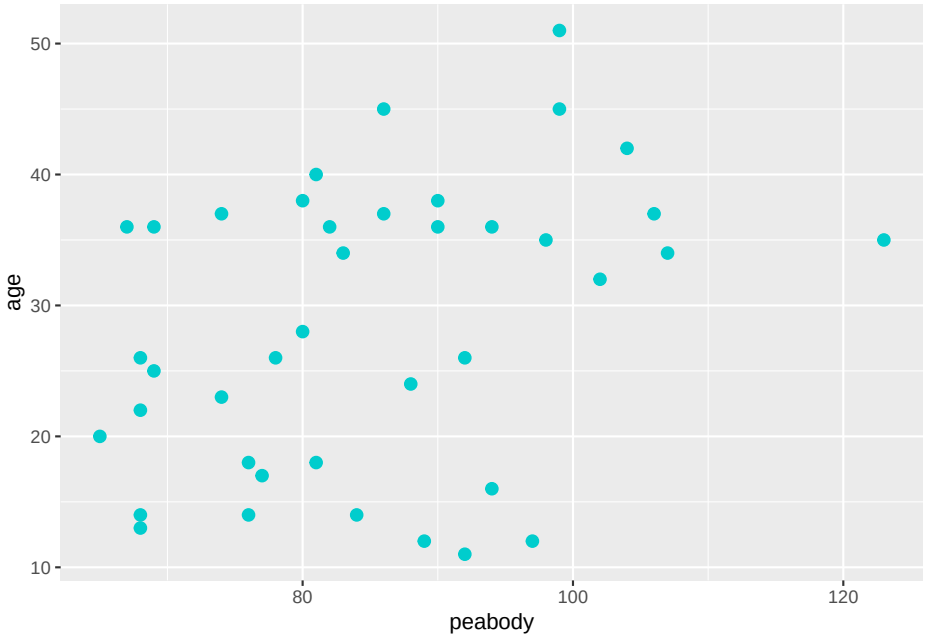
```
ggplot(data = peabody_tib, aes(gender)) +
  geom_bar(color = "black", fill = c("purple","green")) +
  scale_x_discrete(labels = c("Female", "Male")) +
  labs(x='Gender',
       y='Count of Children',
       title='Gender Breakdown of Children') ## Allows you to change the title and axi
```



4.4.5 Scatterplots

- Scatterplots are a useful method for visualizing the relationship between two numerical variables. We will not discuss how to interpret the scatterplot today, but I thought it might be helpful for you to see how the `geom_point()` function works.

```
ggplot(data = peabody_tib, aes(peabody,age)) +
  geom_point(shape = 19, size = 2.5, color = "cyan3")
```



4.4.6 Important points about data visualization

- At its core, the term 'data visualization' refers to any visual display of data that helps us understand the underlying data better.
- Generally, there are a few charactersitics of all godo plots [Note: This is not an exhaustive list].
  - Clearly-labeled axes.
  - Text that are large enough to see.
  - Axes that are not misleading.
  - Data that are displayed appropriately considering the type of data you have.

4.5 Final exercise

- Using the age variable, find the following summary statistics:
  - mean
  - median
  - standard deviation
  - skewness
  - kurtosis
- Using the age variable, create a histogram with the following features:
  - colored bars.

- 10 bins [Hint: use `bins` argument].
- a title of "Age of Children (in months)" [Hint: use `labs()` function].

Chapter 5

Regression and ANOVA

5.1 To be updated...

Ihnwhi is pondering the best way to teach this section.