Assignment 1 Guide

BIOL4062/5062: Analysis of Biological Data

Reid Steele (Based on work by Ana Eguiguren)

2025-09-19

Table of contents

Introduction

Welcome to the assignment guide for BIOL4062/5062: Analysis of Biological Data.

This website is designed to walk you through the assignments for this class. It is a resource to help you figure out how to code for your assignments, and bring attention to key questions to ask yourself as you interpret your results, both statistically and biologically. Keep in mind that there are always different ways to get to the right answer with coding. The content here is not a monolith. You don't need to follow it if you don't want to (more in Assignment Guidelines), but make sure what you are doing is clear and sufficiently analogous to this guide, else you lose marks for being unclear, or running the wrong analyses.

Getting Help

Don't suffer in silence! If you need help on the assignments, there are multiple options available to you:

- Assignment Drop-In Sessions:
  - TA-run in-person help sessions the week before each assignment is due.
  - Optional, but recommended
    - Even if you don't have questions, you may benefit from hearing the questions of others
- BrightSpace Discussion Board:
  - Feel free to ask questions on the BrightSpace discussion board, or peruse questions already asked
    - You may find your answer without even asking!
  - Make sure to start your question with the assignment number
- Email
  - Feel free to ask questions, or set up an appointment

- Direct Assignment 1 questions to the TA, and Assignment 2 and/or class administration (e.g. extension requests) to the instructors

Without further ado, let's get into it!

Assignment Guidelines

This section provide general advice on how to approach your assignments, and also describes how to format them:

General Advice

1. Read the grading rubric carefully!

- It is designed to be as objective as possible. There is little latitude for part marks if you are missing things that are listed.

2. You don't need to describe the statistical theory (unless it's relevant to your answers)

- All you have to do is answer the questions in the assignment. Anything you write outside of that is just eating up your page limit.

3. However, biological interpretations are important: put them together at the end

- Your interpretations are more likely to make sense and be easier to mark if you put them at the end, including all of your results together in them, rather than inserting them throughout piecemeal.
- **Also, make sure your biological interpretations are consistent with your data/results! They don't have to be correct, but they do have to match your data.**

4. Make sure your figures are readable

- We can't tell if your interpretation of your figures is correct if we can't interpret your figures.
- All text on figures should be legible.
- If you use color, make sure that the colors you use are clearly distinguishable (and consider colour-blind safe palettes)

5. You're not alone!

- If you have questions, come to the drop-in sessions, read the Brightspace discussion boards, or email the TA to ask questions or set up a meeting if those options don't work for you.
- Do the first two (drop-in sessions, Brightspace discussion boards) even if you don't have questions: you may find the answer to questions you didn't know you had.
- You're also welcome to ask questions after an assignment has been graded, if you want to know why you were graded the way you were, or if you have questions about the comments provided or what you may have done wrong.

6. Ask if you need an extension

- We're pretty reasonable.

Submission Formatting

Please format your submission following these guidelines: it will make both our lives easier.

1. Hand in your assignment in 3 parts:

a) Your assignment text

- All assignment 1s have a 2 page limit. Put all your text first, with figures and tables together separately at the end. It is easier for the TA to tell how many pages you wrote this way. You're not going to lose marks if you're slightly over (this is not a writing class).
- Should be a word doc or PDF file so it can be opened in BrightSpace. It doesn't matter whether it's produced through word, markdown, etc, as long as it's in one of those two formats.

b) Your script, submitted as a .txt file

- Submitting as a .txt file allows us to open the script in BrightSpace rather than having to download it if it's a .R file. Please don't paste your script in your assignment document.

c) The data you read into your script

- This is just to make it easy to run your script if there is a mistake.

Note: If you do your assignments in markdown, you can combine a) and b)

2. Don't put your name on your assignments, in your scripts, or in any of your file names

- The BrightSpace system of using Banner ID numbers is anonymous so your assignments are marked blind by the TA. That doesn't work if you write your name.
- Delete your file paths in your script for submission if they have your name in them.
- **You do need to put your B0 number, data code number, and whether you're a graduate student or an undergraduate student.**

3. **Follow the assignment guides on this site**

   - You don't have to follow this guide to get full marks on the assignments (as always, there are many correct answers when it comes to statistics and coding). That said, it's easier to follow what you're doing if you're doing the same thing as everyone else.
   - It's OK to do your own thing, but if you make a mistake, its going to be much harder to help you out, and it's going to take significantly more effort to mark.

Part I

R Refresher

1 R Refresher Part 1: First Movements

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers basic R workspace management.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

1.1 Working Directories:

Your working directory is the default file path your R session is working under. You can find your working directory using the `getwd()` command. If you're working in an R project, it will be set to the path of the project by default.

```
# First of all, let's check our working directory
# (where we get files from and save our work in)
getwd()
```

```
[1] "C:/Users/R3686/OneDrive - Dalhousie University/Documents/Teaching/Anaylsis of Biological
5062-Assignment-Guide"
```

You can change your working directory if needed using `setwd()`.

```
# you can change it using the function 'setwd()' and
# pasting the directory path in quotes inside the parentheses
# note that in R we always use "/" or "\\" instead of "\" on mswindows systems

# For example:
# setwd("C:/Users/R3686/OneDrive - Dalhousie University/Documents/Teaching/Anaylsis of Biolog

# You can also set up your working directory in Rstudio
#by clicking in Session>Set Working Directory>Choose Directory;
#or in R by clicking in File>Change Directory
```

You can see what is inside your working directory using `dir()`, and pull out specific files of interest using a pattern with `list.files()`

```
# Let's take a look in what we have in our working directory
dir()
```

```
 [1] "_book"                          "_freeze"
 [3] "_quarto.yml"                    "a1a.qmd"
 [5] "a1b.qmd"                        "a1c.qmd"
 [7] "a1d.qmd"                        "a1e.qmd"
 [9] "Assignment_Helpers.Rproj"       "cover.png"
[11] "cuse.csv"                       "data1c.txt"
[13] "docs"                           "fishcatch.csv"
[15] "FOME_Primary_BlueGradient.png" "FOME_Small.png"
[17] "guidelines.html"                "guidelines.qmd"
[19] "index.html"                     "index.qmd"
[21] "monkey.csv"                     "refresher_1.html"
[23] "refresher_1.qmd"                "refresher_1.rmarkdown"
[25] "refresher_1_files"              "refresher_2.qmd"
[27] "refresher_3.qmd"                "refresher_4.qmd"
[29] "refresher_5.qmd"                "refresher_6.qmd"
[31] "refresher_7.qmd"                "refresher_8.qmd"
[33] "refresher_9.qmd"                "renv"
[35] "renv.lock"                      "Schoenemann.csv"
[37] "site_libs"                      "snake.csv"
[39] "Workspace.RData"
```

```
# Pick out all the .csv files
list.files(pattern = '.csv')
```

```
[1] "cuse.csv"        "fishcatch.csv"   "monkey.csv"      "Schoenemann.csv"
[5] "snake.csv"
```

```
# If you are already sick of R, you can quit by typing:
#q()
```

1.2 Workspace Management

You can save your R workspace into your current directory using `save.image()`, or save specific objects using `save()`. This is useful for saving results that might change due to randomization,

or that may take a long time to generate, such as complex models. You can then load these objects back in later using `load()`.

```
# You can save your work before you go (useful sometimes)
# save.image('Workspace.RData')#but remember to give it a useful name
# save(x, 'Workspace.RData')
#
# load('Workspace.RData')
```

Your R workspace is the environment which contains all of the R objects you've made. By default, everything in your R workspace is visible in the top-right panel in RStudio. You can list everything in your workspace using `ls()`, and delete individual objects using `rm()`. These commands can be combined using `rm(list=ls())` to completely clear your workspace.

You can make objects in your workspace using `<-` or `=`. Object names can be anything - try to make them informative so that you'll know what they are. Remember, everything is case-sensitive in R. X is a different object from x.

```
# Let's take a look in our workspace
ls()
```

```
character(0)
```

```
# It's empty: no objects.Sure? what about x?
x
```

```
Error: object 'x' not found
```

```
# See, there's no such object. Let's create one called x
x <- 2

x = 2

# '<-' assigns values to objects, works the same as '='
# here is our object x

x
```

```
[1] 2
```

```r
# Let's take another look in our workspace. See all you objects there?
ls()
```

```
[1] "x"
```

```r
# RS Note: We can use ls to clear the entire workspace
rm(list = ls())
```

2 R Refresher Part 2: Types of Data and Data Structures

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

There are many different types of objects in R. This section is designed to introduce you to them, and the basics of how to work with them.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

2.1 Data Types

R includes several different data types, all of which behave differently. The data type of an object can be checked using `class()`.

2.1.1 Numeric

Numeric objects are numbers, which are used to perform arithmetic and associated mathematical functions. Other objects can be converted to numeric using `as.numeric()` and you can check if an object is numeric using `is.numeric()`. Integers are a subclass of numerics which lack decimals, and can be converted/checked using `as.integer()` and `is.integer()`.

```
# 2. Types of Data and Data Structures-----------------------------------------------------

# There are many types of data structures in R: vectors, matrices,
#dataframes, arrays, lists, functions etc.
#that contain different types of data

# 2.a) Types of Data -----
# ~~~numeric
a <- c(2,4,7,8)
```

```r
# Check class
class(a)
```

```
[1] "numeric"
```

```r
# Convert to character
test = as.character(a)
as.numeric(test)
```

```
[1] 2 4 7 8
```

```r
# these are also integers
is.integer(a)
```

```
[1] FALSE
```

```r
a# prints the object
```

```
[1] 2 4 7 8
```

```r
summary(a) #summarizes data
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   2.00    3.50    5.50    5.25    7.25    8.00 
```

```r
# mathematical operations can be done:
a + 1
```

```
[1] 3 5 8 9
```

```r
a - 1
```

```
[1] 1 3 6 7
```

```r
mean(a)# get the average
```

```
[1] 5.25
```

2.1.2 Character

Character objects are text strings which are bound and created by `""` or `''`. Mathematical functions do not work on them. Characters which contain only numerals can be translated to numerics as above, but will return `NA` if they contain anything other than a numeral. There is a wide range of functions designed to work on strings for a variety of purposes. Objects can be converted to character using `as.character()` or checked using `is.character`.

```
# ~~~ character
b <- c("hello", "goodbye", "goodbye")
b
```

```
[1] "hello"   "goodbye" "goodbye"
```

```
summary(b)
```

```
   Length     Class      Mode
        3 character character
```

```
# doesn't allow for mathematical operations
mean(b)# get the average
```

```
Warning in mean.default(b): argument is not numeric or logical: returning NA
```

```
[1] NA
```

```
# Check the length of each
nchar(b)
```

```
[1] 5 7 7
```

```
# Check if is a character
test # Note the quotation marks
```

```
[1] "2" "4" "7" "8"
```

```r
is.character(test)
```

```
[1] TRUE
```

```r
# Substitution
b_sub = gsub('good', '', b)
b_sub
```

```
[1] "hello" "bye"   "bye"
```

2.1.3 Factors

Factors are categorical variables which are used to divide data into groups. They can be created using `factor()`, translated using `as.factor()`, and checked using `is.factor()`. In many (but not all) cases they behave similarly to characters, but they also have numeric elements. Each group inside a factor is called a level, and is assigned a numeric value, which is shown upon printing the factor.

```r
# ~~~factor
# factors act as categorical variables
c <- as.factor(b)
c
```

```
[1] hello   goodbye goodbye
Levels: goodbye hello
```

```r
# Levels can be set manually if a certain order is desired. They default to alphabetic/numeri
c = factor(b, levels = c('hello', 'goodbye'))
c # Note the levels are different
```

```
[1] hello   goodbye goodbye
Levels: hello goodbye
```

```r
# Arithmetic doesn't work
mean(c)
```

```
Warning in mean.default(c): argument is not numeric or logical: returning NA
```

```
[1] NA
```

```r
# as.numeric outputs the levels
as.numeric(c)
```

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

```r
# Summary
summary(c)
```

```
  hello goodbye
      1       2
```

2.1.4 Logicals

Logicals are true or false. They can be created by various logical tests using Boolean operators such as `==`, `!=`, `>`, `<`, `<=`, and `>=`, or by logical functions such as those associated with `is`, including those listed above. They can be converted using `as.logical` and tested (logically!) using `is.logical`. Logicals can be written as `TRUE` or `FALSE` and `T` or `F` interchangeably. Numerically, logicals are binary, with `TRUE == 1` and `FALSE == 0`. This behaviour can be used to perform arithmetic on logicals.

```r
# ~~~ logical
# true or false variables
d <- a == b
```

```
Warning in a == b: longer object length is not a multiple of shorter object
length
```

```r
d
```

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

```r
# Boolean tests
1 > 2
```

```
[1] FALSE
```

```r
1 < 2
```

```
[1] TRUE
```

```r
summary(d)
```

```
   Mode   FALSE 
logical       4 
```

```r
mean(d)# logical operation converts F = 0, T = 1
```

```
[1] 0
```

```r
sum(d) # RS Note, you can use sum on logicals to get the number of TRUEs
```

```
[1] 0
```

```r
as.logical(1)
```

```
[1] TRUE
```

```r
# T and TRUE and F and FALSE are interchangeable
T == TRUE
```

```
[1] TRUE
```

2.2 Object Types

In addition to the different types of data, R has different types of objects to contain those different types of data:

2.2.1 Scalars

A scalar is a single object, such as a single number, character string, or T/F. There are no functions specifically associated with scalars. They are treated as vectors of length 1.

2.2.2 Vectors

A vector is a one-dimensional sequence of values. Vectors can be of any data type. They are created using `c()`, and can be converted using `as.vector()` and tested using `is.vector()`. The number of elements in a vector can be determined using `length()`. Vectors are indexed using `[]` (see refresher section 4 for more). By default in R, performing a function on a vector applies it to all elements of the vector.

```
# 2.b) Types of objects -------

# different data types can be stored in different objects
# ~~~ Vector: one-dimensional sequence of values

# it can have numbers, characters, etc
num_vector <- c(3,6,9,12,15)
num_vector
```

```
[1]  3  6  9 12 15
```

```
# check the length
length(num_vector)
```

```
[1] 5
```

```
# Indexing
num_vector[1]
```

```
[1] 3
```

```
num_vector[2]
```

```
[1] 6
```

```
# Operations are applied to all elements
num_vector+1
```

```
[1]  4  7 10 13 16
```

```r
is.numeric(num_vector)
```

```
[1] TRUE
```

```r
as.character(num_vector)
```

```
[1] "3"  "6"  "9"  "12" "15"
```

```r
# character vectors
char_vector <- c("Data", "analysis", "fun")
char_vector
```

```
[1] "Data"     "analysis" "fun"
```

```r
char_vector2 <- c("Data", "analysis", 1)
# when we mix character with numbers, everything becomes character
char_vector2
```

```
[1] "Data"     "analysis" "1"
```

```r
#check data class stored within a vector:
data.class(char_vector2)
```

```
[1] "character"
```

```r
data.class(num_vector)
```

```
[1] "numeric"
```

```r
# A scalar is a vector of 1
is.vector(1)
```

```
[1] TRUE
```

2.2.3 Matrices

Matrices are basic two-dimensional data structures. Matrices are almost always numeric. Unlike data frames, matrices can only contain one data type. Matrices can be created using `matrix()`, translated using `as.matrix()`, and tested using `is.matrix()`. Matrices are indexed using `[row,column]` (see refresher section 4 for more).

```
# ~~~ Matrix and data frames: 2 dimensions
# matrices are usually numerical

#calls for data, n rows, n cols
my_matrix1 <-  matrix(1:6, 2, 3, byrow = T)# data, rows, columns
my_matrix2 <-  rbind(num_vector, num_vector) # or 'rbind()', 'cbind()' and others; RS Note, 
my_matrix3 <-  cbind(my_matrix2, my_matrix2) # or 'rbind()', 'cbind()' and others

# Character matrix
matrix(c('a', 'b', 'c', 'd', 'e', 'f'), nrow = 2, ncol = 3)
```

```
     [,1] [,2] [,3]
[1,] "a"  "c"  "e"
[2,] "b"  "d"  "f"
```

```
# Mixing makes everything character
matrix(c('a', 'b', 'c', 'd', 2, 3), nrow = 2, ncol = 3)
```

```
     [,1] [,2] [,3]
[1,] "a"  "c"  "2"
[2,] "b"  "d"  "3"
```

2.2.4 Data Frames

Data frames are the workhorse two-dimensional data structures in R. Think of them like a standard table. Unlike matrices, each column in a data frame can be a different data type. Data frames can be indexed using `[row,column]` or `$column` (see refresher section 4 for more). Data frames are made using `data.frame()`, translated using `as.data.frame()`, and checked using `is.data.frame()`.

```
# ~~~ data.frames can contain numbers, characters or both
my_df <- data.frame(test = char_vector, char_vector2)
my_df
```

```
      test char_vector2
1     Data         Data
2 analysis     analysis
3      fun            1
```

```r
# Column names can be set with names() or colnames()
names(my_df) <- c("name", "order")
my_df
```

```
      name    order
1     Data     Data
2 analysis analysis
3      fun        1
```

```r
# Basic indexing
my_df[,1]
```

```
[1] "Data"     "analysis" "fun"
```

```r
my_df$name
```

```
[1] "Data"     "analysis" "fun"
```

```r
# Translate a matrix
my_matrix1
```

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

```r
as.data.frame(my_matrix1)
```

```
  V1 V2 V3
1  1  2  3
2  4  5  6
```

2.2.5 Lists

Lists are more freeform data structures in R which can be used to contain multiples of anything. Lists tend to be a bit more difficult to work with than other data types. They are indexed using `[[]]`, created using `list()`, translated using `as.list()` and checked using `is.list()`. Technically, data frames are a special type of list.

```
# ~~~ lists are multidimensional objects of anything
# with any dimension
my_list <- list(a, num_vector, char_vector, my_df)
my_list
```

```
[[1]]
[1] 2 4 7 8

[[2]]
[1]  3  6  9 12 15

[[3]]
[1] "Data"     "analysis" "fun"

[[4]]
      name    order
1     Data     Data
2 analysis analysis
3      fun        1
```

```
#lists hold each of the elements in a "slot"
# These slosts can be indexed using [[]]
my_list[[2]]
```

```
[1]  3  6  9 12 15
```

```
my_list[[2]][1] # We can stack square brackets to go deeper into a list
```

```
[1] 3
```

```
# A data frame is a type of list
is.list(my_df)
```

```
[1] TRUE
```

2.2.6 Tibbles

There are a few other types of more niche data structures you may come across, such as arrays and data tables. The most notable of these is the tibble - a special form of data frame which hails from the tidyverse. Tibbles print differently from data frames, and they have some unique properties which don't always play nice with other functions. It's good to know about them, and how to change them back to data frames if you run into such a situation.

Tibbles are usually created by using a `dplyr` or `tidyr` function on a data frame, but can also be created by `tibble()`. They can be translated back to data frames using `as.data.frame()`.

```
# Tibbles look a little different, particularly when printing large tibbles as compared to la
my_tibble = tibble::tibble(my_df)
my_tibble
```

```
# A tibble: 3 x 2
  name     order
  <chr>    <chr>
1 Data     Data
2 analysis analysis
3 fun      1
```

```
# Tibbles can be translated back using as.data.frame if required
as.data.frame(my_tibble)
```

```
      name    order
1     Data     Data
2 analysis analysis
3      fun        1
```

2.3 Useful functions

Below you can see some useful functions for dealing with data and object types. Play around and get to know them.

```
# 2.c) useful functions for object types----
#2 ~~~ figuring data types and structure
class(my_df)
```

```
[1] "data.frame"
```

```r
data.class(my_df) #synonyms to classify data type or object type
```

```
[1] "data.frame"
```

```r
x = 2; class(x)
```

```
[1] "numeric"
```

```r
str(my_df) # see components of an object
```

```
'data.frame':   3 obs. of  2 variables:
 $ name : chr  "Data" "analysis" "fun"
 $ order: chr  "Data" "analysis" "1"
```

```r
summary(my_df) # summarize each of the components of an object
```

```
     name              order
 Length:3           Length:3
 Class :character   Class :character
 Mode  :character   Mode  :character
```

```r
str(my_list)
```

```
List of 4
 $ : num [1:4] 2 4 7 8
 $ : num [1:5] 3 6 9 12 15
 $ : chr [1:3] "Data" "analysis" "fun"
 $ :'data.frame':   3 obs. of  2 variables:
  ..$ name : chr [1:3] "Data" "analysis" "fun"
  ..$ order: chr [1:3] "Data" "analysis" "1"
```

```r
summary(my_list)
```

```
     Length Class      Mode
[1,] 4      -none-     numeric
[2,] 5      -none-     numeric
[3,] 3      -none-     character
[4,] 2      data.frame list
```

```r
object <- c(1:10)

length(object) # number of elements or components
```

```
[1] 10
```

```r
c(object,object) # combine objects into a vector
```

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

```r
cbind(object, object) # combine objects as columns into an array (NOT A DATA FRAME UNLESS ONE
```

```
      object object
 [1,]      1      1
 [2,]      2      2
 [3,]      3      3
 [4,]      4      4
 [5,]      5      5
 [6,]      6      6
 [7,]      7      7
 [8,]      8      8
 [9,]      9      9
[10,]     10     10
```

```r
rbind(object, object) # combine objects as rows into an array (NOT A DATA FRAME UNLESS ONE O
```

```
       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
object    1    2    3    4    5    6    7    8    9    10
object    1    2    3    4    5    6    7    8    9    10
```

```r
object     # prints the object
```

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

```r
rm(object) #removes object from environment
object
```

```
Error: object 'object' not found
```

3 R Refresher Part 3: Importing Data in R

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers how to input data files into R, and some basic functions for looking at data

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

3.1 Importing .csv files

Most files you'll likely load into R will be `.csv` files. These files can be easily opened and translated in Excel (BUT DON'T EDIT THEM IN EXCEL!). `.csv` files are comma delimited files, meaning if you open the file in the text editor, you'll see each row is a string of characters with each column separated by a comma. Because of this structure, `.csv` files don't play nice with commas inside data fields, as they will cause the field to split into multiple columns. This can happen if you have something like a comments section in your data, where you might use a comma for punctuation when writing a comment. Keep this in mind as it can cause you headaches if you're not careful.

`.csv` files can be read into r using `read.csv()` and written into files using `write.csv()`. By default, `write.csv()` will output the row names of a data frame or matrix, which is generally not desired. You can use the `row.names = F` argument to prevent this.

`head()`, `View()`, `colnames()`, `nrow()`, `ncol()`, and `dim()` are useful functions for looking at a dataset once you've loaded it in.

```
# 3. Importing data in R -----------------------------------

# 3.1 Let's import an external data file "Schoenemann".
# First, make sure the files are placed in your directory folder
# directory

getwd()
```

```
[1] "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-Assignment-
Guide"
```

```r
dir()
```

```
 [1] "_book"                          "_freeze"
 [3] "_quarto.yml"                    "a1a.qmd"
 [5] "a1b.qmd"                        "a1c.qmd"
 [7] "a1d.qmd"                        "a1e.qmd"
 [9] "Assignment_Helpers.Rproj"       "cover.png"
[11] "cuse.csv"                       "data1c.txt"
[13] "docs"                           "fishcatch.csv"
[15] "FOME_Primary_BlueGradient.png" "FOME_Small.png"
[17] "guidelines.html"                "guidelines.qmd"
[19] "index.html"                     "index.qmd"
[21] "monkey.csv"                     "refresher_1.html"
[23] "refresher_1.qmd"                "refresher_2.html"
[25] "refresher_2.qmd"                "refresher_3_files"
[27] "refresher_3.html"               "refresher_3.qmd"
[29] "refresher_3.rmarkdown"          "refresher_4.qmd"
[31] "refresher_5.qmd"                "refresher_6.qmd"
[33] "refresher_7.qmd"                "refresher_8.qmd"
[35] "refresher_9.qmd"                "renv"
[37] "renv.lock"                      "Schoenemann.csv"
[39] "site_libs"                      "snake.csv"
[41] "Workspace.RData"
```

```r
# From a Comma Delimited Text File (.csv)
# first row contains variable names, comma is separator
scho <- read.csv("Schoenemann.csv", header=T)
head(scho) #first 6 rows of a data frame
```

```
      Order     Family   Genus    Species Location   Mass    Fat   FFWT    CNS
1 Carnivora    Felidae   Felis canadensis   Alaska 7688.0 1120.0 6568.0 105.09
2 Carnivora    Felidae   Felis      rufus Virginia 6152.0  738.0 5414.0  81.75
3 Carnivora Mustelidae    Gulo     luscus   Alaska 9362.0  562.0 8800.0  85.36
4 Carnivora Mustelidae Mustela    erminea   Alaska  183.3    3.1  180.2   6.69
5 Carnivora Mustelidae Mustela      vison Virginia 1032.0   66.0  966.0  18.06
6 Carnivora Proyonidae Procyon      lotor Virginia 6040.0 1013.0 5027.0  58.31
  HEART  MUSCLE   BONE
1 27.59 4341.45 631.18
```

```
2 25.45 3600.31 552.23
3 80.96 5271.20 879.12
4  1.87  104.70  21.98
5  7.63  581.53  80.27
6 36.19 2920.69 517.78
```

```r
head(scho$Order) # RS Note: head also works on other structures
```

```
[1] "Carnivora" "Carnivora" "Carnivora" "Carnivora" "Carnivora" "Carnivora"
```

```r
scho
```

```
         Order           Family         Genus        Species Location    Mass
1    Carnivora          Felidae         Felis     canadensis   Alaska 7688.00
2    Carnivora          Felidae         Felis          rufus Virginia 6152.00
3    Carnivora       Mustelidae          Gulo         luscus   Alaska 9362.00
4    Carnivora       Mustelidae       Mustela        erminea   Alaska  183.30
5    Carnivora       Mustelidae       Mustela          vison Virginia 1032.00
6    Carnivora       Proyonidae       Procyon          lotor Virginia 6040.00
7   Chiroptera       Molossidae      Molossus          major   Brazil   11.07
8   Chiroptera   Phyllostomidae      Artibeus    jamaicensis   Brazil   40.47
9   Chiroptera   Phyllostomidae      Artibeus      lituratus   Brazil   63.65
10  Chiroptera   Phyllostomidae   Glossophaga       soricina   Brazil    7.22
11  Chiroptera   Phyllostomidae  Phyllostomus       discolor   Brazil   34.37
12  Chiroptera   Phyllostomidae  Phyllostomus       hastatus   Brazil   92.26
13  Chiroptera   Phyllostomidae      Sturnira         lilium   Brazil   15.39
14  Chiroptera   Phyllostomidae     Vampyrops       lineatus   Brazil   22.03
15  Chiroptera Vespertilionidae     Eptesicus         fuscus Virginia   17.88
16    Edentata      Dasypodidae    Euphractos     sexcinctus   Brazil 2459.00
17 Insectivora         Talpidae      Scalopus      aquaticus Virginia   44.64
18  Lagomorpha      Ochotonidae      Ochotona       collaris   Alaska  120.90
19 Marsupialia     Didelphiidae     Didelphis    marsupialis Virginia 1411.00
20    Primates  Calllitrichidae    Callithrix        jacchus   Brazil  186.00
21    Rodentia       Castoridae        Castor     canadensis Virginia 9331.00
22    Rodentia       Cricetidae Clethrionomys        gapperi Virginia   18.34
23    Rodentia       Cricetidae Clethrionomys        rutilus   Alaska   25.27
24    Rodentia       Cricetidae        Lemmus  trimucronatus   Alaska   41.62
25    Rodentia       Cricetidae      Microtus pennsylvanicus Virginia   31.38
26    Rodentia       Cricetidae      Microtus      oeconomus   Alaska   24.83
27    Rodentia       Cricetidae      Microtus      pinetorum Virginia   19.41
28    Rodentia       Cricetidae       Ondatra      zibethica Virginia 1180.00
```

29	Rodentia	Cricetidae	Oryzomys	palustris	Virginia	61.62
30	Rodentia	Cricetidae	Peromyscus	leucopus	Virginia	16.99
31	Rodentia	Cuniculidae	Cuniculus	paca	Brazil	1565.00
32	Rodentia	Dasyproctidae	Dasyprocta	aguti	Brazil	2097.00
33	Rodentia	Erethizontidae	Erethizon	dorsatum	Virginia	5339.00
34	Rodentia	Muridae	Mus	musculus	Virginia	15.88
35	Rodentia	Sciuridae	Citellus	undulatus	Alaska	479.00
36	Rodentia	Sciuridae	Marmota	caligata	Alaska	3558.00
37	Rodentia	Sciuridae	Marmota	monax	Alaska	2194.00
38	Rodentia	Sciuridae	Sciurus	carolinensis	Virginia	499.00
39	Rodentia	Sciuridae	Tamiasciurus	hudsonicus	Alaska	192.80

	Fat	FFWT	CNS	HEART	MUSCLE	BONE
1	1120.00	6568.00	105.09	27.59	4341.45	631.18
2	738.00	5414.00	81.75	25.45	3600.31	552.23
3	562.00	8800.00	85.36	80.96	5271.20	879.12
4	3.10	180.20	6.69	1.87	104.70	21.98
5	66.00	966.00	18.06	7.63	581.53	80.27
6	1013.00	5027.00	58.31	36.19	2920.69	517.78
7	0.22	10.89	0.35	0.15	5.51	1.36
8	3.79	36.18	0.96	0.47	18.02	4.48
9	6.22	57.19	1.21	0.74	29.05	8.09
10	0.25	7.15	0.37	0.10	3.86	0.69
11	2.38	32.20	1.00	0.36	16.49	3.87
12	5.41	87.05	2.10	0.89	47.01	11.75
13	1.21	14.25	0.62	0.16	6.33	2.04
14	1.59	20.24	0.76	0.24	10.59	2.23
15	1.51	16.37	0.32	0.19	7.43	2.26
16	252.20	2123.00	19.32	12.95	864.06	269.20
17	1.23	43.41	1.01	0.34	21.88	5.30
18	7.00	113.90	3.06	0.73	57.18	11.32
19	107.00	1304.00	7.56	7.56	681.99	203.42
20	8.70	176.20	7.56	1.22	87.92	26.69
21	865.00	8466.00	53.34	27.94	4622.44	897.40
22	0.14	18.20	0.64	0.13	9.25	2.22
23	0.72	24.55	0.60	0.19	11.34	1.94
24	0.75	40.87	1.03	0.28	19.94	3.82
25	1.20	30.18	0.76	0.26	14.46	2.59
26	0.45	24.38	0.67	0.19	11.12	2.44
27	0.45	18.96	0.57	0.15	9.46	1.86
28	86.00	1094.00	7.11	3.50	679.37	115.96
29	7.88	53.74	1.11	0.34	26.92	5.33
30	0.59	16.40	0.61	0.17	8.02	1.49
31	196.50	1368.00	29.00	7.80	737.35	140.90

```
32  263.40 1833.80  25.86 13.94 1115.13 168.53
33  674.00 4725.00  37.80 24.10 2197.13 576.45
34    0.96   14.92   0.48  0.15    7.07   1.22
35   21.00  458.00   6.00  2.56  257.85  38.01
36  749.00 2809.00  20.37 16.57 1671.36 257.02
37  536.50 1657.50  12.73  8.11  817.13 149.49
38   11.00  488.00   8.88  2.83  306.46  51.73
39    3.80  189.00   5.50  1.68  114.16  18.16
```

```r
# View(scho)
colnames(scho)
```

```
 [1] "Order"    "Family"   "Genus"    "Species"  "Location" "Mass"
 [7] "Fat"      "FFWT"     "CNS"      "HEART"    "MUSCLE"   "BONE"
```

```r
nrow(scho)
```

```
[1] 39
```

```r
ncol(scho)
```

```
[1] 12
```

```r
dim(scho)
```

```
[1] 39 12
```

3.2 Importing other files

`.csv` files are just one of many types of data files you may need to load into R. They are also only one type of delimited file you may need to load in. Other types of delimited data can be loaded in using the more general `read.table()` function, which allows you to specify the delimting character through the `sep` argument. Setting `sep = ','` is essentially equivalent to `read.csv()`. Other common uses include `sep = '\t'` to load in tab (space) delimited files, and `sep = ';'` to load in semicolon delimited files.

```r
# From a Tab delimited text file (.txt)
scho_txt <- read.table("Schoenemann.csv", header=TRUE, sep = ',')
scho_txt
```

```
         Order           Family         Genus        Species Location    Mass
1    Carnivora          Felidae         Felis     canadensis   Alaska 7688.00
2    Carnivora          Felidae         Felis          rufus Virginia 6152.00
3    Carnivora       Mustelidae          Gulo         luscus   Alaska 9362.00
4    Carnivora       Mustelidae       Mustela        erminea   Alaska  183.30
5    Carnivora       Mustelidae       Mustela          vison Virginia 1032.00
6    Carnivora       Proyonidae       Procyon          lotor Virginia 6040.00
7   Chiroptera       Molossidae      Molossus          major   Brazil   11.07
8   Chiroptera   Phyllostomidae      Artibeus    jamaicensis   Brazil   40.47
9   Chiroptera   Phyllostomidae      Artibeus      lituratus   Brazil   63.65
10  Chiroptera   Phyllostomidae   Glossophaga       soricina   Brazil    7.22
11  Chiroptera   Phyllostomidae  Phyllostomus       discolor   Brazil   34.37
12  Chiroptera   Phyllostomidae  Phyllostomus       hastatus   Brazil   92.26
13  Chiroptera   Phyllostomidae      Sturnira         lilium   Brazil   15.39
14  Chiroptera   Phyllostomidae     Vampyrops       lineatus   Brazil   22.03
15  Chiroptera Vespertilionidae     Eptesicus         fuscus Virginia   17.88
16    Edentata      Dasypodidae    Euphractos     sexcinctus   Brazil 2459.00
17 Insectivora         Talpidae      Scalopus      aquaticus Virginia   44.64
18  Lagomorpha      Ochotonidae      Ochotona       collaris   Alaska  120.90
19 Marsupialia     Didelphiidae     Didelphis    marsupialis Virginia 1411.00
20    Primates  Calllitrichidae    Callithrix        jacchus   Brazil  186.00
21    Rodentia       Castoridae        Castor     canadensis Virginia 9331.00
22    Rodentia       Cricetidae Clethrionomys        gapperi Virginia   18.34
23    Rodentia       Cricetidae Clethrionomys        rutilus   Alaska   25.27
24    Rodentia       Cricetidae        Lemmus  trimucronatus   Alaska   41.62
25    Rodentia       Cricetidae      Microtus pennsylvanicus Virginia   31.38
26    Rodentia       Cricetidae      Microtus      oeconomus   Alaska   24.83
27    Rodentia       Cricetidae      Microtus      pinetorum Virginia   19.41
28    Rodentia       Cricetidae       Ondatra      zibethica Virginia 1180.00
29    Rodentia       Cricetidae      Oryzomys      palustris Virginia   61.62
30    Rodentia       Cricetidae    Peromyscus       leucopus Virginia   16.99
31    Rodentia      Cuniculidae     Cuniculus           paca   Brazil 1565.00
32    Rodentia    Dasyproctidae    Dasyprocta          aguti   Brazil 2097.00
33    Rodentia   Erethizontidae     Erethizon       dorsatum Virginia 5339.00
34    Rodentia          Muridae           Mus       musculus Virginia   15.88
35    Rodentia        Sciuridae      Citellus      undulatus   Alaska  479.00
36    Rodentia        Sciuridae       Marmota       caligata   Alaska 3558.00
37    Rodentia        Sciuridae       Marmota          monax   Alaska 2194.00
```

38	Rodentia		Sciuridae		Sciurus	carolinensis	Virginia	499.00	
39	Rodentia		Sciuridae		Tamiasciurus	hudsonicus	Alaska	192.80	
	Fat	FFWT	CNS	HEART	MUSCLE	BONE			
1	1120.00	6568.00	105.09	27.59	4341.45	631.18			
2	738.00	5414.00	81.75	25.45	3600.31	552.23			
3	562.00	8800.00	85.36	80.96	5271.20	879.12			
4	3.10	180.20	6.69	1.87	104.70	21.98			
5	66.00	966.00	18.06	7.63	581.53	80.27			
6	1013.00	5027.00	58.31	36.19	2920.69	517.78			
7	0.22	10.89	0.35	0.15	5.51	1.36			
8	3.79	36.18	0.96	0.47	18.02	4.48			
9	6.22	57.19	1.21	0.74	29.05	8.09			
10	0.25	7.15	0.37	0.10	3.86	0.69			
11	2.38	32.20	1.00	0.36	16.49	3.87			
12	5.41	87.05	2.10	0.89	47.01	11.75			
13	1.21	14.25	0.62	0.16	6.33	2.04			
14	1.59	20.24	0.76	0.24	10.59	2.23			
15	1.51	16.37	0.32	0.19	7.43	2.26			
16	252.20	2123.00	19.32	12.95	864.06	269.20			
17	1.23	43.41	1.01	0.34	21.88	5.30			
18	7.00	113.90	3.06	0.73	57.18	11.32			
19	107.00	1304.00	7.56	7.56	681.99	203.42			
20	8.70	176.20	7.56	1.22	87.92	26.69			
21	865.00	8466.00	53.34	27.94	4622.44	897.40			
22	0.14	18.20	0.64	0.13	9.25	2.22			
23	0.72	24.55	0.60	0.19	11.34	1.94			
24	0.75	40.87	1.03	0.28	19.94	3.82			
25	1.20	30.18	0.76	0.26	14.46	2.59			
26	0.45	24.38	0.67	0.19	11.12	2.44			
27	0.45	18.96	0.57	0.15	9.46	1.86			
28	86.00	1094.00	7.11	3.50	679.37	115.96			
29	7.88	53.74	1.11	0.34	26.92	5.33			
30	0.59	16.40	0.61	0.17	8.02	1.49			
31	196.50	1368.00	29.00	7.80	737.35	140.90			
32	263.40	1833.80	25.86	13.94	1115.13	168.53			
33	674.00	4725.00	37.80	24.10	2197.13	576.45			
34	0.96	14.92	0.48	0.15	7.07	1.22			
35	21.00	458.00	6.00	2.56	257.85	38.01			
36	749.00	2809.00	20.37	16.57	1671.36	257.02			
37	536.50	1657.50	12.73	8.11	817.13	149.49			
38	11.00	488.00	8.88	2.83	306.46	51.73			
39	3.80	189.00	5.50	1.68	114.16	18.16			

```r
# 3.2 Make sure your data is correct
# Some functions that help us to check if the data was
# input correctly: str(); dim(); head(); tail()

str(scho) # internal structure of the data
```

```
'data.frame':   39 obs. of  12 variables:
 $ Order   : chr  "Carnivora" "Carnivora" "Carnivora" "Carnivora" ...
 $ Family  : chr  "Felidae" "Felidae" "Mustelidae" "Mustelidae" ...
 $ Genus   : chr  "Felis" "Felis" "Gulo" "Mustela" ...
 $ Species : chr  "canadensis" "rufus" "luscus" "erminea" ...
 $ Location: chr  "Alaska" "Virginia" "Alaska" "Alaska" ...
 $ Mass    : num  7688 6152 9362 183 1032 ...
 $ Fat     : num  1120 738 562 3.1 66 ...
 $ FFWT    : num  6568 5414 8800 180 966 ...
 $ CNS     : num  105.09 81.75 85.36 6.69 18.06 ...
 $ HEART   : num  27.59 25.45 80.96 1.87 7.63 ...
 $ MUSCLE  : num  4341 3600 5271 105 582 ...
 $ BONE    : num  631.2 552.2 879.1 22 80.3 ...
```

```r
dim(scho) # dimensions (numbers of row and column)
```

```
[1] 39 12
```

```r
head(scho) # column names and first rows (are the column names correct?)
```

```
      Order     Family   Genus    Species Location   Mass    Fat   FFWT    CNS
1 Carnivora    Felidae   Felis canadensis   Alaska 7688.0 1120.0 6568.0 105.09
2 Carnivora    Felidae   Felis      rufus Virginia 6152.0  738.0 5414.0  81.75
3 Carnivora Mustelidae    Gulo     luscus   Alaska 9362.0  562.0 8800.0  85.36
4 Carnivora Mustelidae Mustela    erminea   Alaska  183.3    3.1  180.2   6.69
5 Carnivora Mustelidae Mustela      vison Virginia 1032.0   66.0  966.0  18.06
6 Carnivora Proyonidae Procyon      lotor Virginia 6040.0 1013.0 5027.0  58.31
  HEART  MUSCLE   BONE
1 27.59 4341.45 631.18
2 25.45 3600.31 552.23
3 80.96 5271.20 879.12
4  1.87  104.70  21.98
5  7.63  581.53  80.27
6 36.19 2920.69 517.78
```

```r
tail(scho) # last rows
```

```
      Order    Family        Genus      Species Location    Mass    Fat    FFWT
34 Rodentia   Muridae          Mus     musculus Virginia   15.88   0.96   14.92
35 Rodentia Sciuridae     Citellus    undulatus   Alaska  479.00  21.00  458.00
36 Rodentia Sciuridae      Marmota     caligata   Alaska 3558.00 749.00 2809.00
37 Rodentia Sciuridae      Marmota        monax   Alaska 2194.00 536.50 1657.50
38 Rodentia Sciuridae      Sciurus carolinensis Virginia  499.00  11.00  488.00
39 Rodentia Sciuridae Tamiasciurus   hudsonicus   Alaska  192.80   3.80  189.00
     CNS HEART  MUSCLE   BONE
34  0.48  0.15    7.07   1.22
35  6.00  2.56  257.85  38.01
36 20.37 16.57 1671.36 257.02
37 12.73  8.11  817.13 149.49
38  8.88  2.83  306.46  51.73
39  5.50  1.68  114.16  18.16
```

```r
summary(scho)
```

```
   Order              Family             Genus             Species
 Length:39          Length:39          Length:39          Length:39
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character


  Location              Mass              Fat                FFWT
 Length:39          Min.   :   7.22   Min.   :   0.140   Min.   :   7.15
 Class :character   1st Qu.:  25.05   1st Qu.:   1.205   1st Qu.:  24.46
 Mode  :character   Median : 120.90   Median :   6.220   Median : 113.90
                    Mean   :1581.37   Mean   : 187.696   Mean   :1393.02
                    3rd Qu.:1831.00   3rd Qu.: 224.350   3rd Qu.:1512.75
                    Max.   :9362.00   Max.   :1120.000   Max.   :8800.00
     CNS              HEART             MUSCLE              BONE
 Min.   :  0.320   Min.   : 0.100   Min.   :   3.86   Min.   :  0.69
 1st Qu.:  0.715   1st Qu.: 0.215   1st Qu.:  11.23   1st Qu.:  2.35
 Median :  3.060   Median : 0.890   Median :  57.18   Median : 11.75
 Mean   : 15.757   Mean   : 8.120   Mean   : 802.90   Mean   :145.43
 3rd Qu.: 18.690   3rd Qu.: 7.955   3rd Qu.: 777.24   3rd Qu.:159.01
 Max.   :105.090   Max.   :80.960   Max.   :5271.20   Max.   :897.40
```

```r
# # ~~~ saving data
# write.csv(scho, "Datasets/Schoenemann_edited.csv", row.names = F) # RS Note: You almost alw

getwd()
```

```
[1] "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-Assignment-
Guide"
```

```r
dir()
```

```
 [1] "_book"                          "_freeze"
 [3] "_quarto.yml"                    "a1a.qmd"
 [5] "a1b.qmd"                        "a1c.qmd"
 [7] "a1d.qmd"                        "a1e.qmd"
 [9] "Assignment_Helpers.Rproj"       "cover.png"
[11] "cuse.csv"                       "data1c.txt"
[13] "docs"                           "fishcatch.csv"
[15] "FOME_Primary_BlueGradient.png" "FOME_Small.png"
[17] "guidelines.html"                "guidelines.qmd"
[19] "index.html"                     "index.qmd"
[21] "monkey.csv"                     "refresher_1.html"
[23] "refresher_1.qmd"                "refresher_2.html"
[25] "refresher_2.qmd"                "refresher_3_files"
[27] "refresher_3.html"               "refresher_3.qmd"
[29] "refresher_3.rmarkdown"          "refresher_4.qmd"
[31] "refresher_5.qmd"                "refresher_6.qmd"
[33] "refresher_7.qmd"                "refresher_8.qmd"
[35] "refresher_9.qmd"                "renv"
[37] "renv.lock"                      "Schoenemann.csv"
[39] "site_libs"                      "snake.csv"
[41] "Workspace.RData"
```

4 R Refresher Part 4: Indexing

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers indexing, which are methods by which you can access subsets of a data structure. Indexing serves as key fundamental of coding, which is incredibly useful in a wide range of situations.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

4.1 Square Brackets

Most indexing in R is performed through the use of square brackets. as vectors are one-dimensional, they are indexed using a scalar value inside `[]`. Two-dimensional data structures are indexed via `[row,column]`. Two-dimensional data structures can also be indexed using just `[]`, in which they will go in order of individual elements, but this is more commonly done by mistake than to achieve as a desired outcome. Lists are indexed using `[[]]`. Square brackets can also be stacked together - for instance, `data_frame[1,][2]` would pull out the second element of the first column of the object `data_frame`. This is equivalent to `data_frame[1,2]`.

```
# 4. Accessing bits of data: indexing ----------------------------------------

# Ok, now you have your objects, your own data in R.
# How do you access them?
# First thing: make sure they exist...
a <- c(2,4,7,8)
num_vector <- c(3,6,9,12,15)
my_matrix1 <-  matrix(1:6, 2, 3, byrow = T)# data, rows, columns
char_vector <- c("Data", "analysis", "fun")
char_vector2 <- c("Data", "analysis", 1)
my_df <- data.frame(test = char_vector, char_vector2)
my_list <- list(a, num_vector, char_vector, my_df)
scho <- read.csv("Schoenemann.csv", header=T)

num_vector
```

```
[1]  3  6  9 12 15
```

```r
my_matrix1
```

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

```r
my_list
```

```
[[1]]
[1] 2 4 7 8

[[2]]
[1]  3  6  9 12 15

[[3]]
[1] "Data"     "analysis" "fun"

[[4]]
      test char_vector2
1     Data         Data
2 analysis     analysis
3      fun            1
```

```r
scho
```

```
         Order           Family         Genus        Species Location    Mass
1    Carnivora          Felidae         Felis     canadensis   Alaska 7688.00
2    Carnivora          Felidae         Felis          rufus Virginia 6152.00
3    Carnivora       Mustelidae          Gulo         luscus   Alaska 9362.00
4    Carnivora       Mustelidae       Mustela        erminea   Alaska  183.30
5    Carnivora       Mustelidae       Mustela          vison Virginia 1032.00
6    Carnivora       Proyonidae       Procyon          lotor Virginia 6040.00
7   Chiroptera       Molossidae      Molossus          major   Brazil   11.07
8   Chiroptera   Phyllostomidae      Artibeus    jamaicensis   Brazil   40.47
9   Chiroptera   Phyllostomidae      Artibeus      lituratus   Brazil   63.65
10  Chiroptera   Phyllostomidae   Glossophaga       soricina   Brazil    7.22
11  Chiroptera   Phyllostomidae  Phyllostomus       discolor   Brazil   34.37
12  Chiroptera   Phyllostomidae  Phyllostomus       hastatus   Brazil   92.26
```

13	Chiroptera	Phyllostomidae	Sturnira	lilium	Brazil	15.39
14	Chiroptera	Phyllostomidae	Vampyrops	lineatus	Brazil	22.03
15	Chiroptera	Vespertilionidae	Eptesicus	fuscus	Virginia	17.88
16	Edentata	Dasypodidae	Euphractos	sexcinctus	Brazil	2459.00
17	Insectivora	Talpidae	Scalopus	aquaticus	Virginia	44.64
18	Lagomorpha	Ochotonidae	Ochotona	collaris	Alaska	120.90
19	Marsupialia	Didelphiidae	Didelphis	marsupialis	Virginia	1411.00
20	Primates	Calllitrichidae	Callithrix	jacchus	Brazil	186.00
21	Rodentia	Castoridae	Castor	canadensis	Virginia	9331.00
22	Rodentia	Cricetidae	Clethrionomys	gapperi	Virginia	18.34
23	Rodentia	Cricetidae	Clethrionomys	rutilus	Alaska	25.27
24	Rodentia	Cricetidae	Lemmus	trimucronatus	Alaska	41.62
25	Rodentia	Cricetidae	Microtus	pennsylvanicus	Virginia	31.38
26	Rodentia	Cricetidae	Microtus	oeconomus	Alaska	24.83
27	Rodentia	Cricetidae	Microtus	pinetorum	Virginia	19.41
28	Rodentia	Cricetidae	Ondatra	zibethica	Virginia	1180.00
29	Rodentia	Cricetidae	Oryzomys	palustris	Virginia	61.62
30	Rodentia	Cricetidae	Peromyscus	leucopus	Virginia	16.99
31	Rodentia	Cuniculidae	Cuniculus	paca	Brazil	1565.00
32	Rodentia	Dasyproctidae	Dasyprocta	aguti	Brazil	2097.00
33	Rodentia	Erethizontidae	Erethizon	dorsatum	Virginia	5339.00
34	Rodentia	Muridae	Mus	musculus	Virginia	15.88
35	Rodentia	Sciuridae	Citellus	undulatus	Alaska	479.00
36	Rodentia	Sciuridae	Marmota	caligata	Alaska	3558.00
37	Rodentia	Sciuridae	Marmota	monax	Alaska	2194.00
38	Rodentia	Sciuridae	Sciurus	carolinensis	Virginia	499.00
39	Rodentia	Sciuridae	Tamiasciurus	hudsonicus	Alaska	192.80

	Fat	FFWT	CNS	HEART	MUSCLE	BONE
1	1120.00	6568.00	105.09	27.59	4341.45	631.18
2	738.00	5414.00	81.75	25.45	3600.31	552.23
3	562.00	8800.00	85.36	80.96	5271.20	879.12
4	3.10	180.20	6.69	1.87	104.70	21.98
5	66.00	966.00	18.06	7.63	581.53	80.27
6	1013.00	5027.00	58.31	36.19	2920.69	517.78
7	0.22	10.89	0.35	0.15	5.51	1.36
8	3.79	36.18	0.96	0.47	18.02	4.48
9	6.22	57.19	1.21	0.74	29.05	8.09
10	0.25	7.15	0.37	0.10	3.86	0.69
11	2.38	32.20	1.00	0.36	16.49	3.87
12	5.41	87.05	2.10	0.89	47.01	11.75
13	1.21	14.25	0.62	0.16	6.33	2.04
14	1.59	20.24	0.76	0.24	10.59	2.23
15	1.51	16.37	0.32	0.19	7.43	2.26

```
16  252.20 2123.00  19.32 12.95  864.06 269.20
17    1.23   43.41   1.01  0.34   21.88   5.30
18    7.00  113.90   3.06  0.73   57.18  11.32
19  107.00 1304.00   7.56  7.56  681.99 203.42
20    8.70  176.20   7.56  1.22   87.92  26.69
21  865.00 8466.00  53.34 27.94 4622.44 897.40
22    0.14   18.20   0.64  0.13    9.25   2.22
23    0.72   24.55   0.60  0.19   11.34   1.94
24    0.75   40.87   1.03  0.28   19.94   3.82
25    1.20   30.18   0.76  0.26   14.46   2.59
26    0.45   24.38   0.67  0.19   11.12   2.44
27    0.45   18.96   0.57  0.15    9.46   1.86
28   86.00 1094.00   7.11  3.50  679.37 115.96
29    7.88   53.74   1.11  0.34   26.92   5.33
30    0.59   16.40   0.61  0.17    8.02   1.49
31  196.50 1368.00  29.00  7.80  737.35 140.90
32  263.40 1833.80  25.86 13.94 1115.13 168.53
33  674.00 4725.00  37.80 24.10 2197.13 576.45
34    0.96   14.92   0.48  0.15    7.07   1.22
35   21.00  458.00   6.00  2.56  257.85  38.01
36  749.00 2809.00  20.37 16.57 1671.36 257.02
37  536.50 1657.50  12.73  8.11  817.13 149.49
38   11.00  488.00   8.88  2.83  306.46  51.73
39    3.80  189.00   5.50  1.68  114.16  18.16
```

```r
# Each type of object has a specific way to manipulating
# its values.
# Let's start with vectors: square brackets []
num_vector
```

```
[1]  3  6  9 12 15
```

```r
num_vector[1] # the first element
```

```
[1] 3
```

```r
num_vector[5] # the 5th element
```

```
[1] 15
```

```r
num_vector[8] # there's only five, right?
```

```
[1] NA
```

```r
# you can change specific entries of your vector:
num_vector[1] <- 6

num_vector[2] <- NA

num_vector
```

```
[1]  6 NA  9 12 15
```

```r
# Matrices: square brackets and commas [,] i.e. [row, column]
my_matrix1
```

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

```r
my_matrix1[1,1] # first cell, row 1, column 1
```

```
[1] 1
```

```r
my_matrix1[1,] # first row
```

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

```r
my_matrix1[,1] # first column
```

```
[1] 1 4
```

```r
my_matrix1[5,5] # out of bounds!
```

```
Error in my_matrix1[5, 5]: subscript out of bounds
```

```r
# Lists: double square brackets [[]]
my_list
```

```
[[1]]
[1] 2 4 7 8

[[2]]
[1]  3  6  9 12 15

[[3]]
[1] "Data"     "analysis" "fun"

[[4]]
      test char_vector2
1     Data         Data
2 analysis     analysis
3      fun            1
```

```r
my_list[[1]] # that's our x
```

```
[1] 2 4 7 8
```

```r
my_list[[2]] # that's our num_vector
```

```
[1]  3  6  9 12 15
```

```r
my_list[[3]] # that's our matrix1
```

```
[1] "Data"     "analysis" "fun"
```

```r
my_list[[2]][2] # that's the second element of the vector
```

```
[1] 6
```

```r
my_list[[4]][,2] # that's the third column of the dataframe ...
```

```
[1] "Data"     "analysis" "1"
```

4.2 Dollar Sign

The `$` operator in R divides objects into their component parts, and can be used to pull out a desired part of an object using its name. `$` indexes column names for 2 dimensional data structures, and pulls out individual named items of lists. `attach()` can be used to move the components of an object into the workspace, which can later be undone using `detach()`. See the example below for its usage.

```
# data frame: dollar sign $ to access the columns, typing the column names
scho$Mass
```

```
 [1] 7688.00 6152.00 9362.00  183.30 1032.00 6040.00   11.07   40.47   63.65
[10]    7.22   34.37   92.26   15.39   22.03   17.88 2459.00   44.64  120.90
[19] 1411.00  186.00 9331.00   18.34   25.27   41.62   31.38   24.83   19.41
[28] 1180.00   61.62   16.99 1565.00 2097.00 5339.00   15.88  479.00 3558.00
[37] 2194.00  499.00  192.80
```

```
# or...you can use the functions 'attach()' and 'detach()':
attach(scho) # scho was copied to the workspace; the column names became objects and we can s
Mass
```

```
 [1] 7688.00 6152.00 9362.00  183.30 1032.00 6040.00   11.07   40.47   63.65
[10]    7.22   34.37   92.26   15.39   22.03   17.88 2459.00   44.64  120.90
[19] 1411.00  186.00 9331.00   18.34   25.27   41.62   31.38   24.83   19.41
[28] 1180.00   61.62   16.99 1565.00 2097.00 5339.00   15.88  479.00 3558.00
[37] 2194.00  499.00  192.80
```

```
# but remember to 'detach' your dataframe when your done:
detach(scho)
Mass # see? an error, R doesn't recognize it anymore
```

```
Error: object 'Mass' not found
```

```
# you can also call single datapoints of a dataframe:
scho$Mass[5] # gives you the 5th mass entry
```

```
[1] 1032
```

```r
scho$Mass[5] <- NA # assign a new value to that entry
scho$Mass
```

```
 [1] 7688.00 6152.00 9362.00  183.30      NA 6040.00   11.07   40.47   63.65
[10]    7.22   34.37   92.26   15.39   22.03   17.88 2459.00   44.64  120.90
[19] 1411.00  186.00 9331.00   18.34   25.27   41.62   31.38   24.83   19.41
[28] 1180.00   61.62   16.99 1565.00 2097.00 5339.00   15.88  479.00 3558.00
[37] 2194.00  499.00  192.80
```

```r
# RS Note: You can also use $ to create new columns
scho$Potato = 'Potato'
scho$Potato
```

```
 [1] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
 [9] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
[17] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
[25] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
[33] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
```

4.3 Indexing methods

There are many, many, MANY different ways to index things in R, using numerical, categorical, character, and logical methods. There are also various functions designed specifically for indexing, such as `which()`. Take a look through all the examples below and think about what types of indexing feel the best for you, when you might use them, and what you might use them for.

```r
scho[5,] #gives you the 5th row
```

```
      Order     Family   Genus Species Location Mass Fat FFWT   CNS HEART
5 Carnivora Mustelidae Mustela   vison Virginia   NA  66  966 18.06  7.63
  MUSCLE  BONE Potato
5 581.53 80.27 Potato
```

```r
scho[,5] #gives you the 5th column
```

```
 [1] "Alaska"   "Virginia" "Alaska"   "Alaska"   "Virginia" "Virginia"
 [7] "Brazil"   "Brazil"   "Brazil"   "Brazil"   "Brazil"   "Brazil"
[13] "Brazil"   "Brazil"   "Virginia" "Brazil"   "Virginia" "Alaska"
[19] "Virginia" "Brazil"   "Virginia" "Virginia" "Alaska"   "Alaska"
[25] "Virginia" "Alaska"   "Virginia" "Virginia" "Virginia" "Virginia"
[31] "Brazil"   "Brazil"   "Virginia" "Virginia" "Alaska"   "Alaska"
[37] "Alaska"   "Virginia" "Alaska"
```

```r
# RS Note: You can also index things you don't want instead of things you do want using != an
scho[,-5]
```

```
         Order           Family         Genus        Species    Mass     Fat
1    Carnivora          Felidae         Felis     canadensis 7688.00 1120.00
2    Carnivora          Felidae         Felis          rufus 6152.00  738.00
3    Carnivora       Mustelidae          Gulo         luscus 9362.00  562.00
4    Carnivora       Mustelidae       Mustela        erminea  183.30    3.10
5    Carnivora       Mustelidae       Mustela          vison      NA   66.00
6    Carnivora       Proyonidae       Procyon          lotor 6040.00 1013.00
7   Chiroptera       Molossidae      Molossus          major   11.07    0.22
8   Chiroptera   Phyllostomidae      Artibeus    jamaicensis   40.47    3.79
9   Chiroptera   Phyllostomidae      Artibeus      lituratus   63.65    6.22
10  Chiroptera   Phyllostomidae   Glossophaga       soricina    7.22    0.25
11  Chiroptera   Phyllostomidae  Phyllostomus       discolor   34.37    2.38
12  Chiroptera   Phyllostomidae  Phyllostomus       hastatus   92.26    5.41
13  Chiroptera   Phyllostomidae      Sturnira         lilium   15.39    1.21
14  Chiroptera   Phyllostomidae     Vampyrops       lineatus   22.03    1.59
15  Chiroptera Vespertilionidae     Eptesicus         fuscus   17.88    1.51
16    Edentata      Dasypodidae    Euphractos     sexcinctus 2459.00  252.20
17 Insectivora         Talpidae      Scalopus      aquaticus   44.64    1.23
18  Lagomorpha      Ochotonidae      Ochotona       collaris  120.90    7.00
19 Marsupialia     Didelphiidae     Didelphis    marsupialis 1411.00  107.00
20    Primates  Calllitrichidae    Callithrix        jacchus  186.00    8.70
21    Rodentia       Castoridae        Castor     canadensis 9331.00  865.00
22    Rodentia       Cricetidae Clethrionomys        gapperi   18.34    0.14
23    Rodentia       Cricetidae Clethrionomys        rutilus   25.27    0.72
24    Rodentia       Cricetidae        Lemmus  trimucronatus   41.62    0.75
25    Rodentia       Cricetidae      Microtus pennsylvanicus   31.38    1.20
26    Rodentia       Cricetidae      Microtus      oeconomus   24.83    0.45
27    Rodentia       Cricetidae      Microtus      pinetorum   19.41    0.45
28    Rodentia       Cricetidae       Ondatra      zibethica 1180.00   86.00
29    Rodentia       Cricetidae      Oryzomys      palustris   61.62    7.88
30    Rodentia       Cricetidae    Peromyscus       leucopus   16.99    0.59
```

31	Rodentia	Cuniculidae	Cuniculus	paca	1565.00	196.50
32	Rodentia	Dasyproctidae	Dasyprocta	aguti	2097.00	263.40
33	Rodentia	Erethizontidae	Erethizon	dorsatum	5339.00	674.00
34	Rodentia	Muridae	Mus	musculus	15.88	0.96
35	Rodentia	Sciuridae	Citellus	undulatus	479.00	21.00
36	Rodentia	Sciuridae	Marmota	caligata	3558.00	749.00
37	Rodentia	Sciuridae	Marmota	monax	2194.00	536.50
38	Rodentia	Sciuridae	Sciurus	carolinensis	499.00	11.00
39	Rodentia	Sciuridae	Tamiasciurus	hudsonicus	192.80	3.80

	FFWT	CNS	HEART	MUSCLE	BONE	Potato
1	6568.00	105.09	27.59	4341.45	631.18	Potato
2	5414.00	81.75	25.45	3600.31	552.23	Potato
3	8800.00	85.36	80.96	5271.20	879.12	Potato
4	180.20	6.69	1.87	104.70	21.98	Potato
5	966.00	18.06	7.63	581.53	80.27	Potato
6	5027.00	58.31	36.19	2920.69	517.78	Potato
7	10.89	0.35	0.15	5.51	1.36	Potato
8	36.18	0.96	0.47	18.02	4.48	Potato
9	57.19	1.21	0.74	29.05	8.09	Potato
10	7.15	0.37	0.10	3.86	0.69	Potato
11	32.20	1.00	0.36	16.49	3.87	Potato
12	87.05	2.10	0.89	47.01	11.75	Potato
13	14.25	0.62	0.16	6.33	2.04	Potato
14	20.24	0.76	0.24	10.59	2.23	Potato
15	16.37	0.32	0.19	7.43	2.26	Potato
16	2123.00	19.32	12.95	864.06	269.20	Potato
17	43.41	1.01	0.34	21.88	5.30	Potato
18	113.90	3.06	0.73	57.18	11.32	Potato
19	1304.00	7.56	7.56	681.99	203.42	Potato
20	176.20	7.56	1.22	87.92	26.69	Potato
21	8466.00	53.34	27.94	4622.44	897.40	Potato
22	18.20	0.64	0.13	9.25	2.22	Potato
23	24.55	0.60	0.19	11.34	1.94	Potato
24	40.87	1.03	0.28	19.94	3.82	Potato
25	30.18	0.76	0.26	14.46	2.59	Potato
26	24.38	0.67	0.19	11.12	2.44	Potato
27	18.96	0.57	0.15	9.46	1.86	Potato
28	1094.00	7.11	3.50	679.37	115.96	Potato
29	53.74	1.11	0.34	26.92	5.33	Potato
30	16.40	0.61	0.17	8.02	1.49	Potato
31	1368.00	29.00	7.80	737.35	140.90	Potato
32	1833.80	25.86	13.94	1115.13	168.53	Potato
33	4725.00	37.80	24.10	2197.13	576.45	Potato

```
34   14.92   0.48  0.15    7.07   1.22 Potato
35  458.00   6.00  2.56  257.85  38.01 Potato
36 2809.00  20.37 16.57 1671.36 257.02 Potato
37 1657.50  12.73  8.11  817.13 149.49 Potato
38  488.00   8.88  2.83  306.46  51.73 Potato
39  189.00   5.50  1.68  114.16  18.16 Potato
```

```r
# RS Note: You can also index using row/column name text
scho[,'Potato']
```

```
 [1] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
 [9] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
[17] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
[25] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
[33] "Potato" "Potato" "Potato" "Potato" "Potato" "Potato" "Potato"
```

```r
scho = scho[,colnames(scho) != 'Potato']

# RS Note: I recommend avoiding using numbers to index as much as humanly possible
# Using numbers to index like this is called hard coding
# If your data changes, your code will no longer work as intended
test = scho
test[,5]
```

```
 [1] "Alaska"   "Virginia" "Alaska"   "Alaska"   "Virginia" "Virginia"
 [7] "Brazil"   "Brazil"   "Brazil"   "Brazil"   "Brazil"   "Brazil"
[13] "Brazil"   "Brazil"   "Virginia" "Brazil"   "Virginia" "Alaska"
[19] "Virginia" "Brazil"   "Virginia" "Virginia" "Alaska"   "Alaska"
[25] "Virginia" "Alaska"   "Virginia" "Virginia" "Virginia" "Virginia"
[31] "Brazil"   "Brazil"   "Virginia" "Virginia" "Alaska"   "Alaska"
[37] "Alaska"   "Virginia" "Alaska"
```

```r
test = cbind('Carrot', test)
test[,5]
```

```
 [1] "canadensis"     "rufus"          "luscus"         "erminea"
 [5] "vison"          "lotor"          "major"          "jamaicensis"
 [9] "lituratus"      "soricina"       "discolor"       "hastatus"
[13] "lilium"         "lineatus"       "fuscus"         "sexcinctus"
[17] "aquaticus"      "collaris"       "marsupialis"    "jacchus"
```

```
[21] "canadensis"     "gapperi"        "rutilus"        "trimucronatus"
[25] "pennsylvanicus" "oeconomus"      "pinetorum"      "zibethica"
[29] "palustris"      "leucopus"       "paca"           "aguti"
[33] "dorsatum"       "musculus"       "undulatus"      "caligata"
[37] "monax"          "carolinensis"   "hudsonicus"
```

```r
# conditional indexing
# sometimes you want to extract the datapoints that meet certain
# conditions

scho$Mass[which(scho$Mass < 100)]# get the values that meet a condition
```

```
 [1] 11.07 40.47 63.65  7.22 34.37 92.26 15.39 22.03 17.88 44.64 18.34 25.27
[13] 41.62 31.38 24.83 19.41 61.62 16.99 15.88
```

```r
which(scho$Mass < 100)# get the index of rows that meet a condition
```

```
 [1]  7  8  9 10 11 12 13 14 15 17 22 23 24 25 26 27 29 30 34
```

```r
which(is.na(scho$Mass)) # RS Note: Return indexes where scho$Mass = NA
```

```
[1] 5
```

```r
# you can create a new dataframe that includes only rows that
# meet a certain condition:

scho_big <- scho[-which(scho$Mass<100),]
#this means that scho_big will include all the rows of scho
# Except those whose weight is less than 100.
# the "-" sign excludes rows

scho_clean <- scho[-which(is.na(scho$Mass)),]
# here you are removing all rows that have NA'values

# RS Note: There are different ways to do the same thing, remember there are no wrong answers
# Everyone has their own coding style/preferences. I would do the same thing like this:
scho_clean2 <- scho[(is.na(scho$Mass)) == F,]

scho_clean == scho_clean2 # Same result
```

```
   Order Family Genus Species Location Mass  Fat FFWT  CNS HEART MUSCLE BONE
1   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
2   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
3   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
4   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
6   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
7   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
8   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
9   TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
10  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
11  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
12  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
13  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
14  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
15  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
16  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
17  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
18  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
19  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
20  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
21  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
22  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
23  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
24  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
25  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
26  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
27  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
28  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
29  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
30  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
31  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
32  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
33  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
34  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
35  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
36  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
37  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
38  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
39  TRUE   TRUE  TRUE    TRUE     TRUE TRUE TRUE TRUE TRUE  TRUE   TRUE TRUE
```

```r
# you can also do a dataframe that only includes those instances:
scho_small <- scho[which(scho$Mass<100),]
```

```r
# RS Section: Useful Indexers
# Here are some useful indexing methods
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
# Rows
mtcars[mtcars$cyl == 6,] # == means equal to
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```r
mtcars[mtcars$cyl != 6,] # != means does not equal
```

```
                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
```

```
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
mtcars[mtcars$cyl <=6,] # <= means less than or equal to (< is just less than)
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
mtcars[mtcars$cyl >=6,] # >= means greater than or equal to (< is just greater than)
```

```
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
```

```
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

```r
# which
cyl6 = which(mtcars$cyl == 6)
cyl6
```

```
[1]  1  2  4  6 10 11 30
```

```r
mtcars[cyl6,]
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```r
# which.min and which.max
minwt = which.min(mtcars$wt)
maxwt = which.max(mtcars$wt)
minwt
```

```
[1] 28
```

```r
maxwt
```

```
[1] 16
```

```r
# good for finding minimums/maximums in data
mtcars[minwt,]
```

```
               mpg cyl disp  hp drat    wt qsec vs am gear carb
Lotus Europa 30.4   4 95.1 113 3.77 1.513 16.9  1  1    5    2
```

```r
mtcars[maxwt,]
```

```
                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
Lincoln Continental 10.4   8  460 215    3 5.424 17.82  0  0    3    4
```

```r
# Which.min and which.max take the first result only, not for use on categorical data
which.min(mtcars$cyl)
```

```
[1] 3
```

```r
which.max(mtcars$cyl)
```

```
[1] 5
```

```r
which(mtcars$cyl == min(mtcars$cyl)) # use which instead for all results for categorical data
```

```
 [1]  3  8  9 18 19 20 21 26 27 28 32
```

```r
# Text (string) metadata
head(PlantGrowth)
```

```
  weight group
1   4.17  ctrl
2   5.58  ctrl
3   5.18  ctrl
4   6.11  ctrl
5   4.50  ctrl
6   4.61  ctrl
```

```r
# grep matches partial strings
trts = grep('trt', PlantGrowth$group)
trts
```

```
 [1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

```r
PlantGrowth[trts,]
```

```
   weight group
11   4.81  trt1
12   4.17  trt1
13   4.41  trt1
14   3.59  trt1
15   5.87  trt1
16   3.83  trt1
17   6.03  trt1
18   4.89  trt1
19   4.32  trt1
20   4.69  trt1
21   6.31  trt2
22   5.12  trt2
23   5.54  trt2
24   5.50  trt2
25   5.37  trt2
26   5.29  trt2
27   4.92  trt2
28   6.15  trt2
29   5.80  trt2
30   5.26  trt2
```

```r
# grepl does the same, but outputs logical instead of numerical
trtsl = grepl('trt', PlantGrowth$group)
trtsl
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```r
PlantGrowth[trtsl == TRUE,]
```

```
   weight group
11   4.81  trt1
12   4.17  trt1
13   4.41  trt1
14   3.59  trt1
15   5.87  trt1
16   3.83  trt1
17   6.03  trt1
18   4.89  trt1
19   4.32  trt1
20   4.69  trt1
21   6.31  trt2
22   5.12  trt2
23   5.54  trt2
24   5.50  trt2
25   5.37  trt2
26   5.29  trt2
27   4.92  trt2
28   6.15  trt2
29   5.80  trt2
30   5.26  trt2
```

```r
# gsub replaces the specified string with another specifed string, useful for removing things
gsub('trt', '', PlantGrowth$group)
```

```
 [1] "ctrl" "ctrl" "ctrl" "ctrl" "ctrl" "ctrl" "ctrl" "ctrl" "ctrl" "ctrl"
[11] "1"    "1"    "1"    "1"    "1"    "1"    "1"    "1"    "1"    "1"
[21] "2"    "2"    "2"    "2"    "2"    "2"    "2"    "2"    "2"    "2"
```

```r
potatoes = gsub('trt', 'POTATO', PlantGrowth$group)
ptatoes = gsub('O.*?', '', potatoes)
potatoes
```

```
 [1] "ctrl"    "ctrl"    "ctrl"    "ctrl"    "ctrl"    "ctrl"    "ctrl"
 [8] "ctrl"    "ctrl"    "ctrl"    "POTATO1" "POTATO1" "POTATO1" "POTATO1"
[15] "POTATO1" "POTATO1" "POTATO1" "POTATO1" "POTATO1" "POTATO1" "POTATO2"
[22] "POTATO2" "POTATO2" "POTATO2" "POTATO2" "POTATO2" "POTATO2" "POTATO2"
[29] "POTATO2" "POTATO2"
```

```r
ptatoes
```

```
 [1] "ctrl"  "ctrl"  "ctrl"  "ctrl"  "ctrl"  "ctrl"  "ctrl"  "ctrl"  "ctrl"
[10] "ctrl"  "PTAT1" "PTAT1" "PTAT1" "PTAT1" "PTAT1" "PTAT1" "PTAT1" "PTAT1"
[19] "PTAT1" "PTAT1" "PTAT2" "PTAT2" "PTAT2" "PTAT2" "PTAT2" "PTAT2" "PTAT2"
[28] "PTAT2" "PTAT2" "PTAT2"
```

```r
# %in%
PlantGrowth[PlantGrowth$group %in% c('ctrl', 'trt1'),]
```

```
   weight group
1    4.17  ctrl
2    5.58  ctrl
3    5.18  ctrl
4    6.11  ctrl
5    4.50  ctrl
6    4.61  ctrl
7    5.17  ctrl
8    4.53  ctrl
9    5.33  ctrl
10   5.14  ctrl
11   4.81  trt1
12   4.17  trt1
13   4.41  trt1
14   3.59  trt1
15   5.87  trt1
16   3.83  trt1
17   6.03  trt1
18   4.89  trt1
19   4.32  trt1
20   4.69  trt1
```

```r
# and
PlantGrowth[(PlantGrowth$weight < 5) & (PlantGrowth$group == 'ctrl'),]
```

```
  weight group
1   4.17  ctrl
5   4.50  ctrl
6   4.61  ctrl
8   4.53  ctrl
```

```r
# or
PlantGrowth[(PlantGrowth$weight < 5) | (PlantGrowth$group == 'ctrl'),]
```

```
   weight group
1    4.17  ctrl
2    5.58  ctrl
3    5.18  ctrl
4    6.11  ctrl
5    4.50  ctrl
6    4.61  ctrl
7    5.17  ctrl
8    4.53  ctrl
9    5.33  ctrl
10   5.14  ctrl
11   4.81  trt1
12   4.17  trt1
13   4.41  trt1
14   3.59  trt1
16   3.83  trt1
18   4.89  trt1
19   4.32  trt1
20   4.69  trt1
27   4.92  trt2
```

```r
# !
PlantGrowth[!(PlantGrowth$group %in% c('ctrl', 'trt1')),]
```

```
   weight group
21   6.31  trt2
22   5.12  trt2
23   5.54  trt2
24   5.50  trt2
25   5.37  trt2
26   5.29  trt2
27   4.92  trt2
28   6.15  trt2
29   5.80  trt2
30   5.26  trt2
```

```r
# -
PlantGrowth[-trts,]
```

```
   weight group
1    4.17  ctrl
2    5.58  ctrl
3    5.18  ctrl
4    6.11  ctrl
5    4.50  ctrl
6    4.61  ctrl
7    5.17  ctrl
8    4.53  ctrl
9    5.33  ctrl
10   5.14  ctrl
```

5 R Refresher Part 5: Using Functions and Packages

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers the use of R functions as well as how to download, load in, and use R packages. We'll cover user-defined functions in Part 8.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

5.1 Functions

Functions are objects in R which are used to compress and run larger blocks of code all at once. For example, `mean(x)` is a faster way of doing `sum(x)/length(x)`. Of course, `sum()` and `length()` are also functions. If it ends in a paranthesis, its a function.

The general structure of functions is `function(argument_1 = value_1,...argument_n = value_n)` Running a function will output some sort of value. You can find out what a function does using the Help tab in RStudio by running `?function`. If an argument has a value in the help tab, that is the default value of that argument - that means if you don't specify a value for that argument, it will use the value shown in the help tab. Just typing the name of a function with no parentheses will print the internal code of the function.

```
# 5. Functions and packages ----------------------------------

# Functions are objects too. The basic structure of a function is:
# function(argument1 = value, argument = value, ...)
# everytime you have an object with followed by parentheses,
# there is a function
# Inside the parentheses you can place the function arguments

# If you type the function's name with no parentheses
# you will get the code for the function:
```

```r
data.frame
```

```
function (..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
    fix.empty.names = TRUE, stringsAsFactors = FALSE)
{
    data.row.names <- if (check.rows && is.null(row.names))
        function(current, new, i) {
            if (is.character(current))
                new <- as.character(new)
            if (is.character(new))
                current <- as.character(current)
            if (anyDuplicated(new))
                return(current)
            if (is.null(current))
                return(new)
            if (all(current == new) || all(current == ""))
                return(new)
            stop(gettextf("mismatch of row names in arguments of 'data.frame', item %d",
                i), domain = NA)
        }
    else function(current, new, i) {
        current %||% if (anyDuplicated(new)) {
            warning(gettextf("some row.names duplicated: %s --> row.names NOT used",
                paste(which(duplicated(new)), collapse = ",")),
                domain = NA)
            current
        }
        else new
    }
    object <- as.list(substitute(list(...)))[-1L]
    mirn <- missing(row.names)
    mrn <- is.null(row.names)
    x <- list(...)
    n <- length(x)
    if (n < 1L) {
        if (!mrn) {
            if (is.object(row.names) || !is.integer(row.names))
                row.names <- as.character(row.names)
            if (anyNA(row.names))
                stop("row names contain missing values")
            if (anyDuplicated(row.names))
```

```
            stop(gettextf("duplicate row.names: %s", paste(unique(row.names[duplicated(r
              collapse = ", ")), domain = NA)
    }
    else row.names <- integer()
    return(structure(list(), names = character(), row.names = row.names,
        class = "data.frame"))
}
vnames <- names(x)
if (length(vnames) != n)
    vnames <- character(n)
no.vn <- !nzchar(vnames)
vlist <- vnames <- as.list(vnames)
nrows <- ncols <- integer(n)
for (i in seq_len(n)) {
    xi <- if (is.character(x[[i]]) || is.list(x[[i]]))
        as.data.frame(x[[i]], optional = TRUE, stringsAsFactors = stringsAsFactors)
    else as.data.frame(x[[i]], optional = TRUE)
    nrows[i] <- .row_names_info(xi)
    ncols[i] <- length(xi)
    namesi <- names(xi)
    if (ncols[i] > 1L) {
        if (length(namesi) == 0L)
            namesi <- seq_len(ncols[i])
        vnames[[i]] <- if (no.vn[i])
            namesi
        else paste(vnames[[i]], namesi, sep = ".")
    }
    else if (length(namesi)) {
        vnames[[i]] <- namesi
    }
    else if (fix.empty.names && no.vn[[i]]) {
        tmpname <- deparse(object[[i]], nlines = 1L)[1L]
        if (startsWith(tmpname, "I(") && endsWith(tmpname,
            ")")) {
            ntmpn <- nchar(tmpname, "c")
            tmpname <- substr(tmpname, 3L, ntmpn - 1L)
        }
        vnames[[i]] <- tmpname
    }
    if (mirn && nrows[i] > 0L) {
        rowsi <- attr(xi, "row.names")
        if (any(nzchar(rowsi)))
            row.names <- data.row.names(row.names, rowsi,
```

```
            i)
    }
    nrows[i] <- abs(nrows[i])
    vlist[[i]] <- xi
}
nr <- max(nrows)
for (i in seq_len(n)[nrows < nr]) {
    xi <- vlist[[i]]
    if (nrows[i] > 0L && (nr%%nrows[i] == 0L)) {
        xi <- unclass(xi)
        fixed <- TRUE
        for (j in seq_along(xi)) {
            xi1 <- xi[[j]]
            if (is.vector(xi1) || is.factor(xi1))
              xi[[j]] <- rep(xi1, length.out = nr)
            else if (is.character(xi1) && inherits(xi1, "AsIs"))
              xi[[j]] <- structure(rep(xi1, length.out = nr),
                class = class(xi1))
            else if (inherits(xi1, "Date") || inherits(xi1,
              "POSIXct"))
              xi[[j]] <- rep(xi1, length.out = nr)
            else {
              fixed <- FALSE
              break
            }
        }
        if (fixed) {
            vlist[[i]] <- xi
            next
        }
    }
    stop(gettextf("arguments imply differing number of rows: %s",
        paste(unique(nrows), collapse = ", ")), domain = NA)
}
value <- unlist(vlist, recursive = FALSE, use.names = FALSE)
vnames <- as.character(unlist(vnames[ncols > 0L]))
if (fix.empty.names && any(noname <- !nzchar(vnames)))
    vnames[noname] <- paste0("Var.", seq_along(vnames))[noname]
if (check.names) {
    if (fix.empty.names)
        vnames <- make.names(vnames, unique = TRUE)
    else {
        nz <- nzchar(vnames)
```

```
            vnames[nz] <- make.names(vnames[nz], unique = TRUE)
        }
    }
    names(value) <- vnames
    if (!mrn) {
        if (length(row.names) == 1L && nr != 1L) {
            if (is.character(row.names))
                row.names <- match(row.names, vnames, 0L)
            if (length(row.names) != 1L || row.names < 1L ||
                row.names > length(vnames))
                stop("'row.names' should specify one of the variables")
            i <- row.names
            row.names <- value[[i]]
            value <- value[-i]
        }
        else if (!is.null(row.names) && length(row.names) !=
            nr)
            stop("row names supplied are of the wrong length")
    }
    else if (!is.null(row.names) && length(row.names) != nr) {
        warning("row names were found from a short variable and have been discarded")
        row.names <- NULL
    }
    class(value) <- "data.frame"
    if (is.null(row.names))
        attr(value, "row.names") <- .set_row_names(nr)
    else {
        if (is.object(row.names) || !is.integer(row.names))
            row.names <- as.character(row.names)
        if (anyNA(row.names))
            stop("row names contain missing values")
        if (anyDuplicated(row.names))
            stop(gettextf("duplicate row.names: %s", paste(unique(row.names[duplicated(row.na
                collapse = ", ")), domain = NA)
        row.names(value) <- row.names
    }
    value
}
<bytecode: 0x555b9f16d2c8>
<environment: namespace:base>
```

```r
# Example: the function 'mean()' returns the mean value of a object
num_vector <- c(3,6,9,12,15)
mean(num_vector)
```

```
[1] 9
```

```r
# type the function with nothing inside the parentheses
#and get a error!
mean()
```

```
Error in mean.default(): argument "x" is missing, with no default
```

```r
# Arguments can change the way a function behaves
num_vector2 <- c(NA,6,9,12,15)
mean(num_vector2)
```

```
[1] NA
```

```r
mean(num_vector2, na.rm = F) # Default
```

```
[1] NA
```

```r
mean(num_vector2, na.rm = T)
```

```
[1] 10.5
```

```r
# 3.1 There are SO MANY functions in R. How do we figure out
#how they work?
#HELP!
help(mean)
?mean

# this will lead you to the help page with
#the description of the function, how you use it,
#what arguments it has, retuned values, references,
#examples...everything you need to know how any function works
```

5.2 Packages

Packages are groups of functions packaged together, which are generally built, maintained, and made publicly available by other R users. In order to use the functions in a package, the package must first be installed, and then libraried, which loads it into your R workspace. Most public packages are installed from the Comprehensive R Archive Network (CRAN), which is an online repository of R packages and their associated code and documentation which are stored in servers across the world. Alternatively, smaller, more niche packages may not be on the CRAN, and thus must be downloaded directly from GitHub.

```
# 4.2 Where are the functions?
# R has several base functions and many,
# many others created by users around the world.
# Formally they are organized into modules, called packages.
# They come in a standardized way, with the help files,
# to help users understandand how to use them.
# We have access to virtually any kind of function in the CRAN
# repository.
# A repository is a "place" where a lot of packages are stored
# and from which they can be accessed
# Examples inlcude CRAN and GitHub
# First, let's see the packages you already have installed:
head(installed.packages())
```

```
          Package
abind     "abind"
ade4      "ade4"
agridat   "agridat"
askpass   "askpass"
babynames "babynames"
backports "backports"
          LibPath
abind     "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-
Assignment-Guide/renv/library/linux-ubuntu-noble/R-4.5/x86_64-pc-linux-gnu"
ade4      "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-
Assignment-Guide/renv/library/linux-ubuntu-noble/R-4.5/x86_64-pc-linux-gnu"
agridat   "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-
Assignment-Guide/renv/library/linux-ubuntu-noble/R-4.5/x86_64-pc-linux-gnu"
askpass   "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-
Assignment-Guide/renv/library/linux-ubuntu-noble/R-4.5/x86_64-pc-linux-gnu"
babynames "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-
Assignment-Guide/renv/library/linux-ubuntu-noble/R-4.5/x86_64-pc-linux-gnu"
```

```
backports "/home/runner/work/BIOL4062-5062-Assignment-Guide/BIOL4062-5062-
Assignment-Guide/renv/library/linux-ubuntu-noble/R-4.5/x86_64-pc-linux-gnu"
          Version  Priority Depends
abind     "1.4-8"  NA       "R (>= 1.5.0)"
ade4      "1.7-23" NA       "R (>= 3.5.0)"
agridat   "1.24"   NA       NA
askpass   "1.2.1"  NA       NA
babynames "1.0.1"  NA       "R (>= 2.10)"
backports "1.5.0"  NA       "R (>= 3.0.0)"
          Imports
abind     "methods, utils"
ade4      "graphics, grDevices, methods, stats, utils, MASS, pixmap, sp,\nRcpp"
agridat   NA
askpass   "sys (>= 2.1)"
babynames "tibble"
backports NA
          LinkingTo
abind     NA
ade4      "Rcpp, RcppArmadillo"
agridat   NA
askpass   NA
babynames NA
backports NA
          Suggests
abind     NA
ade4      "ade4TkGUI, adegraphics, adephylo, adespatial, ape, CircStats,\ndeldir, lattice, sp
agridat   "AER, agricolae, betareg, broom, car, coin, corrgram, desplot,\ndplyr, effects, emm
askpass   "testthat"
babynames "testthat (>= 3.0.0)"
backports NA
          Enhances License              License_is_FOSS License_restricts_use
abind     NA       "MIT + file LICENSE" NA              NA
ade4      NA       "GPL (>= 2)"         NA              NA
agridat   NA       "MIT + file LICENSE" NA              NA
askpass   NA       "MIT + file LICENSE" NA              NA
babynames NA       "CC0"                NA              NA
backports NA       "GPL-2 | GPL-3"      NA              NA
          OS_type MD5sum NeedsCompilation Built
abind     NA      NA     "no"             "4.5.0"
ade4      NA      NA     "yes"            "4.5.0"
agridat   NA      NA     "no"             "4.5.0"
askpass   NA      NA     "yes"            "4.5.0"
babynames NA      NA     NA               "4.5.1"
```

```
backports NA      NA     "yes"            "4.5.0"
```

```r
# # Let's install the package "vegan" that contains many useful ecological tools. We can do i
# install.packages("vegan")

# or going in Tools>Install packages>
# Select a CRAN> select the package you want.
# Google is good to figure out which package has the function you want
# Now, and everytime you start a new R session, you have to
# load the packages you will use. Go ahead and type

library(vegan)
```

```
Loading required package: permute
```

```r
help(package="vegan") # take a look in the package documentation

citation("vegan")
```

```
To cite package 'vegan' in publications use:

  Oksanen J, Simpson G, Blanchet F, Kindt R, Legendre P, Minchin P,
  O'Hara R, Solymos P, Stevens M, Szoecs E, Wagner H, Barbour M,
  Bedward M, Bolker B, Borcard D, Borman T, Carvalho G, Chirico M, De
  Caceres M, Durand S, Evangelista H, FitzJohn R, Friendly M, Furneaux
  B, Hannigan G, Hill M, Lahti L, Martino C, McGlinn D, Ouellette M,
  Ribeiro Cunha E, Smith T, Stier A, Ter Braak C, Weedon J (2025).
  _vegan: Community Ecology Package_. R package version 2.7-1,
  <https://vegandevs.github.io/vegan/>.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {vegan: Community Ecology Package},
    author = {Jari Oksanen and Gavin L. Simpson and F. Guillaume Blanchet and Roeland Kindt a
Helene Ouellette and Eduardo {Ribeiro Cunha} and Tyler Smith and Adrian Stier and Cajo J.F. t
    year = {2025},
    note = {R package version 2.7-1},
    url = {https://vegandevs.github.io/vegan/},
  }
```

```r
# install Packages from GitHub
library(devtools)
```

```
Loading required package: usethis


Attaching package: 'devtools'

The following object is masked from 'package:permute':

    check
```

```r
# devtools::install_github("hadley/babynames", force = TRUE)
library(babynames)
```

6 R Refresher Part 6: Summarizing and Visualizing Data

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers basic summary statistics and plotting.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

6.1 Viewing and summarizing data

Base R contains a variety of functions for examining and summarizing data:

```
# 6. Summarizing and visualizing data -------------------------------------------------------

# Basic functions for summarizing your data
# This are the basic functions to explore your data. Their names are pretty self-explanatory
# mean(), var(), sd(), min(), max(), range(), sum().
# They operate in the entire object

num_vector <- rnorm(50, mean = 20, sd = 10)
num_vector
```

```
 [1] 19.3827590 15.3615014 42.2062728 28.6556675 39.9319400 27.0582816
 [7] 14.9888646 22.2283281  7.1716642 16.7645564 20.1103360 35.7380876
[13] 21.4262599 19.9876690 20.4537753 26.6341665 12.3233295 36.8295804
[19]  3.4798000 28.6062937 26.1920732 17.9435729 19.4967718 39.7140566
[25] 17.2866603 10.2791705 19.2102070  8.2757061 28.4318737 12.8765648
[31] 19.1171651 12.3986905 13.8559217 30.7090330  5.7632458 -0.2397478
[37] 27.6783829 23.1176770 36.2734982  8.2406331 22.6685706  6.3728882
[43] 21.4142009 23.7878705 13.2856378 19.6507817 12.6981731  7.2270723
[49] 23.8792579 38.5203833
```

```r
summary(num_vector) # the basics all together
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
-0.2397 12.9788 19.8192 20.5093 26.9523 42.2063 
```

```r
mean(num_vector)
```

```
[1] 20.5093
```

```r
median(num_vector)
```

```
[1] 19.81923
```

```r
var(num_vector)
```

```
[1] 104.6843
```

```r
sd(num_vector)
```

```
[1] 10.23154
```

```r
min(num_vector)
```

```
[1] -0.2397478
```

```r
max(num_vector)
```

```
[1] 42.20627
```

```r
range(num_vector)
```

```
[1] -0.2397478 42.2062728
```

```r
quantile(num_vector)
```

```
        0%        25%        50%        75%       100% 
-0.2397478 12.9788330 19.8192254 26.9522528 42.2062728 
```

```r
IQR(num_vector) # inter quarter range
```

```
[1] 13.97342
```

```r
sum(num_vector)
```

```
[1] 1025.465
```

```r
cumsum(num_vector) # cumulative sum
```

```
 [1]   19.38276   34.74426   76.95053  105.60620  145.53814  172.59642
 [7]  187.58529  209.81362  216.98528  233.74984  253.86017  289.59826
[13]  311.02452  331.01219  351.46596  378.10013  390.42346  427.25304
[19]  430.73284  459.33913  485.53121  503.47478  522.97155  562.68561
[25]  579.97227  590.25144  609.46165  617.73735  646.16923  659.04579
[31]  678.16296  690.56165  704.41757  735.12660  740.88985  740.65010
[37]  768.32848  791.44616  827.71966  835.96029  858.62886  865.00175
[43]  886.41595  910.20382  923.48946  943.14024  955.83841  963.06548
[49]  986.94474 1025.46513
```

6.2 Apply

The family of `apply()` functions is designed to apply functions across the rows/columns of a matrix/data frame. They are a little old school nowadays (as opposed to the hip new tidyverse pipes covered in section 9), but consider using them in your workflows if they make sense to you. A lot of coding is about finding your own style.

```r
# Hint: the function 'apply()' is a nice way to
#apply any kind of function to parts of you data frame,
#matrix, array. It basically work like this:
# apply(X, MARGIN, FUN), where X is the object;
#MARGIN is 1(row) or 2(column); and FUN is the function
?apply # to explore examples
```

```
starting httpd help server ... done
```

```r
my_matrix1 <-  matrix(1:6, 2, 3, byrow = T)# data, rows, columns
my_matrix1
```

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

```r
apply(my_matrix1, MARGIN=1, FUN=mean) # mean values for each row
```

```
[1] 2 5
```

```r
apply(my_matrix1, MARGIN=2, FUN=mean) # mean values for each column
```

```
[1] 2.5 3.5 4.5
```

```r
scho <- read.csv("Schoenemann.csv", header=T)

# Exploring factor data with tapply
scho # this is the csv data we imported from the Schoenemann dataset
```

```
         Order           Family         Genus        Species Location    Mass
1    Carnivora          Felidae         Felis     canadensis   Alaska 7688.00
2    Carnivora          Felidae         Felis          rufus Virginia 6152.00
3    Carnivora       Mustelidae          Gulo         luscus   Alaska 9362.00
4    Carnivora       Mustelidae       Mustela        erminea   Alaska  183.30
5    Carnivora       Mustelidae       Mustela          vison Virginia 1032.00
6    Carnivora       Proyonidae       Procyon          lotor Virginia 6040.00
7   Chiroptera       Molossidae      Molossus          major   Brazil   11.07
8   Chiroptera   Phyllostomidae      Artibeus    jamaicensis   Brazil   40.47
9   Chiroptera   Phyllostomidae      Artibeus      lituratus   Brazil   63.65
10  Chiroptera   Phyllostomidae   Glossophaga       soricina   Brazil    7.22
11  Chiroptera   Phyllostomidae  Phyllostomus       discolor   Brazil   34.37
12  Chiroptera   Phyllostomidae  Phyllostomus       hastatus   Brazil   92.26
13  Chiroptera   Phyllostomidae      Sturnira         lilium   Brazil   15.39
14  Chiroptera   Phyllostomidae     Vampyrops       lineatus   Brazil   22.03
15  Chiroptera Vespertilionidae     Eptesicus         fuscus Virginia   17.88
16    Edentata      Dasypodidae    Euphractos     sexcinctus   Brazil 2459.00
```

17	Insectivora	Talpidae	Scalopus	aquaticus	Virginia	44.64
18	Lagomorpha	Ochotonidae	Ochotona	collaris	Alaska	120.90
19	Marsupialia	Didelphiidae	Didelphis	marsupialis	Virginia	1411.00
20	Primates	Calllitrichidae	Callithrix	jacchus	Brazil	186.00
21	Rodentia	Castoridae	Castor	canadensis	Virginia	9331.00
22	Rodentia	Cricetidae	Clethrionomys	gapperi	Virginia	18.34
23	Rodentia	Cricetidae	Clethrionomys	rutilus	Alaska	25.27
24	Rodentia	Cricetidae	Lemmus	trimucronatus	Alaska	41.62
25	Rodentia	Cricetidae	Microtus	pennsylvanicus	Virginia	31.38
26	Rodentia	Cricetidae	Microtus	oeconomus	Alaska	24.83
27	Rodentia	Cricetidae	Microtus	pinetorum	Virginia	19.41
28	Rodentia	Cricetidae	Ondatra	zibethica	Virginia	1180.00
29	Rodentia	Cricetidae	Oryzomys	palustris	Virginia	61.62
30	Rodentia	Cricetidae	Peromyscus	leucopus	Virginia	16.99
31	Rodentia	Cuniculidae	Cuniculus	paca	Brazil	1565.00
32	Rodentia	Dasyproctidae	Dasyprocta	aguti	Brazil	2097.00
33	Rodentia	Erethizontidae	Erethizon	dorsatum	Virginia	5339.00
34	Rodentia	Muridae	Mus	musculus	Virginia	15.88
35	Rodentia	Sciuridae	Citellus	undulatus	Alaska	479.00
36	Rodentia	Sciuridae	Marmota	caligata	Alaska	3558.00
37	Rodentia	Sciuridae	Marmota	monax	Alaska	2194.00
38	Rodentia	Sciuridae	Sciurus	carolinensis	Virginia	499.00
39	Rodentia	Sciuridae	Tamiasciurus	hudsonicus	Alaska	192.80

	Fat	FFWT	CNS	HEART	MUSCLE	BONE
1	1120.00	6568.00	105.09	27.59	4341.45	631.18
2	738.00	5414.00	81.75	25.45	3600.31	552.23
3	562.00	8800.00	85.36	80.96	5271.20	879.12
4	3.10	180.20	6.69	1.87	104.70	21.98
5	66.00	966.00	18.06	7.63	581.53	80.27
6	1013.00	5027.00	58.31	36.19	2920.69	517.78
7	0.22	10.89	0.35	0.15	5.51	1.36
8	3.79	36.18	0.96	0.47	18.02	4.48
9	6.22	57.19	1.21	0.74	29.05	8.09
10	0.25	7.15	0.37	0.10	3.86	0.69
11	2.38	32.20	1.00	0.36	16.49	3.87
12	5.41	87.05	2.10	0.89	47.01	11.75
13	1.21	14.25	0.62	0.16	6.33	2.04
14	1.59	20.24	0.76	0.24	10.59	2.23
15	1.51	16.37	0.32	0.19	7.43	2.26
16	252.20	2123.00	19.32	12.95	864.06	269.20
17	1.23	43.41	1.01	0.34	21.88	5.30
18	7.00	113.90	3.06	0.73	57.18	11.32
19	107.00	1304.00	7.56	7.56	681.99	203.42

```
20    8.70  176.20   7.56  1.22   87.92  26.69
21  865.00 8466.00  53.34 27.94 4622.44 897.40
22    0.14   18.20   0.64  0.13    9.25   2.22
23    0.72   24.55   0.60  0.19   11.34   1.94
24    0.75   40.87   1.03  0.28   19.94   3.82
25    1.20   30.18   0.76  0.26   14.46   2.59
26    0.45   24.38   0.67  0.19   11.12   2.44
27    0.45   18.96   0.57  0.15    9.46   1.86
28   86.00 1094.00   7.11  3.50  679.37 115.96
29    7.88   53.74   1.11  0.34   26.92   5.33
30    0.59   16.40   0.61  0.17    8.02   1.49
31  196.50 1368.00  29.00  7.80  737.35 140.90
32  263.40 1833.80  25.86 13.94 1115.13 168.53
33  674.00 4725.00  37.80 24.10 2197.13 576.45
34    0.96   14.92   0.48  0.15    7.07   1.22
35   21.00  458.00   6.00  2.56  257.85  38.01
36  749.00 2809.00  20.37 16.57 1671.36 257.02
37  536.50 1657.50  12.73  8.11  817.13 149.49
38   11.00  488.00   8.88  2.83  306.46  51.73
39    3.80  189.00   5.50  1.68  114.16  18.16
```

```r
# available in the course excel files
# the function 'tapply()' is the right tool for the job:
# tapply(X, INDEX, FUN); same as before, but now INDEX
# is the factor

#first, transform into factors
scho$Location_f <- factor(scho$Location)
scho$Order_f <- factor(scho$Order)



tapply(scho$Fat, INDEX = scho$Location_f, FUN=mean)
```

```
  Alaska   Brazil Virginia
273.1200  61.8225 223.3725
```

```r
# or the standard deviation of Fat..
tapply(scho$Fat, INDEX = scho$Location_f, FUN=sd)
```

```
  Alaska   Brazil Virginia
399.8816 106.9835 365.0005
```

```r
# and so on

# Hint: Contingency tables - the 'table()' function
# also indicates how many samples/category

table(scho$Location_f) #This is particularly useful for plotting (see below)
```

```
  Alaska   Brazil Virginia
      11       12       16
```

```r
table(scho$Location_f, scho$Order_f)
```

```
           Carnivora Chiroptera Edentata Insectivora Lagomorpha Marsupialia
  Alaska           3          0        0           0          1           0
  Brazil           0          8        1           0          0           0
  Virginia         3          1        0           1          0           1

           Primates Rodentia
  Alaska          0        7
  Brazil          1        2
  Virginia        0       10
```

```r
# this is even more useful when we have MORE than one category!!
```

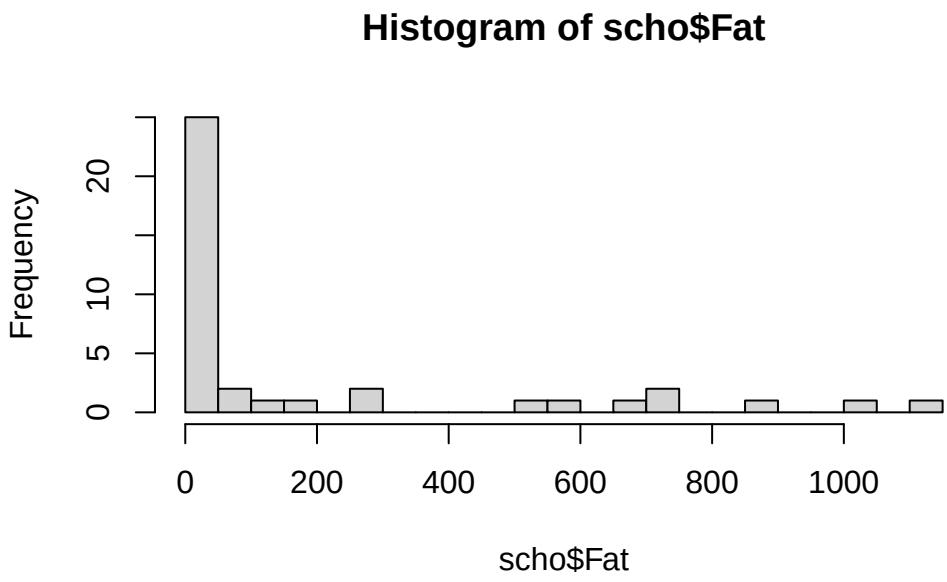
6.3 Base R Plotting

Base R contains a variety of functions that can be used for plotting, see some examples below. R plots work a bit like building blocks. Once you generate a plot, you can use successive functions to add more elements onto the plot.

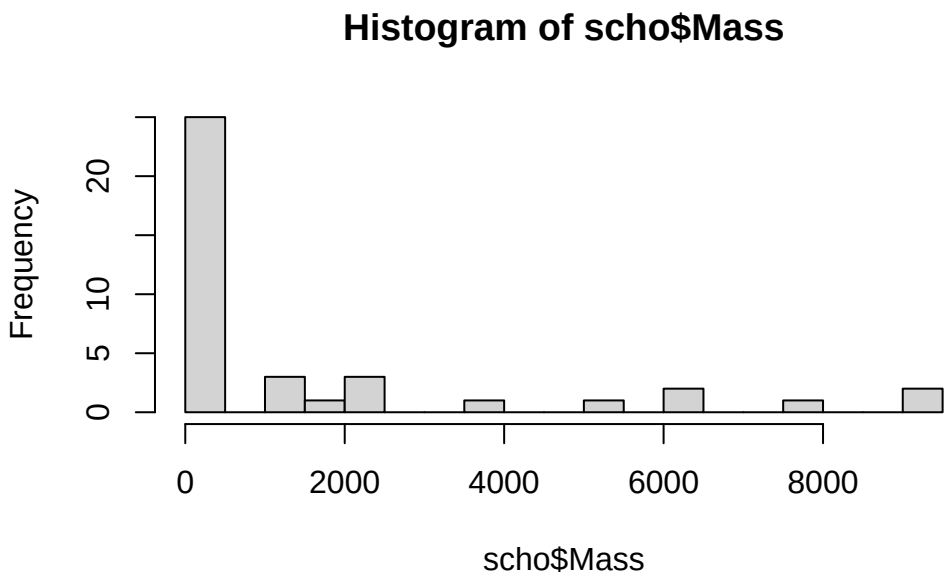
```r
# 6b. Basic plotting  ------------------

# Let's cover the very basics of plotting in R: again,
# we create objects, use functions and voilá!
# We will not cover formatting details.
# It's a bit of work to get a nice plot in R (by nice plot I mean one ready for publication)
# Again, it's worthy anyway, because we get a script to generate the figure as many times you
```

```r
# Histograms and barplots: frequency
hist(scho$Fat)
```

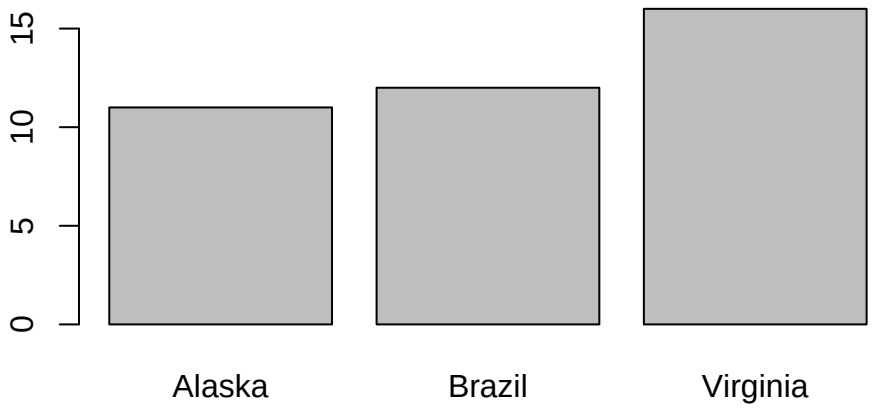

```r
hist(scho$Fat, breaks=20) # a little more detail
```



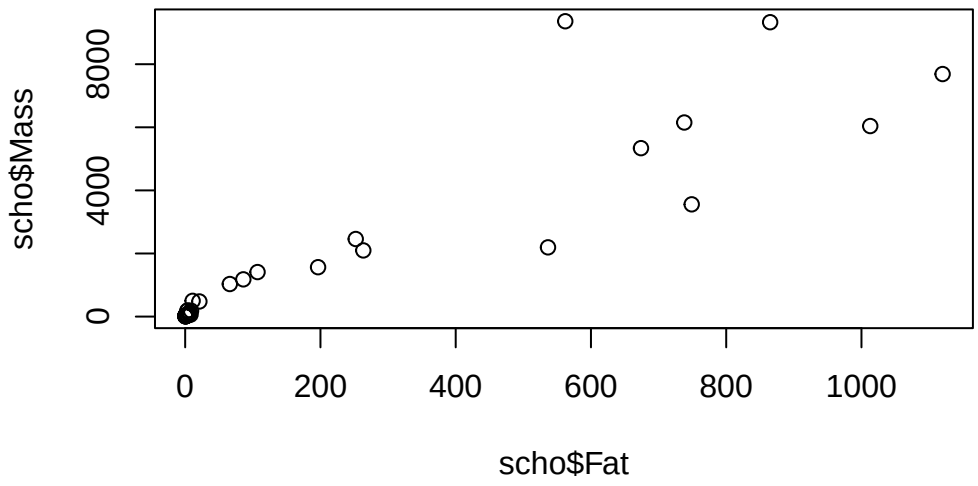

```r
hist(scho$Mass, breaks=20) # a little more detail
```



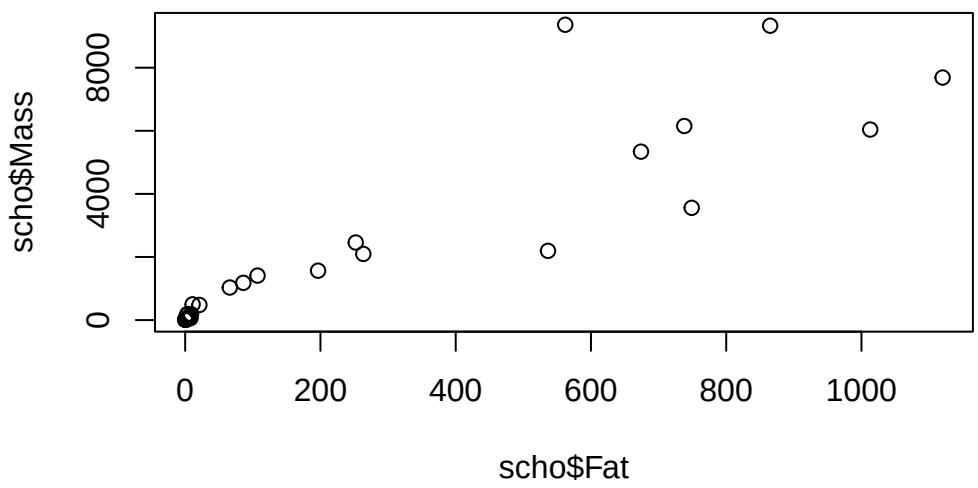
```r
# Barplots for number of cases
barplot(table(scho$Location_f))
```



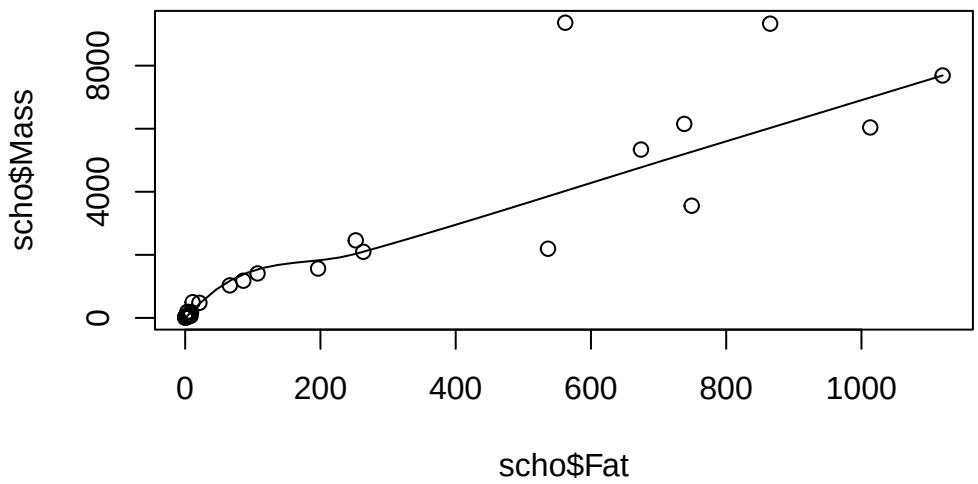

```r
# Scatterplots (x vs y)
plot(scho$Fat, scho$Mass) # plot where x = Fat, y = Mass
```



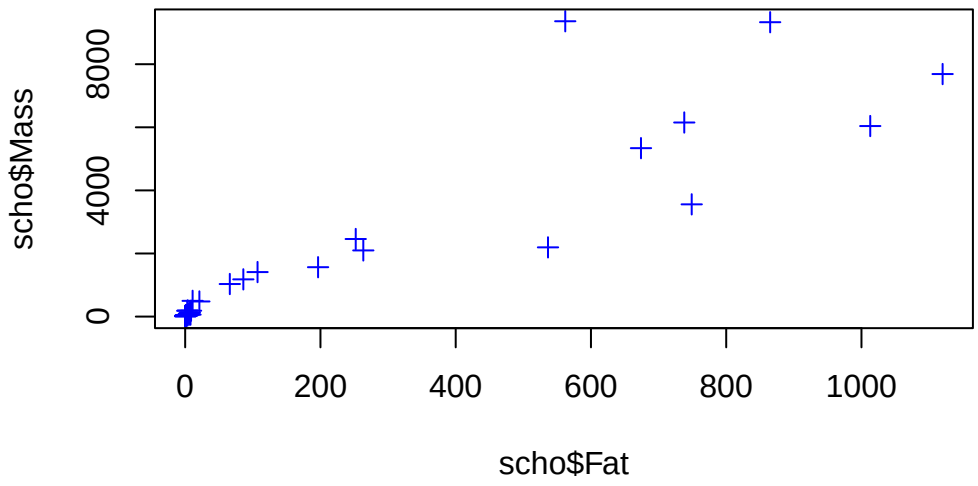

```r
plot(scho$Mass ~ scho$Fat) # plot where x = Fat, y = Mass too
```



```r
scatter.smooth(scho$Mass ~ scho$Fat) # add a trend line
```



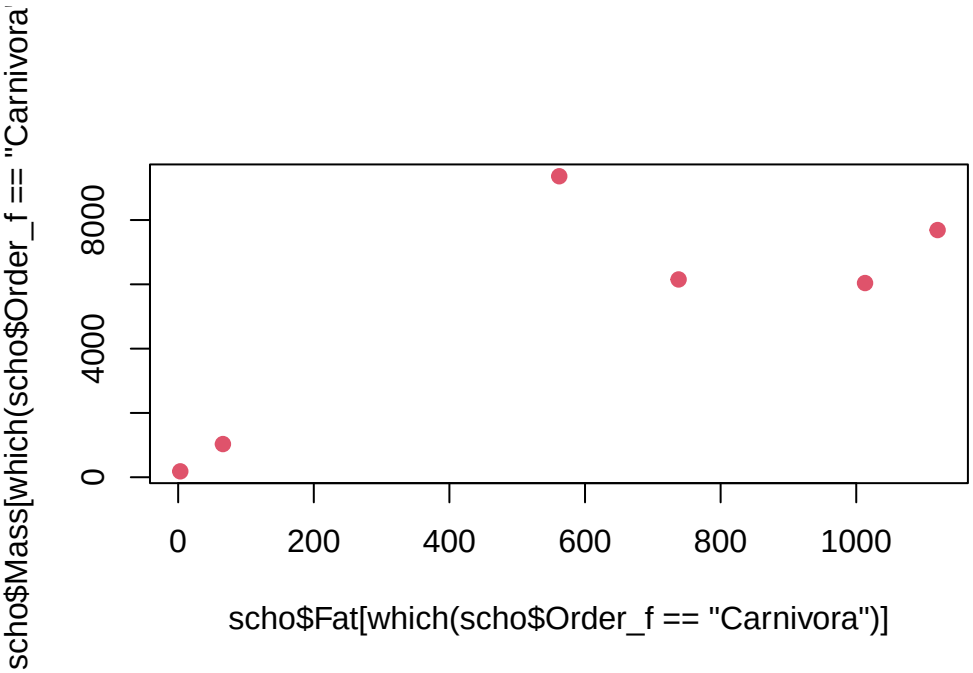

```r
# Basic formatting
plot(scho$Fat, scho$Mass, col = "blue", cex = 1, pch = 3)
```



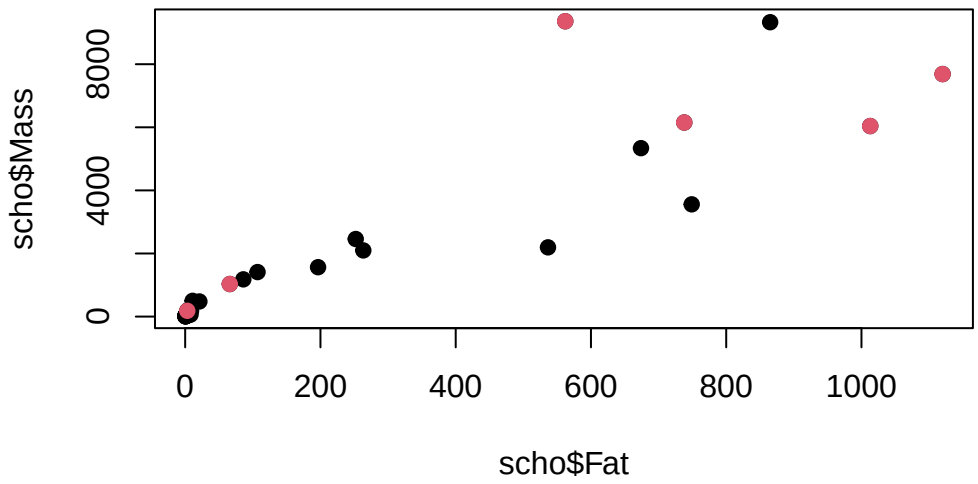

```
# col = color (numbers or names, google options)
# cex = relative size
# pch = type of point - PLAY withe the options

levels(scho$Order_f)
```

```
[1] "Carnivora"   "Chiroptera"  "Edentata"    "Insectivora" "Lagomorpha"
[6] "Marsupialia" "Primates"    "Rodentia"
```

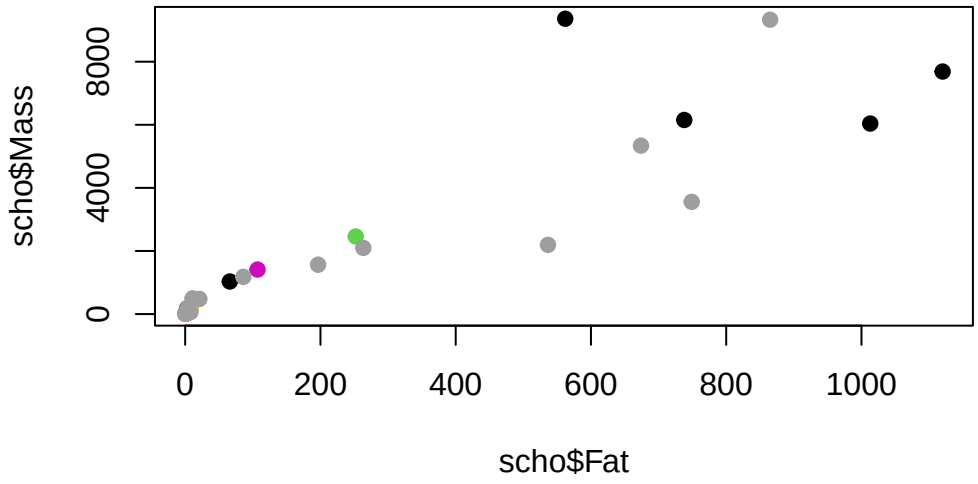
```
# add only carnivores to see if there's a pattern
plot(scho$Fat[which(scho$Order_f == "Carnivora")],
     scho$Mass[which(scho$Order_f == "Carnivora")],
     col = 2, cex = 1, pch = 19)
```



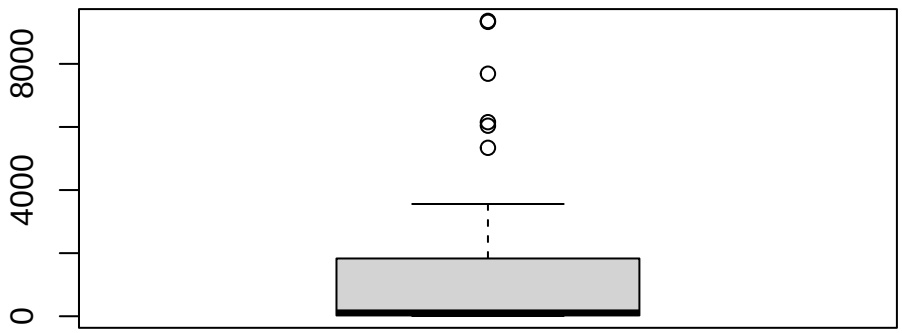

```r
# if I want this to go over my previous graph,
# I can add points
plot(scho$Fat, scho$Mass, col = 1, cex = 1, pch = 19)
points(scho$Fat[which(scho$Order_f == "Carnivora")],
     scho$Mass[which(scho$Order_f == "Carnivora")],
     col = 2, cex = 1, pch = 19)
```



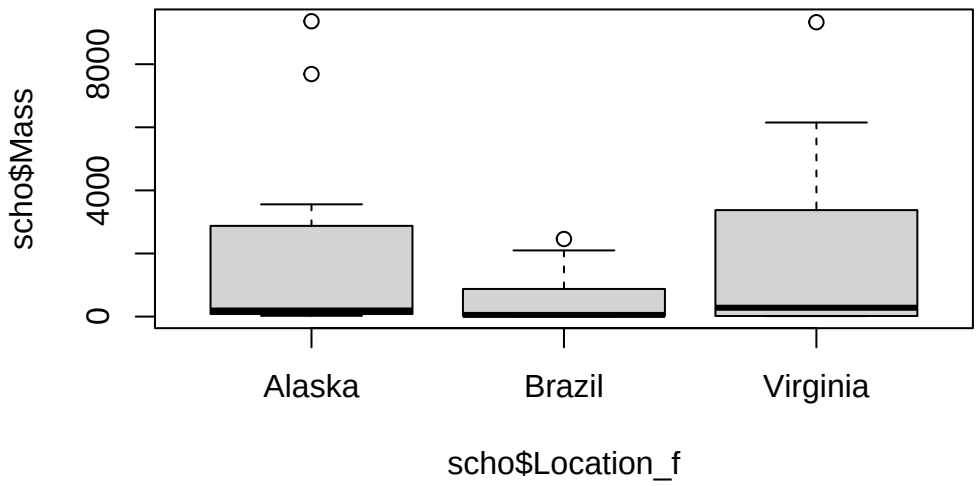

```r
# Color points by category
plot(scho$Fat,
     scho$Mass, col = scho$Order_f, cex = 1, pch = 19)
```



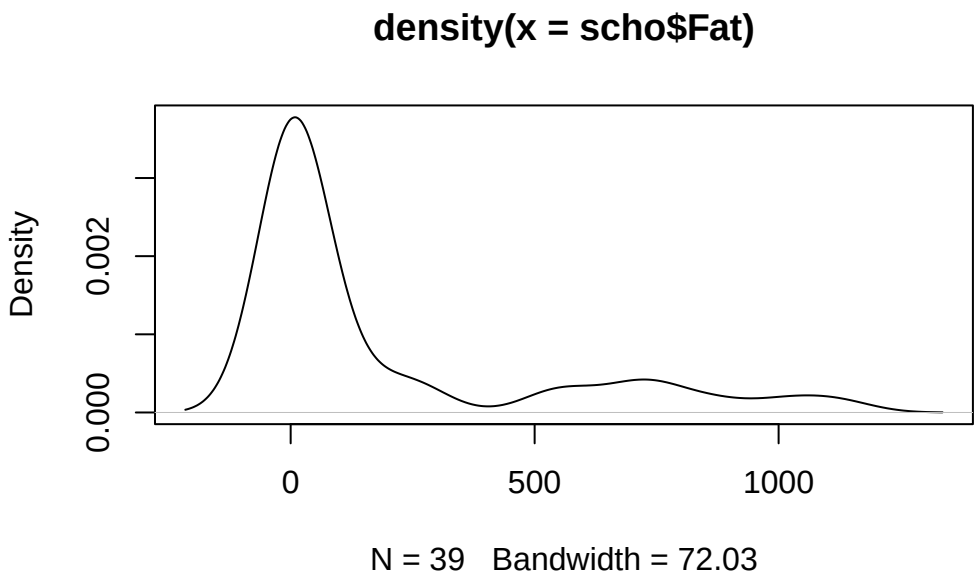
```r
#turns each factor level into a number that corresponds to a color

# Box plot
boxplot(scho$Mass)#one variable (numerical)
```




```r
boxplot(scho$Mass ~ scho$Location_f)# One numerical variable as a function of a categorical v
```




```
?boxplot

# Density plots
plot(density(scho$Fat))
```



6.4 ggplot2

Alternatively, ggplot2 is a plotting package which is part of the tidyverse family that is designed to build plots more quickly (and ideally, prettily). Base plot and ggplot2 is very much a stylistic difference - you can (and should experiment with) build the same plots in either, but you may have a preference one way or the other. ggplot2 is generally more restrictive than R base plot, which can be frustrating in some cases, but may also be beneficial in others, as it can prevent you from making certain mistakes (e.g. mixing up your data and your legend).

```
# 6 BONUS ~~~ Brief intro to ggplot----
# taken and modified from Rebecca Barter:
# URL: http://www.rebeccabarter.com/blog/2017-11-17-ggplot2_tutorial/

# 1. install & upload package
# installl.packages("ggplot2")
library(ggplot2)#first install this package

# 2. BONUS! you can upload data directly
gapminder <- read.csv("https://raw.githubusercontent.com/zief0002/miniature-garbanzo/main/dat
# this will fail if you are not connected to internet though
```

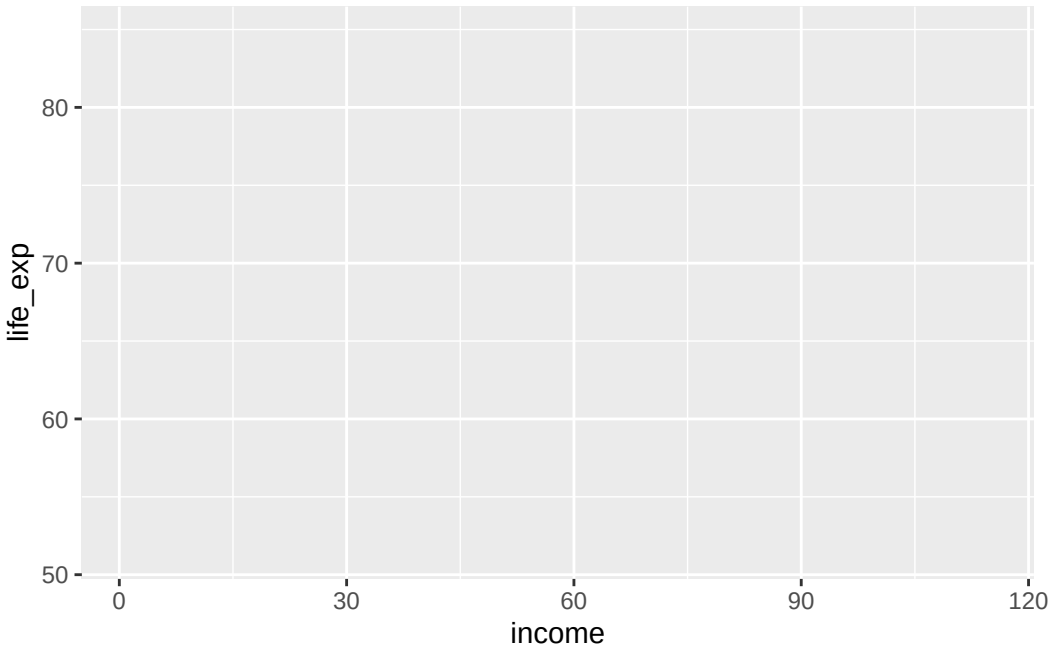
```r
# 3. see what is in there
head(gapminder)
```

```
              country   region income income_level life_exp   co2 co2_change
1         Afghanistan     Asia   2.03      Level 1     62.7 0.254   increase
2             Albania   Europe  13.30      Level 3     78.4 1.590   increase
3             Algeria   Africa  11.60      Level 3     76.0 3.690   increase
4             Andorra   Europe  58.30      Level 4     82.1 6.120   decrease
5              Angola   Africa   6.93      Level 2     64.6 1.120   decrease
6 Antigua and Barbuda Americas  21.00      Level 3     76.2 5.880   increase
  population
1    37.2000
2     2.8800
3    42.2000
4     0.0770
5    30.8000
6     0.0963
```

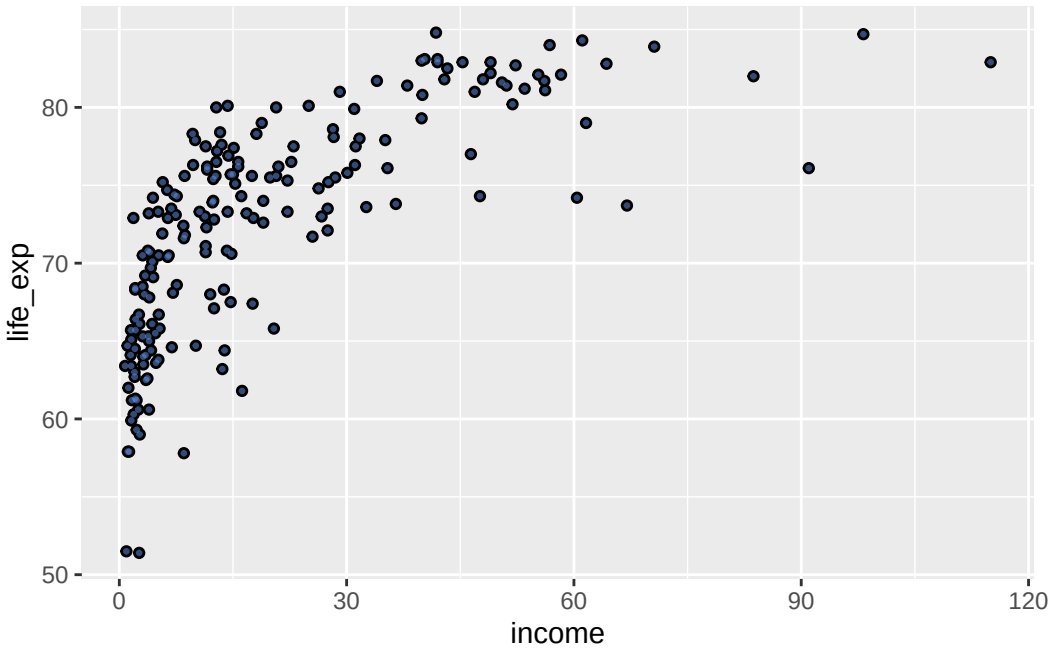
```r
str(gapminder)
```

```
'data.frame':   193 obs. of  8 variables:
 $ country     : chr  "Afghanistan" "Albania" "Algeria" "Andorra" ...
 $ region      : chr  "Asia" "Europe" "Africa" "Europe" ...
 $ income      : num  2.03 13.3 11.6 58.3 6.93 21 22.7 12.7 49 55.3 ...
 $ income_level: chr  "Level 1" "Level 3" "Level 3" "Level 4" ...
 $ life_exp    : num  62.7 78.4 76 82.1 64.6 76.2 76.5 75.6 82.9 82.1 ...
 $ co2         : num  0.254 1.59 3.69 6.12 1.12 5.88 4.41 1.89 16.9 7.75 ...
 $ co2_change  : chr  "increase" "increase" "increase" "decrease" ...
 $ population  : num  37.2 2.88 42.2 0.077 30.8 0.0963 44.4 2.95 24.9 8.89 ...
```

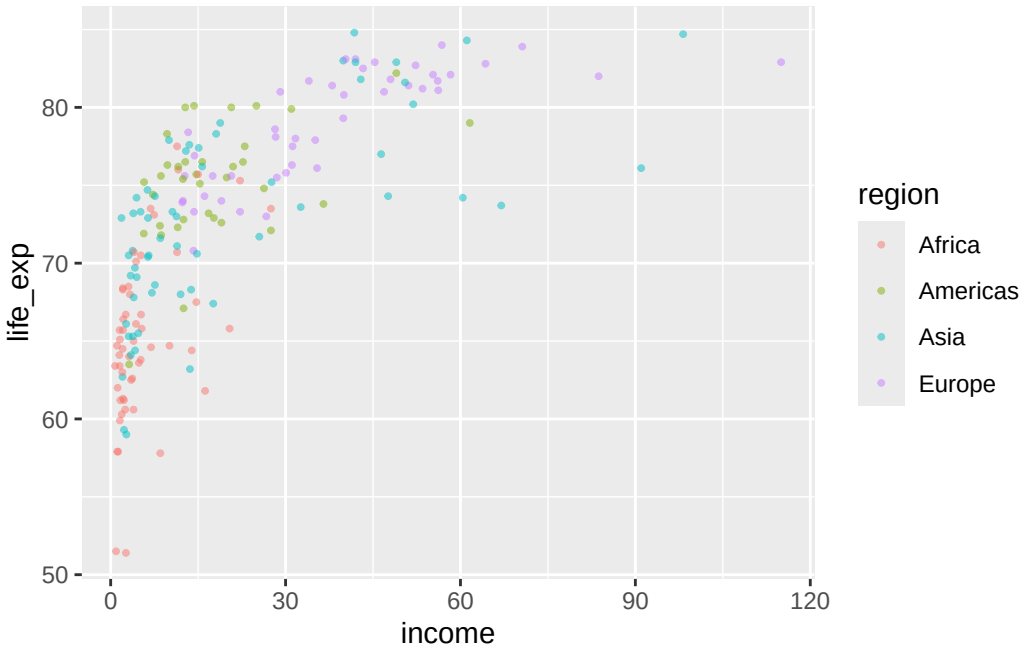
```r
# 4. Start plot:
# ggplots are built in layers, to which you add elements
# the first bit is a "canvas" that holds the x and y axis

ggplot(gapminder, aes(x = income, y = life_exp))
```



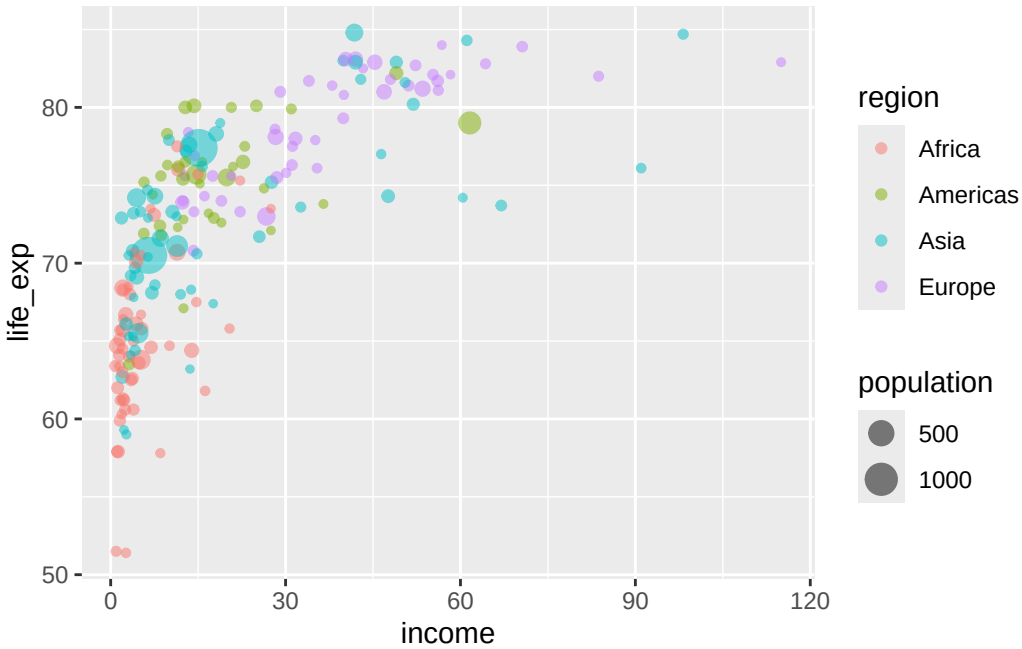

```
# we are telling the function to use the "gapminder dataset"
# and to draw the x axis based on gdp/capita and y axis as life expectancy
# but this is still a blank canvas

# 5. Populate plot
# start adding things by following your "canvas function" by a
# + sign:
p<- ggplot(gapminder, aes(x = income, y = life_exp)) +
  geom_point() # add a points layer on top


# 6. Make things easier to see

# you can modify the format of your points by adding
# specifications within  gemo_point()
# alpha = transparency (0 - 1)
# col = color (explore RColor Palettes for options)
# size = point size

p +
  geom_point(alpha = 0.5, col = "cornflowerblue", size = 0.5)
```




```
# you can also color each point by a category:
# here, because we are plotting color based on a variabe, we
# add it to the aes() input
# so that now points are colored by continent

ggplot(gapminder, aes(x = income, y = life_exp, color = region)) +
  geom_point(alpha = 0.5, size = 0.7)
```




```
# you can also modify the size of points based on a variable
# so that bigger points show larger populations

ggplot(gapminder, aes(x = income, y = life_exp, , color = region,
                      size = population)) +
  geom_point(alpha = 0.5)
```




```
# To make things easier to see, we will look at
# only one year at a time

library(dplyr)
```

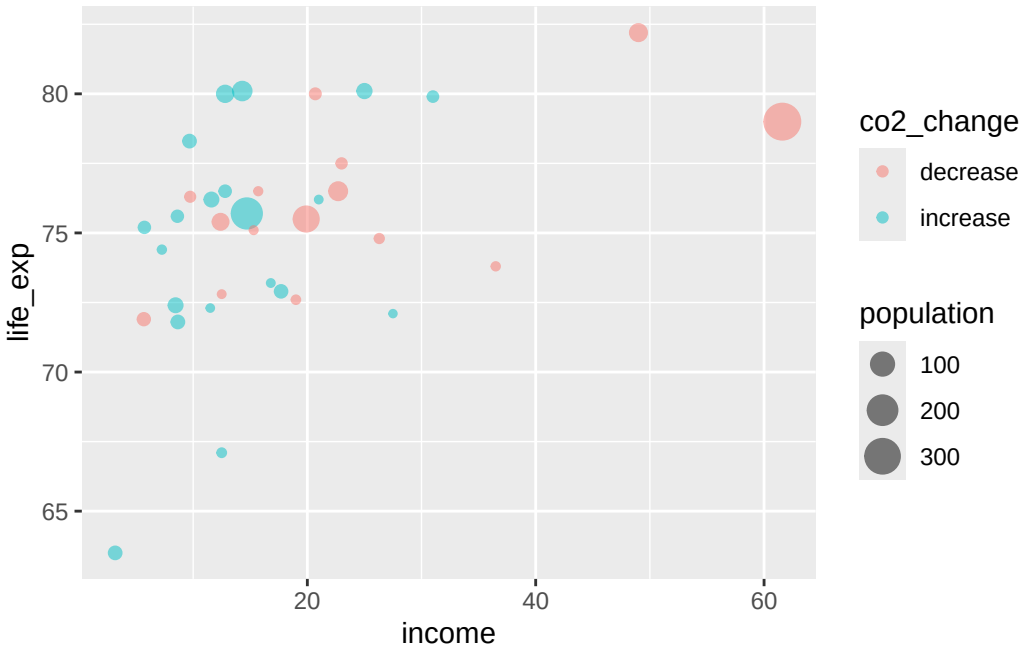
```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
# the dplyr package allows another way to access
# bits of your data
# this can be read as:
# First, take "gapminder" data and THEN
# filter out data for the americas
```

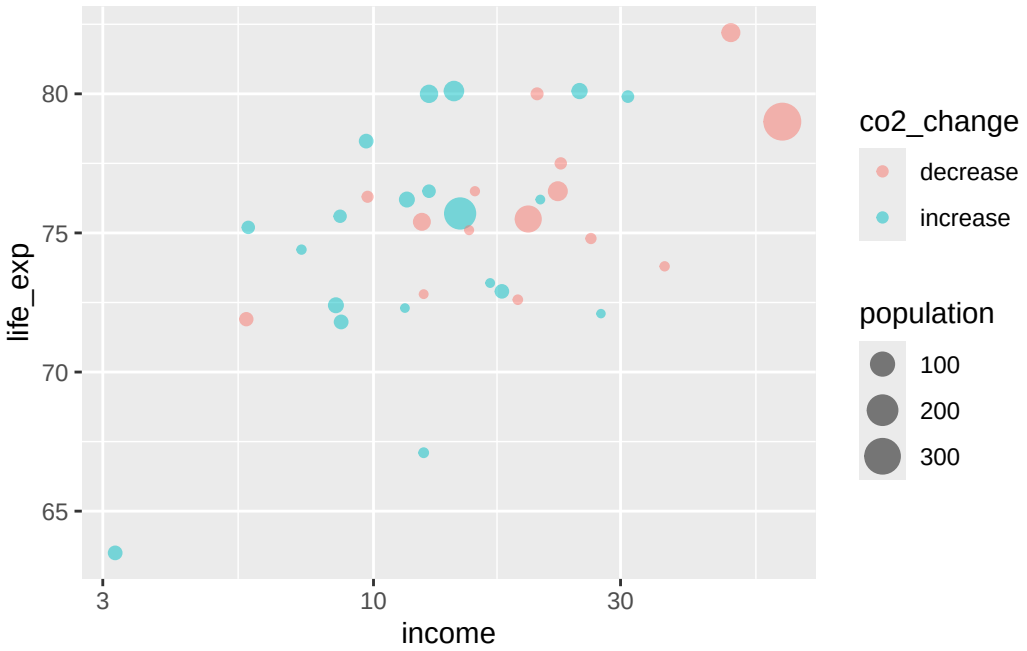
```r
gapminder_Am <- gapminder %>% filter(region == "Americas")
summary(gapminder_Am)
```

```
   country             region              income      income_level
 Length:35          Length:35          Min.   : 3.17   Length:35
 Class :character   Class :character   1st Qu.:10.62   Class :character
 Mode  :character   Mode  :character   Median :14.70   Mode  :character
                                       Mean   :18.02
                                       3rd Qu.:21.85
                                       Max.   :61.60
    life_exp          co2          co2_change          population
 Min.   :63.50   Min.   : 0.270   Length:35          Min.   :  0.0524
 1st Qu.:72.85   1st Qu.: 1.850   Class :character   1st Qu.:  0.4810
 Median :75.50   Median : 2.460   Mode  :character   Median :  6.4700
 Mean   :75.24   Mean   : 4.204                      Mean   : 28.5978
 3rd Qu.:77.00   3rd Qu.: 4.445                      3rd Qu.: 17.9500
 Max.   :82.20   Max.   :31.300                      Max.   :327.0000
```

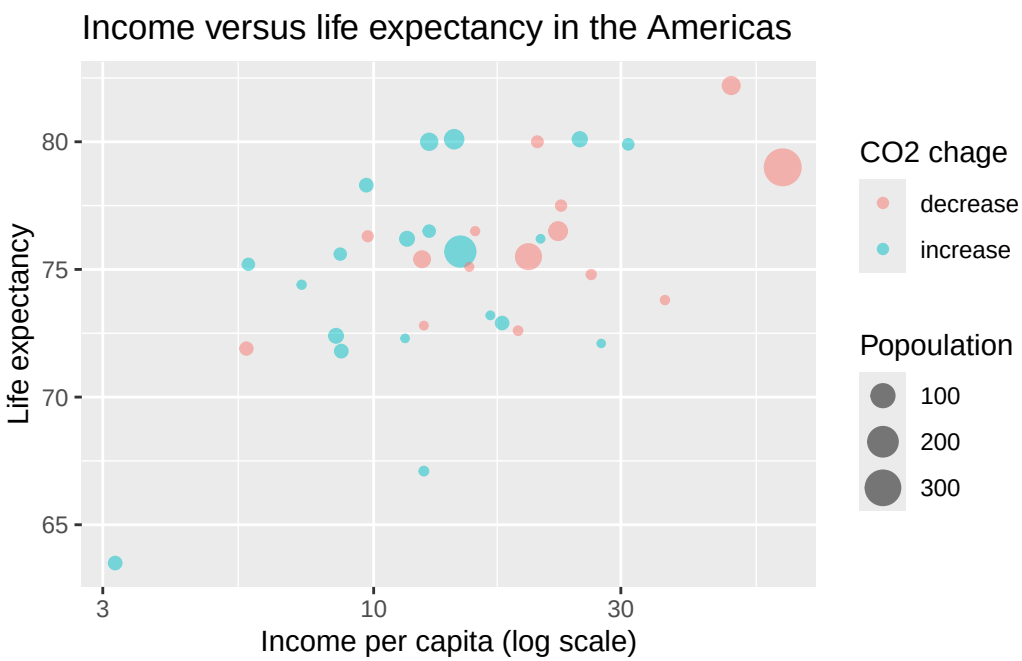
```r
# first we replicate our previous plot
# it is now a bit easier to see patterns

ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = population)) +
  geom_point(alpha = 0.5)
```



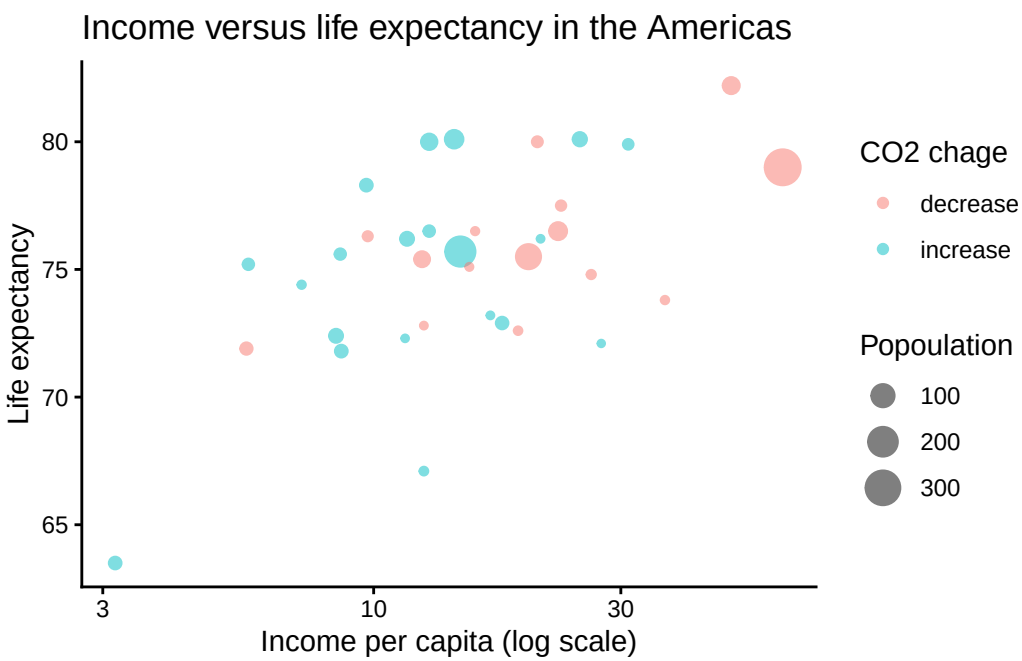

```
# because there is so much variation in income, it
# may be easier to see it in a logarithmic scale

ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = population)) +
  geom_point(alpha = 0.5) +
  scale_x_log10()# this prints the x axis in a log10
```



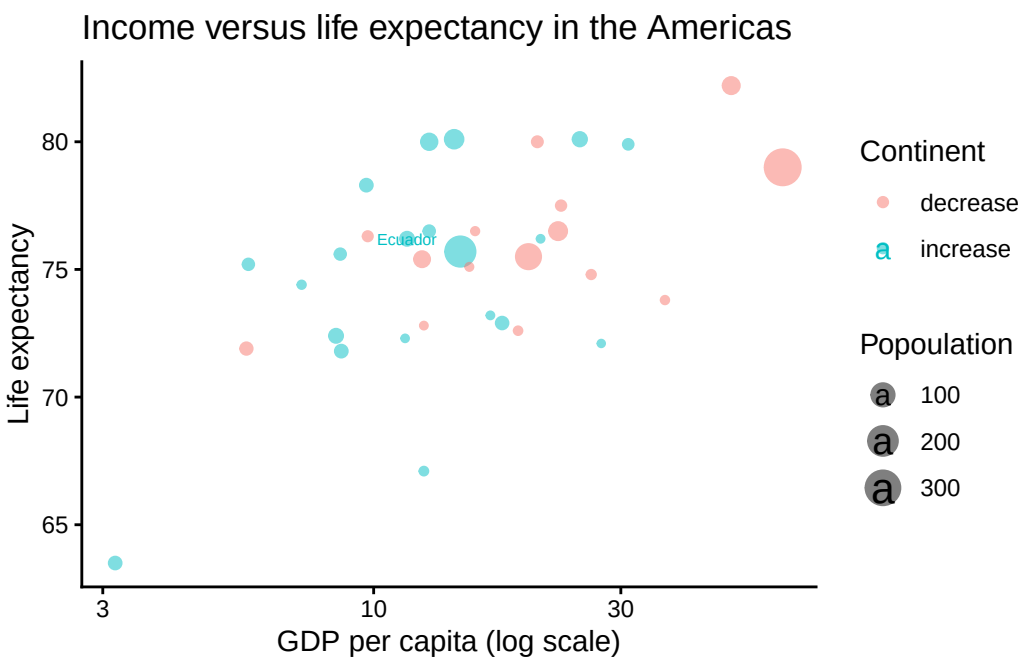

```
# you can now add more informative titles using the
# + labs() element

ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = population)) +
  # add scatter points
  geom_point(alpha = 0.5) +
  # log-scale the x-axis
  scale_x_log10() +
  # change labels
  labs(title = "Income versus life expectancy in the Americas",
       x = "Income per capita (log scale)",
       y = "Life expectancy",
       size = "Popoulation",
       color = "CO2 chage")
```



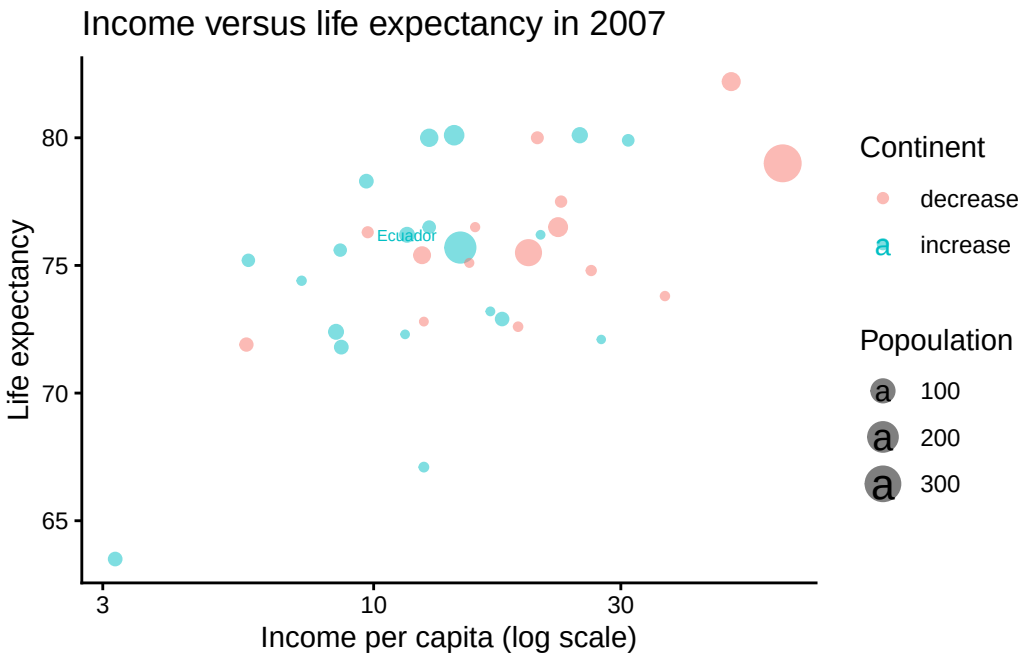

```
# and make things prettier by adding the
# + themes() options
ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = population)) +
  # add scatter points
  geom_point(alpha = 0.5) +
  # log-scale the x-axis
  scale_x_log10() +
  # change labels
  labs(title = "Income versus life expectancy in the Americas",
       x = "Income per capita (log scale)",
       y = "Life expectancy",
       size = "Popoulation",
       color = "CO2 chage")+
       theme_classic()
```




```
# add labels!
# I want to know where Ecuador fits here
ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = population)) +
  # add scatter points
  geom_point(alpha = 0.5) +
  # log-scale the x-axis
  scale_x_log10() +
  # change labels
  labs(title = "Income versus life expectancy in the Americas",
       x = "GDP per capita (log scale)",
       y = "Life expectancy",
       size = "Popoulation",
       color = "Continent")+
       theme_classic() +
    geom_text(
    data=gapminder_Am %>% filter(country == "Ecuador"),
    # Filter data first
    aes(label=country))
```

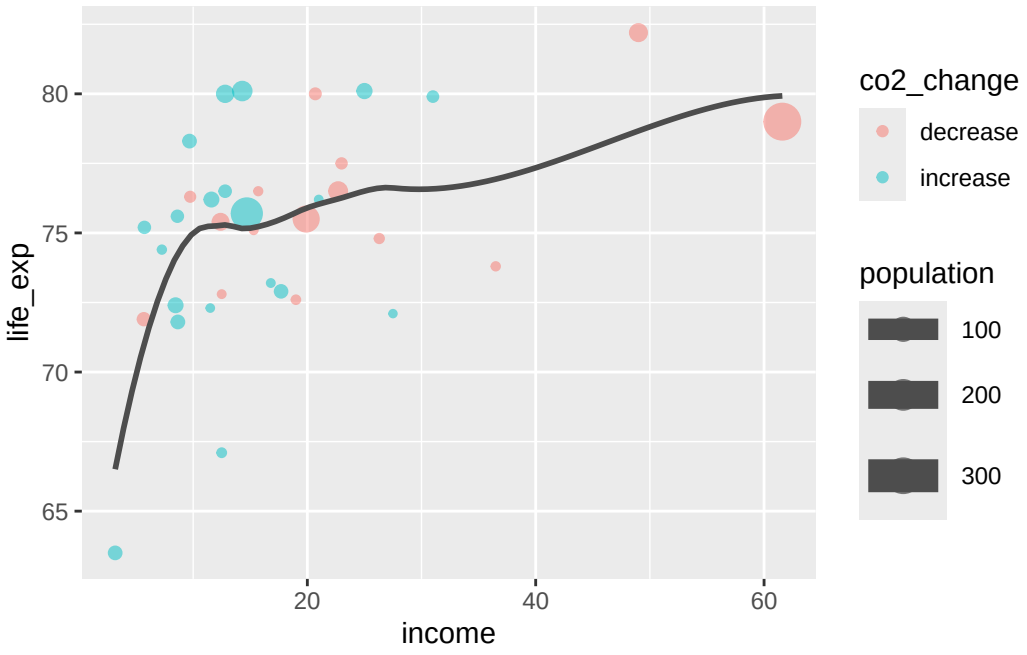



```
# save your plot
# first put it in an object:
p <- ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = populati
  # add scatter points
  geom_point(alpha = 0.5) +
  # log-scale the x-axis
  scale_x_log10() +
  # change labels
  labs(title = "Income versus life expectancy in 2007",
       x = "Income per capita (log scale)",
       y = "Life expectancy",
       size = "Popoulation",
       color = "Continent")+
       theme_classic() +
    geom_text(
    data=gapminder_Am %>% filter(country == "Ecuador"),
    # Filter data first
    aes(label=country))



plot(p)
```

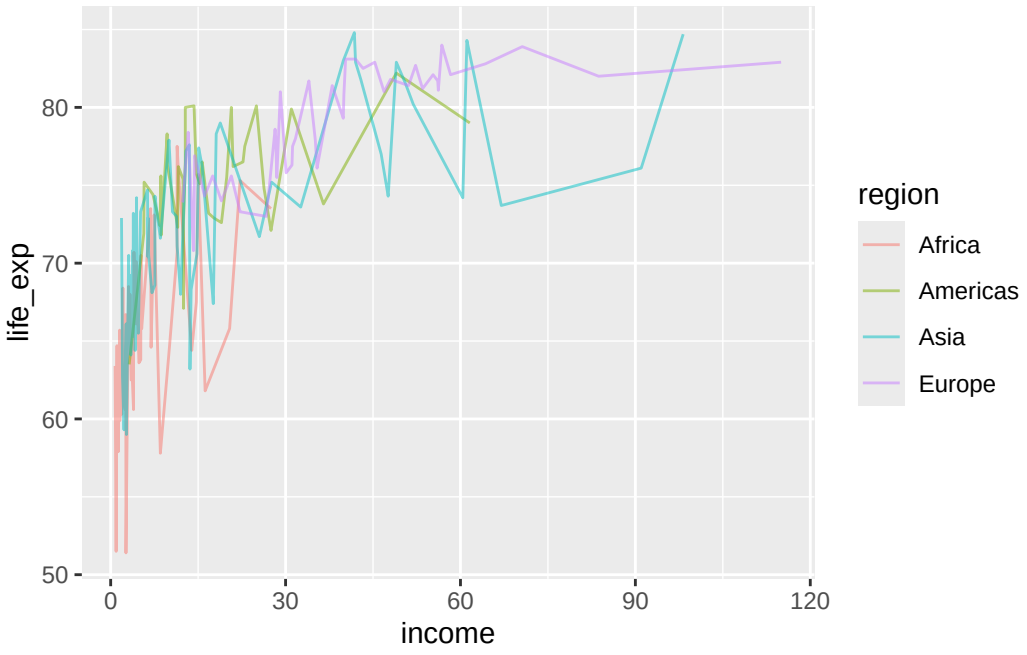



```
# # Save a plot
# ggsave("Graphical_outputs/beautiful_plot.png", p,
#        dpi = 500, width = 10, height = 10)



# you can also overlap things:
# here, we have the initial point graph + a smoother line
ggplot(gapminder_Am, aes(x = income, y = life_exp, color = co2_change , size = population)) +
  geom_point(alpha = 0.5) +
  geom_smooth(se = FALSE, method = "loess", color = "grey30")
```

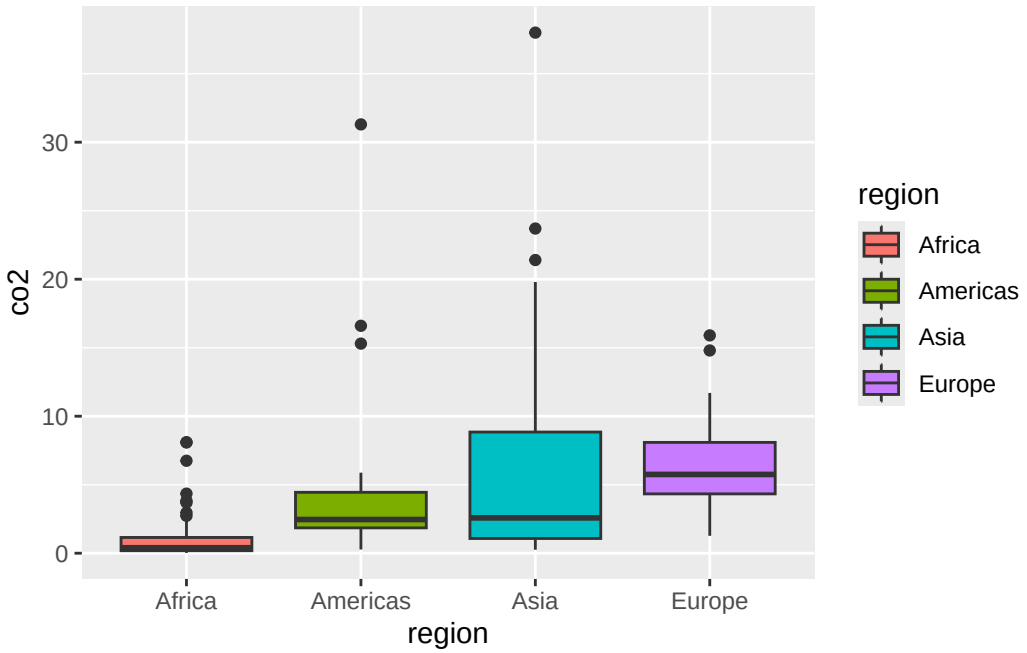
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.

`geom_smooth()` using formula = 'y ~ x'

Warning: The following aesthetics were dropped during statistical transformation: size.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```



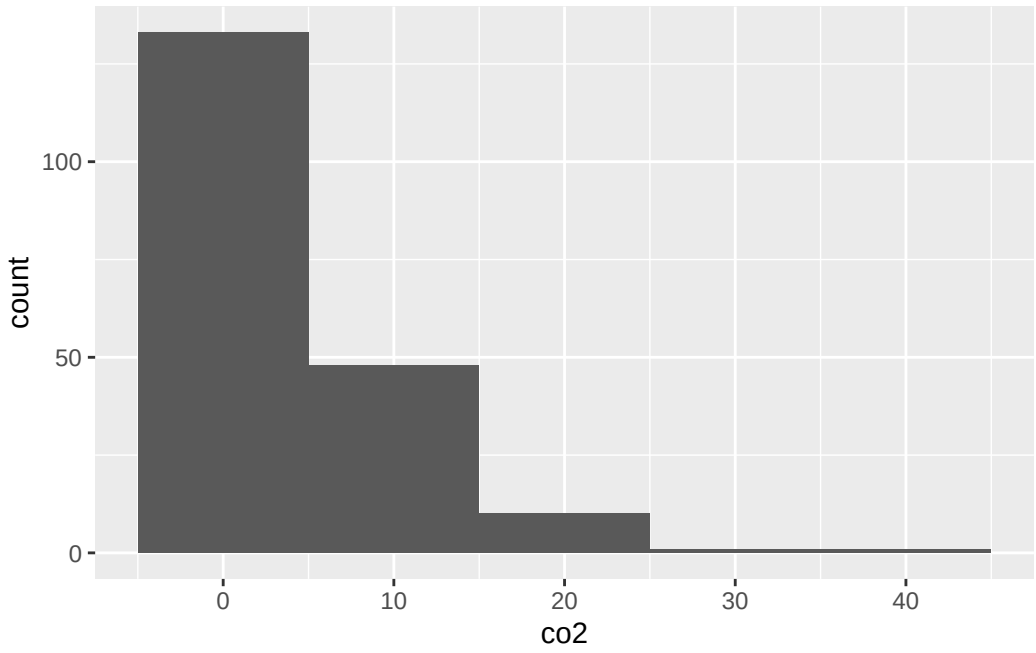

```
# 7.b Different types of plots
# instead of points, you can create a line plot
# (note how contents in the aes() change)
ggplot(gapminder, aes(x = income, y = life_exp,
                      color = region)) +
  geom_line(alpha = 0.5)
```




```
# a boxplot
# (try replacing "fill" by "col")
ggplot(gapminder, aes(x = region, y = co2,
                      fill = region)) +
  geom_boxplot()
```




```
# a histogram
# explore modifying the bindwidth

ggplot(gapminder, aes(x = co2)) +
  geom_histogram(binwidth =10)
```


count
100
50
0
0
10
20
30
40
co2

7 R Refresher Part 7: Linear Modelling

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers basic linear modelling in R.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

7.1 Running linear models

Linear modelling in R is done through the `lm()` function. The `lm()` function requires an x/y formula and data, which can be supplied directly via x/y vectors or through the data frame containing the data. Printing `lm()` and `summary(lm())` prints various results of interest from the model.

```
# 7. Linear models in R (the basics) -------------------------------------

# This is a VERY BASIC introduction to models in R. The idea here is just to illustrate how w
# Basically, here we will repeat the same steps of our recipe: create an object, explore it w

# we already have our data to work with.
# Let's take a look in the function 'lm()' for linear models
help(lm)
```

```
starting httpd help server ... done
```

```
# Basically we need a formula and data.
# Here's another way of getting data that is associated

# # with a package:
# install.packages("agridat")

# if slow connection try running : options(timeout = 400)
```

```r
library(agridat)#this package has many datasets from agriculture
?agridat #shows all different datasets
?lord.rice.uniformity # gives details for this dataset



data("lord.rice.uniformity")
head(lord.rice.uniformity)
```

```
  field row col grain straw
1    10   1   1   9.2  12.2
2    10   1   2   8.4  11.7
3    10   1   3   8.5  12.5
4    10   1   4   9.2  12.0
5    10   1   5   8.0  12.2
6    10  10   1   9.2  10.2
```

```r
#lets give it a shorter name to make our life easier
rice <- lord.rice.uniformity

#Let's pretend we are interested in the linear
#relationship between straw weight and grain weight

lm(grain ~ straw, data = rice)
```

```
Call:
lm(formula = grain ~ straw, data = rice)

Coefficients:
(Intercept)        straw
     0.5103       0.6731
```

```r
# What do we have? The estimates for the linear function
# parameters (intercept and slope)
# if we save this model as an object:
rice_mod <- lm(rice$grain ~ rice$straw) # RS Note: This is the same as lm(grain ~ straw, data

# now we can take a look in other details:
summary(rice_mod)
```

```
Call:
lm(formula = rice$grain ~ rice$straw)

Residuals:
    Min      1Q  Median      3Q     Max 
-5.3147 -0.6476  0.1738  0.8877  3.2412 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.51027    0.21608   2.361   0.0185 *  
rice$straw   0.67313    0.02382  28.262   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.212 on 558 degrees of freedom
Multiple R-squared:  0.5887,	Adjusted R-squared:  0.588 
F-statistic: 798.7 on 1 and 558 DF,  p-value: < 2.2e-16
```

```r
anova(rice_mod)
```

```
Analysis of Variance Table

Response: rice$grain
            Df  Sum Sq Mean Sq F value    Pr(>F)    
rice$straw   1 1173.22 1173.22  798.73 < 2.2e-16 ***
Residuals  558  819.62    1.47                      
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Good! We got the same estimates but with t-tests,
# R^2, Residual standard error, F-test and more..
# you can also explore these by calling them directly:
rice_mod$coefficients
```

```
(Intercept)  rice$straw 
  0.5102717   0.6731326 
```

```r
rice_mod$residuals
```

1	2	3	4	5	6
0.477510692	0.014076986	-0.424429084	0.612137209	-0.722489308	1.823775867
7	8	9	10	11	12
0.068101266	1.560342161	0.423775867	-0.022489308	2.321836091	0.958402385
13	14	15	16	17	18
1.121836091	1.260342161	1.558402385	0.948703503	1.260342161	2.131534972
19	20	21	22	23	24
-0.524429084	0.521836091	-0.832188190	1.958402385	3.168101266	2.233474748
25	26	27	28	29	30
1.521836091	2.021836091	0.650643279	0.512137209	-0.149356721	0.585269797
31	32	33	34	35	36
1.212137209	-0.051296497	0.923775867	0.958402385	0.348703503	1.058402385
37	38	39	40	41	42
1.160342161	0.496908455	1.021836091	-0.214730203	-0.414730203	1.696908455
43	44	45	46	47	48
1.731534972	1.258402385	0.148703503	1.458402385	0.158402385	1.160342161
49	50	51	52	53	54
-0.076224133	-0.014730203	1.385269797	1.594968678	1.668101266	0.558402385
55	56	57	58	59	60
0.923775867	1.521836091	1.996908455	1.094968678	0.660342161	1.358402385
61	62	63	64	65	66
0.485269797	0.760342161	0.860342161	-0.051296497	-0.187862791	1.721836091
67	68	69	70	71	72
0.770041042	1.896908455	0.385269797	0.448703503	0.177800147	0.231534972
73	74	75	76	77	78
0.158402385	1.241233853	0.060342161	3.241233853	1.477800147	1.479739924
79	80	81	82	83	84
0.504667560	1.868101266	0.958402385	0.668101266	1.568101266	1.114366441
85	86	87	88	89	90
1.421836091	1.460342161	-0.914730203	0.804667560	1.268101266	-0.214730203
91	92	93	94	95	96
0.923775867	0.931534972	0.794968678	1.431534972	0.821836091	1.058402385
97	98	99	100	101	102
0.831534972	-0.014730203	1.731534972	0.358402385	0.150932735	0.577800147
103	104	105	106	107	108
-1.031898734	0.587499029	1.004667560	1.641233853	1.577800147	0.577800147
109	110	111	112	113	114
1.141233853	0.614366441	-0.168465028	1.477800147	1.879739924	0.804667560
115	116	117	118	119	120
1.014366441	1.006607336	0.631534972	0.841233853	0.668101266	1.070041042
121	122	123	124	125	126
1.404667560	1.677800147	0.941233853	1.031534972	0.558402385	1.631534972
127	128	129	130	131	132

1.443173630	0.968101266	1.531534972	1.333474748	0.048703503	0.770041042
133	134	135	136	137	138
1.668101266	-0.314730203	1.294968678	-0.987862791	0.668101266	1.094968678
139	140	141	142	143	144
0.731534972	1.296908455	1.170041042	-0.495332440	0.194968678	0.170041042
145	146	147	148	149	150
-0.441597615	1.152872511	-2.495332440	-1.773994901	-3.603091545	-0.514730203
151	152	153	154	155	156
1.079739924	-1.075934678	-3.422199853	-1.722199853	1.768101266	-0.122199853
157	158	159	160	161	162
-3.256826370	-3.249067265	-2.031898734	0.148703503	-1.103091545	-3.358766147
163	164	165	166	167	168
-3.022199853	-1.468465028	1.458402385	-0.049067265	-0.631898734	0.223775867
169	170	171	172	173	174
-0.151296497	-0.441597615	0.416306217	-0.573994901	-0.312500971	0.426005099
175	176	177	178	179	180
-1.568465028	-0.093392664	-0.883693783	-1.537428608	0.014366441	-0.103091545
181	182	183	184	185	186
-0.195332440	-0.239368384	-1.702802090	0.552872511	1.306607336	0.150932735
187	188	189	190	191	192
-1.047127489	-0.993103209	-0.539368384	0.904667560	0.987499029	-1.610561195
193	194	195	196	197	198
-1.302802090	-1.139368384	1.343173630	0.116306217	-1.647127489	-2.102802090
199	200	201	202	203	204
-1.339368384	1.016306217	0.650643279	-2.049067265	-2.439368384	-2.312500971
205	206	207	208	209	210
-0.358766147	1.243173630	-2.556826370	-2.883693783	-0.820260076	0.679739924
211	212	213	214	215	216
-1.731898734	-2.203091545	-2.003091545	-2.166525252	-2.022199853	-1.631898734
217	218	219	220	221	222
-1.714730203	-2.368465028	-0.668465028	-0.312790427	-1.858766147	-2.022199853
223	224	225	226	227	228
-0.839368384	-1.531898734	-1.695332440	0.050932735	-1.949067265	-1.375934678
229	230	231	232	233	234
-0.020260076	-1.705031322	-1.422199853	-1.968465028	-3.214730203	-1.585633559
235	236	237	238	239	240
-1.814730203	-0.776224133	-0.941597615	-1.222489308	-2.387862791	1.704667560
241	242	243	244	245	246
-0.073994901	-2.775934678	-0.975934678	-3.958766147	0.331534972	-2.512790427
247	248	249	250	251	252
-2.447127489	-1.739368384	-1.522199853	0.414366441	-0.458766147	-0.537428608
253	254	255	256	257	258
-0.756826370	-1.122199853	-1.975934678	-0.878163909	-1.575934678	-1.610561195

259	260	261	262	263	264
-1.022199853	-1.868465028	-1.629958958	-2.047127489	-2.393392664	-1.310561195
265	266	267	268	269	270
-2.895332440	-0.683693783	-2.175934678	-3.668465028	-1.275934678	-3.093392664
271	272	273	274	275	276
-1.056826370	-1.383693783	-5.314730203	-1.247127489	-1.356826370	-1.031898734
277	278	279	280	281	282
-1.495332440	-1.295332440	0.204667560	-1.849356721	0.460342161	0.323775867
283	284	285	286	287	288
0.012137209	-3.141887071	-0.503091545	-0.249356721	-0.476224133	0.743173630
289	290	291	292	293	294
0.458402385	-1.551296497	-0.487862791	0.494968678	-0.493392664	-1.651296497
295	296	297	298	299	300
-1.932188190	0.321836091	-0.231898734	-0.131898734	-1.078163909	-1.387862791
301	302	303	304	305	306
0.658402385	1.677800147	0.594968678	-0.314730203	-1.487862791	0.521836091
307	308	309	310	311	312
-1.060995378	-1.160995378	-1.797561672	-0.760995378	1.614366441	1.416306217
313	314	315	316	317	318
1.150932735	-1.068465028	0.870041042	0.533474748	0.404667560	0.150932735
319	320	321	322	323	324
0.068101266	-1.087862791	1.168101266	0.668101266	0.614366441	1.104667560
325	326	327	328	329	330
-1.051296497	0.368101266	0.970041042	1.216306217	-0.141597615	-0.678163909
331	332	333	334	335	336
1.858402385	-0.005031322	-0.431898734	-1.422489308	-2.124429084	0.294968678
337	338	339	340	341	342
0.796908455	0.814366441	1.304667560	-0.041597615	0.970041042	0.394968678
343	344	345	346	347	348
1.014366441	1.033474748	-1.024429084	1.504667560	1.914366441	0.316306217
349	350	351	352	353	354
0.670041042	0.085269797	-1.124429084	-0.432188190	0.102438328	0.548703503
355	356	357	358	359	360
0.931534972	0.070041042	0.224065322	-0.212500971	0.350932735	0.714366441
361	362	363	364	365	366
0.904667560	0.614366441	0.714366441	0.177800147	0.741233853	0.158402385
367	368	369	370	371	372
1.041233853	-0.131898734	1.041233853	-0.129958958	1.168101266	1.216306217
373	374	375	376	377	378
1.114366441	1.314366441	1.504667560	0.431534972	0.968101266	0.623775867
379	380	381	382	383	384
-1.058766147	0.968101266	1.185269797	0.294968678	0.368101266	1.341233853
385	386	387	388	389	390

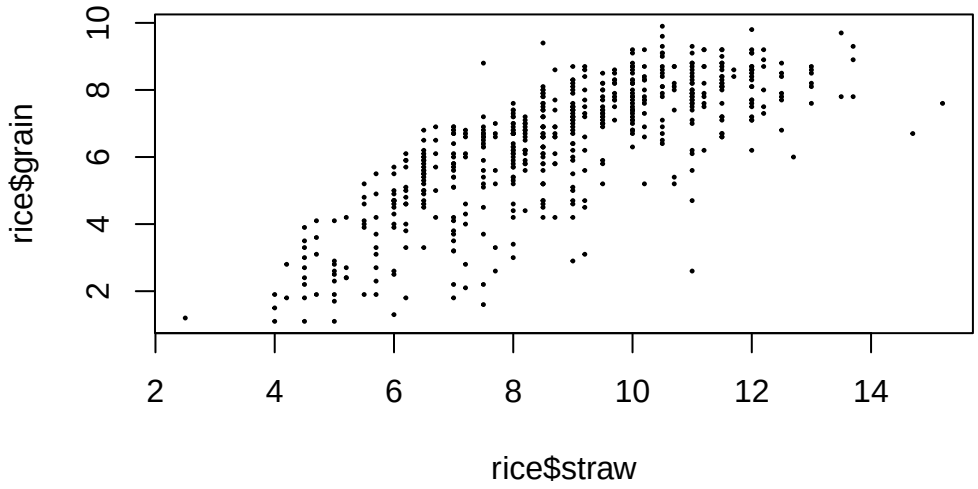
1.368101266	0.748703503	0.531534972	0.304667560	1.141233853	0.221836091
391	392	393	394	395	396
1.150643279	0.094968678	0.404667560	0.914366441	0.906607336	1.023775867
397	398	399	400	401	402
-0.229958958	0.477800147	0.189438805	-0.549067265	0.885269797	0.741233853
403	404	405	406	407	408
-0.185633559	-0.649067265	-0.147127489	1.568101266	0.777800147	-0.385633559
409	410	411	412	413	414
-0.647127489	0.150932735	-0.151296497	0.114366441	-0.083693783	0.387499029
415	416	417	418	419	420
-1.178163909	0.368101266	0.360631616	-0.039368384	-0.083693783	0.514366441
421	422	423	424	425	426
-0.431898734	-3.059055602	-3.705320777	0.758402385	-1.651296497	0.258402385
427	428	429	430	431	432
0.170041042	0.377800147	-0.095332440	0.831534972	0.660342161	0.333474748
433	434	435	436	437	438
-0.158766147	-0.495332440	0.131534972	-0.549356721	-0.429958958	-1.531898734
439	440	441	442	443	444
0.804667560	-1.105031322	0.877800147	-0.112500971	-0.249067265	-0.285633559
445	446	447	448	449	450
-0.695332440	1.304667560	0.631534972	0.104667560	0.577800147	0.341233853
451	452	453	454	455	456
-2.176224133	-0.566525252	-2.312790427	-0.229958958	0.368101266	0.233474748
457	458	459	460	461	462
0.831534972	0.494968678	-1.031898734	-2.314730203	0.996908455	-0.266525252
463	464	465	466	467	468
0.094968678	0.170041042	0.894968678	0.158402385	-0.441597615	-1.051296497
469	470	471	472	473	474
0.570041042	0.487209573	-0.241597615	-1.005031322	-0.568465028	-0.478163909
475	476	477	478	479	480
-0.449356721	0.468101266	0.533474748	1.214366441	0.643173630	0.596908455
481	482	483	484	485	486
-0.368465028	0.068101266	0.314366441	0.804667560	0.948703503	0.470041042
487	488	489	490	491	492
-0.168465028	0.304667560	-0.541597615	0.548703503	-0.660995378	-1.660995378
493	494	495	496	497	498
-0.887862791	-0.887862791	0.214076986	0.112137209	1.331534972	0.950932735
499	500	501	502	503	504
-1.151296497	1.768101266	-0.287862791	0.394968678	0.114366441	0.058402385
505	506	507	508	509	510
0.094968678	0.112137209	-0.141597615	0.450932735	-0.814730203	0.931534972
511	512	513	514	515	516
-0.251296497	-0.195332440	-0.595332440	-1.551296497	0.631534972	-0.014730203

```
         517          518          519          520          521          522
-0.114730203  0.085269797  1.177800147  1.104667560  0.987209573  1.368101266
         523          524          525          526          527          528
 0.804667560  1.294968678  1.323775867 -0.151296497  0.323775867 -0.566525252
         529          530          531          532          533          534
-1.487862791  0.150643279  0.987209573  0.287209573  1.041233853 -0.651296497
         535          536          537          538          539          540
 0.387209573 -0.124429084  0.012137209  0.204667560 -0.560995378 -1.224429084
         541          542          543          544          545          546
 0.177510692 -0.487862791  0.531534972 -0.087862791  0.285269797  0.785269797
         547          548          549          550          551          552
 0.885269797  0.004667560  0.285269797  0.921836091  0.512137209 -0.114730203
         553          554          555          556          557          558
 0.041233853  0.233474748  0.685269797  1.150643279  0.012137209 -0.122199853
         559          560
 0.468101266  0.012137209
```

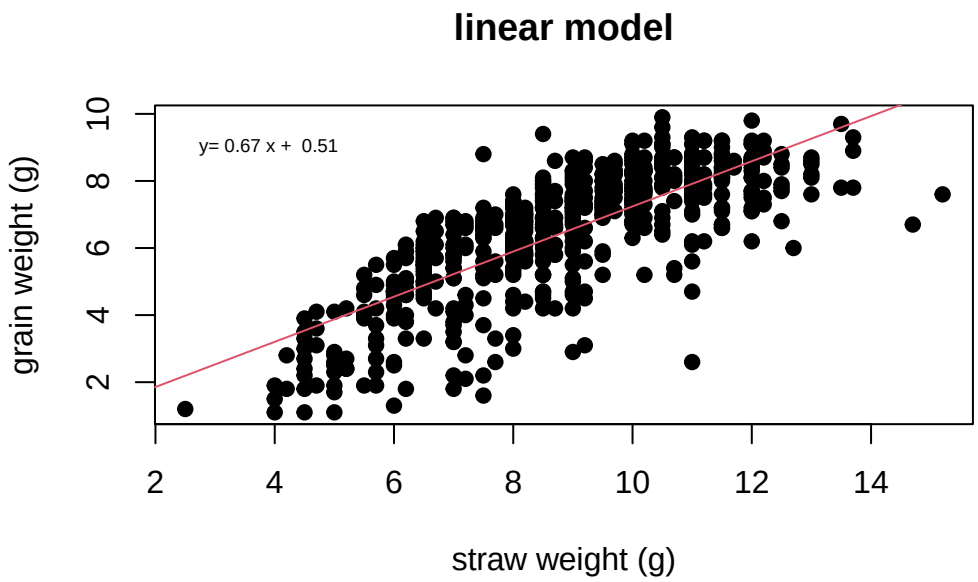
7.2 Plotting linear models

Linear models can be plotted in base R using the same syntax as `lm()`. You can add the regression line directly onto the plot using `abline(lm())`. In ggplot, you can use `stat_smooth()` or `geom_smooth()` with `method = 'lm'` to plot a linear model.

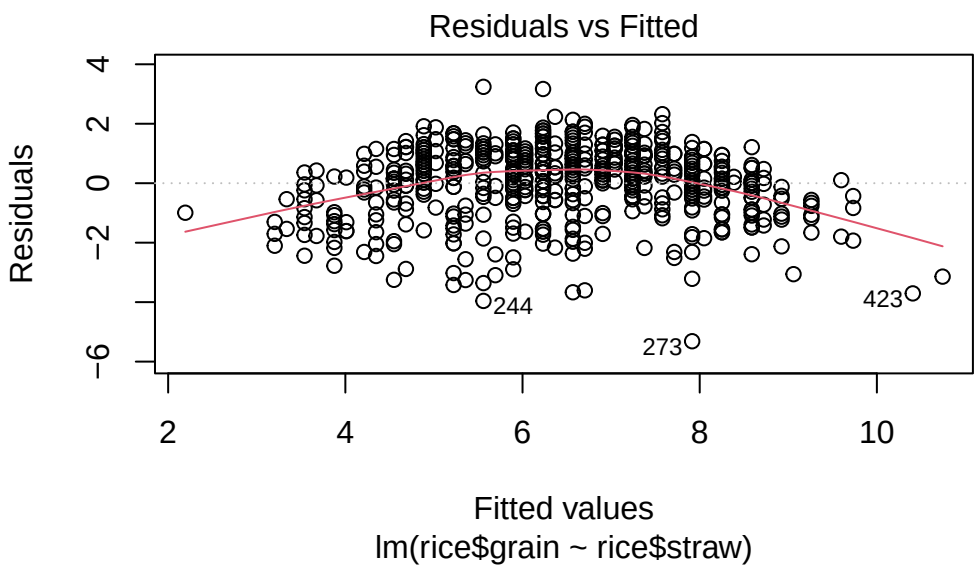
```r
# Let's see this things in a plot:
plot(rice$grain ~ rice$straw, pch = 19, cex = 0.2)
```

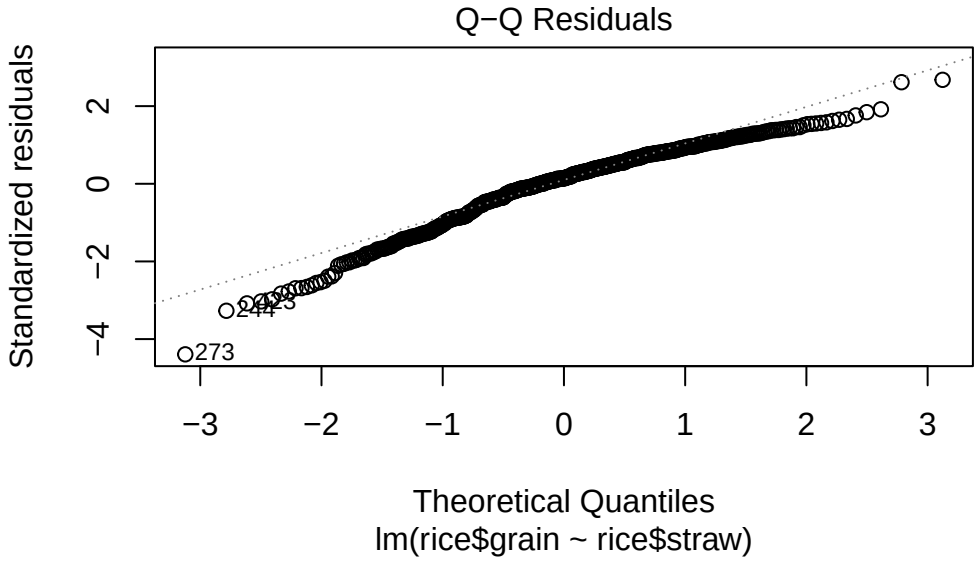


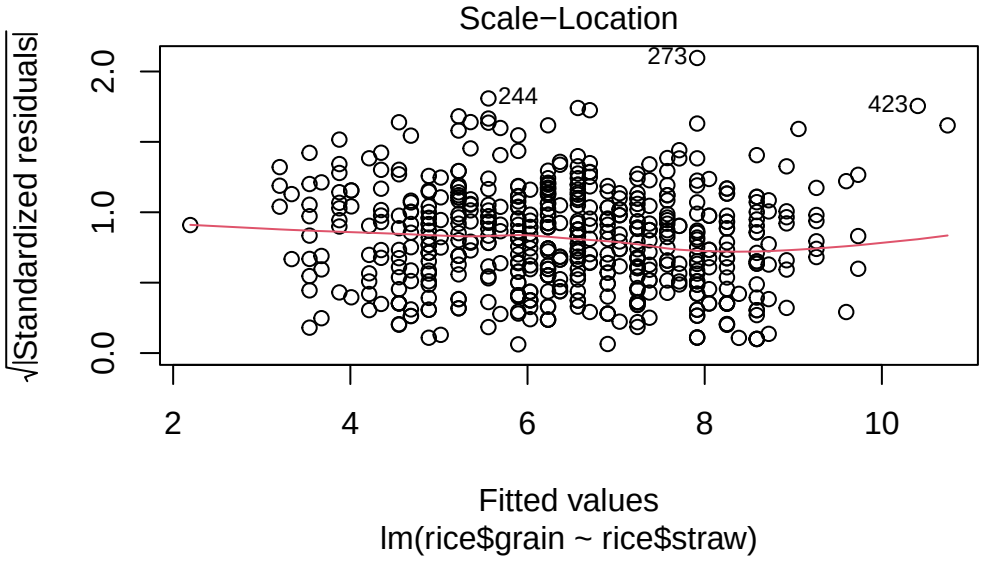

```
# Ok, we can do it a bit better:
plot(rice$grain ~ rice$straw, pch = 19,  cex = 1,
     ylab="grain weight (g)",
     xlab="straw weight (g)",
     main="linear model")

# adding the model
abline(rice_mod, col = 2)# adds the linear function from the model
# adding the equation


#first "write it" (paste joins bits of characters)
# y = mx + b
eqn <-paste("y=", round(rice_mod$coefficients[2], 2),
            "x + ", round(rice_mod$coefficients[1], 2) )
text(3.9,9, eqn, cex = 0.6)# this adds text to the graph
```

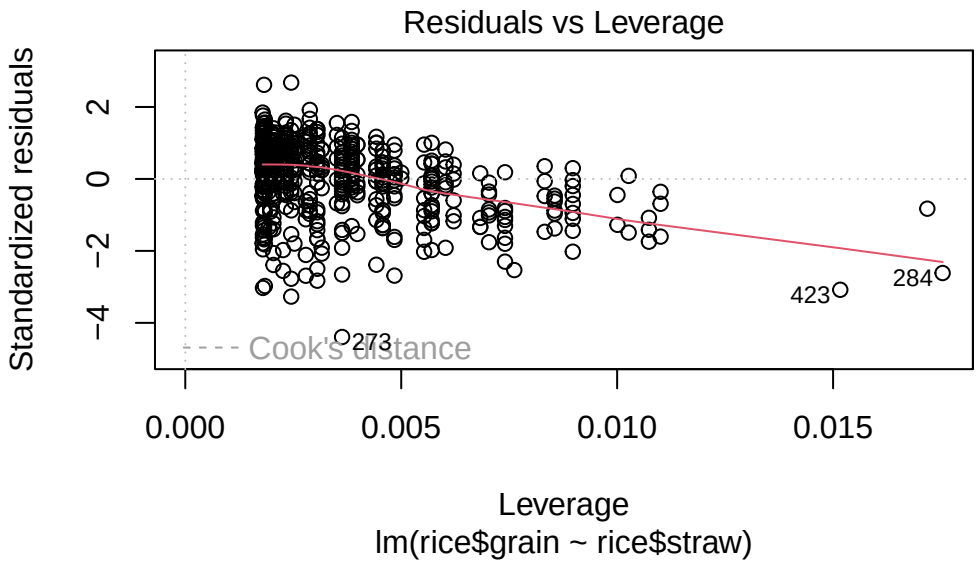



```
# We can also take a look in the model premisses
plot(rice_mod)# here you have to hit enter for new graphs to
```

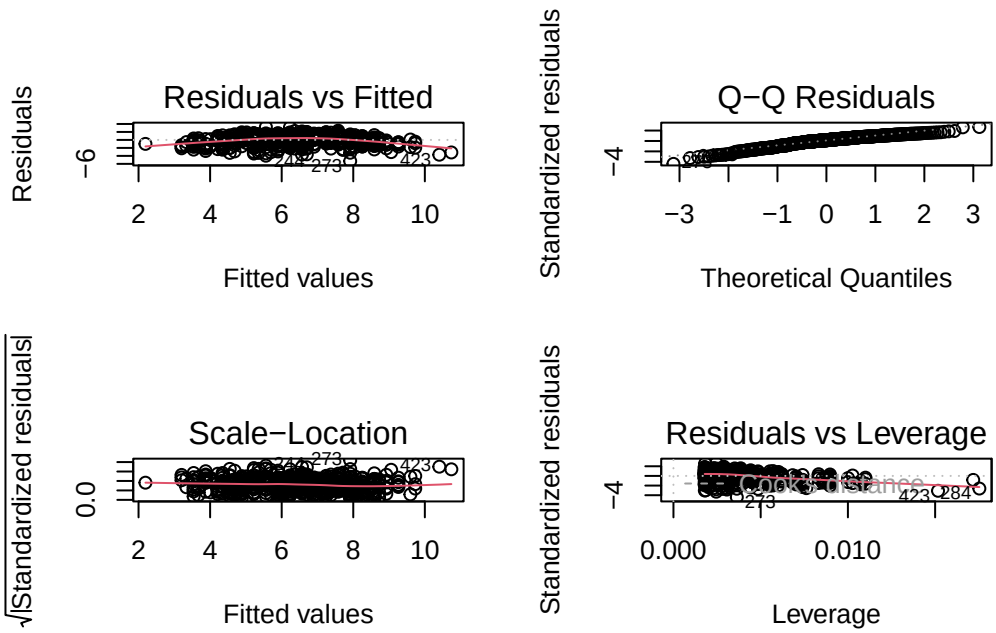



Q−Q Residuals
Standardized residuals
2
0
−2
−4
244
423
273
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(rice$grain ~ rice$straw)


Scale−Location
√|Standardized residuals|
2.0
1.0
0.0
273
244
423
2
4
6
8
10
Fitted values
lm(rice$grain ~ rice$straw)

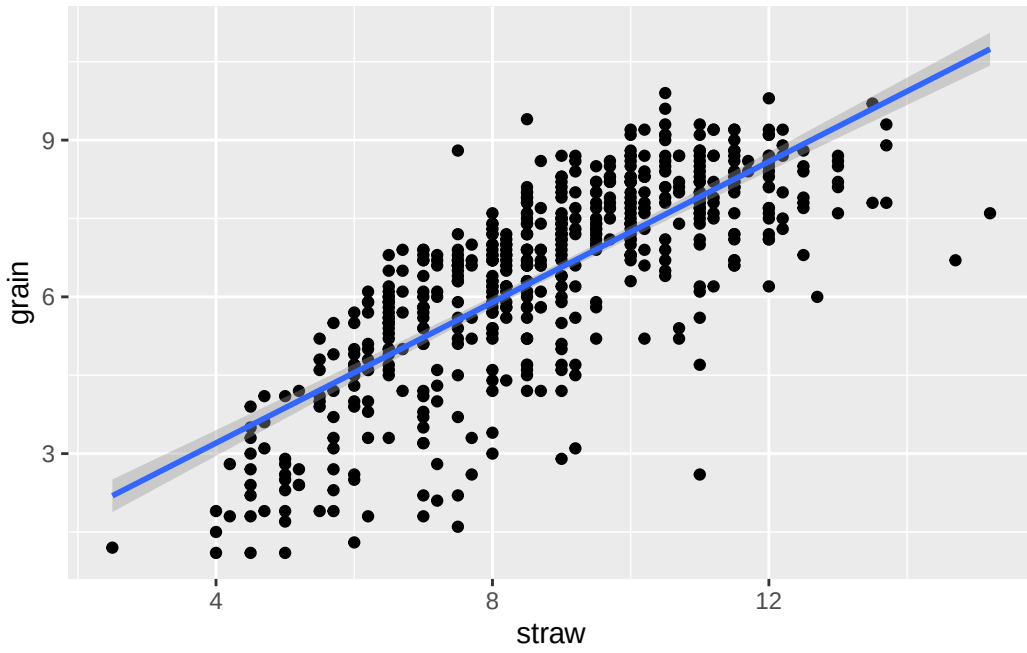



```
#show up

par(mfrow=c(2,2))
plot(rice_mod)# here all graphs are plotted together
```




```
# ggplot
library(ggplot2)
ggplot(rice, aes(x = straw, y = grain)) + geom_point() + stat_smooth(method = 'lm')
```

```
`geom_smooth()` using formula = 'y ~ x'
```




```
# We can use other kinds of models
#(e.g. General Linear models, Additive linear models etc).
# they all mostly have similar formats to the basic lm()
```

8 R Refresher Part 8: Programming Fundamentals

Created by Mauricio Cantor, with modifications by Laura J. Feyrer, Ana Eguiguren, and Reid Steele

This section covers basic programming fundamentals in R, such as loops, user-defined functions, and if/else statements.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

8.1 User-defined functions

In addition to the functions included in base R and in packages, you can also create your own functions to help streamline and share your code (or even make your own package!). Functions are created using the `function()` command, with the syntax `function(arguments){code}`. Functions will refer to their arguments as variables internal to them, even if they do not exist in the environment. If they do exist in the environment, the function will prioritize its arguments before the global environment. Functions should almost always end in the `return()` command, which tells the function what to output when it is run.

```
# 8. I. Programming fundamentals - Functions -----------------------------------


# User-defined functions: function(){}
# To create your own function, again, the recipe is the same:
# create another object using pre-defined functions. What?
# We will use a function called "function()" (boring, I know, but effective!).
# For example, a very simple, but limited, function could be the one that sums 5 input values
# We will call it 'sum_5_numbers'



sum_5_numbers <- function(value1, value2, value3, value4, value5){
```

```r
  object <- value1 + value2 + value3 + value4 + value5

  return(object)
}

# What happened? R saved our function in the workspace
# Note the structure: we give it a name, arguments (value1 to 5). It has internal objects (ob

sum_5_numbers(value1=10, value2=20, value3=30, value4=40, value5=50)
```

```
[1] 150
```

```r
sum_5_numbers(3,6,90,1,2)
```

```
[1] 102
```

```r
# Sweet! It did his job: summed 10+20+30+40+50.


# use this for any formula that you apply frequently
# ex: the vertical distance travelled by a free falling
# object after x seconds
d <- 1/2 *9.8 * 5^2
d
```

```
[1] 122.5
```

```r
d2 <- 1/2 *9.8 * 2^2

dist_fall <- function(time_sec){
  d <- 1/2 *9.8 * time_sec^2
  return(d)
}

dist_fall(20)
```

```
[1] 1960
```

```r
#~~~~ A bit more complex functions----
# It is good to have a workflow:
# 1. design the steps of your process
# 2. identify the inputs of your function
# 3. change specific inputs to generic names
# 4. add process into the body of your function




# I want to simulate rolling a 6-faced dice once:
dice <- seq(1:6)
sample(dice, 1)
```

```
[1] 5
```

```r
# What about twice?
sample(dice, 2, replace = T)
```

```
[1] 6 6
```

```r
# How can I generalize this?
n_rolls <- 7
sample(dice, n_rolls, replace = T)
```

```
[1] 2 2 4 6 6 6 4
```

```r
# Put it in a function
dice_rolling <- function(n_rolls){
  results<-sample(dice, n_rolls, replace = T)
  return(results)
}

dice_rolling(100)
```

```
  [1] 6 6 1 6 6 4 4 1 5 2 3 6 3 6 1 2 1 6 3 6 3 1 3 3 4 6 6 2 3 3 6 1 4 2 6 2 1
 [38] 5 6 2 2 4 2 3 2 4 3 6 1 6 4 3 6 6 2 4 4 1 6 4 6 6 3 1 2 5 5 4 1 6 2 3 2 3
 [75] 3 4 2 3 5 6 4 3 2 4 2 2 1 5 3 3 2 5 4 1 2 4 3 1 2 5
```

```r
# A function for a rolling dice in a game
# in which you get 10 dollars each time you roll a six
n_rolls <- 8
win_numb <- 6
price <- 0.10

dice_rolling_money <- function(n_rolls, win_numb, price){
  results <-sample(dice, n_rolls, replace = T)
  money <- sum(results == win_numb)*price
  return(list(results,money))
}


dice_rolling_money(5, 3, 50)
```

```
[[1]]
[1] 4 6 1 2 1

[[2]]
[1] 0
```

8.2 for loops

`for()` loops are coding structures designed to repeatedly execute code over a range. Their syntax is `for(index in range){code}`. When run, the `for()` loop will repeat the code with the value of the index variable (typically denoted `i`, but it can be anything) set to every value contained in the range.

```r
# 8. II. Programming fundamentals - For Loops ----------------------------------
# used to repeat an action for i number of times
# imagine I want to write sentence: today is (weekday)
# for each of the week.
# Initially you could try:


# first making a vector that contains all weekdays

weekday <- c("Monday", "Tuesday", "Wednesday", "Thursday",
             "Friday", "Saturday", "Sunday")
```

```r
# then pasting it to "today is" for each one:
paste("Today is", weekday[1])
```

```
[1] "Today is Monday"
```

```r
# and I'd have to copy and paste this for each day of the week:
paste("Today is", weekday[2])
```

```
[1] "Today is Tuesday"
```

```r
paste("Today is", weekday[3])
```

```
[1] "Today is Wednesday"
```

```r
paste("Today is", weekday[4])
```

```
[1] "Today is Thursday"
```

```r
# etc...

# When coding, we try to avoid copying and pasting things
# more than twice because there is usually a more efficient
# way of doing this.
# One option is to use a for loop:

for(i in seq(1:7)){
    print(paste("Today is", weekday[i]))
}
```

```
[1] "Today is Monday"
[1] "Today is Tuesday"
[1] "Today is Wednesday"
[1] "Today is Thursday"
[1] "Today is Friday"
[1] "Today is Saturday"
[1] "Today is Sunday"
```

```r
# you can also save each of your results in a vector
weekd_sent <- vector(length = 7)

for(i in seq(1:7)){
  weekd_sent[i] <- (paste("Today is", weekday[i]))
}

weekd_sent
```

```
[1] "Today is Monday"    "Today is Tuesday"   "Today is Wednesday"
[4] "Today is Thursday"  "Today is Friday"    "Today is Saturday"
[7] "Today is Sunday"
```

```r
# if you don't know the sequence lenth you can use the argument
# seq_along()
day_sentence <- vector(length = length(weekday)) # create an empty vector

for(i in seq_along(day_sentence)){
  day_sentence[i]<- paste("Today is", weekday[i])#carries operation for ith day
}

day_sentence
```

```
[1] "Today is Monday"    "Today is Tuesday"   "Today is Wednesday"
[4] "Today is Thursday"  "Today is Friday"    "Today is Saturday"
[7] "Today is Sunday"
```

```r
# RS Note: Some alternative ideas for the same thing
day_sentence <- NULL

for(i in 1:length(weekday)){
  day_sentence <- c(day_sentence, paste("Today is", weekday[i]))#carries operation for ith da
}

day_sentence
```

```
[1] "Today is Monday"    "Today is Tuesday"   "Today is Wednesday"
[4] "Today is Thursday"  "Today is Friday"    "Today is Saturday"
[7] "Today is Sunday"
```

```r
# This is very useful for processing files and making plots:
# create an empty list to save each plot in a slot
library(dplyr) # package for filtering data
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(ggplot2) #package for plotting

data(iris) # load iris dataset that comes in base R
head(iris) # see what is in there
```

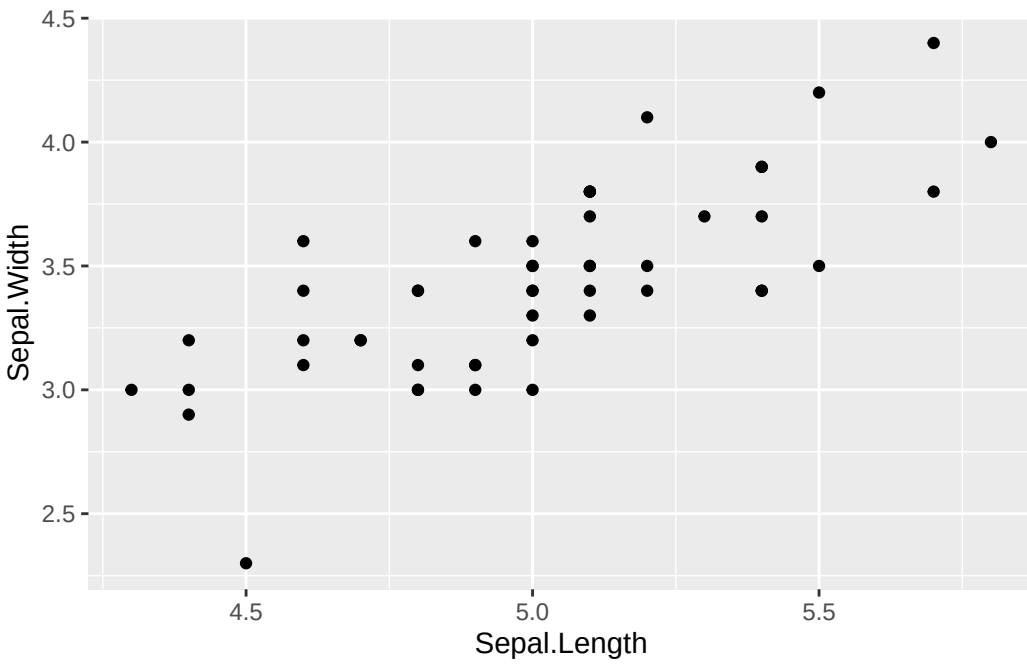
```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```r
# I want to make one graph for each species:

levels(iris$Species) # how many species are there?
```

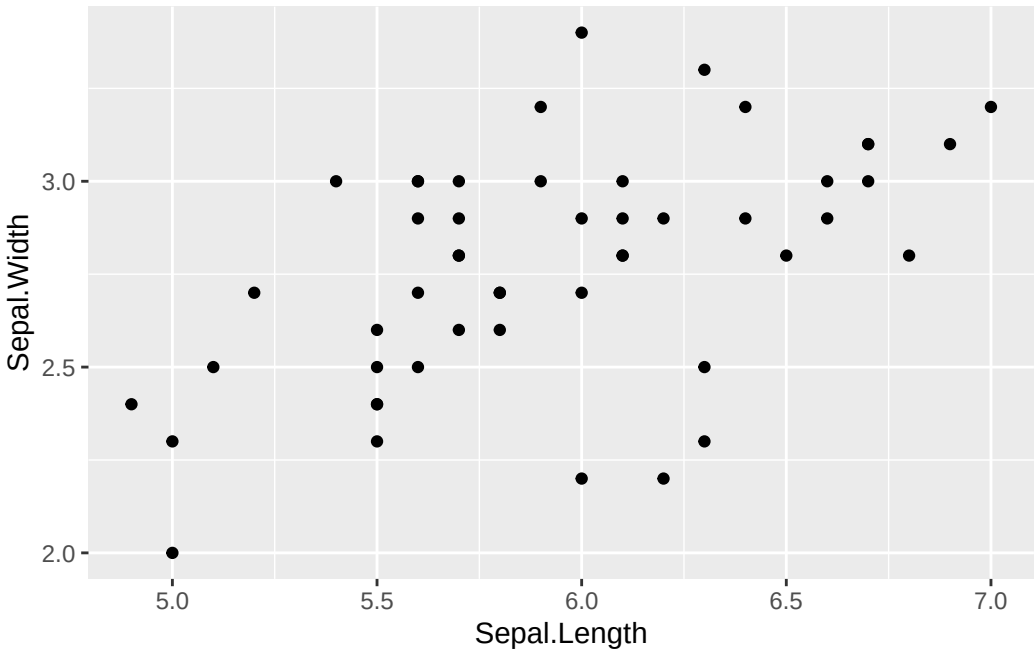
```
[1] "setosa"     "versicolor" "virginica"
```

```r
# create list with slot for each plot = species
p <- vector("list", length = length(levels(iris$Species)))


for(i in seq(1:3)){
```

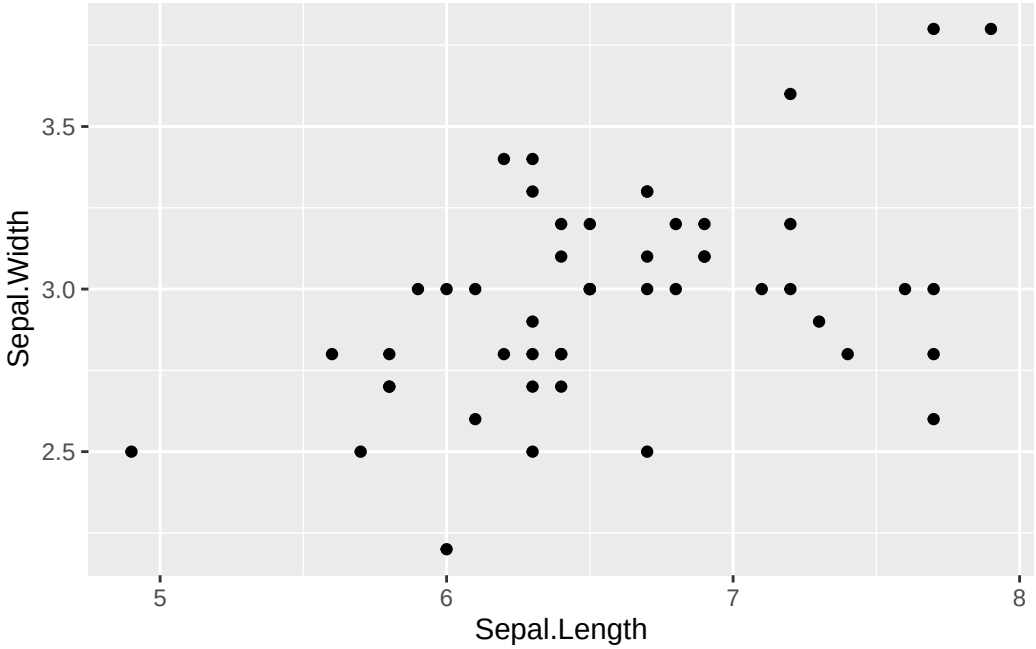
```
  subset_species <- iris %>% filter(Species == levels(iris$Species)[i])
  p[[i]]<- ggplot(subset_species,
         aes(x = Sepal.Length, y = Sepal.Width))+
        geom_point()

}

plot(p[[1]])
```




```
plot(p[[2]])
```


3.0
2.5
2.0
Sepal.Width
5.0
5.5
6.0
6.5
7.0
Sepal.Length

```
plot(p[[3]])
```


3.5
3.0
2.5
Sepal.Width
5
6
7
8
Sepal.Length

8.3 if, else, and ifelse

`if()` statements are coding structures which are designed to perform an action only if a certain condition is met. Their syntax is `if(conditon){action}`, where the action is only carried out if the condition is `TRUE`. `else` is an extension of `if` which provides an alternate action to perform if the condition is false, using the syntax `if(conditon){action} else {alternative action}`. `ifelse()` is a function which compresses `if` and `else` together to be applied to vectors. Its syntax is `ifelse(conditon, value if true, value if false)`, where the condition must be a vector.

```
# 8. III. Programming fundamentals - If Else  -----------------------------------
# Perhaps you want parts of you code to run only when
# satisfying some conditions. We then use a logical test.
# if(): if(condition=true) do_command

if(1 == 0) print("what?")
if(1 != 0) print("OK!")
```

```
[1] "OK!"
```

```
# ifelse(): if(condition=true, do_command1, do_command2_instead)
ifelse(1 == 0, print("no way!"), print("OK!"))
```

```
[1] "OK!"

[1] "OK!"
```

```
ifelse(1 != 0, print("all right!"), print("what?!"))
```

```
[1] "all right!"

[1] "all right!"
```

```
# RS Note - ifelse is vectorized, while if a scalar

# if{}else{}: if(condition=true) {do_command1} else{do_command2_instead}
# this is useful for long conditions or commands
if (1 == 0 ){
  print("are you crazy?")
```

```r
} else {
  print("OK!")
}
```

```
[1] "OK!"
```

```r
# within a function:
# Going back to the dice game, we can create similar game
# in which you win if you roll the same number
# two consecutive times
# to keep it simple, we will only do this for 2 rolls

#first I'll make
n_rolls <- 2

dice_rolling_pair <- function(n_rolls){
  dice <- seq(1:6)
  results <-sample(dice, n_rolls, replace = T)
  if(results[1]==results[2]){print("win")}else{
    print("loose")
  }
}

dice_rolling_pair(2)
```

```
[1] "loose"
```

9 R Refresher Part 9: Useful tidyr and dplyr functions

Created by Reid Steele

This section covers some useful functions from the tidyverse family, belonging to the `tidyr` and `dplyr` packages.

NOTE: On refresher pages, some code lines will be commented out to avoid file structure issues surrounding saving files, downloading packages, and changing working directories.

9.1 Joins

Joins are useful for combining two data frames which have a common column. These are particularly useful when working with databases, which often have separate data and metadata files.

```
##############################################################################################

# 9. RS Section: Useful tidyr and dplyr functions
library(tidyr)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
# Joining
band_members
```

```
# A tibble: 3 x 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

```r
band_instruments
```

```
# A tibble: 3 x 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```

```r
# left_join matches values from first data frame and merges second data frame into it
left_join(band_members, band_instruments, by = 'name')
```

```
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
```

```r
# right_join does the same but in reverse
right_join(band_members, band_instruments, by = 'name')
```

```
# A tibble: 3 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
3 Keith <NA>    guitar
```

```r
# full_join fully joins both data frames
full_join(band_members, band_instruments, by = 'name')
```

```
# A tibble: 4 x 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```

9.2 Wide and long form data

Data is generally stored in one of two forms - long format, and wide format.

Wide format is likely the form you are used to. In wide format data, columns are used to distinguish different types of data. Most data you've seen or worked with in Excel is likely in wide format. It is called wide format because you have many columns and fewer rows.

Long format data compresses all of the actual data values together into a single column, with different groups of data being distinguished with metadata identifiers. This makes a data structure with many rows and few columns - hence, long format. Long format data is common in databases, since it is much easier to add new rows to data structures than it is to add new columns.

You can switch between long and wide format data using the `pivot_longer` and `pivot_wider` `tidyr` functions. You may wish to do this because generally, base R functions and plotting work better with wide format data, while tidyverse and ggplot2 work better (or more often, exclusively) on long format data.

```r
# Wide and long form data:
# Most data you have seen or worked with is likely in what is called 'wide format'
# Wide format data is laid out in a grid format, and data is contained in multiple columns
# Long format data compresses all data into a single column, which is defined by several meta
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

```
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
# the mtcars dataset is in wide format - each individual data type has its own column, into w

# pivot_longer changes data from wide format to long format
mtcars_wide = cbind(car = rownames(mtcars), mtcars) # Make row names its own column so we car
mtcars_wide
```

```
                                  car  mpg cyl  disp  hp drat    wt  qsec vs
Mazda RX4                     Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0
Mazda RX4 Wag             Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0
Datsun 710                   Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1
Hornet 4 Drive           Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1
Hornet Sportabout     Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0
Valiant                         Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1
Duster 360                   Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0
Merc 240D                     Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1
Merc 230                       Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1
Merc 280                       Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1
Merc 280C                     Merc 280C 17.8   6 167.6 123 3.92 3.440 18.90  1
Merc 450SE                   Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0
Merc 450SL                   Merc 450SL 17.3   8 275.8 180 3.07 3.730 17.60  0
Merc 450SLC                 Merc 450SLC 15.2   8 275.8 180 3.07 3.780 18.00  0
Cadillac Fleetwood   Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0
Lincoln Continental Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0
Chrysler Imperial     Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0
Fiat 128                       Fiat 128 32.4   4  78.7  66 4.08 2.200 19.47  1
Honda Civic                 Honda Civic 30.4   4  75.7  52 4.93 1.615 18.52  1
Toyota Corolla           Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1
Toyota Corona            Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1
Dodge Challenger       Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0
AMC Javelin                 AMC Javelin 15.2   8 304.0 150 3.15 3.435 17.30  0
Camaro Z28                   Camaro Z28 13.3   8 350.0 245 3.73 3.840 15.41  0
Pontiac Firebird       Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0
Fiat X1-9                     Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1
Porsche 914-2             Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0
Lotus Europa               Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1
Ford Pantera L           Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0
Ferrari Dino               Ferrari Dino 19.7   6 145.0 175 3.62 2.770 15.50  0
Maserati Bora             Maserati Bora 15.0   8 301.0 335 3.54 3.570 14.60  0
Volvo 142E                   Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1
```

```
                    am gear carb
Mazda RX4            1    4    4
Mazda RX4 Wag        1    4    4
Datsun 710           1    4    1
Hornet 4 Drive       0    3    1
Hornet Sportabout    0    3    2
Valiant              0    3    1
Duster 360           0    3    4
Merc 240D            0    4    2
Merc 230             0    4    2
Merc 280             0    4    4
Merc 280C            0    4    4
Merc 450SE           0    3    3
Merc 450SL           0    3    3
Merc 450SLC          0    3    3
Cadillac Fleetwood   0    3    4
Lincoln Continental  0    3    4
Chrysler Imperial    0    3    4
Fiat 128             1    4    1
Honda Civic          1    4    2
Toyota Corolla       1    4    1
Toyota Corona        0    3    1
Dodge Challenger     0    3    2
AMC Javelin          0    3    2
Camaro Z28           0    3    4
Pontiac Firebird     0    3    2
Fiat X1-9            1    4    1
Porsche 914-2        1    5    2
Lotus Europa         1    5    2
Ford Pantera L       1    5    4
Ferrari Dino         1    5    6
Maserati Bora        1    5    8
Volvo 142E           1    4    2
```

```r
# pivot mtcars_wide to long format
metadata_cols = colnames(mtcars_wide)[colnames(mtcars_wide) != 'car'] # Separate out metadata
mtcars_long = pivot_longer(mtcars_wide,
                           cols = all_of(metadata_cols),
                           names_to = 'variable', values_to = 'value') # Pivot longer with me
head(mtcars_long)
```

```
# A tibble: 6 x 3
```

```
  car       variable  value
  <chr>     <chr>     <dbl>
1 Mazda RX4 mpg       21
2 Mazda RX4 cyl        6
3 Mazda RX4 disp     160
4 Mazda RX4 hp       110
5 Mazda RX4 drat       3.9
6 Mazda RX4 wt         2.62
```

```r
# Data is now in long format - the variable name is contained in the variable column, and the

# pivot back to wide format
pivot_wider(mtcars_long, names_from = variable, values_from = value)
```

```
# A tibble: 32 x 12
   car           mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 Mazda RX4    21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2 Mazda RX4 ~  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3 Datsun 710   22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4 Hornet 4 D~  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5 Hornet Spo~  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6 Valiant      18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7 Duster 360   14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8 Merc 240D    24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9 Merc 230     22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10 Merc 280     19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# i 22 more rows
```

```r
# ggplot typically requires long format data, whereas something like apply may require wide f
```

9.3 Pipes

The `%>%` operator is a special operator which can be used to chain multiple functions together. These chains are known as pipes. Pipes are very useful when combined with various `tidyverse` functions in order to quickly and easily summarize data and run calculations on data frames. This is a very powerful data analysis tool, which has essentially replaced `apply()` for many people. Take a look at some of the examples below.

```r
# dplyr pipes
# The %>% operator in dplyr/tidyr passes data to the next function

# filter filters logicals, similar to indexing
mtcars_long %>% filter(variable == 'cyl')
```

```
# A tibble: 32 x 3
   car               variable value
   <chr>             <chr>    <dbl>
 1 Mazda RX4         cyl          6
 2 Mazda RX4 Wag     cyl          6
 3 Datsun 710        cyl          4
 4 Hornet 4 Drive    cyl          6
 5 Hornet Sportabout cyl          8
 6 Valiant           cyl          6
 7 Duster 360        cyl          8
 8 Merc 240D         cyl          4
 9 Merc 230          cyl          4
10 Merc 280          cyl          6
# i 22 more rows
```

```r
# This is the same as:
filter(mtcars_long, variable == 'cyl')
```

```
# A tibble: 32 x 3
   car               variable value
   <chr>             <chr>    <dbl>
 1 Mazda RX4         cyl          6
 2 Mazda RX4 Wag     cyl          6
 3 Datsun 710        cyl          4
 4 Hornet 4 Drive    cyl          6
 5 Hornet Sportabout cyl          8
 6 Valiant           cyl          6
 7 Duster 360        cyl          8
 8 Merc 240D         cyl          4
 9 Merc 230          cyl          4
10 Merc 280          cyl          6
# i 22 more rows
```

```r
# %>% operators can be chained together
# summarize calculates statistics
mtcars_long %>% filter(variable == 'cyl') %>%
  summarize(mean = mean(value))
```

```
# A tibble: 1 x 1
   mean
  <dbl>
1  6.19
```

```r
# This code filters to cyl and then calculates the mean

# mutate adds columns
mtcars_long %>% filter(variable == 'cyl') %>%
  summarize(mean = mean(value)) %>%
  mutate(stat = 'mean')
```

```
# A tibble: 1 x 2
   mean stat
  <dbl> <chr>
1  6.19 mean
```

```r
# group_by groups by an identifying column or set of identifying columns for all subsequent a
mtcars_long %>% group_by(variable) %>% # Groups the data by variable
  summarize(mean = mean(value), sd = sd(value), n = n()) %>% # Calculates mean, standard devi
  filter(variable != 'cyl') %>% # removes rows where variable == cyl
  mutate(data_is = 'cars', CV = (sd/mean)*100) %>% # Adds a column that says data is from car
  select(data_is, variable, CV) # Select chooses and reorders requested columns
```

```
# A tibble: 10 x 3
   data_is variable    CV
   <chr>   <chr>    <dbl>
 1 cars    am       123.
 2 cars    carb      57.4
 3 cars    disp      53.7
 4 cars    drat      14.9
 5 cars    gear      20.0
 6 cars    hp        46.7
 7 cars    mpg       30.0
 8 cars    qsec      10.0
 9 cars    vs       115.
10 cars    wt        30.4
```

```
################################################################################
##### I hope you have enjoyed this tutorial on the basics of R.
##### I also hope that you take the journey to learn R, or other programming language.
##### Let me know if I can help out.
################################################################################
```

Part II

Assignment 1

10 Assignment 1a: Principal Components Analysis

Assignment 1a focuses on Principal Components Analysis (PCA). Think of PCA as a method of finding associations between data series.

For this tutorial, we're going to use the dataset in `fishcatch.csv`.

10.1 Looking at the Data

With any data analysis, step 1 is always to look at your data:

```
# Load in data
data = read.csv('fishcatch.csv')

# View data structure
head(data)
```

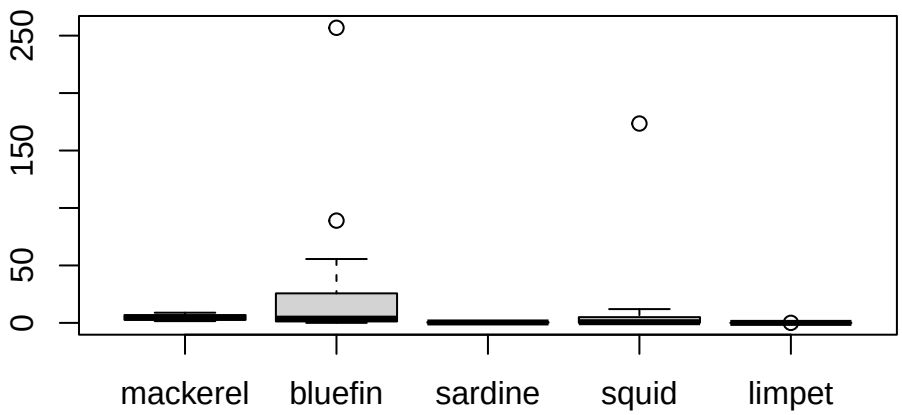
```
  Hauls mackerel bluefin sardine squid limpet
1     1    1.851   55.60   0.058  6.00 0.0004
2     2    1.925    1.20   0.252  0.08 0.0027
3     3    2.506    1.56   0.133  0.06 0.0015
4     4    1.537   30.00   0.064  9.35 0.0013
5     5    1.795    0.04   0.086  4.70 0.0022
6     6    3.371   45.00   0.078  7.66 0.0006
```

```
dim(data)
```

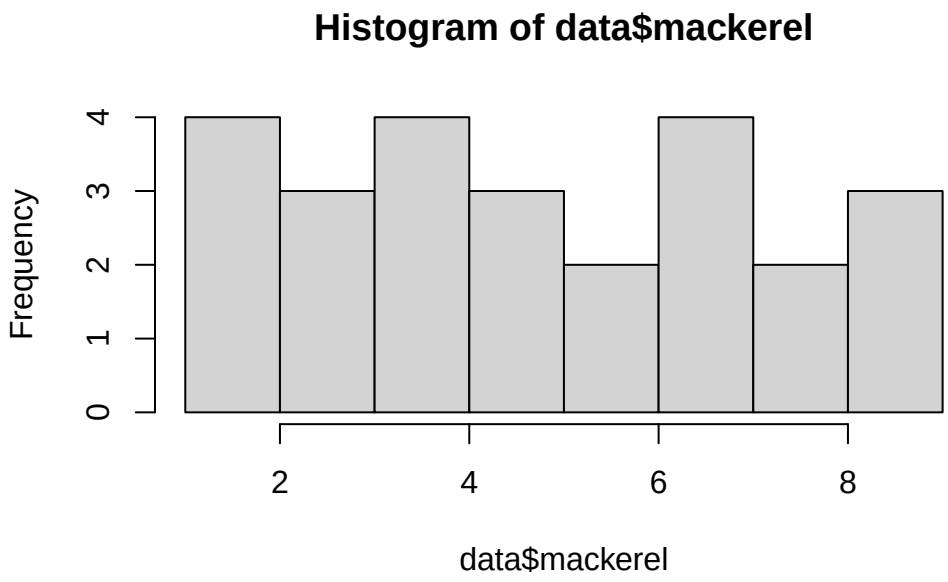
```
[1] 25  6
```

Our data is a 25 row, 6 column data frame, describing catch of 5 different fisheries species (columns 2-6) caught across 25 hauls (column 1). We want to know if certain species are associated with each other. Lets look a little deeper at the data:

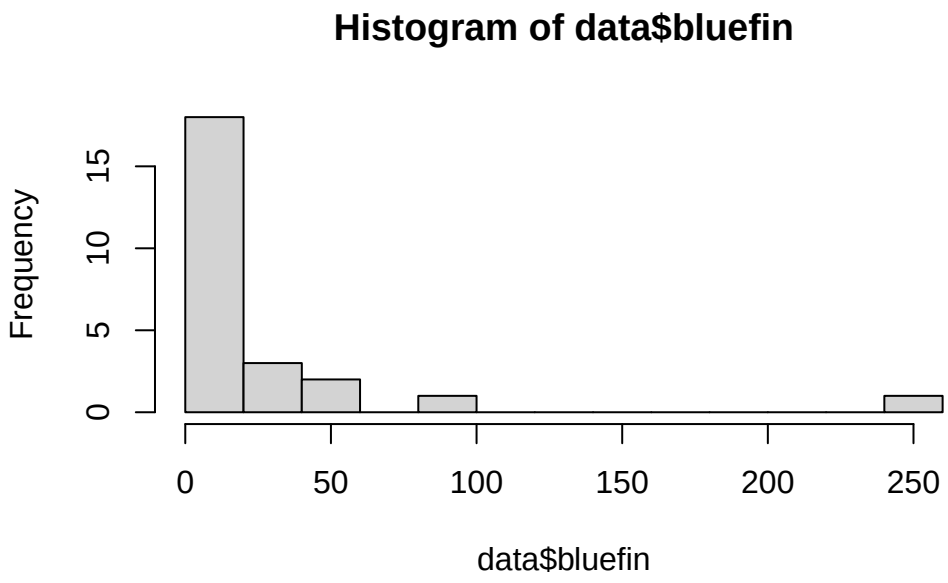
```r
# Generate boxplots
boxplot(data[,-1]) # Exclude haul
```



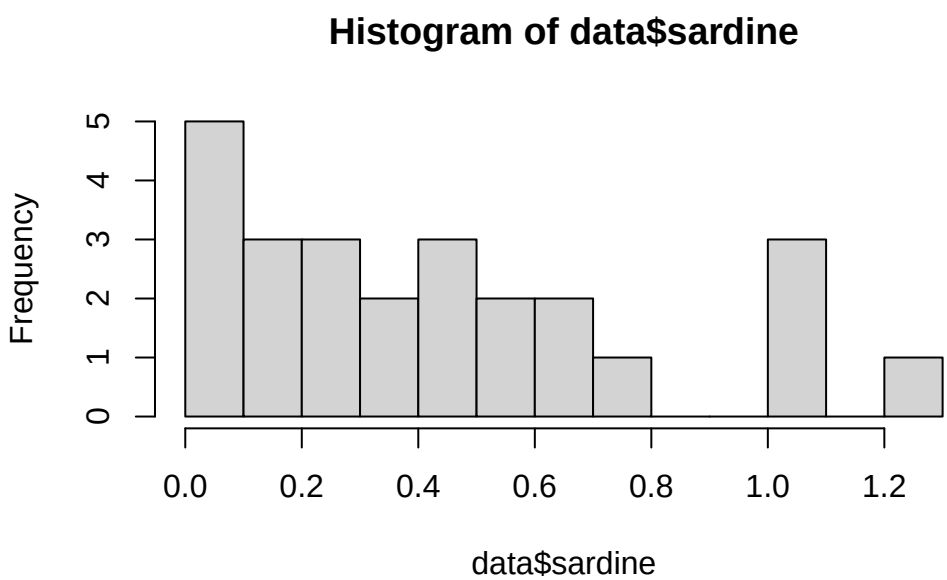

```r
# look at data distribution
# par(mfrow = c(3,2)) # 1 column 5 row grid plot
hist(data$mackerel, breaks = 10)
```



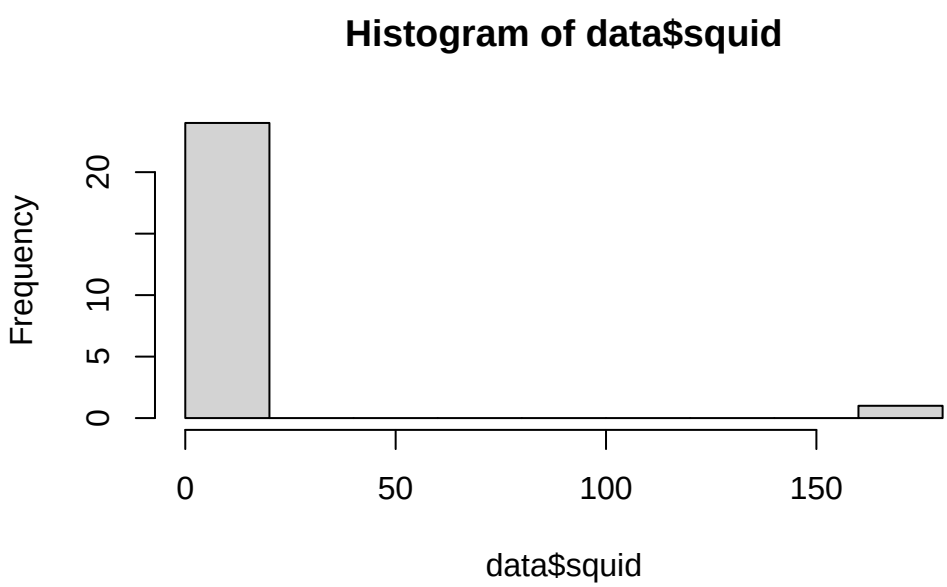

```r
hist(data$bluefin, breaks = 10)
```



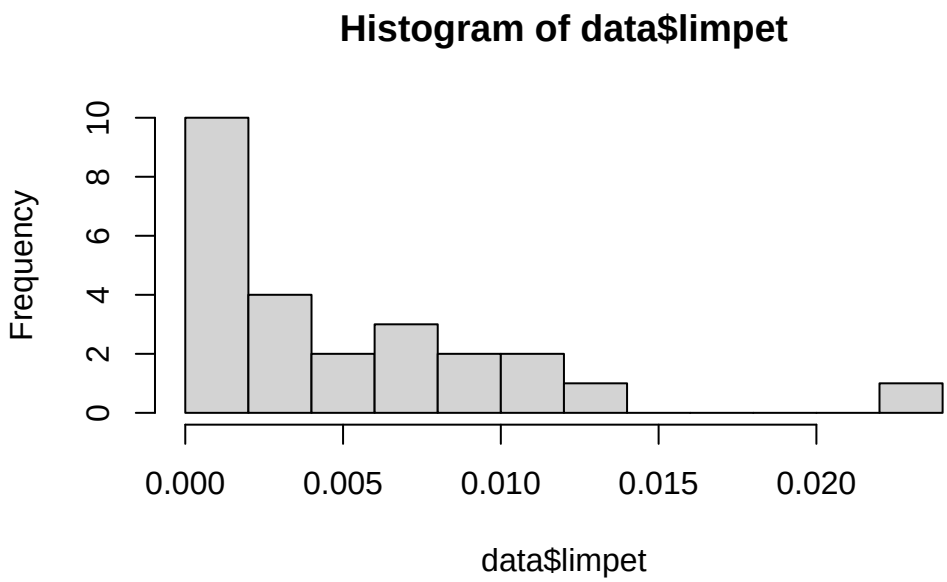
```r
hist(data$sardine, breaks = 10)
```




```r
hist(data$squid, breaks = 10)
```



```r
hist(data$limpet, breaks = 10)
```



A few things are immediately obvious from looking at our data:

1. There are some large outliers
2. The data scales vary greatly across species
3. The species all have relatively different distributions, none of which look normal.

Are these issues? How do we fix them?

10.2 Transformations

Look back at the PCA lecture. What are potential problems with PCA?

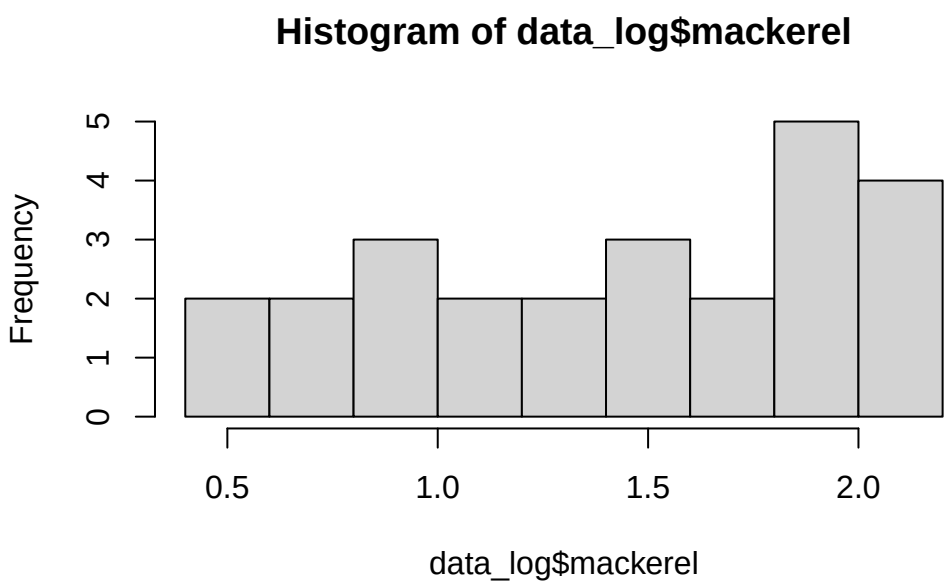
1. Covariance Matrix PCA requires data to be in the same units
2. Normality is desirable, but not essential
3. Precision is desireable, but not essential
4. Many zeroes in the data

We can fix issue 1 by logging our data:

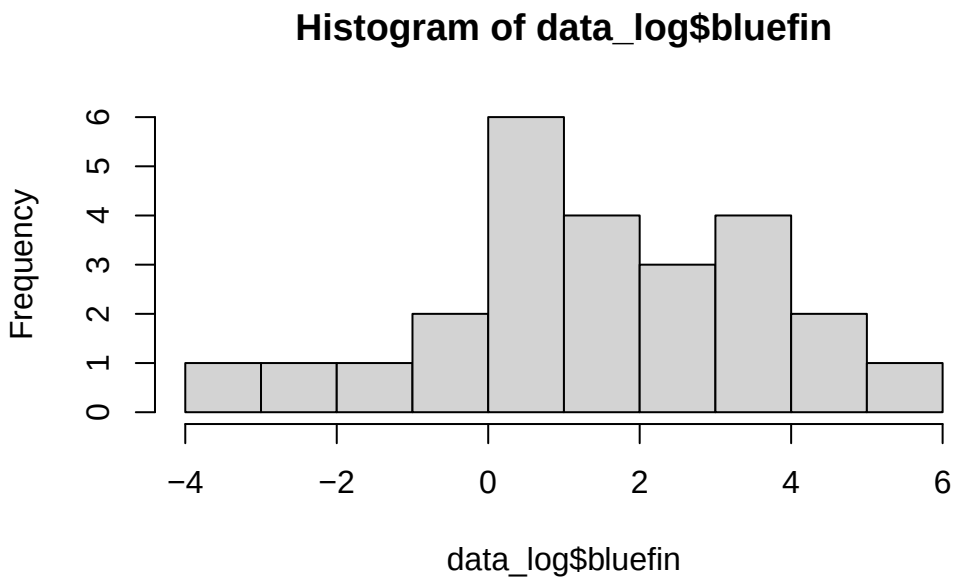
```r
# Create a new data object so we can log the data
data_log = data

# Log data
data_log[,-1] = log(data_log[,-1]) # Remember to exclude haul
```

Now that we've transformed the data, let's check for normality again:

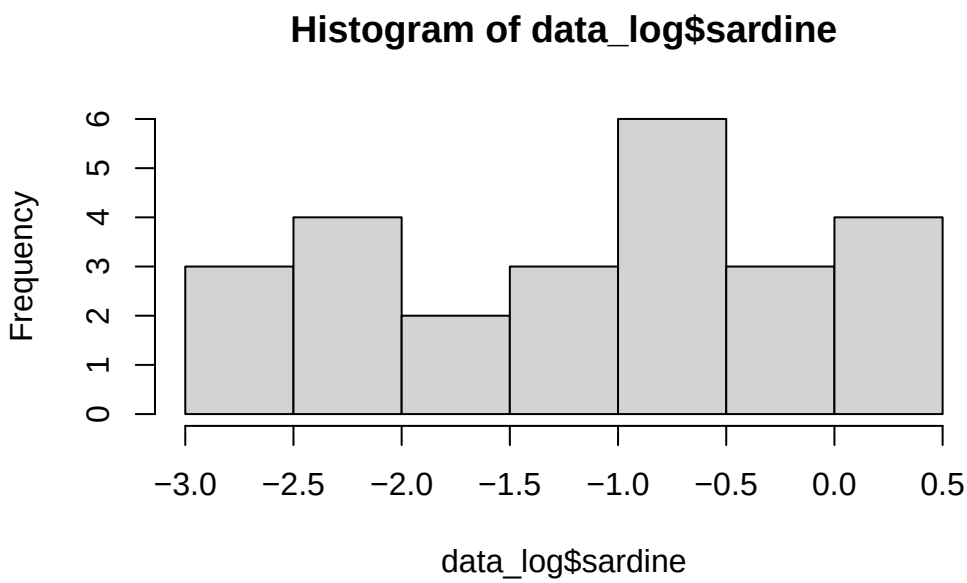
```r
# Generate histograms
# par(mfrow = c(3,2)) # 1 column 5 row grid plot
hist(data_log$mackerel, breaks = 10)
```



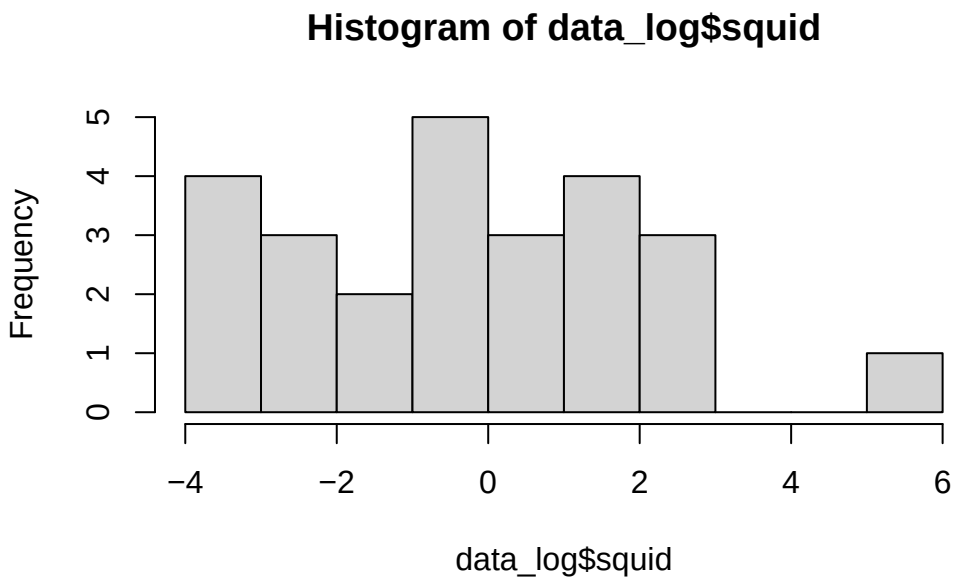
```r
hist(data_log$bluefin, breaks = 10)
```



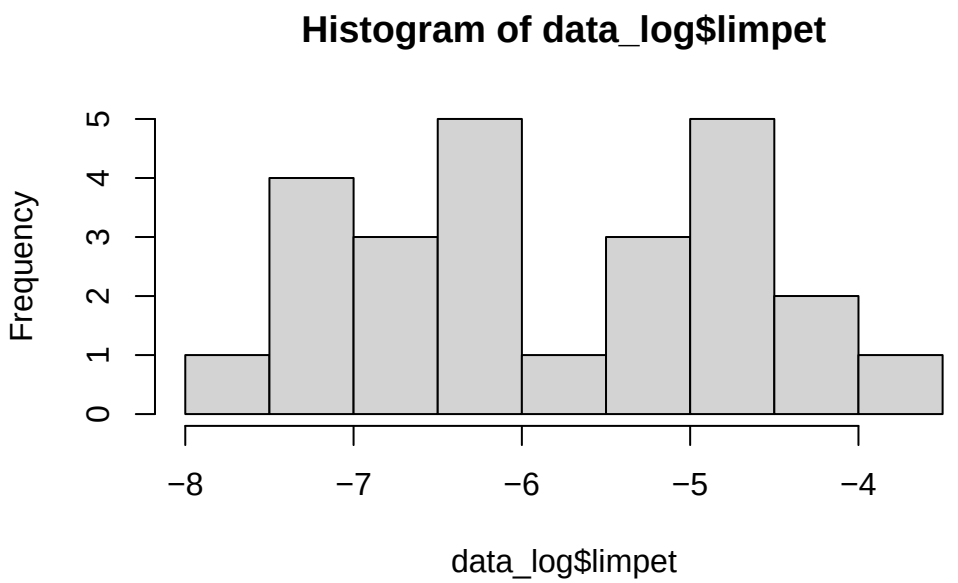

```r
hist(data_log$sardine, breaks = 10)
```



```r
hist(data_log$squid, breaks = 10)
```




```r
hist(data_log$limpet, breaks = 10)
```



These look much better. We can also confirm this statistically:

```r
# Generate histograms
shapiro.test(data_log$mackerel)
```

```
    Shapiro-Wilk normality test

data:  data_log$mackerel
W = 0.9425, p-value = 0.1691
```

```r
shapiro.test(data_log$bluefin)
```

```
    Shapiro-Wilk normality test

data:  data_log$bluefin
W = 0.98186, p-value = 0.9193
```

```r
shapiro.test(data_log$sardine)
```

```
    Shapiro-Wilk normality test

data:  data_log$sardine
W = 0.94113, p-value = 0.1572
```

```r
shapiro.test(data_log$squid)
```

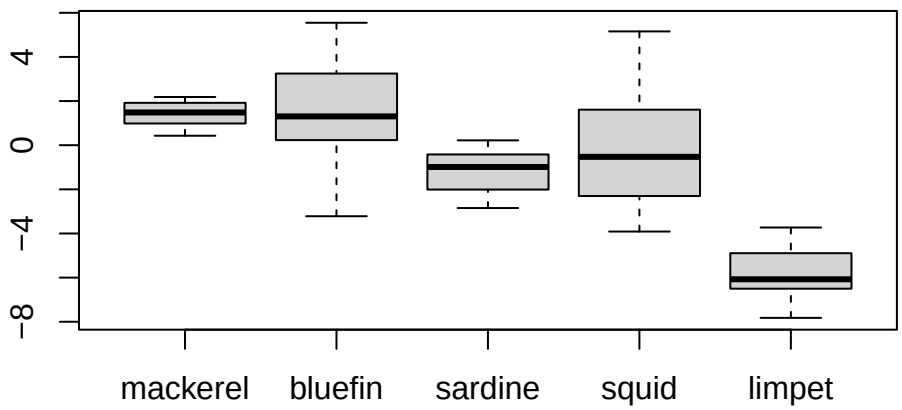
```
    Shapiro-Wilk normality test

data:  data_log$squid
W = 0.96226, p-value = 0.4613
```

```r
shapiro.test(data_log$limpet)
```

```
    Shapiro-Wilk normality test

data:  data_log$limpet
W = 0.96437, p-value = 0.5082
```

All 5 species fail to reject the null hypothesis that the data are normally distributed. Logging the data also helps deal with the outliers:

```
# Generate boxplots
boxplot(data_log[,-1])
```



Note that we can only log the data if there are no zeroes:

```
# Generate test data
data_test = data; data_test[1,6] = 0 # Change the first limpet value to 0

# Try to log the data
data_test[1,] # Print first row
```

```
  Hauls mackerel bluefin sardine squid limpet
1     1    1.851    55.6   0.058     6      0
```

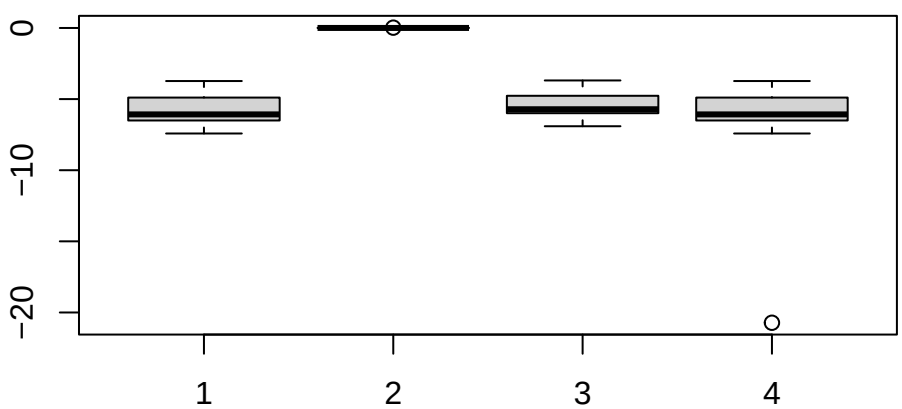
```r
log(data_test[,-1])[1,] # Print logs of the first row
```

```
  mackerel  bluefin   sardine    squid limpet
1 0.615726 4.018183 -2.847312 1.791759   -Inf
```

log(0) returnes negative infinity. That's going to be a problem later in our analysis. We can fix that by adding a small increment before taking the log. Keep in mind though that each species has a different magnitude in this dataset, and adding an inappropriate increment could cause us trouble later:

```r
# Test boxplots of different increments
boxplot(log(data_test$limpet), # Warning because of -Inf
        log(data_test$limpet + 1),
        log(data_test$limpet + 0.001),
        log(data_test$limpet + 0.000000001))
```

```
Warning in bplt(at[i], wid = width[i], stats = z$stats[, i], out =
z$out[z$group == : Outlier (-Inf) in boxplot 1 is not drawn
```



If the increment is too big, we eliminate the variance in our data. If the increment is to small, we create an outlier.

10.3 Running PCA

Now that we've checked and transformed our data, we're ready to run PCA. There are two kinds of PCA: We can run PCA on the Covariance Matrix, or the Correlation Matrix.

10.3.1 Covariance Matrix

We can run PCA on the covariance matrix as follows:

```
# Run PCA - Covariance
pca_1 = princomp(data_log[,-1]) # We don't want haul in our PCA!
summary(pca_1)
```

```
Importance of components:
                          Comp.1    Comp.2    Comp.3     Comp.4      Comp.5
Standard deviation     2.8055354 1.3857803 1.3351790 0.55247629 0.182937780
Proportion of Variance 0.6607195 0.1612035 0.1496458 0.02562199 0.002809263
Cumulative Proportion  0.6607195 0.8219229 0.9715687 0.99719074 1.000000000
```

Running a summary on our PCA gives us the standard deviation of each principal component, the proportion of variance explained by each principal component, and the cumulative variance explained as we add each component.

Each principal component is an **eigenvector** of the correlation/covariance matrix (remember from lecture that the jth principal component is the jth eigenvector of the correlation/covariance matrix). The **eigenvalues** are the variance of each individual principal component. The principal components are organized by their eigenvalues - the first principal component is the eigenvector with the largest eigenvalue, the second principal component is the eigenvector with the second largest eigenvalue, and so on.

The `princomp()` function gives us the standard deviation of each principal component; We can square these to get the eigenvalues:

```
# Calculate and pring eigenvalues
eigenvalues = pca_1$sdev^2
eigenvalues
```

```
    Comp.1     Comp.2     Comp.3     Comp.4     Comp.5
7.87102905 1.92038712 1.78270294 0.30523005 0.03346623
```

We can use the eigenvalues to reproduce the reset of the `princomp()` output. The proportion of variance is the value of each eigenvalue divided by the sum of all the eigenvalues. The cumulative proportion is the cumulative sum of the proportions of variance. Since there are 5 components, the cumulative proportion of component 5 is 1 (i.e. all of the variance).

```
# Calculate and print proportion of variance
prop_var = eigenvalues/sum(eigenvalues); prop_var
```

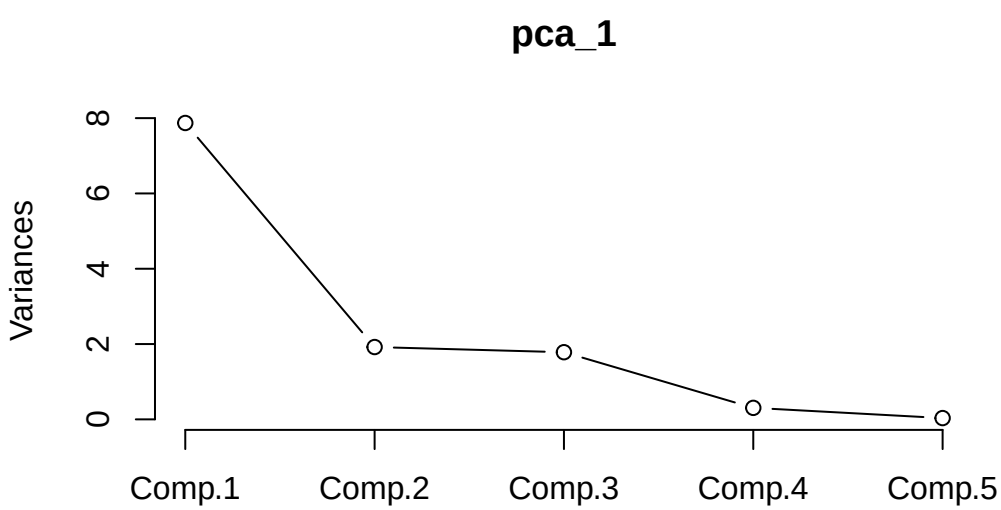
```
     Comp.1      Comp.2      Comp.3      Comp.4      Comp.5
0.660719468 0.161203465 0.149645813 0.025621991 0.002809263
```

```
# Calculate cumulative proportion of variance
cum_prop = cumsum(prop_var); cum_prop
```

```
   Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
0.6607195 0.8219229 0.9715687 0.9971907 1.0000000
```

Here, we see the first principal component explains 66% of the variance. The second explains 16%, which adds up to 82% with the first component, and so on up to component 5. We can visualize the cumulative variance explained with a scree plot:

```
# Generate scree plot
plot(pca_1, type = 'l') # Scree is built into the plot for PCA
```



We see most of the variance is explained by component 1, then a similar lesser amount is explained by 2 and 3, followed by another drop to 4 and 5.

```
# Print loadings
print(loadings(pca_1),cutoff=0.00) #all loadings!
```

```
Loadings:
         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
mackerel  0.018  0.294  0.047  0.527  0.796
bluefin  -0.654  0.136  0.739 -0.089 -0.020
sardine   0.060  0.626 -0.015  0.520 -0.577
squid    -0.745  0.049 -0.664  0.029  0.018
limpet    0.116  0.707 -0.102 -0.665  0.183

               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings       1.0    1.0    1.0    1.0    1.0
Proportion Var    0.2    0.2    0.2    0.2    0.2
Cumulative Var    0.2    0.4    0.6    0.8    1.0
```

The PCA loadings are the correlations between the variables and each component. Here, we see bluefin and squid are strongly negatively correlated with component 1, while mackerel, sardine, and limpet are weakly positively correlated with component 1. We can continue this type of interpretation through the other components as well.

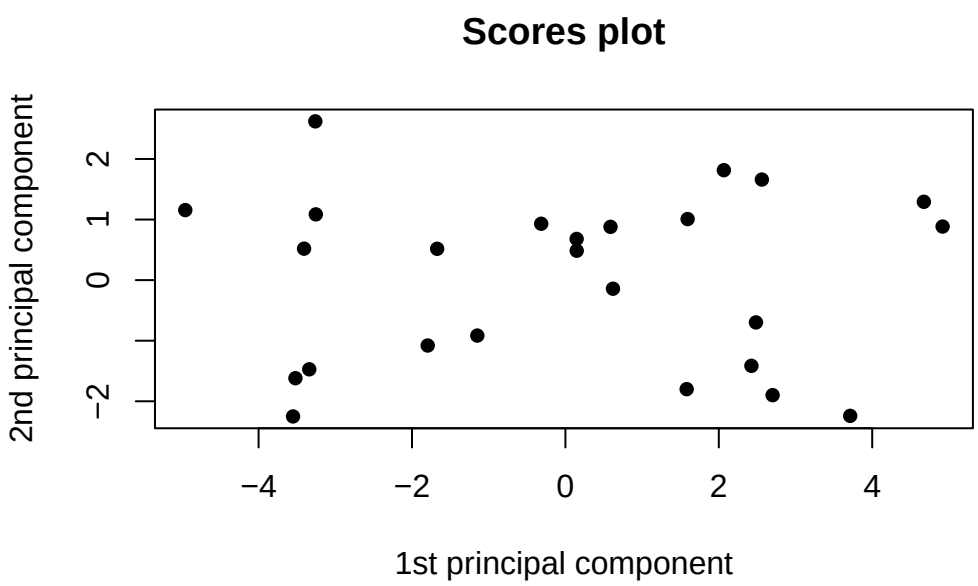
Our PCA object also contains the PCA scores for each individual data point:

```
# Print PCA scores
head(pca_1$scores)
```

```
        Comp.1     Comp.2     Comp.3     Comp.4      Comp.5
[1,] -3.551007 -2.2507718  0.6725187 -0.1223707 -0.06754585
[2,]  2.484116 -0.6980669  0.4886052 -0.3932653 -0.53609582
[3,]  2.425206 -1.4149241  0.9556963 -0.2274184 -0.07560834
[4,] -3.339469 -1.4723306 -0.2089590 -0.8854067 -0.03598265
[5,]  1.580988 -1.8002844 -4.6958830 -0.4313924  0.13590020
[6,] -3.519487 -1.6187995  0.3362833  0.1040827  0.32134755
```

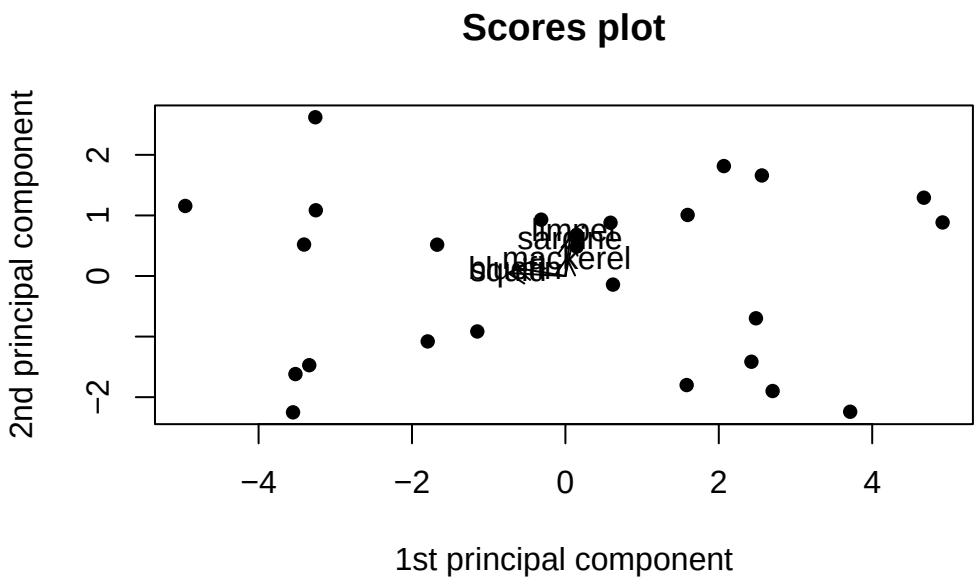
Scores are the value of each data point on each principal component. Lets try plotting them:

```
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # Axis
```



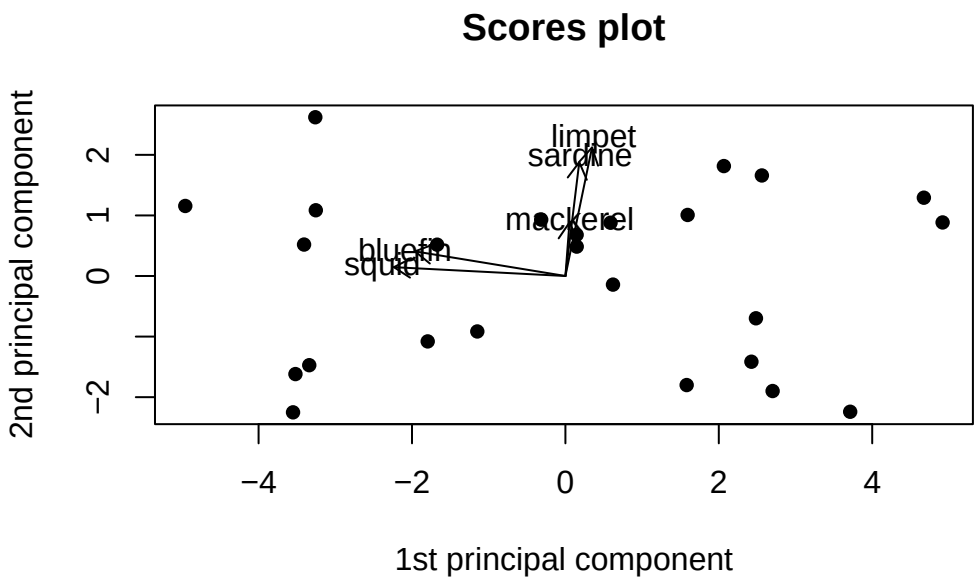
This generates a scatterplot showing us the value of each data point in principal components 1 (x) and 2 (y).

```
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # Axis

# Add loadings to plot
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1], # Draw to PC1 loading in X
       pca_1$loadings[,2], # Draw to PC2 loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1],pca_1$loadings[,2],names(data_log[,-1]),cex=1.0 ,col="black") # Add t
```



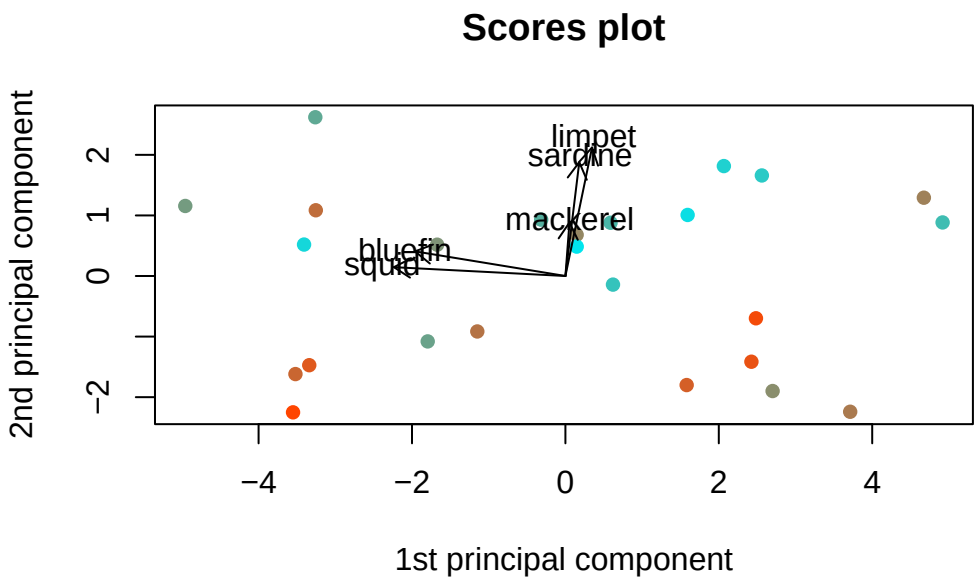
The arrows are a little small, so let's add a scaling factor:

```
# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # Axis

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_1$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1]*sft,pca_1$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="blac
```



What about the haul number? Does that have an effect? Let's try adding that on as well:

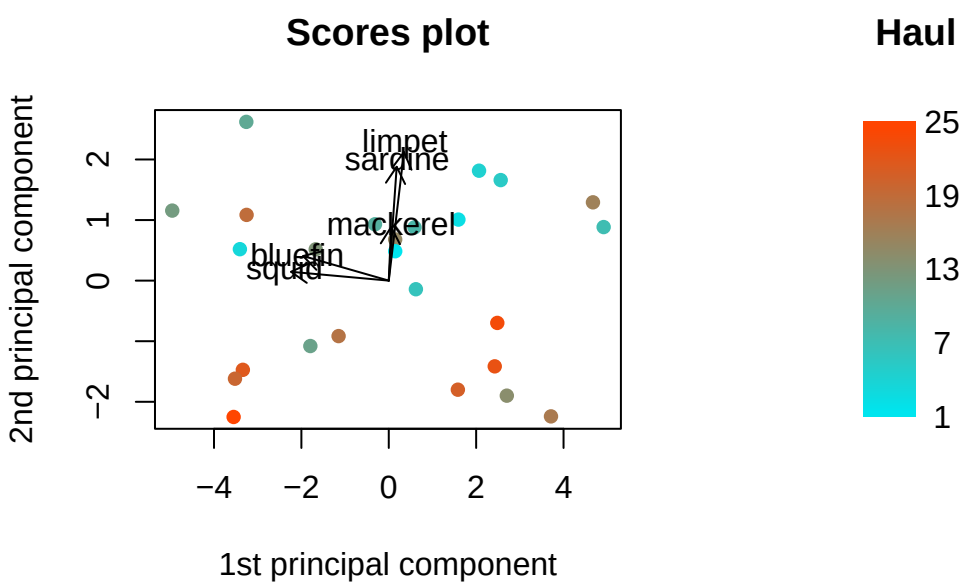
```
# Create a color palette
colfunc = colorRampPalette(c('orangered1', 'turquoise2'))

# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     col = colfunc(nrow(pca_1$scores)), # Color points by haul using our color palette
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # Axis

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_1$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1]*sft,pca_1$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="black
```



Since we used color for haul, we need to add a legend:

```
# Set plot layout
layout(matrix(1:2,ncol=2), # 1 row, 2 columns
       width = c(2,1), # Width
       height = c(1,1)) # Height

# Create a color palette
colfunc = colorRampPalette(c('orangered1', 'turquoise2'))

# Plot scores - components 1 and 2
plot(pca_1$scores[,1], # Scores on component 1
     pca_1$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     col = colfunc(nrow(pca_1$scores)), # Color points by haul using our color palette
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # Axis

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_1$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_1$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
```

```
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_1$loadings[,1]*sft,pca_1$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="blac

# Generate legend
legend_image <- as.raster(matrix(colfunc(nrow(pca_1$scores)), ncol=1))
plot(c(0,2),c(0,1),type = 'n', axes = F,xlab = '', ylab = '', main = 'Haul')
text(x=1.5, y =seq(0,1,l=5), labels = seq(1,25,l=5))
rasterImage(legend_image, 0, 0, 1,1)
```



Now we have a completed scores plot with loadings arrows. How would you interpret this plot?

10.3.2 Correlation Matrix

Now let's try the correlation matrix. The correlation matrix performs the same analysis, but on standardized data. The princomp() function does this for us if we set cor = T:

```
# Run PCA - Correlation
pca_2 = princomp(data_log[,-1], cor = T)
summary(pca_2)
```

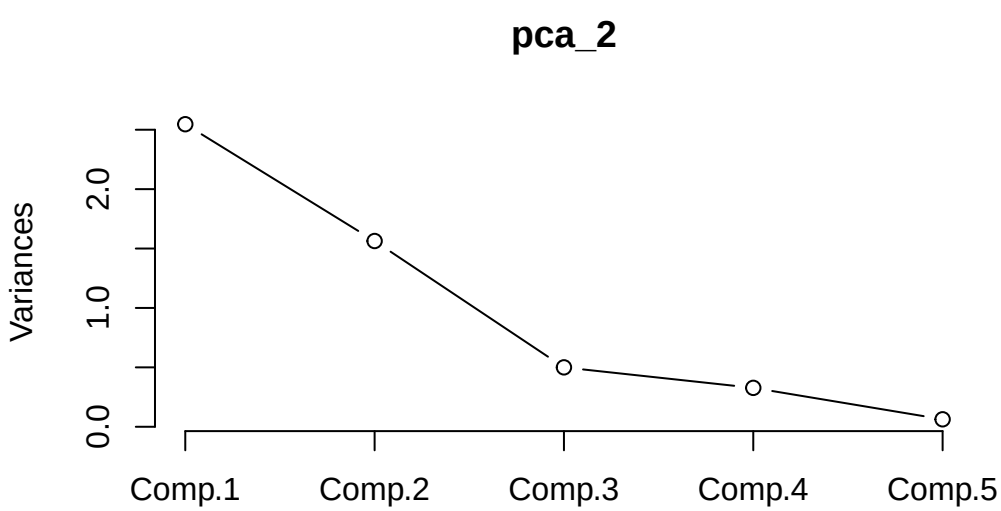
```
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
Standard deviation     1.595782 1.2503536 0.70708572 0.57220519 0.25041296
Proportion of Variance 0.509304 0.3126768 0.09999404 0.06548376 0.01254133
Cumulative Proportion  0.509304 0.8219809 0.92197491 0.98745867 1.00000000
```

```r
# In case you don't believe me, heres the covariance matrix if we pre-standardize the data
pca_test = princomp(scale(data_log[-1]))
summary(pca_test)
```

```
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
Standard deviation     1.563541 1.2250914 0.69279969 0.56064429 0.24535359
Proportion of Variance 0.509304 0.3126768 0.09999404 0.06548376 0.01254133
Cumulative Proportion  0.509304 0.8219809 0.92197491 0.98745867 1.00000000
```

Now we can go through the same pattern of analyses as we did for covariance:

```r
# Generate scree plot
plot(pca_2, type = 'l') # Scree is built into the plot for PCA
```

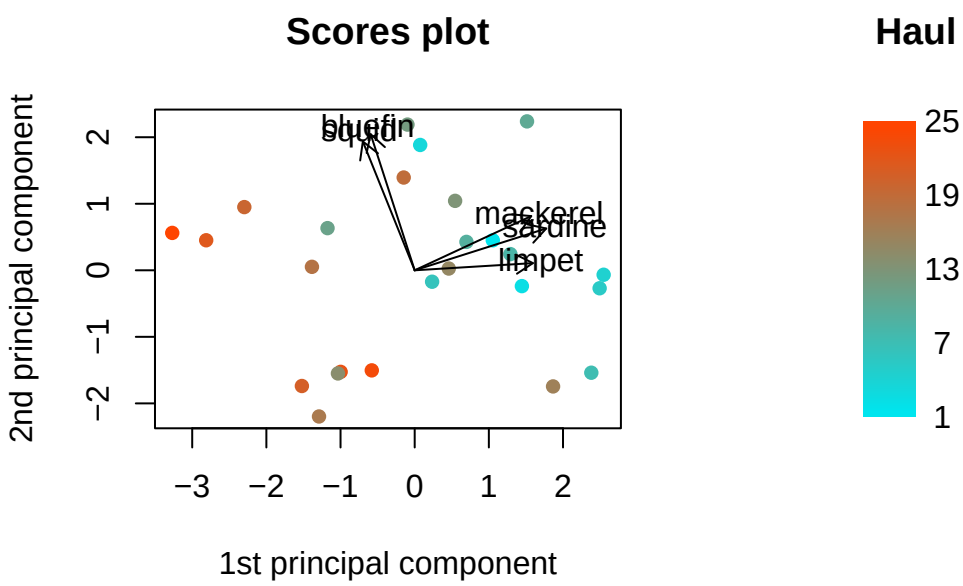


```r
# Print loadings
print(loadings(pca_2),cutoff=0.00) #all loadings!
```

```
Loadings:
         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
mackerel  0.524  0.272  0.527  0.297  0.535
bluefin  -0.198  0.682  0.264 -0.651 -0.050
sardine   0.591  0.209  0.025  0.109 -0.771
squid    -0.233  0.645 -0.472  0.550  0.059
limpet    0.532  0.036 -0.655 -0.416  0.338

               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
SS loadings       1.0    1.0    1.0    1.0    1.0
Proportion Var    0.2    0.2    0.2    0.2    0.2
Cumulative Var    0.2    0.4    0.6    0.8    1.0
```

```r
# Set plot layout
layout(matrix(1:2,ncol=2), # 1 row, 2 columns
       width = c(2,1), # Width
       height = c(1,1)) # Height

# Create a color palette
colfunc = colorRampPalette(c('orangered1', 'turquoise2'))

# Plot scores - components 1 and 2
plot(pca_2$scores[,1], # Scores on component 1
     pca_2$scores[,2], # Scores on component 3
     pch=16, # Point 16 (colored circle)
     col = colfunc(nrow(pca_2$scores)), # Color points by haul using our color palette
     xlab="1st principal component",ylab="2nd principal component",main="Scores plot") # Axis

# Add loadings to plot
sf = 3 # Scaling factor
sft = 3.2 # Scaling factor for text
arrows(0,0, # Draw arrows from zero
       pca_2$loadings[,1]*sf, # Draw to PC1 * scaling factor loading in X
       pca_2$loadings[,2]*sf, # Draw to PC2 * scaling factor loading in Y
       col="black", length = 0.1) # Arrow color and arrowhead length
text(pca_2$loadings[,1]*sft,pca_2$loadings[,2]*sft, names(data_log[,-1]), cex=1.0, col="black
```

```r
# Generate legend
legend_image <- as.raster(matrix(colfunc(nrow(pca_2$scores)), ncol=1))
plot(c(0,2),c(0,1),type = 'n', axes = F,xlab = '', ylab = '', main = 'Haul')
text(x=1.5, y =seq(0,1,l=5), labels = seq(1,25,l=5))
rasterImage(legend_image, 0, 0, 1,1)
```



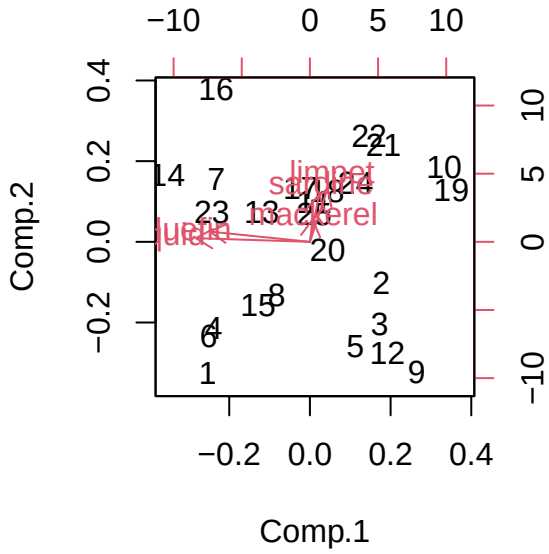
How would you interpret this plot? Does it differ from the covariance plot?

10.3.3 Alternative Methods

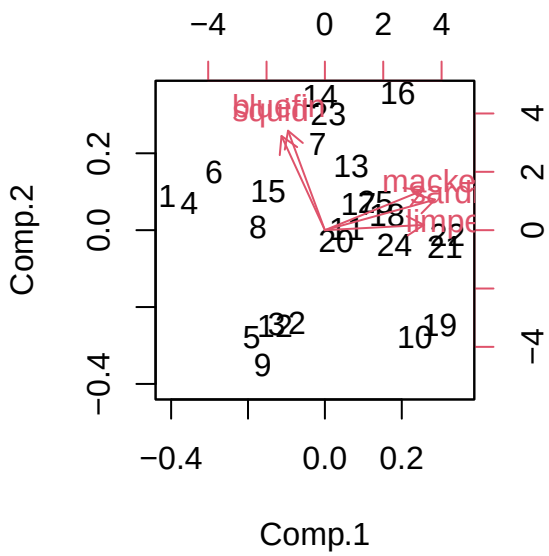
There are a few other ways you can generate, and/or plot your PCAs if you prefer.

10.3.3.1 Biplot

```r
# Exploring biplot
biplot(pca_1) # Covariance
```

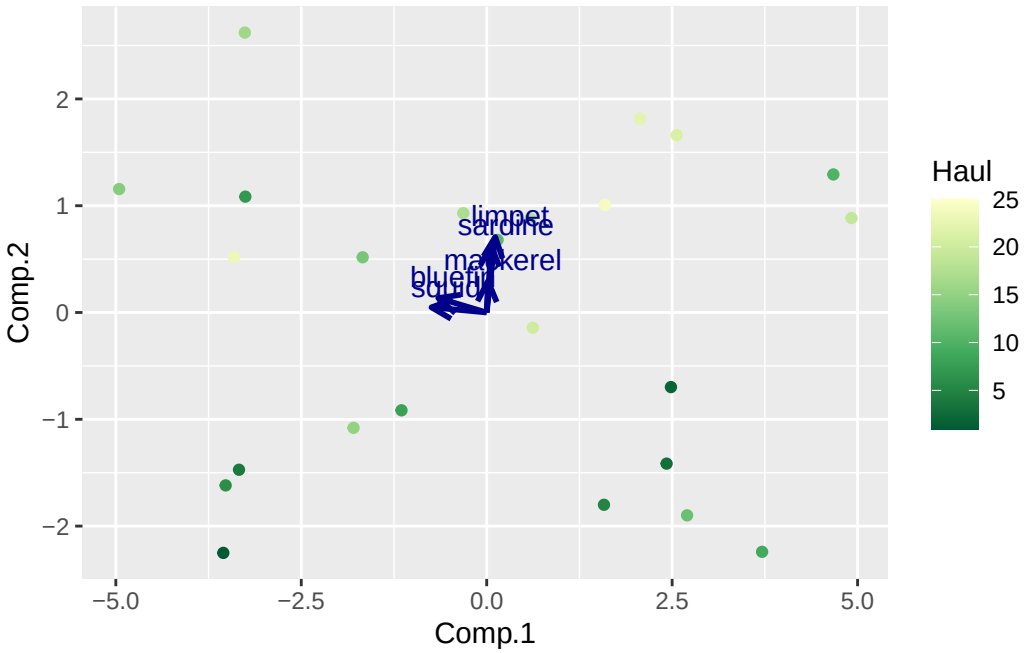



```
biplot(pca_2) # Correlation
```



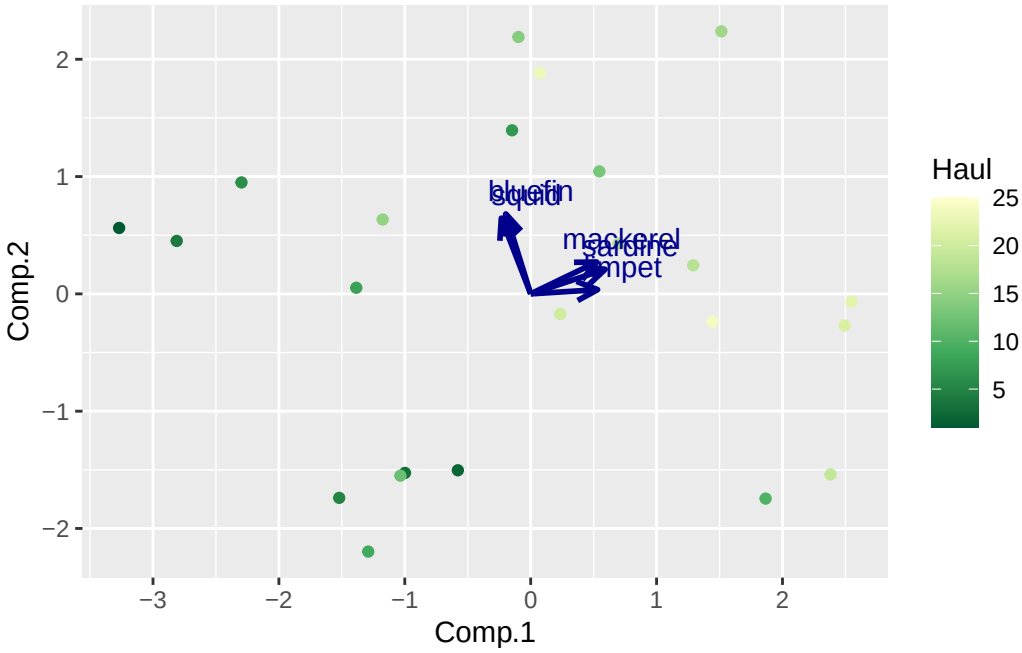
10.3.3.2 ggplot

```
library(ggplot2)

# ggplot version - Covariance

# turn PCA scores into data frame
pca_1_plot = data.frame(Haul = data_log$Haul, pca_1$scores)

# Turn PCA loadings into data frame (This gets a little complicated)
pca_1_loadings = as.data.frame(matrix(as.numeric(pca_1$loadings),
                                      dim(pca_1$loadings)[1], dim(pca_1$loadings)[2]))
colnames(pca_1_loadings) = colnames(pca_1_plot)[-1]

# Plot
ggplot(pca_1_plot, aes(x = Comp.1, y = Comp.2, color = Haul)) +

  # Scores
  geom_point() + scale_colour_distiller(palette = 15) +

  # Loadings
  geom_segment(data = pca_1_loadings, aes(x = 0, y = 0,xend = Comp.1 , yend = Comp.2),
    arrow = arrow(length = unit(0.3, "cm"), type = "open", angle = 25),
    linewidth = 1, color = "darkblue") +

  # Labels
  geom_text(data = pca_1_loadings, color = 'darkblue', nudge_x = 0.2, nudge_y = 0.2, # Labels
                aes(x = Comp.1, y = Comp.2, label = colnames(data_log)[-1]))
```




```
# ggplot version - Correlation

# turn PCA scores into data frame
pca_2_plot = data.frame(Haul = data_log$Haul, pca_2$scores)

# Turn PCA loadings into data frame
pca_2_loadings = as.data.frame(matrix(as.numeric(pca_2$loadings),
                                      dim(pca_2$loadings)[1], dim(pca_2$loadings)[2]))
colnames(pca_2_loadings) = colnames(pca_2_plot)[-1]

# Plot
ggplot(pca_2_plot, aes(x = Comp.1, y = Comp.2, color = Haul)) +

  # Scores
  geom_point() + scale_colour_distiller(palette = 15) +

  # Loadings
  geom_segment(data = pca_2_loadings, aes(x = 0, y = 0,xend = Comp.1 , yend = Comp.2),
               arrow = arrow(length = unit(0.3, "cm"), type = "open", angle = 25),
               linewidth = 1, color = "darkblue") +

  # Labels
  geom_text(data = pca_2_loadings, color = 'darkblue', nudge_x = 0.2, nudge_y = 0.2, # Labels
```

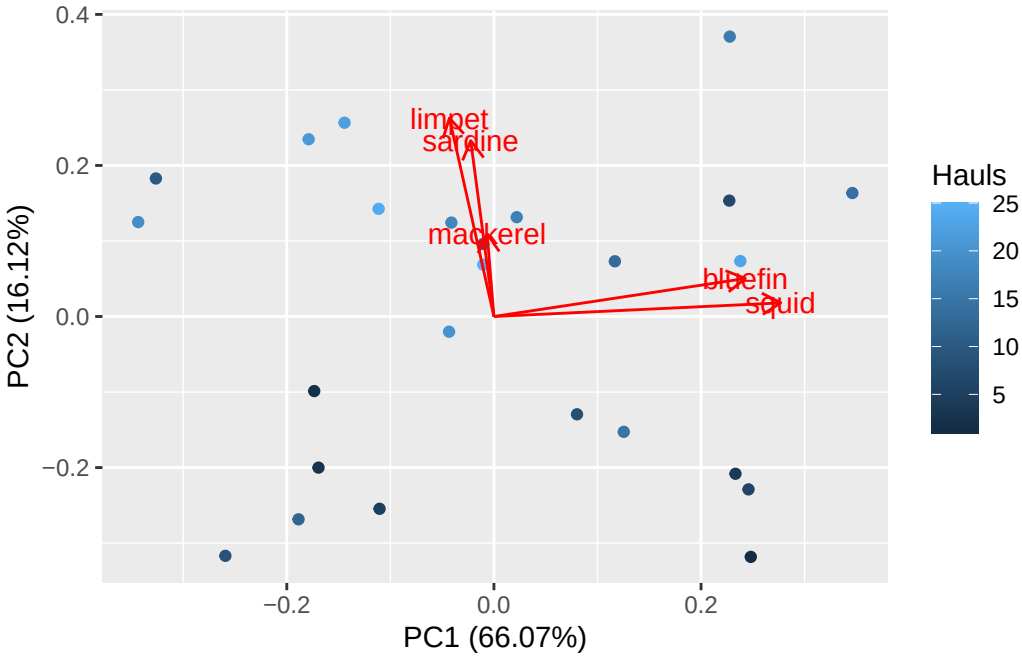
```
            aes(x = Comp.1, y = Comp.2, label = colnames(data_log)[-1]))
```



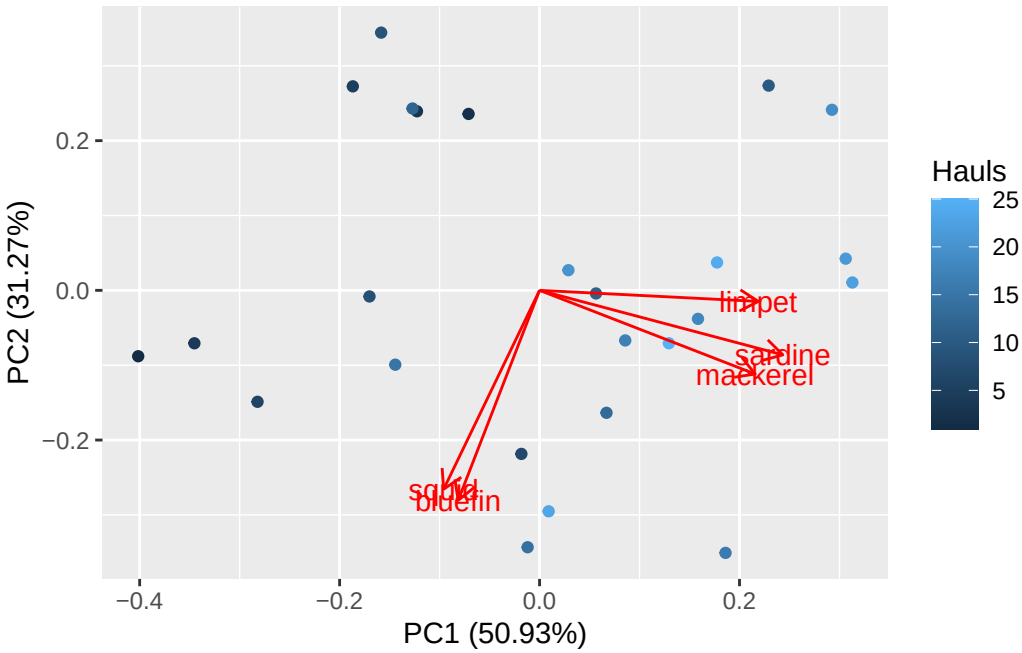
You can also run PCA using the prcomp() function instead of princomp(), setting scale = T if you want the correlation matrix. You can then use autoplot() with the ggfortify package to plot the results.

```
# ggplot v2
library(ggfortify)

# Run PCA - Covariance
pca_1a = prcomp(data_log[,-1])

# Run autoplot
autoplot(pca_1a, data = data_log, color = 'Hauls', loadings = T, loadings.label = T)
```

```
Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
i Please use tidy evaluation idioms with `aes()`.
i See also `vignette("ggplot2-in-packages")` for more information.
i The deprecated feature was likely used in the ggfortify package.
  Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
```



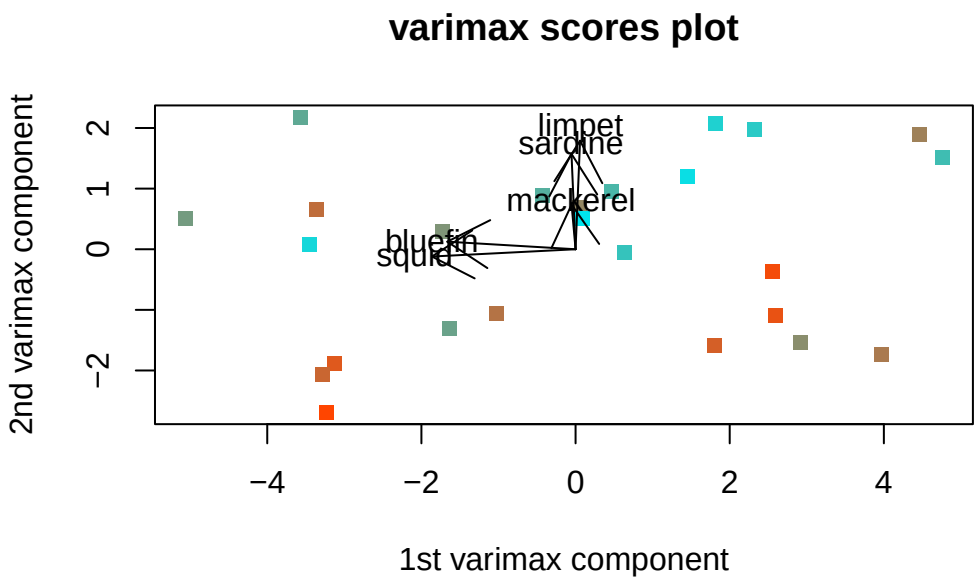

```
# Run PCA - Correlation
pca_2a = prcomp(data_log[,-1], scale = T)

# Run autoplot
autoplot(pca_2a, data = data_log, color = 'Hauls', loadings = T, loadings.label = T)
```



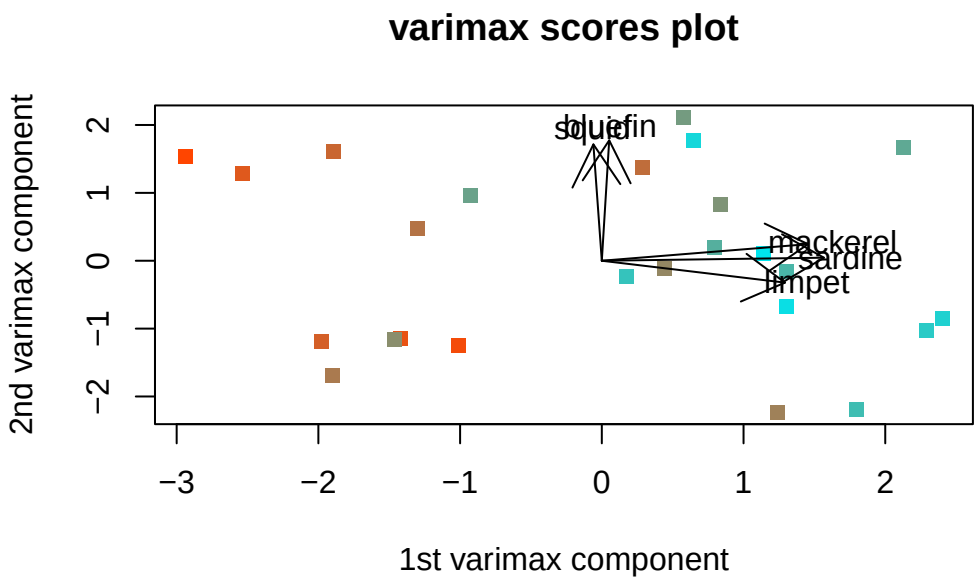
10.4 Varimax Rotation (Optional)

Varimax rotation attempts to improve the interpretability of PCA results by lining up loadings with the axes. This can be useful, particularly with large numbers of variables.

```
# Scaling factors
sf = 2.5
sft = 2.8

# Varimax rotation - Covariance
v1 = varimax(pca_1$loadings[,1:2])
v1_scores = pca_1$scores[,1:2]%*%v1$rotmat

# Plot scores - components 1 and 2
plot(v1_scores[,1],v1_scores[,2],pch=15, col = colfunc(nrow(v1_scores)),
     xlab="1st varimax component",ylab="2nd varimax component",main="varimax scores plot")

# Add loadings
arrows(0,0,v1$loadings[,1]*sf,v1$loadings[,2]*sf,col="black")
text(v1$loadings[,1]*sft,v1$loadings[,2]*sft,names(data_log[,-1]),asp=1,cex=1.0 ,col="black")
```




```
# Varimax rotation - Correlation
v2 = varimax(pca_2$loadings[,1:2])
v2_scores = pca_2$scores[,1:2]%*%v2$rotmat

# Plot scores - components 1 and 2
plot(v2_scores[,1],v2_scores[,2],pch=15, col = colfunc(nrow(v2_scores)),
     xlab="1st varimax component",ylab="2nd varimax component",main="varimax scores plot")

# Add loadings
arrows(0,0,v2$loadings[,1]*sf,v2$loadings[,2]*sf,col="black")
text(v2$loadings[,1]*sft,v2$loadings[,2]*sft,names(data_log[,-1]),asp=1,cex=1.0 ,col="black")
```



Note that it's pretty hard to tell the hauls apart using this color scale. Make sure your plots are always clear and readable.

10.5 Tips for your assignment:

Some things you may want to think about for your assignment:

1. Do your covariance and correlation plots differ? Do you think one is better suited to answering your research question? Why? Is your answer conceptual, or does it have to do with the results? Both?

2. How would you quantitatively examine the effect of haul on the PCA scores above? Is it associated with any of the principal components?

3. How would you interpret your statistical results biologically? You don't have to be right, but don't be vague, and don't contradict your results.

11 Assignment 1b: Linear Discriminant Analysis

Assignment 1b focuses on Linear Discriminant Analysis (LDA), also known as Canonical Variate Analysis. LDA is used to disclose relationships between groups, create models to differentiate between groups based on data, and discern the contribution of different variables to a model's ability do discriminate between groups.

For this tutorial, we'll be using `snake.csv`.

11.1 Looking at the data

```
# Load in data
snake = read.csv('snake.csv')

# Look at data
head(snake)
```

```
  Species   M1   M2   M3   M4   M5   M6
1     A   41.6  6.7  8.2 12.2 24.7 27.0
2     A   40.2  8.5  9.2 15.5 27.1 30.3
3     A   40.4 12.6 14.2 19.6 46.9 26.8
4     A   26.4  9.0  8.6 14.0 37.6 32.2
5     A   34.4  7.0 12.1 11.1 31.0 35.8
6     A   38.8  8.2 10.2 12.4 42.2 33.6
```

```
dim(snake)
```

```
[1] 35  7
```

Our data is a 35 row, 7 column data frame. The first column identifies the species of snake (A or B). The other columns are morphological measurements of each individual snake. We want to know if we can use the morphological measurements of the snakes to determine their species. Let's keep examining the data:

```r
# Make a boxplot
library(tidyverse)
```

```
-- 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   1.5.2
v ggplot2   4.0.0     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.1.0
-- 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 to becom
```

```r
# Convert the data to long format so we can use ggplot
snake_long = pivot_longer(snake, # Enter data
                          colnames(snake)[-1], # Pivot all columns except species
                          names_to = 'Measurement', values_to = 'Value') # Feed labels to new

# Lets take a look at the new data frame
head(snake_long)
```

```
# A tibble: 6 x 3
  Species  Measurement Value
  <chr>    <chr>       <dbl>
1 "   A  " M1           41.6
2 "   A  " M2            6.7
3 "   A  " M3            8.2
4 "   A  " M4           12.2
5 "   A  " M5           24.7
6 "   A  " M6           27
```

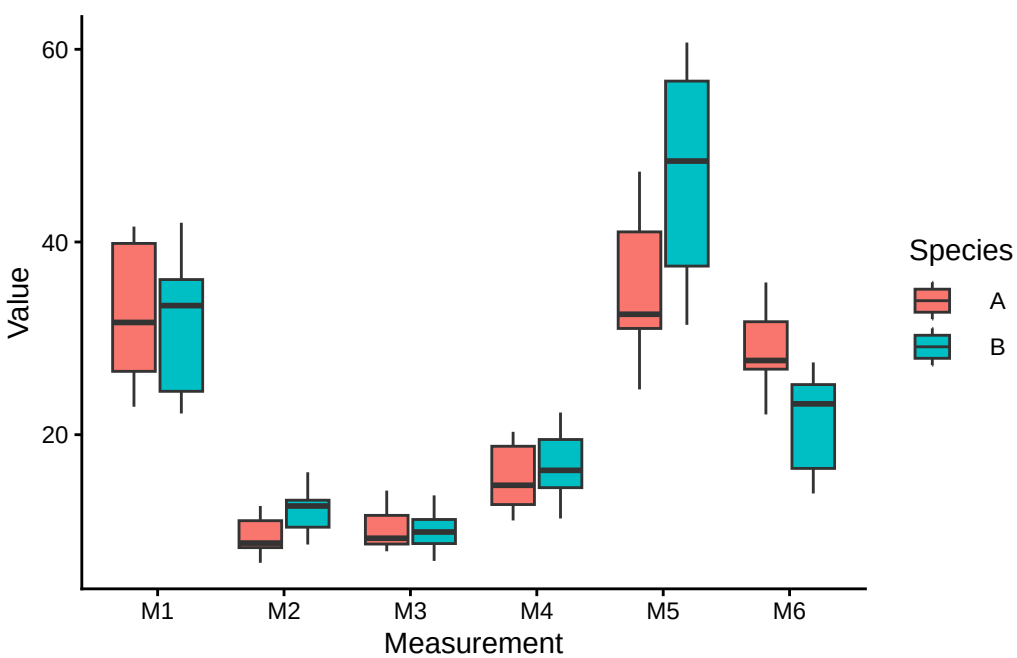
```r
# We've converted from wide format to long format,
# now all the data values are contained in a single column
# which is described by a metadata column

# You can also do this with melt from reshape2
library(reshape2)
```

```
Attaching package: 'reshape2'

The following object is masked from 'package:tidyr':

    smiths
```

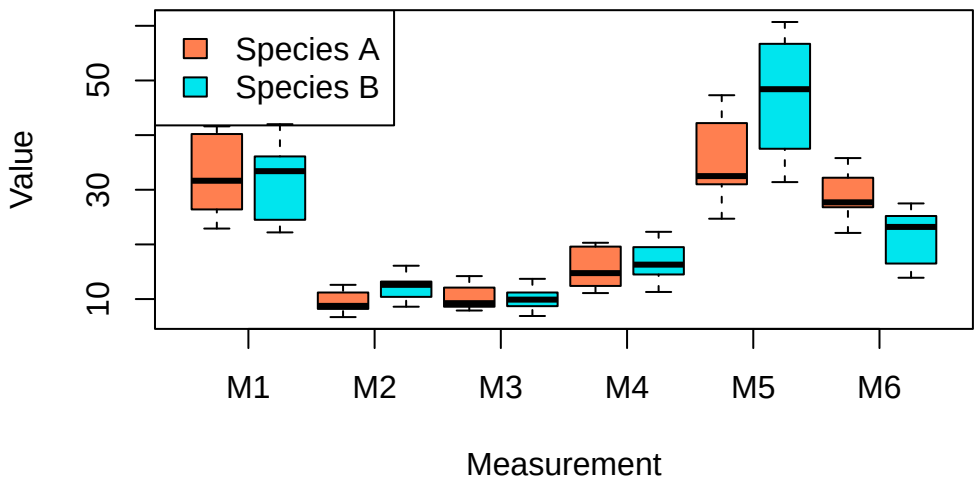
```r
head(melt(snake))
```

```
Using Species as id variables

  Species variable value
1     A         M1  41.6
2     A         M1  40.2
3     A         M1  40.4
4     A         M1  26.4
5     A         M1  34.4
6     A         M1  38.8
```

```r
# Let's make a boxplot
ggplot(snake_long, aes(x = Measurement, y = Value, fill = Species)) +
  geom_boxplot() + theme_classic()
```



```
# We can do this in R base plot too
boxplot(Value ~ Species*Measurement, # Plot value by species and measurement
        data = snake_long, col = c('coral', 'turquoise2'), # Color by species
        xaxt = 'n', xlab = 'Measurement') # Remove and label x axis
legend('topleft', legend = c('Species A', 'Species B'), fill = c('coral', 'turquoise2')) # Ad
axis(1, at = seq(1.5,11.5,2), labels = colnames(snake)[-1]) # Add x axis back in with appropr
```



Some of our measurements are very similar across species, and others are quite different. Do they differ statistically as a whole?

11.2 MANOVA

The purpose of LDA is to try to discriminate our snakes into species based on their measurements. However, that only makes sense to do if our two species of snake actually differ across the measurements. Our first step then is to discern whether our snake species differ as a multivariate whole. We'll do this using a MANOVA.

```
# Run MANOVA
sm = manova(cbind(M1,M2,M3,M4,M5,M6) ~ Species, data = snake)
summary(sm, test = 'Hotelling')
```

```
          Df Hotelling-Lawley approx F num Df den Df   Pr(>F)
Species    1           1.2263   5.7229      6     28 0.000552 ***
Residuals 33
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(sm, test = 'Wilks')
```

```
          Df   Wilks approx F num Df den Df   Pr(>F)
Species    1 0.44917   5.7229      6     28 0.000552 ***
Residuals 33
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

By both the Hotelling's and Wilks' tests, our MANOVA is significant, indicating the snake species vary as a multivariate whole.

What about our assumptions though? Our MANOVA assumptions are independence, normality, linearity, and homogeneity of covariances. You've been told to assume the latter, so let's skip that one. Independence states that measurements of each snake are independent from all others. For example, it would be violated if our data were related to each other - for example, if some of our snakes were closely related, or if the graduate students measuring them were using different methods. We don't have information about how this data was collected, so we cannot assess independence. We'll skip that one as well.

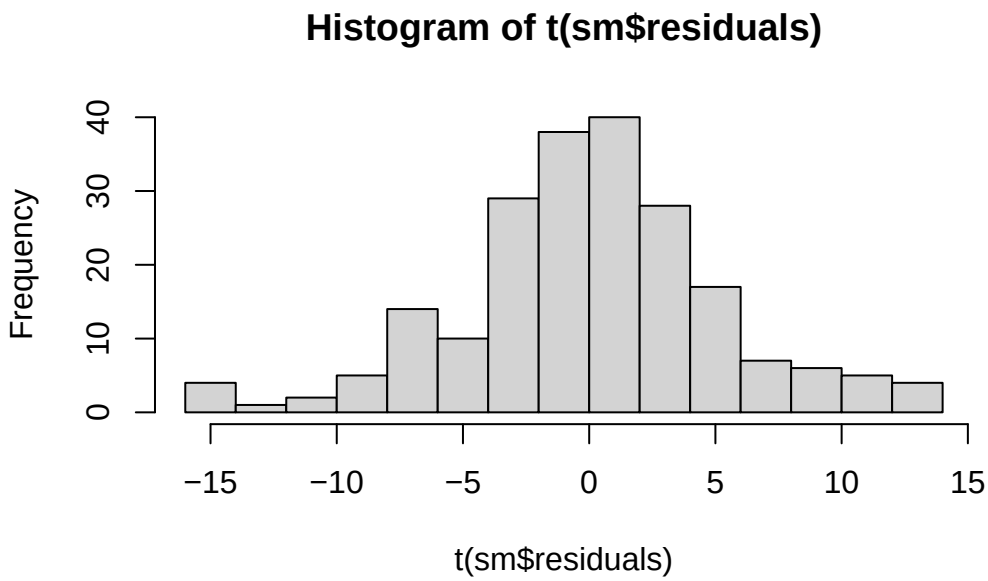
Let's start by testing for normality:

```r
# Testing normality
library(mvnormtest)
mshapiro.test(t(sm$residuals))
```

```
    Shapiro-Wilk normality test

data:  Z
W = 0.91571, p-value = 0.01075
```

Uh oh, the residuals are significantly non-normal. Let's take a look at them visually:

```r
# Residual histogram
hist(t(sm$residuals), breaks = 20)
```



Visually, our residuals actually look quite close to normal. There may be some slight skew, or outliers that are forcing our residuals to statistical non-normality. We might be able to fix this by removing multivariate outliers, or by transforming some of our data (feel free to play around with these ideas!), but based on the shape of our residuals, it is unlikely that our model is fatally biased, and we may end up doing more harm than good. Based on this, we can conclude that our two species have significantly different morphometries given the measurements provided.

11.3 Linear Discriminant Analysis

Now that we've confirmed our species differ as a multivariate whole, we can try to use LDA to build a model to predict which species each snake belongs to based on its measurements.

```
# LDA
library(MASS)
```

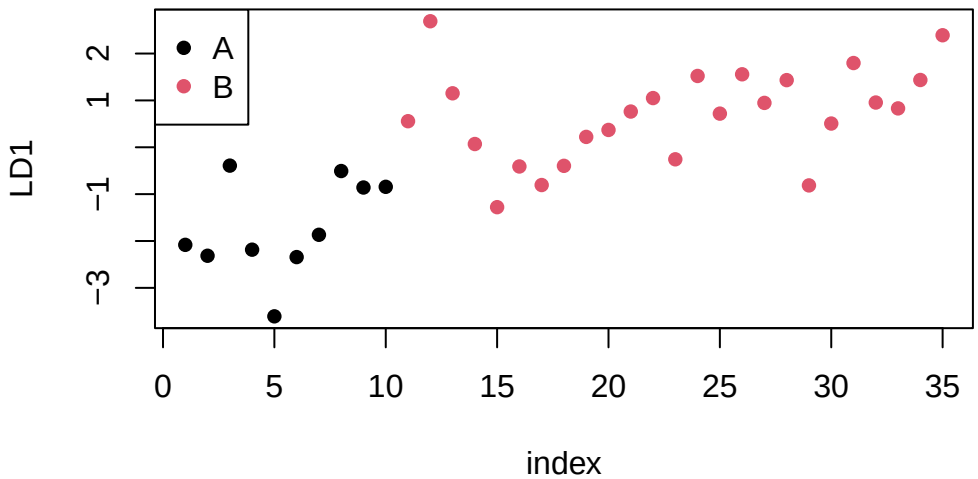
```
Attaching package: 'MASS'

The following object is masked from 'package:dplyr':

    select
```

```r
ldaf1 <- lda(Species ~ M1+M2+M3+M4+M5+M6, snake)
ldaf1
```

```
Call:
lda(Species ~ M1 + M2 + M3 + M4 + M5 + M6, data = snake)

Prior probabilities of groups:
      A         B 
0.2857143 0.7142857 

Group means:
           M1     M2    M3    M4     M5     M6
   A   32.700  9.410 10.16 15.54 35.290 28.950
   B   31.496 12.128 10.18 16.78 47.356 21.752

Coefficients of linear discriminants:
           LD1
M1  0.01428023
M2  0.29104494
M3 -0.07327616
M4 -0.05544769
M5  0.03629586
M6 -0.17208517
```

Running our LDA object tells us the prior probabilities used for each species (the proportion of each species in the data), the group means for each measure on each species, and the linear discriminant (LD1) for each measure. We can then plot the LD1 value for each individual:

```r
# Plot discriminant function analysis

# Create a data frame to plot
ldaf_plot = cbind(snake, # Data
                  predict(ldaf1)$x, # LD1 value for each individual given its measurements
                  index = seq(1,nrow(snake), 1)) # Row/Individual number

# Plot
plot(LD1 ~ index, data = ldaf_plot, col = as.factor(snake$Species), pch = 16)
legend('topleft', legend = c('A', 'B'), col = c(1, 2), pch = 16) # Add legend
```



Here we can see higher LD1 values are associated with species B, while lower LD1 values are associated with species A. This is just based on model fit however; how do we know we aren't overfitting? One way to avoid overfitting is by jackknifing (AKA leave-one-out cross validation in this context). This method runs the model once without each point in the dataset, then calculates the posterior probability that the left out point belongs to each species. Let's try it out:

```
# LDA 2, CV = T
ldaf2 = lda(Species ~ M1+M2+M3+M4+M5+M6, snake, CV = T)

# Gather posteriors
as.data.frame(cbind(ldaf2$posterior, # Pull posteriors from ldaf2
                    ResultantSpp=as.character(ldaf2$class))) # Pull predicted species (i.e. s
```

```
                    A                    B   ResultantSpp
1     0.897801237948675    0.102198762051325           A
2     0.957033498274347   0.0429665017256533           A
3   0.00486396795570835    0.995136032044292           B
4     0.939579607872302   0.0604203921276982           A
5     0.999020574119129 0.000979425880871105           A
6     0.958283942083953   0.0417160579160474           A
7     0.859914694048174    0.140085305951826           A
8     0.079027668947901    0.920972331052099           B
```

```
9     0.250711809994117    0.749288190005883          B
10    0.277233534989758    0.722766465010243          B
11   0.0654339037846645    0.934566096215336          B
12 8.13045175684641e-05    0.999918695482432          B
13  0.00857331675606217    0.991426683243938          B
14    0.119793120831736    0.880206879168264          B
15    0.868897347918874    0.131102652081126          A
16    0.291404395123414    0.708595604876586          B
17    0.580893601645515    0.419106398354485          A
18    0.407526292222816    0.592473707777184          B
19   0.0971393472407664    0.902860652759234          B
20   0.0629455676122023    0.937054432387798          B
21   0.0262176442553782    0.973782355744622          B
22   0.0110464412594654    0.988953558740534          B
23    0.379676706769168    0.620323293230832          B
24  0.00300786068222621    0.996992139317774          B
25   0.0331152011340242    0.966884798865976          B
26  0.00270158005931189    0.997298419940688          B
27   0.0161164609849136    0.983883539015086          B
28  0.00346528534198867    0.996534714658011          B
29    0.761253716426844    0.238746283573156          A
30   0.0597294571669353    0.940270542833065          B
31  0.00139900114299065    0.998600998857009          B
32    0.014630451548708    0.985369548451292          B
33   0.0215114427320868    0.978488557267913          B
34  0.00359029891803416    0.996409701081966          B
35 8.92739861715449e-05    0.999910726013828          B
```

How does this differ from the predictions from our first model?

```r
# Pull ldaf1 model predictions
ldaf_pred = predict(ldaf1)$class

# Gather Predictions
ldaf_diff = data.frame(ldaf1 = as.character(ldaf_pred), ldaf2 = as.character(ldaf2$class))

# Add match column
ldaf_diff$match = (ldaf_diff$ldaf1 == ldaf_diff$ldaf2)

# Which ones are different?
ldaf_diff[which(ldaf_diff$match == F),]
```

```
    ldaf1  ldaf2 match
```

```
17    B      A   FALSE
29    B      A   FALSE
```

Individuals 17 and 29 both differed in species prediction between the model fit and the jackknife posterior probability. Now let's check the accuracy of our model fit:

```r
# Calculate error
ldaf_wrong = length(which(ldaf_pred != snake$Species)) # Number of incorrect predictions
ldaf_err = ldaf_wrong/nrow(snake) # Divide by number of individuals for error

# Print error
ldaf_wrong
```

```
[1] 5
```

```r
ldaf_err
```

```
[1] 0.1428571
```

Our model classified 5 out of 35 (~14.3%) of the snakes as the incorrect species, meaning 30/35 were correct (~85.7%). Not bad, but can we do better?

11.4 Model Selection

Our previous model used all 6 measurements, but do we really need all of them, or are some of them unhelpful (or even detrimental)? To test this, we can run model selection using the `stepclass()` function:

```r
# stepclass package
library(klaR)

# Model selection (forward)
ms_f = stepclass(Species ~ M1+M2+M3+M4+M5+M6,data=snake,
          method="lda", fold=35, direction="forward")
```

```
 `stepwise classification', using 35-fold cross-validated correctness rate of method lda'.

35 observations of 6 variables in 2 classes; direction: forward
```

```
stop criterion: improvement less than 5%.

correctness rate: 0.85714;  in: "M6";  variables (1): M6

 hr.elapsed min.elapsed sec.elapsed
       0.00        0.00        0.44
```

```r
# Print model selection result
ms_f
```

```
method      : lda
final model : Species ~ M6
<environment: 0x00000228524a6110>

correctness rate = 0.8571
```

After model selection, we end up with a model using only M6 to predict species, with a correctness rate of 85.7%. This model has the same correctness as the full model, using only one measurement. In other words, this model is more **efficient** - it gets to the same accuracy using less information.

This model was generated using forward model selection, meaning the selection process works exclusively by adding variables to the model. We can also do the opposite:

```r
# stepclass package
library(klaR)

# Model selection (forward)
ms_b = stepclass(Species ~ M1+M2+M3+M4+M5+M6,data=snake,
          method="lda", fold=35, direction="backward")
```

```
 `stepwise classification', using 35-fold cross-validated correctness rate of method lda'.

35 observations of 6 variables in 2 classes; direction: backward

stop criterion: improvement less than 5%.

correctness rate: 0.8;  starting variables (6): M1, M2, M3, M4, M5, M6
correctness rate: 0.85714;  out: "M5";  variables (5): M1, M2, M3, M4, M6

 hr.elapsed min.elapsed sec.elapsed
       0.00        0.00        0.58
```

```r
# Print model selection result
ms_b
```

```
method      : lda
final model : Species ~ M1 + M2 + M3 + M4 + M6
<environment: 0x0000022849853cf8>

correctness rate = 0.8571
```

Backwards model selection works by removing variables from the full model. This means backwards selection usually returns a model with equal or more variables than forwards selection.

Lastly, we can run both:

```r
# stepclass package
library(klaR)

# Model selection (forward)
ms_d = stepclass(Species ~ M1+M2+M3+M4+M5+M6,data=snake,
          method="lda", fold=35, direction="both")
```

```
 `stepwise classification', using 35-fold cross-validated correctness rate of method lda'.

35 observations of 6 variables in 2 classes; direction: both

stop criterion: improvement less than 5%.

correctness rate: 0.85714;  in: "M6";  variables (1): M6

 hr.elapsed min.elapsed sec.elapsed
       0.00        0.00        0.42
```

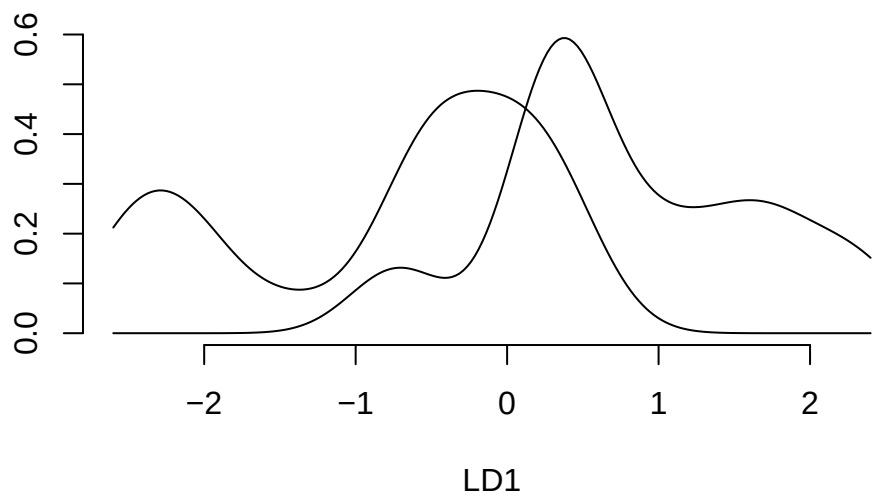
```r
# Print model selection result
ms_d
```

```
method      : lda
final model : Species ~ M6
<environment: 0x00000228478047a0>

correctness rate = 0.8571
```

11.5 Plotting Probabilities

Lets finish off by making some plots to visualize our LDA model results.

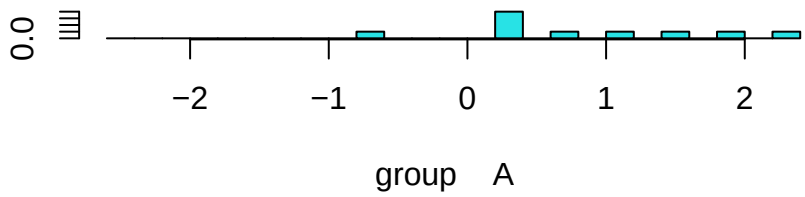
```
# Pick a model to plot
ldaf3 = lda(Species ~ M6, data = snake)

# Plot density curve
plot(ldaf3, dimen = 1, type = 'dens')
```

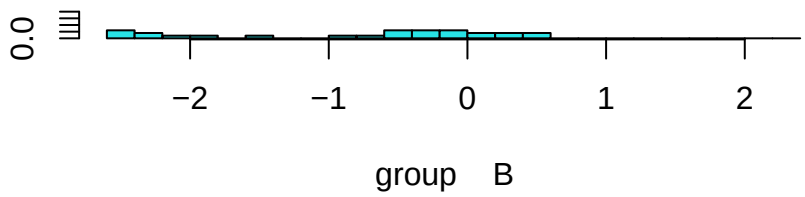


This plots the posterior probabilites of an individual belonging to either species given its LD1 value. Remember from earlier that species A is associated with lower LD1 values.

We can also make this plot as a histogram:

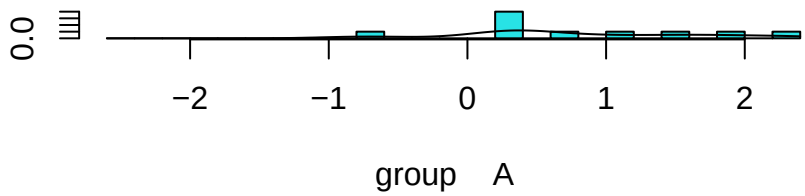
```
# Plot density curve
par(mar = c (4,4,4,4))
plot(ldaf3, dimen = 1, type = 'hist')
```

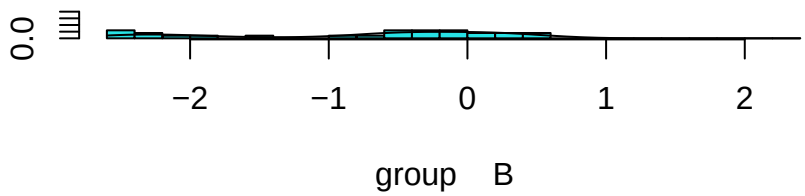




Or combine both plots:

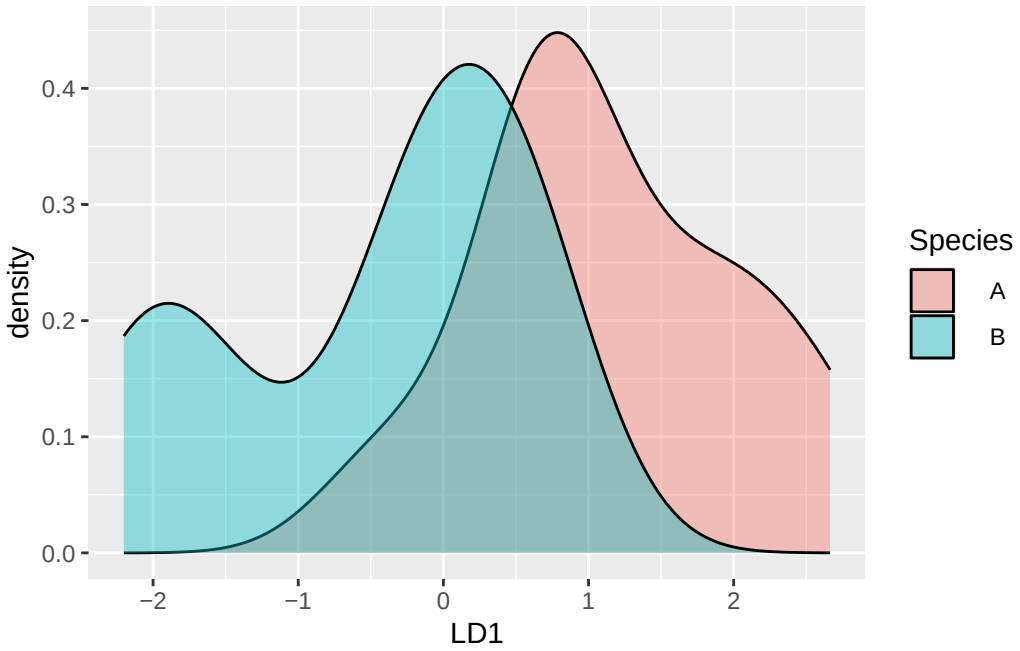
```
# Plot density curve
par(mar = c (4,4,4,4))
plot(ldaf3, dimen = 1, type = 'both')
```





As always, we can also do this with ggplot too:

```
# Predict species
ldaf3_pred = predict(ldaf3)

# Plot
pred_species = as.data.frame(ldaf3_pred$x) # Gather LD1 values
pred_species$Species = snake$Species # Gather true species from data

# Plot
ggplot(pred_species, aes(x = LD1, fill = Species))+
  geom_density(alpha = 0.4)# alpha tells you how transparent the plots will be
```



11.6 Tips for your assignment

Some things you may want to think about for your assignment:

1. How would you pick which model you think is best? What factors would you consider? Are there any factors you would consider other than those discussed in this tutorial?

2. How would you interpret your statistical results biologically (can be in terms of the snakes, how you would study them, or both)? You don't have to be right, but don't be vague, and don't contradict your results.

12 Assignment 1c: Cluster Analysis and Multidimensional Scaling

This assignment is centered on cluster analysis and multidimensional scaling (MDS), which are both methods of measuring associations within a group (e.g. associations between individuals within a population).

For this tutorial, we'll be using `monkey.csv`.

12.1 Looking at the data

You know the drill by now:

```
# Load in data
data = read.csv('monkey.csv', row.names = 1) # First column is row names
data # Print data
```

```
      ind1 ind2 ind3 ind4 ind5 ind6 ind7 ind8 ind9 ind10 ind11 ind12 ind13
ind1    21    2    2   10    2    2    8    0    0     8    14    12     4
ind2     2   21   16    2   16    8    2    2    4     4     4     0     2
ind3     2   16   21    0   10   16    2    0    2     4     4     0     2
ind4    10    2    0   21    2    2   16    2    2     8    12     8     4
ind5     2   16   10    2   21   10    2    4    0     2     4     4     2
ind6     2    8   16    2   10   21    4    2    0     0     0     4     4
ind7     8    2    2   16    2    4   21    4    2    16     8     8     4
ind8     0    2    0    2    4    2    4   21    0     2     0     0     0
ind9     0    4    2    2    0    0    2    0   21     0     4     0     0
ind10    8    4    4    8    2    0   16    2    0    21    14    14     2
ind11   14    4    4   12    4    0    8    0    4    14    21    12     4
ind12   12    0    0    8    4    4    8    0    0    14    12    21     2
ind13    4    2    2    4    2    4    4    0    0     2     4     2    21
```

Our data is a matrix containing the number of social interactions observed between individuals in a group of monkeys at the zoo. The matrix is symmetrical - the top/right half is identical to the bottom/left half.

12.2 Calculating Dissimilarity

For this assignment we'll be using 3 R functions: `hclust`, `metaMDS` (from the `vegan` package), `isoMDS` (from the `MASS`package), and `cmdscale()`. Let's see what type of input data those functions need:

```
# Check help functions
library(vegan)
library(MASS)
?hclust()
?metaMDS()
?isoMDS()
?cmdscale()
```

You'll notice all of these functions require a **dissimilarity matrix** produced by `dist`. Let's start by running `dist()`.

```
# Convert data to a dist object
dist = as.dist(data)
dist # Print dist
```

```
      ind1 ind2 ind3 ind4 ind5 ind6 ind7 ind8 ind9 ind10 ind11 ind12
ind2     2
ind3     2   16
ind4    10    2    0
ind5     2   16   10    2
ind6     2    8   16    2   10
ind7     8    2    2   16    2    4
ind8     0    2    0    2    4    2    4
ind9     0    4    2    2    0    0    2    0
ind10    8    4    4    8    2    0   16    2    0
ind11   14    4    4   12    4    0    8    0    4    14
ind12   12    0    0    8    4    4    8    0    0    14    12
ind13    4    2    2    4    2    4    4    0    0     2     4     2
```

Now our data is in a `dist` object. All of the redundant entries in the data have been removed.

Right now, our data reflects **similarity (i.e. high numbers reflect greater association between individuals)**. We need to convert it to **dissimilarity**. Dissimilarity is simply the opposite of similarity. We can convert similarity to dissimilarity by subtracting each data value from the maximum of the data.

```r
# Convert to dissimilarity
dist = max(dist) - dist
dist # Print dist
```

```
      ind1 ind2 ind3 ind4 ind5 ind6 ind7 ind8 ind9 ind10 ind11 ind12
ind2    14
ind3    14    0
ind4     6   14   16
ind5    14    0    6   14
ind6    14    8    0   14    6
ind7     8   14   14    0   14   12
ind8    16   14   16   14   12   14   12
ind9    16   12   14   14   16   16   14   16
ind10    8   12   12    8   14   16    0   14   16
ind11    2   12   12    4   12   16    8   16   12     2
ind12    4   16   16    8   12   12    8   16   16     2     4
ind13   12   14   14   12   14   12   12   16   16    14    12    14
```

Now we're ready to run our analyses!

12.3 Hierarchical Cluster Analysis

Remember from lecture there are 4 types of hierarchical cluster analysis:

1. Single linkage
2. Average linkage
3. Complete linkage
4. Ward linkage

Let's run through them one by one:

12.3.1 Single linkage

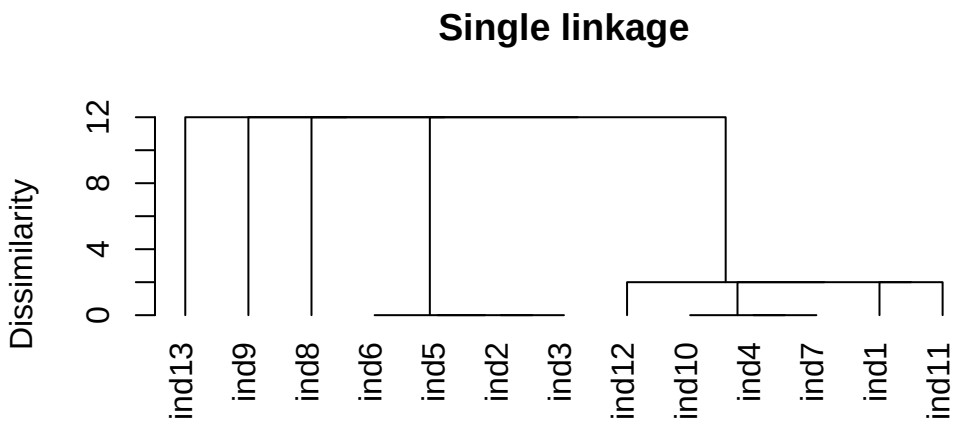
We can run all 4 types of cluster analysis using the `hclust()` R function:

```r
# run single linkage cluster analysis
clust_1 = hclust(dist, method = 'single')
clust_1 # print object
```

```
Call:
hclust(d = dist, method = "single")

Cluster method   : single
Number of objects: 13
```

Printing the `hclust` object doesn't really tell us much. For more detail, we're going to have to plot it:

```r
# Plot single linkage tree
plot(clust_1, hang = -1, main = 'Single linkage',
     ylab = 'Dissimilarity', # Label y axis
     xlab = '', sub = '') # Remove x-axis label
```



This outputs a tree showing the associations between our individual monkeys. dissimilarity is on the y-axis. The greater the distance between individuals on the y-axis, the greater their dissimilarity. Our tree has grouped the monkeys according to how frequently they interact with each other. For example. individuals 2, 3, 5, and 6 interact often, as evidenced by their low dissimilarity.

But how well does this tree fit the data? To answer that question, we need to calculate the cophenetic correlation coefficient (CCC):

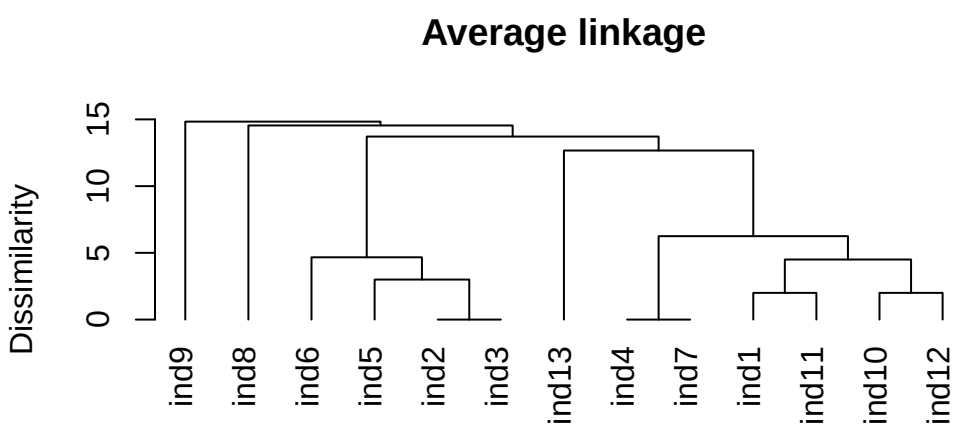
```r
# Calculate CCC
coph_1 = cophenetic(clust_1) # Get cophenetic
ccc_1 = cor(coph_1, dist) # Calculate correlation of the cophenetic with the data
ccc_1 # Print CCC
```

```
[1] 0.9036043
```

That's a pretty high correlation coefficient, indicating our dendrogram represented the structure in the original data very well. Let's try some other methods:

12.3.2 Average Linkage

```r
# run cluster analysis
clust_2 = hclust(dist, method = 'average')

# Plot
plot(clust_2, hang = -1, main = 'Average linkage', ylab = 'Dissimilarity', xlab = '', sub =
```

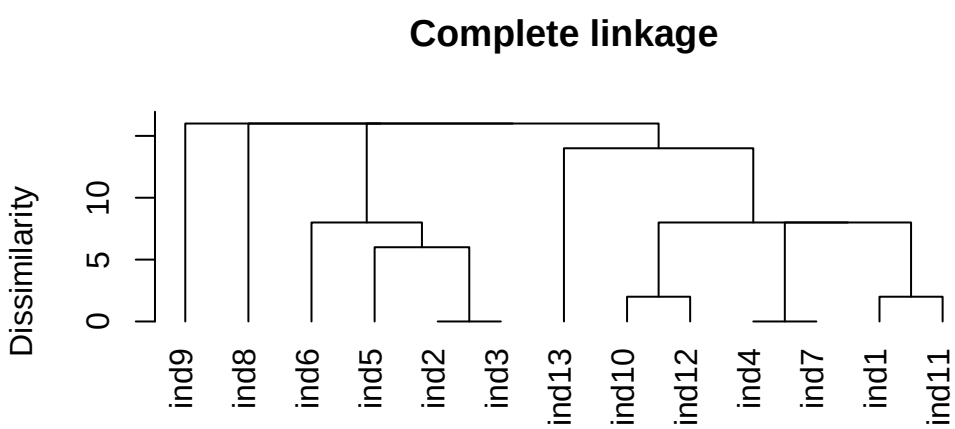


```r
# Calculate CCC
coph_2 = cophenetic(clust_2)
ccc_2 = cor(coph_2, dist)
ccc_2
```

```
[1] 0.9288949
```

12.3.3 Complete Linkage

```r
# run cluster analysis
clust_3 = hclust(dist, method = 'complete')

# Plot
plot(clust_3, hang = -1, main = 'Complete linkage', ylab = 'Dissimilarity', xlab = '', sub =
```

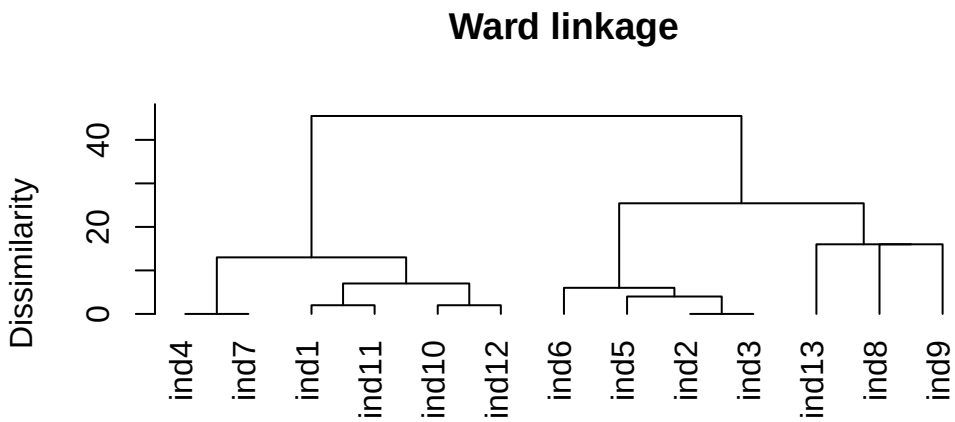



```r
# Calculate CCC
coph_3 = cophenetic(clust_3)
ccc_3 = cor(coph_3, dist)
ccc_3
```

```
[1] 0.9141956
```

12.3.4 Ward Linkage

```
# run cluster analysis
clust_4 = hclust(dist, method = 'ward.D')

# Plot
plot(clust_4, hang = -1, main = 'Ward linkage', ylab = 'Dissimilarity', xlab = '', sub = '')
```




```
# Calculate CCC
coph_4 = cophenetic(clust_4)
ccc_4 = cor(coph_4, dist)
ccc_4
```

```
[1] 0.7633159
```

Each method gives a slightly different tree and CCC value. Where are they similar? Where do they differ? Which one(s) would you trust? Why?

12.4 Multidimensional Scaling

Another method we can use to test for associations between our monkeys is multidimensional scaling (MDS). There are two types of MDS: non-metric, and metric MDS. Let's start with non-metric MDS.

12.4.1 Non-Metric MDS

```r
# Run non-metric MDS - metaMDS
mds1 = metaMDS(dist, wascores = F)
```

```
Run 0 stress 0.07592385
Run 1 stress 0.08569496
Run 2 stress 0.07239301
... New best solution
... Procrustes: rmse 0.2256252  max resid 0.66433
Run 3 stress 0.08055013
Run 4 stress 0.1396043
Run 5 stress 0.07239301
... Procrustes: rmse 1.824264e-05  max resid 3.008e-05
... Similar to previous best
Run 6 stress 0.07358653
Run 7 stress 0.1183846
Run 8 stress 0.1782434
Run 9 stress 0.07882366
Run 10 stress 0.07358653
Run 11 stress 0.07239301
... Procrustes: rmse 0.0001239058  max resid 0.0002604218
... Similar to previous best
Run 12 stress 0.08569059
Run 13 stress 0.07366297
Run 14 stress 0.07308015
Run 15 stress 0.07875788
Run 16 stress 0.08571329
Run 17 stress 0.1193093
Run 18 stress 0.1802626
Run 19 stress 0.07366297
Run 20 stress 0.1426486
*** Best solution repeated 2 times
```

```r
# Print mds results
mds1
```

```
Call:
metaMDS(comm = dist, wascores = F)

global Multidimensional Scaling using monoMDS

Data:     dist
Distance: user supplied

Dimensions: 2
Stress:     0.07239301
Stress type 1, weak ties
Best solution was repeated 2 times in 20 tries
The best solution was from try 2 (random start)
Scaling: centring, PC rotation
Species: scores missing
```

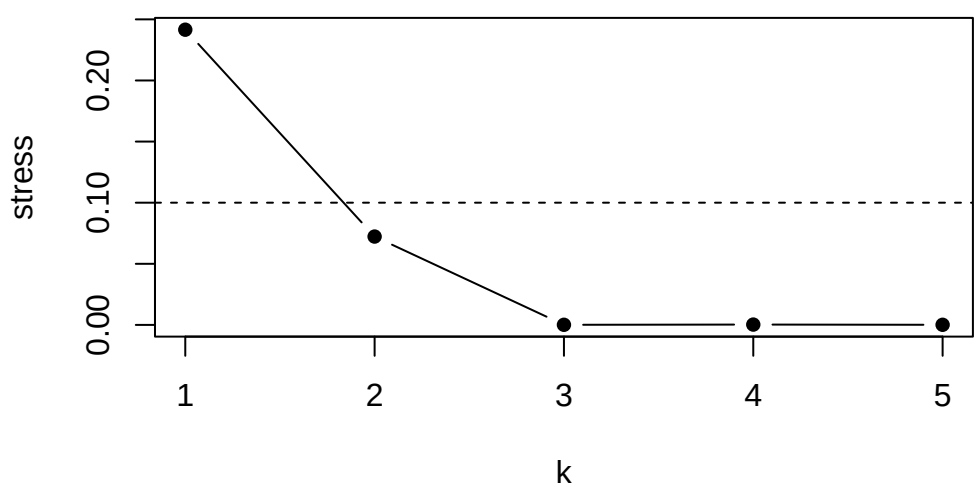
By default, metaMDS has two dimensions. This MDS has a stress value of 0.072. Remember from lecture that stress < 0.10 is a "good representation", so this MDS result is pretty good. If we want, we can test different numbers of dimensions (k) and create a scree plot to find the best one:

```r
# Create a container object
scree = data.frame(k = 1:5, stress = NA)

# Loop through k 1 to 5
for(k in 1:5){

  # Run MDS
  mds = metaMDS(dist, wascores = F, k = k) # Set k to our loop index

  # Pull out stress
  scree[k,'stress'] = mds$stress # Fill kth row of the column 'stress' in scree

} # End loop
```

```r
# Print results
scree
```

```
  k       stress
1 1 2.415380e-01
2 2 7.227846e-02
3 3 8.541956e-05
4 4 2.760417e-04
5 5 9.545373e-05
```

```r
# Make scree plot
plot(stress ~ k, data = scree, # Plot stress against k
     type = 'b', # Lines and points
     pch = 16) # Point 16 (filled circle)
abline(h = 0.1, lty = 'dashed') # Plot a dashed line at 0.1
```

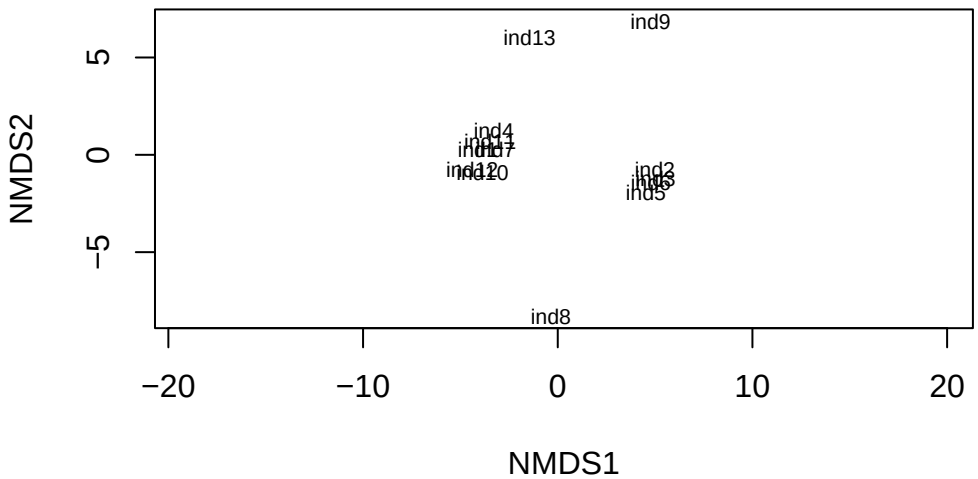


We have an elbow at k=3, but we also get warnings that our dataset may be too small using k=3. The stress at k=2 is low enough that we can stick to using that.

Let's plot our results:

```r
# Plot result
plot(mds1, type = 't')
```

```
species scores not available
```



Here we've plotted the values of our two MDS dimensions against each other for each individual. Similar to the cluster analysis, we see certain individuals are grouped together. Is it the same groups of individuals? What does that tell you about your results?

Let's try a different non-metric MDS function:

```
# Run non-metric MDS - isoMDS
mds2 = isoMDS(dist)
```

```
Error in isoMDS(dist): zero or negative distance between objects 2 and 3
```

Uh oh. This function doesn't like zeroes in the data. Let's fix that by translating our data to proportions, and adding a small increment.

```
# Translate to proportions
dist2 = dist/max(dist)

# Add an increment
dist2 = dist2 + 0.0001

# Print new dist
dist2
```

```
        ind1   ind2   ind3   ind4   ind5   ind6   ind7   ind8   ind9  ind10
ind2  0.8751
ind3  0.8751 0.0001
ind4  0.3751 0.8751 1.0001
ind5  0.8751 0.0001 0.3751 0.8751
ind6  0.8751 0.5001 0.0001 0.8751 0.3751
ind7  0.5001 0.8751 0.8751 0.0001 0.8751 0.7501
ind8  1.0001 0.8751 1.0001 0.8751 0.7501 0.8751 0.7501
ind9  1.0001 0.7501 0.8751 0.8751 1.0001 1.0001 0.8751 1.0001
ind10 0.5001 0.7501 0.7501 0.5001 0.8751 1.0001 0.0001 0.8751 1.0001
ind11 0.1251 0.7501 0.7501 0.2501 0.7501 1.0001 0.5001 1.0001 0.7501 0.1251
ind12 0.2501 1.0001 1.0001 0.5001 0.7501 0.7501 0.5001 1.0001 1.0001 0.1251
ind13 0.7501 0.8751 0.8751 0.7501 0.8751 0.7501 0.7501 1.0001 1.0001 0.8751
       ind11  ind12
ind2
ind3
ind4
ind5
ind6
ind7
ind8
ind9
ind10
ind11
ind12 0.2501
ind13 0.7501 0.8751
```

Let's make sure this doesn't mess with our results:

```r
# Run non-metric MDS - metaMDS
mds1 = metaMDS(dist2, wascores = F)
```

```
Run 0 stress 0.07575137
Run 1 stress 0.07366297
... New best solution
... Procrustes: rmse 0.1934883  max resid 0.5648741
Run 2 stress 0.0757299
Run 3 stress 0.07239302
... New best solution
... Procrustes: rmse 0.1707871  max resid 0.5155241
Run 4 stress 0.07366297
Run 5 stress 0.120356
```

```
Run 6 stress 0.07239299
... New best solution
... Procrustes: rmse 4.37661e-05  max resid 9.044302e-05
... Similar to previous best
Run 7 stress 0.08568228
Run 8 stress 0.08568213
Run 9 stress 0.08055013
Run 10 stress 0.08569059
Run 11 stress 0.07239301
... Procrustes: rmse 3.180058e-05  max resid 5.990962e-05
... Similar to previous best
Run 12 stress 0.07239303
... Procrustes: rmse 7.675115e-05  max resid 0.0001696219
... Similar to previous best
Run 13 stress 0.07239299
... New best solution
... Procrustes: rmse 1.353324e-05  max resid 3.011726e-05
... Similar to previous best
Run 14 stress 0.07366297
Run 15 stress 0.08568213
Run 16 stress 0.072393
... Procrustes: rmse 4.039663e-05  max resid 8.493912e-05
... Similar to previous best
Run 17 stress 0.1802625
Run 18 stress 0.07572991
Run 19 stress 0.1765419
Run 20 stress 0.08055013
*** Best solution repeated 2 times
```

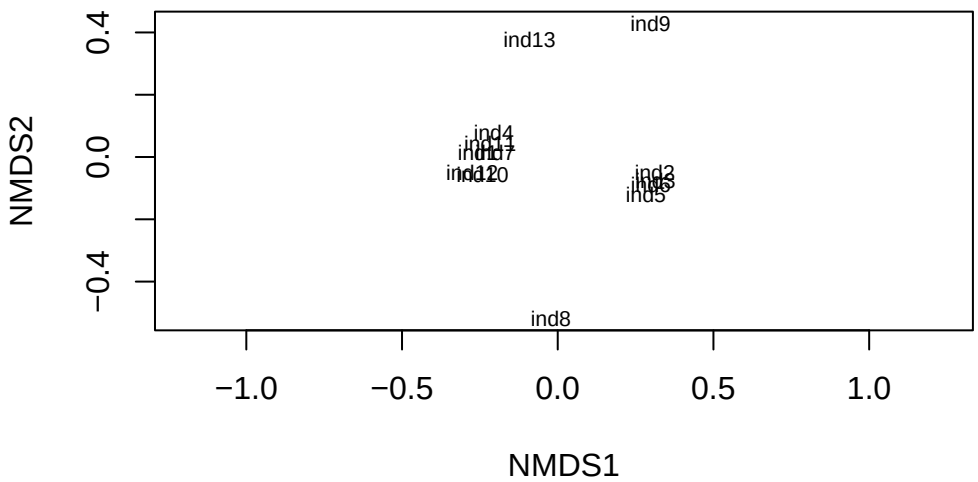
```r
# Print mds results
mds1
```

```
Call:
metaMDS(comm = dist2, wascores = F)

global Multidimensional Scaling using monoMDS

Data:     dist2
Distance: user supplied

Dimensions: 2
```

```
Stress:     0.07239299
Stress type 1, weak ties
Best solution was repeated 2 times in 20 tries
The best solution was from try 13 (random start)
Scaling: centring, PC rotation
Species: scores missing
```

```r
# Plot result
plot(mds1, type = 't')
```

```
species scores not available
```



The values have shifted around a bit but the structure and interpretation of the plot is the same. Let's continue on:

```r
# Run non-metric MDS - isoMDS
mds2 = isoMDS(dist2)
```

```
initial  value 24.760322
iter   5 value 14.153502
iter  10 value 12.254154
iter  15 value 11.639473
```

```
iter  20 value 11.360460
final  value 11.341572
converged
```

```r
# Print output
mds2
```

```
$points
            [,1]         [,2]
ind1   0.5060798  0.103253718
ind2  -0.6157097 -0.285522725
ind3  -0.6133549 -0.310223762
ind4   0.5008471  0.144443835
ind5  -0.6264450 -0.279477605
ind6  -0.6228134 -0.325577873
ind7   0.4940123  0.147103149
ind8  -0.6783876  0.680257989
ind9  -0.2575043  1.075686777
ind10  0.5482152  0.033286470
ind11  0.5478840  0.082682890
ind12  0.5895070  0.001005794
ind13  0.2276697 -1.066918657

$stress
[1] 11.34157
```

The modelling algorithms seems to be a little different, and we end up with a different stress result - in this case, one that is above the 10% threshold (note that stress is in % in this function, unlike metaMDS where it is in proportion). Let's try another scree plot:

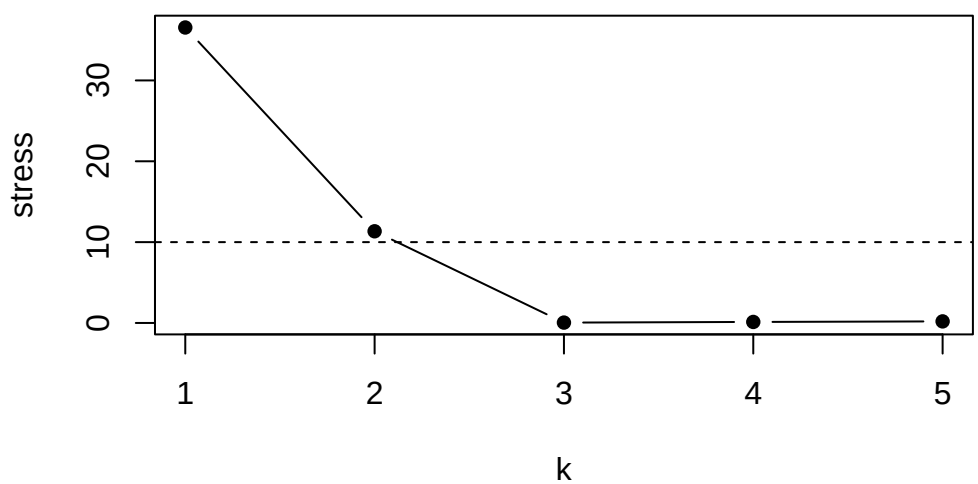
```r
# Create a container object
scree = data.frame(k = 1:5, stress = NA)

# Loop through k 1 to 5
for(k in 1:5){

  # Run MDS
  mds = isoMDS(dist2, k = k) # Set k to our loop index

  # Pull out stress
  scree[k,'stress'] = mds$stress # Fill kth row of the column 'stress' in scree
```

```r
} # End loop
```

```r
# Print results
scree
```

```
  k      stress
1 1 36.54857293
2 2 11.34157190
3 3  0.04614441
4 4  0.12630298
5 5  0.19439990
```

```r
# Make scree plot
plot(stress ~ k, data = scree, # Plot stress against k
     type = 'b', # Lines and points
     pch = 16) # Point 16 (filled circle)
abline(h = 10, lty = 'dashed') # Plot a dashed line at 0.1
```



In this case, it seems we're better off using 3 dimensions:

```r
# Run non-metric MDS - isoMDS
mds2 = isoMDS(dist2, k = 3)
```

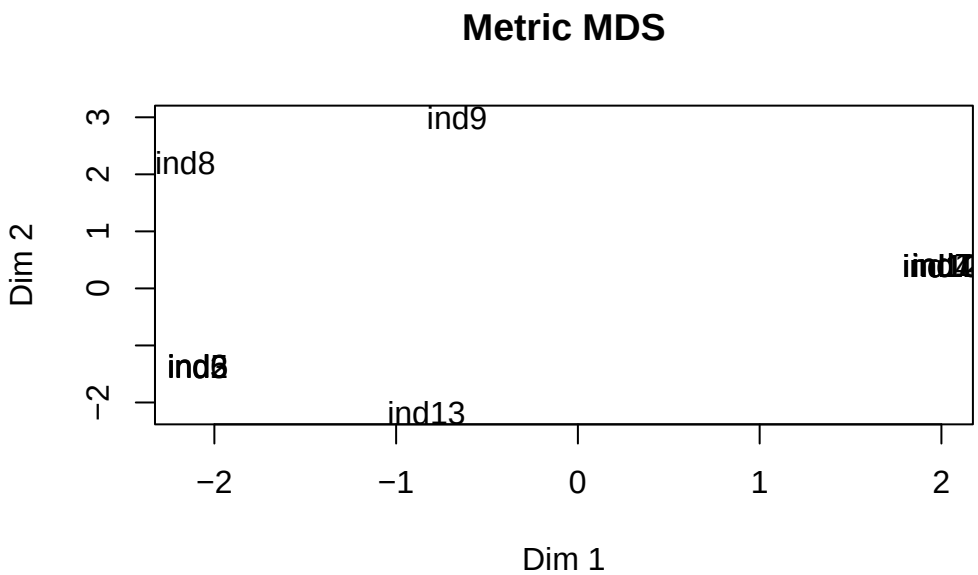
```
initial  value 18.960422
iter   5 value 11.725940
iter  10 value 6.417141
iter  15 value 4.149185
iter  20 value 1.466748
iter  25 value 0.764657
iter  30 value 0.449114
iter  35 value 0.302911
iter  40 value 0.156116
iter  45 value 0.087536
iter  50 value 0.046144
final  value 0.046144
stopped after 50 iterations
```

```r
# Print output
mds2
```

```
$points
            [,1]       [,2]       [,3]
ind1   2.0028765  0.4079841 -0.4202376
ind2  -2.0901193 -1.3657278  1.2720375
ind3  -2.0910832 -1.3666943  1.2729356
ind4   2.0066696  0.4042391 -0.4161698
ind5  -2.0896284 -1.3631915  1.2769256
ind6  -2.0921681 -1.3637290  1.2724192
ind7   2.0032691  0.4111298 -0.4185160
ind8  -2.1600174  2.2033842 -1.8938700
ind9  -0.6641520  2.9972846  2.5707229
ind10  2.0009095  0.4046552 -0.4185896
ind11  2.0041166  0.4044773 -0.4197885
ind12  2.0019583  0.4023552 -0.4223419
ind13 -0.8326313 -2.1761669 -3.2555274

$stress
[1] 0.04614441
```

Let's plot our results:

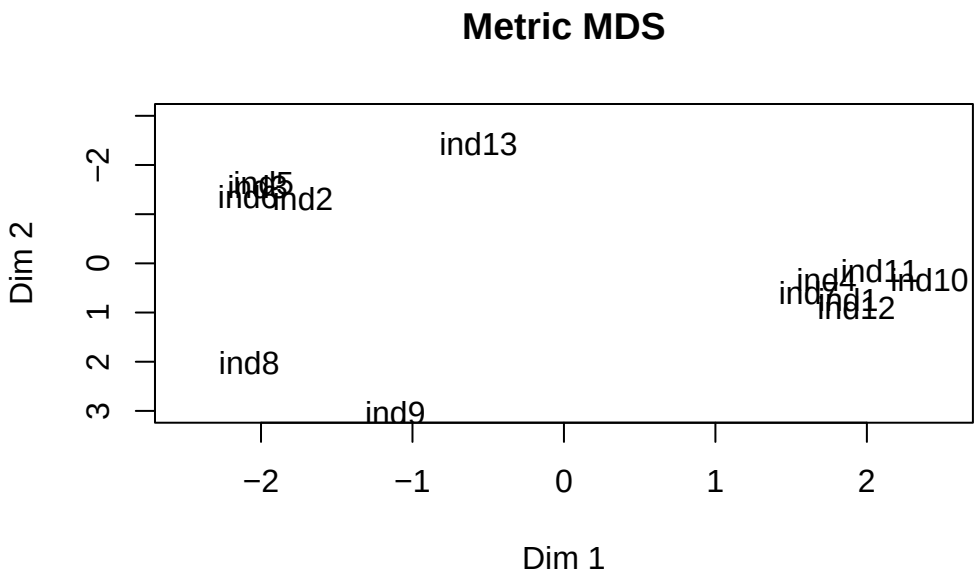
```r
# Plot isoMDS
plot(mds2$points[,1], mds2$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS') # Labelling

# Plot individual names
text(mds2$points[,1], mds2$points[,2], rownames(data))
```



All of our grouped individuals are plotted on top of each other. Let's try adding some random jiggle so we can see them

```r
# Plot isoMDS
plot(mds2$points[,1], mds2$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS', # Labelling
     xlim = c(-2.5, 2.5), ylim = c(3, -3)) # Set axis limits

# Set random seed for consistency
set.seed(1212)

# Plot individual names
text(mds2$points[,1] + rnorm(13, 0, 0.2), # Add random values pulled from a
```

```
     mds2$points[,2] + rnorm(13, 0, 0.2), # normal distribution with mean 0, sd 0.2
     rownames(data)) # Add names
```

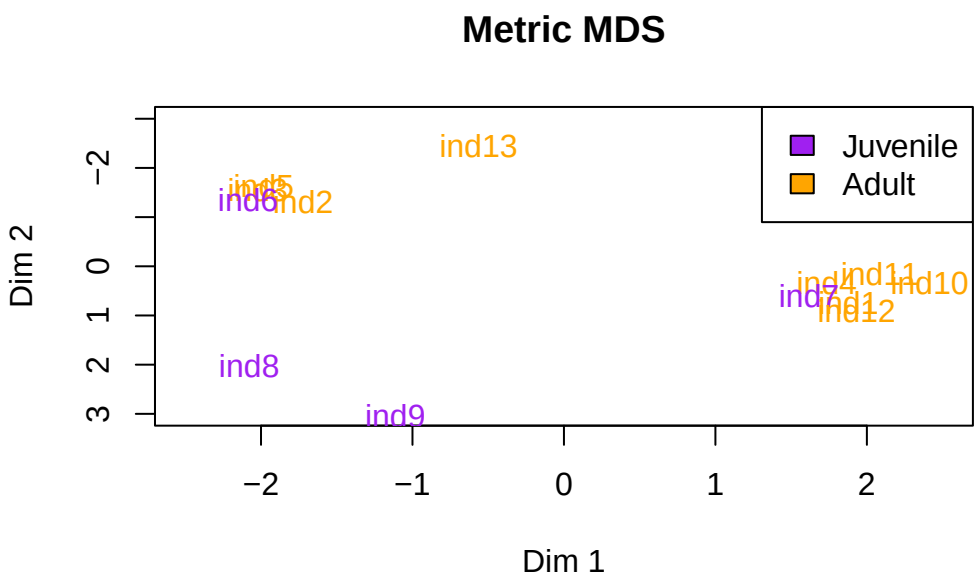


That's a bit better. We can also add some color to this plot if we want - say, individuals 6 to 9 are juveniles:

```
# Plot isoMDS
plot(mds2$points[,1], mds2$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS', # Labelling
     xlim = c(-2.5, 2.5), ylim = c(3, -3)) # Set axis limits

# Set random seed for consistency
set.seed(1212)

# Juvenile identifier
ad = c(rep(1,5), rep(0,4), rep(1,4)) # ad is 1 for first 5 and last 4
ad
```

```
 [1] 1 1 1 1 1 0 0 0 0 1 1 1 1
```

```r
# Plot individual names
text(mds2$points[,1] + rnorm(13, 0, 0.2), # Add random values pulled from a
     mds2$points[,2] + rnorm(13, 0, 0.2), # normal distribution with mean 0, sd 0.2
     rownames(data), # Add names
     col = ifelse(ad == 0, 'purple', 'orange')) # color
# Add a legend
legend('topright', legend = c('Juvenile', 'Adult'), fill = c('purple', 'orange'))
```



Does this plot match the previous one, and/or the cluster analyses?

12.4.2 Metric MDS

We can run metric MDS using the `cmdscale()` function:

```r
# run metric MDS
mds3 = cmdscale(dist, eig = T)
mds3
```

```
$points
             [,1]       [,2]
ind1   5.84977914  2.7981654
ind2  -7.46899061 -0.4452479
```

```
ind3  -7.87360160  2.4742095
ind4   6.04970828 -1.1964346
ind5  -6.77887791  2.8518073
ind6  -7.21161499  3.9150940
ind7   4.79289905 -0.9896933
ind8  -2.46317365 -5.3304368
ind9  -1.86216019 -9.7770302
ind10  5.45394235  1.1317599
ind11  5.27120921 -0.1097836
ind12  6.20052885  3.6063451
ind13  0.04035206  1.0712450

$eig
 [1]  4.150450e+02  1.794715e+02  1.684147e+02  1.309366e+02  6.931537e+01
 [6]  5.811388e+01  3.306204e+01  1.780496e+01 -4.263256e-14 -8.643935e+00
[11] -2.208342e+01 -4.468321e+01 -7.952271e+01

$x
NULL

$ac
[1] 0

$GOF
[1] 0.4844901 0.5545014
```

For metric MDS, we look at goodness of fit (GOF) instead of stress to assess how well the analysis worked. GOF is similar to an R^2 value, where numbers closer to 1 indicate a better fit (though be wary of overfitting!). There are two different GOF values for each metric MDS.

As with the other MDS functions, k defaults to 2. We can make another scree plot:

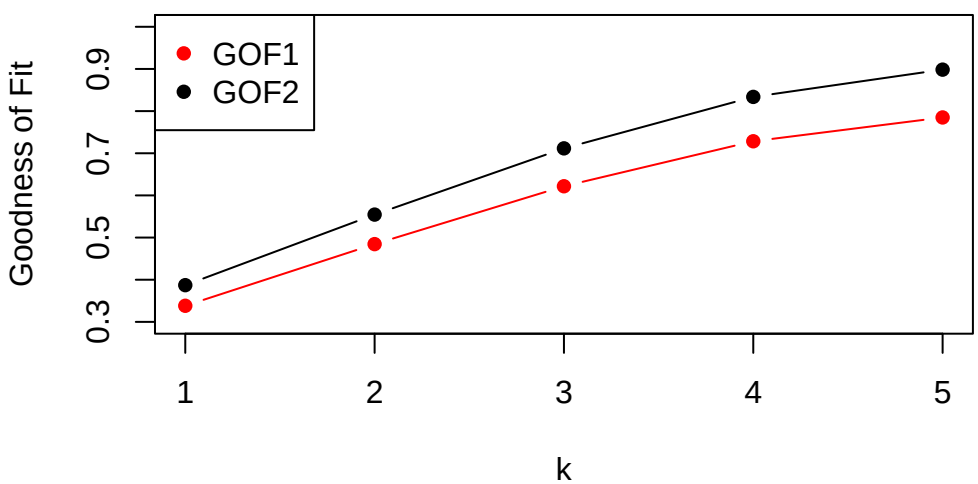
```r
# Create a container object
scree = data.frame(k = 1:5, GOF1 = NA, GOF2 = NA)

# Loop through k 1 to 5
for(k in 1:5){

  # Run MDS
  mds = cmdscale(dist, eig = T, k = k) # Set k to our loop index

  # Pull out stress
  scree[k,c(2,3)] = mds$GOF # Fill kth row of the GOF columns in scree
```

```r
} # End loop
```

```r
# Print results
scree
```

```
  k      GOF1      GOF2
1 1 0.3382331 0.3871096
2 2 0.4844901 0.5545014
3 3 0.6217365 0.7115806
4 4 0.7284408 0.8337043
5 5 0.7849281 0.8983543
```

```r
# Make scree plot
plot(GOF2 ~ k, data = scree, # Plot stress against k
     type = 'b', # Lines and points
     pch = 16, # Point 16 (filled circle)
     ylab = 'Goodness of Fit', ylim = c(0.3, 1))
points(GOF1 ~ k, data = scree, type = 'b', pch = 16, col = 'red') # Add second GOF value
abline(h = 0.1, lty = 'dashed') # Plot a dashed line at 0.1
legend('topleft', pch = 16, legend = c('GOF1', 'GOF2'), col = c('red', 'black')) # Add legend
```



Goodness of fit scales linearly, so what k to use is more of a judgement call.

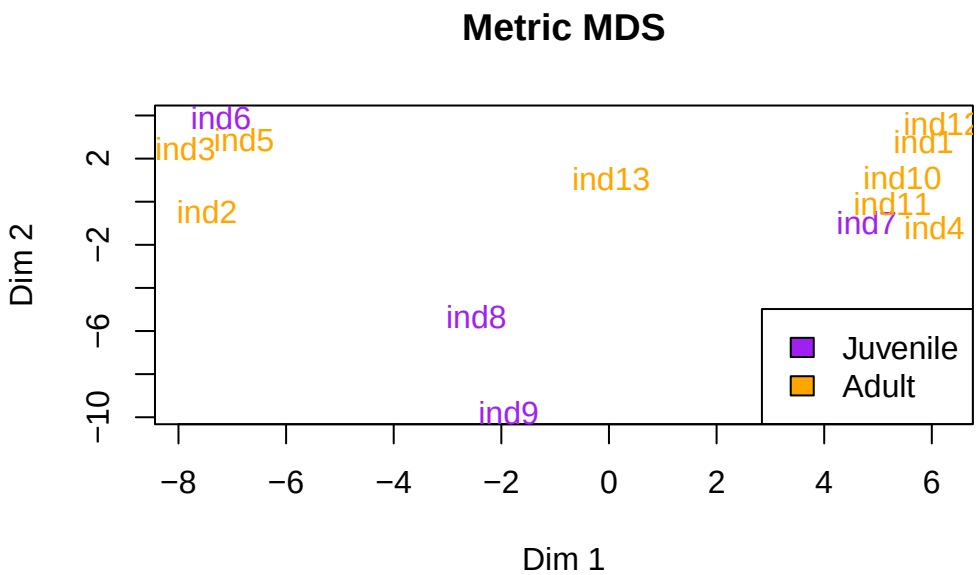
```r
# run metric MDS
mds3 = cmdscale(dist, k=4, eig = T)
mds3
```

```
$points
             [,1]       [,2]       [,3]       [,4]
ind1   5.84977914  2.7981654  1.2685787 -0.6738375
ind2  -7.46899061 -0.4452479  2.6496404  2.1091477
ind3  -7.87360160  2.4742095  3.4326421  1.2339175
ind4   6.04970828 -1.1964346 -0.9306802 -1.2319775
ind5  -6.77887791  2.8518073 -1.2465315  2.3989342
ind6  -7.21161499  3.9150940 -1.9699687 -2.4369605
ind7   4.79289905 -0.9896933 -2.7067201 -0.2043847
ind8  -2.46317365 -5.3304368 -9.4034890  2.0126521
ind9  -1.86216019 -9.7770302  5.2905629 -1.1532342
ind10  5.45394235  1.1317599  0.3565324  4.1657272
ind11  5.27120921 -0.1097836  4.1678975  1.4238728
ind12  6.20052885  3.6063451 -0.2995254  1.5298300
ind13  0.04035206  1.0712450 -0.6089393 -9.1736869

$eig
 [1]  4.150450e+02  1.794715e+02  1.684147e+02  1.309366e+02  6.931537e+01
 [6]  5.811388e+01  3.306204e+01  1.780496e+01 -4.263256e-14 -8.643935e+00
[11] -2.208342e+01 -4.468321e+01 -7.952271e+01

$x
NULL

$ac
[1] 0

$GOF
[1] 0.7284408 0.8337043
```

Let's plot the first two dimensions:

```r
# Plot metric MDS
plot(mds3$points[,1], mds3$points[,2], # MDS dimension 1 and 2 values
     type = 'n', # Don't plot any points
     xlab = 'Dim 1', ylab = 'Dim 2', main = 'Metric MDS') # Labelling
```

```r
# Plot individual names
text(mds3$points[,1], # Add random values pulled from a
     mds3$points[,2], # normal distribution with mean 0, sd 0.2
     rownames(data), # Add names
     col = ifelse(ad == 0, 'purple', 'orange')) # color
# Add a legend
legend('bottomright', legend = c('Juvenile', 'Adult'), fill = c('purple', 'orange'))
```

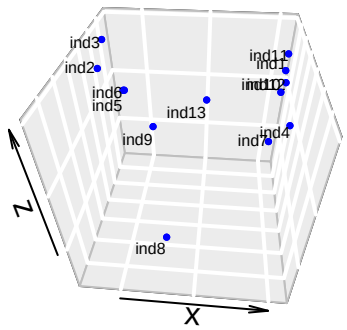


12.4.3 3D Plotting (Optional)

It may not be necessary, but if your MDS has more than 2 dimensions, you can try plotting it in three dimensions and see if it helps:

```r
library(plot3D)

# Prepare data to plot
x = mds3$points[,1]
y = mds3$points[,2]
z = mds3$points[,3]

# Create 3D plot
```

```
scatter3D(x,y,z, colvar = NULL, col = 'blue',
          pch = 16, cex = 0.5, bty = 'g', theta = 5)

# Add text
text3D(x,
       # Add some jiggle to the labels
       y+rnorm(13, mean = 0, sd = 0.5), z + rnorm(13, mean = 0, sd = 0.5),
                    labels = names(mds3$points[,1]), add = T, colkey = F,
                    cex = 0.5, adj = 1, d = 2)
```



12.5 Mantel Test (Graduate Students Only)

We can infer to some extent whether juveniles and adults preferentially associate with each other from our colored MDS plots, but we can also test it statistically using a Mantel test. To run the Mantel test, we need to convert our adult index into a `dist` object:

```
# Create dist matrix for adults
ad_dist = dist(ad)
ad_dist
```

```
   1 2 3 4 5 6 7 8 9 10 11 12
2  0
3  0 0
4  0 0 0
5  0 0 0 0
6  1 1 1 1 1
7  1 1 1 1 1 0
8  1 1 1 1 1 0 0
9  1 1 1 1 1 0 0 0
10 0 0 0 0 0 1 1 1 1
11 0 0 0 0 0 1 1 1 1  0
12 0 0 0 0 0 1 1 1 1  0  0
13 0 0 0 0 0 1 1 1 1  0  0  0
```

Note this is dissimilarity: adult-juvenile pairs are assigned 1, and same-class pairs are assigned 0.

The Mantel test looks for correlation between this matrix and our original dissociation matrix, and statistically tests if the associations are different from what we would expect due to chance.

```r
# Run mantel test
library(ade4)
mantel.rtest(ad_dist, dist, nrepet = 999)
```

```
Warning in is.euclid(m1): Zero distance(s)

Warning in is.euclid(m2): Zero distance(s)

Monte-Carlo test
Call: mantelnoneuclid(m1 = m1, m2 = m2, nrepet = nrepet)

Observation: 0.1686576

Based on 999 replicates
Simulated p-value: 0.073
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
 1.369210491 -0.001062026  0.015364686
```

It's very close, but we don't have statistically significant evidence that juveniles and adults associate preferentially with each other in this case.

12.6 Tips for your Assignment:

Some things you may want to think about for your assignment:

1. How would you pick which cluster analyses and MDS analyses are best for your data? Are they conceptual, or do they have to do with the results? Do they agree?

2. How would you interpret your statistical results biologically? You don't have to be right, but don't be vague, and don't contradict your results.

13 Assignment 1d: Multiple Linear Regression

This assignment is all about multiple linear regression. Linear regression is used to model relationships between a dependent (response) variable and one or more independent (predictor) variables. Multiple linear regression involves multiple predictor variables.

For this tutorial we're going to use `Schoenemann.csv`, derived from the data in this paper.

13.1 Looking at the data

```
# Read in data
data = read.csv('Schoenemann.csv')

# View data structure
head(data)
```

```
      Order     Family   Genus    Species Location   Mass    Fat   FFWT    CNS
1 Carnivora    Felidae   Felis canadensis   Alaska 7688.0 1120.0 6568.0 105.09
2 Carnivora    Felidae   Felis      rufus Virginia 6152.0  738.0 5414.0  81.75
3 Carnivora Mustelidae    Gulo     luscus   Alaska 9362.0  562.0 8800.0  85.36
4 Carnivora Mustelidae Mustela    erminea   Alaska  183.3    3.1  180.2   6.69
5 Carnivora Mustelidae Mustela      vison Virginia 1032.0   66.0  966.0  18.06
6 Carnivora Proyonidae Procyon      lotor Virginia 6040.0 1013.0 5027.0  58.31
  HEART  MUSCLE   BONE
1 27.59 4341.45 631.18
2 25.45 3600.31 552.23
3 80.96 5271.20 879.12
4  1.87  104.70  21.98
5  7.63  581.53  80.27
6 36.19 2920.69 517.78
```

```
dim(data)
```

```
[1] 39 12
```

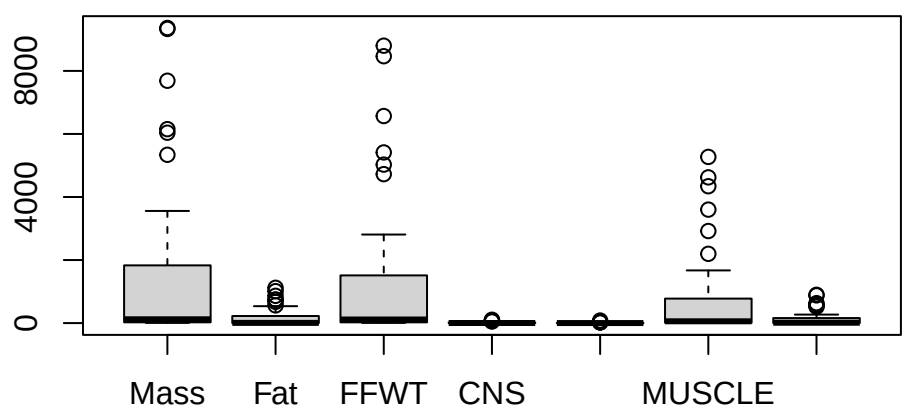
The Schoenemann dataset contains 39 observations of 12 variables, describing to the morphometry of different species of mammals, along with taxonomic and location information. Let's start by getting rid of the non-morphological data. We won't need it for this assignment.

```
# Remove metadata
data = data[,which(colnames(data) == 'Mass'):ncol(data)] # I do it this way to avoid hard co

# check if it worked
head(data)
```

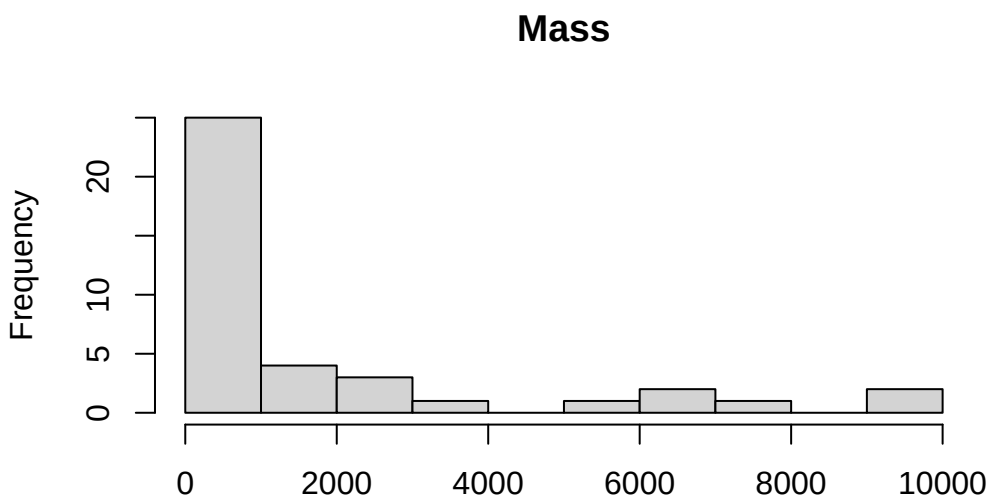
```
    Mass    Fat   FFWT    CNS HEART  MUSCLE   BONE
1 7688.0 1120.0 6568.0 105.09 27.59 4341.45 631.18
2 6152.0  738.0 5414.0  81.75 25.45 3600.31 552.23
3 9362.0  562.0 8800.0  85.36 80.96 5271.20 879.12
4  183.3    3.1  180.2   6.69  1.87  104.70  21.98
5 1032.0   66.0  966.0  18.06  7.63  581.53  80.27
6 6040.0 1013.0 5027.0  58.31 36.19 2920.69 517.78
```

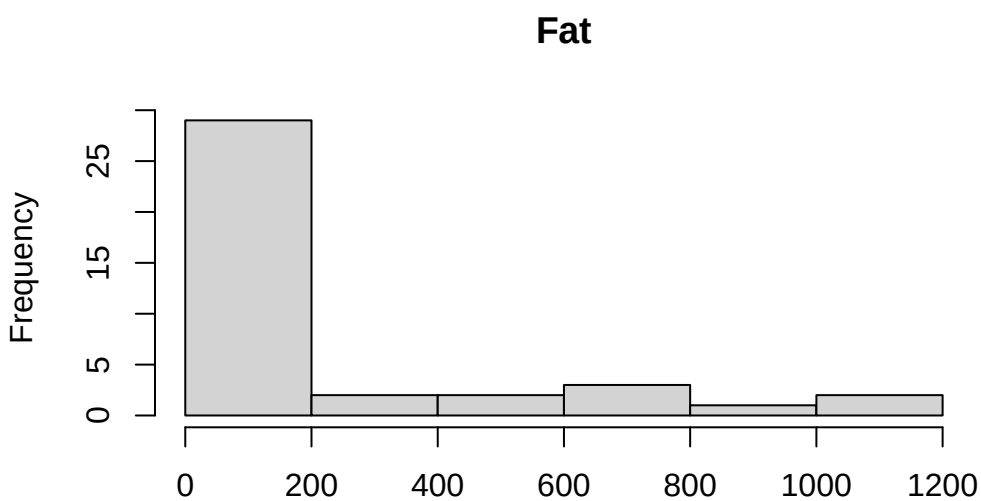
Now we only have the numeric morphologicall data left. Let's take a look at the data graphically.

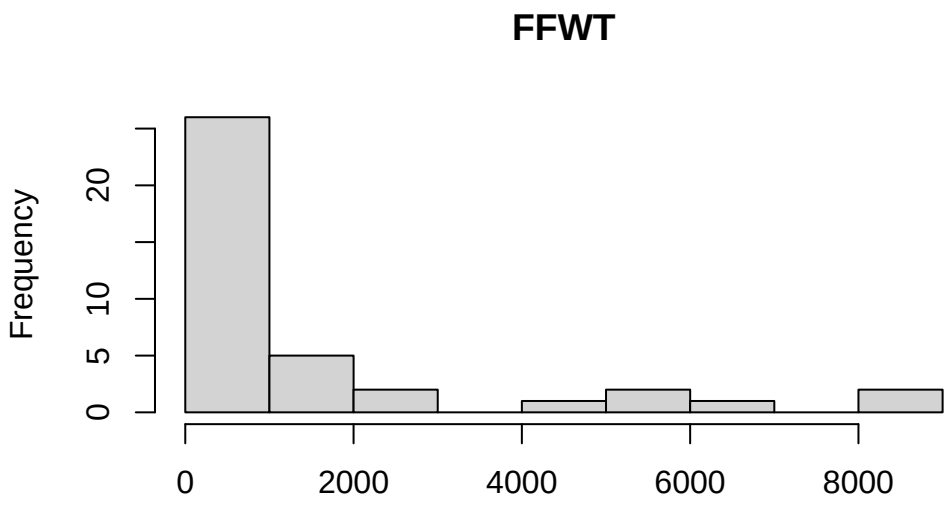
```
# Visualize the data
boxplot(data)
```

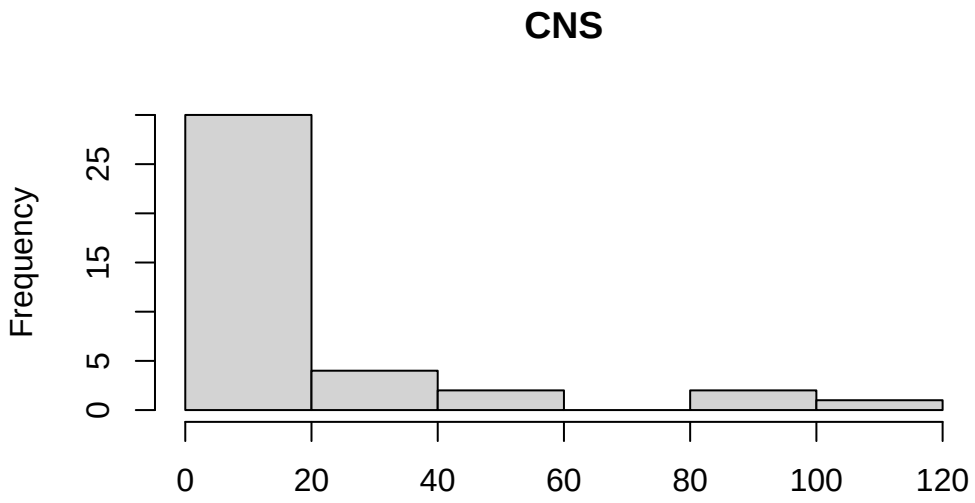


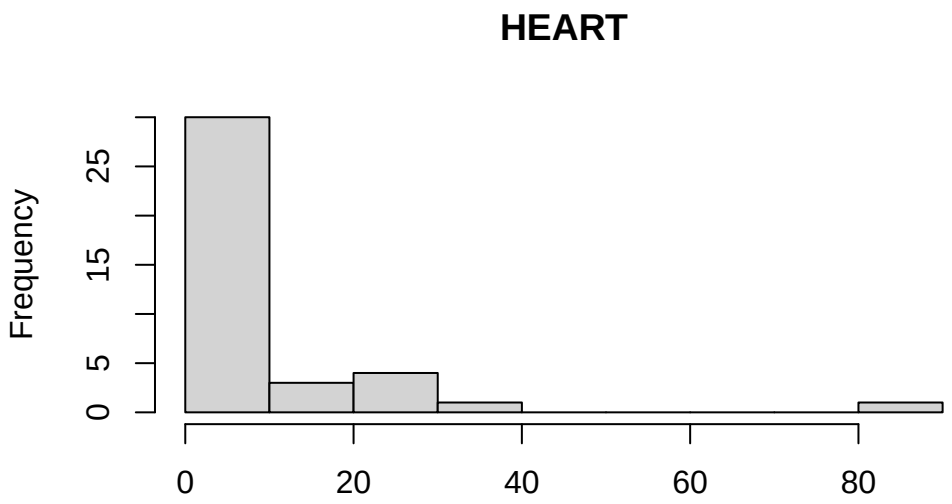
```r
# Loop through columns to create histograms
for(i in 1:ncol(data)){hist(data[,i], main = colnames(data)[i], xlab = "")} # Name histogram
```

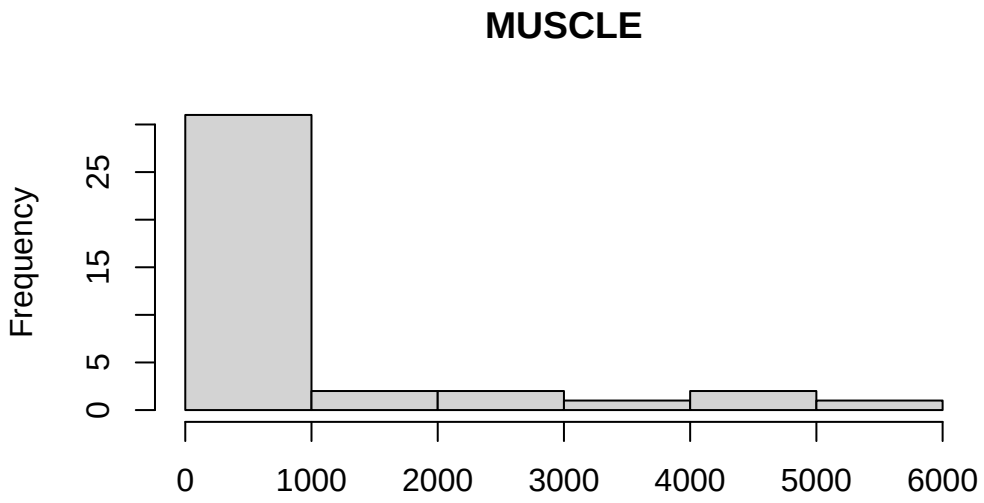


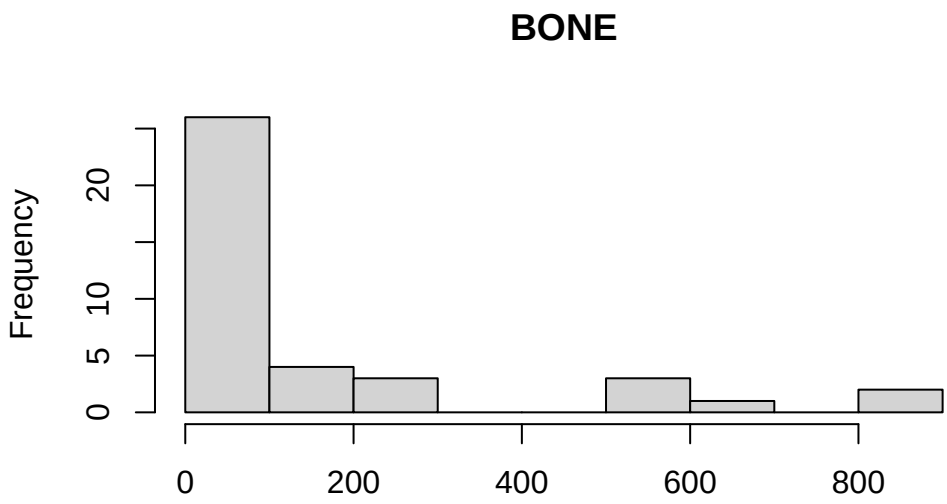



FFWT
Frequency
0
5
10
20
0
2000
4000
6000
8000


CNS
Frequency
0
5
15
25
0
20
40
60
80
100
120


HEART
Frequency
0
5
15
25
0
20
40
60
80

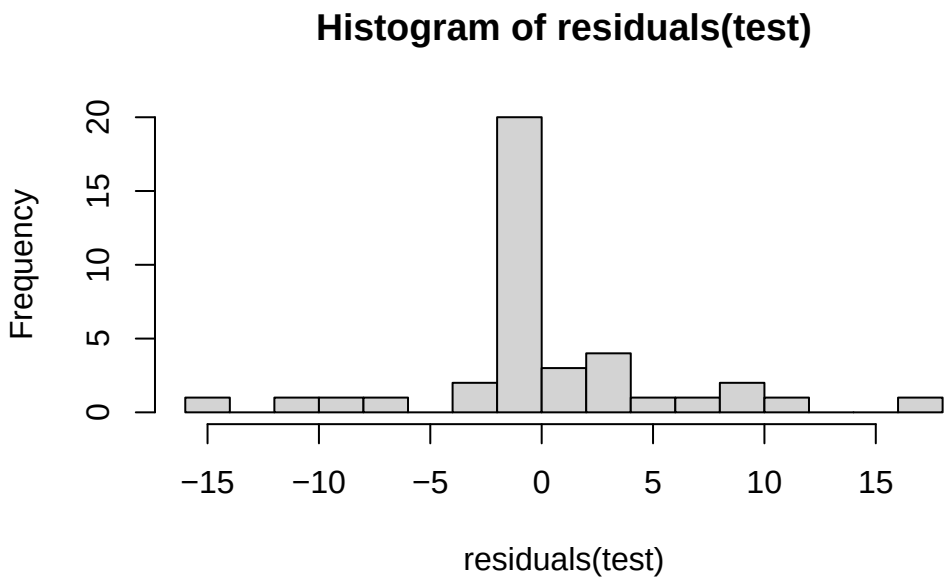

MUSCLE
Frequency
0
5
15
25
0
1000
2000
3000
4000
5000
6000



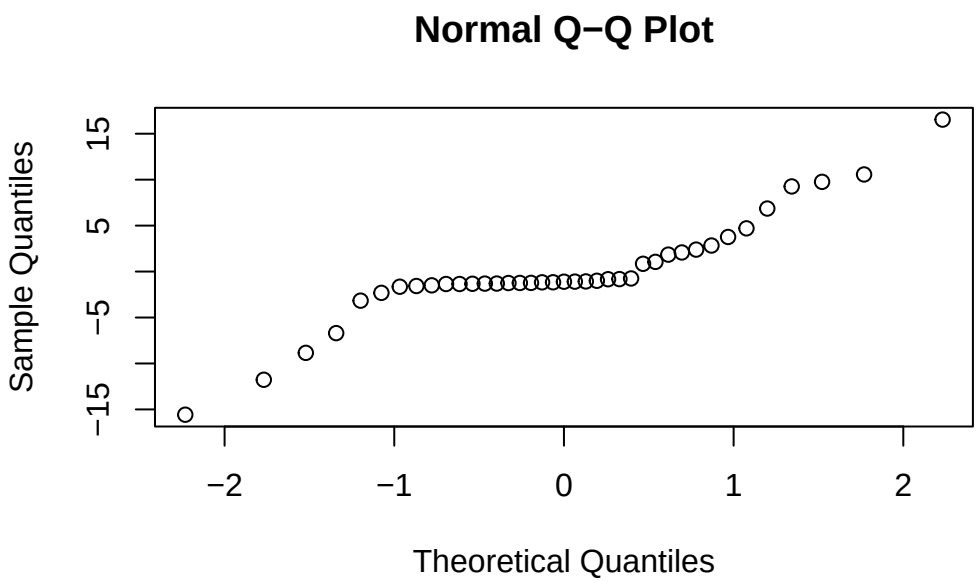
13.2 Considering Transformations

We can see that these data have many more small values, and the data at higher values has higher variance [try e.g. plot(data$Mass, data$Fat) to see this]. If we run our regressions on these data, the assumption of heteroscedasticity is going to be violated:

```
# Run a test model and check assumptions
test = lm(CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE, data = data)

# Check for normality as an example

# Residual histogram
hist(residuals(test), 20)
```




```r
# QQplot
qqnorm(residuals(test))
```



```r
# Statistical test for normality
shapiro.test(residuals(test))
```

```
    Shapiro-Wilk normality test

data:  residuals(test)
W = 0.87473, p-value = 0.0004494
```

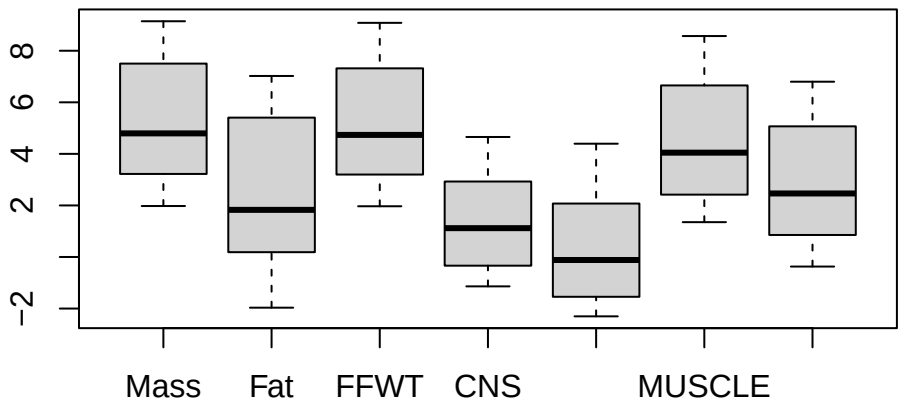
These diagnostics look… less than ideal.

This is a textbook case of when to apply a log transformation to standardize variance - remember logging is the opposite of exponentiating:

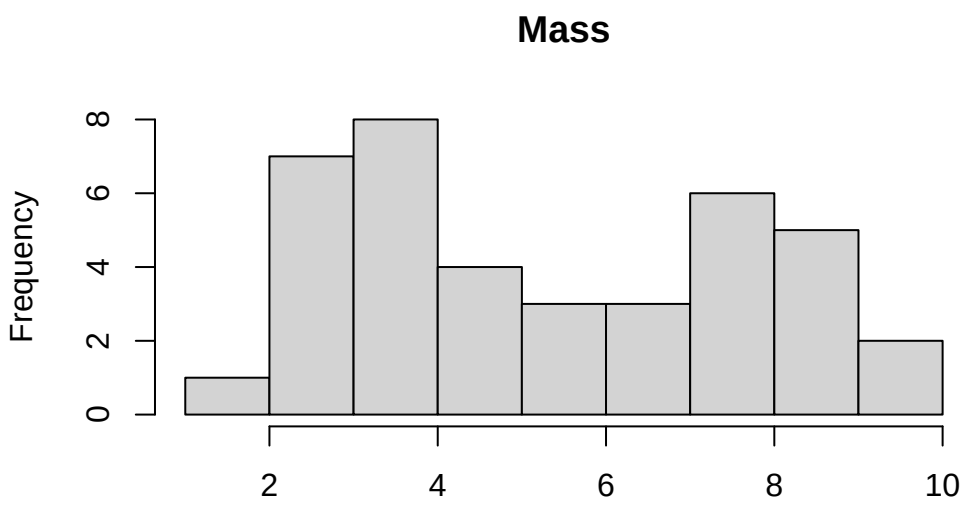
```r
# Apply log transformation
data_l = log(data)

# Check out the new data
head(data_l)
```

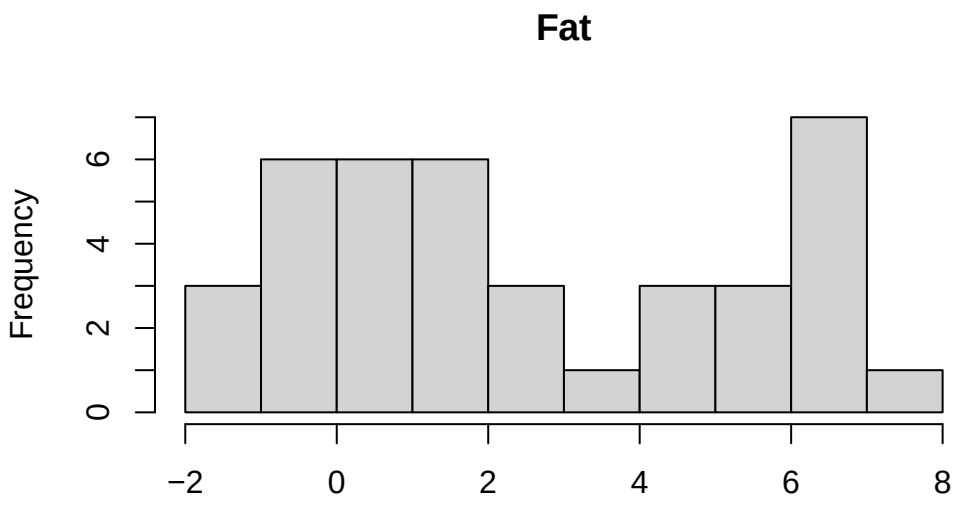
```
      Mass      Fat     FFWT      CNS     HEART   MUSCLE     BONE
1 8.947416 7.021084 8.789965 4.654817 3.3174534 8.375964 6.447591
2 8.724533 6.603944 8.596743 4.403666 3.2367157 8.188775 6.313965
3 9.144414 6.331502 9.082507 4.446878 4.3939552 8.570013 6.778921
4 5.211124 1.131402 5.194067 1.900614 0.6259384 4.651099 3.090133
5 6.939254 4.189655 6.873164 2.893700 2.0320878 6.365663 4.385396
6 8.706159 6.920672 8.522579 4.065774 3.5887828 7.979575 6.249550
```

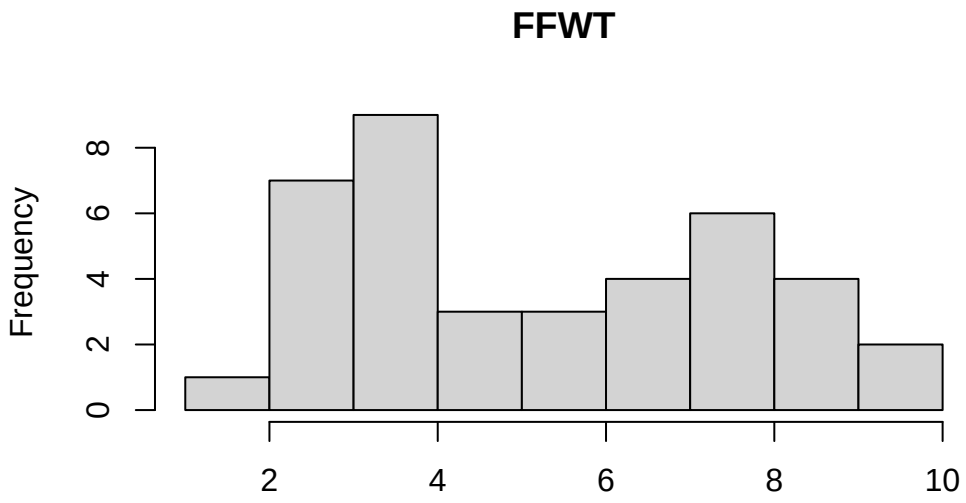
```r
# Looking at the data
boxplot(data_l)
```

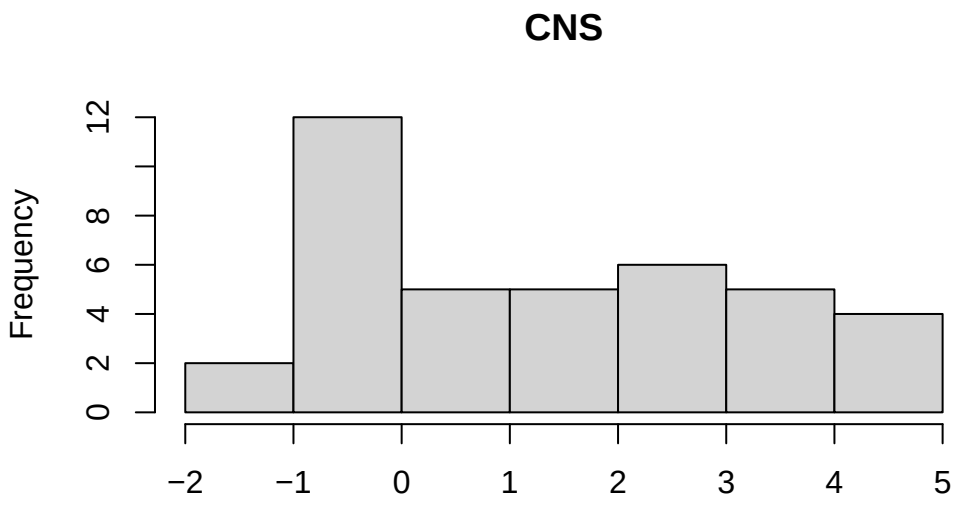


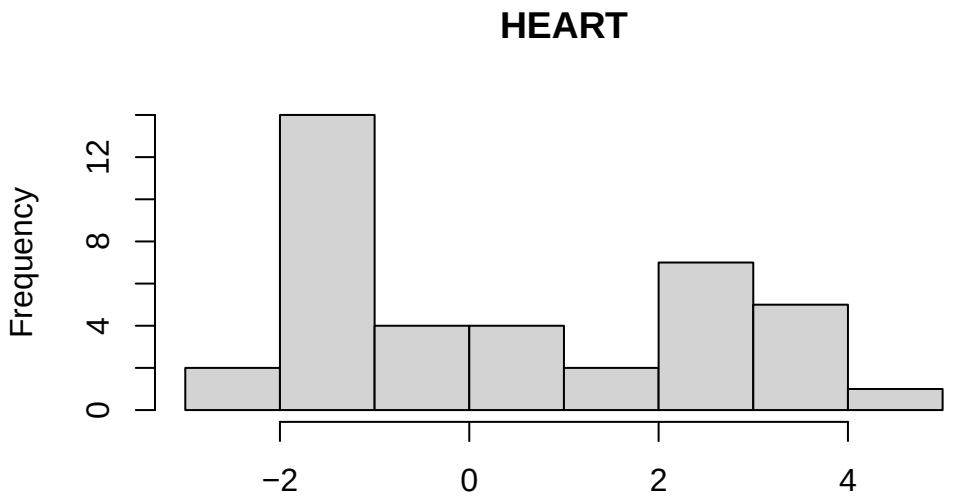

```
# Loop through columns to create histograms
for(i in 1:ncol(data_l)){hist(data_l[,i], main = colnames(data_l)[i], xlab = "")} # Name hist
```

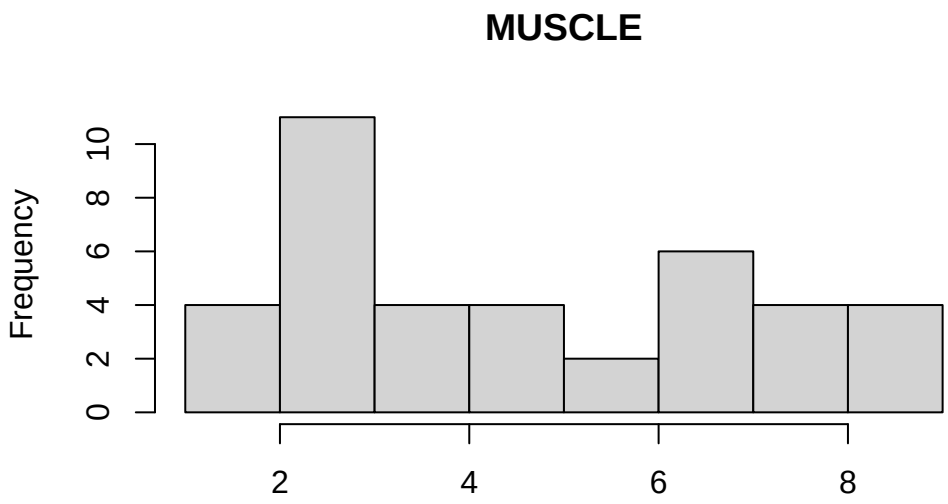


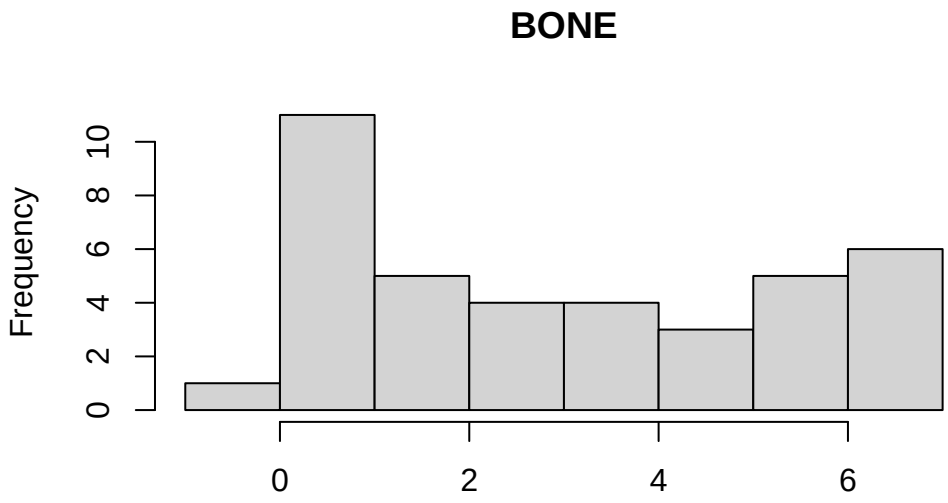

Fat
Frequency
0
2
4
6
−2
0
2
4
6
8


FFWT
Frequency
0
2
4
6
8
2
4
6
8
10


CNS
Frequency
0
2
4
6
8
12
−2
−1
0
1
2
3
4
5


HEART
Frequency
0
4
8
12
−2
0
2
4





Now our data looks much more uniform.

Always remember that transforming your data incorrectly or unnecessarily can do more harm than good. How do you decide if it is helpful to transform your data? What

is the purpose of transforming your data? Think carefully about these questions for your assignment when you're deciding whether to transform any data.

13.3 Simple Linear Regression

Now that our data is good to go, we're going to run some simple linear regressions on the logged data to predict central nervous system mass (CNS). Simple linear regressions only have one predictor variable, so we will run separate models for each predictor. Linear regression is run using the `lm()` command:

```
# Run simple linear regressions - Mass
m1 = lm(CNS ~ Mass, data = data_l) # run model
summary(m1) # model summary
```

```
Call:
lm(formula = CNS ~ Mass, data = data_l)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.77785 -0.20227 -0.05439  0.19607  0.78453 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.79097    0.14844  -18.80   <2e-16 ***
Mass         0.77105    0.02556   30.16   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3657 on 37 degrees of freedom
Multiple R-squared:  0.9609,	Adjusted R-squared:  0.9599 
F-statistic: 909.7 on 1 and 37 DF,  p-value: < 2.2e-16
```

```
# Run simple linear regressions - Fat
m2 = lm(CNS ~ Fat, data = data_l) # run model
summary(m2) # model summary
```

```
Call:
lm(formula = CNS ~ Fat, data = data_l)
```

```
Residuals:
     Min       1Q   Median       3Q      Max 
-1.22918 -0.41510  0.01431  0.36008  1.38000 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.15712    0.13223  -1.188    0.242    
Fat          0.59903    0.03518  17.028   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6223 on 37 degrees of freedom
Multiple R-squared:  0.8868,	Adjusted R-squared:  0.8838 
F-statistic: 289.9 on 1 and 37 DF,  p-value: < 2.2e-16
```

```r
# Run simple linear regressions - FFWT
m3 = lm(CNS ~ FFWT, data = data_l) # run model
summary(m3) # model summary
```

```
Call:
lm(formula = CNS ~ FFWT, data = data_l)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.8057 -0.2112 -0.0535  0.1907  0.7654 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.80193    0.14382  -19.48   <2e-16 ***
FFWT         0.78494    0.02515   31.20   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3539 on 37 degrees of freedom
Multiple R-squared:  0.9634,	Adjusted R-squared:  0.9624 
F-statistic: 973.7 on 1 and 37 DF,  p-value: < 2.2e-16
```

```r
# Run simple linear regressions - HEART
m4 = lm(CNS ~ HEART, data = data_l) # run model
summary(m4) # model summary
```

```
Call:
lm(formula = CNS ~ HEART, data = data_l)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.75646 -0.16000 -0.03248  0.15018  0.85234 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.99514    0.05853   17.00   <2e-16 ***
HEART        0.88201    0.02872   30.71   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3594 on 37 degrees of freedom
Multiple R-squared:  0.9622,	Adjusted R-squared:  0.9612 
F-statistic:   943 on 1 and 37 DF,  p-value: < 2.2e-16
```

```r
# Run simple linear regressions - MUSCLE
m5 = lm(CNS ~ MUSCLE, data = data_l) # run model
summary(m5) # model summary
```

```
Call:
lm(formula = CNS ~ MUSCLE, data = data_l)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.82059 -0.15588 -0.00489  0.17331  0.80475 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.20579    0.11856  -18.61   <2e-16 ***
MUSCLE       0.76488    0.02296   33.31   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3323 on 37 degrees of freedom
Multiple R-squared:  0.9677,	Adjusted R-squared:  0.9669 
F-statistic:  1109 on 1 and 37 DF,  p-value: < 2.2e-16
```

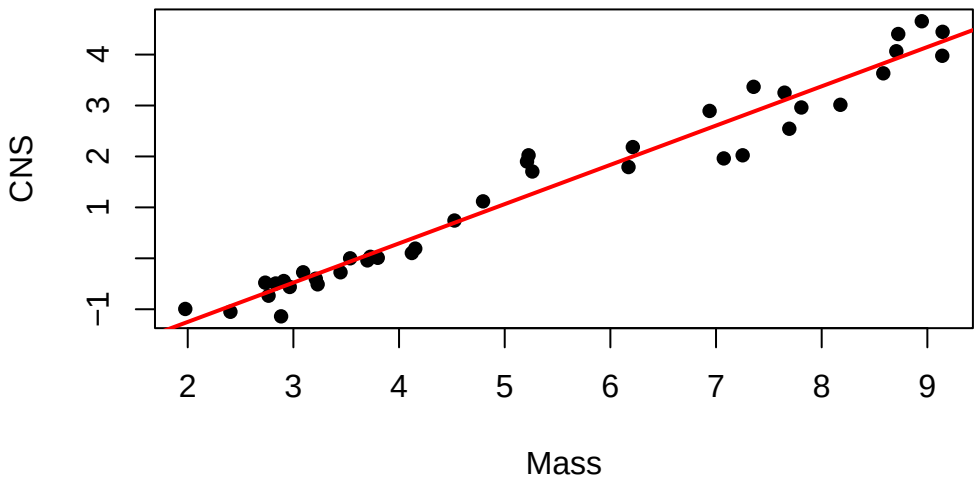
```r
# Run simple linear regressions - BONE
m6 = lm(CNS ~ BONE, data = data_l) # run model
summary(m6) # model summary
```

```
Call:
lm(formula = CNS ~ BONE, data = data_l)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.10309 -0.24611  0.01155  0.25195  0.63931 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -1.0497     0.1042  -10.07 3.75e-12 ***
BONE          0.7856     0.0277   28.36  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3879 on 37 degrees of freedom
Multiple R-squared:  0.956, Adjusted R-squared:  0.9548 
F-statistic: 804.4 on 1 and 37 DF,  p-value: < 2.2e-16
```

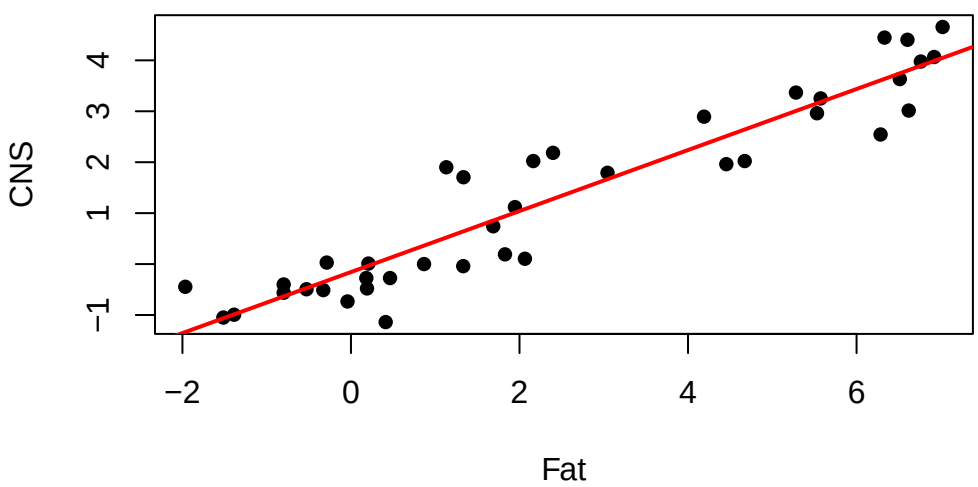
In this case, it looks like all of our variables are strong, significant predictors with high R^2 values.

Let's plot all of these regressions:

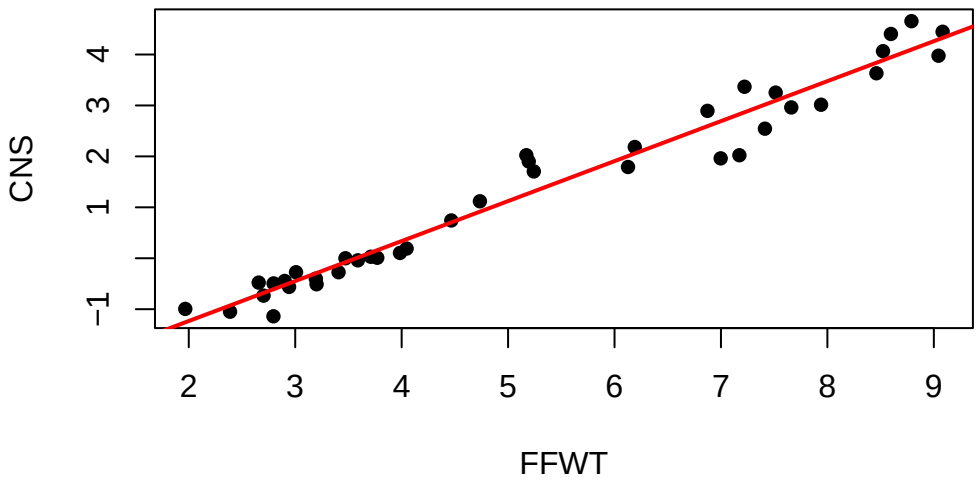
```r
# Plot regressions

# Plot simple linear regressions - Mass
plot(CNS ~ Mass, data = data_l, pch = 16) # plot points
abline(m1, lwd = 2, col = 'red') # Plot model
```



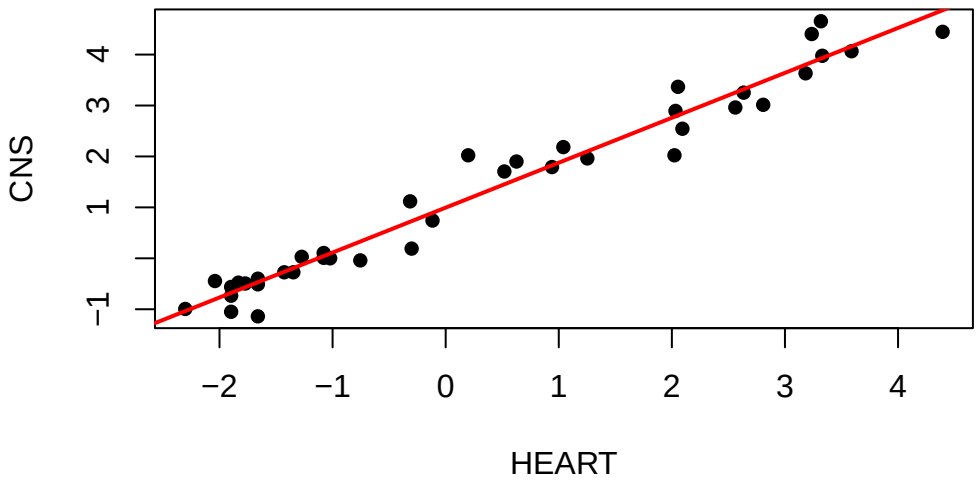

```
# Plot simple linear regressions - Fat
plot(CNS ~ Fat, data = data_l, pch = 16) # plot points
abline(m2, lwd = 2, col = 'red') # Plot model
```



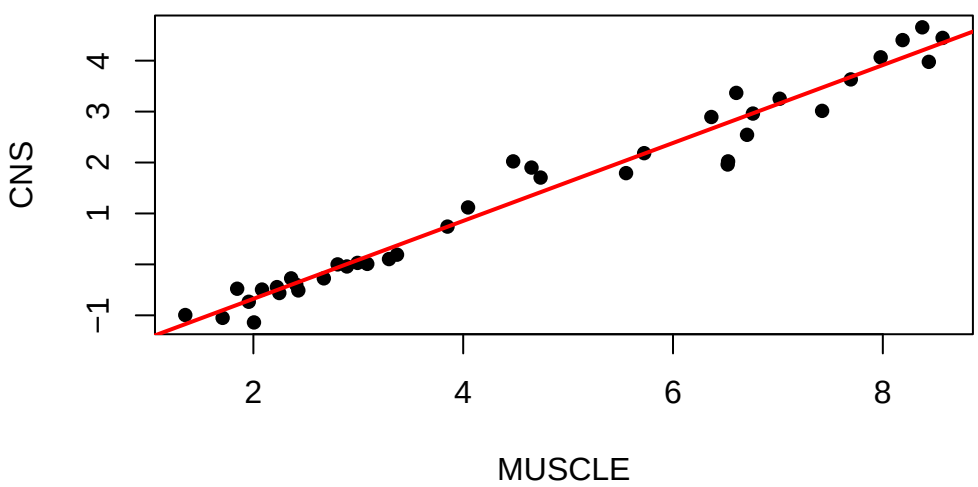
```r
# Plot simple linear regressions - FFWT
plot(CNS ~ FFWT, data = data_l, pch = 16) # plot points
abline(m3, lwd = 2, col = 'red') # Plot model
```



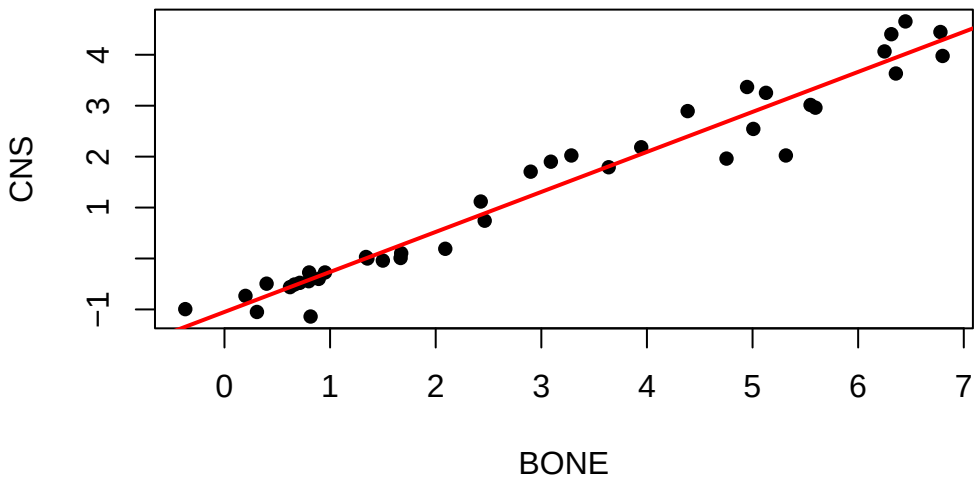

```r
# Plot simple linear regressions - HEART
plot(CNS ~ HEART, data = data_l, pch = 16) # plot points
abline(m4, lwd = 2, col = 'red') # Plot model
```




```
# Plot simple linear regressions - MUSCLE
plot(CNS ~ MUSCLE, data = data_l, pch = 16) # plot points
abline(m5, lwd = 2, col = 'red') # Plot model
```



```r
# Plot simple linear regressions - BONE
plot(CNS ~ BONE, data = data_l, pch = 16) # plot points
abline(m6, lwd = 2, col = 'red') # Plot model
```



All of the regression slopes are positive. This makes sense - larger animals tend to have larger brains. Remember to always think about whether your results make biological sense.

13.4 Multiple Linear Regression

We've made 6 models using 1 variable. Now, let's try making 1 model with 6 variables:

```r
# Run full model
m7 = lm(CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE, data = data_l)
summary(m7)
```

```
Call:
lm(formula = CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE,
    data = data_l)

Residuals:
```

```
     Min       1Q   Median       3Q      Max 
-0.72690 -0.12073  0.00376  0.08672  0.85638 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) -0.61138    1.18496  -0.516   0.6094  
Mass        -0.46867    2.14250  -0.219   0.8282  
Fat         -0.06818    0.15489  -0.440   0.6628  
FFWT        -0.02606    2.30347  -0.011   0.9910  
HEART        0.41894    0.21913   1.912   0.0649 .
MUSCLE       1.03524    0.61123   1.694   0.1000  
BONE        -0.06339    0.32054  -0.198   0.8445  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3286 on 32 degrees of freedom
Multiple R-squared:  0.9727,	Adjusted R-squared:  0.9676 
F-statistic: 190.1 on 6 and 32 DF,  p-value: < 2.2e-16
```

Our full model has a very(!) high R^2 value, and, in contrast to the simple linear regressions where every predictor was significant, none of our predictors are considered significant in the final model at $\alpha = 0.05$. Why do you think that is?

13.5 Checking Assumptions

Now that we've run our full model, it's time to check its assumptions. Those assumptions are **Independence, Linearity, Homoscedasticity, and Normality.** By now, you should be familiar with what these all mean, but let's run through them anyways:

13.5.1 Independence

The assumption of independence states that the value of each data point ('datum', if you will) is independent of all other data points. Some of the ways in which it could be violated may not be testable (e.g. if they have to do with how the data was collected), but what we *can* test for is **autocorrelation**. Autocorrelation translates to self correlation (auto = self). We can test for autocorrelation statistically using a Durbin-Watson test, and visually using an autocorrelation function on the residuals:

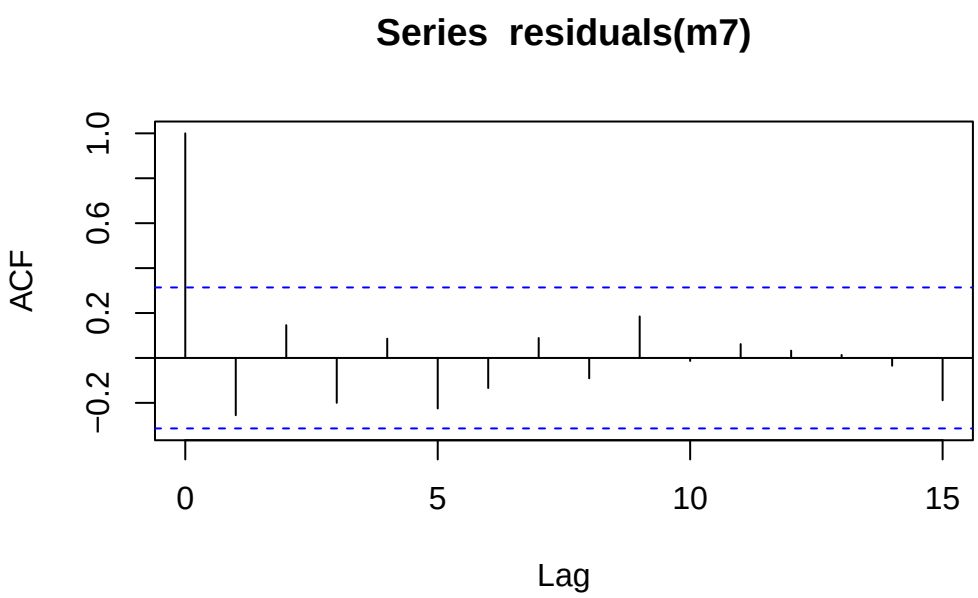
```r
library(lmtest)
```

```
Loading required package: zoo


Attaching package: 'zoo'


The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

```r
# Durbin-watson test
dwtest(m7)
```

```
    Durbin-Watson test

data:  m7
DW = 2.4315, p-value = 0.8545
alternative hypothesis: true autocorrelation is greater than 0
```

```r
# Autocorrelation function
acf(residuals(m7))
```



The Durbin-Watson test returns an insignificant p-value, indicating no autocorrelation structure is present. The ACF plots the correlation coefficient of the data against itself using lags. Lag 0 correlates the data against itself, which is always 1. Lag 1 correlates each data point against the point after it, and so on. All of the correlation coefficients are between the blue lines, so again, we have no autocorrelation structure, and we can say independence is respected.

13.5.2 Linearity

The assumption of linearity states that the response variable consistently scales linearly with its predictors. We can test for linearity statistically using Ramsey's RESET test on our model:

```
# Run RESET test
resettest(m7)
```

```
    RESET test

data:  m7
RESET = 0.12784, df1 = 2, df2 = 30, p-value = 0.8805
```

In this case, the p-value is not significant, meaning the assumption of linearity is respected.

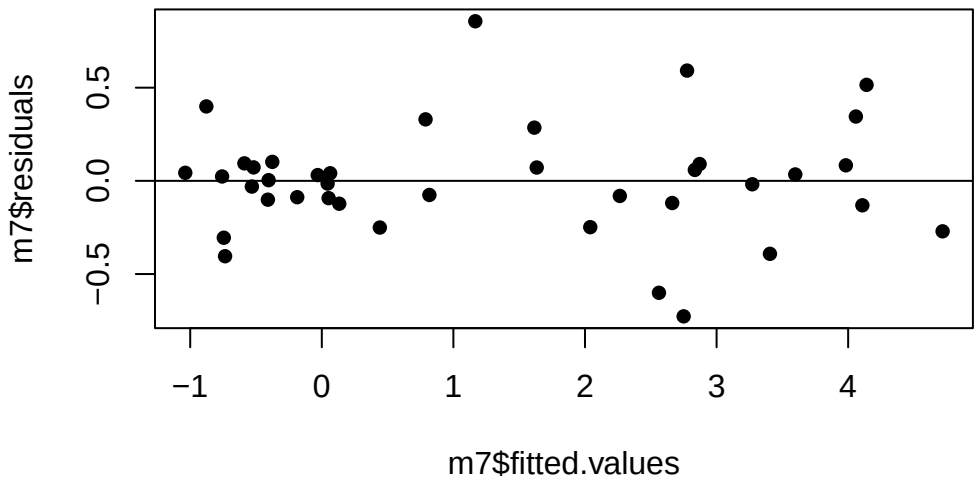
13.5.3 Homoscedasticity

The assumption of homoscedasticity is that the variance in the data is independent of the value of the data - i.e. the variance in the data is consistent. We can test this statistically using the Breusch-Pagan test, and visually by plotting the model residuals against the fitted values.

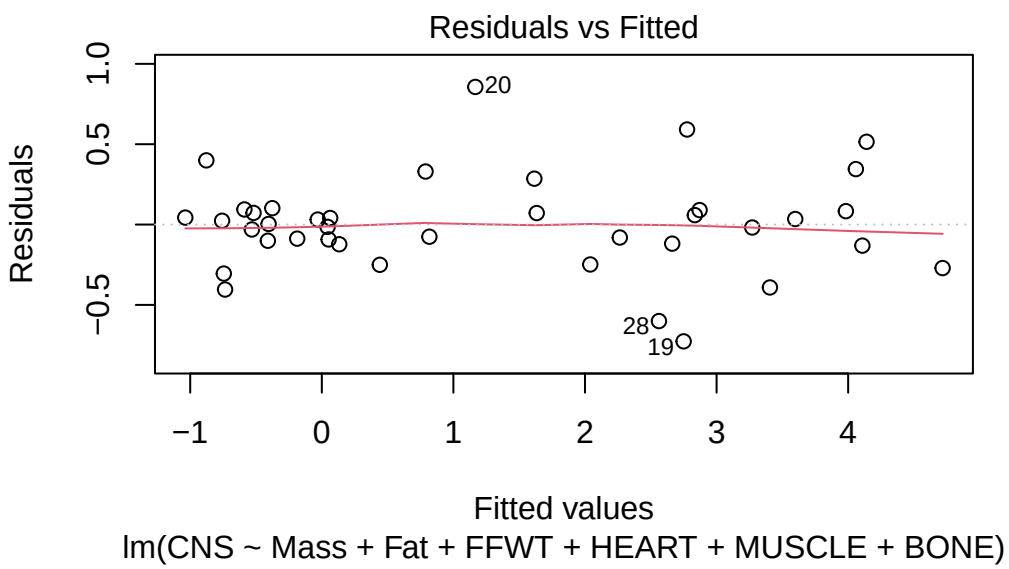
```
# Run Breusch-Pagan test
bptest(m7)
```

```
    studentized Breusch-Pagan test

data:  m7
BP = 15.974, df = 6, p-value = 0.01389
```

```
# Plot residuals vs fitted
plot(m7$residuals ~ m7$fitted.values, pch = 16); abline(h = 0)
```




```
# Can also be done using plot.lm, ?plot.lm for details
plot(m7, 1)
```

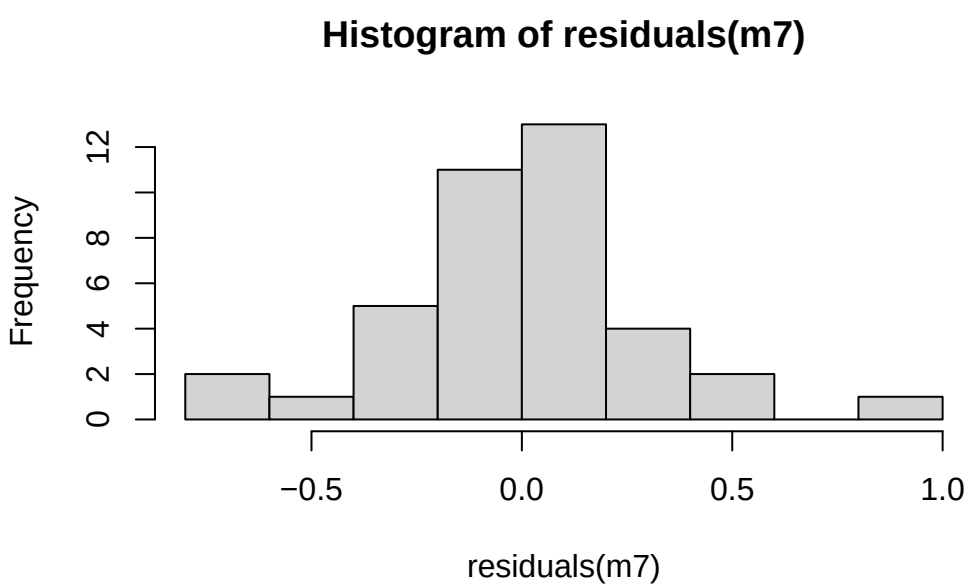


The Breusch-Pagan returns a significant p-value, indicating the assumption of homoscedasticity is violated. We can see in the residuals versus fitted plot that the variance in the data is smaller at low values than it is at higher values (the points on the left of the plot are clustered more closely than they are on the right). Let's come back to this later.

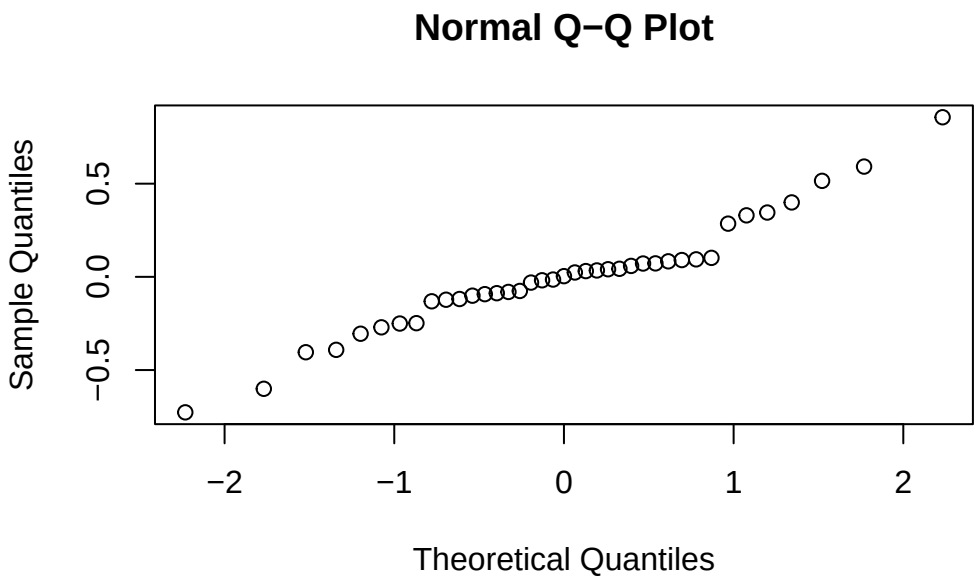
13.5.4 Normality

The assumption of normality states that the residuals of our model should be normally distributed. If they aren't, that would indicate that our model is biased towards overprediction or underprediction in some way. As we did earlier in the transformation section, we can check for normality visually by looking at histograms and QQ plots of our residuals, and statistically by running a Shapiro-Wilk test on the residuals.

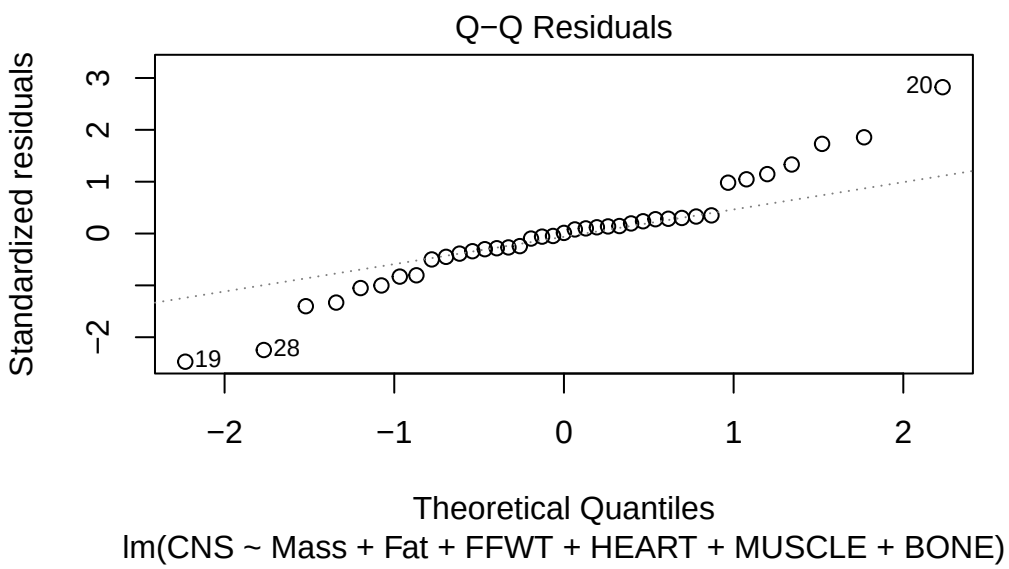
```
# Residual histogram
hist(residuals(m7))
```




```
# QQplot
qqnorm(residuals(m7))
```




```r
# Can also use plot.lm for qqplot
plot(m7, 2)
```



```r
# Statistical test for normality
shapiro.test(residuals(m7))
```

```
    Shapiro-Wilk normality test

data:  residuals(m7)
W = 0.95391, p-value = 0.1113
```

The Shapiro-Wilk test p-value is not significant (though it comes close), meaning the assumption of normality is respected. The residual histogram largely looks normal, and the QQ plot tails start to pull off the line at high and low values, possibly indicating outliers are causing us some trouble, but not enough to violate the assumption.

13.5.5 What if my assumptions aren't respected?

The typical fixes for violated assumptions are data transformations, and the removal of outliers. In our case, we pass all assumptions except for homoscedasticity. We've already transformed our data to meet the assumption of normality, so further transformation is likely off the table, though we could potentially try different transformations. We could also try outlier removal - our model diagnostics using `plot.lm()` identify three outliers - 19, 20, and 28. Feel free to play around with removing outliers if you want, although in general it is good practice to only remove outliers where absolutely necessary, as they may contain important biological or other information.

Keep in mind that data transformations and removing outliers both represent trade-offs. Removing outliers may help meet your model assumptions, but you may also be removing data that reflects reality from your model. In that case, is it really helping you to remove outliers? Similarly, transforming your data may help you meet your assumptions, but in a case like this, transforming our data further or in a different way could end up violating other assumptions. Sometimes the best way to deal with violated assumptions is simply to state that they are violated and think about what that means for the interpretation of your model. Play around with all these different ideas, and come up with what you think is best. At the end of the day, a lot of statistical choices are judgement calls, with no perfect right answer.

13.6 Model Selection

In assignment 1b, we created 1 model with 6 variables, then tested if we could get a similarly effective mode using fewer variables - i.e. a more **efficient** model. Let's do the same thing here:

```r
# Stepwise model selection - forward
m8 = step(lm(CNS ~1, data = data_l),
          scope=(CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE),
          direction='forward')
```

```
Start:  AIC=47.92
CNS ~ 1

         Df Sum of Sq     RSS     AIC
+ MUSCLE  1    122.51   4.086 -83.984
+ FFWT    1    121.96   4.634 -79.074
+ HEART   1    121.82   4.780 -77.869
+ Mass    1    121.65   4.948 -76.519
+ BONE    1    121.03   5.567 -71.921
+ Fat     1    112.27  14.327 -35.056
<none>                126.595  47.920

Step:  AIC=-83.98
CNS ~ MUSCLE

        Df Sum of Sq    RSS     AIC
+ HEART  1  0.233798 3.8522 -84.282
+ Mass   1  0.212536 3.8735 -84.067
<none>               4.0860 -83.984
+ Fat    1  0.152823 3.9332 -83.470
+ FFWT   1  0.139313 3.9467 -83.337
+ BONE   1  0.021542 4.0645 -82.190

Step:  AIC=-84.28
CNS ~ MUSCLE + HEART

       Df Sum of Sq    RSS     AIC
+ Mass  1   0.35155 3.5007 -86.014
+ Fat   1   0.29737 3.5549 -85.415
+ FFWT  1   0.23682 3.6154 -84.756
<none>              3.8522 -84.282
+ BONE  1   0.10287 3.7494 -83.337

Step:  AIC=-86.01
CNS ~ MUSCLE + HEART + Mass

       Df Sum of Sq    RSS     AIC
```

```
<none>              3.5007 -86.014
+ Fat   1  0.040574 3.4601 -84.468
+ FFWT  1  0.018388 3.4823 -84.219
+ BONE  1  0.000458 3.5002 -84.019
```

```r
summary(m8)
```

```
Call:
lm(formula = CNS ~ MUSCLE + HEART + Mass, data = data_l)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.74267 -0.11882 -0.00818  0.10790  0.84702 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)   
(Intercept)  -0.1840     0.8704  -0.211  0.83383   
MUSCLE        1.2436     0.4254   2.923  0.00603 **
HEART         0.3855     0.1997   1.931  0.06166 . 
Mass         -0.8197     0.4372  -1.875  0.06919 . 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3163 on 35 degrees of freedom
Multiple R-squared:  0.9723,	Adjusted R-squared:   0.97 
F-statistic: 410.2 on 3 and 35 DF,  p-value: < 2.2e-16
```

Running forward model selection cuts the model down to 3 variables. As with 1b, we can also do backward:

```r
# Stepwise model selection - backwards
m9 = step(m7, direction = 'backward')
```

```
Start:  AIC=-80.52
CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE

         Df Sum of Sq    RSS     AIC
- FFWT    1   0.00001 3.4552 -82.523
- BONE    1   0.00422 3.4594 -82.476
- Mass    1   0.00517 3.4604 -82.465
```

```
- Fat     1   0.02092 3.4761 -82.288
<none>                3.4552 -80.523
- MUSCLE  1   0.30975 3.7650 -79.175
- HEART   1   0.39468 3.8499 -78.305

Step:  AIC=-82.52
CNS ~ Mass + Fat + HEART + MUSCLE + BONE

         Df Sum of Sq    RSS     AIC
- BONE    1   0.00487 3.4601 -84.468
- Fat     1   0.04499 3.5002 -84.019
- Mass    1   0.05110 3.5063 -83.951
<none>                3.4552 -82.523
- MUSCLE  1   0.38148 3.8367 -80.439
- HEART   1   0.39621 3.8514 -80.289

Step:  AIC=-84.47
CNS ~ Mass + Fat + HEART + MUSCLE

         Df Sum of Sq    RSS     AIC
- Fat     1   0.04057 3.5007 -86.014
- Mass    1   0.09476 3.5549 -85.415
<none>                3.4601 -84.468
- HEART   1   0.40303 3.8631 -82.171
- MUSCLE  1   0.41063 3.8707 -82.095

Step:  AIC=-86.01
CNS ~ Mass + HEART + MUSCLE

         Df Sum of Sq    RSS     AIC
<none>                3.5007 -86.014
- Mass    1   0.35155 3.8522 -84.282
- HEART   1   0.37281 3.8735 -84.067
- MUSCLE  1   0.85479 4.3555 -79.493
```

```r
summary(m9)
```

```
Call:
lm(formula = CNS ~ Mass + HEART + MUSCLE, data = data_l)

Residuals:
```

```
     Min       1Q   Median       3Q      Max 
-0.74267 -0.11882 -0.00818  0.10790  0.84702 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -0.1840     0.8704  -0.211  0.83383    
Mass         -0.8197     0.4372  -1.875  0.06919 .  
HEART         0.3855     0.1997   1.931  0.06166 .  
MUSCLE        1.2436     0.4254   2.923  0.00603 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3163 on 35 degrees of freedom
Multiple R-squared:  0.9723,	Adjusted R-squared:   0.97 
F-statistic: 410.2 on 3 and 35 DF,  p-value: < 2.2e-16
```

And both:

```r
# Stepwise model selection - both
m10 = step(m7, direction = 'both')
```

```
Start:  AIC=-80.52
CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE

         Df Sum of Sq    RSS     AIC
- FFWT    1   0.00001 3.4552 -82.523
- BONE    1   0.00422 3.4594 -82.476
- Mass    1   0.00517 3.4604 -82.465
- Fat     1   0.02092 3.4761 -82.288
<none>                3.4552 -80.523
- MUSCLE  1   0.30975 3.7650 -79.175
- HEART   1   0.39468 3.8499 -78.305

Step:  AIC=-82.52
CNS ~ Mass + Fat + HEART + MUSCLE + BONE

         Df Sum of Sq    RSS     AIC
- BONE    1   0.00487 3.4601 -84.468
- Fat     1   0.04499 3.5002 -84.019
- Mass    1   0.05110 3.5063 -83.951
<none>                3.4552 -82.523
+ FFWT    1   0.00001 3.4552 -80.523
```

```
- MUSCLE  1   0.38148 3.8367 -80.439
- HEART   1   0.39621 3.8514 -80.289

Step:  AIC=-84.47
CNS ~ Mass + Fat + HEART + MUSCLE

         Df Sum of Sq    RSS     AIC
- Fat     1   0.04057 3.5007 -86.014
- Mass    1   0.09476 3.5549 -85.415
<none>                3.4601 -84.468
+ BONE    1   0.00487 3.4552 -82.523
+ FFWT    1   0.00067 3.4594 -82.476
- HEART   1   0.40303 3.8631 -82.171
- MUSCLE  1   0.41063 3.8707 -82.095

Step:  AIC=-86.01
CNS ~ Mass + HEART + MUSCLE

         Df Sum of Sq    RSS     AIC
<none>                3.5007 -86.014
+ Fat     1   0.04057 3.4601 -84.468
- Mass    1   0.35155 3.8522 -84.282
+ FFWT    1   0.01839 3.4823 -84.219
- HEART   1   0.37281 3.8735 -84.067
+ BONE    1   0.00046 3.5002 -84.019
- MUSCLE  1   0.85479 4.3555 -79.493
```

```r
summary(m10)
```

```
Call:
lm(formula = CNS ~ Mass + HEART + MUSCLE, data = data_l)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.74267 -0.11882 -0.00818  0.10790  0.84702 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept)  -0.1840     0.8704  -0.211  0.83383  
Mass         -0.8197     0.4372  -1.875  0.06919 .
HEART         0.3855     0.1997   1.931  0.06166 .
```

```
MUSCLE        1.2436     0.4254   2.923  0.00603 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3163 on 35 degrees of freedom
Multiple R-squared:  0.9723,    Adjusted R-squared:   0.97
F-statistic: 410.2 on 3 and 35 DF,  p-value: < 2.2e-16
```

If you want to get fancy, we can even look at every possible model

```r
library(MuMIn)

# Set global options to avoid error
options('na.action' = na.fail)

# Run dredge to get full selection table
dredge(m7, rank = 'AIC')
```

```
Fixed term is "(Intercept)"


Global model call: lm(formula = CNS ~ Mass + Fat + FFWT + HEART + MUSCLE + BONE,
    data = data_l)
---
Model selection table
   (Intrc)     BONE       Fat     FFWT  HEART    Mass  MUSCL df  logLik   AIC
57 -0.1840                            0.3855 -0.8197 1.2440  5  -8.332  26.7
43 -1.1710          -0.124700         0.3924         0.5758  5  -8.631  27.3
45 -0.2475                    -0.85860 0.3612         1.2890  5  -8.961  27.9
59 -0.4891          -0.061790         0.4060 -0.5708 1.0530  6  -8.104  28.2
47 -0.5634          -0.096360 -0.57650 0.4140         1.0840  6  -8.124  28.2
41 -1.1430                            0.2965         0.5107  4 -10.198  28.4
61 -0.3049                     0.58870 0.3869 -1.2410 1.0870  6  -8.229  28.5
49 -1.7220                                   -0.6191 1.3760  4 -10.305  28.6
58 -0.2132 -0.01926                   0.3888 -0.8022 1.2420  6  -8.329  28.7
33 -2.2060                                           0.7649  3 -11.347  28.7
44 -1.2470 -0.17370 -0.114900         0.4320         0.6975  6  -8.363  28.7
35 -2.4620          -0.085950                        0.8664  4 -10.603  29.2
42 -1.2550 -0.24820                   0.3639         0.6919  5  -9.670  29.3
37 -1.7080                    -0.64360               1.3900  4 -10.670  29.3
46 -0.3146 -0.03584           -0.80970 0.3672         1.2710  6  -8.953  29.9
60 -0.6121 -0.06454 -0.066880         0.4188 -0.4915 1.0320  7  -8.077  30.2
```

63	-0.4991		-0.070960	-0.17140	0.4086	-0.4111	1.0710	7	-8.101	30.2
48	-0.6664	-0.05386	-0.097010	-0.50110	0.4234		1.0550	7	-8.106	30.2
15	-1.3260		-0.141700	0.53690	0.4770			5	-10.136	30.3
62	-0.4490	-0.07620		0.72770	0.4004	-1.2710	1.0430	7	-8.195	30.4
29	-0.9446			2.32300	0.4383	-1.8930		5	-10.222	30.4
53	-1.8390			0.54460		-1.0080	1.2310	5	-10.226	30.5
50	-1.4810	0.10920				-0.7284	1.3790	5	-10.226	30.5
34	-2.3590	-0.10740					0.8684	4	-11.244	30.5
51	-1.9150		-0.030810			-0.4897	1.2850	5	-10.253	30.5
27	-1.3680		-0.174800		0.4968	0.5506		5	-10.291	30.6
39	-2.0530		-0.062690	-0.43960			1.2660	5	-10.335	30.7
36	-2.5030	-0.03440	-0.083230				0.8963	5	-10.593	31.2
38	-1.4490	0.10160		-0.79230			1.4370	5	-10.612	31.2
16	-1.6470	-0.20880	-0.139600	0.71490	0.5072			6	-9.877	31.8
30	-1.3160	-0.23860		2.54100	0.4741	-1.9060		6	-9.885	31.8
13	-1.0500			0.42240	0.4110			4	-11.885	31.8
31	-1.1730		-0.087570	1.35400	0.4642	-0.8572		6	-10.045	32.1
64	-0.6114	-0.06339	-0.068180	-0.02606	0.4189	-0.4687	1.0350	8	-8.077	32.2
28	-1.6410	-0.16770	-0.182100		0.5254	0.6983		6	-10.119	32.2
54	-1.6320	0.07991		0.40060		-0.9854	1.2720	6	-10.189	32.4
52	-1.6650	0.09609	-0.024700			-0.6115	1.3050	6	-10.194	32.4
55	-1.8600		-0.005303	0.48770		-0.9454	1.2310	6	-10.225	32.5
40	-1.7890	0.10350	-0.062910	-0.59040			1.3130	6	-10.273	32.5
21	-2.8120			2.53800		-1.7250		4	-12.557	33.1
25	-0.7492				0.4796	0.3550		4	-12.592	33.2
14	-1.4090	-0.23040		0.62070	0.4454			5	-11.597	33.2
32	-1.4890	-0.22300	-0.075840	1.68700	0.4941	-1.0080		7	-9.751	33.5
5	-2.8020			0.78490				3	-13.802	33.6
7	-3.2380		-0.110900	0.92010				4	-12.820	33.6
56	-1.6680	0.08319	-0.011140	0.27490		-0.8521	1.2720	7	-10.186	34.4
19	-3.4510		-0.166800			0.9721		4	-13.253	34.5
9	0.9951				0.8820			3	-14.404	34.8
12	0.3485	0.33210	-0.111800		0.6667			5	-12.415	34.8
10	0.3284	0.25560			0.5985			4	-13.429	34.9
22	-3.0100	-0.08987		2.62700		-1.7240		5	-12.513	35.0
23	-2.8680		-0.014310	2.38200		-1.5540		5	-12.553	35.1
26	-0.8465	-0.06583			0.4905	0.4097		5	-12.568	35.1
6	-3.0010	-0.09063		0.87490				4	-13.760	35.5
8	-3.3620	-0.05836	-0.109700	0.97660				5	-12.802	35.6
11	1.1350		-0.071490		0.9798			4	-13.981	36.0
17	-2.7910					0.7710		3	-15.079	36.2
20	-3.4240	0.01165	-0.166400			0.9601		5	-13.252	36.5
24	-3.0350	-0.08761	-0.007846	2.53900		-1.6310		6	-12.511	37.0

18	-2.5850	0.09433		0.6790	4	-15.033	38.1
2	-1.0500	0.78560			3	-17.378	40.8
4	-1.1140	0.84970	-0.052420		4	-17.190	42.4
3	-0.1571		0.599000		3	-35.811	77.6
1	1.3230				2	-78.299	160.6

	delta	weight
57	0.00	0.104
43	0.60	0.077
45	1.26	0.056
59	1.55	0.048
47	1.58	0.047
41	1.73	0.044
61	1.79	0.042
49	1.95	0.039
58	1.99	0.038
33	2.03	0.038
44	2.06	0.037
35	2.54	0.029
42	2.68	0.027
37	2.68	0.027
46	3.24	0.021
60	3.49	0.018
63	3.54	0.018
48	3.55	0.018
15	3.61	0.017
62	3.73	0.016
29	3.78	0.016
53	3.79	0.016
50	3.79	0.016
34	3.82	0.015
51	3.84	0.015
27	3.92	0.015
39	4.01	0.014
36	4.52	0.011
38	4.56	0.011
16	5.09	0.008
30	5.11	0.008
13	5.11	0.008
31	5.43	0.007
64	5.49	0.007
28	5.57	0.006
54	5.71	0.006
52	5.72	0.006

```
55   5.79  0.006
40   5.88  0.005
21   6.45  0.004
25   6.52  0.004
14   6.53  0.004
32   6.84  0.003
5    6.94  0.003
7    6.98  0.003
56   7.71  0.002
19   7.84  0.002
9    8.14  0.002
12   8.17  0.002
10   8.19  0.002
22   8.36  0.002
23   8.44  0.002
26   8.47  0.002
6    8.86  0.001
8    8.94  0.001
11   9.30  0.001
17   9.49  0.001
20   9.84  0.001
24  10.36  0.001
18  11.40  0.000
2   14.09  0.000
4   15.72  0.000
3   50.96  0.000
1  133.93  0.000
Models ranked by AIC(x)
```

Here, we're ranking models by AIC. AIC balances fit with model complexity. Lower values of AIC are considered better. Generally, 2 is used as a rule of thumb for delta AIC: if delta AIC is >2, the model with the higher AIC value has little support. If delta AIC is <2, there is at least some support for the model with the higher AIC.

13.7 Tips for your Assignment:

Some things you may want to think about for your assignment:

1. What role is collinearity playing in your assignment? Is it something you should be concerned about? Why or why not?

2. What does it mean if your assumptions are violated? How would you fix it? Is it worth fixing it? Why or why not?
3. How would you interpret your statistical results biologically? You don't have to be right, but don't be vague, and don't contradict your results.

14 Assignment 1e: Bayesian Data Analysis

Assignment 1e is an introduction to Bayesian data analysis, using Bayesian generalized linear models.

For this tutorial, we'll be using `cuse.csv` .

14.1 Looking at the Data

```
# Load in data
data = read.csv('cuse.csv')

# Look at the data structure
head(data)
```

```
  X   age education wantsMore notUsing using
1 1   <25       low       yes       53     6
2 2   <25       low        no       10     4
3 3   <25      high       yes      212    52
4 4   <25      high        no       50    10
5 5 25-29       low       yes       60    14
6 6 25-29       low        no       19    10
```

```
dim(data)
```

```
[1] 16  6
```

Our data contains 16 observations of 5 variables - a two-column binomial matrix (notUsing and using) of the number of women using and not using birth control within 16 groups, and three categorical predictors - age, expressed as categories, education, and whether they want more children. The first column is a duplicate of our row names. We can get rid of that:

```r
# Remove column 1
data = data[,-1]
head(data)
```

```
    age education wantsMore notUsing using
1   <25       low       yes       53     6
2   <25       low        no       10     4
3   <25      high       yes      212    52
4   <25      high        no       50    10
5 25-29       low       yes       60    14
6 25-29       low        no       19    10
```

14.2 Binomial GLM

Binomial generalized linear models (logistic regression with a binary response variable and a logit link) are used to calculate the probability of a binomial response - in this case, whether someone is using or not using birth control. Binomial GLM responses can be fed in either as a true/false set, or as a two-column matrix of successes and failures. According to `?family`, we need to feed in the data with successes (whatever that means in each context) first and failures second. Let's create the matrix:

```r
# create response matrix
resp = cbind(data$using, data$notUsing)
head(resp)
```

```
     [,1] [,2]
[1,]    6   53
[2,]    4   10
[3,]   52  212
[4,]   10   50
[5,]   14   60
[6,]   10   19
```

In this case, all of our variables are categorical, and they are currently stored as characters:

```r
# Check predictor classes
class(data$age)
```

```
[1] "character"
```

```r
class(data$education)
```

```
[1] "character"
```

```r
class(data$wantsMore)
```

```
[1] "character"
```

These should function fine as categorical variables. Let's make our GLM:

```r
# Run GLM
m1 = glm(resp ~ age + education + wantsMore, family = 'binomial', data = data)
summary(m1) # Summary
```

```
Call:
glm(formula = resp ~ age + education + wantsMore, family = "binomial",
    data = data)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -0.8082     0.1590  -5.083 3.71e-07 ***
age25-29       0.3894     0.1759   2.214  0.02681 *
age30-39       0.9086     0.1646   5.519 3.40e-08 ***
age40-49       1.1892     0.2144   5.546 2.92e-08 ***
educationlow  -0.3250     0.1240  -2.620  0.00879 **
wantsMoreyes  -0.8330     0.1175  -7.091 1.33e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 165.772  on 15  degrees of freedom
Residual deviance:  29.917  on 10  degrees of freedom
AIC: 113.43

Number of Fisher Scoring iterations: 4
```

In our summary we see we have 5 predictor categories with parameter estimates: different age bins (values representing the difference from the age <25 bin), low education (representing the

difference from high), and wanting more kids (representing the difference from not wanting more kids). We can also see the values of our model coefficients, their standard errors, and the model AIC.

14.3 Making it Bayesian

The default GLM function is frequentist (that's why we have p-values). Now lets try a Bayesian approach:

```
# Stan
library(rstanarm)
```

```
Loading required package: Rcpp

This is rstanarm version 2.32.1

- See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

- Default priors may change, so it's safest to specify priors, even if equivalent to the defa

- For execution on a local, multicore CPU with excess RAM we recommend calling

  options(mc.cores = parallel::detectCores())
```

```
library(bayesplot)
```

```
This is bayesplot version 1.14.0

- Online documentation and vignettes at mc-stan.org/bayesplot

- bayesplot theme set to bayesplot::theme_default()

   * Does _not_ affect other ggplot2 plots

   * See ?bayesplot_theme_set for details on theme setting
```

```r
library(shinystan)
```

```
Loading required package: shiny


This is shinystan version 2.6.0
```

```r
library(ggplot2)

# Run glm
m2 = stan_glm(resp ~ age + education + wantsMore, family = 'binomial', data = data)
```

```
SAMPLING FOR MODEL 'binomial' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 4.7e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.47 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.119 seconds (Warm-up)
Chain 1:                0.114 seconds (Sampling)
Chain 1:                0.233 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'binomial' NOW (CHAIN 2).
Chain 2:
```

```
Chain 2: Gradient evaluation took 1.3e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.109 seconds (Warm-up)
Chain 2:                0.119 seconds (Sampling)
Chain 2:                0.228 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'binomial' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 1.2e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
```

```
Chain 3:  Elapsed Time: 0.119 seconds (Warm-up)
Chain 3:                0.127 seconds (Sampling)
Chain 3:                0.246 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'binomial' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1.3e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.109 seconds (Warm-up)
Chain 4:                0.101 seconds (Sampling)
Chain 4:                0.21 seconds (Total)
Chain 4:
```

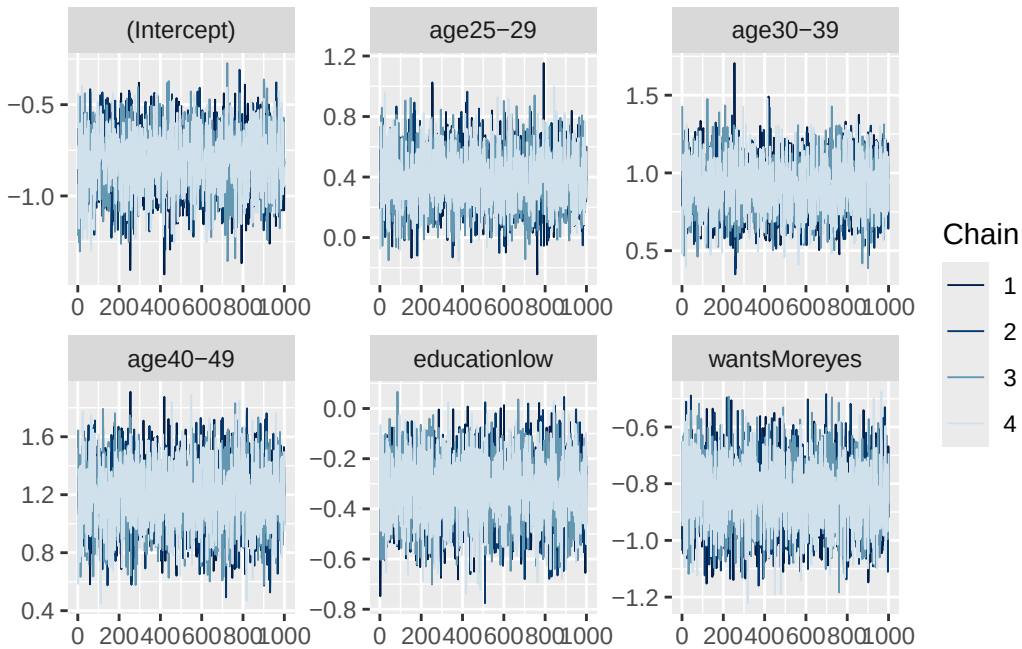
```r
summary(m2) # Summary
```

```
Model Info:
 function:     stan_glm
 family:       binomial [logit]
 formula:      resp ~ age + education + wantsMore
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 16
 predictors:   6
```

```
Estimates:
               mean   sd   10%   50%   90%
(Intercept)  -0.8    0.2 -1.0  -0.8  -0.6
age25-29      0.4    0.2  0.2   0.4   0.6
age30-39      0.9    0.2  0.7   0.9   1.1
age40-49      1.2    0.2  0.9   1.2   1.5
educationlow -0.3    0.1 -0.5  -0.3  -0.2
wantsMoreyes -0.8    0.1 -1.0  -0.8  -0.7

Fit Diagnostics:
           mean   sd   10%   50%   90%
mean_PPD 31.7    1.6 29.8  31.7  33.8

The mean_ppd is the sample average posterior predictive distribution of the outcome variable

MCMC diagnostics
              mcse Rhat n_eff
(Intercept)   0.0  1.0  2292
age25-29      0.0  1.0  2450
age30-39      0.0  1.0  1905
age40-49      0.0  1.0  2441
educationlow  0.0  1.0  2984
wantsMoreyes  0.0  1.0  3592
mean_PPD      0.0  1.0  3976
log-posterior 0.0  1.0  1871

For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective
```

The `stan_glm` function automatically feeds our model into Stan, which is a Hamiltonian Markov Chain Monte Carlo (MCMC) sampler. Running `summary` on our model gives us some model diagnostics - all our Rhat values are 1 and all our n_eff values are well into the thousands, both of which are a good sign. We can also do some visual checks and tests:

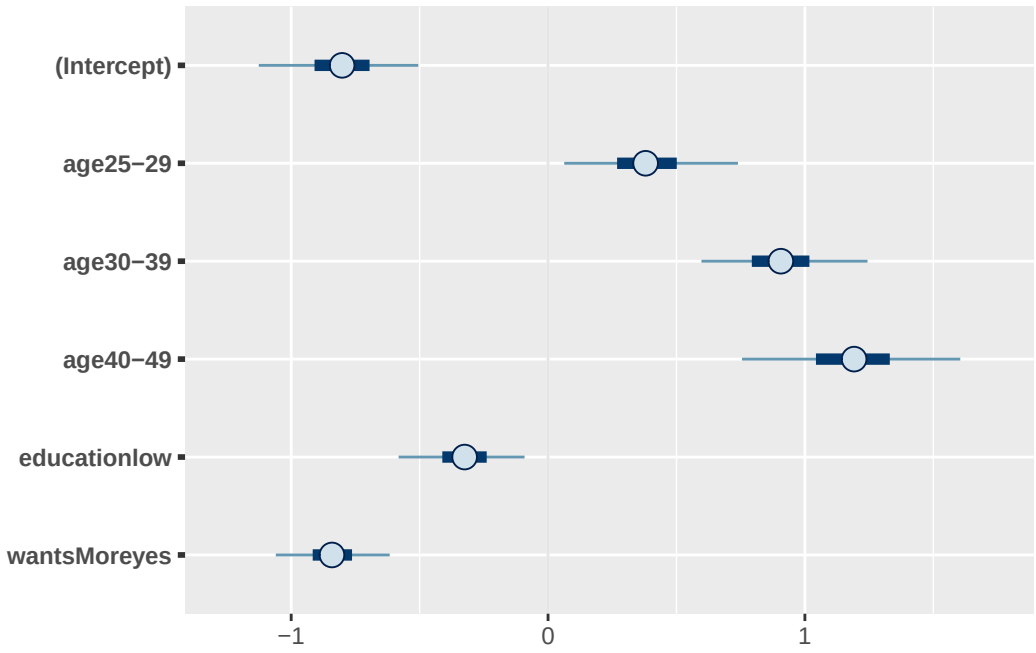
```r
# Trace plot
plot(m2, 'trace')
```



These are trace plots, which show us the parameter values selected for each iteration of the MCMC chain. We want these to look "fuzzy" - that indicates the sampler is exploring the full range of possible values. If these lines were flat, that would indicate the sampler got "stuck" and didn't sample the full posterior distributions. These look good.

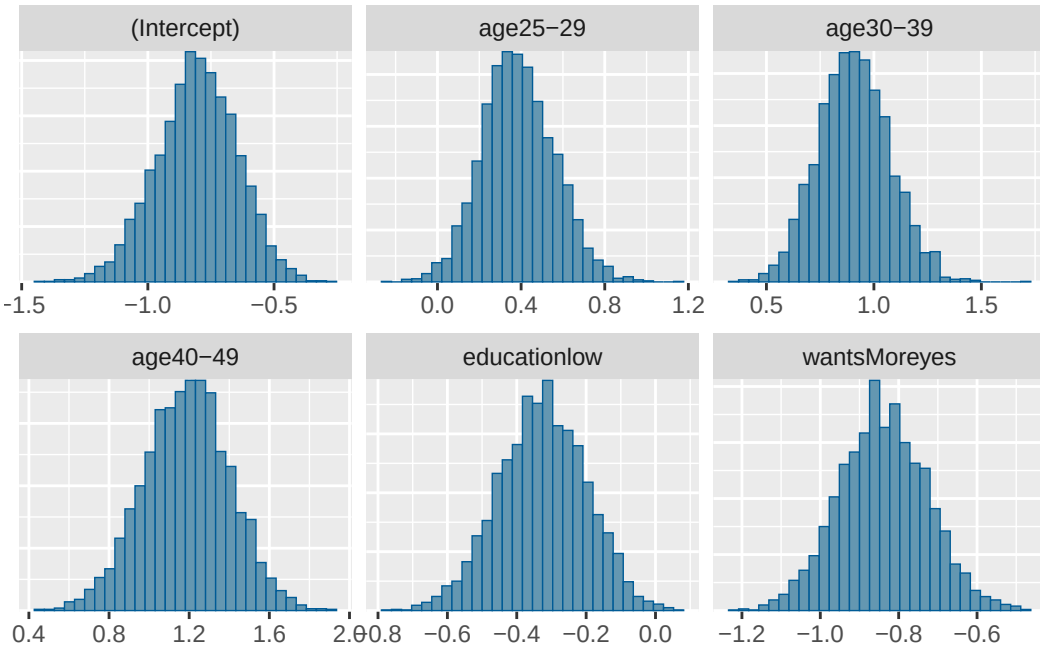
Lets look at our posteriors:

```
# Plot parameter values with uncertainties
plot(m2, prob_outer = 0.95)
```




```
# Plot posterior distributions
plot(m2, 'mcmc_hist')
```

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



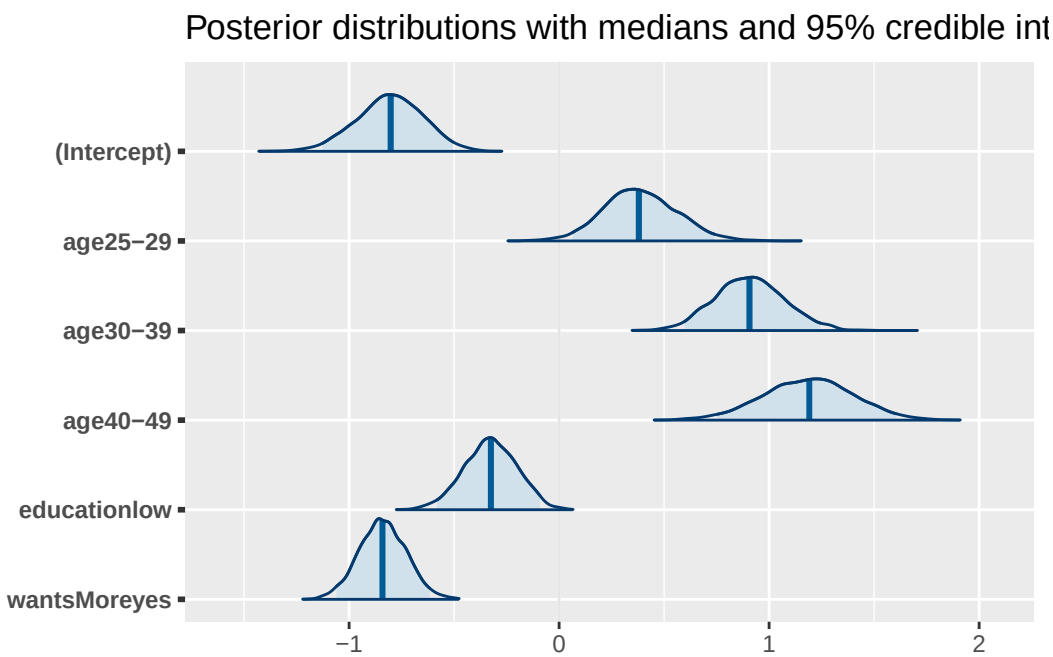
These plots both give us an idea of our parameter values and their posterior distributions. The former plot shows the median parameter estimates (circle), their 50% quantiles (dark blue box), and their 95% quantiles (thin blue line). The latter shows histograms of the posterior distributions of each of our parameters.

We can also pull out our coefficients and posteriors directly

```
# Model coefficients
m2$coefficients
```

```
 (Intercept)     age25-29     age30-39     age40-49 educationlow wantsMoreyes
  -0.8017060    0.3798471    0.9064734    1.1916628   -0.3247363   -0.8412193
```

```
# Model posteriors
posterior <- as.matrix(m2)

# Plot model posteriors (95% quantile)
plot_title <- ggtitle("Posterior distributions with medians and 95% credible intervals")

mcmc_areas(posterior, pars = names(m2$coefficients),
           prob = 0.95) + plot_title
```



How would you interpret these plots? Note the asymmetry in the histograms, in contrast to typical frequentist approaches

14.4 Adding Priors

Lets try adding some priors:

```
# Run glm with priors
m3 = stan_glm(resp ~ age + education + wantsMore, family = 'binomial', data = data,
              prior = normal(location = c(0.2, 1.5, 2, -1, -0.25), # Normal priors, means
                             scale = c(0.03, 0.03, 0.03, 0.03, 0.03))) # And standard deviati
```

```
SAMPLING FOR MODEL 'binomial' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 2e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.2 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
```

```
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.156 seconds (Warm-up)
Chain 1:                0.077 seconds (Sampling)
Chain 1:                0.233 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'binomial' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 1.2e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.122 seconds (Warm-up)
Chain 2:                0.071 seconds (Sampling)
Chain 2:                0.193 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'binomial' NOW (CHAIN 3).
```

```
Chain 3:
Chain 3: Gradient evaluation took 1.2e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.136 seconds (Warm-up)
Chain 3:                0.076 seconds (Sampling)
Chain 3:                0.212 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'binomial' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1.2e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
```

```
Chain 4:
Chain 4:  Elapsed Time: 0.134 seconds (Warm-up)
Chain 4:                0.074 seconds (Sampling)
Chain 4:                0.208 seconds (Total)
Chain 4:
```

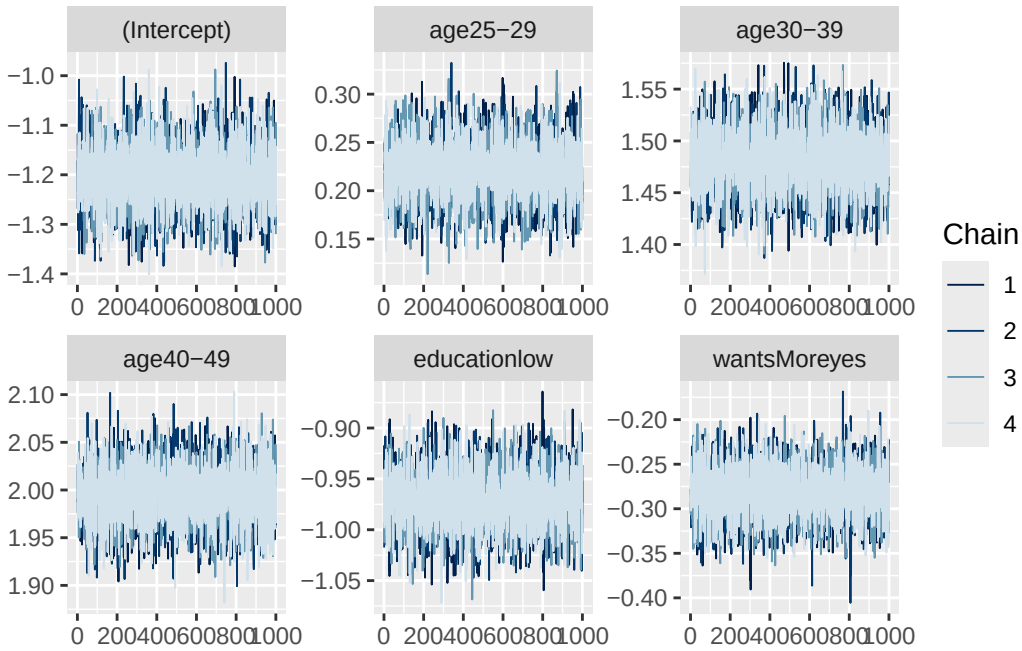
```r
summary(m3) # Summary
```

```
Model Info:
 function:     stan_glm
 family:       binomial [logit]
 formula:      resp ~ age + education + wantsMore
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 16
 predictors:   6

Estimates:
               mean   sd   10%   50%   90%
(Intercept)  -1.2    0.1 -1.3  -1.2  -1.1
age25-29      0.2    0.0  0.2   0.2   0.3
age30-39      1.5    0.0  1.4   1.5   1.5
age40-49      2.0    0.0  2.0   2.0   2.0
educationlow -1.0    0.0 -1.0  -1.0  -0.9
wantsMoreyes -0.3    0.0 -0.3  -0.3  -0.2

Fit Diagnostics:
           mean   sd   10%   50%   90%
mean_PPD 31.7    1.5 29.7  31.7  33.8

The mean_ppd is the sample average posterior predictive distribution of the outcome variable

MCMC diagnostics
              mcse Rhat n_eff
(Intercept)   0.0  1.0  6574
age25-29      0.0  1.0  5638
age30-39      0.0  1.0  6199
age40-49      0.0  1.0  6705
educationlow  0.0  1.0  6761
wantsMoreyes  0.0  1.0  7387
```

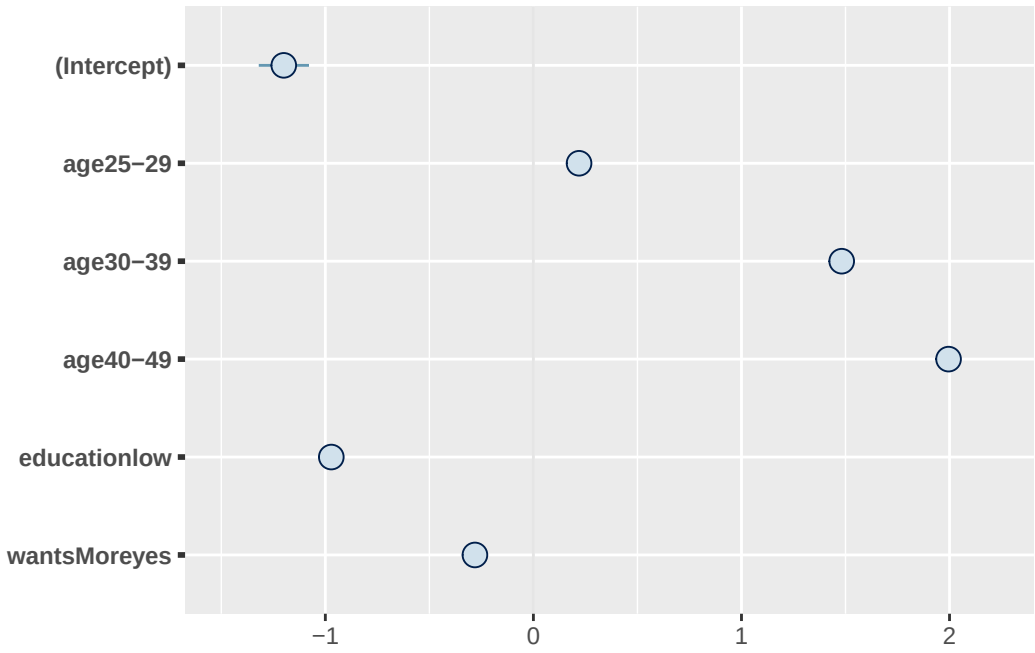
```
mean_PPD      0.0  1.0  5118
log-posterior 0.0  1.0  1973

For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective
```

Lets look at our plots again:

```r
# Trace plot
plot(m3, 'trace')
```

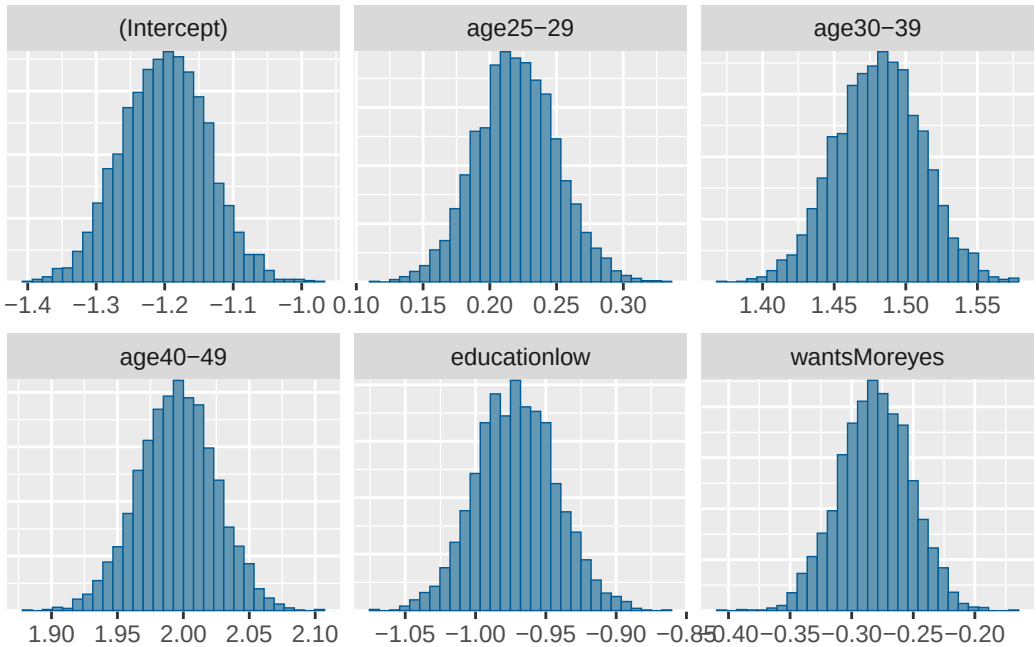



```r
# Plot parameter values with uncertainties
plot(m3, prob_outer = 0.95)
```




```
# Plot posterior distributions
plot(m3, 'mcmc_hist')
```

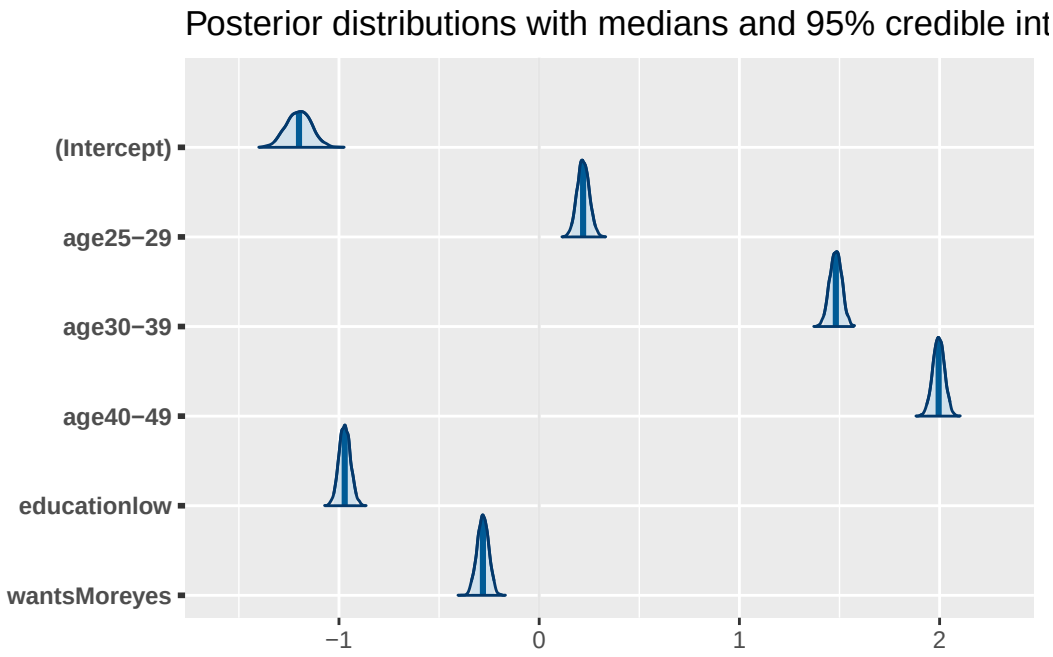
```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```




```r
# Model coefficients
m3$coefficients
```

```
 (Intercept)     age25-29     age30-39     age40-49 educationlow wantsMoreyes
  -1.1997315    0.2194363    1.4816879    1.9952694   -0.9713091   -0.2810344
```

```r
# Model posteriors
posterior <- as.matrix(m3)

# Plot model posteriors (95% quantile)
plot_title <- ggtitle("Posterior distributions with medians and 95% credible intervals")

mcmc_areas(posterior, pars = names(m3$coefficients),
           prob = 0.95) + plot_title
```



What has changed versus the model without priors?

You can also look at all of your Stan model results using `shinystan` by running `launch_shinystan(model)`. Try it out on your end (it doesn't work in markdown)

14.5 Tips for your Assignment:

Some things you may want to think about for your assignment:

1. How do the results of these three models differ? Why do they or don't they?
2. Do you interpret certain models as being more or less correct? Why or why not?
3. How would you interpret your statistical results biologically? You don't have to be right, but don't be vague, and don't contradict your results.