Workspace

Table of contents

Welcome

This Quarto Book[1] is a workspace for the notes and projects of the programming/scripting languages that I am learning.

[1] Quarto Books Documentation

Part I

Notebooks

This section contains notes for the programming/scripting languages that I am learning.

1 R

1.1 General Notes

- When writing R code, create a project instead of a file which will enable saving the workspace settings
- An R package usually includes:
    a. reusable functions
    b. documentation describing how to use the function
    c. sample data
- Before running a project, clear the objects in its workspace environment to avoid mixing up objects created in other files. This can be done either:
    a. pragmatically as shown below OR
    b. Environment window -> Broom icon

1.2 Clear Workspace

Always start by clearing the workspace. This ensure objects created in other files are not used used here.

```
rm(list = ls())
```

1.3 Packages

1.3.1 List

List all the packages that will be used in this script.

```
packages <- c("here")
```

1.3.2 Install Missing

Any missing package will be installed automatically. This ensure smoother execution when run by others.

```
# Do NOT modify
install.packages(setdiff(packages, rownames(installed.packages())))
```

1.3.3 Load

Load all packages

```
# Do NOT modify
lapply(packages, require, character.only = TRUE)
```

```
[[1]]
[1] TRUE
```

1.4 Data Types

- character
- numeric
- logical
- raw
- imaginary numbers

To know the datatype of an object, run the command:

```
class(x) # give the data type of x
```

1.4.1 Mixing Data Types

- character + numeric –> character
- numeric + logical –> numeric
- numeric + character + logical –> character

1.5 Data Structures

- **vector**: hold single type of data

> **Creating Vector**
>
> The `c()` function (**c**ombine multiple elements) can be used to create vectors in R.
>
> `x <- c(1, 2, 3, 4)`

- **matrix**: 2D vector
- **array**: nD vector
- **list**: generic vector, can hold mixed type of data, eg, one element can a character, another a list of integers, and the third could be a logical
- **data frame**: table where columns represent vectors
- **tibbles**: data frames, but slightly tweaked to work better with `tidyverse`–when printing tibbles, only the first few columns that fit into the screen will be shown

> **Printing tibble All Columns**
>
> To force the `print()` function to print all the columns of a `tibble`, set the `width` parameter to `Inf` as follows:
>
> ```
> print(ds, width = Inf)
> ```

To construct small tibble by hand, use the `tribble` function as follows:

```
df <- tribble(
  ~var1, ~var2, ~var3
  "A", 1, TRUE,
  "B", 2, FALSE,
  "C", 3, TRUE
)
```

- **factor**

To know the data structure and length of the object, run the command:

```
str(x) # give the structure type of x
length(x) # length of structure
```

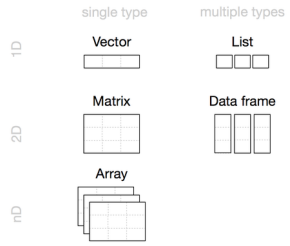


Figure 1.1: Common data structures in R (Source: Grolemund, 2014)

1.6 Basics Operations

Assignment

```
x <- 3 # assign 3 to x (x gets 3)
(x <- 3) # assign 3 to x (x gets 3) & print the result to console
```

```
[1] 3
```

Getting Help

```
args(round) # print the argument list of function
?round # show documentation of function in Help window
```

Dealing with Structure

```
# concatenate set of values to create vector
weight_g <- c(50, 60, 3, 9)
animals <- c("dog", "bat", "cat")

# utilizing logical values to pull specific values
weight_g[weight_g < 10 & weight_g > 60 | weight_g == 50]
```

```
[1] 50
```

```
# pull dog & cat records
animals[animals %in% c("dog", "cat")]
```

```
[1] "dog" "cat"
```

```
animals[animals == "dog" | animals == "cat"]
```

```
[1] "dog" "cat"
```

Statistics

```
# signaling missing data using NA
heights <- c(2, 3, NA, 4)

# get mean while ignoring missing data
mean(heights, na.rm = TRUE)
```

```
[1] 3
```

```
# how to use mean
# ?mean
```

1.7 Exploratory Operations

The `here` package makes it easy to point to files starting from the project main directory.

```
library(here)
```

Loading file from repository and saving it locally on disk. It is always a good idea to structure the workspace–for more information, see Best Practices for Scientific Computing paper.

```
download.file(
  url = "https://ndownloader.figshare.com/files/2292169",
  destfile = here("data", "portal_data_joined.csv")
)
```

Load file to R as data frame

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

Inspecting data frame

```r
class(surveys) # data type
```

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

```r
str(surveys) # structure
```

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

```r
dim(surveys) # dimensions
```

```
[1] 34786    13
```

```r
nrow(surveys)
```

```
[1] 34786
```

```r
ncol(surveys)
```

```
[1] 13
```

```r
summary(surveys)
```

```
   record_id         month              day             year         plot_id
 Min.   :    1   Min.   : 1.000   Min.   : 1.0   Min.   :1977   Min.   : 1.00
 1st Qu.: 8964   1st Qu.: 4.000   1st Qu.: 9.0   1st Qu.:1984   1st Qu.: 5.00
 Median :17762   Median : 6.000   Median :16.0   Median :1990   Median :11.00
 Mean   :17804   Mean   : 6.474   Mean   :16.1   Mean   :1990   Mean   :11.34
 3rd Qu.:26655   3rd Qu.:10.000   3rd Qu.:23.0   3rd Qu.:1997   3rd Qu.:17.00
 Max.   :35548   Max.   :12.000   Max.   :31.0   Max.   :2002   Max.   :24.00

  species_id            sex            hindfoot_length     weight
 Length:34786       Length:34786       Min.   : 2.00   Min.   :  4.00
 Class :character   Class :character   1st Qu.:21.00   1st Qu.: 20.00
 Mode  :character   Mode  :character   Median :32.00   Median : 37.00
                                       Mean   :29.29   Mean   : 42.67
                                       3rd Qu.:36.00   3rd Qu.: 48.00
                                       Max.   :70.00   Max.   :280.00
                                       NA's   :3348    NA's   :2503
    genus              species              taxa            plot_type
 Length:34786       Length:34786       Length:34786       Length:34786
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

Show first/last few objects/records/rows

```
head(surveys)
```

```
  record_id month day year plot_id species_id sex hindfoot_length weight
1         1     7  16 1977       2         NL   M              32     NA
2        72     8  19 1977       2         NL   M              31     NA
3       224     9  13 1977       2         NL                  NA     NA
4       266    10  16 1977       2         NL                  NA     NA
5       349    11  12 1977       2         NL                  NA     NA
6       363    11  12 1977       2         NL                  NA     NA
    genus  species   taxa plot_type
1 Neotoma albigula Rodent   Control
2 Neotoma albigula Rodent   Control
3 Neotoma albigula Rodent   Control
4 Neotoma albigula Rodent   Control
5 Neotoma albigula Rodent   Control
```

```
6 Neotoma albigula Rodent   Control
```

```
tail(surveys)
```

```
      record_id month day year plot_id species_id sex hindfoot_length weight
34781     26787     9  27 1997       7         PL   F               21     16
34782     26966    10  25 1997       7         PL   M               20     16
34783     27185    11  22 1997       7         PL   F               21     22
34784     27792     5   2 1998       7         PL   F               20      8
34785     28806    11  21 1998       7         PX                   NA     NA
34786     30986     7   1 2000       7         PX                   NA     NA
           genus  species   taxa        plot_type
34781  Peromyscus leucopus Rodent Rodent Exclosure
34782  Peromyscus leucopus Rodent Rodent Exclosure
34783  Peromyscus leucopus Rodent Rodent Exclosure
34784  Peromyscus leucopus Rodent Rodent Exclosure
34785 Chaetodipus      sp. Rodent Rodent Exclosure
34786 Chaetodipus      sp. Rodent Rodent Exclosure
```

1.7.1 Retrieve specific element/row/column

```
surveys[1, 1] # element[1,1]
```

```
[1] 1
```

```
surveys[1, ] # row 1
```

```
  record_id month day year plot_id species_id sex hindfoot_length weight
1         1     7  16 1977       2         NL   M               32     NA
    genus  species   taxa plot_type
1 Neotoma albigula Rodent   Control
```

```
head(surveys[, 1]) # column 1
```

```
[1]   1  72 224 266 349 363
```

```
head(surveys$sex) # column by name
```

```
[1] "M" "M" ""  ""  ""  ""
```

1.7.2 Dealing with factor (categorical) columns

R convert columns that contain characters to factors by default. Factors are treated as integer vectors. By default, R sorts levels in alphabetical order.

```
levels(surveys$sex)
```

```
NULL
```

```
nlevels(surveys$sex)
```

```
[1] 0
```

Reorder factors (to get better plots)

```
surveys$sex_ordered <- factor(surveys$sex, level = c("F", "M", ""))
str(surveys$sex_ordered)
```

```
 Factor w/ 3 levels "F","M","": 2 2 3 3 3 3 3 3 2 3 ...
```

```
levels(surveys$sex_ordered)
```

```
[1] "F" "M" ""
```

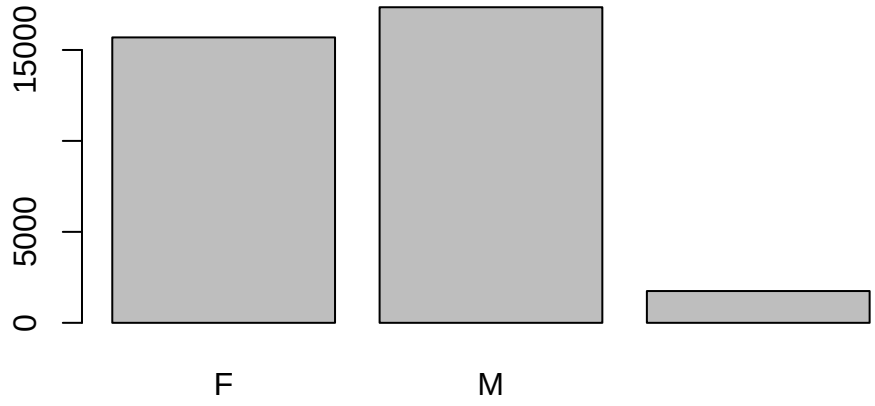
```
nlevels(surveys$sex_ordered)
```

```
[1] 3
```

1.7.3 Basic Plotting

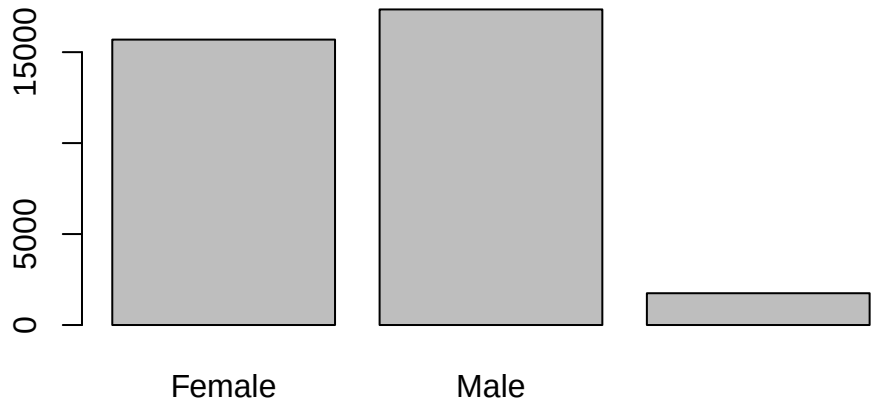
Histogram

```
# plot(surveys$sex)  # not possible
plot(surveys$sex_ordered)
```



Enhance the plot

```
levels(surveys$sex_ordered)[1] <- "Female"
levels(surveys$sex_ordered)[2] <- "Male"
plot(surveys$sex_ordered)
```



1.8 Data Manipulation

- `tdlyr`
  - makes manipulation of data easier
  - built to work with data frames directly
  - can directly work with data stored in an external database which give the advantage of only bringing what we need to the memory to work on without having to bring the whole database
- `tidyr`
  - allows to swiftly convert b/w different data formats for plotting & analysis in order to accommodate the different requirements by different functions
    - sometime we want one row per measurement
    - other times we want the data aggregated like when plotting

Before using `tdlyr` and `tidyr`:

- Install `tidyverse` package: umbrella-package that install several packages (tidyr, dplyr, ggplot2 tibble, magrittr, etc.)
- Load the package each session

Load packages

```
library("tidyverse")
```

Load & inspect data

```
# notice the '_' instead of '.' of basic R
surveys <- read_csv(here("data", "portal_data_joined.csv"))
```

```
Rows: 34786 Columns: 13
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (6): species_id, sex, genus, species, taxa, plot_type
dbl (7): record_id, month, day, year, plot_id, hindfoot_length, weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
str(surveys) # structure: tbl_df (tibble)
```

```
spc_tbl_ [34,786 x 13] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ record_id      : num [1:34786] 1 72 224 266 349 363 435 506 588 661 ...
 $ month          : num [1:34786] 7 8 9 10 11 11 12 1 2 3 ...
 $ day            : num [1:34786] 16 19 13 16 12 12 10 8 18 11 ...
 $ year           : num [1:34786] 1977 1977 1977 1977 1977 ...
 $ plot_id        : num [1:34786] 2 2 2 2 2 2 2 2 2 2 ...
 $ species_id     : chr [1:34786] "NL" "NL" "NL" "NL" ...
 $ sex            : chr [1:34786] "M" "M" NA NA ...
 $ hindfoot_length: num [1:34786] 32 31 NA NA NA NA NA NA NA NA ...
 $ weight         : num [1:34786] NA NA NA NA NA NA NA NA 218 NA ...
 $ genus          : chr [1:34786] "Neotoma" "Neotoma" "Neotoma" "Neotoma" ...
 $ species        : chr [1:34786] "albigula" "albigula" "albigula" "albigula" ...
 $ taxa           : chr [1:34786] "Rodent" "Rodent" "Rodent" "Rodent" ...
 $ plot_type      : chr [1:34786] "Control" "Control" "Control" "Control" ...
 - attr(*, "spec")=
  .. cols(
  ..   record_id = col_double(),
  ..   month = col_double(),
  ..   day = col_double(),
  ..   year = col_double(),
  ..   plot_id = col_double(),
```

```
  ..   species_id = col_character(),
  ..   sex = col_character(),
  ..   hindfoot_length = col_double(),
  ..   weight = col_double(),
  ..   genus = col_character(),
  ..   species = col_character(),
  ..   taxa = col_character(),
  ..   plot_type = col_character()
  .. )
 - attr(*, "problems")=<externalptr>
```

```
# view(surveys) # preview in the viewer window, avoid when rendering
```

1.8.1 Selection

Select certain columns

```
select(surveys, plot_id, species_id, weight)
```

```
# A tibble: 34,786 x 3
   plot_id species_id weight
     <dbl> <chr>       <dbl>
 1       2 NL             NA
 2       2 NL             NA
 3       2 NL             NA
 4       2 NL             NA
 5       2 NL             NA
 6       2 NL             NA
 7       2 NL             NA
 8       2 NL             NA
 9       2 NL            218
10       2 NL             NA
# i 34,776 more rows
```

Select all columns except ...

```
select(surveys, -sex)
```

```
# A tibble: 34,786 x 12
   record_id month   day  year plot_id species_id hindfoot_length weight genus
```

```
      <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>                <dbl>  <dbl> <chr>
 1        1     7    16  1977       2 NL                      32     NA Neotoma
 2       72     8    19  1977       2 NL                      31     NA Neotoma
 3      224     9    13  1977       2 NL                      NA     NA Neotoma
 4      266    10    16  1977       2 NL                      NA     NA Neotoma
 5      349    11    12  1977       2 NL                      NA     NA Neotoma
 6      363    11    12  1977       2 NL                      NA     NA Neotoma
 7      435    12    10  1977       2 NL                      NA     NA Neotoma
 8      506     1     8  1978       2 NL                      NA     NA Neotoma
 9      588     2    18  1978       2 NL                      NA    218 Neotoma
10      661     3    11  1978       2 NL                      NA     NA Neotoma
# i 34,776 more rows
# i 3 more variables: species <chr>, taxa <chr>, plot_type <chr>
```

Select rows based on criteria

```
filter(surveys, year == 1995)
```

```
# A tibble: 1,180 x 13
   record_id month   day  year plot_id species_id sex   hindfoot_length weight
       <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
 1     22314     6     7  1995       2 NL         M                  34     NA
 2     22728     9    23  1995       2 NL         F                  32    165
 3     22899    10    28  1995       2 NL         F                  32    171
 4     23032    12     2  1995       2 NL         F                  33     NA
 5     22003     1    11  1995       2 DM         M                  37     41
 6     22042     2     4  1995       2 DM         F                  36     45
 7     22044     2     4  1995       2 DM         M                  37     46
 8     22105     3     4  1995       2 DM         F                  37     49
 9     22109     3     4  1995       2 DM         M                  37     46
10     22168     4     1  1995       2 DM         M                  36     48
# i 1,170 more rows
# i 4 more variables: genus <chr>, species <chr>, taxa <chr>, plot_type <chr>
```

1.8.2 Piping

Sending the results of one function to another

```
# in multiple steps
survey_less5 <- filter(surveys, weight < 5)
survey_sml <- select(survey_less5, species_id, sex, weight)
```

```
# in one long step
survey_sml <- select(filter(surveys, weight < 5), species_id, sex, weight)

# using pipe %>% of magritter package.  Use Ctrl + Shift + M to add
survey_sml <- surveys %>%
  filter(weight < 5) %>%
  select(species_id, sex, weight)
```

1.8.3 Summary

Summary of groups (1+ columns)

one factor

```
surveys %>%
  group_by(sex) %>%
  summarise(mean_weight = mean(weight, na.rm = TRUE))
```

```
# A tibble: 3 x 2
  sex   mean_weight
  <chr>       <dbl>
1 F            42.2
2 M            43.0
3 <NA>         64.7
```

two factors

```
surveys %>%
  group_by(sex, species) %>%
  summarise(mean_weight = mean(weight, na.rm = TRUE))
```

```
`summarise()` has grouped output by 'sex'. You can override using the `.groups`
argument.
```

```
# A tibble: 81 x 3
# Groups:   sex [3]
   sex   species     mean_weight
   <chr> <chr>             <dbl>
 1 F     albigula          154.
```

```
 2 F     baileyi           30.2
 3 F     eremicus          22.8
 4 F     flavus             7.97
 5 F     fulvescens        13.7
 6 F     fulviventer       69
 7 F     hispidus          69.0
 8 F     leucogaster       31.1
 9 F     leucopus          19.3
10 F     maniculatus       22.1
# i 71 more rows
```

```r
surveys %>%
  group_by(species, sex) %>%
  summarise(mean_weight = mean(weight, na.rm = TRUE))
```

```
`summarise()` has grouped output by 'species'. You can override using the
`.groups` argument.
```

```
# A tibble: 81 x 3
# Groups:   species [40]
   species         sex   mean_weight
   <chr>           <chr>       <dbl>
 1 albigula        F           154.
 2 albigula        M           166.
 3 albigula        <NA>        168.
 4 audubonii       <NA>        NaN
 5 baileyi         F            30.2
 6 baileyi         M            33.8
 7 baileyi         <NA>         30.6
 8 bilineata       <NA>        NaN
 9 brunneicapillus <NA>        NaN
10 chlorurus       <NA>        NaN
# i 71 more rows
```

to avoid using `na.rm = FALSE` each statistics

```r
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(species, sex) %>%
  summarise(mean_weight = mean(weight), sd_weight = sd(weight), sd_count = n())
```

```
`summarise()` has grouped output by 'species'. You can override using the
`.groups` argument.
```

```
# A tibble: 59 x 5
# Groups:   species [22]
   species  sex   mean_weight sd_weight sd_count
   <chr>    <chr>       <dbl>     <dbl>    <int>
 1 albigula F           154.      39.2        652
 2 albigula M           166.      49.0        484
 3 albigula <NA>        168.      44.2         16
 4 baileyi  F            30.2      5.27      1617
 5 baileyi  M            33.8      8.27      1188
 6 baileyi  <NA>         30.6      9.96         5
 7 eremicus F            22.8      4.57       568
 8 eremicus M            20.6      3.49       689
 9 eremicus <NA>         17.7      0.577        3
10 flavus   F             7.97     1.69       742
# i 49 more rows
```

arrange by mean weight

```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(species, sex) %>%
  summarise(mean_weight = mean(weight), sd_weight = sd(weight), sd_count = n()) %>%
  arrange(mean_weight)
```

```
`summarise()` has grouped output by 'species'. You can override using the
`.groups` argument.
```

```
# A tibble: 59 x 5
# Groups:   species [22]
   species   sex   mean_weight sd_weight sd_count
   <chr>     <chr>       <dbl>     <dbl>    <int>
 1 flavus    <NA>         6        1.63         4
 2 taylori   M            7.36     0.842       14
 3 flavus    M            7.89     1.59       802
 4 flavus    F            7.97     1.69       742
 5 taylori   F            9.16     2.24        31
 6 montanus  M            9.5      1.29         4
 7 megalotis M           10.1      1.73      1339
```

```
 8 montanus  F           11        2.16         4
 9 megalotis <NA>        11.1      2.57        12
10 megalotis F           11.1      2.56      1184
# i 49 more rows
```

in descending order

```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(species, sex) %>%
  summarise(mean_weight = mean(weight), sd_weight = sd(weight), sd_count = n()) %>%
  arrange(desc(mean_weight))
```

```
`summarise()` has grouped output by 'species'. You can override using the
`.groups` argument.
```

```
# A tibble: 59 x 5
# Groups:   species [22]
   species     sex   mean_weight sd_weight sd_count
   <chr>       <chr>       <dbl>     <dbl>    <int>
 1 albigula    <NA>        168.       44.2       16
 2 albigula    M           166.       49.0      484
 3 albigula    F           154.       39.2      652
 4 hispidus    <NA>        130        NA          1
 5 spilosoma   M           130        NA          1
 6 spectabilis M           122.       24.0     1220
 7 spectabilis <NA>        120        18.5       18
 8 spectabilis F           118.       21.5     1106
 9 fulviventer F            69        37.8       16
10 hispidus    F            69.0      29.7       98
# i 49 more rows
```

by count

```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(species, sex) %>%
  summarise(mean_weight = mean(weight), sd_weight = sd(weight), sd_count = n()) %>%
  arrange(sd_count)
```

```
`summarise()` has grouped output by 'species'. You can override using the
`.groups` argument.
```

```
# A tibble: 59 x 5
# Groups:   species [22]
   species     sex   mean_weight sd_weight sd_count
   <chr>       <chr>       <dbl>     <dbl>    <int>
 1 hispidus    <NA>        130      NA            1
 2 intermedius <NA>         18      NA            1
 3 leucopus    <NA>         25      NA            1
 4 spilosoma   F            57      NA            1
 5 spilosoma   M           130      NA            1
 6 fulviventer <NA>         40.5     6.36         2
 7 leucogaster <NA>         29      11.3          2
 8 eremicus    <NA>         17.7     0.577        3
 9 ordii       <NA>         50.7     6.51         3
10 sp.         F            20.7     1.15         3
# i 49 more rows
```

1.8.4 Count

Count of a categorical column

```
surveys %>%
  count(sex)
```

```
# A tibble: 3 x 2
  sex       n
  <chr> <int>
1 F     15690
2 M     17348
3 <NA>   1748
```

1.8.5 Reshaping

Using gather & spreed

prepare the needed data first

```
surveys_gw <- surveys %>%
  filter(!is.na(weight)) %>%
  group_by(genus, plot_id) %>%
  summarize(mean_weight = mean(weight))
```

```
`summarise()` has grouped output by 'genus'. You can override using the
`.groups` argument.
```

creating a 2D table where each dimension represent a category the cell will represent a statistics

```
surveys_spread <- surveys_gw %>%
  spread(key = genus, value = mean_weight)
str(surveys_spread)
```

```
tibble [24 x 11] (S3: tbl_df/tbl/data.frame)
 $ plot_id        : num [1:24] 1 2 3 4 5 6 7 8 9 10 ...
 $ Baiomys        : num [1:24] 7 6 8.61 NA 7.75 ...
 $ Chaetodipus    : num [1:24] 22.2 25.1 24.6 23 18 ...
 $ Dipodomys      : num [1:24] 60.2 55.7 52 57.5 51.1 ...
 $ Neotoma        : num [1:24] 156 169 158 164 190 ...
 $ Onychomys      : num [1:24] 27.7 26.9 26 28.1 27 ...
 $ Perognathus    : num [1:24] 9.62 6.95 7.51 7.82 8.66 ...
 $ Peromyscus     : num [1:24] 22.2 22.3 21.4 22.6 21.2 ...
 $ Reithrodontomys: num [1:24] 11.4 10.7 10.5 10.3 11.2 ...
 $ Sigmodon       : num [1:24] NA 70.9 65.6 82 82.7 ...
 $ Spermophilus   : num [1:24] NA NA NA NA NA NA NA NA NA NA ...
```

```
head(surveys_spread)
```

```
# A tibble: 6 x 11
  plot_id Baiomys Chaetodipus Dipodomys Neotoma Onychomys Perognathus Peromyscus
    <dbl>   <dbl>       <dbl>     <dbl>   <dbl>     <dbl>       <dbl>      <dbl>
1       1    7           22.2      60.2    156.      27.7        9.62       22.2
2       2    6           25.1      55.7    169.      26.9        6.95       22.3
3       3    8.61        24.6      52.0    158.      26.0        7.51       21.4
4       4   NA           23.0      57.5    164.      28.1        7.82       22.6
5       5    7.75        18.0      51.1    190.      27.0        8.66       21.2
6       6   NA           24.9      58.6    180.      25.9        7.81       21.8
# i 3 more variables: Reithrodontomys <dbl>, Sigmodon <dbl>, Spermophilus <dbl>
```

bring spread back

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

```
tibble [240 x 3] (S3: tbl_df/tbl/data.frame)
 $ plot_id    : num [1:240] 1 2 3 4 5 6 7 8 9 10 ...
 $ genus      : chr [1:240] "Baiomys" "Baiomys" "Baiomys" "Baiomys" ...
 $ mean_weight: num [1:240] 7 6 8.61 NA 7.75 ...
```

```
head(surveys_gw)
```

```
# A tibble: 6 x 3
  plot_id genus   mean_weight
    <dbl> <chr>         <dbl>
1       1 Baiomys        7
2       2 Baiomys        6
3       3 Baiomys        8.61
4       4 Baiomys       NA
5       5 Baiomys        7.75
6       6 Baiomys       NA
```

1.8.6 Filtering

Remove missing data

```
survey_complete <- surveys %>%
  filter(!is.na(weight), !is.na(hindfoot_length), !is.na(sex))
```

Filter those that has sample greater than 50

```
species_counts <- survey_complete %>%
  count(species_id) %>%
  filter(n >= 50)
```

filter only those in the indicated category

```r
surveys_com <- surveys %>%
  filter(species_id %in% c("albigula", "eremicus"))
```

1.8.7 Saving to disk

```r
write_cvs()
```

1.9 Visualization

- Help in making complex plots from data frames in simple steps
- ggplot graphics are built step by step by adding new elements; this makes it flexible as well as customization

Step 1: Bind the plot to specific data frame

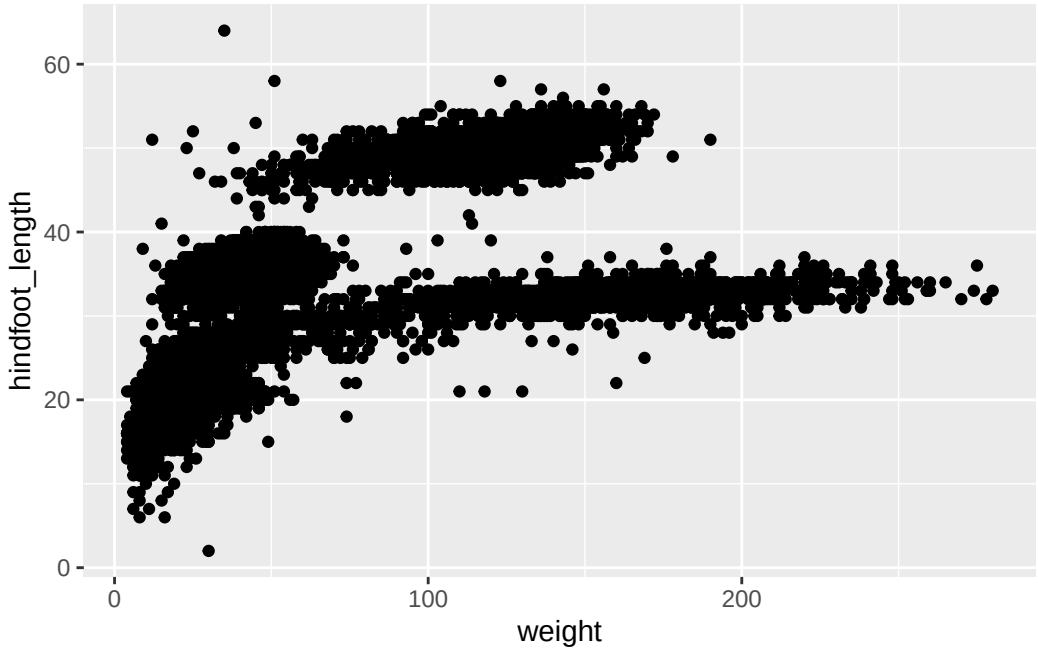
```r
surveys_plot <- ggplot(
  data = survey_complete,
  mapping = aes(x = weight, y = hindfoot_length)
)

# Color for each group
surveys_plot <- ggplot(
  data = survey_complete,
  mapping = aes(x = weight, y = hindfoot_length),
  color = species_id
)
```

Step 2: Select the type of the plot

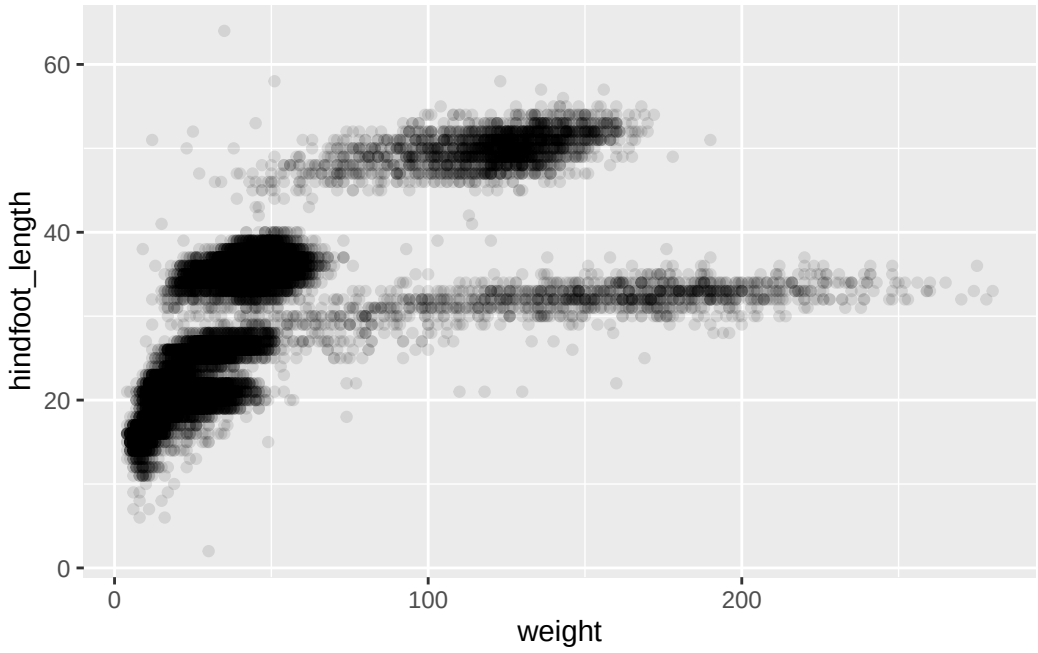
- scatter plot, dot plots, etc. > geom_point()
- boxplots > geom_boxplot()
- trend lines, time series, etc. > geom_line()

1.9.1 Scatter plot

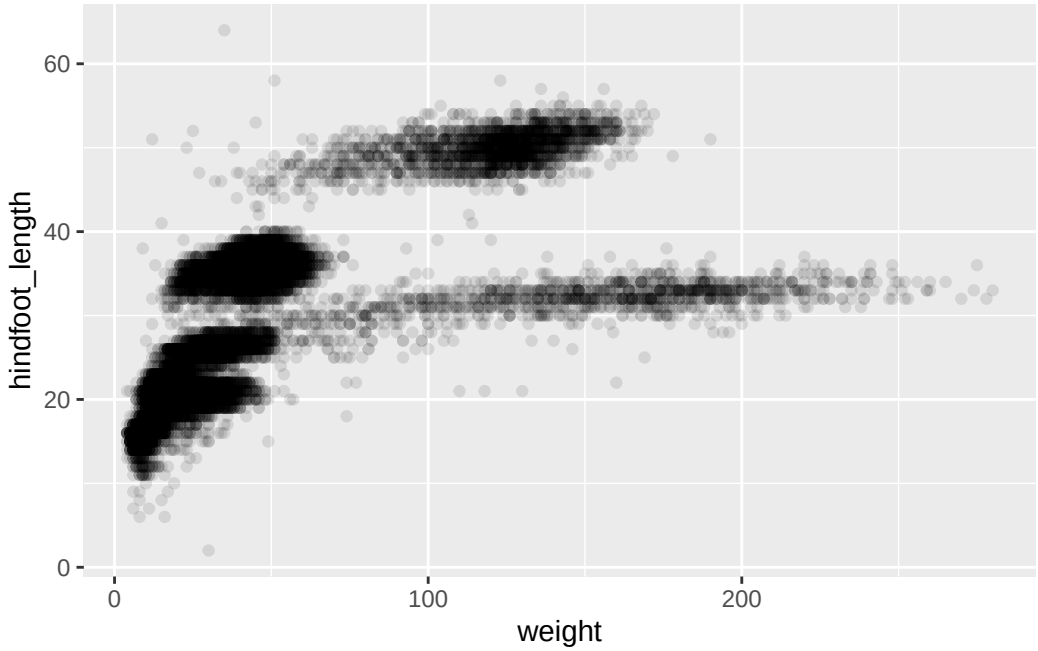
```r
surveys_plot + geom_point()
```



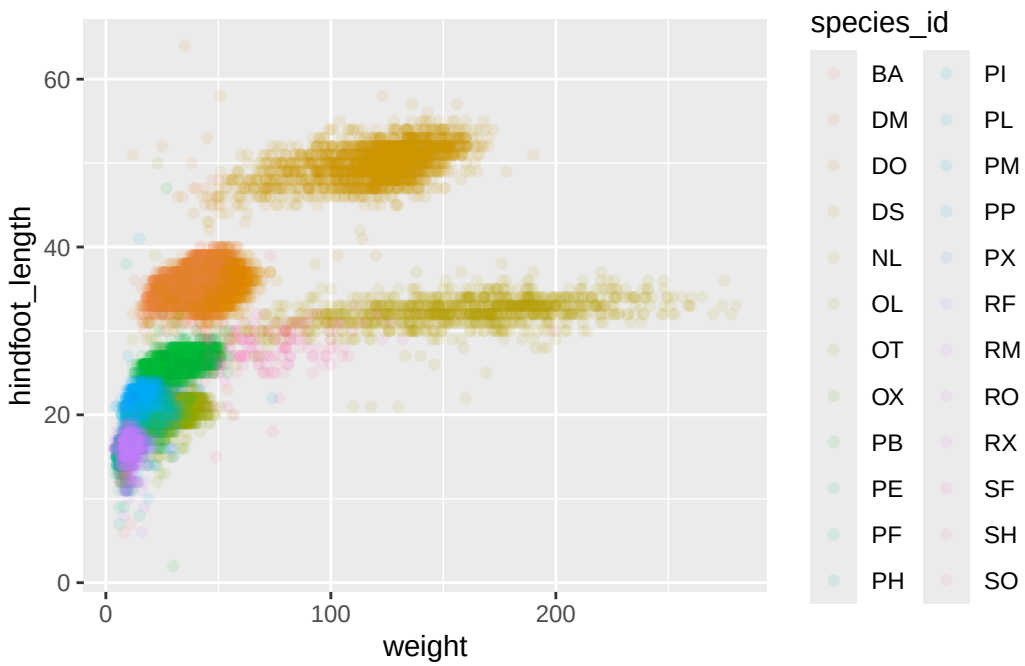

```
# add transparency
surveys_plot + geom_point(alpha = 0.1)
```



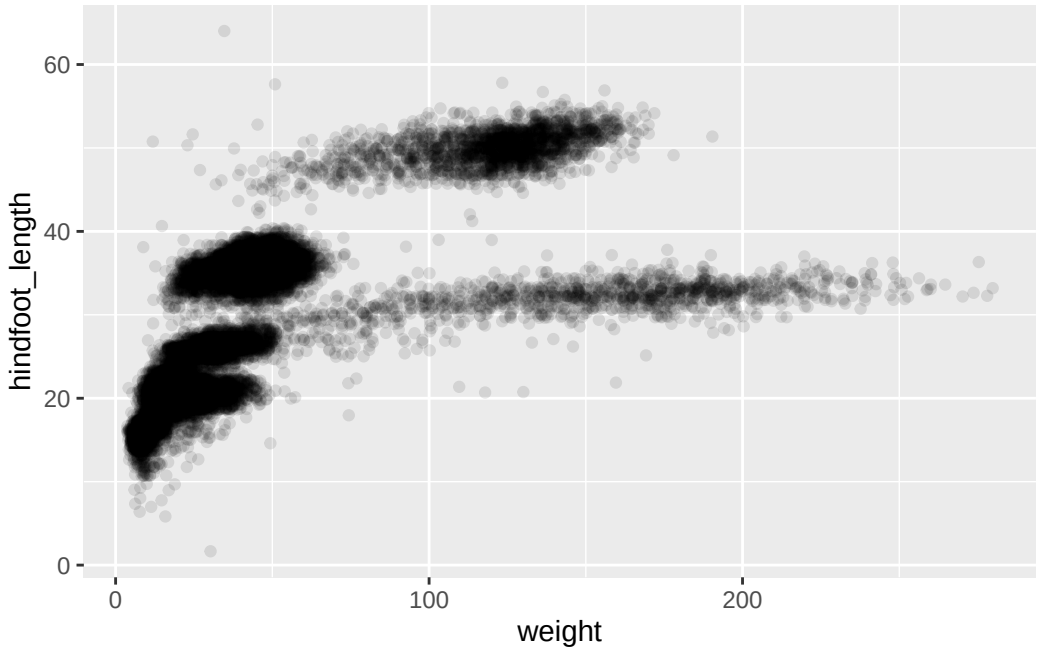
```
# color if not used in binding
surveys_plot + geom_point(alpha = 0.1, color = "black")
```




```
# add color if not used in binding
surveys_plot + geom_point(alpha = 0.1, aes(color = species_id))
```

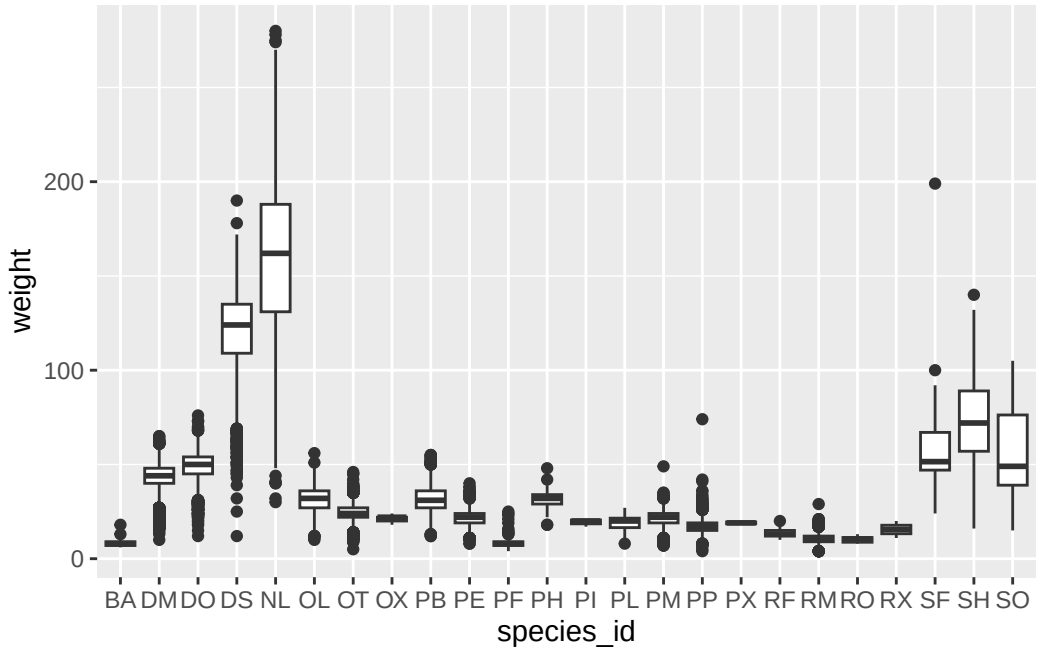



```
# make the color blend by introducing small random variation in points locations
# used when having small data sets
surveys_plot + geom_jitter(alpha = 0.1)
```

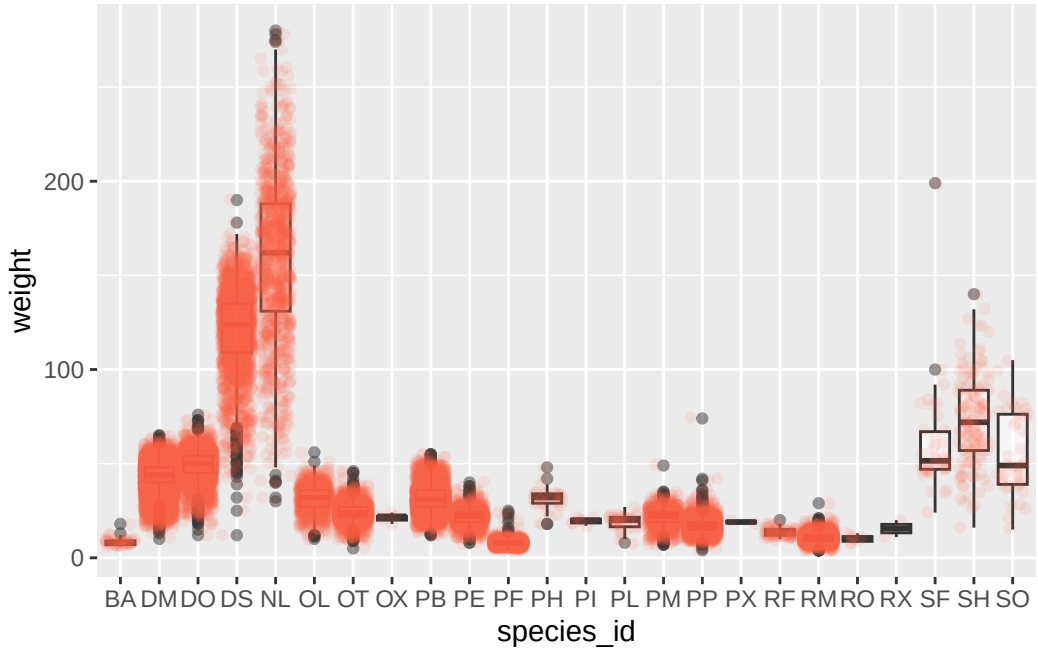


1.9.2 Boxplot

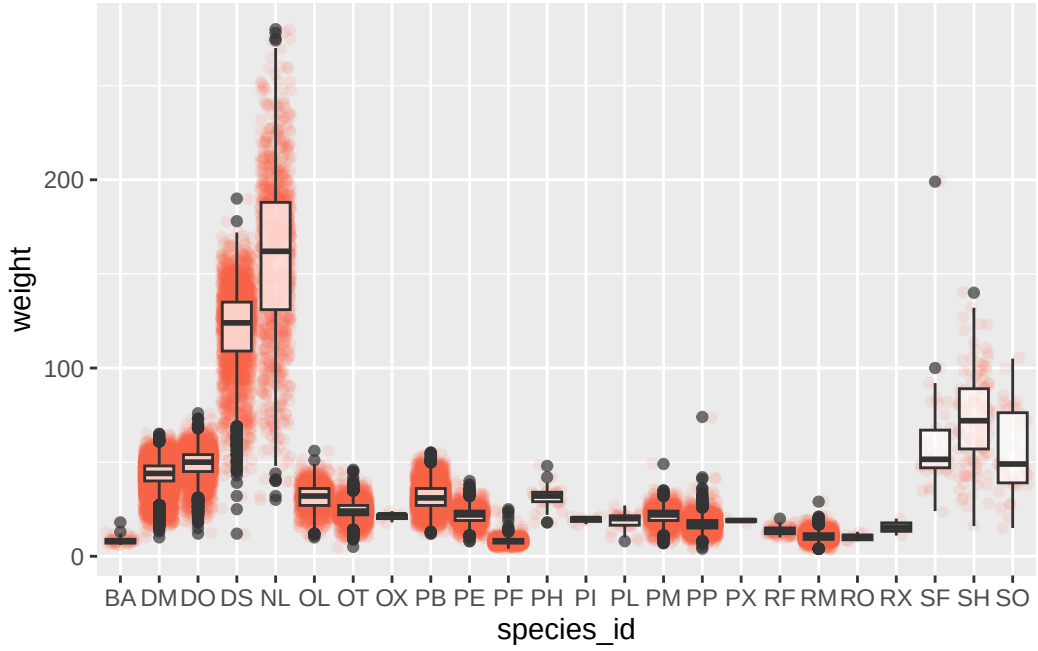
```
surveys_plot <- ggplot(
  data = survey_complete,
  mapping = aes(x = species_id, y = weight)
)

surveys_plot + geom_boxplot()
```




```
# show data
surveys_plot + geom_boxplot(alpha = 0.5) +
  geom_jitter(alpha = 0.1, color = "tomato")
```

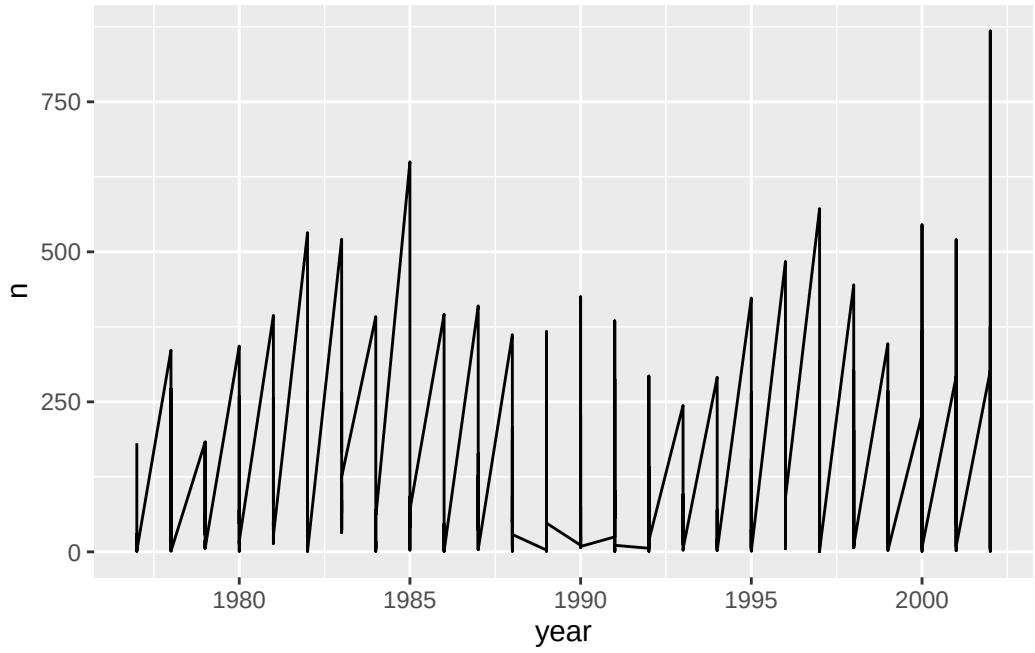



```
# bring boxplot layer in front
surveys_plot + geom_jitter(alpha = 0.1, color = "tomato") +
  geom_boxplot(alpha = 0.7)
```

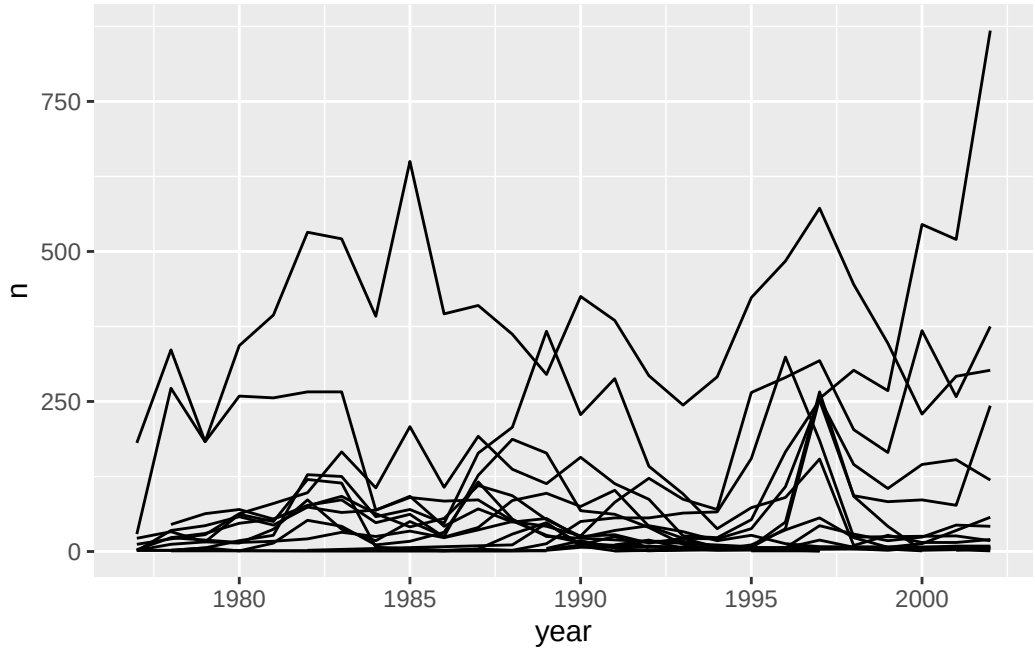


1.9.3 Time series data

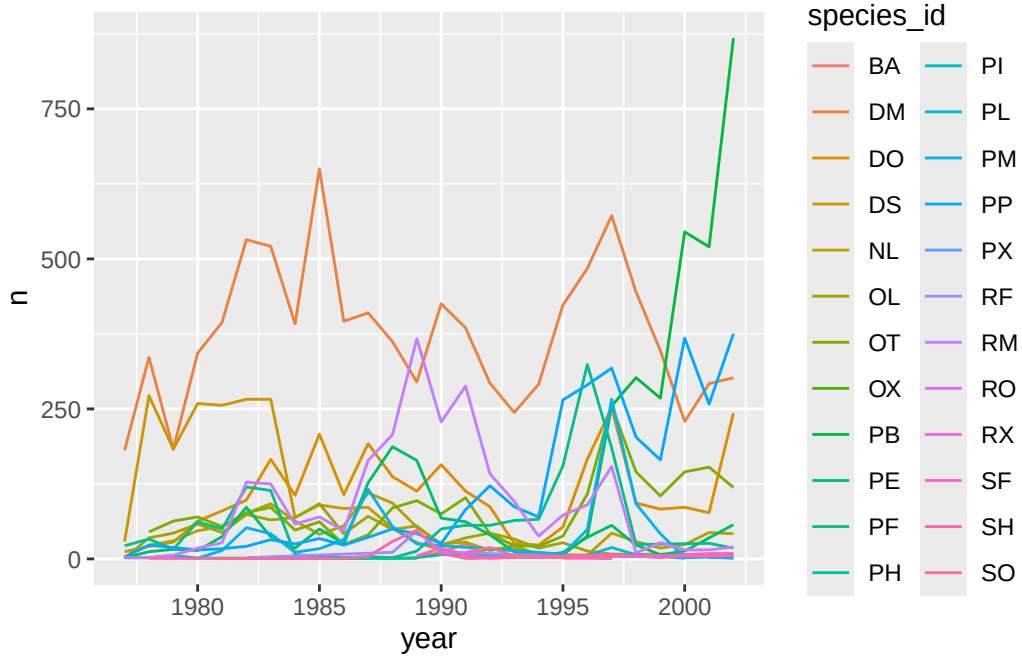
```
# create appropriate dataset
yearly_count <- survey_complete %>%
  count(year, species_id)

surveys_plot <- ggplot(
  data = yearly_count,
  mapping = aes(x = year, y = n)
)

surveys_plot + geom_line()
```




```
# make it more meaningful by breaking it by category
surveys_plot + geom_line(aes(group = species_id))
```

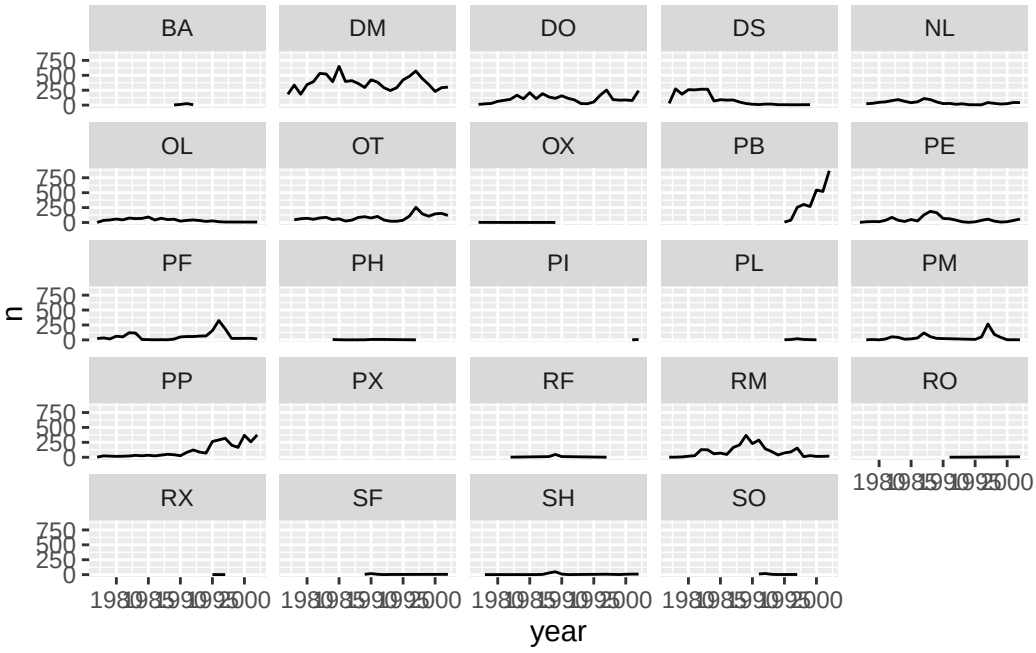



```
# make it more colorful
surveys_plot + geom_line(aes(color = species_id))
```



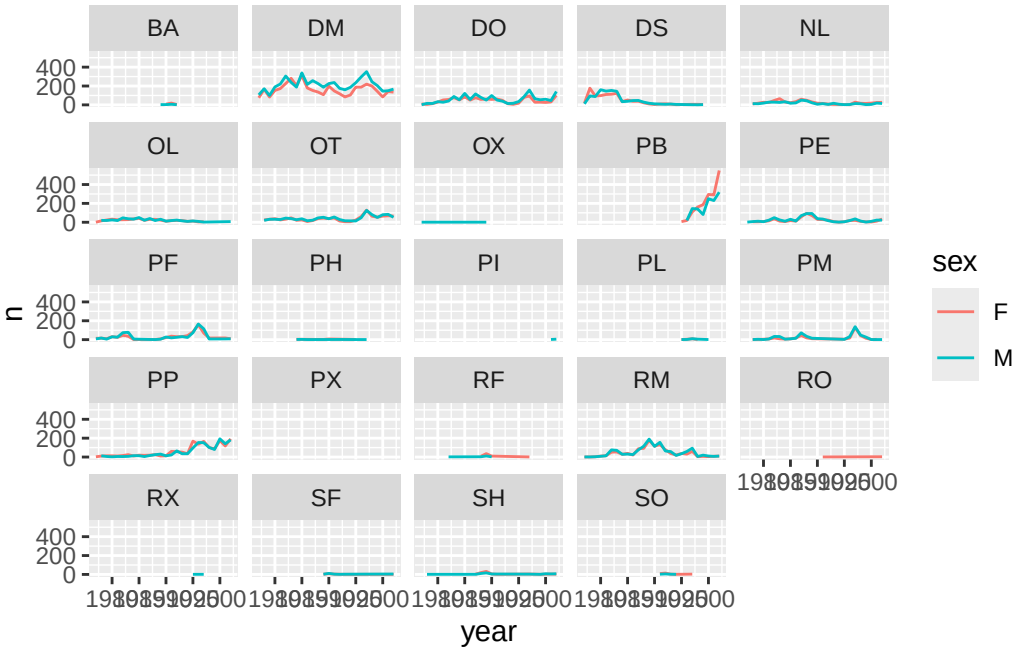
```r
# split into multiple plots
surveys_plot + geom_line() + facet_wrap(~species_id)
```

```
`geom_line()`: Each group consists of only one observation.
i Do you need to adjust the group aesthetic?
```



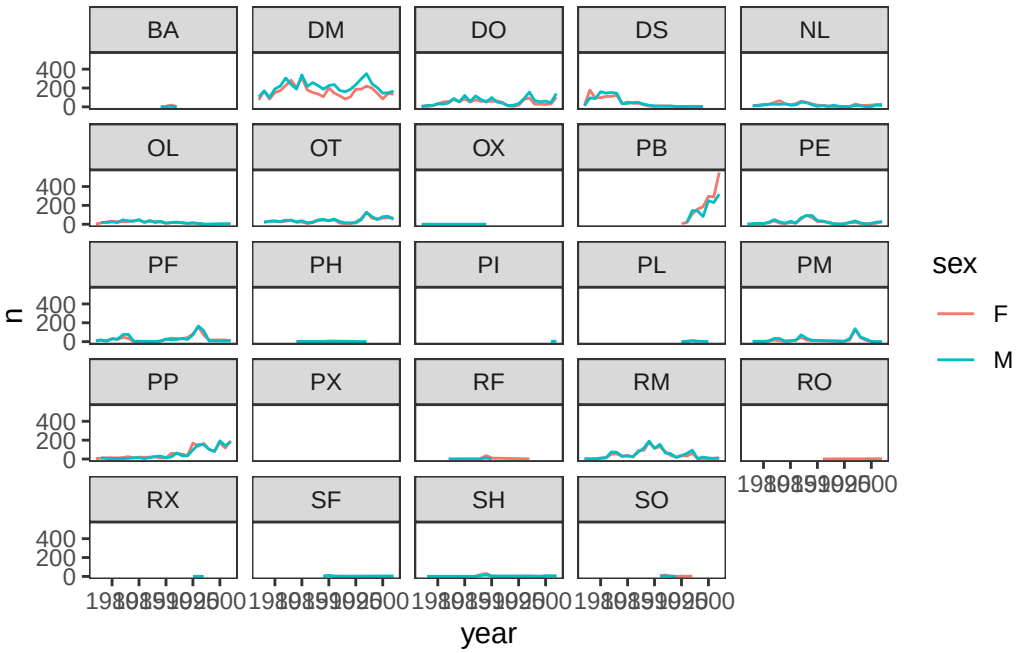

```r
# split the line in each plot by sex
yearly_sex_counts <- survey_complete %>%
  count(year, species_id, sex)

surveys_plot <- ggplot(
  data = yearly_sex_counts,
  mapping = aes(x = year, y = n)
)

surveys_plot + geom_line(aes(color = sex)) +
  facet_wrap(~species_id)
```

```
`geom_line()`: Each group consists of only one observation.
i Do you need to adjust the group aesthetic?
```




```
# remove background
surveys_plot + geom_line(aes(color = sex)) +
  facet_wrap(~species_id) +
  theme_bw() +
  theme(panel.grid = element_blank())
```

```
`geom_line()`: Each group consists of only one observation.
i Do you need to adjust the group aesthetic?
```



1.10 References

- OU Software Carpentry Workshop (check other workshops here)
  - Main Tutorial
  - Data Carpentry with R
  - Software Carpentry with R
  - Etherpad
  - Google Doc
- Intro to ggplot by Allison Horst
- R for Data Science book by Garrett Grolemund and Hadley Wickham
- Best Practices for Scientific Computing paper

2 Quarto

2.1 Render & Review

1. VSCode/RStudio -> *Render* button
2. Terminal -> `quarto preview`
3. Terminal -> `quarto preview help`

2.2 Render w/o Review

1. Terminal -> `quarto render`
2. Terminal -> `quarto render help`

2.3 References

- Quarto Reference

3 RStudio

3.1 Keyboard Shortcuts

Below is a set of helpful keyboard shortcut. The full list can be reached by clicking Tools -> Keyboard Shortcut Help (`Alt+Shift+K`) -> See All Shortcuts

- `Ctrl+Enter`: run line of code on which cursor is standing
- `Ctrl+Alt+I`: insert new code chunk
- `Ctrl+Shift+C`: comment/un-comment
- `Alt-` : move line up
- `Alt-` : move line down
- `Ctrl+D`: delete line
- `Ctrl+Shift+A`: format code
- `Ctrl+M`: add pipe `|>` operator–to change from `%>%`, go to Tools -> Global Options… -> Code section -> Editing tab -> check Use native pipe operator, `|>` (requires R 4.1+)
- `Ctrl++`: increase font size of all windows
- `Ctrl+-`: decrease font size of all windows
- `Alt+-` : insert assignment `<-`. Notice:
  - the inserted assignment it is surrounded by spaces
  - the action happen when the cursor is inside an R chunk or R script file

Part II

R for Data Science (2e)

This section contains replications for the examples used in the R for Data Science (2e) book, one Quarto file per chapter of the book.

4 Data Visualization

4.1 Introduction

This page introduces the plot creation vocabulary step by step

4.2 Clear Workspace

Always start by clearing the workspace. This ensure objects created in other files are not used used here.

```
rm(list = ls())
```

4.3 Packages

4.3.1 List

List all the packages that will be used in this script.

```
packages = c("palmerpenguins", 'ggthemes', 'ggplot2', 'dplyr', 'here', 'knitr')
```

4.3.2 Install Missing

Any missing package will be installed automatically. This ensure smoother execution when run by others.

```
# Do NOT modify
install.packages(setdiff(packages, rownames(installed.packages())))
```

4.3.3 Load

Load all packages

```
Loading required package: 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

Loading required package: printr

Registered S3 method overwritten by 'printr':
  method                from
  knit_print.data.frame rmarkdown

Loading required package: here

here() starts at C:/Users/Amin Alhashim/Documents/GitHub/hashcx/notebooks

Loading required package: knitr

[[1]]
[1] TRUE

[[2]]
[1] TRUE

[[3]]
[1] TRUE

[[4]]
```

```
[1] TRUE

[[5]]
[1] TRUE

[[6]]
[1] TRUE

[[7]]
[1] TRUE
```

4.4 Plot Creation Process

4.4.1 Load Dataset

The `penguins` dataset from the `palmerpenguins` package will be used for plotting. Typically, the package is loaded using the `library` function as shown in the code chunk below. However, a better approach is the one outlined in Section 8.3.

```
library(palmerpenguins)
```

Explore the dataset

```
help(penguins)
```

```
Size measurements for adult foraging penguins near Palmer Station,
Antarctica

Description:

     Includes measurements for penguin species, island in Palmer
     Archipelago, size (flipper length, body mass, bill dimensions),
     and sex. This is a subset of 'penguins_raw'.

Usage:

     penguins

Format:
```

A tibble with 344 rows and 8 variables:

species a factor denoting penguin species (Adélie, Chinstrap and Gentoo)

island a factor denoting island in Palmer Archipelago, Antarctica (Biscoe, Dream or Torgersen)

bill_length_mm a number denoting bill length (millimeters)

bill_depth_mm a number denoting bill depth (millimeters)

flipper_length_mm an integer denoting flipper length (millimeters)

body_mass_g an integer denoting body mass (grams)

sex a factor denoting penguin sex (female, male)

year an integer denoting the study year (2007, 2008, or 2009)

Source:

Adélie penguins: Palmer Station Antarctica LTER and K. Gorman. 2020. Structural size measurements and isotopic signatures of foraging among adult male and female Adélie penguins (Pygoscelis adeliae) nesting along the Palmer Archipelago near Palmer Station, 2007-2009 ver 5. Environmental Data Initiative. doi: 10.6073/pasta/98b16d7d563f265cb52372c8ca99e60f (URL: https://doi.org/10.6073/pasta/98b16d7d563f265cb52372c8ca99e60f)

Gentoo penguins: Palmer Station Antarctica LTER and K. Gorman. 2020. Structural size measurements and isotopic signatures of foraging among adult male and female Gentoo penguin (Pygoscelis papua) nesting along the Palmer Archipelago near Palmer Station, 2007-2009 ver 5. Environmental Data Initiative. doi: 10.6073/pasta/7fca67fb28d56ee2ffa3d9370ebda689 (URL: https://doi.org/10.6073/pasta/7fca67fb28d56ee2ffa3d9370ebda689)

Chinstrap penguins: Palmer Station Antarctica LTER and K. Gorman. 2020. Structural size measurements and isotopic signatures of foraging among adult male and female Chinstrap penguin (Pygoscelis antarcticus) nesting along the Palmer Archipelago near Palmer Station, 2007-2009 ver 6. Environmental Data Initiative. doi:

```
10.6073/pasta/c14dfcfada8ea13a17536e73eb6fbe9e (URL:
https://doi.org/10.6073/pasta/c14dfcfada8ea13a17536e73eb6fbe9e)

Originally published in: Gorman KB, Williams TD, Fraser WR (2014)
Ecological Sexual Dimorphism and Environmental Variability within
a Community of Antarctic Penguins (Genus Pygoscelis). PLoS ONE
9(3): e90081. doi:10.1371/journal.pone.0090081
```

Explore the dataset differently

```
dplyr::glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

4.4.2 Load Plotting Package

The `ggplot2` package will be used for plotting. The package is typically loaded using the `library` function as shown in the code chunk below. However, a better approach is the one outlined in Section 8.3.

```
library(ggplot2)
```

4.4.3 Create `ggplot` object

Create an empty canvas by instantiating a `ggplot` object using the `ggplot()` function.

```
ggplot()
```

4.4.4 Link Dataset

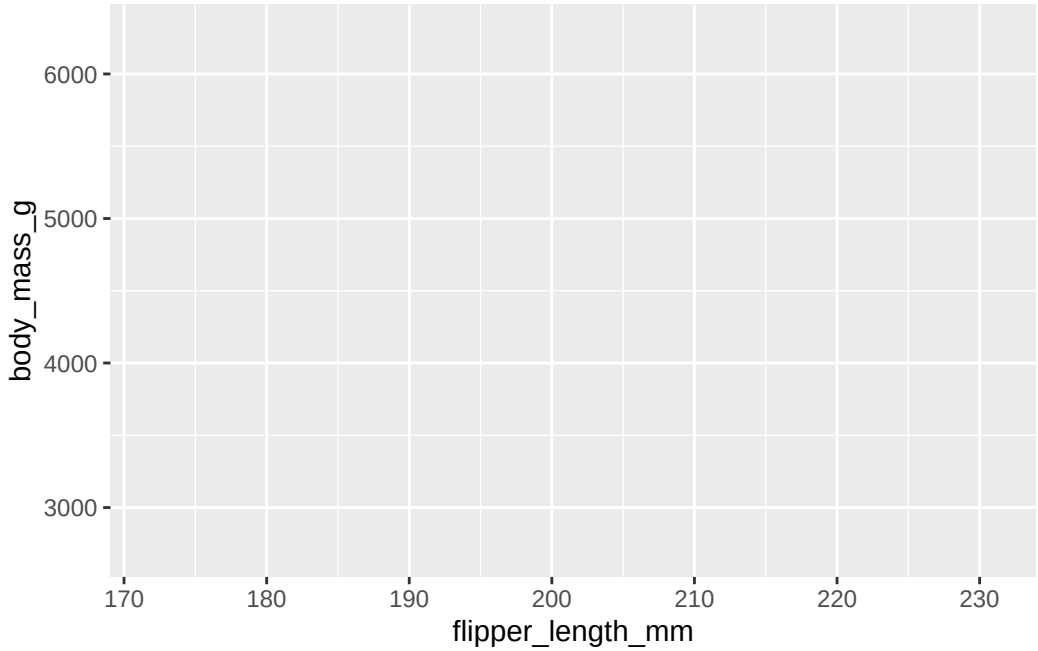
Link the dataset with the instantiated `ggplot` object using the `data` parameter.

```
ggplot(data = penguins)
```

4.4.5 Map Two Variables

Specify which of the variables in the dataset will be used as the plot aesthetics (visual properties) using the `mapping` argument done via the `aes()` function.

```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g))
```

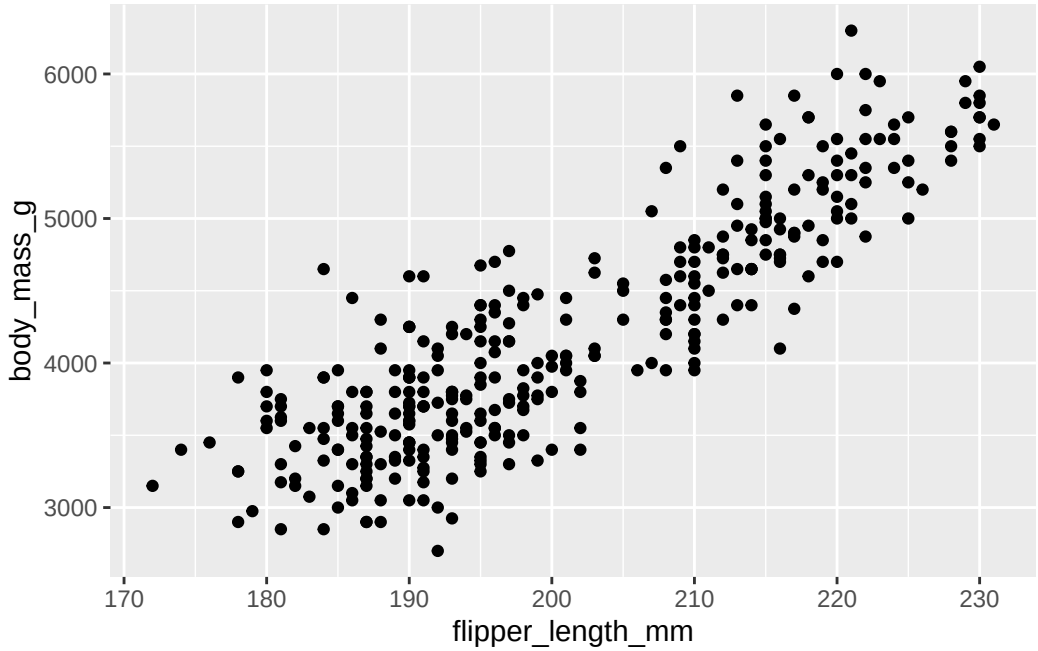


4.4.6 Display Data

Specify how the data (observations) will be represented geometrically on the plot, eg, bars, points, or line. The functions starting with `geom_` is used for this purpose. These functions add layer of the selected geometric object to the plot.

```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```

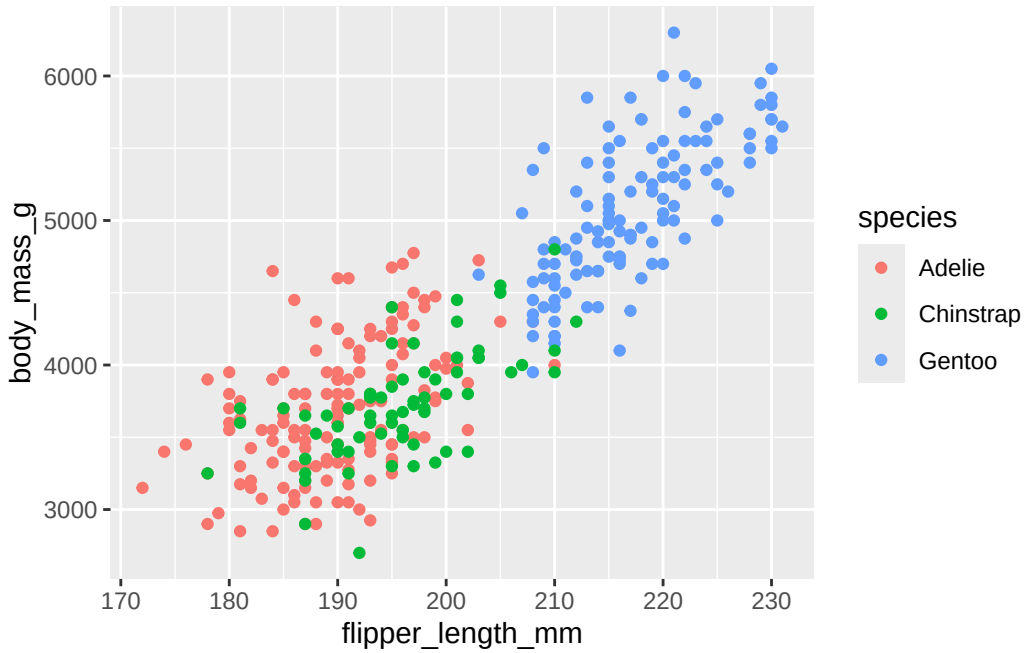


4.4.7 Map Third Variables

Other variables in the dataset can be linked to plot aesthetics (visual properties) using the `mapping` argument done via the `aes()` function.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point()
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



4.4.8 Display Three Trendlines

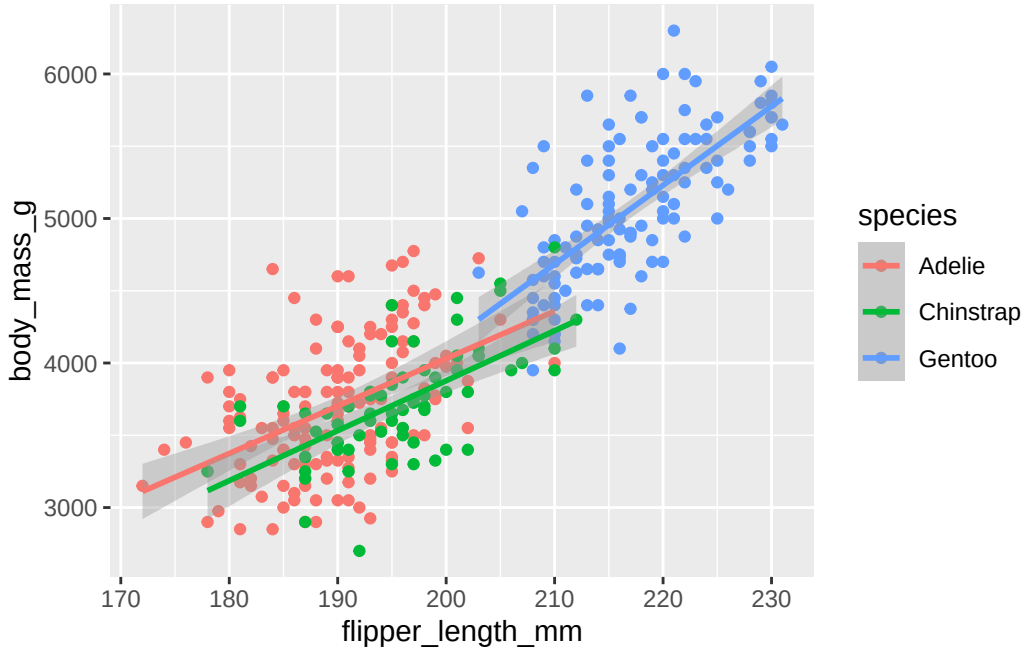
More geometric representations for the data can be specified using the functions starting with `geom_` which will add layer of the selected geometric object to the plot.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point() +
  geom_smooth(method = "lm")
```

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

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



4.4.9 Display One Trendline

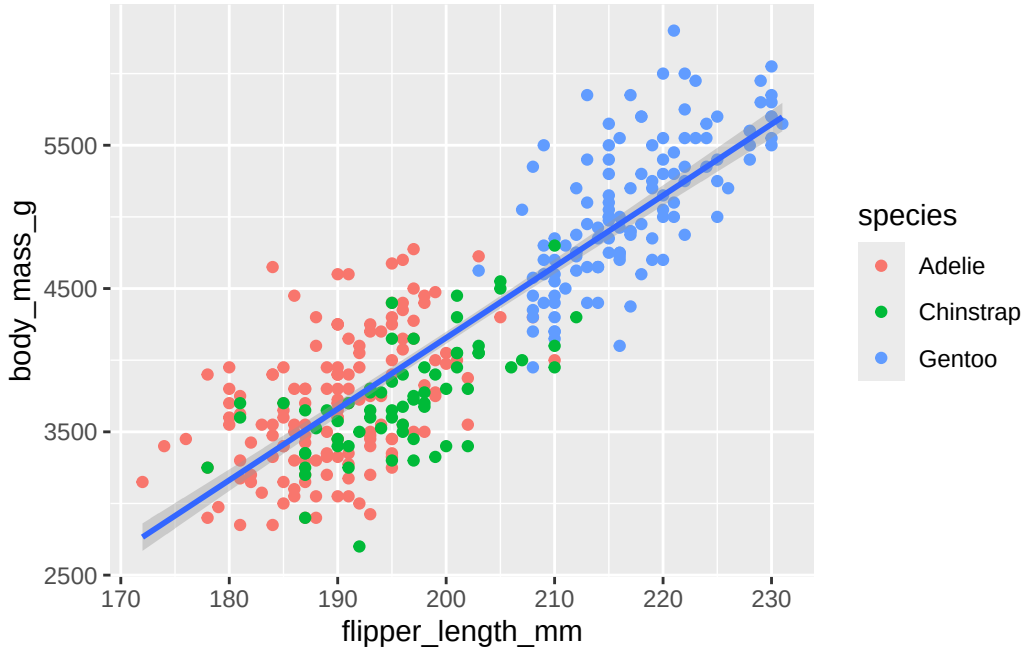
The aesthetic mapping defined in the `ggplot()` function is *global* meaning that all the `geom_()` functions inherit it. However, the aesthetic mapping defined in the `geom_()` functions are *local*, ie, not shared with other `gemo_()` functions.

```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(mapping = aes(color = species)) +
  geom_smooth(method = "lm")
```

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

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



4.4.10 Map One Variable Twice

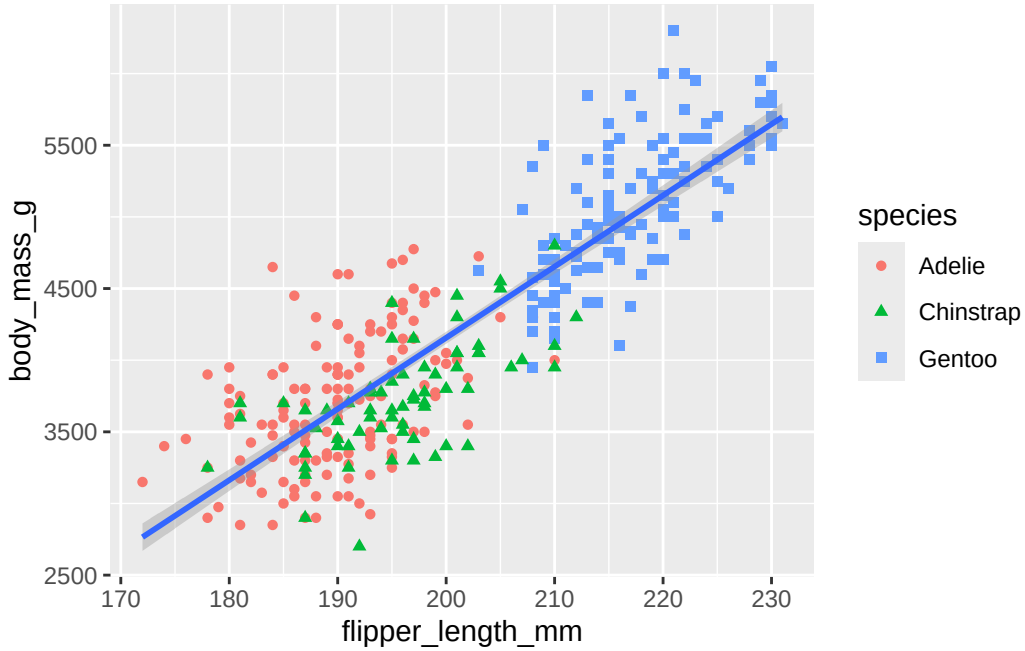
We can link the same variable to multiple plot aesthetics (visual properties) using the `mapping` parameter done via the `aes()` function.

```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = "lm")
```

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

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



4.4.11 Fix Labels

The `labs()` function can be used to make the plot more accessible. The function will add new layer to the plot and the following items can be added to the layer using the corresponding parameters

- a title using the `title` parameter
- a sub-title, if necessary, using the `subtitle` parameter
- x-axis title using the `x` parameter
- y-axis title using the `y` parameter
- data-series label or legend using the `color` and/or `shape` parameters

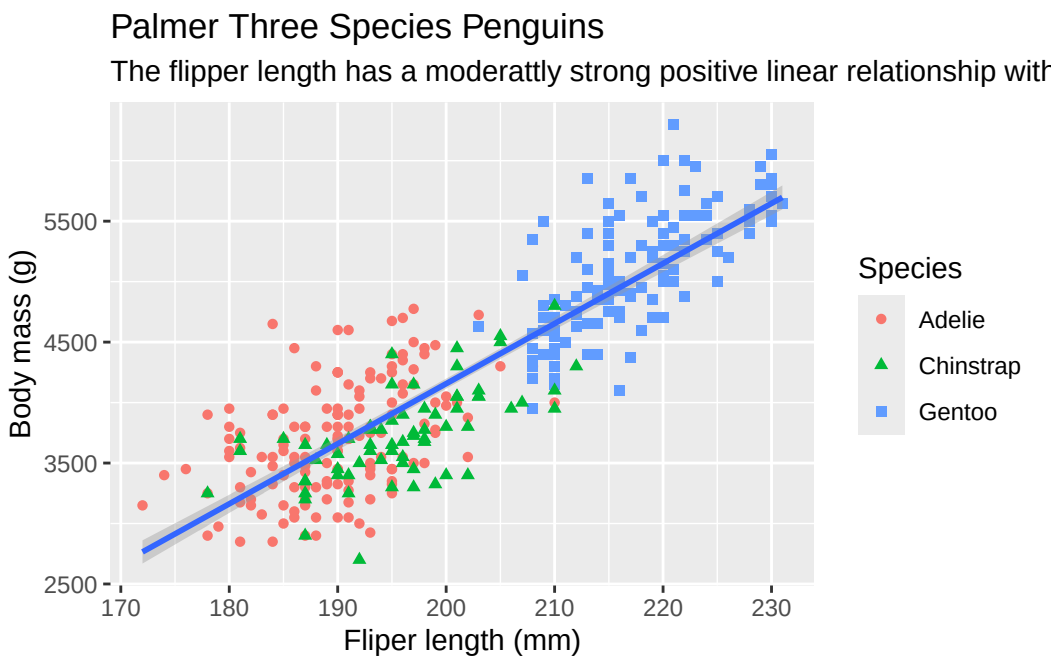
```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = "lm") +
  labs(
    title = 'Palmer Three Species Penguins',
    subtitle = 'The flipper length has a moderattly strong positive linear relationship with
    x = 'Fliper length (mm)',
    y = 'Body mass (g)',
    shape = 'Species',
```

```
    color = 'Species'
  )
```

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

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



Other types of texts can be added using other functions. The other types of texts are:

- x-axis label
- y-axis label
- data labels, if necessary
- annotation for interesting or important data, if exist

4.4.12 Ensure Color-blind Safe

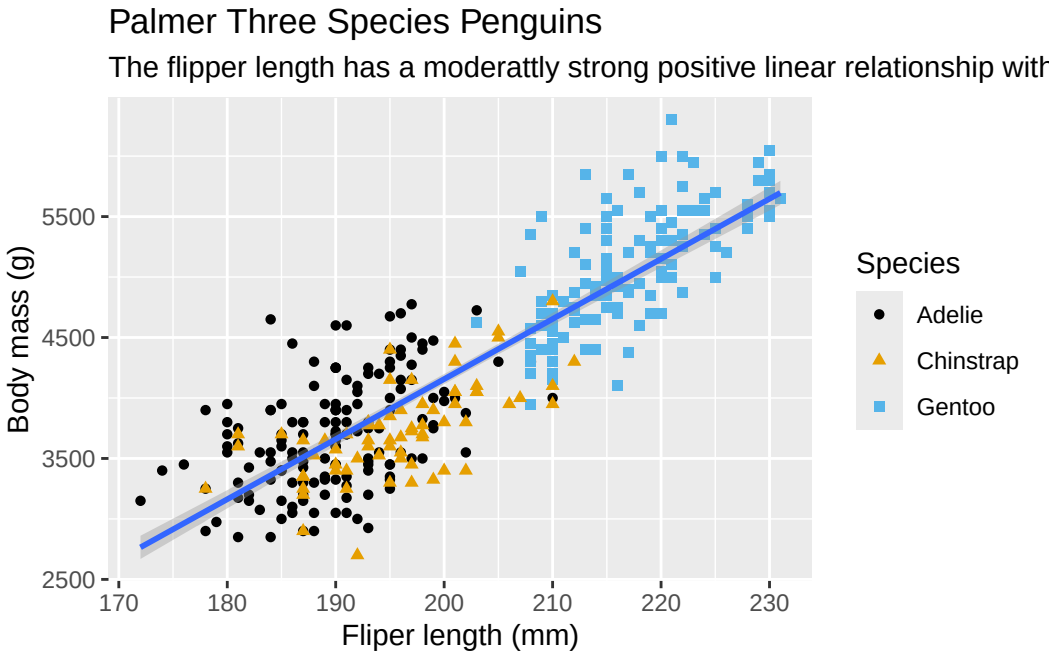
Make the plot more color-blind safe by using the `scale_color_colorblind()` function from the `ggthemes` package which will add new layer to the plot.

```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = "lm") +
  labs(
    title = 'Palmer Three Species Penguins',
    subtitle = 'The flipper length has a moderattly strong positive linear relationship with
    x = 'Fliper length (mm)',
    y = 'Body mass (g)',
    shape = 'Species',
    color = 'Species'
  ) +
  scale_color_colorblind()
```

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

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



4.4.13 Can Call Implicitly

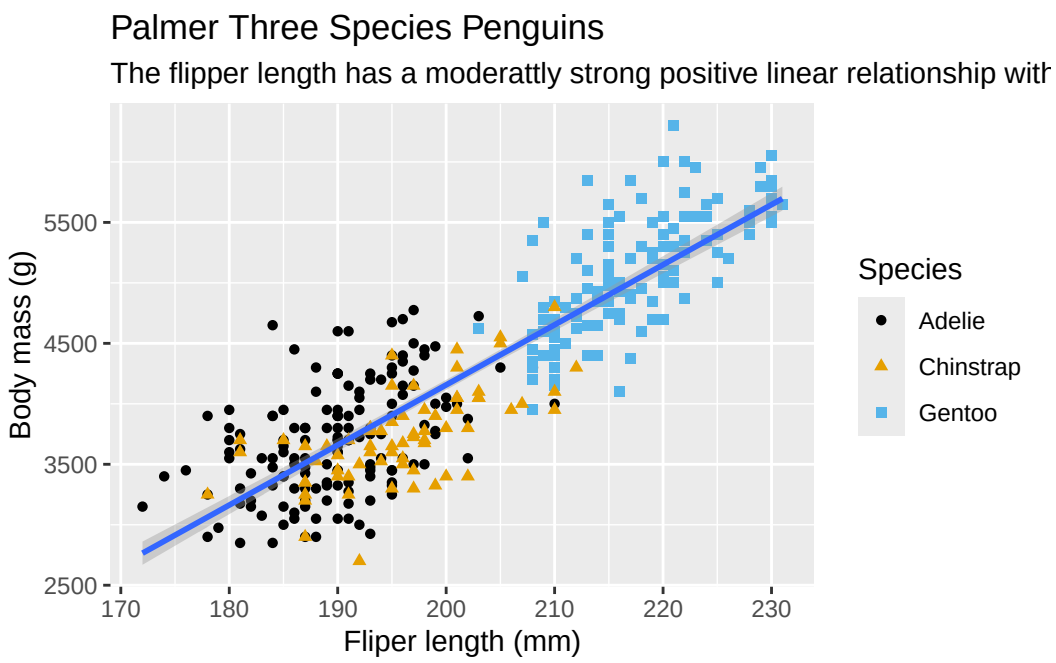
The first one or two arguments of functions are so important that scientists should know them by heart. Hence, to save some typing, the name of these arguments are usually omitted and only the values assigned to them are kept, ie, the names becomes implicit and no more explicit. Hence, the above call can be written as follows–the arguments `data` and `mapping` were omitted.

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(aes(color = species, shape = species)) +
  geom_smooth(method = "lm") +
  labs(
    title = 'Palmer Three Species Penguins',
    subtitle = 'The flipper length has a moderattly strong positive linear relationship with
    x = 'Fliper length (mm)',
    y = 'Body mass (g)',
    shape = 'Species',
    color = 'Species'
  ) +
  scale_color_colorblind()
```

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

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



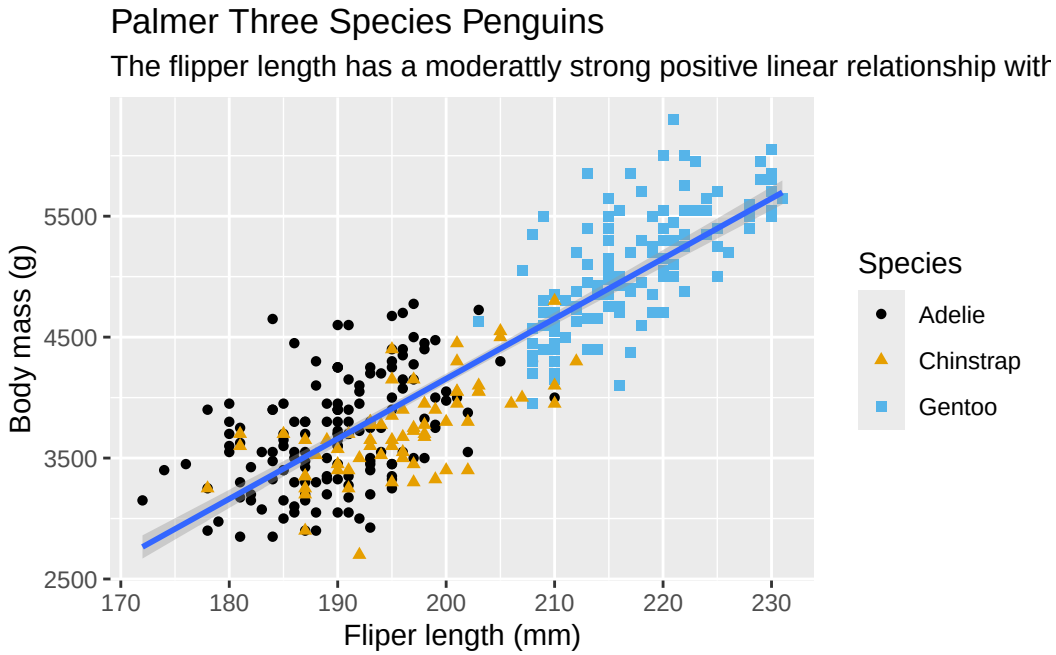
4.4.14 Use Pipe Operator

The pipe operator `\>` (shortcut: `Ctrl+M`) can be used to make the code tidy. The above code can be re-written as follow–notice the dataset was pulled before the call to the `ggplot()` function.

```
penguins |>
  ggplot(aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(aes(color = species, shape = species)) +
  geom_smooth(method = "lm") +
  labs(
    title = 'Palmer Three Species Penguins',
    subtitle = 'The flipper length has a moderattly strong positive linear relationship with
    x = 'Fliper length (mm)',
    y = 'Body mass (g)',
```

```
    shape = 'Species',
    color = 'Species'
  ) +
  scale_color_colorblind()
```

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

Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).

Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



4.5 Visualizing Distribution

4.5.1 Categorical Variables

Plot options to visualize how a categorical variable is distributed:

- bar chart, if the counts are not computed, using `gemo_bar()` function
- column chart, if the counts are computed, `gemo_col()` function

4.5.2 Numerical Variables

Plot options to visualize how a numerical (discrete or continuous) variable is distributed:

- histogram, using `geom_histogram()` function

> **Histogram Bin Width**
>
> The bin width of the histogram is in the unit of the variable mapped to the plot `x` (or `y`) aesthetic (visual property)

- density plot, using `geom_density()` function
- boxplot, using `geom_boplot()` function

> **Boxplot Components**
>
> As described beautifully in R4DS, a boxplot consists of:
>
> 1. A box that describes the range of the middle half of the data, a distance known as the interquartile range (IQR), stretching from the 25th percentile of the distribution to the 75th percentile.
> 2. A line in the middle of the box displaying the median, ie, the 50th percentile, of the distribution.
> 3. The box and the line give sense of the spread of the distribution and whether or not the distribution is symmetric about the median or skewed to one side
> 4. Visual points that display the observations that fall more than 1.5 time the IQR from either edge of the box. These outlying points (hence called outliers) are unusual so are plotted individually
> 5. A whisker that extend from each end of the box and goes to the farthest non-outlier point in the distribution
>
> Below is the diagram from R4DS showing the above components and how the boxpot is created.

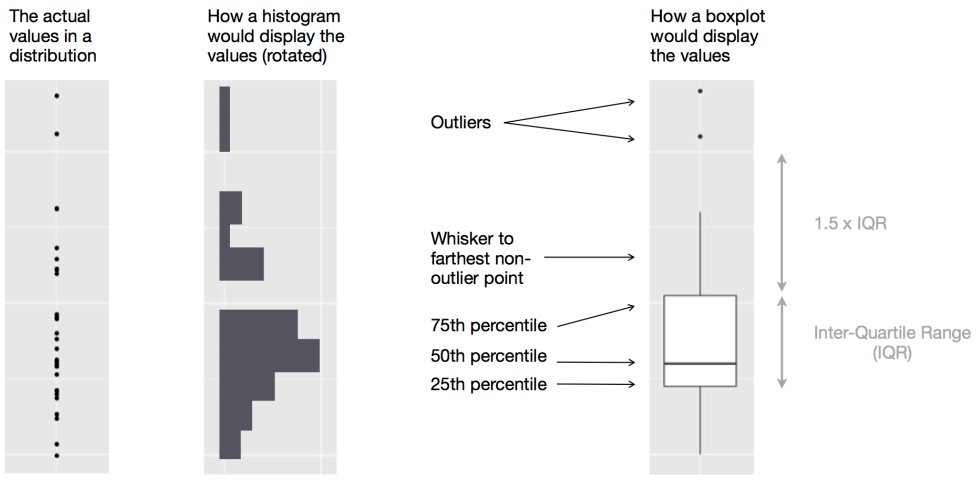


Figure 4.1: Boxplot Components (taken from R4DS)

4.6 Visualizing Relationships

4.6.1 One Categorical + One Numerical

For each category of the categorical variable, We can use any of the plot options mentioned above for the numerical variables

4.6.2 Two Categoricals

Each category of one of the categorical variables will be placed on the x-axis (or the y-axis) by mapping it to the plot `x` (or `y`) aesthetic (visual property) of the `geom_bar()` and the distribution of the categories of the other categorical variables by mapping it to the plot `fill` aesthetic (visual property). The second variable can be shown as:

- pure counts (stacked bar chart), or
- percentages (percent stack bar chart) by setting the `position` attribute of the `geom_bar()` to `fill`.

4.6.3 Two Numerical

Plot options to show the relationship between two numerical variables are:

- Scatter plot using the `geom_point()` function
- trend line using `geom_smooth()` function
- line graph using `geom_line()` function if one of the variables is monotonic, eg, time or date.

4.6.4 Three or More Variables

To visualize 3+ variables, We can either

- map variables to other aesthetics of the plot, eg, *color*, *size*, and *shape*
- split plot into facets, subplots that each display one subset of the data, based on a categorical variable using `facet_wrap()` function where its first argument is a formula created using `~` followed by a (categorical) variable name.

5 Workflow: Basics

5.1 Introduction

This page covers basic concepts when working with R. I took note for those that were new to me or found useful to remind myself with.

5.2 Clear Workspace

Always start by clearing the workspace. This ensure objects created in other files are not used used here.

```
rm(list = ls())
```

5.3 Packages

5.3.1 List

List all the packages that will be used in this script.

```
packages = c()
```

5.3.2 Install Missing

Any missing package will be installed automatically. This ensure smoother execution when run by others.

```
# Do NOT modify
install.packages(setdiff(packages, rownames(installed.packages())))
```

5.3.3 Load

Load all packages

```
list()
```

5.4 Comments

Use comments to explain the *why* of your code, eg, you changed the default value of a parameter of a function from say `.2` to `.9`, why?

5.5 Nameing Objects Rules

- Allowed characters when naming objects
  - letters
  - numbers
  - `_`
  - `.`
- All names must start with a letter
- R is case-sensitive, ie, `var`, `Var`, and `VAR` are different names

6 Data Transformation

6.1 Introduction

This page introduces the `dplyr` package used to transform data such as creating new variables, editing existing variables, filtering out observations, and creating summaries.

6.2 Clear Workspace

Always start by clearing the workspace. This ensure objects created in other files are not used used here.

```
rm(list = ls())
```

6.3 Packages

6.3.1 List

List all the packages that will be used in this script.

```
packages = c("nycflights13", "ggplot2", "dplyr")
```

6.3.2 Install Missing

Any missing package will be installed automatically. This ensure smoother execution when run by others.

```
# Do NOT modify
install.packages(setdiff(packages, rownames(installed.packages())))
```

6.3.3 Load

Load all packages

```
Loading required package: nycflights13

Loading required package: ggplot2

Loading required package: 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

[[1]]
[1] TRUE

[[2]]
[1] TRUE

[[3]]
[1] TRUE
```

6.4 `dplyr` Functions (Verbs)

6.4.1 Four Groups

`dplyr` functions (verbs) can be grouped into functions that work on:

- rows, eg, `filter()`, `arrange()`, `distinct()`, `count()`
- columns, eg, `mutate()`, `select()`, `rename()`, `relocate()`
- groups, eg, `summarize()`, `slice_max`, `group_by`, `ungroup()`, `.by`
- tables

6.4.2 Common Characteristics

All the functions have the followings in common:

- their first argument is always a data frame
- their subsequent arguments typically describe which columns to operate on using variable names *without quotes*
- they always output a new data frame, they don't modify the passed one

6.4.3 Pipe |> Operator

- The pipe `|>` operator takes what on its left and pass it to the function on its right so that `x |> f(y)` (pronounced as `x` *then* `f(y)`) is equivalent to `f(x, y)` and `x |> f(y) |> g(z)` (pronounced as `x` then `f(y)` then `g(z)`) is equivalent to `g(f(x, y), z)`
- The `base` R pipe operator `|>` was introduced in R 4.1.0 in 2021 while the tidyverse `magrittr` pipe operator `%>%` was introduced in 2014. Using `|>` instead of `%>%` makes our code run when we don't use tidyverse

6.4.4 Row Functions

- The following `filter()` statements are equivalent:
  - `filter(ds, var == 1 or var == 2)`
  - `filter(ds, var == 1 | var == 2)`
  - `filter(ds, var %in% c(1,2))`
- The following `filter()` statements are equivalent:
  - `filter(ds, var == 1 and var == 2)`
  - `filter(ds, var == 1, var == 2)`
- The following `arrange()` statements order data differently
  - `arrange(ds, var)` ascendant order
  - `arrange(ds, desc(var))` decedent order
- The following `distinct()` statements return different data frames
  - `distinct(ds, var1, var2)` only keep columns `var1` and `var2`
  - `distinct(ds, var1, var2, .keep_all = TRUE)` keep all the columns–find the *first* observation where `var1` and `var2` are distinct and discard the rest
- The following `count()` statements order the results differently
  - `count(ds, var1, var2)` arrange results in order they are encountered
  - `count(ds, var1, var2, sort = TRUE)` arrange results in descending order of number of occurrence

6.4.5 Column Functions

- `mutate()`
  - Instead of adding the newly created variable to the right hand side of the data frame, we can instruct `mutate()` to adding before a variable using the `.before` attribute or after a variable using the `.after` attribute
  - To only keep the variables involved in the creation of the new variables, we can instruct `mutate()` to do so by setting the `.keep = "used"` attribute
- `select()`
  - select range of variables: `select(ds, var_x:var_y)`
  - select all variables except certain range: `select(ds, !var_x:var_y)`
  - select character variables only: `select(ds, where(is.character))`
  - select variables whose name start with something: `select(ds, start_with("m"))`
  - select variables whose name end with something: `select(ds, end_with("m"))`
  - select variables whose name contain something: `select(ds, contains("m"))`
  - select variables whose name follow some range: `select(ds, num_range("x", 1:3))`
  - select and rename variable: `select(ds, var1_new = var1, var2_new = var2)`
- `rename()`
  - to rename many columns, it is better to use `janitor::clean_names()` function
- `relocate()`
  - By default, bring columns to left hand side of the data frame
  - `relocate(ds, var1, .after = var2)` puts `var1` after `var2`
  - `relocate(ds, var1, .before = var2)` puts `var1` before `var2`

6.4.6 Groups Functions

- `group_by()`
  - divides the data into groups so that subsequent operations work on these groups
  - it added a *class* to the dataset to indicate the grouping
- `summarize()` or `summarise()`
  - To prevent summary statistics functions, eg, `mean()` to give `NA` due to some groups has `NA` (missing) values, set their argument: `na.rm = TRUE`
  - The summary statistics function `n()` gives the number of observations in the group

  - Each summary peels off the last group. To prevent this behavior, change the default value `drop_last` of the `.groups` argument of the summary statistic function to either `keep` to keep all groups or `drop` to drop all groups

- `` `slice_` functions ``
  - The functions are: `slice_head()`, `slice_tail()`, `slice_min()`, `slice_max()`, and `slice_sample()`
  - To slice a number of rows from each group, use the `n` arguments, eg, `n = 1`
  - To slice percentage of rows from each group, use the `prop` argument, eg, `prop = .1` (10%)
  - To prevent ties from showing, use `with_ties = FALSE` argument

- `` `.by` argument ``
  - New addition to dplyr 1.1.0 (more information at dplyr 1.1.0 blog post)
  - Per-operation grouping–can be used all verbs. The advantage is that we don't need to use the `.groups` argument to suppress the warning message raised by `summarize()` when grouping by multiple variables and we don't need to use `ungroup()` when done with our summary.

7 Workflow: Code Style

7.1 Introduction

This page covers code style concepts when working with R. I took note for those that were new to me or found useful to remind myself with.

> **Code Style & Punctuation**
>
> Code style is like punctuation, when used correctly, itmakecodereadeasily.

7.2 Clear Workspace

Always start by clearing the workspace. This ensure objects created in other files are not used used here.

```
rm(list = ls())
```

7.3 Packages

7.3.1 List

List all the packages that will be used in this script.

```
packages <- c("styler")
```

7.3.2 Install Missing

Any missing package will be installed automatically. This ensure smoother execution when run by others.

```
# Do NOT modify
install.packages(setdiff(packages, rownames(installed.packages())))
```

7.3.3 Load

Load all packages

```
Loading required package: styler
```

```
[[1]]
[1] TRUE
```

7.4 Styling Overview

7.4.1 Consistency

Although there are styling guidelines (see below for example) that one can follow, it is important that a programmer pick one and stick with it to make easy for other including future self to read your work.

7.4.2 Guidelines

There is not official styling guideline for R. However, there are different styling guidelines that one can adopt, below are some of those found by search R styling guidelines (html):

- tidyverse Style Guide (html) by Hadley Wickham. **This is the adopted guidelines in these notes.**
- R Style Guide (html) by Google
- R Coding Conventions (html) by Henrik Bengtsson, Assoc Professor, Dept of Statistics, University of California, Berkeley
- Coding Style (html) by Bioconductor project (website)
- R Style Guide (html) by Jean Fan (GitHub), Assistant Professor, Center for Computational Biology, Department of Biomedical Engineering, Johns Hopkins University

7.4.3 Automatic

There are package that can be used to automatically style existing code. Below are some of those:

- `styler` package (website) by Lorenz Walthert (website). After installing the package, launch RStudio's command palette using the keyboard shortcut `Ctrl+Shift+P`, type `styler`, and select from the available commands

7.5 Styling Specifics

7.5.1 Names

- Use meaningful names
- snake_case is used to separate_multi_word_variables
- variables with certain theme should start with the same common word/letter to make use of the auto-complete functionality

7.5.2 Spaces

- Except `^`, put spaces on both sides of mathematical operators
- Put spaces on both sides of the assignment operator, `<-`

```
# Strive for
z <- (a + b)^2 / d

# Avoid
z<-( a + b ) ^ 2/d
```

- Don't put spaces inside or outside parentheses for regular function calls
- Always put a space after a comma

```
# Strive for
mean(x, na.rm = TRUE)

# Avoid
mean (x ,na.rm=TRUE)
```

- It is okay to use extra space so align things.

```r
flights |>
  mutate(
    speed      = distance / air_time,
    dep_hour   = dep_time %/% 100,
    dep_minute = dep_time %%  100
  )
```

7.5.3 Pipes |>

The roles for pipes are nicely summarized in R4DS. Most of them are copied below.

- Put a space before it
- It should typically the last thing on a line. This make it easy to
  - add new steps
  - rearrange existing steps
  - modify elements within a step
  - quickly skip the verbs on the left-hand side
- After the first step of the pipeline, indent each line by two spaces

```r
# Strive for
flights |>
  filter(!is.na(arr_delay), !is.na(tailnum)) |>
  count(dest)

# Avoid
flights|>filter(!is.na(arr_delay), !is.na(tailnum))|>count(dest)
```

- If piping to a function without named arguments and its arguments fit on one line,
  - put all of them on one line.
- If piping to a function with named arguments OR the function has not named arguments but the arguments do not fit on line,
  - put each argument on new line indented by two spaces
  - make sure the `)` is on its own line and un-indented to match the horizontal position of the function name

```r
# Strive for
flights |>
  group_by(tailnum) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()
  )

# Avoid
flights |>
  group_by(
    tailnum
  ) |>
  summarize(delay = mean(arr_delay, na.rm = TRUE), n = n())

# Avoid
flights|>
  group_by(tailnum) |>
  summarize(
              delay = mean(arr_delay, na.rm = TRUE),
              n = n()
            )

# Avoid
flights|>
  group_by(tailnum) |>
  summarize(
  delay = mean(arr_delay, na.rm = TRUE),
  n = n()
  )
```

> **Long Pipeline**
>
> Break long pipelines (tasks) to meaningful pipelines (sub-tasks) and save the intermediate steps. This will make the code more readable and easy to check and debug.

7.5.4 `ggplot2`

The same rules from pipe can be applied to `ggplot2`.

```r
flights |>
  group_by(dest) |>
  summarize(
    distance = mean(distance),
    speed = mean(distance / air_time, na.rm = TRUE)
  ) |>
  ggplot(aes(x = distance, y = speed)) +
  geom_smooth(
    method = "loess",
    span = 0.5,
    se = FALSE,
    color = "white",
    linewidth = 4
  ) +
  geom_point()
```

7.5.5 Sectioning Comments

When writing long scripts, it is advisable to break the code into sections and using *sectioning* comments to label them. The RStudio keyboard shortcut to create such comment is `Cnrl+Shift+R`.

```r
# Load data --------------------------------------

# Plot data --------------------------------------
```

8 Data Tidying

8.1 Introduction

This page explains how use the `tidyr` package to put data in tidy form where:

- each row represents an observation
- each column represents a variable
- each cell contain a single value

Putting data in tidy form will make it easy to process using tidyverse packages.

8.2 Clear Workspace

Always start by clearing the workspace. This ensure objects created in other files are not used used here.

```
rm(list = ls())
```

8.3 Packages

8.3.1 List

List all the packages that will be used in this script.

```
packages <- c("styler", "dplyr", "tidyr")
```

8.3.2 Install Missing

Any missing package will be installed automatically. This ensure smoother execution when run by others.

```
# Do NOT modify
install.packages(setdiff(packages, rownames(installed.packages())))
```

8.3.3 Load

Load all packages

```
Loading required package: styler
```

```
Loading required package: 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
```

```
Loading required package: tidyr
```

```
[[1]]
[1] TRUE

[[2]]
[1] TRUE

[[3]]
[1] TRUE
```

8.4 Lengthening Data, `pivot_longer`

8.4.1 One Variable in Column Headers

8.4.1.1 Toy Example

```
ds <- tribble(
  ~id, ~A, ~B_1, ~B_2,
  "A", 1, 10.1, 10.2,
  "B", 2, 20.1, NA,
  "C", 3, NA, 30.2
)

ds
```

```
# A tibble: 3 x 4
  id        A   B_1   B_2
  <chr> <dbl> <dbl> <dbl>
1 A         1  10.1  10.2
2 B         2  20.1  NA
3 C         3  NA    30.2
```

8.4.1.2 Lengthen

> I want the values in all the columns that start with `B_` to be placed into a (single) column named `value`. To distinguish which value belong to which column, create a new column called `B_type` for this purpuse.

```
ds_lengthen <- ds |>
  pivot_longer(
    cols = starts_with("B_"),
    names_to = "B_type",
    values_to = "value"
  )

ds_lengthen
```

```
# A tibble: 6 x 4
  id        A B_type value
```

```
  <chr> <dbl> <chr>  <dbl>
1 A         1 B_1     10.1
2 A         1 B_2     10.2
3 B         2 B_1     20.1
4 B         2 B_2     NA
5 C         3 B_1     NA
6 C         3 B_2     30.2
```

8.4.1.3 Remove NA

Use the argument `values_drop_na = TRUE`

```
ds_lengthen <- ds |>
  pivot_longer(
    cols = starts_with("B_"),
    names_to = "B_type",
    values_to = "value",
    values_drop_na = TRUE
  )

ds_lengthen
```

```
# A tibble: 4 x 4
  id        A B_type value
  <chr> <dbl> <chr>  <dbl>
1 A         1 B_1     10.1
2 A         1 B_2     10.2
3 B         2 B_1     20.1
4 C         3 B_2     30.2
```

8.4.1.4 Fix Cell Values

Use the `readr::parse_number()` function to extract the first number from var2 variable and ignore all other text.

```
ds_lengthen <- ds |>
  pivot_longer(
    cols = starts_with("B_"),
    names_to = "B_type",
    values_to = "value",
```

```
    values_drop_na = TRUE
  ) |>
  mutate(
    B_type = readr::parse_number(B_type)
  )

ds_lengthen
```

```
# A tibble: 4 x 4
  id        A B_type value
  <chr> <dbl>  <dbl> <dbl>
1 A         1      1  10.1
2 A         1      2  10.2
3 B         2      1  20.1
4 C         3      2  30.2
```

8.4.2 Multiple Variables in Column Headers

8.4.2.1 Toy Example

```
ds2 <- tribble(
  ~id, ~A, ~B1_C1, ~B1_C2, ~B2_C1, ~B2_C2,
  "A", 1, 10.11, 10.12, 10.21, 10.22,
  "B", 2, 20.11, 20.12, NA, 20.22,
  "C", 3, 30.11, NA, 30.21, 30.22
)

ds2
```

```
# A tibble: 3 x 6
  id        A B1_C1 B1_C2 B2_C1 B2_C2
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
1 A         1  10.1  10.1  10.2  10.2
2 B         2  20.1  20.1  NA    20.2
3 C         3  30.1  NA    30.2  30.2
```

8.4.2.2 Lengthening w/o Seperating Variables

```
ds_lengthen <- ds2 |>
  pivot_longer(
    cols = starts_with("B"),
    names_to = "B_C",
    values_to = "value"
  )

ds_lengthen
```

```
# A tibble: 12 x 4
   id        A B_C   value
   <chr> <dbl> <chr> <dbl>
 1 A         1 B1_C1  10.1
 2 A         1 B1_C2  10.1
 3 A         1 B2_C1  10.2
 4 A         1 B2_C2  10.2
 5 B         2 B1_C1  20.1
 6 B         2 B1_C2  20.1
 7 B         2 B2_C1  NA
 8 B         2 B2_C2  20.2
 9 C         3 B1_C1  30.1
10 C         3 B1_C2  NA
11 C         3 B2_C1  30.2
12 C         3 B2_C2  30.2
```

8.4.2.3 Lengthening w/ Seperating Variables

```
ds_lengthen <- ds2 |>
  pivot_longer(
    cols = !(id:A),
    names_sep = "_",
    names_to = c("B", "C"),
    values_to = "value"
  )

ds_lengthen
```

```
# A tibble: 12 x 5
   id        A B     C     value
   <chr> <dbl> <chr> <chr> <dbl>
 1 A         1 B1    C1     10.1
 2 A         1 B1    C2     10.1
 3 A         1 B2    C1     10.2
 4 A         1 B2    C2     10.2
 5 B         2 B1    C1     20.1
 6 B         2 B1    C2     20.1
 7 B         2 B2    C1     NA
 8 B         2 B2    C2     20.2
 9 C         3 B1    C1     30.1
10 C         3 B1    C2     NA
11 C         3 B2    C1     30.2
12 C         3 B2    C2     30.2
```

8.4.2.4 Dropping NA

```
ds_lengthen <- ds2 |>
  pivot_longer(
    cols = !(id:A),
    names_sep = "_",
    names_to = c("B", "C"),
    values_to = "value",
    values_drop_na = TRUE
  )

ds_lengthen
```

```
# A tibble: 10 x 5
   id        A B     C     value
   <chr> <dbl> <chr> <chr> <dbl>
 1 A         1 B1    C1     10.1
 2 A         1 B1    C2     10.1
 3 A         1 B2    C1     10.2
 4 A         1 B2    C2     10.2
 5 B         2 B1    C1     20.1
 6 B         2 B1    C2     20.1
 7 B         2 B2    C2     20.2
 8 C         3 B1    C1     30.1
```

```
 9 C         3 B2    C1     30.2
10 C         3 B2    C2     30.2
```

8.4.3 Data and Variable Names in Colmnn Headers

8.4.3.1 Toy Example

```
ds3 <- tribble(
  ~id, ~child1_name, ~child1_age, ~child2_name, ~child2_age,
  "A", "A1", 11, "A2", 12,
  "B", "B1", 21, NA, NA,
  "C", NA, NA, "C2", 32
)

ds3
```

```
# A tibble: 3 x 5
  id    child1_name child1_age child2_name child2_age
  <chr> <chr>            <dbl> <chr>            <dbl>
1 A     A1                  11 A2                  12
2 B     B1                  21 <NA>                NA
3 C     <NA>                NA C2                  32
```

8.4.3.2 Lengthening w/o Removing NA

```
ds_lengthen <- ds3 |>
  pivot_longer(
    cols = starts_with("child"),
    names_sep = "_",
    names_to = c("child", ".value")
  )

ds_lengthen
```

```
# A tibble: 6 x 4
  id    child  name    age
  <chr> <chr>  <chr> <dbl>
1 A     child1 A1       11
```

```
2 A     child2 A2       12
3 B     child1 B1       21
4 B     child2 <NA>     NA
5 C     child1 <NA>     NA
6 C     child2 C2       32
```

8.4.3.3 Lengthening w Removing NA

```
ds_lengthen <- ds3 |>
  pivot_longer(
    cols = starts_with("child"),
    names_sep = "_",
    names_to = c("child", ".value"),
    values_drop_na = TRUE
  )

ds_lengthen
```

```
# A tibble: 4 x 4
  id    child  name    age
  <chr> <chr>  <chr> <dbl>
1 A     child1 A1       11
2 A     child2 A2       12
3 B     child1 B1       21
4 C     child2 C2       32
```

8.4.3.4 Fixing Cell Values

```
ds_lengthen <- ds3 |>
  pivot_longer(
    cols = starts_with("child"),
    names_sep = "_",
    names_to = c("child", ".value"),
    values_drop_na = TRUE
  ) |>
  mutate(
    child = readr::parse_number(child)
  )
```

```
ds_lengthen
```

```
# A tibble: 4 x 4
  id    child name    age
  <chr> <dbl> <chr> <dbl>
1 A         1 A1       11
2 A         2 A2       12
3 B         1 B1       21
4 C         2 C2       32
```

Part III

Practice

This section contains practices for the programming/scripting languages that I am learning.

9 Topic Modeling in R

9.1 Introduction

An attempt to understand Sherlock Holmes short stories found in Adventures of Sherlock Holmes book by Arthur Conan Doyle. After inspecting the table of content, the book seems to have 12 stories, one story per chapter. The analysis is inspired by Julia Silge's YouTube video Topic modeling with R and tidy data principles

9.2 Download Book

```
library(gutenbergr)  # download books from Project Gutenberg using book ID
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   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(tidytext)

# Download the book, each line of the book is read into a seperate row
sherlock_raw <- gutenberg_download(48320)
```

```
Determining mirror for Project Gutenberg from https://www.gutenberg.org/robot/harvest
Using mirror http://aleph.gutenberg.org
```

```
dim(sherlock_raw)
```

```
[1] 12350     2
```

```
head(sherlock_raw)
```

```
# A tibble: 6 x 2
  gutenberg_id text
         <int> <chr>
1        48320 "ADVENTURES OF SHERLOCK HOLMES"
2        48320 ""
3        48320 ""
4        48320 ""
5        48320 ""
6        48320 "[Illustration:"
```

```
tail(sherlock_raw)
```

```
# A tibble: 6 x 2
  gutenberg_id text
         <int> <chr>
1        48320 "  boisterious fashion, and on the whole _changed to_"
2        48320 "  boisterous fashion, and on the whole"
3        48320 ""
4        48320 "  Page 297"
5        48320 "  wrapt in the peaceful beauty _changed to_"
6        48320 "  rapt in the peaceful beauty"
```

9.3 Wrangle: Label Stories

```
sherlock <- sherlock_raw %>%

  # determine start of each story/chapter
  mutate(story = ifelse(str_detect(text, "^(A SCANDAL IN BOHEMIA|THE RED-HEADED LEAGUE|A CASE

  # determine lines belonging to each story/chapter by
  # filling down the N/A rows of story column
```

```
  fill(story) %>%

  # remove the part that does not belong to any story/chapter,
  # i.e, the introduction
  filter(!is.na(story)) %>%

  # convert story column to factor
  mutate(story = factor(story))
```

9.4 Wrangle: Put in Tidy Format

The row of `text` column contains multiple words/tokens. We want to put each word/token of each `text` row into a separate row. This makes the dataframe follows the tidy format and hence makes it easy to process.

```
tidy_sherlock <- sherlock %>%

  # number the rows
  mutate(line = row_number()) %>%

  # break the text column into multiple row where each row contain one token
  unnest_tokens(word, text) %>%

  # remove the stopwords--the rows where the word column is a stopword
  anti_join(stop_words) %>%

  # remove holmes rows which might affect our topic models
  filter(word != "holmes")
```

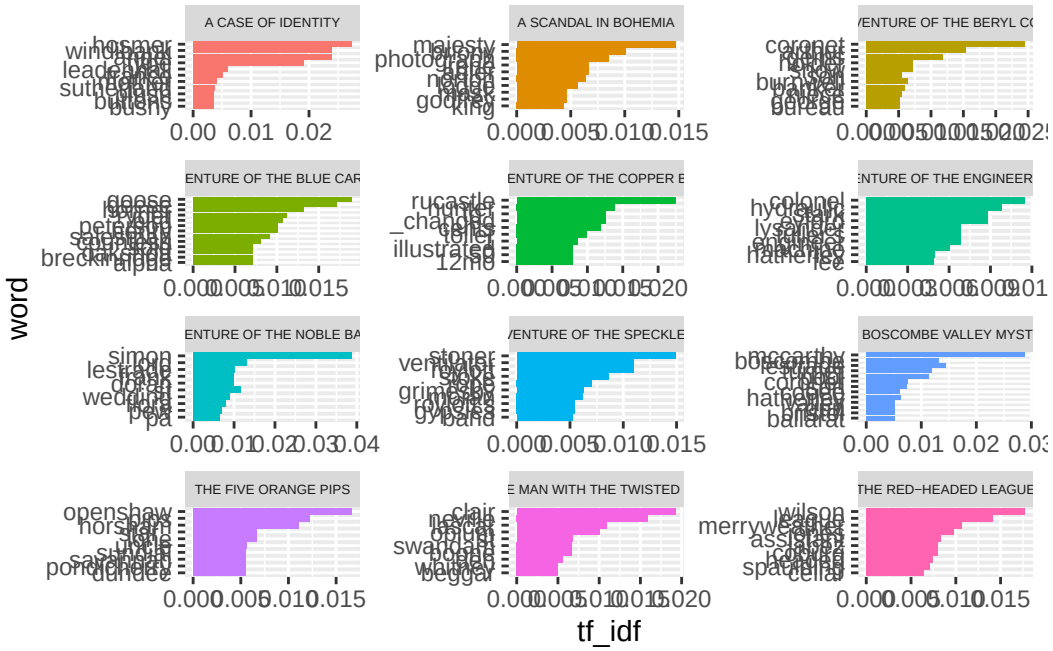
```
Joining with `by = join_by(word)`
```

9.5 Explore tf-idf

- To see which words are important in each story/chapter, i.e.,the words that appears many times in that story but few or none in the other stories.
- tf-idf (term frequency-inverse document frequency) is a great exploratory tool before starting with topic modeling

```r
library(ggplot2)

tidy_sherlock %>%

  # count number of occurrence of words in stories
  count(story, word, sort = TRUE) %>%

  # compute and add tf, idf, and tf_idf values for words
  bind_tf_idf(word, story, n) %>%

  # group by story
  group_by(story) %>%

  # take top 10 words of each story with highest tf_idf (last column)
  top_n(10) %>%

  # unpack
  ungroup() %>%

  # turn words into factors and order them based on their tf_idf values
  # NOTE: This will not affect order the dataframe rows which is can be
  #   done via the arrange function
  # NOTE: Recording the word column this way is for ggplot to visualize them
  #   as desired from top tf_idf to lowest
  mutate(word = reorder(word, tf_idf)) %>%

  # plot
  ggplot(aes(word, tf_idf, fill = story)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~story, scales = "free", ncol = 3) +
  theme(strip.text.x = element_text(size = 5)) +
  coord_flip()
```

```
Selecting by tf_idf
```



9.6 Implement Topic Modeling

Training the model for the topics

```
library(stm)        # for do topic modeling
```

```
stm v1.3.7 successfully loaded. See ?stm for help.
 Papers, resources, and other materials at structuraltopicmodel.com
```

```
library(quanteda)   # great text mining, will be used to structure the input
```

```
Package version: 4.0.2
Unicode version: 15.1
ICU version: 74.1
```

```
Parallel computing: 8 of 8 threads used.
```

```
See https://quanteda.io for tutorials and examples.
```

```
                    #   to stm

# Convert from tidy form to quanteda form (document x term matrix)
sherlock_stm <- tidy_sherlock %>%
  count(story, word, sort = TRUE) %>%
  cast_dfm(story, word, n)

# Train the model
topic_model <- stm(sherlock_stm, K=6, init.type = "Spectral")
```

```
Beginning Spectral Initialization
     Calculating the gram matrix...
     Finding anchor words...
    ......
     Recovering initialization...
    ..........................................................................
Initialization complete.
............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 1 (approx. per word bound = -7.785)
............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 2 (approx. per word bound = -7.593, relative change = 2.458e-02)
............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 3 (approx. per word bound = -7.481, relative change = 1.473e-02)
............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 4 (approx. per word bound = -7.455, relative change = 3.469e-03)
............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 5 (approx. per word bound = -7.450, relative change = 7.612e-04)
Topic 1: st, simon, lord, day, lady
 Topic 2: door, miss, house, rucastle, matter
 Topic 3: hat, goose, stone, bird, geese
 Topic 4: father, time, mccarthy, son, hand
 Topic 5: house, time, night, door, heard
```

```
 Topic 6: red, time, wilson, business, headed
.............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 6 (approx. per word bound = -7.449, relative change = 1.233e-04)
.............
Completed E-Step (0 seconds).
Completed M-Step.
Completing Iteration 7 (approx. per word bound = -7.449, relative change = 1.168e-05)
.............
Completed E-Step (0 seconds).
Completed M-Step.
Model Converged
```

```
summary(topic_model)
```

```
A topic model with 6 topics, 12 documents and a 7709 word dictionary.

Topic 1 Top Words:
     Highest Prob: st, simon, lord, day, lady, found, matter
     FREX: simon, clair, neville, lascar, opium, doran, flora
     Lift: aloysius, ceremony, doran, millar, 2_s, aberdeen, absurdly
     Score: simon, st, clair, neville, _danseuse_, lestrade, doran
Topic 2 Top Words:
     Highest Prob: door, miss, house, rucastle, matter, street, lady
     FREX: rucastle, hosmer, hunter, angel, windibank, _changed, 1
     Lift: advertised, angel, annoyance, brothers, employed, factor, fowler
     Score: rucastle, hosmer, angel, windibank, hunter, type, 1
Topic 3 Top Words:
     Highest Prob: hat, goose, stone, bird, geese, baker, sir
     FREX: geese, horner, ryder, henry, peterson, salesman, countess
     Lift: battered, bet, bred, brixton, cosmopolitan, covent, cream
     Score: goose, geese, horner, _alias_, ryder, henry, peterson
Topic 4 Top Words:
     Highest Prob: father, time, mccarthy, son, hand, lestrade, left
     FREX: mccarthy, pool, boscombe, openshaw, pips, horsham, turner
     Lift: bone, dundee, horsham, pondicherry, presumption, savannah, sundial
     Score: mccarthy, pool, lestrade, boscombe, openshaw, _détour_, turner
Topic 5 Top Words:
     Highest Prob: house, time, night, door, heard, hand, round
     FREX: coronet, stoner, arthur, roylott, ventilator, gems, stoke
     Lift: _absolute_, _all_, _en, 1100, 16a, 3d, 4000
```

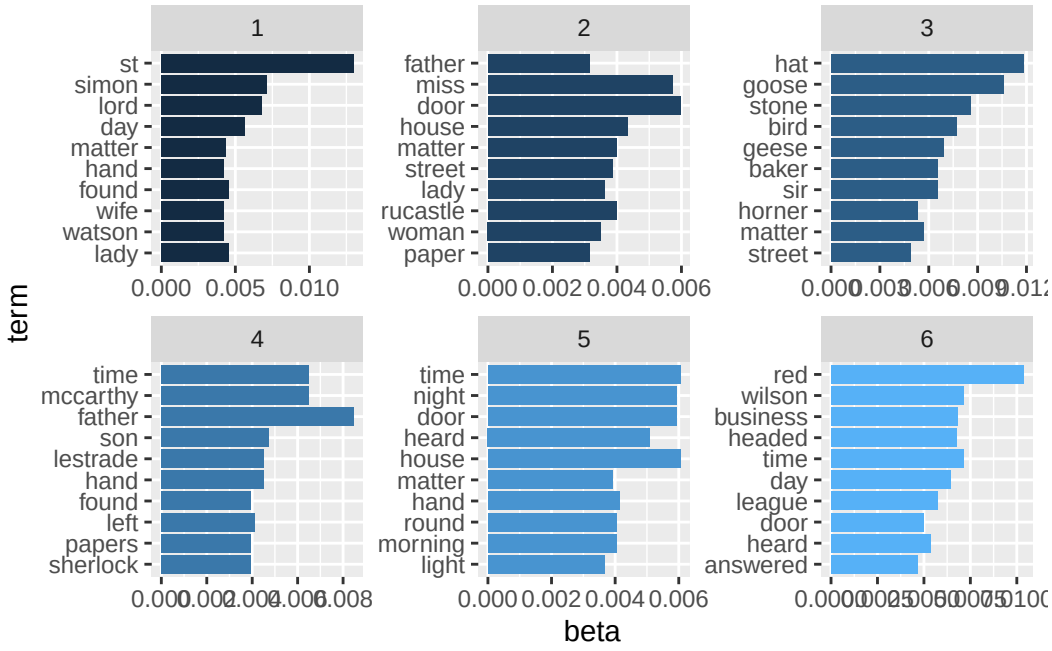
```
     Score: coronet, arthur, stoner, gems, 4000, roylott, ventilator
Topic 6 Top Words:
     Highest Prob: red, time, wilson, business, headed, day, league
     FREX: wilson, league, merryweather, jones, coburg, jabez, headed
     Lift: daring, saturday, vincent, _employé_, _october, _partie, 17
     Score: wilson, league, merryweather, _employé_, jones, headed, coburg
```

9.7 Contribution of Words in Topics

Looking at which words contribute the most in each topic.

```
# Extracting betas and putting them in a tidy format
tm_beta <- tidy(topic_model)

# Visualizing the top words contributing to each topic
tm_beta %>%
  group_by(topic) %>%
  # top 10 word in each topic with higest beta (last column)
  top_n(10) %>%
  ungroup() %>%
  # turn words into factors and order them based on their tf_idf values
  # NOTE: This will not affect order the dataframe rows which is can be
  #   done via the arrange function
  # NOTE: Recording the word column this way is for ggplot to visualize them
  #   as desired from top tf_idf to lowest
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = topic)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~topic, scales = "free", ncol = 3) +
  coord_flip()
```

```
Selecting by beta
```

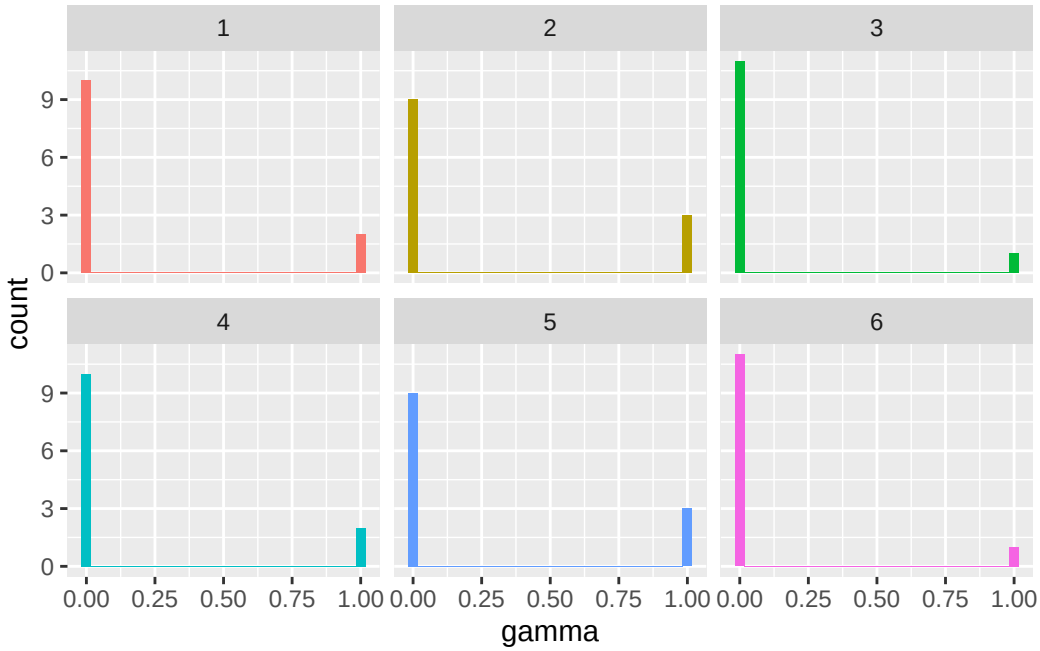


9.8 Distribution of Topics in Stories

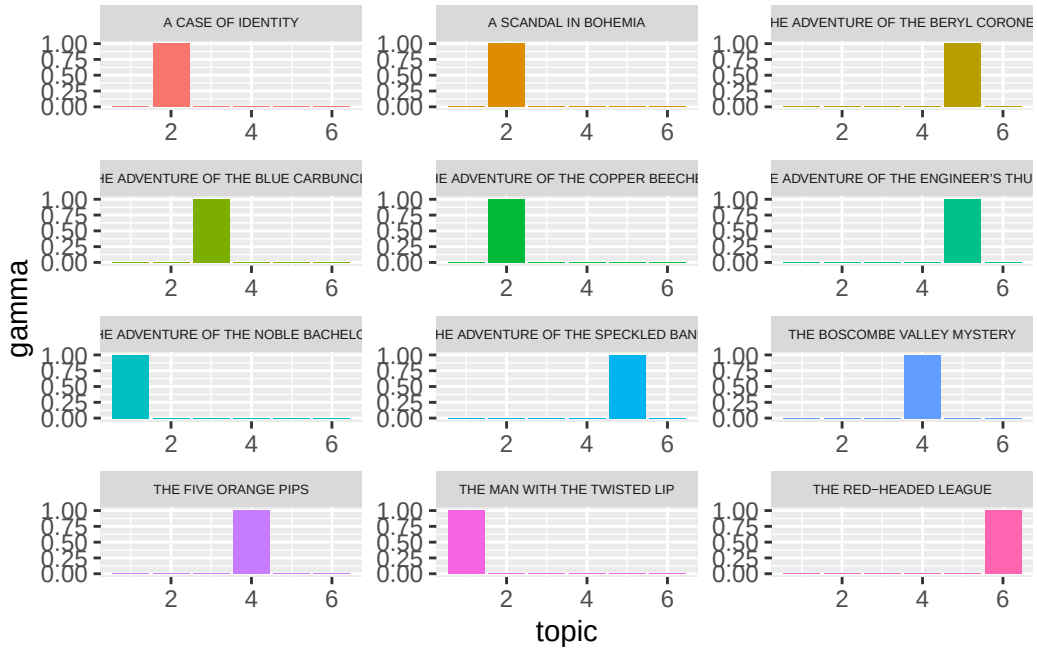
Looking at how the stories are associated with each topic and how strong each association is.

```
# Extracting gammas and putting them in a tidy format
tm_gamma <- tidy(topic_model, matrix = "gamma",
                 # use the names of the stories instead of the default numbers
                 document_names = rownames(sherlock_stm))


# Visualizing the number of stories belonging to each topics and the confidence
#   of the belonging
tm_gamma %>%
  ggplot(aes(gamma, fill = as.factor(topic))) +
  geom_histogram(show.legend = FALSE) +
  facet_wrap(~topic, ncol = 3)
```

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




```
# Visualizing how much each topic appear in each story
tm_gamma %>%
  ggplot(aes(topic, gamma, fill = document)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~document, scales = "free", ncol = 3) +
  theme(strip.text.x = element_text(size = 5))
```



The model did an excellent job strongly associating the stories into one or more topics. This perfect association is rare in the world of topic modeling. The reason behind this perfect association here could be due to the small number of documents that we have.

9.9 References

- Adventures of Sherlock Holmes book by Arthur Conan Doyle on Project Gutenberg
- Regular Expressions 101