R for Data Analytics Part 1, Lecture 1

Michèle Fille

Table of contents

Lecture 1 - Basics and Plotting

1.1 A Quick Tour of R and RStudio

Exercise 1.1. The Basics

Exercise 1.1 – Task 1: Magic with numbers

Execute in the Console window the following mathematical operations. If you execute everything correctly, you should end up with the same number that you started with!

1. Choose any number and add 2 to it.
2. Multiply the result by 3.
3. Subtract 6 from the answer.
4. Divide what you get by 3.

```
step1 <- 7 + 2      # = 9
step2 <- step1 * 3  # = 27
step3 <- step2 - 6  # = 21
step4 <- step3 / 3  # = 7
step4
```

```
[1] 7
```

```
#or
(((7+2)*3)-6)/3
```

```
[1] 7
```

Exercise 1.1 – Task 2: Create an R-skript

- Create a new **folder** „Lecture_1“ on your computer.
- Create a new **R-project** in that folder.
- Create as new **R-skript** in that project.
- Recreate the **display** on the right (see: Figure 1 and for solution Figure 2.

Note:

- Don‘t forget to install the ggplot2 package before loading it.
- The operator <- assigns the plot output to a variable (mpg_plot).
- The data set mpg is automatically loaded together with ggplot2. Use ?mpg to see what it is about.

Note:

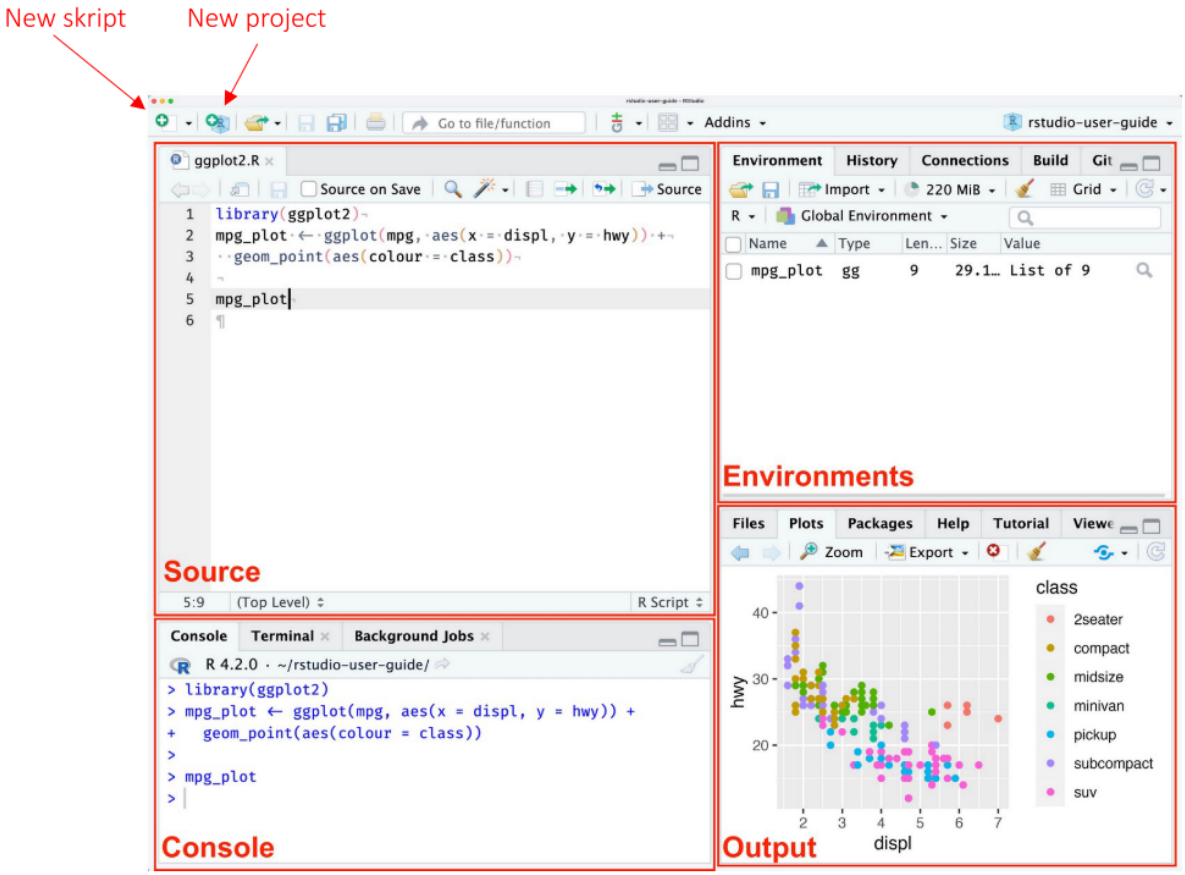


Figure 1: Desired Output

- To execute a command in the Source pane, press Strg+Enter (Windows) or command+Enter (Mac).

Remark:

- It is helpful to make a new R Project in a new folder for each major project you start in on.

```
# install.packages("ggplot2") # already installed
library(ggplot2)
mpg_plot <- ggplot(mpg, aes(x = displ, y = hwy)) + geom_point(aes(colour = class))

mpg_plot
```

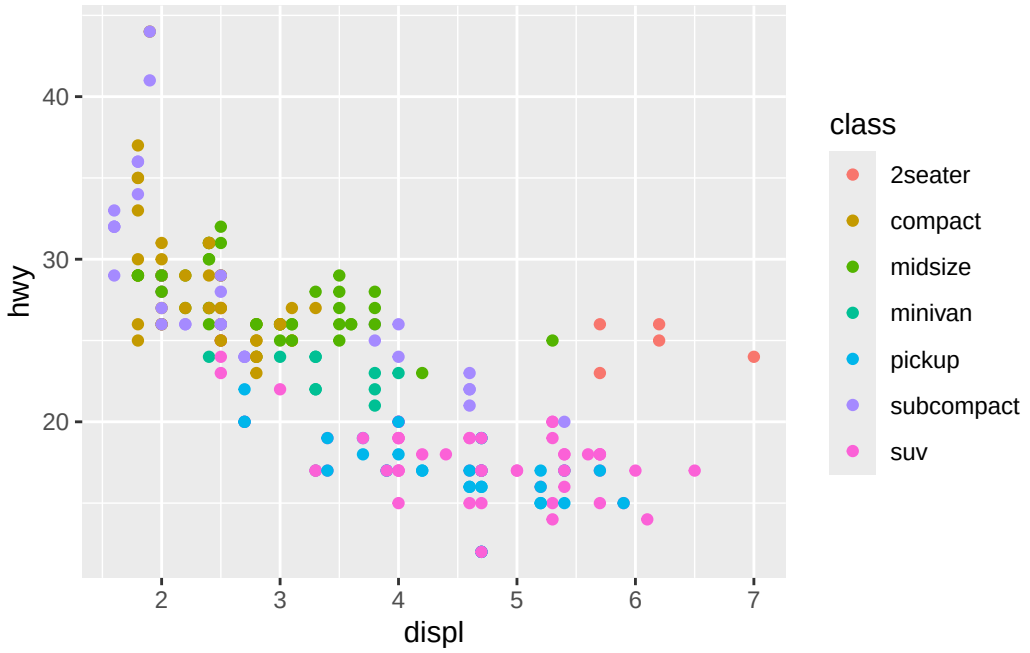


Figure 2: Scatterplot of Fuel economy data from 1999 to 2008 for 38 popular models of cars

Exercise 1.1 – Task 3: Load a csv file with base R

1. Install and load the libraries readr and data.table.
2. Download the dataset **female_names.csv** from Moodle.

3. Load the data set into R using *read_csv()* from base R. **Store it in a variable fnames**. To do that, use **the assignment operator <-** : *fnames <- read.csv("female_names.csv")*
4. To get a first impression of the data set, type **View(fnames)**. Alternatively, you can click on the variable name in the Environments pane.
5. Now load it again,

- using *read.csv()* from package readr, and store it in the variable fnames1.
- using *fread()*, from from package data.table, and store it in the variable fnames2.

Do you experience a difference in performance?

```
#install.packages("readr") # already installed
#install.packages("data.table") # already installed
library(readr)
library(data.table)

fnames <- read_csv("female_names.csv")
```

```
New names:
Rows: 731574 Columns: 5
-- Column specification
-------------------------------------------------------- Delimiter: "," chr
(1): name dbl (4): ...1, X, prop, year
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.
* `` -> `...1`
```

```
View(fnames)
fnames1 <- read.csv("female_names.csv")
fnames2 <- fread("female_names.csv")
```

Exercise 1.1 – Task 4: Work with Data Types

What is the data type of each of the following objects. If you don't know, check the type using *typeof()*.

- 1
- 5L

- sqrt(-1 +0i)
- “25”

```
typeof(1)  # double
```

```
[1] "double"
```

```
typeof(5L) # integer
```

```
[1] "integer"
```

```
typeof(sqrt(-1+0i)) #complex
```

```
[1] "complex"
```

```
typeof("25") # character
```

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

Which of these coercions work? What do they produce?

- as.numeric(FALSE)
- as.logical(“Hello!”)
- as.integer(4.4)
- as.character(TRUE)
- as.numeric(“scooby snacks”)

```
as.numeric(FALSE) # 0
```

```
[1] 0
```

```
as.logical("Hello!") # NA
```

```
[1] NA
```

```r
as.integer(4.4) # 4
```

```
[1] 4
```

```r
as.character(TRUE) # "TRUE"
```

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

```r
as.numeric("scooby snacks") # NA
```

```
Warning: NAs durch Umwandlung erzeugt
```

```
[1] NA
```

What is the output of each of these lines of code? Why?

- sqrt(-1)
- “1” + “5”
- 1 + 7
- TRUE + FALSE

```r
sqrt(-1) # NaN (Not a Number -> is a complex number, to make it clear sqrt(-1+0i))
```

```
Warning in sqrt(-1): NaNs wurden erzeugt
```

```
[1] NaN
```

```r
"1" + "5" # Error: strings cannot be conncatenated with +
```

```
Error in "1" + "5": nicht-numerisches Argument für binären Operator
```

```r
1 + 7 # 8
```

```
[1] 8
```

```r
TRUE + FALSE # 1, cz logic can be coerced (gezwungen) to numeric
```

```
[1] 1
```

Exercise 1.1 – Task 5: Work with Data Structures

Create a vector of integers of length 20 and store it in the variable v.

```r
v <- 1:20
v
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Use your vector v to create a matrix of integers with dimensions 4x5 and store it in the variable m.

```r
m <- matrix(v,nrow = 4)
m
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Coerce your matrix m into a data frame and store the result in the variable df.

```r
df <- as.data.frame(m)
df
```

```
  V1 V2 V3 V4 V5
1  1  5  9 13 17
2  2  6 10 14 18
3  3  7 11 15 19
4  4  8 12 16 20
```

Create a list called my_list with the following 4 elements:

- "I love summer"

- TRUE
- “fun temperatures”
- c(24,25,26)

```
my_list <- list("I love summer", TRUE, "fun temperatures", c(24,25,26))
my_list
```

```
[[1]]
[1] "I love summer"

[[2]]
[1] TRUE

[[3]]
[1] "fun temperatures"

[[4]]
[1] 24 25 26
```

Create a data frame with the following columns (vectors) and save it in the variable my_df:

- Your name and the name of your 2 best friends.
- Your age and your 2 best friends’ ages.
- Whether you and each of your 2 friends own a car or not (as Boolean values).

```
names <- c("Michele", "Pelin", "Selina")
ages <- c(28,29,28)
cars <- c(TRUE, TRUE, TRUE)

my_df <- data.frame(names, ages, cars)
my_df
```

```
   names ages cars
1 Michele   28 TRUE
2   Pelin   29 TRUE
3  Selina   28 TRUE
```

```r
# or

my_df1 <- data.frame(names = c("Michele", "Noe", "Pariya"),
                     ages = c(28, 22, 26),
                     car_owner = c(TRUE, FALSE, TRUE))
my_df1
```

```
    names ages car_owner
1 Michele   28      TRUE
2     Noe   22     FALSE
3  Pariya   26      TRUE
```

Save your data frame my_df to your disk

- as a csv file,
- as a RDS file.

Then load it again, using a different name.

```r
fwrite(my_df, file = "bffs.csv")
saveRDS(my_df, file = "bffs.RDS")
bffs1 <- fread("bffs.csv")
bffs2 <- readRDS("bffs.RDS")
```

Save my_list and my_df as an Rdata file. Then remove both variables from your environment and load the saved RData fiel again.

```r
save(my_list, my_df, file = "my_stuff.Rdata")
rm(my_list) # rm() removes objects from the environment.
rm(my_df)
load("my_stuff.Rdata")
```

Notice: Since more than one object is stored in my_stuff.Rdata, I did not assign the output to a variable (like I did with readRDS above).

Instead, the objects are automatically loaded under the original variable names. (Check the environment pane: my_list and my_df are there again.)

This can accidentally overwrite an object name that is already in your envirinment - especially, when you don't know or don't remember what is inside the Rdata file.

Thus, it is always safer to use saveRDS(), saving all objects separately.

You can check what happens if you assign the output of load() to a varible:

```
my_stuff <- load("my_stuff.Rdata")
typeof(my_stuff)
```

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

The names of the included objects are stored in a character vector, but the objects themselves are not loaded.

Self-Study 1.1. The Basics

Which one is larger: sin(10) or cos(10)? Write an expression to solve this.

```
sin(10) # -0.55 <- larger
```

```
[1] -0.5440211
```

```
cos(10) # -084
```

```
[1] -0.8390715
```

```
sin(10) > cos(10) # TRUE
```

```
[1] TRUE
```

Do the following:

- Create a variable ‘puppies‘ equal to the number of puppies you’d like to have
- Create a variable ‘puppy_price‘, which is how much you think a puppy costs
- Create a variable ‘total_cost‘ that has the total cost of all of your puppies
- Create a boolean variable ‘too_expensive‘, set to TRUE if the cost is greater than $1,000
- Create a variable ‘max_puppies‘, which is the number of puppies you can afford for $1,000

```r
puppies <- 3
puppy_price <- 450
total_cost <- puppies * puppy_price
total_cost
```

```
[1] 1350
```

```r
too_expensive <- total_cost > 1000
too_expensive
```

```
[1] TRUE
```

```r
max_puppies <- as.integer(1000 / puppy_price)
max_puppies
```

```
[1] 2
```

What is the data type of each of the following objects:

- c(1, 2, 3)
- c('d', 'e', 'f')
- c("d", "e", "f")
- c(TRUE,1L,10)
- c("11",10,12)
- c("Sun","night", FALSE)

–> They are all vector objects with elements/values, the type of these elemensts inside the vector can be checked with the typeof() function.

```r
typeof(c(1, 2, 3))
```

```
[1] "double"
```

```r
typeof(c('d', 'e', 'f'))
```

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

```r
typeof(c("d", "e", "f"))
```

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

```r
typeof(c(TRUE,1L,10))
```

```
[1] "double"
```

```r
typeof(c("11",10,12))
```

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

```r
typeof(c("Sun","night", FALSE))
```

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

To answer the following questions, use your preferred search engine and/or the inbuilt help:

1. Find out what the abs() function does using the inbuilt help. How much is abs(10)?
2. What is the square root of 11? Is there a function for this in R?
3. How do you round numbers to the nearest integer in R? Is there a function? Round 3.5 to the nearest integer.
4. Create a vector that ranges from 10 to 50 in steps of 3.

```r
?abs() # abs(x) computes the absolute value of x
```

```
starte den http Server für die Hilfe fertig
```

```r
abs(10) # 10
```

```
[1] 10
```

```r
abs(-10) # 10
```

```
[1] 10
```

```
  sqrt(11) # 3.32
```

```
[1] 3.316625
```

```
  round(3.5)
```

```
[1] 4
```

```
  round(sqrt(11), digits=2) # round to a specific number of decimal places
```

```
[1] 3.32
```

```
  seq(from = 10, to = 50, by = 3)
```

```
 [1] 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

```
  seq(10, 50, by = 3)
```

```
 [1] 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

5. Install the package "RXKCD". What can you do with this package?

   The getXKCD() functions give back an image see Figure 3

```
  # install.packages("RXKCD") # already installed
  library("RXKCD")
  ?RXKCD
```

```
Keine Dokumentation für 'RXKCD' in angegebenen Paketen und Bibliotheken:
Sie können '??RXKCD' versuchen
```

```
  ??RXKCD # Visualize your favorite XKCD comic strip directly from R. XKCD
  getXKCD(which = "current", display = TRUE, html = FALSE, saveImg = FALSE)
```

```
$month
[1] "4"

$num
[1] 2921

$link
[1] ""
```

```
$year
[1] "2024"

$news
[1] ""

$safe_title
[1] "Eclipse Path Maps"

$transcript
[1] ""

$alt
[1] "Okay, this eclipse will only be visible from the Arctic in February 2063, when the

$img
[1] "https://imgs.xkcd.com/comics/eclipse_path_maps.png"

$title
[1] "Eclipse Path Maps"

$day
[1] "17"

attr(,"class")
[1] "rxkcd"
```

Figure 3: Image of favorite XKCD comic strip

1.2 Data Viz with ggplot2 and plotly

Exercise 1.2. Data Viz with ggplot2 and plotly

Exercise 1.2 – Task 1: Create a basic Scatterplot

Install and load the libraries “tidyverse”, “ggplot2”, “plotly” and “palmerpenguins”.

View the data using View(penguins).

Read the help page of the penguins dataset using help(penguins). It contains a data dictionary explaining attribute semantics.

Type glimpse(penguins) and summary(penguins) and inspect the output.

- glimpse() shows you all the attributes, their data types and some example values. glimpse() is part of the dyplyr package, which in turn is part of the tidyverse collection of packages.
- summary() is a generic function. Applied to a data set, it shows you summary statistics of all attributes.

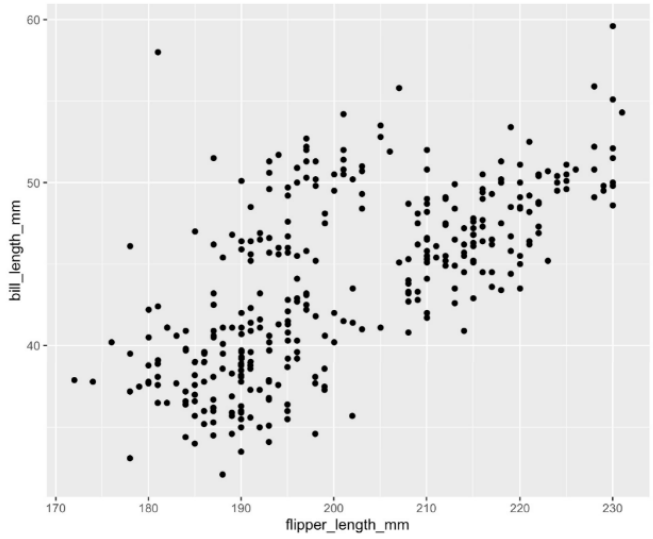


Figure 4: Desired Scatterplot

Plot all pairs of variables against each other using ggpairs().

Recreate the scatterplot on the right (see Figure 4 and for solution Figure 5a and Figure 5b):

- Start with an empty canvas using ggplot() and specify the data set.
- Then add an aestetic mapping using mapping() and aes().
- In aes(), map flipper_length_mm to the x-axes and bill_length_mm to the y axes.
- Add a layer geom_point() to visualize the corresponding data as a scatterplot.

```
# install.packages("tidyverse") # already installed
# install.packages("ggplot2") # already installed
# install.packages("plotly") # already installed
# install.packages("palmerpenguins") # already installed

# install.packages("GGally") # already isntalled # needed for the ggpairs()
library("GGally") # needed for the ggpairs()

library("tidyverse")
library("ggplot2")
library("plotly")
library("palmerpenguins")

View(penguins)
```

```r
help("penguins")
```

```r
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~
```

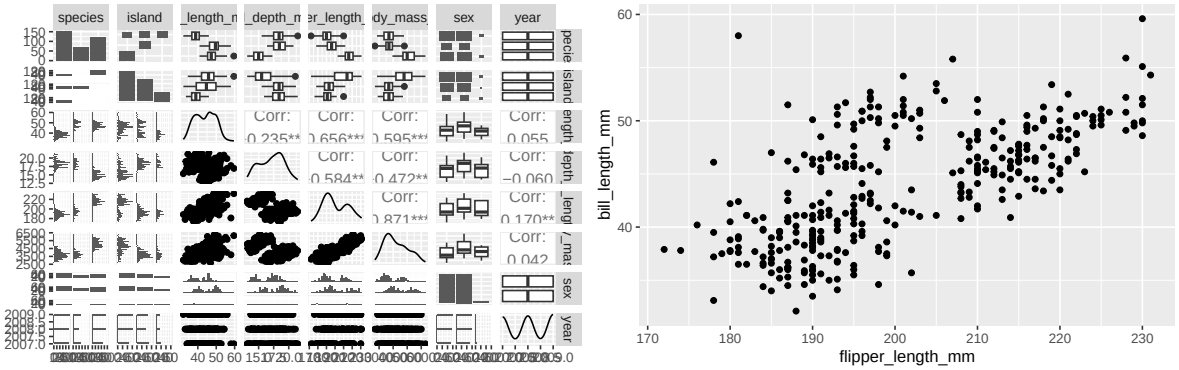
```r
summary(penguins)
```

```
      species          island    bill_length_mm  bill_depth_mm
 Adelie   :152   Biscoe   :168   Min.   :32.10   Min.   :13.10
 Chinstrap: 68   Dream    :124   1st Qu.:39.23   1st Qu.:15.60
 Gentoo   :124   Torgersen: 52   Median :44.45   Median :17.30
                                 Mean   :43.92   Mean   :17.15
                                 3rd Qu.:48.50   3rd Qu.:18.70
                                 Max.   :59.60   Max.   :21.50
                                 NA's   :2       NA's   :2
 flipper_length_mm  body_mass_g       sex           year
 Min.   :172.0     Min.   :2700   female:165   Min.   :2007
 1st Qu.:190.0     1st Qu.:3550   male  :168   1st Qu.:2007
 Median :197.0     Median :4050   NA's  : 11   Median :2008
 Mean   :200.9     Mean   :4202                Mean   :2008
 3rd Qu.:213.0     3rd Qu.:4750                3rd Qu.:2009
 Max.   :231.0     Max.   :6300                Max.   :2009
 NA's   :2         NA's   :2
```

```r
ggpairs(penguins)
```

```r
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm,
```

```
                    y = bill_length_mm)) +
  geom_point()
```



(a) Penguins ggpairs

(b) Penguins Scatterplot

Figure 5: Penguins

Exercise 1.2 – Task 2: Pimp Up your Scatterplot

Add the variable island to your plot by mapping it to the color channel.

Add another layer labs() to your plot and use it to add the title "Flipper Length vs. Bill Length (in mm) by Island". To see all arguments of the labs() function, type help(labs).

Use labs() to add the subtitle "Source: Palmer Station LTER / palmerpenguins package"

Use labs() to change the labels of the x and y axes to "Flipper Length (mm)" and "Bill Length (mm)", respectively.

Use labs() to change the legend title to "Island".

Now your plot should look like the one on the right! (see Figure 6 and for solution Figure 7)

```
# install.packages("tidyverse") # already installed
# install.packages("ggplot2") # already installed
# install.packages("plotly") # already installed
# install.packages("palmerpenguins") # already installed
library("tidyverse")
library("ggplot2")
library("plotly")
library("palmerpenguins")

ggplot(data = penguins,
```

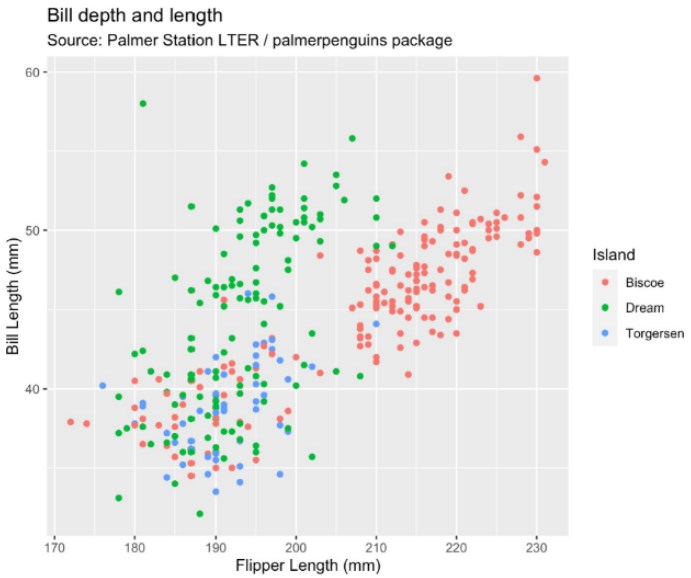


Figure 6: Desired Scatterplot (pimed)

```
     mapping = aes(x = flipper_length_mm,
                   y = bill_length_mm,
                   color = island)) +
geom_point() +
labs(title = "Flipper Length vs. Bill Length (in mm) by Island",
     subtitle = "Source: Palmer Station LTER / palmerpenguins package",
        x = "Flipper Length (mm)", y = "Bill Length (mm)",
     color = "Island") # Set legend title for color scale
```

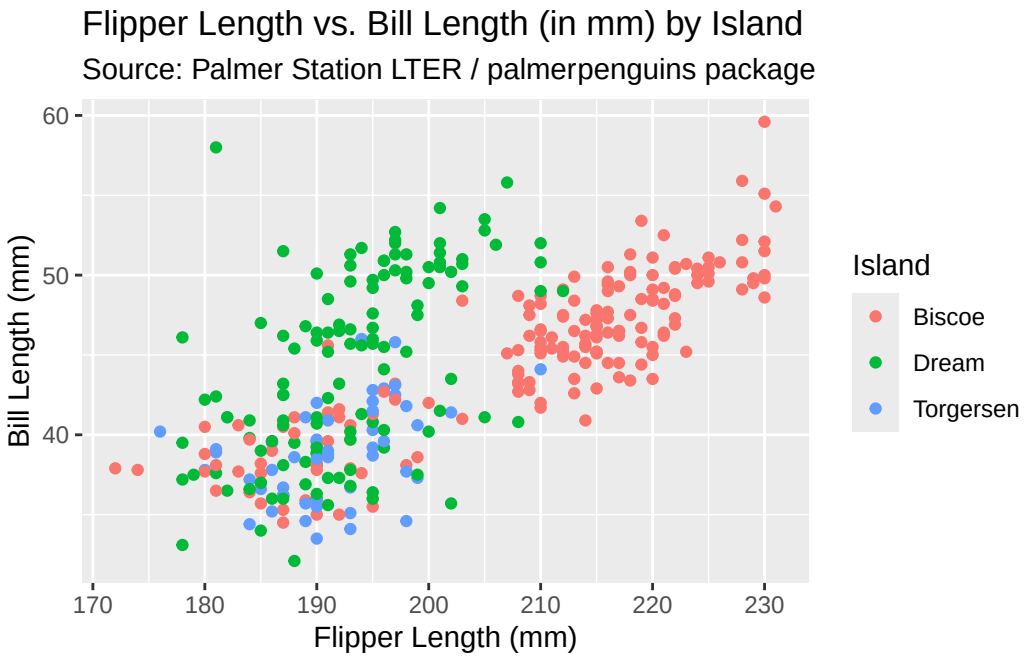


Figure 7: Penguins pimped Scatterplot

Exercise 1.2 – Task 3: Add a Smoothed Trendline

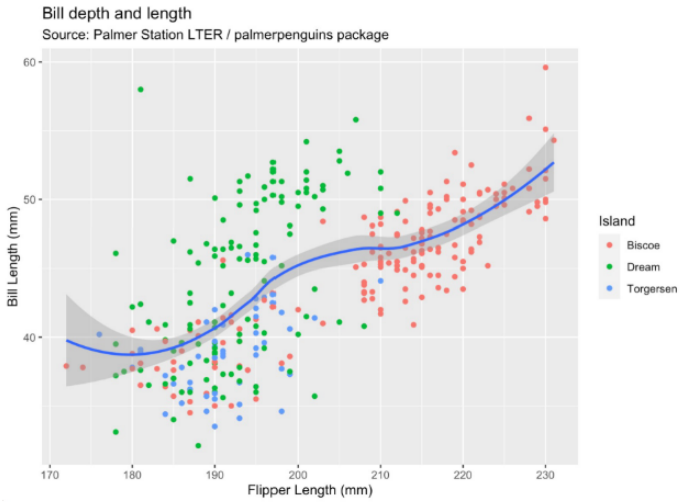


Figure 8: Desired Scatterplot with trendline

Now add a layer geom_smooth to visualize a smoothed trendline. What happens here?

- Answer: 3 trendlines get shown, as the variables island is mapped to the color scale

Change your plot so that only one trendline is displayed.

Make it interactive using ggplotly().

- Remark: To do that, assign your whole plot from (b) to a variable, e.g., the variable my_plot. Then call ggploty(my_plot).

Now your plot should look like the one on the right! (see Figure 8 and for solution Figure 9)

```
# install.packages("tidyverse") # already installed
# install.packages("ggplot2") # already installed
# install.packages("plotly") # already installed
# install.packages("palmerpenguins") # already installed
library("tidyverse")
library("ggplot2")
library("plotly")
library("palmerpenguins")

my_plot <- ggplot(data = penguins,
                  mapping = aes(x = flipper_length_mm,
                                y = bill_length_mm)) +
  geom_point(mapping = aes(color = island)) + # mapping for the color scale to the "point"
  geom_smooth() +
  labs(title = "Flipper Length vs. Bill Length (in mm) by Island",
       subtitle = "Source: Palmer Station LTER / palmerpenguins package",
       x = "Flipper Length (mm)", y = "Bill Length (mm)",
       color = "Island") # Set legend title for color scale

my_plot # not interactive

#ggplotly(my_plot) # does not work as pdf output
```

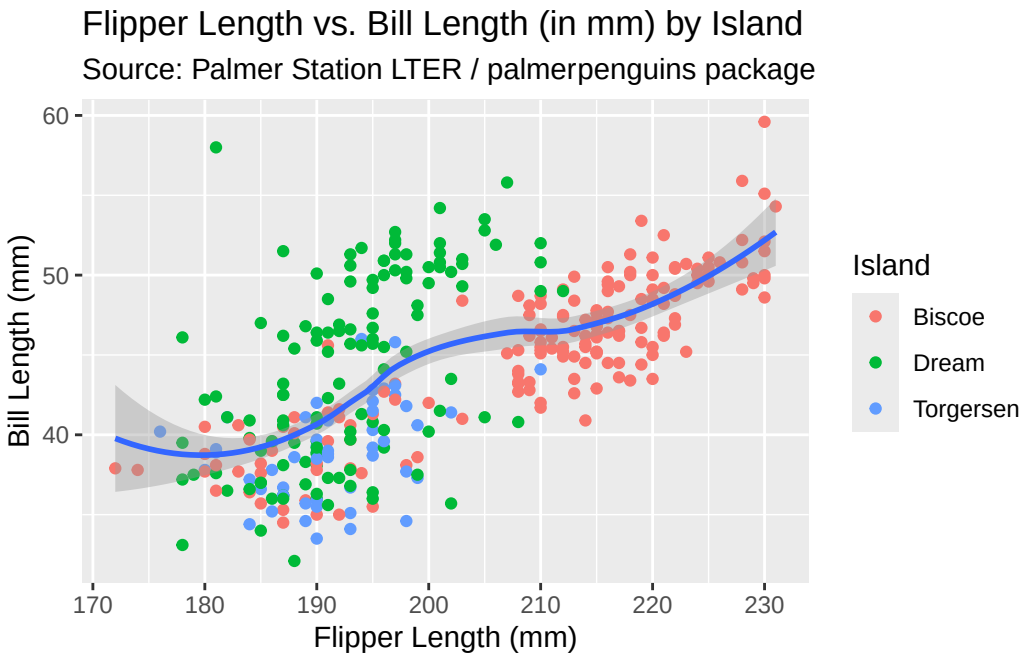


Figure 9: Penguins Scatterplot with trendline

Exercise 1.2 – Task 4: Create a Stacked Bar Plot

Create a new canvas for penguins.

Create a bar plot of the variable species using geom_bar.

Use facet_wrap() to create a sequence of facets along the variable island.

In facet_wrap(), add the argument ncol=1. (See help(facet_wrap) for more arguments!)

In aes(), add the position argument fill = sex. This gives you stacked bar plots according tp sex.

- Remark: A position adjustment for a geometry (like geom_bar here) specifies a "rule" as to how different components should be positioned relative to each other to make sure they don't overlap. This position adjustment is inherent in geom_bar, and we can make it visible by mapping a different variable to the color encoding (using the fill aesthetic here).

Now your plot should look like the one on the right! (see Figure 10 and for solution Figure 11)

```
ggplot(data=penguins) +
  geom_bar(mapping = aes(x=species, fill=sex)) +
```

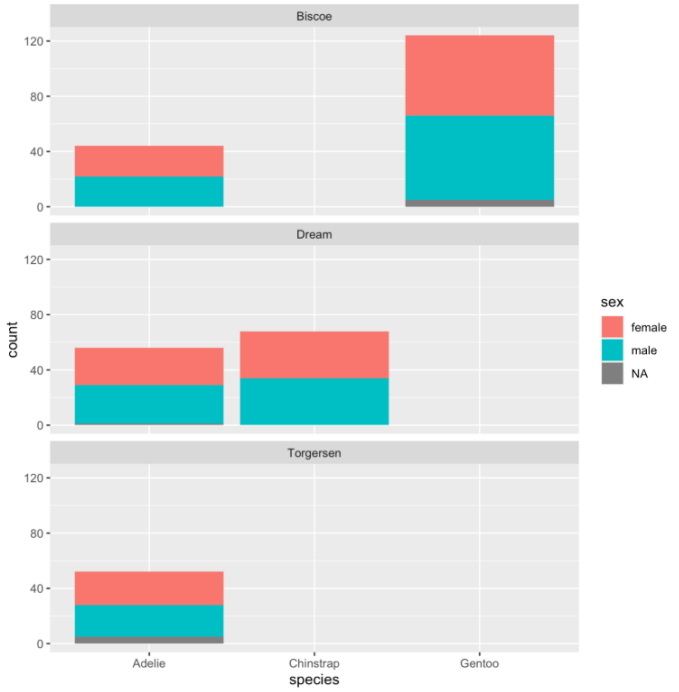


Figure 10: Desired Output

```
  facet_wrap(facets=~island,
             ncol=1)
```

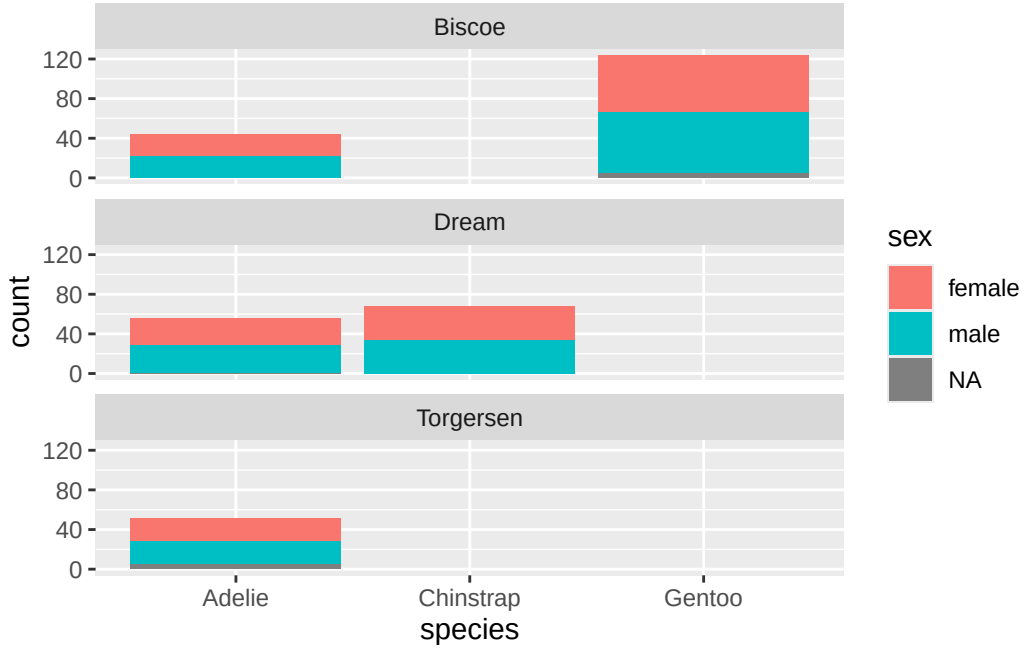


Figure 11: Penguins Barchart per islands

Self-Study 1.2. Data Viz with ggplot2 and plotly

Self Study 1.2 – Task 1: Business Understanding

1. Install and load the libraries "tidyverse", "ggplot2", "plotly", "skimr" and "Ggally"
2. The data set mpg is included in the ggplot2 package. Read the help page of the dataset using help(mpg). It contains a data dictionary.
3. Try to understand what the attribute names mean.
4. View the data using View(mpg).

```
# install.packages("tidyverse") # already installed
# install.packages("ggplot2") # already installed
# install.packages("plotly") # already installed
# install.packages("skimr") # already installed
# install.packages("GGally") # already installed
```

```
library("tidyverse")
library("ggplot2")
library("plotly")
library("skimr")
library("GGally")

help(mpg)

View(mpg)
```

Self Study 1.2 – Task 2: Data Understanding based on Summary Statistics

Use the functions class(), typeof(), attributes() and dim() to check the class, data type, attributes and dimensions of the mpg dataset. Notice that mpg is loaded as a table (tbl) - a special kind of data frame (see later).

Get a first impression of the structure of the data by calling glimpse(mpg) and summary(mpg)

Now try the skim() function from the skimr library to get a better overview over the dataset. Look into the help page of skim(). (see Figure 13 for the output)

Now answer the following questions:

- Data of how many cars are stored in the data set?

  Solution: 234 (number of rows)

- How many car attributes have been recorded in the mpg data set?

  Solution: 11 (number of columns)

- How many of them are numerical, how many categorical?

  Solution: 5 numeric (numerical) and 6 character (categorical)

- Has the attribute type of drive train been recorded for all cars in this data set, or are there any missing values for this attribute?

  Solution: there are 0 missing values

- How many car models does this data set contain?

  Solution: there are 38 car models (n_unique)

- How many manufacturers?

  Solution: there are 15 manufacturers (n_unique)

```r
class(mpg)
```

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

```r
typeof(mpg)
```

```
[1] "list"
```

```r
attributes(mpg) # gives: class, row.names, names(headers)
```

```
$class
[1] "tbl_df"     "tbl"        "data.frame"

$row.names
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
[163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
[181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
[199] 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
[217] 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234

$names
 [1] "manufacturer" "model"        "displ"        "year"         "cyl"
 [6] "trans"        "drv"          "cty"          "hwy"          "fl"
[11] "class"
```

```r
dim(mpg) # dimension: rows x columns
```

```
[1] 234  11
```

```r
glimpse(mpg)
```

```
Rows: 234
Columns: 11
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
$ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)", "auto~
$ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4", "4", "4~
$ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17, 17, 1~
$ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25, 2~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
$ class        <chr> "compact", "compact", "compact", "compact", "compact", "c~
```

```r
summary(mpg)
```

```
 manufacturer          model               displ            year     
 Length:234         Length:234         Min.   :1.600   Min.   :1999  
 Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999  
 Mode  :character   Mode  :character   Median :3.300   Median :2004  
                                       Mean   :3.472   Mean   :2004  
                                       3rd Qu.:4.600   3rd Qu.:2008  
                                       Max.   :7.000   Max.   :2008  
      cyl           trans               drv                 cty       
 Min.   :4.000   Length:234         Length:234         Min.   : 9.00  
 1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00  
 Median :6.000   Mode  :character   Mode  :character   Median :17.00  
 Mean   :5.889                                         Mean   :16.86  
 3rd Qu.:8.000                                         3rd Qu.:19.00  
 Max.   :8.000                                         Max.   :35.00  
      hwy             fl                 class          
 Min.   :12.00   Length:234         Length:234         
 1st Qu.:18.00   Class :character   Class :character   
 Median :24.00   Mode  :character   Mode  :character   
 Mean   :23.44                                         
 3rd Qu.:27.00                                         
 Max.   :44.00                                         
```

```
skim(mpg)
```

Table 1: Data summary

Name	mpg
Number of rows	234
Number of columns	11
_____	
Column type frequency:	
character	6
numeric	5
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
manufacturer	0	1	4	10	0	15	0
model	0	1	2	22	0	38	0
trans	0	1	8	10	0	10	0
drv	0	1	1	1	0	3	0
fl	0	1	1	1	0	5	0
class	0	1	3	10	0	7	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
displ	0	1	3.47	1.29	1.6	2.4	3.3	4.6	7	
year	0	1	2003.50	4.51	1999.0	1999.0	2003.5	2008.0	2008	
cyl	0	1	5.89	1.61	4.0	4.0	6.0	8.0	8	
cty	0	1	16.86	4.26	9.0	14.0	17.0	19.0	35	
hwy	0	1	23.44	5.95	12.0	18.0	24.0	27.0	44	

- What is the minimum number of cylinders that occur in a car of this data set?

  Solution: 4 (in skim the min appears under "p0" - the zero percentile)

- What is the range of highway miles per gallon of the cars in this data set? (range = max-min)

  Solution: 12 - 44 = 32 (p0 - p100)

- What is the mean and standard deviation of highway miles per gallon of the cars in this data set?

  Solution: mean= 23.44, sd =5.95

- In the character variables section, what do you think do the column names min, max and empty stand for?

  – Hint: Execute unique(mpg$manufacturer). Does it give you an idea?

  Solution: ***min = 4*** *is the minimum number of characters of any manufacturer (audi, ford, jeep),* ***max = 10*** *is the maximum number of characters of any manufacturer (volkswagen, land rover),* ***empty = 0*** *says that no character strings are empty*

- Is the variable cty normally distributed? Is it skewed? (Read this off the output of skim.)

  Solution: Skewness is a measure of the asymmetry of the probability distribution about its mean. The little picture displayed by skim() suggests that cty is right skewed, since it shows a long tail on the right. Yet, the mean is a bit left of the median (p50). So it is not clear from the data we see here.

- Now plot the histogram of cty to double check the sample distribution. Do you think it makes sense to check for outliers?

  Solution: We see also here no clear picture: The mass of the distribution may actually be equally distributed about the mean of 16.9. The little picture may have given a wrong impression because of the 2 outliers on the right. As a next step, we could remove the outliers and then check the distribution again. But first lets check if these are indeed outliers or not… (We will learn in a later lecture how to remove them, if necessary.)

- Create a boxplot of the variable cty. Make your boxplot interactive using ggplotly(). How many outliers do you see (according to the interquartile range (IQR) criterion)?

  – Hint: In the boxplot, outliers according to the IQR criterion are all points outside of the whiskers. (see Figure 12)

  Solution: There are 4 outliers visible in the boxplot: The two we recognized in the histogram (33 and 35: double check wit the histogram), and 2 more (28 and 29).

- You can check if you are right using the function boxplot.stats(): Executing boxplot.stats(mpg$cty)$out will give you all outliers. Does it fit? (Remember that the $ sign lets you access the single attributes of a data frame.)

  Solution: No, it doesnt fit full: We have actually 5 outliers, because 28 appears twice. We couldn't see that in the boxplot, since they are drawn on top of each other.

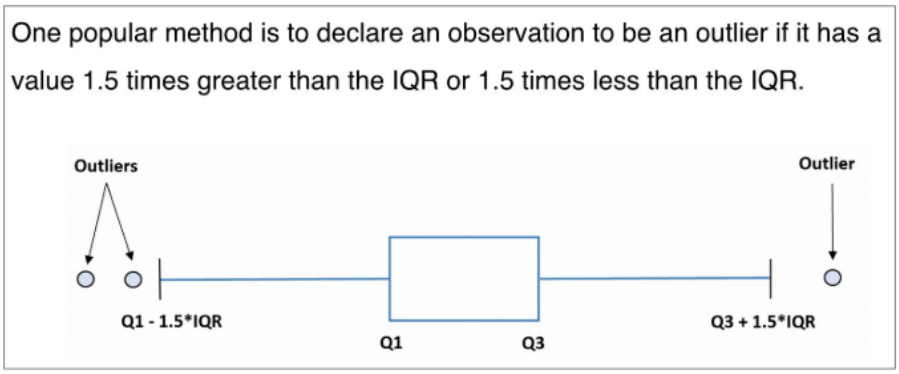


Figure 12: The IQR criterion.

```
unique(mpg$manufacturer)
my_hist_cty <- ggplot(data = mpg) +
  geom_histogram(mapping = aes(x=cty))
my_hist_cty # not interactive
```

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

```
#ggplotly(my_hist_cty) # does not work as pdf output

my_boxplot_cty <- ggplot(data = mpg) +
  geom_boxplot(mapping = aes(y=cty))
my_boxplot_cty # not interactive
#ggplotly(my_boxplot_cty) # does not work as pdf output

boxplot.stats(mpg$cty)$out
```

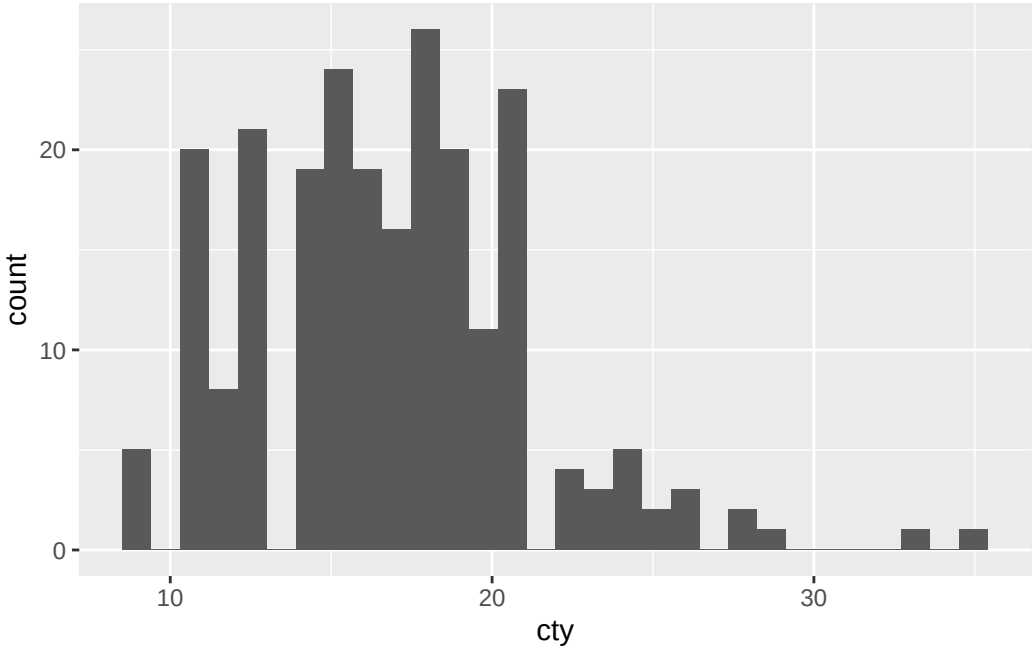
Self Study 1.2 – Task 3: Data Understanding by Visual EDA

Now let's do a bit of Exploratory Data Analysis (EDA) using ggplot!

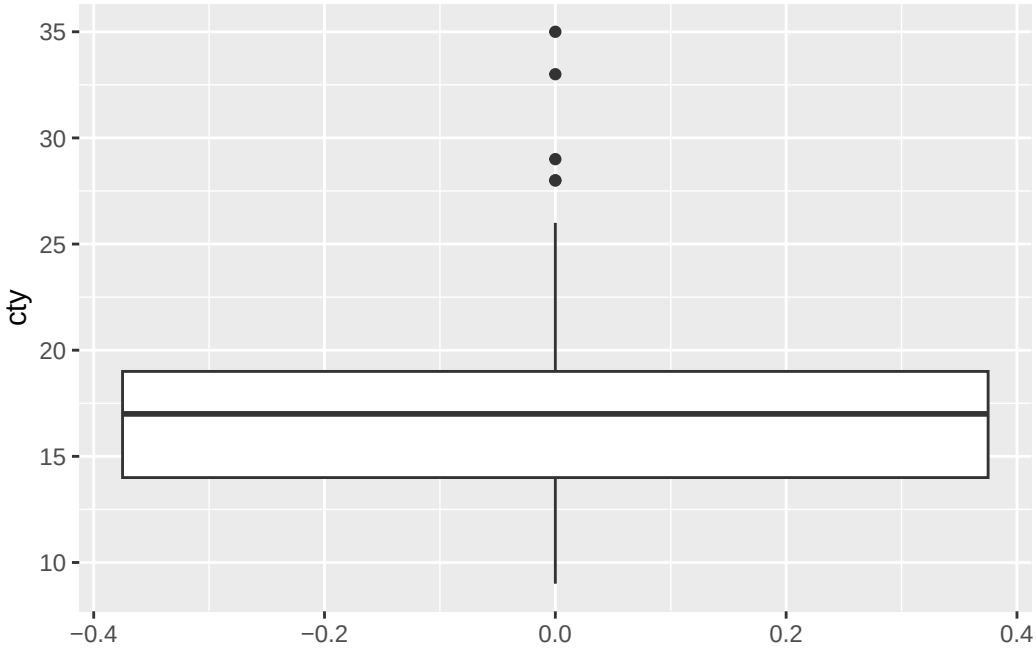
I assume that the number of highway miles a car can go per gallon may depend functionally on the engine displacement of the car (e.g., linearly, quadratically,...). Do you think our sample data will support my hypothesis? Check by visual inspection.

- Hint: For the visual inspection, make a scatterplot of hwy against displ.

```
 [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"
 [6] "hyundai"    "jeep"       "land rover" "lincoln"    "mercury"
[11] "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
[1] 28 28 33 35 29
```



(a) histogram of cty



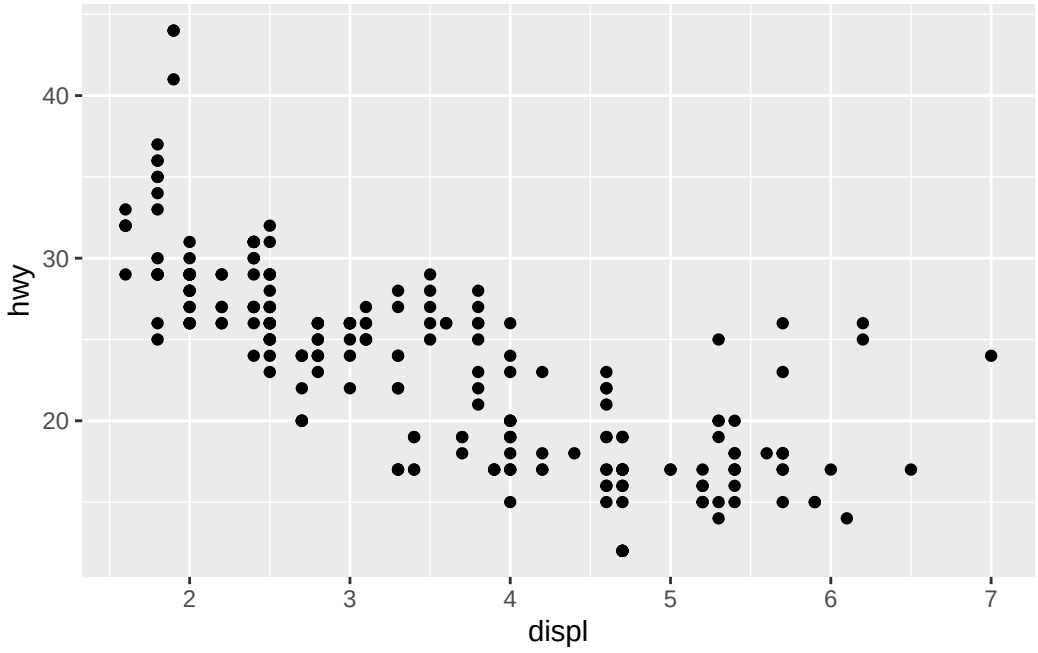
(b) outliers

Figure 13: MPG data

- Remark: To really check the hypothesis, we would need to do more: We could do a statistical hypothesis test, or we could model a functional relationship (as part of the modelling phase of the data science lifecycle) and evaluate the modelling results. The step of visual inspection is a way of scouting if its worth the effort, and to give us ideas. :)

Solution: It looks as if the hypothesis is correct. There is a clear pattern discernible.

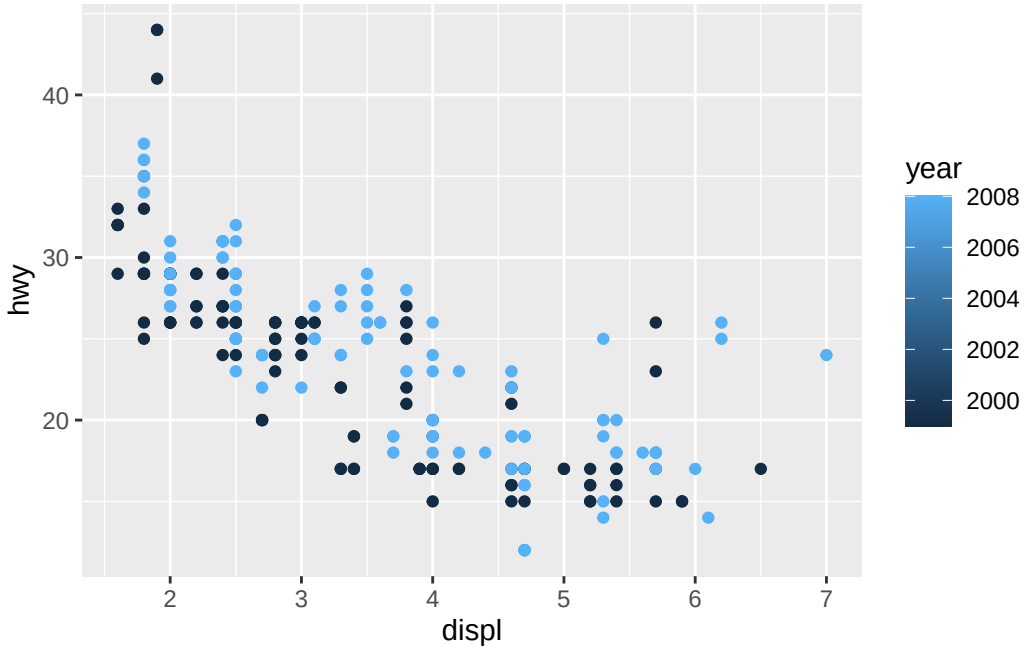
```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy)) +
  geom_point()
```



Add the variable year to your plot using color as a third dimension. What is going on with the legend here?

Solution: The legend shows a continuous scale, even though we actually only have 2 values of the year variable (1999 and 2008 - check with unique(mpg$year)). The reason is that year has been imported as a numeric data type (check with skim(mpg) or str(mpg)), and this implies a continuous color scale.

```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     color=year)) +
  geom_point()
```




```
unique(mpg$year)
```

```
[1] 1999 2008
```

```
skim(mpg)
```

Table 4: Data summary

Name	mpg
Number of rows	234
Number of columns	11
Column type frequency:	

character	6
numeric	5
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
manufacturer	0	1	4	10	0	15	0
model	0	1	2	22	0	38	0
trans	0	1	8	10	0	10	0
drv	0	1	1	1	0	3	0
fl	0	1	1	1	0	5	0
class	0	1	3	10	0	7	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
displ	0	1	3.47	1.29	1.6	2.4	3.3	4.6	7	
year	0	1	2003.50	4.51	1999.0	1999.0	2003.5	2008.0	2008	
cyl	0	1	5.89	1.61	4.0	4.0	6.0	8.0	8	
cty	0	1	16.86	4.26	9.0	14.0	17.0	19.0	35	
hwy	0	1	23.44	5.95	12.0	18.0	24.0	27.0	44	

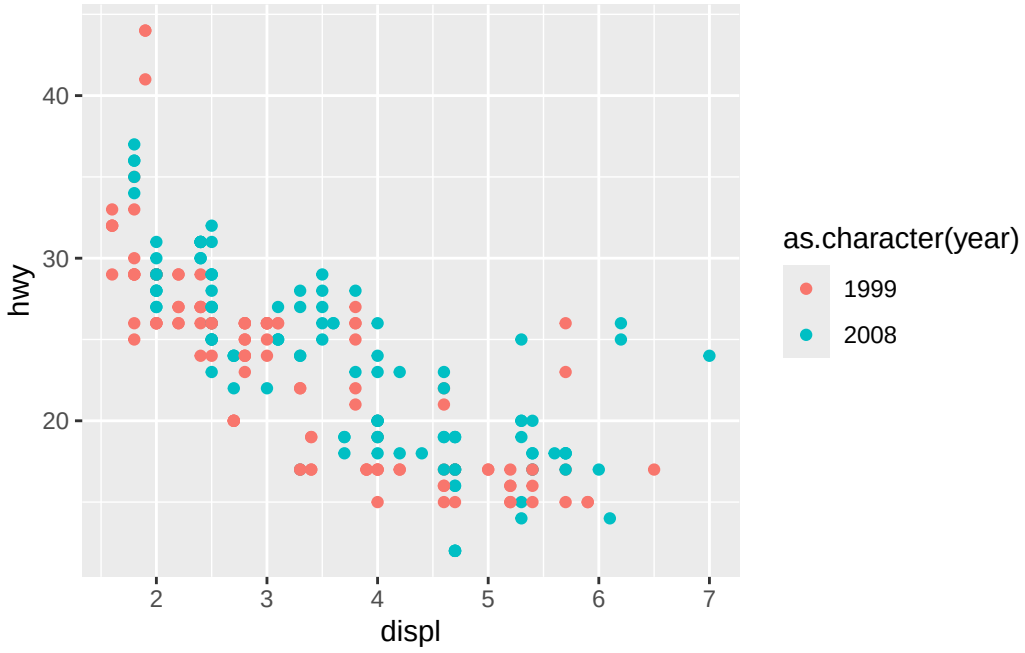
```
str(mpg)
```

```
tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
 $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
 $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
 $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv         : chr [1:234] "f" "f" "f" "f" ...
 $ cty         : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : chr [1:234] "p" "p" "p" "p" ...
 $ class       : chr [1:234] "compact" "compact" "compact" "compact" ...
```

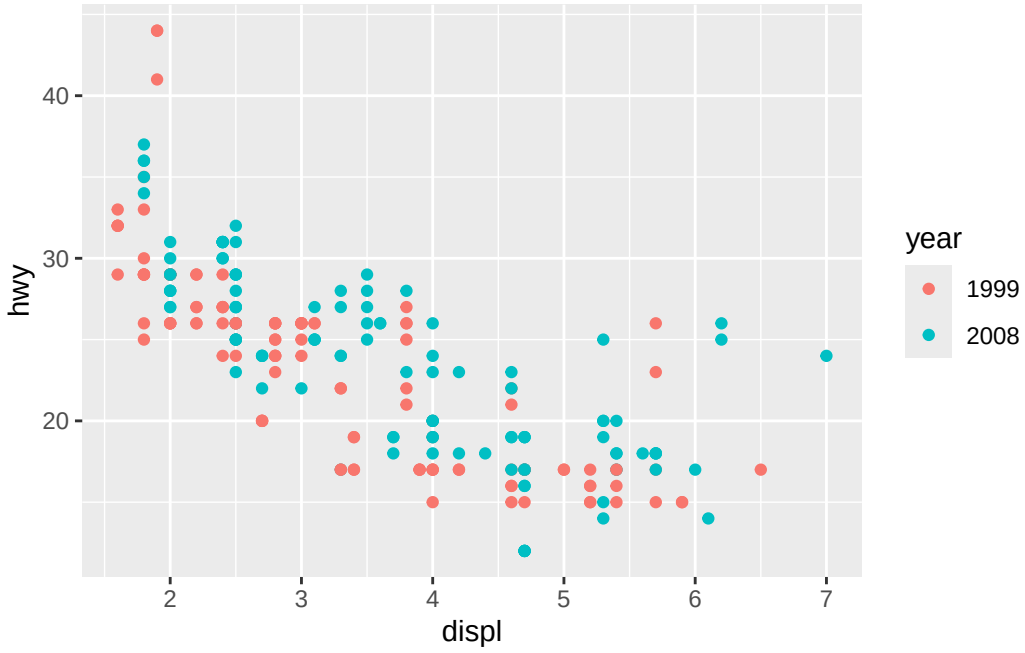
Make the numerical variable year categorical by applying the function as.character(). Then redo your plot from (b).

- Hint: To do that, follow these steps:
  - Before you modify the original dataset mpg, store a copy using a new name, e.g. do my_mpg <- mpg.
  - Remember from Task 1 that mpg (and now also my_mpg) is a data frame. You can address a single attribute in a data frame using the $ sign.
  - This means you can address the attribute year by typing my_mpg$year.
  - Now apply the function as.character() to year to change its data type. Overwrite the original variable year in my_mpg with the transformed one.

```
ggplot(data = mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     color=as.character(year))) +
  geom_point()
```



```r
# or

my_mpg <- mpg
my_mpg$year <- as.character(my_mpg$year)

ggplot(data = my_mpg,
       mapping = aes(x = displ,
                     y = hwy,
                     color = year)) +
  geom_point()
```



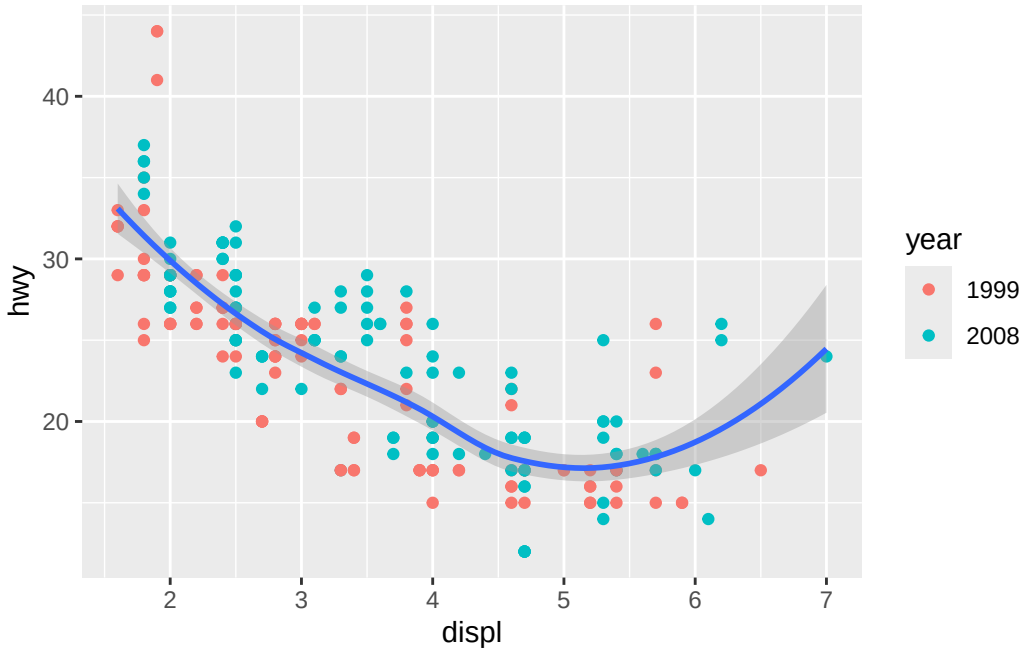
Overlay the scatterplot of my_mpg with a smoothed trendline using geom_smooth. Make sure that only *one* trendline is diplayed.

Make your plot interactive using ggplotly().

```r
my_mpg <- mpg
my_mpg$year <- as.character(my_mpg$year)

ggplot(data = my_mpg,
       mapping = aes(x = displ,
                     y = hwy)) +
```

```
    geom_point(mapping = aes(color = year)) +
    geom_smooth()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```
#ggplotly(my_mpg) # does not work as pdf output
```

Now create facets of this plot along the discrete variable year, but don't make it interactive for now.

- Remark: Notice that year is used twice here. Though color-coding year is not necessary here, it still helps to stress this variable.

  Now create facets of this plot along the discrete variable drv (instead of year).

  Next create facets of this plot along the discrete variable manufacturer (instead of drv).

  What can you read out of these facet plots? Do some of them catch your eye?

Solution:

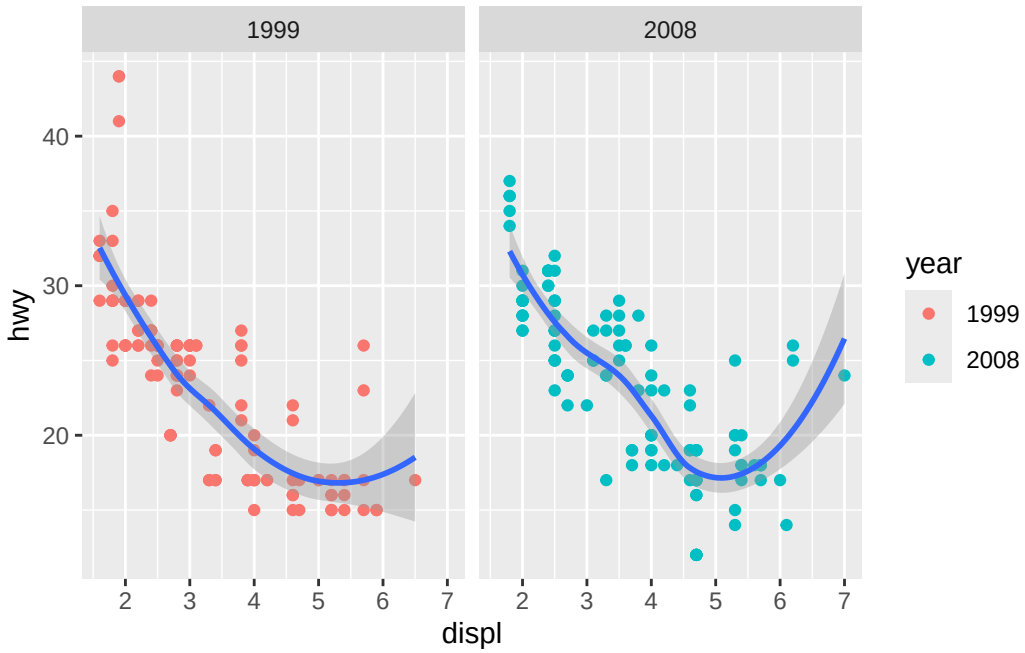
1) jeep, mercury and nissan look weird: the grey band around the trendline is very broad. It is the 95%-confidence interval around the trendline: a true observation will lie within this band

around the trendline prediction with a probability of 95%. It thus means that there is a lot of uncertainty involved in producing the trendline. (We don't know why yet, but we will come back to that later.)

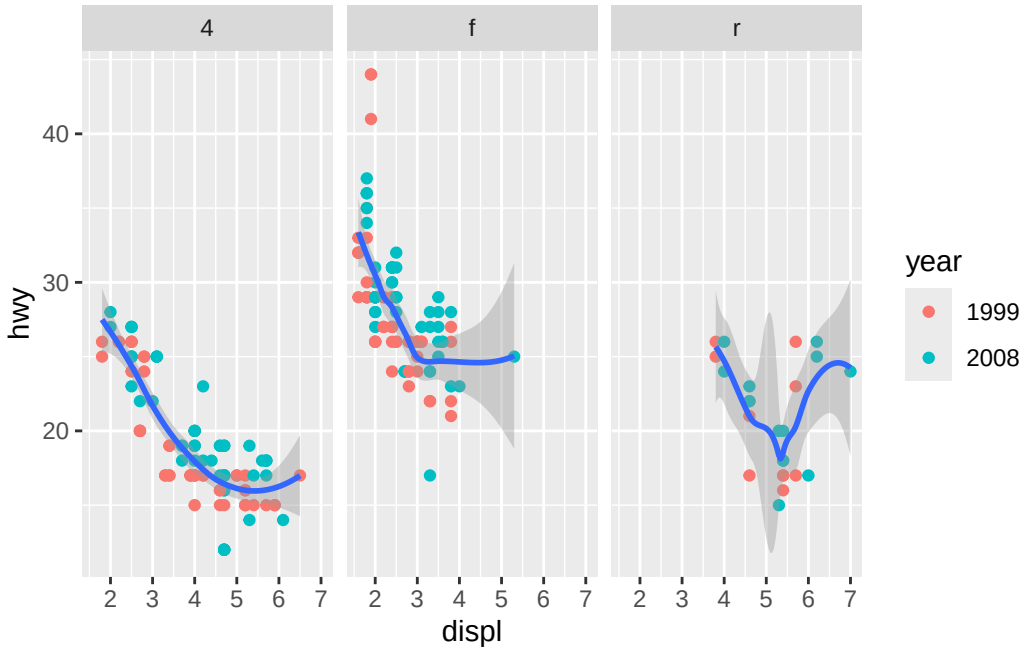
2) lincols, pontiac and subaru look weird as well: its not clear why no trendline is shown at all. (We will come back to this as well!)

3) volkswagen is also bit weird. It makes an unmotivated dip at approximately x=2 that does not seem to correspond to any of the data points shown. (We will see the reason later.)

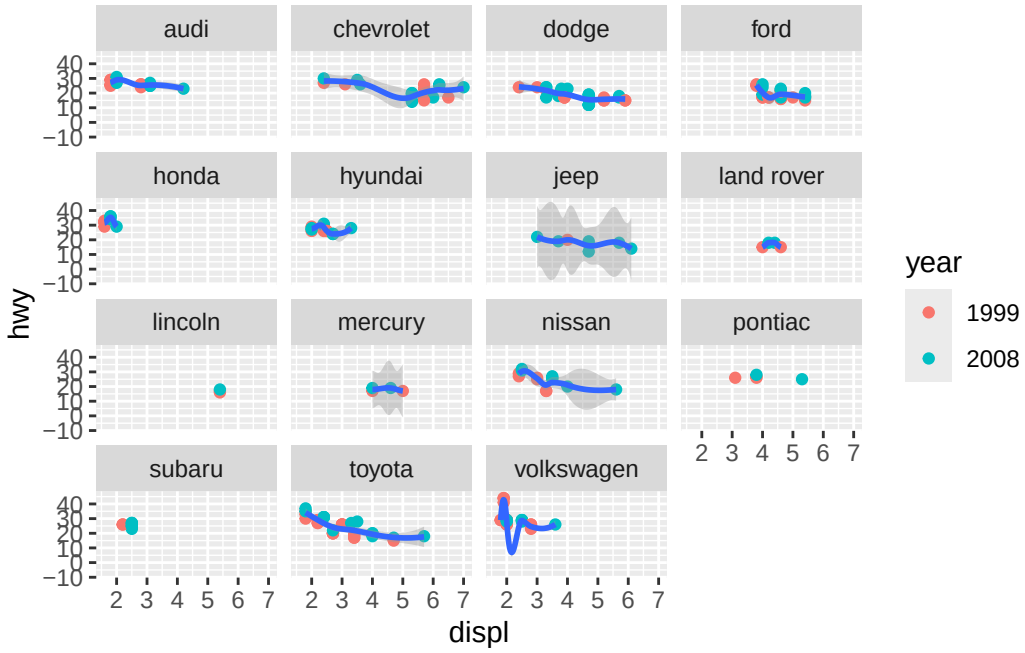
```
my_mpg <- mpg
my_mpg$year <- as.character(my_mpg$year)

my_plot <- ggplot(data = my_mpg,
       mapping = aes(x = displ,
                     y = hwy)) +
  geom_point(mapping = aes(color = year)) +
  geom_smooth()

my_plot + facet_wrap(~ year)
```




```
my_plot + facet_wrap(~ drv)
```

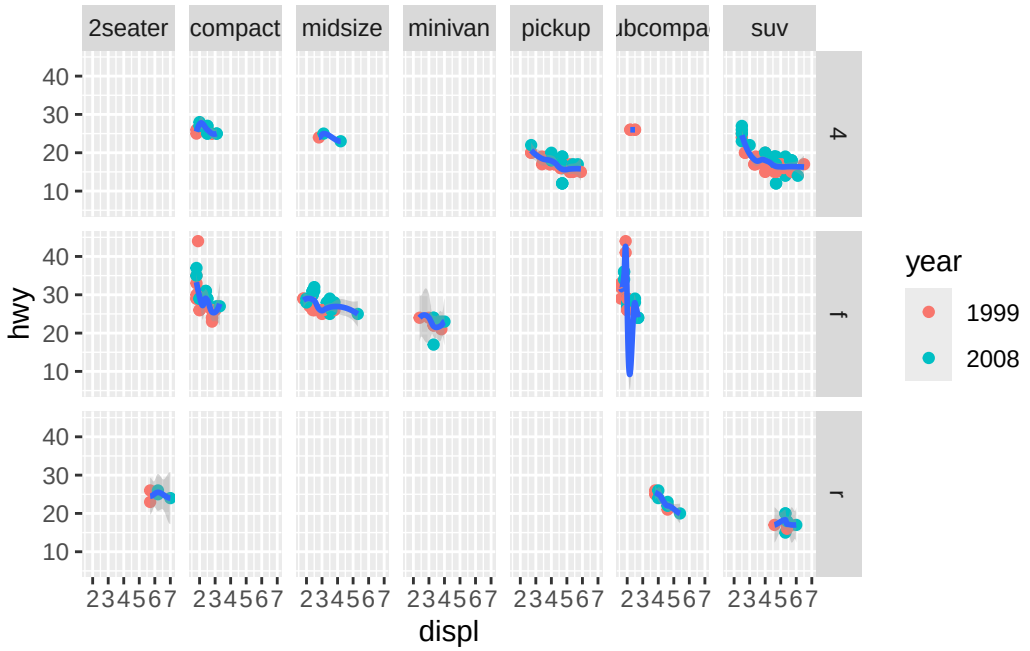



```
my_plot + facet_wrap(~ manufacturer)
```



Drop the manufacturer facets again. Instead create a grid of facets along the discrete variables drv and class.

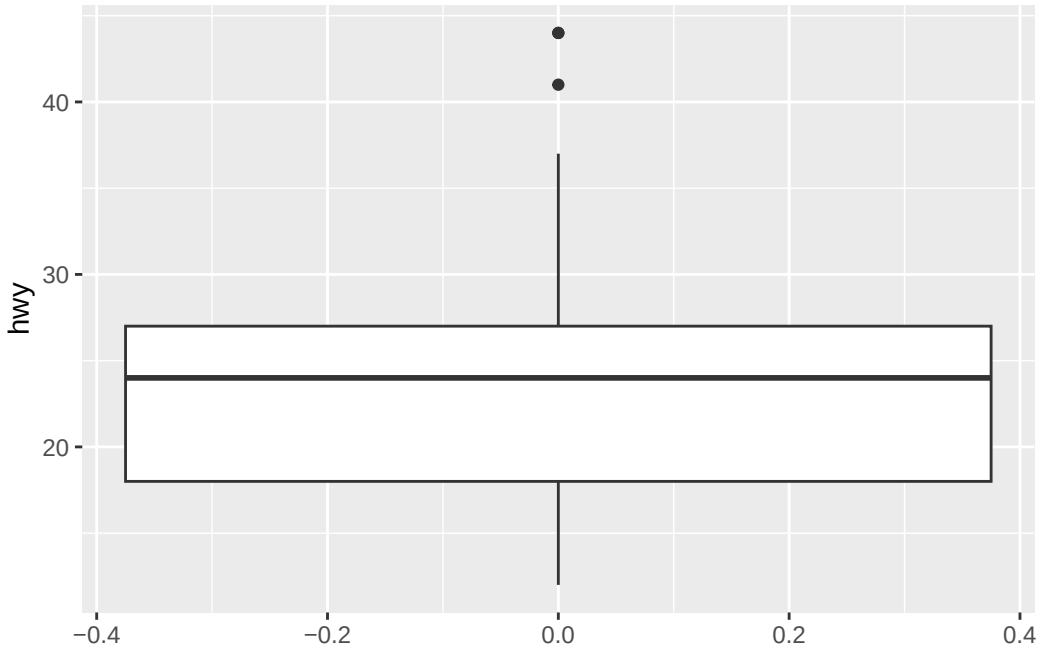
```
my_mpg <- mpg
my_mpg$year <- as.character(my_mpg$year)

my_plot <- ggplot(data = my_mpg,
       mapping = aes(x = displ,
                     y = hwy)) +
  geom_point(mapping = aes(color = year)) +
  geom_smooth()

my_plot + facet_grid(drv ~ class)
```



Create a boxplot now that visualizes statistics about the the highway miles per gallon of the cars in my_mpg.

```
my_mpg <- mpg

ggplot(data = my_mpg,
       mapping = aes(y = hwy)) +
```

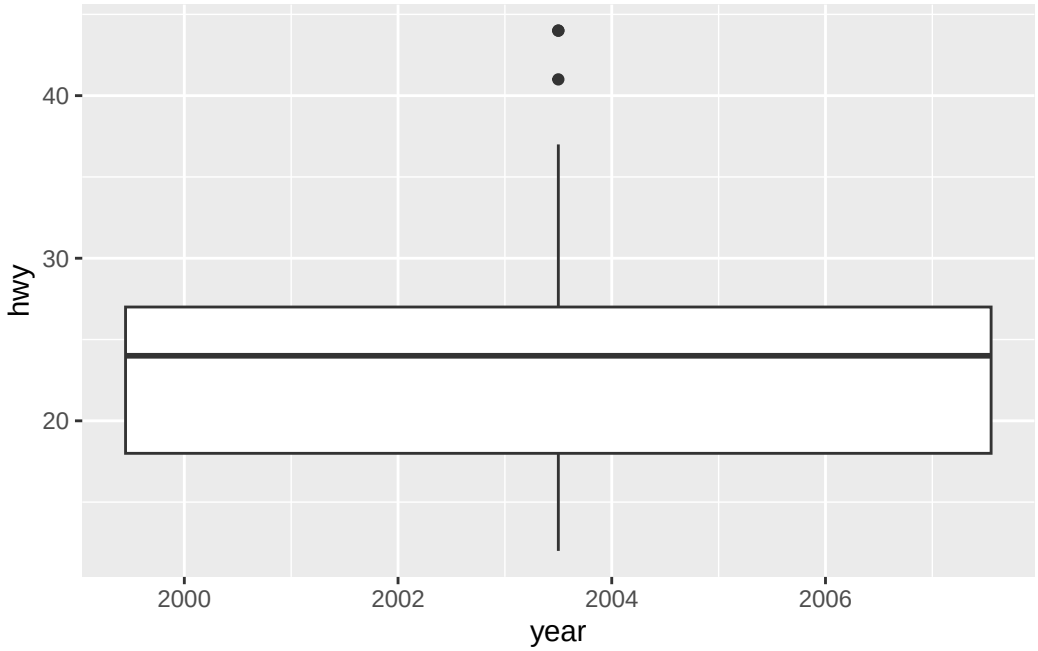
```
  geom_boxplot()
```



Now make one such boxplot per year (1999 and 2008). Make it interactive. What can you read out of it?

Solution: While the median didnt change much, the cars in 2008 show more variability in terms of highway miles per gallon.

```
my_mpg <- mpg

my_plot <- ggplot(data = my_mpg,
       mapping = aes(x = year,
                     y = hwy)) +
  geom_boxplot(mapping = aes(color = year))

my_plot # not interactive
```

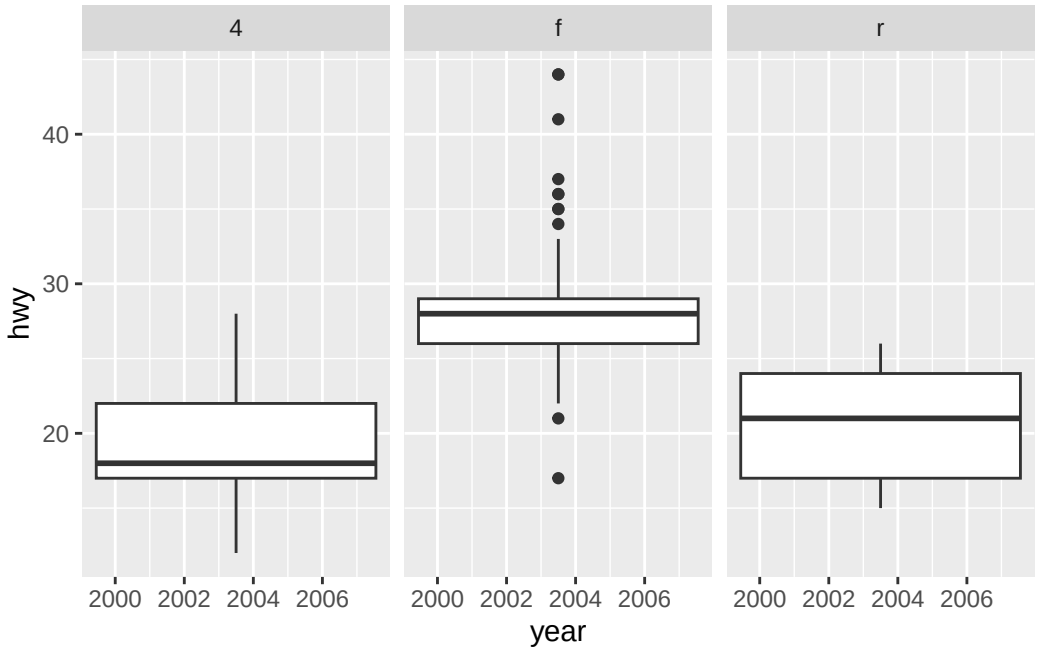



```
#ggplotly(my_plot) # does not work as pdf output
```

Now create facets of this plot along the variable drv. What new insight(s) does it give you?

Solution: Most cars with a front-wheel drive are more efficient than 4wds and rwds: they go farther per gallon of gas - independently of the year of manufacture. Not surprisingly, most 4-wheel drive cars are more expensive in terms of gas needed per mile than front- or rear wheel drive cars. Additionally, they show the biggest change from 1999 to 2008 in terms of variability (inter-quartiule range and whiskers).

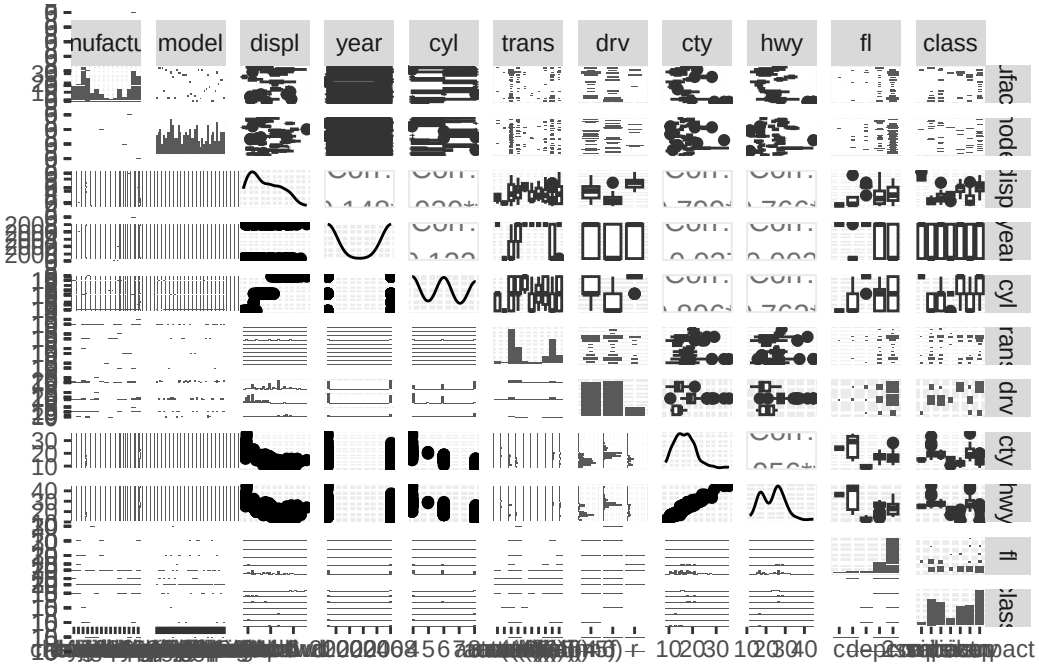
```
my_plot_fac <- my_plot + facet_wrap(~ drv)

my_plot_fac # not interactive
```




```
#ggplotly(my_plot_fac) # does not work as pdf output
```

Finally let's take a look at ggpairs: Try to create a pairwise variable plot of mpg using ggpairs().

```
ggpairs(mpg, cardinality_threshold = 38)
```



We get an error message here! It says that the categorical variable model has more than 15 levels (i.e. more than 15 different values), ggpairs() cannot display more. Check if this is true by typing length(unique(mpg$model)).

- Remark: This is what happens here:
  - mpg$model addresses only the variable column of the data frame.
  - Remember that every column of a data frame is a vector. We can apply the function unique() to see its unique values.
  - The output is again a vector, namely the vector of unique values. (You can check with is.vector().)
  - Since unique(mpg$model) is again a vector, we can apply the function length() to count the number of entries.

```
length(unique(mpg$model))
```

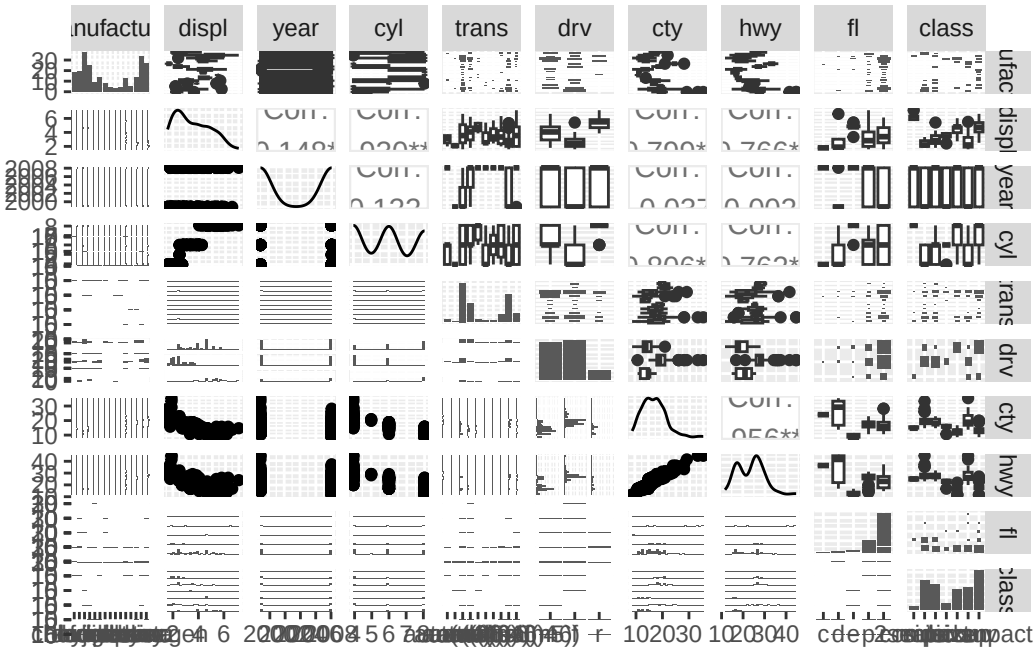
```
[1] 38
```

```
is.vector(unique(mpg$model))
```

```
[1] TRUE
```

The advice in the error message is to exclude the variable model. So let's do that. You can do that by typing ggpairs(mpg[,-c(2)]). Remark: Don't worry, we will learn how this works a bit later in the lecture about data frames.

```
ggpairs(mpg[,-c(2)])
```



You can now sit down, get a tea and take some time to contemplate this plot for a bit. Trying to make sense of the different subplots, if you can. Don't get frustrated if you cannot figure everything out: there is a lot of information in this plot!

- Remark: The logic is the following:
  - In the diagonal, there is only one variable to display (e.g. manufacturer). Since it does not make sense to plot manufacturer vs manufacturer, a plot type is chosen that is suitable for displaying only one variable. If the variable is categorical (like manufacturer), this is a frequency diagram (bar plot). If the variable is numerical (like hwy), it is a density plot.
  - In the rest right of the matrix, plot types are chosen that are suitable to visualize categorial vs categorical, numerical vs numerical or categorical vs numerical

variables. Since the upper right triangle and the lower left triangle are redundant, different plot types are chosen to capture as much info as possible.

– If you are interested, you can get more info about ggpairs, e.g., here: https://r-charts.com/correlation/ggpairs/ and https://www.rdocumentation.org/packages/GGally/versions/2.