Data Programming Course Exercises

May 4, 2016

Andrea Spanò
andrea.spano@quantide.com[1]

[1] mailto:andrea.spano@quantide.com

Contents

Chapter 1

Introduction

In this document you will find some exercises about these sections:

- *Data Objects*
- *Data Import and Export*
- *Data Manipulation*
- *Data Visualization with ggplot2*
- *Writing R functions*

Chapter 2

Data Object

2.1 Vectors

2.1.1 Exercise 1

a. Create a vector, named `vec1`, containing the following values:
   1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90)
```

b. Select the 5-th element of `vec1`.

```
vec1[5]
```

```
## [1] 5
```

c. Select the first 10 elements of `vec1`.

```
vec1[1:10]
```

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

d. Select all the elements of `vec1` apart from the 2nd and the 6th element.

```
vec1[-c(2,6)]
```

```
##  [1]  1  3  4  5  7  8  9 10 15 20 25 30 35 40 45 50 60 70 80 90
```

2.1.2 Exercise 2

a. Generate a vector, named `vec2`, containing the numbers from 1 to 10 and of length 8, using the function `seq()`.

```
vec2 <- seq(from=1, to=10, length.out = 8)
```

b. Select the values of `vec2` which are greater than 4.

```
vec2[vec2>4] # or y > 4; b[y]
```

```
## [1]  4.857143  6.142857  7.428571  8.714286 10.000000
```

c. Select the values of `vec2` which are equal or less than 2 or which are equal or greater than 6.

```
vec2[vec2<=2 | vec2>=6]
```

```
## [1]  1.000000  6.142857  7.428571  8.714286 10.000000
```

2.1.3 Exercise 3

a. Generate the following vector using the function `rep()`:
`vec3 <- c("one", "two", "one", "two", "one", "two")`

```
vec3 <- rep(c("one", "two"), times=3)
```

b. Generate a new vector, named `vec5`, combining the previous vector, `vec3`, with the following one:

```
vec4 <- c("three", "four")
```

```
vec5 <- c(vec3, vec4)
vec5
```

```
## [1] "one"   "two"   "one"   "two"   "one"   "two"   "three" "four"
```

2.2 Matrices

2.2.1 Exercise 1

Generate a matrix, named `mat1`, with 5 rows and 3 columns, using `matrix()` function:

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
## [5,]   13   14   15
```

```
mat1 <- matrix(1:15, nrow = 5, ncol = 3, byrow = TRUE)
```

2.2.2 Exercise 2

Starting from the following vector:

```
mat2 <- 1:8
```

Generate a matrix with 2 rows and 4 columns using `dim()` function.

```
dim(mat2) <- c(2,4)
mat2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

2.2.3 Exercise 3

a. Generate a matrix, named `mat3`, combining the following columns:

```
a <- 1:3
b <- 7:9
c <- 8:6
```

```
mat3 <- cbind(a,b,c)
mat3
```

```
##      a b c
## [1,] 1 7 8
## [2,] 2 8 7
## [3,] 3 9 6
```

b. Add the following row to `mat3`:

```r
d <- 4:6

mat3 <-rbind(mat3, d)
mat3
```

```
##   a b c
##   1 7 8
##   2 8 7
##   3 9 6
## d 4 5 6
```

2.2.4 Exercise 4

Considering the following matrix, named `mat4`:

```
mat4 <- matrix(1:24, nrow = 6, ncol = 4, byrow = TRUE)
mat4
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
## [6,]   21   22   23   24
```

a. Select the third and the fifth row of `mat4`.

```
mat4[c(3,5),]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    9   10   11   12
## [2,]   17   18   19   20
```

b. Select all columns of `mat4` apart from the first.

```
mat4[, -1]
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    6    7    8
## [3,]   10   11   12
## [4,]   14   15   16
## [5,]   18   19   20
## [6,]   22   23   24
```

c. Select second and third rows and second and third columns of `mat4`.

```
mat4[2:3, 2:3] # or mat4[c(2,3) , c(2,3)]
```

```
##      [,1] [,2]
## [1,]    6    7
## [2,]   10   11
```

2.3 Lists

2.3.1 Exercise 1

a. Generate a list, named `list1` that contains the following R elements:

```
vec <- 1:10
mat <- matrix(1:9, ncol = 3)
name <- "Oscar"
```

```
list1 <- list(vec = 1:10, mat = matrix(1:9, ncol = 3), name = "Oscar")
list1
```

```
## $vec
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $mat
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $name
## [1] "Oscar"
```

b. Add to `list1` the following element:

```
letters <- c("a", "b", "c", "d")
```

```
list1$letters <- letters
list1
```

```
## $vec
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $mat
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $name
## [1] "Oscar"
##
## $letters
## [1] "a" "b" "c" "d"
```

2.3.2 Exercise 2

Given the following list, named `list2`:

```
list2 <- list(vec = c(1,3,5,7,8), mat = matrix(1:12, ncol = 4),
              sub_list = list(names = c("Veronica", "Enrico", "Andrea", "Anna"),
                              numbers = 1:4))
list2
```

```
## $vec
## [1] 1 3 5 7 8
##
## $mat
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## $sub_list
## $sub_list$names
## [1] "Veronica" "Enrico"   "Andrea"   "Anna"
##
## $sub_list$numbers
## [1] 1 2 3 4
```

a. Entract the first element of `list2`.

```
list2[1]
```

```
## $vec
## [1] 1 3 5 7 8
```

b. Extract the objects contained in the first element of `list2`.

```
list2[[1]]
```

```
## [1] 1 3 5 7 8
```

c. Extract the element named `sub_list` of `list2`.

```
list2$sub_list
```

```
## $names
## [1] "Veronica" "Enrico"   "Andrea"   "Anna"
##
## $numbers
## [1] 1 2 3 4
```

d. Extract the second rows of the matrix included in the second element of `list2`.

```
list2[[2]][2,] # or list2$mat[2,]
```

```
## [1]  2  5  8 11
```

2.4 Factors

2.4.1 Exercise 1

Starting from the vector:

```
fac1 <- c("F", "F", "M", "M" , "F")
```

Generate the corresponding factor with two levels: “F” and “M”

```
fac1 <- factor(fac1, levels = c("F", "M"))
fac1
```

```
## [1] F F M M F
## Levels: F M
```

2.4.2 Exercise 2

Starting from the vector:

```
fac2 <- c(1, 1, 1, 2, 2, 2)
```

a. Generate the corresponding factor considering that 1 = “Female”, 2 = “Male” e 3 = “Trans”.

```
fac2 <- factor(fac2, levels = c(1,2,3), labels = c("Female", "Male", "Trans"))
fac2
```

```
## [1] Female Female Female Male   Male   Male
## Levels: Female Male Trans
```

b. Select the all elements of `fac2` apart from “Male”.

```
fac2[fac2!= "Male"]
```

```
## [1] Female Female Female
## Levels: Female Male Trans
```

2.5 Data Frames

2.5.1 Exercise 1

a. Generate a data frame, named `df1`, corresponding to:

```
##    id     name class mean
## 1   1     Luca    5A  6.0
## 2   2   Chiara    5A  7.0
## 3   3     Lisa    5A  5.0
## 4   4   Matteo    5A  6.5
## 5   5    Alice    5A  7.5
## 6   6    Marco    5B  4.5
## 7   7 Veronica    5B  9.0
## 8   8   Nicola    5B  8.0
## 9   9    Elena    5B  8.5
## 10 10  Daniele    5B  7.0
```

Remember to maintain character vectors as they are, specifiyng `stringsAsFactors = FALSE`.

```
df1 <- data.frame(id=1:10,
                  name=c("Luca", "Chiara", "Lisa", "Matteo", "Alice", "Marco",
                         "Veronica", "Nicola", "Elena", "Daniele"),
                  class=c(rep("5A", times=5), rep("5B", times=5)),
                  mean= c(6,7,5,6.5,7.5,4.5, 9, 8, 8.5, 7), stringsAsFactors = FALSE)
df1
```

```
##    id     name class mean
## 1   1     Luca    5A  6.0
## 2   2   Chiara    5A  7.0
## 3   3     Lisa    5A  5.0
## 4   4   Matteo    5A  6.5
## 5   5    Alice    5A  7.5
## 6   6    Marco    5B  4.5
## 7   7 Veronica    5B  9.0
## 8   8   Nicola    5B  8.0
## 9   9    Elena    5B  8.5
## 10 10  Daniele    5B  7.0
```

```
# Other solution
id <- 1:10
name <- c("Luca", "Chiara", "Lisa", "Matteo", "Alice", "Marco",
          "Veronica", "Nicola", "Elena", "Daniele")
class <- c(rep("5A", times=5), rep("5B", times=5))
mean <- c(6,7,5,6.5,7.5,4.5, 9, 8, 8.5, 7)
df1 <- data.frame(id, name, class, mean, stringsAsFactors = FALSE)
df1
```

```
##    id     name class mean
## 1   1     Luca    5A  6.0
## 2   2   Chiara    5A  7.0
## 3   3     Lisa    5A  5.0
## 4   4   Matteo    5A  6.5
## 5   5    Alice    5A  7.5
## 6   6    Marco    5B  4.5
## 7   7 Veronica    5B  9.0
## 8   8   Nicola    5B  8.0
## 9   9    Elena    5B  8.5
## 10 10  Daniele    5B  7.0
```

b. Select the first 3 rows of `df1`.

```
df1[1:3,]
```

```
##   id   name class mean
## 1  1   Luca    5A    6
## 2  2 Chiara    5A    7
## 3  3   Lisa    5A    5
```

c. Select the last 6 rows and the first 3 columns of `df1`.

```
df1[5:10, 1:3]
```

```
##    id     name class
## 5   5    Alice    5A
## 6   6    Marco    5B
## 7   7 Veronica    5B
## 8   8   Nicola    5B
## 9   9    Elena    5B
## 10 10  Daniele    5B
```

d. Select the column `class` of `df1`.

```
df1$class
```

```
##  [1] "5A" "5A" "5A" "5A" "5A" "5B" "5B" "5B" "5B" "5B"
```

e. Convert the column `class` of `df1` in a factor with levels: “5A” and “5B”

```
df1$class <- factor(df1$class, levels = c("5A", "5B"))
df1$class
```

```
##  [1] 5A 5A 5A 5A 5A 5B 5B 5B 5B 5B
## Levels: 5A 5B
```

f. How many columns and rows `df1` has?

```
dim(df1) # or ncol(df1) and nrow(df1)
```

```
## [1] 10  4
```

g. Generate another dataframe, named `df2` composed by the columns `name` and `mean` of `df1`, specifying the argument `stringsAsFactors = FALSE`.

```
df2 <- data.frame(name = df1$name, mean=df1$mean, stringsAsFactors = FALSE)
df2
```

```
##        name mean
## 1      Luca  6.0
## 2    Chiara  7.0
## 3      Lisa  5.0
## 4    Matteo  6.5
## 5     Alice  7.5
## 6     Marco  4.5
## 7  Veronica  9.0
## 8    Nicola  8.0
## 9     Elena  8.5
## 10  Daniele  7.0
```

h. Show the first rows and the structure of `df2`.

```
head(df2)
```

```
##     name mean
## 1   Luca  6.0
## 2 Chiara  7.0
## 3   Lisa  5.0
## 4 Matteo  6.5
## 5  Alice  7.5
## 6  Marco  4.5
```

```
str(df2)
```

```
## 'data.frame':    10 obs. of  2 variables:
##  $ name: chr  "Luca" "Chiara" "Lisa" "Matteo" ...
##  $ mean: num  6 7 5 6.5 7.5 4.5 9 8 8.5 7
```

Chapter 3

Data Import

First of all, set your working directory in the *data* folder, using `setwd()` function, like in this example

```
setwd("C:/Users/Veronica/Documents/rbase/data)
```

We will work inside this folder.

3.1 Text Files

3.1.1 Exercise 1

a. Import text file named *"tuscany.txt"* and save it in an R object named `tuscany_df`. Open the text file before importing it to control if the first row contains column names and to control the field and the decimal separator characters. Remember to not import the character columns as factors.

```
tuscany_df <- read.table("tuscany.txt", header = TRUE, sep = "|",
                         dec=".", stringsAsFactors = FALSE)
```

b. Visualize the first rows of `tuscany_df`

```
head(tuscany_df)
```

```
##   id sex year_of_birth marital_status  income house_number
## 1  1   M          1969        married 16101.1       5144.0
## 2  2   M          1962         single 17220.0       6158.0
## 3  3   M          1965       divorcee 28801.9      10078.0
## 4  4   F          1968         single 25964.0      11133.7
## 5  5   M          1975        married 16522.5       5078.0
## 6  6   M          1977        married 18124.0       5115.0
```

```
##                city_name province provincial_acronym
## 1            Riparbella     Pisa                 PI
## 2              Capolona   Arezzo                 AR
## 3             Pomarance     Pisa                 PI
## 4               Cascina     Pisa                 PI
## 5              Quarrata  Pistoia                 PT
## 6 Castiglion Fiorentino   Arezzo                 AR
```

3.1.2 Exercise 2

Import 7 rows of the text file named *"solar.txt"* skipping the first two rows. Save it in the object `solar_df`.
Open the text file before importing it to control if the first row contains column names and to control the field and the decimal separator characters. Remember to not import the character columns as factors.

```
solar_df <- read.table("solar.txt", header = FALSE, sep = ",",
                       dec=".", stringsAsFactors = FALSE,
                       nrows = 7, skip = 2)
solar_df
```

```
##    V1    V2    V3    V4
## 1 mar 23877 24671 22455
## 2 apr 24377 23677 23670
## 3 mag 24581 25476 24999
## 4 giu 22154 21998 22451
## 5 lug 20924 21645 23871
## 6 ago 23183 22576 23556
## 7 set 27446 27695 28664
```

3.1.3 Exercise 3

Considering the following data frame, named `df`:

```
df <- data.frame(col1=1:4, col2=4:1, col3=c("one", "two", "three", "four"),
                 stringsAsFactors = FALSE)
```

Save it in a .txt file named *"exercise-3.txt"* in *data* folder.

```
write.table(df, file="exercise-3.txt")
```

3.2 Excel Files

3.2.1 Exercise 1

a. Import .xlsx file *"flowers.xlsx"* using `XLConnect` function `loadWorkbook()` and save it in a R workbook object named `flowers`.
   Remember to load `XLConnect` package, supposing it is already installed.

```
require(XLConnect)
```

```
flowers <- loadWorkbook("flowers.xlsx")
```

b. Read *iris* sheet with `readWorksheet()` function and save it in `flower_df` object. Then, visualize its first rows.

```
flowers_df <- readWorksheet(flowers, sheet = 'iris')
head(flowers_df)
```

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

3.2.2 Exercise 2

a. Create a new file xlsx, named *"exercise-2.xlsx"*, and save it in the R worksheet object, named `ex_2`. Use: `loadWorkbook()` and `saveWorkbook()` functions of `XLConnect`.

```
require(XLConnect)
ex_2 <- loadWorkbook(filename = "exercise-2.xlsx", create = TRUE)
saveWorkbook(ex_2)
```

b. Create a sheet, named `df`, in the R workbook object using `createSheet()` function. Remember to save the changes also in .xlsx file (use `saveWorkbook()` function).

```
createSheet(object = ex_2, name = 'df')
saveWorkbook(ex_2)
```

c. Considering the following data frame, named `numbers_df`:

```
numbers_df <- data.frame(a= 1:4, b=c("one", "two", "three", "four"),
                         stringsAsFactors = FALSE)
numbers_df
```

```
##   a     b
## 1 1   one
## 2 2   two
## 3 3 three
## 4 4  four
```

Add it to `df` sheet of `ex_2` R workbook object, starting from row 3 and from column 2. Use the function `writeWorksheet()`. Remember to save the changes also in .xlsx file (use `saveWorkbook()` function).

```
writeWorksheet(object = ex_2, data = numbers_df, sheet = "df", startRow = 3, startCol = 3)
saveWorkbook(ex_2)
```

3.3 Databases

3.3.1 Exercise 1

a. Connect to *"plant.sqlite"* SQLite database, using `dbConnect()` function of `RSQLite` package. Save the connection in an R object, named `con`.
   Remember to load `RSQLite` package, supposing it is already installed.

```
require(RSQLite)
```

```
con <- dbConnect(RSQLite::SQLite(), "plant.sqlite")
```

b. See the list of available tables in *"plant.sqlite"* db, using `dbListTables()` function.

```
dbListTables(con)
```

```
## [1] "PlantGrowth"
```

c. See list of fields in *"PlantGrowth"* table of *"plant.sqlite"* db, using `dbListFields()` function.

```
dbListFields(con, name = "PlantGrowth")
```

```
## [1] "weight" "group"
```

d. Send query to *"PlantGrowth"* table of *"plant.sqlite"* which select the records with `weight` greater than 5.5.

```
dbGetQuery(con, "SELECT * FROM PlantGrowth WHERE weight >= 5.5")
```

```
##   weight group
## 1   5.58  ctrl
## 2   6.11  ctrl
## 3   5.87  trt1
## 4   6.03  trt1
## 5   6.31  trt2
## 6   5.54  trt2
## 7   5.50  trt2
## 8   6.15  trt2
## 9   5.80  trt2
```

e. Disconnect from the database, using `dbDisconnect()` function.

```
dbDisconnect(con)
```

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

3.4 R Data Files

3.4.1 Exercise 1

Given the following data frame, named `df_rdata`:

```
df_rdata <- data.frame(a=1:20, b=20:1)
```

Save it in *.Rda* format in the file *"df_rdata.Rda"*, using `save()` function.

```
save(df_rdata, file = "df_rdata.Rda")
```

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

3.4.2 Exercise 2

Load *"drug.Rda"* file into the environment, using `load()` function.

```
load("drug.Rda")
```

Chapter 4

Data Manipulation with `dplyr`

Load `dplyr` package, supposing it is already installed.

```
require(dplyr)
```

4.1 Data

All the following exercises are based on the `nycflights13` data, taken from the `nycflights13` package.
So first of all, install and load this package

```
install.packages("nycflights13")
require(nycflights13)
```

The `nycflights13` package contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013: 336,776 flights in total.

```
ls(pos = "package:nycflights13")
```

```
## [1] "airlines" "airports" "flights"  "planes"   "weather"
```

To help understand what causes delays, it includes a number of useful datasets:

- `flights`: information about all flights that departed from NYC
- `weather`: hourly meterological data for each airport;
- `planes`: construction information about each plane;
- `airports`: airport names and locations;

- `airlines`: translation between two letter carrier codes and names.

Let us explore the features of `flights` datasets, which will be used in the following exercises.

```
data("flights")
```

4.1.1 flights

This dataset contains on-time data for all flights that departed from NYC (i.e. JFK, LGA or EWR) in 2013. The data frame has 16 variables and 336776 observations. The variables are organised as follow:

- Date of departure: `year`, `month`, `day`;
- Departure and arrival times (local tz): `dep_time`, `arr_time`;
- Departure and arrival delays, in minutes: `dep_delay`, `arr_delay` (negative times represent early departures/arrivals);
- Time of departure broken in to hour and minutes: `hour`, `minute`;
- Two letter carrier abbreviation: `carrier`;
- Plane tail number: `tailnum`;
- Flight number: `flight`;
- Origin and destination: `origin`, `dest`;
- Amount of time spent in the air: `air_time`;
- Distance flown: `distance`.

```
dim(flights)
```

```
## [1] 336776     16
```

```
head(flights)
```

```
##   year month day dep_time dep_delay arr_time arr_delay carrier tailnum flight
## 1 2013     1   1      517         2      830        11      UA  N14228   1545
## 2 2013     1   1      533         4      850        20      UA  N24211   1714
## 3 2013     1   1      542         2      923        33      AA  N619AA   1141
## 4 2013     1   1      544        -1     1004       -18      B6  N804JB    725
## 5 2013     1   1      554        -6      812       -25      DL  N668DN    461
## 6 2013     1   1      554        -4      740        12      UA  N39463   1696
##   origin dest air_time distance hour minute
## 1    EWR  IAH      227     1400    5     17
## 2    LGA  IAH      227     1416    5     33
```

```
## 3     JFK  MIA       160      1089     5      42
## 4     JFK  BQN       183      1576     5      44
## 5     LGA  ATL       116       762     5      54
## 6     EWR  ORD       150       719     5      54
```

```r
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  16 variables:
##  $ year     : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ dep_delay: num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ arr_delay: num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier  : chr  "UA" "UA" "AA" "B6" ...
##  $ tailnum  : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ flight   : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ origin   : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest     : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance : num  1400 1416 1089 1576 762 ...
##  $ hour     : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ minute   : num  17 33 42 44 54 54 55 57 57 58 ...
```

4.2 Select

4.2.1 Exercise 1

Extract the following information:

- month;
- day;
- air_time;
- distance.

```
select(flights, month, day, air_time, distance)
```

```
## Source: local data frame [336,776 x 4]
##
##    month   day air_time distance
##    (int) (int)    (dbl)    (dbl)
## 1      1     1      227     1400
## 2      1     1      227     1416
## 3      1     1      160     1089
## 4      1     1      183     1576
## 5      1     1      116      762
## 6      1     1      150      719
## 7      1     1      158     1065
## 8      1     1       53      229
## 9      1     1      140      944
## 10     1     1      138      733
## ..   ...   ...      ...      ...
```

```
# flights %>% select(month, day, air_time, distance)
```

4.2.2 Exercise 2

Extract all information about `flights` except hour and minute.

```
select(flights, -c(hour, minute))
```

```
## Source: local data frame [336,776 x 14]
##
##     year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1   2013     1     1      517         2      830        11      UA  N14228
## 2   2013     1     1      533         4      850        20      UA  N24211
```

```
## 3   2013    1    1    542     2   923   33   AA  N619AA
## 4   2013    1    1    544    -1  1004  -18   B6  N804JB
## 5   2013    1    1    554    -6   812  -25   DL  N668DN
## 6   2013    1    1    554    -4   740   12   UA  N39463
## 7   2013    1    1    555    -5   913   19   B6  N516JB
## 8   2013    1    1    557    -3   709  -14   EV  N829AS
## 9   2013    1    1    557    -3   838   -8   B6  N593JB
## 10  2013    1    1    558    -2   753    8   AA  N3ALAA
## ..   ...  ...  ...    ...   ...   ...  ...  ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl)
```

```
# flights %>% select(-c(hour, minute))
```

4.2.3 Exercise 3

Extract `tailnum` variable and rename it into `tail_num`

```
select(flights, tail_num=tailnum)
```

```
## Source: local data frame [336,776 x 1]
##
##    tail_num
##       (chr)
## 1    N14228
## 2    N24211
## 3    N619AA
## 4    N804JB
## 5    N668DN
## 6    N39463
## 7    N516JB
## 8    N829AS
## 9    N593JB
## 10   N3ALAA
## ..      ...
```

```
# flights %>% select(tail_num=tailnum)
```

4.3 Filter

4.3.1 Exercise 1

Select all flights which delayed more than 1000 minutes at departure.

```
filter(flights, dep_delay > 1000)
```

```
## Source: local data frame [5 x 16]
##
##    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1  2013     1     9      641      1301     1242      1272      HA  N384HA
## 2  2013     1    10     1121      1126     1239      1109      MQ  N517MQ
## 3  2013     6    15     1432      1137     1607      1127      MQ  N504MQ
## 4  2013     7    22      845      1005     1044       989      MQ  N665MQ
## 5  2013     9    20     1139      1014     1457      1007      AA  N338AA
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl)
```

```
# flights %>% filter(dep_delay > 1000)
```

4.3.2 Exercise 2

Select all flights which delayed more than 1000 minutes at departure or at arrival.

```
filter(flights, dep_delay > 1000 | arr_delay >1000)
```

```
## Source: local data frame [5 x 16]
##
##    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1  2013     1     9      641      1301     1242      1272      HA  N384HA
## 2  2013     1    10     1121      1126     1239      1109      MQ  N517MQ
## 3  2013     6    15     1432      1137     1607      1127      MQ  N504MQ
## 4  2013     7    22      845      1005     1044       989      MQ  N665MQ
## 5  2013     9    20     1139      1014     1457      1007      AA  N338AA
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl)
```

```
# flights %>% filter(dep_delay > 1000 | arr_delay >1000)
```

4.3.3 Exercise 3

Select all flights which took off from “EWR” and landed in “IAH”.

```
filter(flights, origin == "EWR" & dest == "IAH")
```

```
## Source: local data frame [3,973 x 16]
##
##     year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1   2013     1     1      517         2      830        11      UA  N14228
## 2   2013     1     1      739         0     1104        26      UA  N37408
## 3   2013     1     1      908         0     1228         9      UA  N12216
## 4   2013     1     1     1044        -1     1352         1      UA  N667UA
## 5   2013     1     1     1205         5     1503        -2      UA  N39418
## 6   2013     1     1     1356         6     1659        19      UA  N26906
## 7   2013     1     1     1527        12     1854        44      UA  N69059
## 8   2013     1     1     1620         0     1945        23      UA  N18119
## 9   2013     1     1     1725         5     2045        24      UA  N17122
## 10  2013     1     1     1959        -1     2310         3      UA  N76514
## ..   ...   ...   ...      ...       ...      ...       ...     ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl)
```

```
# flights %>% filter(origin == "EWR" & dest == "IAH")
```

4.4 Arrange

4.4.1 Exercise 1

Sort the flights in chronological order.

```
arrange(flights, year, month, day)
```

```
## Source: local data frame [336,776 x 16]
##
##     year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1   2013     1     1      517         2      830        11      UA  N14228
## 2   2013     1     1      533         4      850        20      UA  N24211
## 3   2013     1     1      542         2      923        33      AA  N619AA
## 4   2013     1     1      544        -1     1004       -18      B6  N804JB
## 5   2013     1     1      554        -6      812       -25      DL  N668DN
## 6   2013     1     1      554        -4      740        12      UA  N39463
## 7   2013     1     1      555        -5      913        19      B6  N516JB
## 8   2013     1     1      557        -3      709       -14      EV  N829AS
## 9   2013     1     1      557        -3      838        -8      B6  N593JB
## 10  2013     1     1      558        -2      753         8      AA  N3ALAA
## ..   ...   ...   ...      ...       ...      ...       ...     ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl)
```

```
# flights %>% arrange(year, month, day)
```

4.4.2 Exercise 2

Sort the flights by decreasing arrival delay.

```
arrange(flights, desc(arr_delay))
```

```
## Source: local data frame [336,776 x 16]
##
##     year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1   2013     1     9      641      1301     1242      1272      HA  N384HA
## 2   2013     6    15     1432      1137     1607      1127      MQ  N504MQ
## 3   2013     1    10     1121      1126     1239      1109      MQ  N517MQ
## 4   2013     9    20     1139      1014     1457      1007      AA  N338AA
## 5   2013     7    22      845      1005     1044       989      MQ  N665MQ
## 6   2013     4    10     1100       960     1342       931      DL  N959DL
## 7   2013     3    17     2321       911      135       915      DL  N927DA
```

```
## 8    2013     7    22     2257       898      121       895      DL  N6716C
## 9    2013    12     5      756       896     1058       878      AA  N5DMAA
## 10   2013     5     3     1133       878     1250       875      MQ  N523MQ
## ..    ...   ...   ...      ...       ...      ...       ...     ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl)
```

```
# flights %>% arrange(desc(arr_delay))
```

4.4.3 Exercise 3

Sort the flights by origin (in alphabetical order) and decreasing arrival delay.

```
arrange(flights, origin, desc(arr_delay))
```

```
## Source: local data frame [336,776 x 16]
##
##      year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##     (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1    2013     1    10     1121      1126     1239      1109      MQ  N517MQ
## 2    2013    12     5      756       896     1058       878      AA  N5DMAA
## 3    2013     5     3     1133       878     1250       875      MQ  N523MQ
## 4    2013    12    19      734       849     1046       847      DL  N375NC
## 5    2013    12    17      705       845     1026       846      AA  N5EMAA
## 6    2013    11     3      603       798      829       796      DL  N990AT
## 7    2013     2    24     1921       786     2135       773      DL  N348NW
## 8    2013    10    14     2042       702     2255       688      DL  N943DL
## 9    2013     7    21     1555       580     1955       645      AA  N3EMAA
## 10   2013     7     7     2123       653       17       632      VX  N521VA
## ..    ...   ...   ...      ...       ...      ...       ...     ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl)
```

```
# flights %>% arrange(origin, desc(arr_delay))
```

4.5 Mutate

4.5.1 Exercise 1

Add the following new variable to the `flights` dataset:

- the speed in miles per hour, named `speed` (`distance` / `air_time` * 60).

Consider that times are in minutes and distances are in miles.

```
mutate(flights, speed = distance / air_time * 60)
```

```
## Source: local data frame [336,776 x 17]
##
##     year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1   2013     1     1      517         2      830        11      UA  N14228
## 2   2013     1     1      533         4      850        20      UA  N24211
## 3   2013     1     1      542         2      923        33      AA  N619AA
## 4   2013     1     1      544        -1     1004       -18      B6  N804JB
## 5   2013     1     1      554        -6      812       -25      DL  N668DN
## 6   2013     1     1      554        -4      740        12      UA  N39463
## 7   2013     1     1      555        -5      913        19      B6  N516JB
## 8   2013     1     1      557        -3      709       -14      EV  N829AS
## 9   2013     1     1      557        -3      838        -8      B6  N593JB
## 10  2013     1     1      558        -2      753         8      AA  N3ALAA
## ..   ...   ...   ...      ...       ...      ...       ...     ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl), speed (dbl)
```

```
# flights %>% mutate(speed =distance / air_time * 60)
```

4.5.2 Exercise 2

Add the following new variables to the `flights` dataset:

- the gained time in minutes (named `gain`), defined as the difference between delay at departure and delay at arrival;
- the gain time per hours, defined as `gain` / (`air_time` / 60)

```
mutate(flights, gain = arr_delay - dep_delay,
  gain_per_hour = gain / (air_time / 60))
```

```
## Source: local data frame [336,776 x 18]
##
##     year month   day dep_time dep_delay arr_time arr_delay carrier tailnum
##    (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)
## 1   2013     1     1      517         2      830        11      UA  N14228
## 2   2013     1     1      533         4      850        20      UA  N24211
## 3   2013     1     1      542         2      923        33      AA  N619AA
## 4   2013     1     1      544        -1     1004       -18      B6  N804JB
## 5   2013     1     1      554        -6      812       -25      DL  N668DN
## 6   2013     1     1      554        -4      740        12      UA  N39463
```

```
## 7   2013    1    1     555      -5     913      19     B6  N516JB
## 8   2013    1    1     557      -3     709     -14     EV  N829AS
## 9   2013    1    1     557      -3     838      -8     B6  N593JB
## 10  2013    1    1     558      -2     753       8     AA  N3ALAA
## ..   ...  ...  ...     ...     ...     ...     ...    ...     ...
## Variables not shown: flight (int), origin (chr), dest (chr), air_time (dbl),
##   distance (dbl), hour (dbl), minute (dbl), gain (dbl), gain_per_hour (dbl)
```

```
# flights %>% mutate(gain = arr_delay - dep_delay,
#     gain_per_hour = gain / (air_time / 60))
```

4.6 Summarise

4.6.1 Exercise 1

Calculate minimum, mean and maximum delay at arrival. Remember to add `na.rm=TRUE` option to all calculations.

```
summarise(flights, min_delay = min(arr_delay, na.rm=TRUE),
          mean_delay = mean(arr_delay, na.rm=TRUE),
          max_delay = max(arr_delay, na.rm=TRUE))
```

```
## Source: local data frame [1 x 3]
##
##   min_delay mean_delay max_delay
##       (dbl)      (dbl)     (dbl)
## 1       -86   6.895377      1272
```

```
# flights %>% summarise(min_delay = min(arr_delay, na.rm=TRUE),
#     mean_delay = mean(arr_delay, na.rm=TRUE),
#     max_delay = max(arr_delay, na.rm=TRUE))
```

4.7 Group_by

4.7.1 Exercise 1

Calculate number of flights, minimum, mean and maximum delay at departure for flights by month.
Remember to add `na.rm=TRUE` option to all calculations.

```
by_month <- group_by(flights, month)
```

```
summarise(by_month, min_delay = min(dep_delay, na.rm=TRUE),
          mean_delay = mean(dep_delay, na.rm=TRUE),
          max_delay = max(dep_delay, na.rm=TRUE))
```

```
## Source: local data frame [12 x 4]
##
##    month min_delay mean_delay max_delay
##    (int)     (dbl)      (dbl)     (dbl)
## 1      1       -30  10.036665      1301
## 2      2       -33  10.816843       853
## 3      3       -25  13.227076       911
## 4      4       -21  13.938038       960
## 5      5       -24  12.986859       878
## 6      6       -21  20.846332      1137
## 7      7       -22  21.727787      1005
## 8      8       -26  12.611040       520
## 9      9       -24   6.722476      1014
## 10    10       -25   6.243988       702
## 11    11       -32   5.435362       798
## 12    12       -43  16.576688       896
```

```
# flights %>% group_by(month)  %>%
#      summarise(min_delay = min(dep_delay, na.rm=TRUE),
#      mean_delay = mean(dep_delay, na.rm=TRUE),
#      max_delay = max(dep_delay, na.rm=TRUE))
```

4.7.2 Exercise 2

Calculate number of flights (using `n()` operator), mean delay at departure and arrival for flights by origin.
Remember to add `na.rm=TRUE` option to mean calculations.

```
by_origin <- group_by(flights, origin)

summarise(by_origin, n_flights = n(),
          mean_dep_delay = mean(dep_delay, na.rm=TRUE),
          mean_arr_delay = max(arr_delay, na.rm=TRUE))
```

```
## Source: local data frame [3 x 4]
##
##   origin n_flights mean_dep_delay mean_arr_delay
##    (chr)     (int)          (dbl)          (dbl)
## 1    EWR    120835       15.10795           1109
## 2    JFK    111279       12.11216           1272
## 3    LGA    104662       10.34688            915
```

```
# flights %>% group_by(origin)  %>%
#     summarise(n_flights = n(),
#     mean_dep_delay = mean(dep_delay, na.rm=TRUE),
#     mean_arr_delay = max(arr_delay, na.rm=TRUE))
```

4.8 Chain multiple operations (%>%)

4.8.1 Exercise 1

Calculate number of flights, minimum, mean and maximum delay at departure for flights by month.
Remember to add `na.rm=TRUE` option to all calculations.

```
flights %>% group_by(month)  %>%
    summarise(min_delay = min(dep_delay, na.rm=TRUE),
    mean_delay = mean(dep_delay, na.rm=TRUE),
    max_delay = max(dep_delay, na.rm=TRUE))
```

```
## Source: local data frame [12 x 4]
##
##    month min_delay mean_delay max_delay
##    (int)     (dbl)      (dbl)     (dbl)
## 1      1       -30  10.036665      1301
## 2      2       -33  10.816843       853
## 3      3       -25  13.227076       911
## 4      4       -21  13.938038       960
## 5      5       -24  12.986859       878
## 6      6       -21  20.846332      1137
## 7      7       -22  21.727787      1005
## 8      8       -26  12.611040       520
## 9      9       -24   6.722476      1014
## 10    10       -25   6.243988       702
## 11    11       -32   5.435362       798
## 12    12       -43  16.576688       896
```

4.8.2 Exercise 2

Calculate the monthly mean gained time in minutes, where the gained time is defined as the difference between delay at departure and delay at arrival. Remember to add `na.rm=TRUE` option to mean calculations.

```
flights %>% group_by(month)  %>%
  mutate(gain = dep_delay - arr_delay) %>%
  summarise(mean_gain = mean(gain, na.rm=TRUE))
```

```
## Source: local data frame [12 x 2]
##
##    month mean_gain
##    (int)     (dbl)
## 1      1  3.855519
## 2      2  5.147220
## 3      3  7.356713
## 4      4  2.673124
## 5      5  9.370201
## 6      6  4.244284
## 7      7  4.810872
## 8      8  6.529872
## 9      9 10.648649
## 10    10  6.400238
## 11    11  4.958993
## 12    12  1.611806
```

4.8.3 Exercise 3

For each destination, select all days where the mean delay at arrival is greater than 30 minutes. Remember to add `na.rm=TRUE` option to mean calculations.

```
flights %>% group_by(dest)  %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm=TRUE)) %>%
  filter(mean_arr_delay > 30)
```

```
## Source: local data frame [3 x 2]
##
##    dest mean_arr_delay
##   (chr)          (dbl)
## 1   CAE       41.76415
## 2   OKC       30.61905
## 3   TUL       33.65986
```

Chapter 5

Data Visualization with ggplot2

Load ggplot2 package, supposing it is already installed.

```
require(ggplot2)
```

5.1 Data

5.1.1 iris

Almost all the following exercises are based on the iris data, taken from the datasets package. It is a base package so it is already installed and loaded.

```
data("iris")
```

This dataset gives the measurements in centimeters of length and width of sepal and petal, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

iris dataset contains the following variables:

- Sepal.Length: length of iris sepal
- Sepal.Width: width of iris sepal
- Petal.Length: length of iris petal
- Petal.Width: width of iris petal
- Species: species of iris

```
dim(iris)
```

```
## [1] 150   5
```

```
head(iris)
```

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

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

5.1.2 mpg

Some of the exercises are based on `mpg` dataset, taken from the `datasets` package.
It is a base package so it is already installed and loaded.

```
data("mpg")
```

This dataset contains the fuel economy data from 1999 and 2008 for 38 popular models of car.

```
dim(mpg)
```

```
## [1] 234  11
```

```
head(mpg)
```

```
##   manufacturer model displ year cyl      trans drv cty hwy fl   class
## 1         audi    a4   1.8 1999   4   auto(l5)   f  18  29  p compact
## 2         audi    a4   1.8 1999   4 manual(m5)   f  21  29  p compact
## 3         audi    a4   2.0 2008   4 manual(m6)   f  20  31  p compact
## 4         audi    a4   2.0 2008   4   auto(av)   f  21  30  p compact
## 5         audi    a4   2.8 1999   6   auto(l5)   f  16  26  p compact
## 6         audi    a4   2.8 1999   6 manual(m5)   f  18  26  p compact
```

```r
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...
```

5.2 Scatterplot

5.2.1 Exercise 1

a. Generate a scatterplot to analyze the relationship between `Sepal.Width` and `Sepal.Length` variables.

b. Set the size of the point as 3 and their colour (`colour` and `fill` arguments as "green").

```
pl <- ggplot(data = iris, mapping = aes(x=Sepal.Width, y=Sepal.Length)) +
       geom_point(size=3, colour="green", fill="green")
pl
```

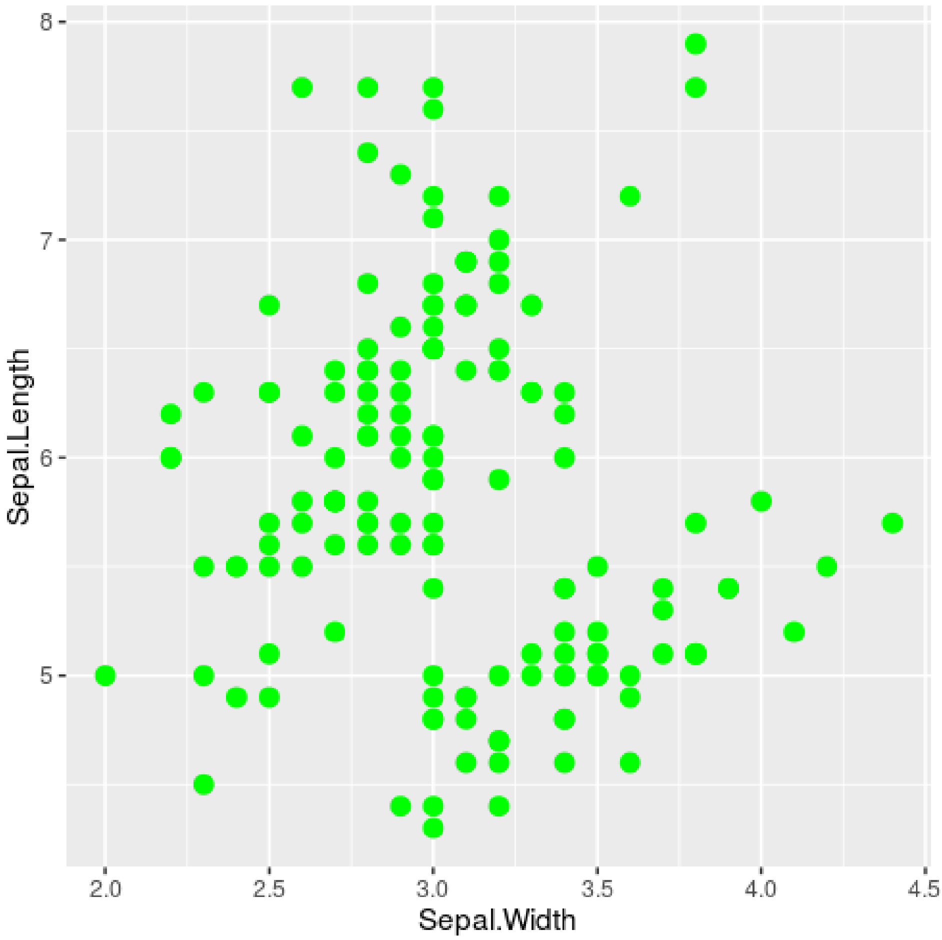

Figure 5.1:

5.2.2 Exercise 2

a. Generate a scatterplot to analyze the relationship between `Petal.Width` and `Petal.Length` variables according to iris species, mapped as `colour` aes.

```
pl <- ggplot(data = iris, mapping = aes(x=Sepal.Width, y=Sepal.Length, colour=Species)) +
        geom_point()
pl
```

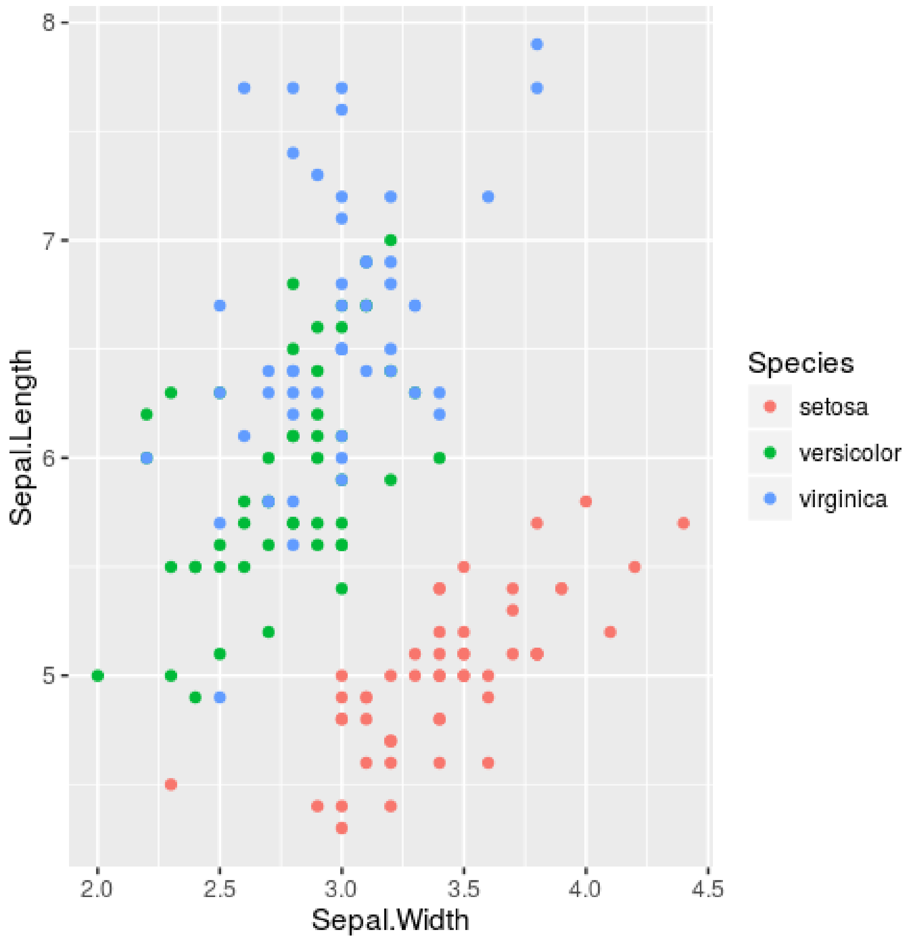

Figure 5.2:

5.3 Box Plot

5.3.1 Exercise 1

a. Build a box plot to compare the differences of sepal width accordingly to the type of iris species.

b. Set the fill of boxes as "#00FFFF", the colour as "#0000FF" and the outlier colours as "red".

c. Add the plot title: "Boxplot of Sepal.Width vs Species"

```
pl <- ggplot(data=iris, aes(x=Species, y=Sepal.Width)) +
  geom_boxplot(fill="#00FFFF", colour="#0000FF", outlier.colour = "red") +
  ggtitle("Boxplot of Sepal.Width vs Species")
pl
```

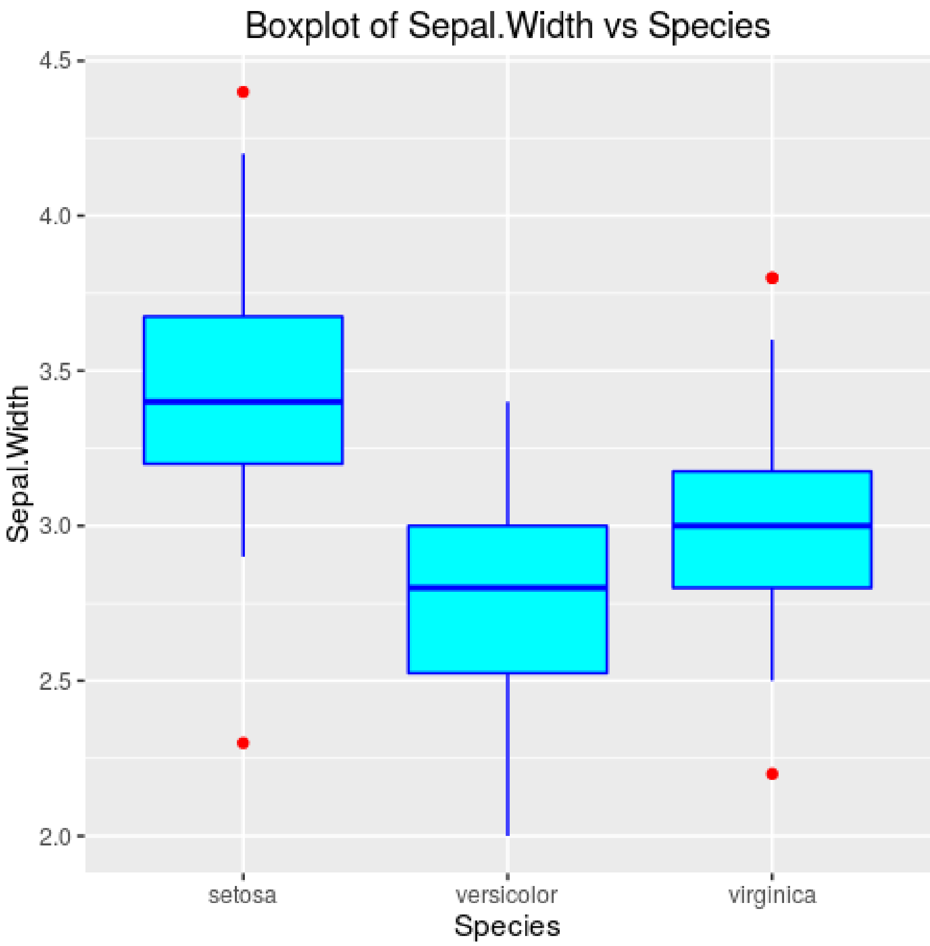
Boxplot of Sepal.Width vs Species
Sepal.Width
4.5
4.0
3.5
3.0
2.5
2.0
setosa
versicolor
virginica
Species

Figure 5.3:

5.4 Histogram

5.4.1 Exercise 1

a. Represent the distribution of sepal length with an histogram.

b. Set bins fill as “hotpink” and colour as “deeppink”.

c. Set the number of bins as 15.

```
pl <- ggplot(data=iris, aes(x=Sepal.Length)) +
    geom_histogram(fill="hotpink", colour="deeppink", bins=15)
pl
```

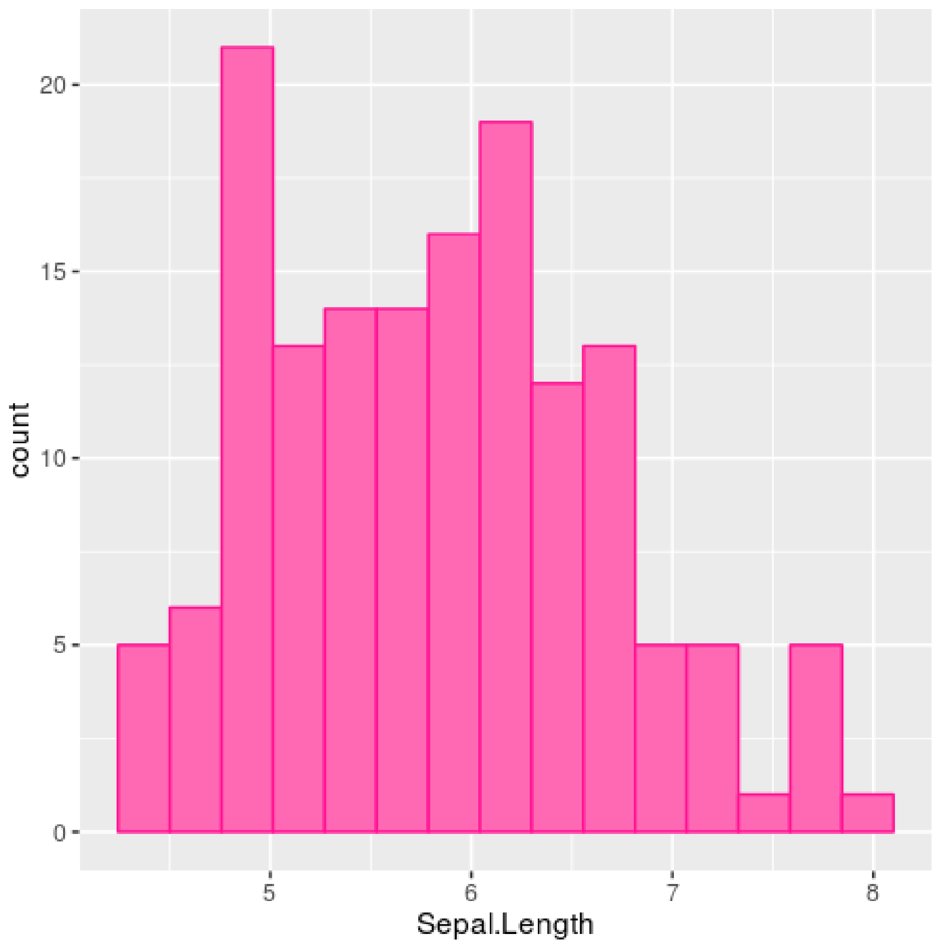

Figure 5.4:

5.5 Lineplot

5.5.1 Exercise 1

Let us suppose that the observations on flowers are taken along time, so let us consider the following dataset:

```
require(dplyr)
iris2 <- iris %>% mutate(time=1:150)
```

a. Build a line plot to visualize the Sepal.Length along time.

```
ggplot(data = iris2, mapping = aes(y=Sepal.Width, x= time)) + geom_line()
```

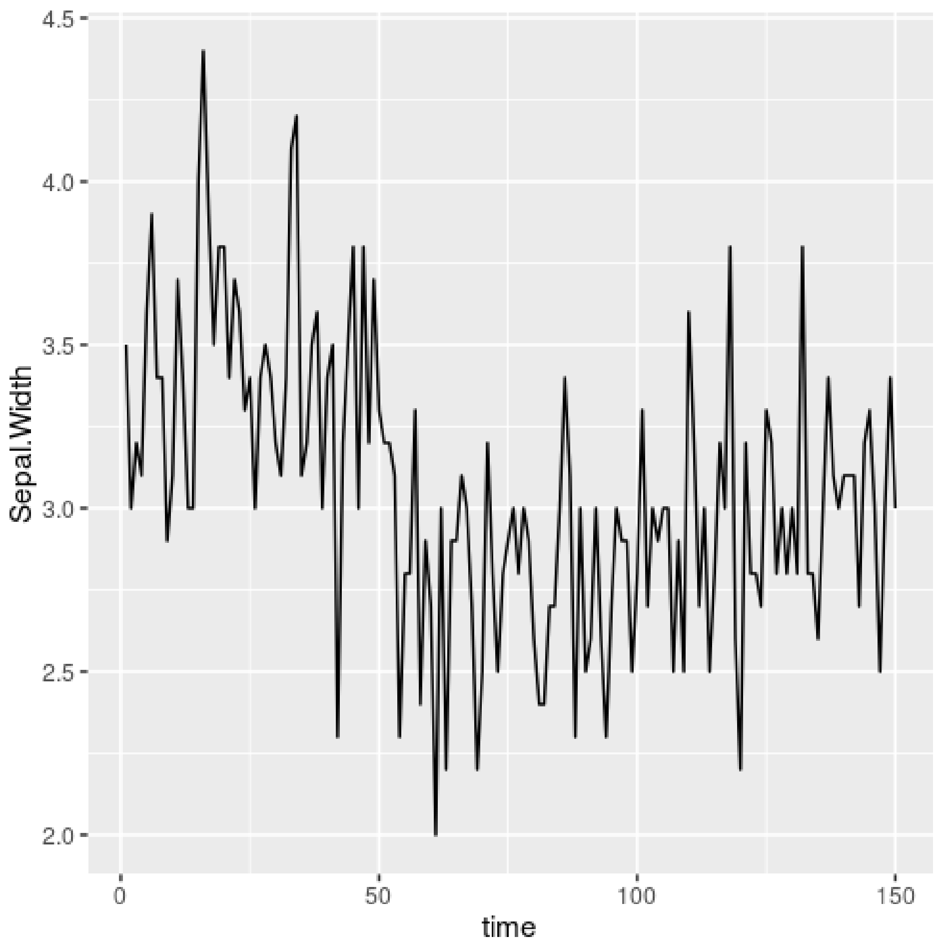

Figure 5.5:

5.5.2 Exercise 2

Let us suppose that the observations on flowers are taken along time, so let us consider the following dataset:

```
iris3 <- iris %>% mutate(time=rep(1:50, times=3))
```

a. Build a line plot to visualize the `Sepal.Length` along time, according to the `Species`.
b. Set linetype as "twodash".

```
ggplot(data = iris3, mapping = aes(y=Sepal.Length, x= time, colour=Species)) +
  geom_line(linetype=6)
```

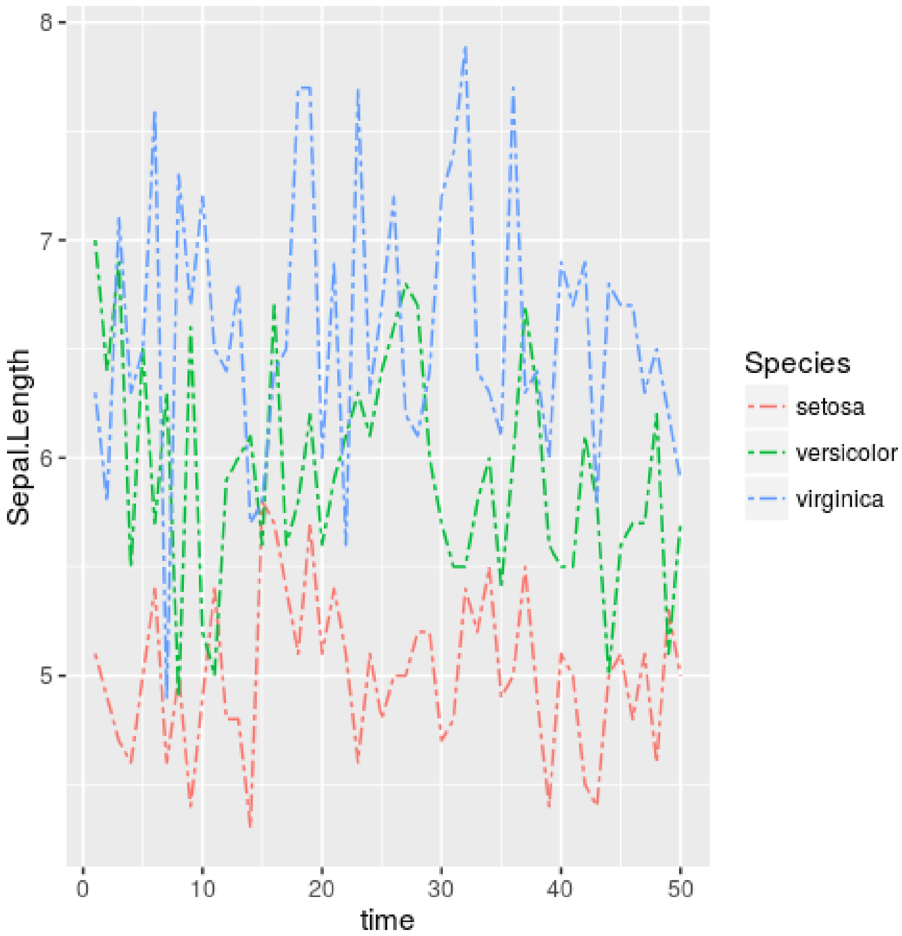

Figure 5.6:

5.6 Bar graph

Let us consider `mpg` dataset.

5.6.1 Exercise 1

a. Represent graphically with a bar graph, how many cars there are for each class.
b. Represent horizontal bar and set bar width as 0.6

```
pl <- ggplot(mpg, aes(class)) +
      coord_flip() +
      geom_bar(width=0.6)
pl
```

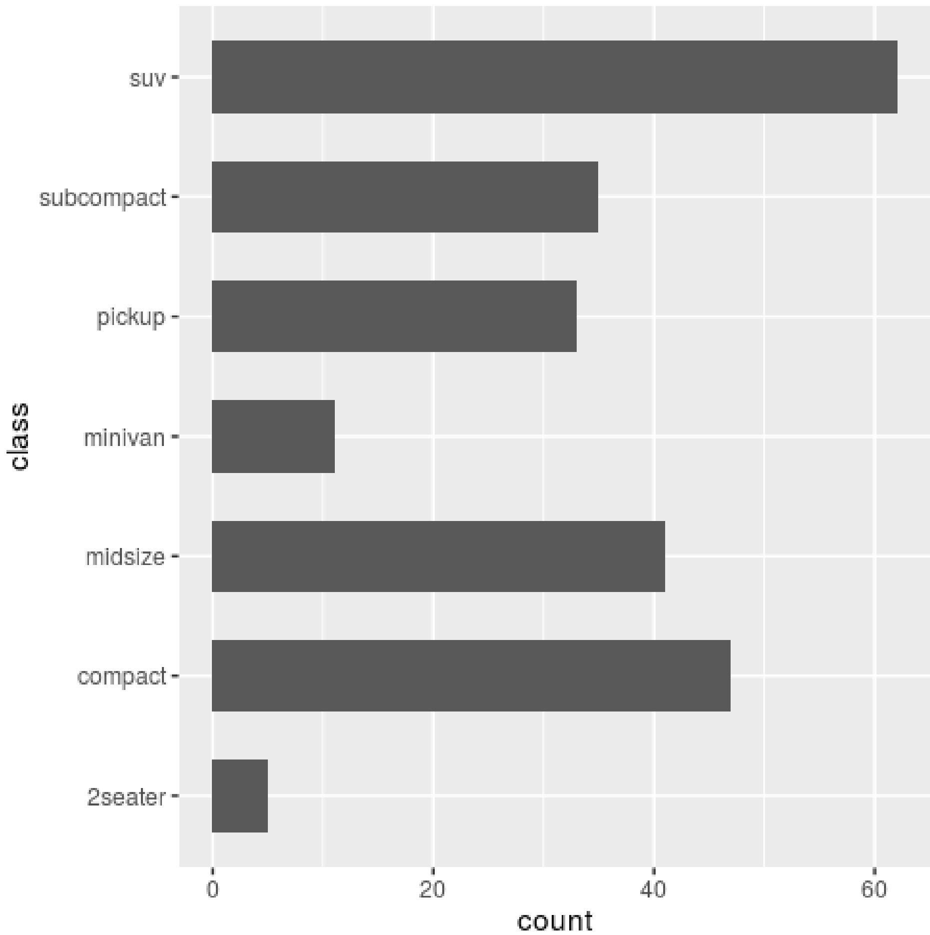

Figure 5.7:

5.6.2 Exercise 2

a. Represent graphically with a bar graph, how many cars there are for each class according to manifacturer.

```
pl <- ggplot(mpg, aes(class, fill=manufacturer)) +
  geom_bar()
pl
```

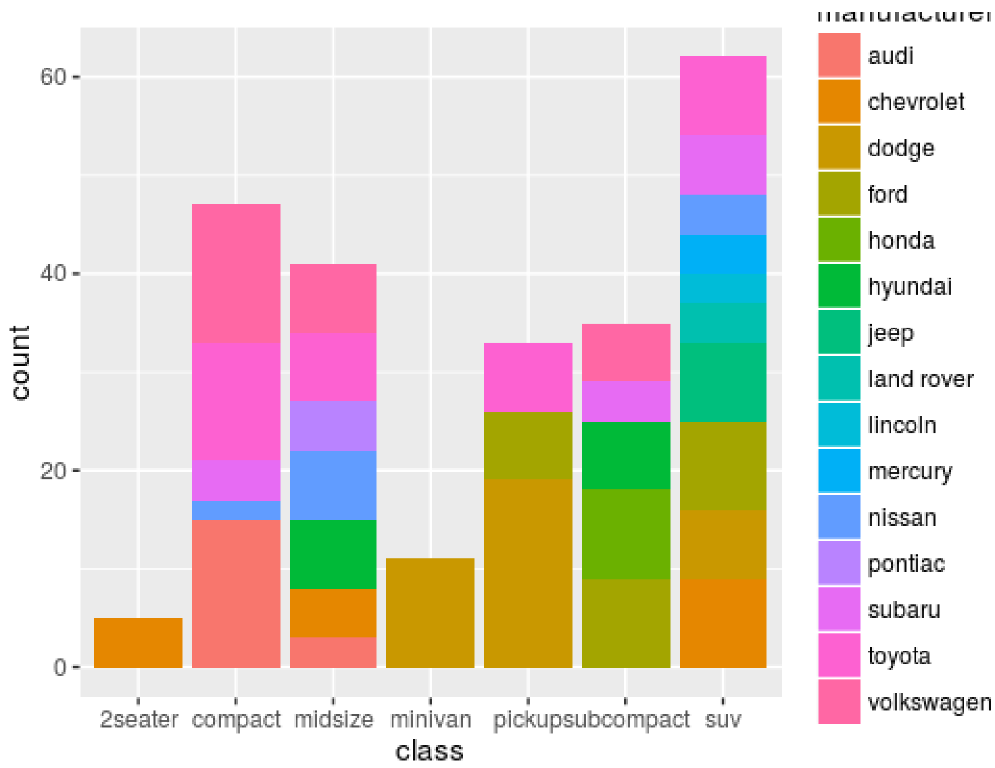

Figure 5.8:

b. Represent graphically with a bar graph, the distribution of manifacturerfor each class.

```
pl <- ggplot(mpg, aes(class, fill=manufacturer)) +
  geom_bar(position ="fill")
pl
```

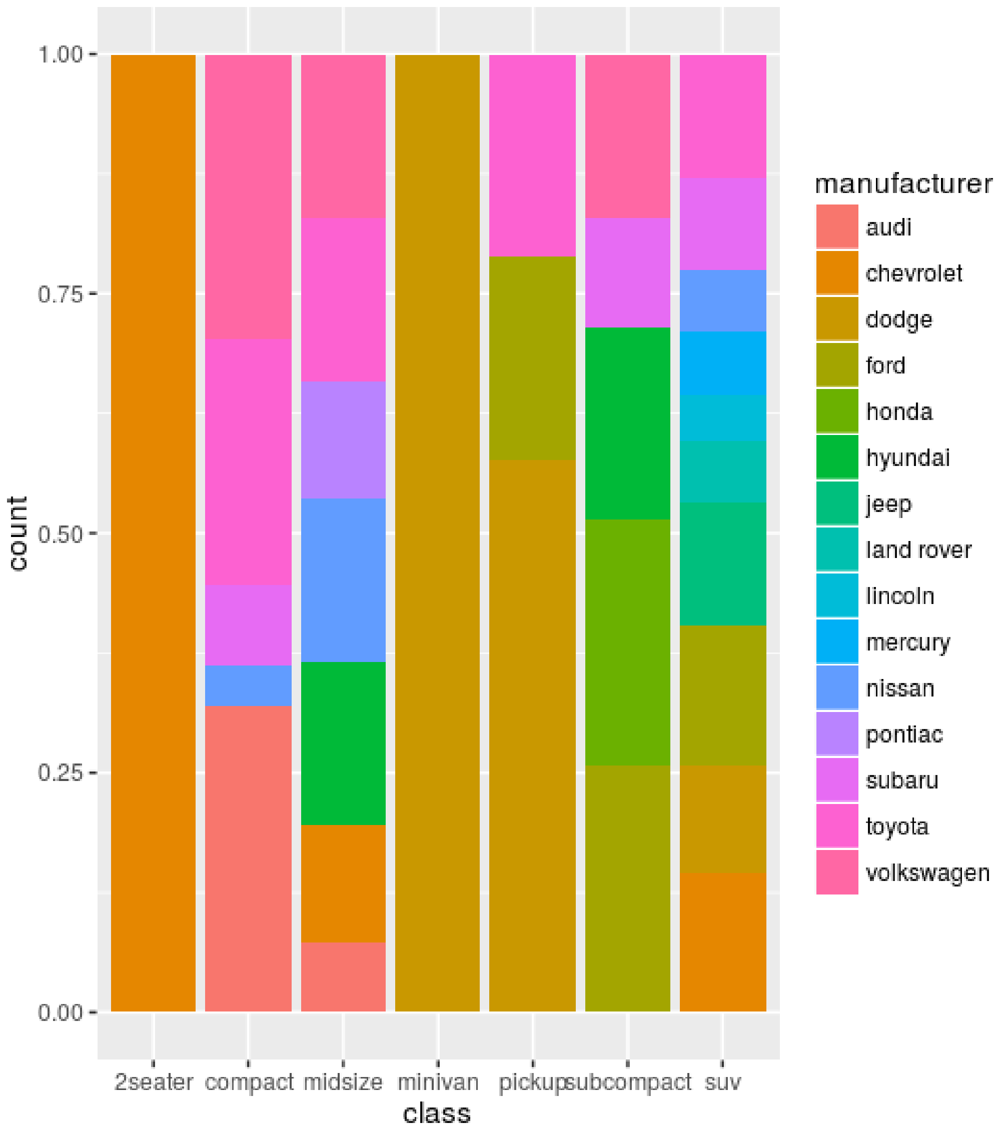

Figure 5.9:

Chapter 6

Writing R functions

6.1 Writing R functions

6.1.1 Exercise 1

Write a function, named `compute_summary`, which computes: sum, subtraction, multiplication and division of two numbers. The function arguments should be the two numbers, named as: `x` and `y`. The function should return all amounts computed.

```
compute_summary <- function(x, y){
  sum_op <- x+y
  sub_op <- x-y
  mul_op <- x*y
  div_op <- x/y
  return(list(sum_op=sum_op, sub_op=sub_op, mul_op=mul_op, div_op=div_op))
}

compute_summary(x=4, y=2)

## $sum_op
## [1] 6
##
## $sub_op
## [1] 2
##
## $mul_op
## [1] 8
##
## $div_op
## [1] 2


compute_summary(x=3, y=7)
```

```
## $sum_op
## [1] 10
##
## $sub_op
## [1] -4
##
## $mul_op
## [1] 21
##
## $div_op
## [1] 0.4285714
```

6.1.2 Exercise 2

Write a function, named `compute_gain`, which computes the income by multiplying the amount produced for sale price and then computes the gain by subtracting the costs to income.
The function arguments should be: `amount`, `price`, and `costs`; `price` should have a default value equal to 5. The function should return the gain.

```
compute_gain <- function(amount, costs, price=5){
  income = amount * price
  gain = income - costs
  return(gain)
}

compute_gain(amount = 40, costs = 50)
```

```
## [1] 150
```

```
compute_gain(amount = 100,costs = 70,price = 1)
```

```
## [1] 30
```