Data Programming Course Exercises

October 18, 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 Manipulation with dplyr*
- *Data Visualization with ggplot2*
- *Writing R functions*

Chapter 2

Data Manipulation with dplyr

Load dplyr package, supposing it is already installed.

```
require(dplyr)
```

2.1 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")
```

2.1.1 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 ...
```

2.2 Select

2.2.1 Exercise 1

Extract the following information:

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

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

```
## # A tibble: 336,776 × 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
## # ... with 336,766 more rows
```

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

2.2.2 Exercise 2

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

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

```
## # A tibble: 336,776 × 17
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
```

```
## 5   2013     1     1      554            600        -6      812            837
## 6   2013     1     1      554            558        -4      740            728
## 7   2013     1     1      555            600        -5      913            854
## 8   2013     1     1      557            600        -3      709            723
## 9   2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 9 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, time_hour <dttm>
```

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

2.2.3 Exercise 3

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

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

```
## # A tibble: 336,776 × 1
##    tail_num
##       <chr>
## 1    N14228
## 2    N24211
## 3    N619AA
## 4    N804JB
## 5    N668DN
## 6    N39463
## 7    N516JB
## 8    N829AS
## 9    N593JB
## 10   N3ALAA
## # ... with 336,766 more rows
```

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

2.3 Filter

2.3.1 Exercise 1

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

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

```
## # A tibble: 5 × 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     9      641            900      1301     1242           1530
## 2  2013     1    10     1121           1635      1126     1239           1810
## 3  2013     6    15     1432           1935      1137     1607           2120
## 4  2013     7    22      845           1600      1005     1044           1815
## 5  2013     9    20     1139           1845      1014     1457           2210
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

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

2.3.2 Exercise 2

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

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

```
## # A tibble: 5 × 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     9      641            900      1301     1242           1530
## 2  2013     1    10     1121           1635      1126     1239           1810
## 3  2013     6    15     1432           1935      1137     1607           2120
## 4  2013     7    22      845           1600      1005     1044           1815
## 5  2013     9    20     1139           1845      1014     1457           2210
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

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

2.3.3 Exercise 3

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

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

```
## # A tibble: 3,973 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
```

```
## 2    2013     1     1      739            739          0      1104           1038
## 3    2013     1     1      908            908          0      1228           1219
## 4    2013     1     1     1044           1045         -1      1352           1351
## 5    2013     1     1     1205           1200          5      1503           1505
## 6    2013     1     1     1356           1350          6      1659           1640
## 7    2013     1     1     1527           1515         12      1854           1810
## 8    2013     1     1     1620           1620          0      1945           1922
## 9    2013     1     1     1725           1720          5      2045           2021
## 10   2013     1     1     1959           2000         -1      2310           2307
## # ... with 3,963 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

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

2.4 Arrange

2.4.1 Exercise 1

Sort the flights in chronological order.

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

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
## 2   2013     1     1      533            529         4      850            830
## 3   2013     1     1      542            540         2      923            850
## 4   2013     1     1      544            545        -1     1004           1022
## 5   2013     1     1      554            600        -6      812            837
## 6   2013     1     1      554            558        -4      740            728
## 7   2013     1     1      555            600        -5      913            854
## 8   2013     1     1      557            600        -3      709            723
## 9   2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

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

2.4.2 Exercise 2

Sort the flights by decreasing arrival delay.

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

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     9      641            900      1301     1242           1530
## 2   2013     6    15     1432           1935      1137     1607           2120
## 3   2013     1    10     1121           1635      1126     1239           1810
## 4   2013     9    20     1139           1845      1014     1457           2210
## 5   2013     7    22      845           1600      1005     1044           1815
## 6   2013     4    10     1100           1900       960     1342           2211
## 7   2013     3    17     2321            810       911      135           1020
## 8   2013     7    22     2257            759       898      121           1026
## 9   2013    12     5      756           1700       896     1058           2020
```

```
## 10  2013     5     3     1133           2055       878     1250             2215
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

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

2.4.3 Exercise 3

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

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

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1    10     1121           1635      1126     1239           1810
## 2   2013    12     5      756           1700       896     1058           2020
## 3   2013     5     3     1133           2055       878     1250           2215
## 4   2013    12    19      734           1725       849     1046           2039
## 5   2013    12    17      705           1700       845     1026           2020
## 6   2013    11     3      603           1645       798      829           1913
## 7   2013     2    24     1921            615       786     2135            842
## 8   2013    10    14     2042            900       702     2255           1127
## 9   2013     7    21     1555            615       580     1955            910
## 10  2013     7     7     2123           1030       653       17           1345
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

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

2.5 Mutate

2.5.1 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)
```

```
## # A tibble: 336,776 × 20
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
## 2   2013     1     1      533            529         4      850            830
## 3   2013     1     1      542            540         2      923            850
## 4   2013     1     1      544            545        -1     1004           1022
## 5   2013     1     1      554            600        -6      812            837
## 6   2013     1     1      554            558        -4      740            728
## 7   2013     1     1      555            600        -5      913            854
## 8   2013     1     1      557            600        -3      709            723
## 9   2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   speed <dbl>
```

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

2.5.2 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))
```

```
## # A tibble: 336,776 × 21
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
## 2   2013     1     1      533            529         4      850            830
## 3   2013     1     1      542            540         2      923            850
## 4   2013     1     1      544            545        -1     1004           1022
## 5   2013     1     1      554            600        -6      812            837
## 6   2013     1     1      554            558        -4      740            728
## 7   2013     1     1      555            600        -5      913            854
## 8   2013     1     1      557            600        -3      709            723
## 9   2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 13 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
```

```
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   gain <dbl>, gain_per_hour <dbl>
```

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

2.6 Summarise

2.6.1 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))
```

```
## # A tibble: 1 × 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))
```

2.7 Group_by

2.7.1 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))
```

```
## # A tibble: 12 × 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))
```

2.7.2 Exercise 2

Calculate number of flights (using `n()` operator), mean delay at departure and at 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))
```

```
## # A tibble: 3 × 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))
```

2.8 Chain multiple operations (%>%)

2.8.1 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))
```

```
## # A tibble: 12 × 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
```

2.8.2 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))
```

```
## # A tibble: 12 × 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
```

2.8.3 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)
```

```
## # A tibble: 3 × 2
##    dest mean_arr_delay
##   <chr>          <dbl>
## 1   CAE       41.76415
## 2   OKC       30.61905
## 3   TUL       33.65986
```

Chapter 3

Data Visualization with `ggplot2`

Load `ggplot2` package, supposing it is already installed.

```
require(ggplot2)
```

3.1 Data

3.1.1 iris

Almost all the following exercises are based on the `iris` dataset, 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
```

```r
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
```

```r
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 ...
```

3.1.2 mpg

Some of the exercises are based on `mpg` dataset, taken from the `ggplot2` package.

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

This dataset contains the fuel economy data from 1999 and 2008 for 38 popular models of car. `mpg` dataset contains the following variables:

- `manufacturer`
- `model`
- `displ`: engine displacement, in litres
- `year`
- `cyl`: number of cylinders
- `trans`: type of transmission
- `drv`: drivetrain type, f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- `cty`: city miles per gallon

- `hwy`: highway miles per gallon
- `fl`: fuel type

```
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
```

```
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" ...
```

3.2 Scatterplot

Let us consider `iris` dataset.

3.2.1 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
```

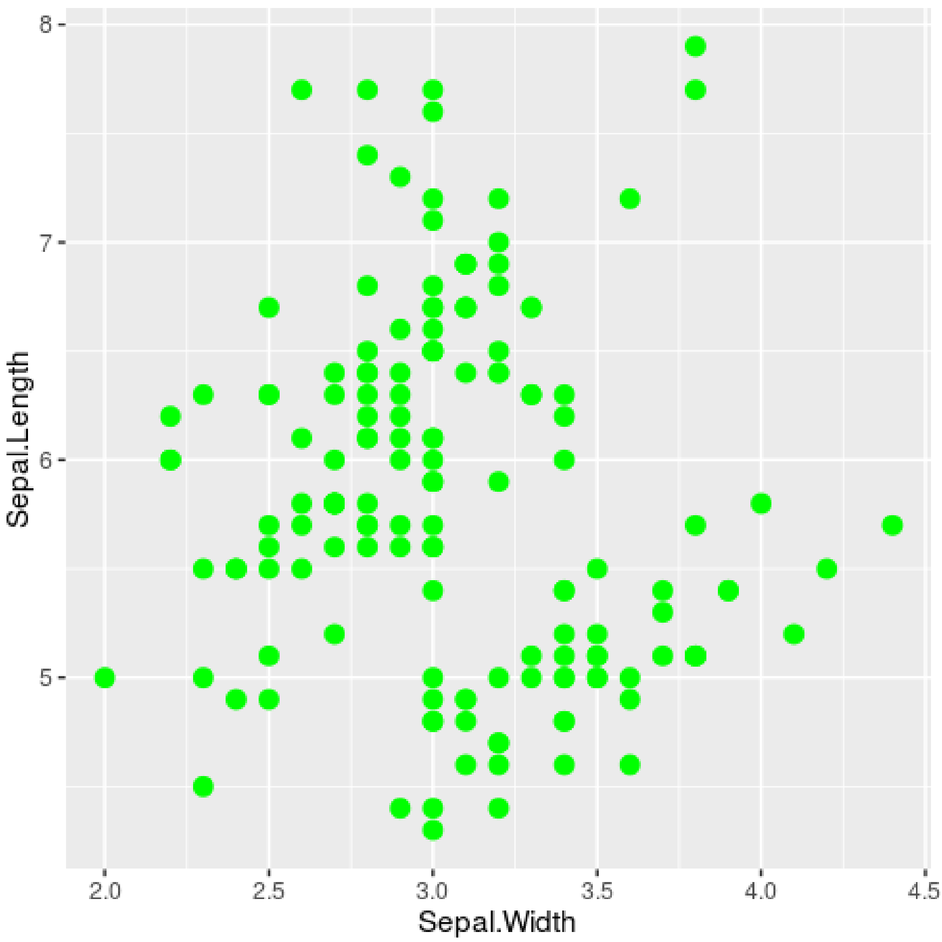

Figure 3.1:

3.2.2 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
```

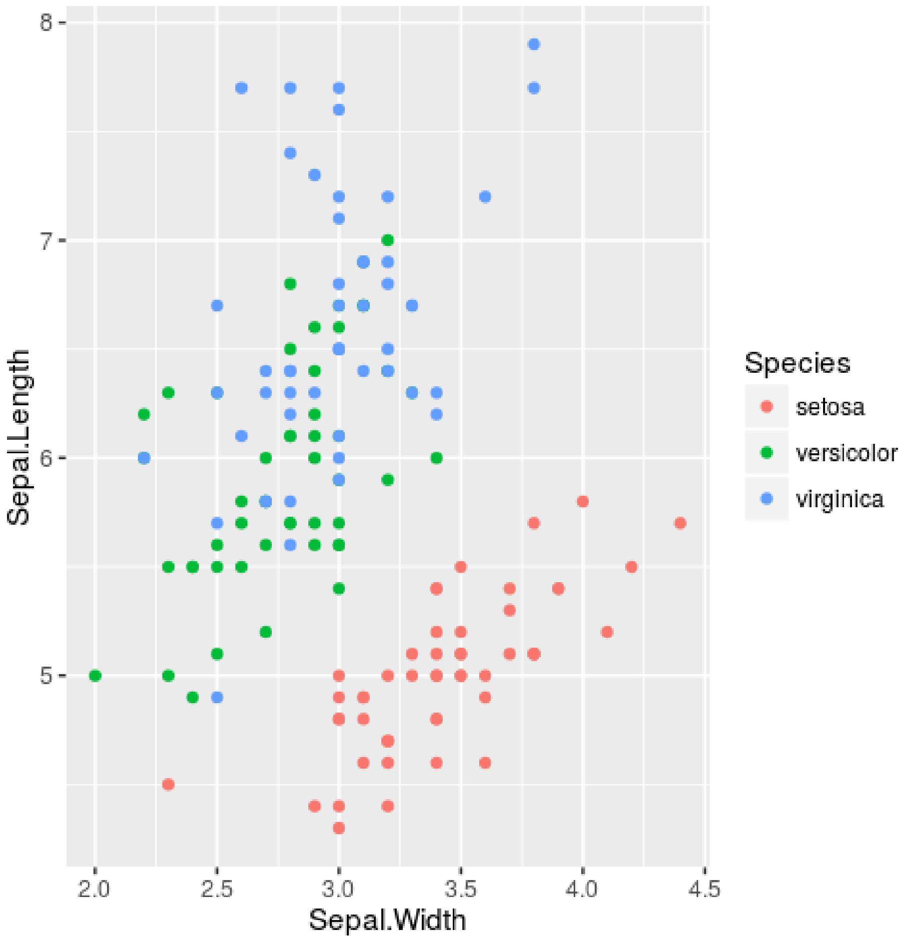

Figure 3.2:

3.3 Box Plot

Let us consider `iris` dataset.

3.3.1 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 colour of boxes as “#00FFFF”, the lines colour of boxes as “#0000FF” and the outliers colour 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
```

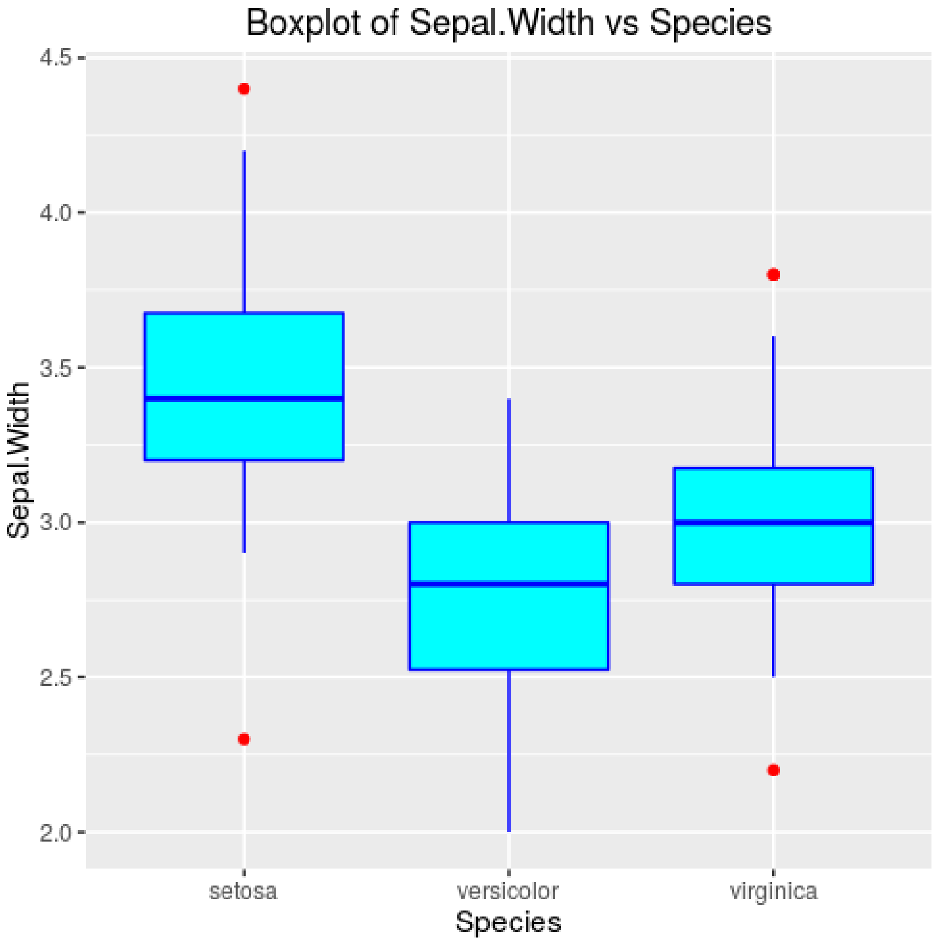
Boxplot of Sepal.Width vs Species
Sepal.Width
4.5
4.0
3.5
3.0
2.5
2.0
setosa
versicolor
virginica
Species

Figure 3.3:

3.4 Histogram

Let us consider `iris` dataset.

3.4.1 Exercise 1

a. Represent the distribution of `Sepal_Length` variable with an histogram.
b. Set bins fill colour as "hotpink" and bins line 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
```

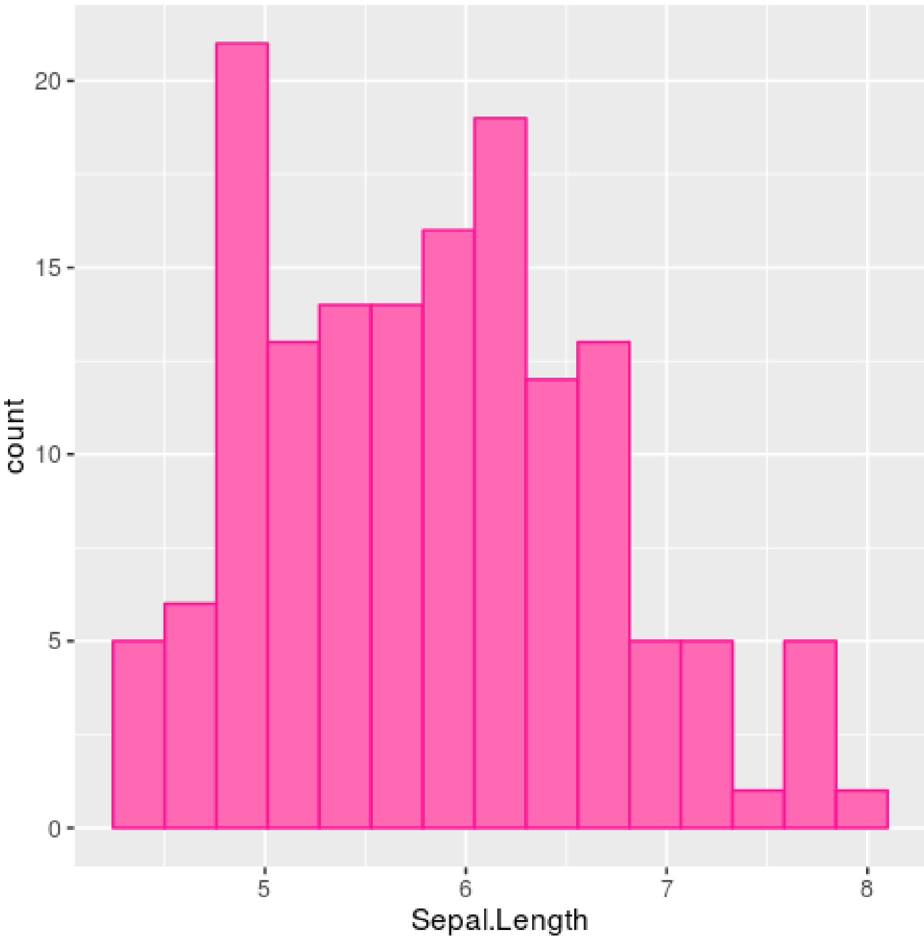

Figure 3.4:

3.5 Line graph

3.5.1 Exercise 1

Let us suppose that the observations on `iris` are taken along time.
So let us consider the following dataset, named `iris2`, in which `time` variable is added:

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

a. Build a line graph to visualize the measures of `Sepal.Length` variable along time.

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

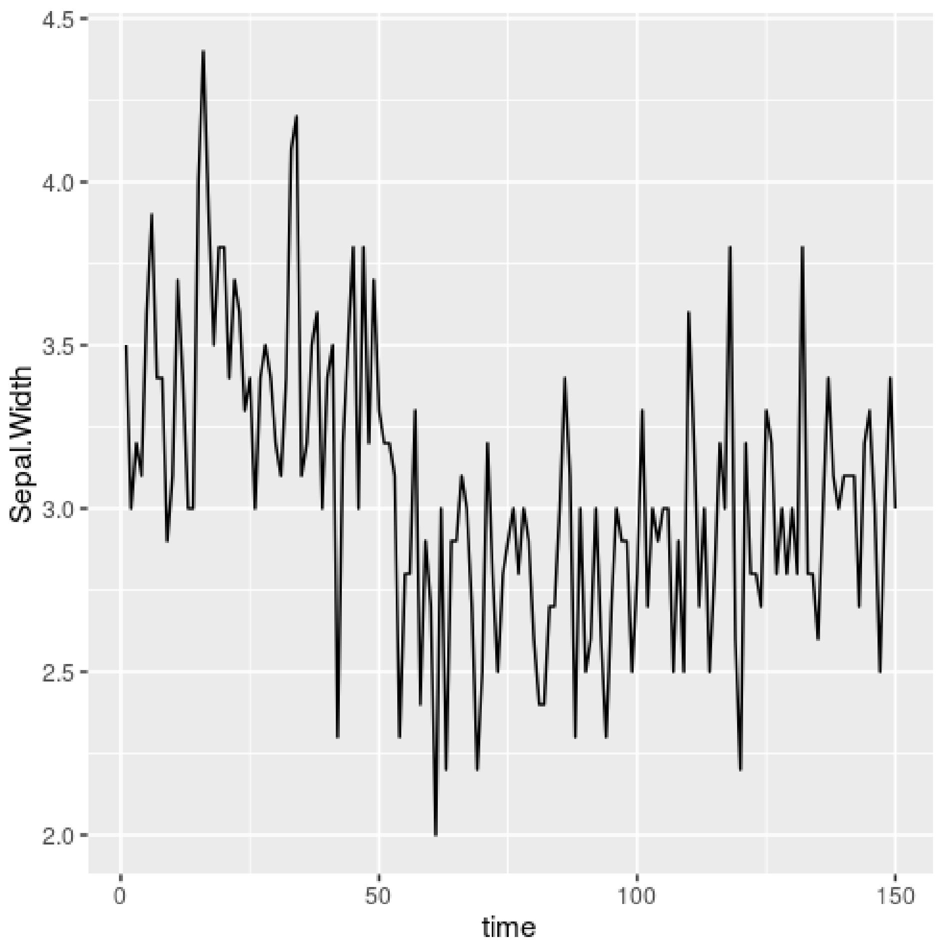

Figure 3.5:

3.5.2 Exercise 2

Let us suppose that the observations on `iris` are taken along time.
So let us consider the following dataset, named `iris3`, in which `time` variable is added:

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

a. Build a line graph to visualize the measures of `Sepal.Length` variable along time, according to the `Species` variable, mapped as `colour` aes.

b. Set linetype as "twodash".

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

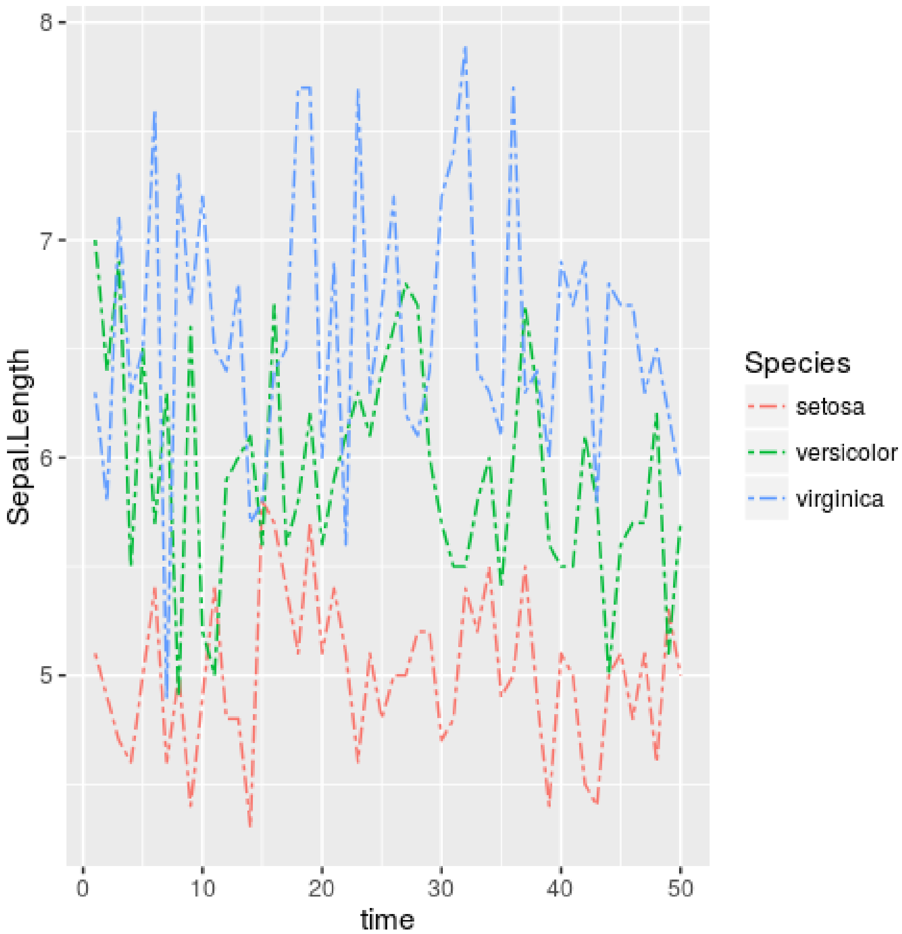

Figure 3.6:

3.6 Bar graph

Let us consider `mpg` dataset.

3.6.1 Exercise 1

a. Represent graphically with a bar graph how many cars there are for each class.

b. Represent horizontal bars and set bars width as 0.6.

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

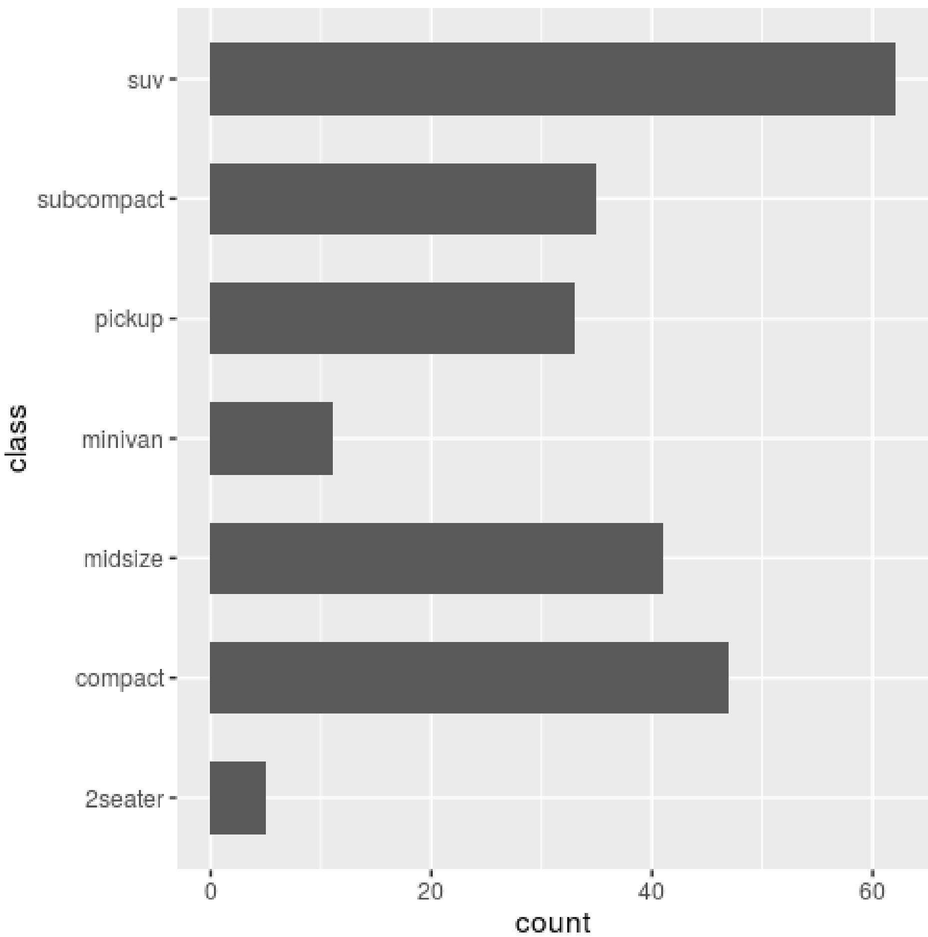

Figure 3.7:

3.6.2 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
```

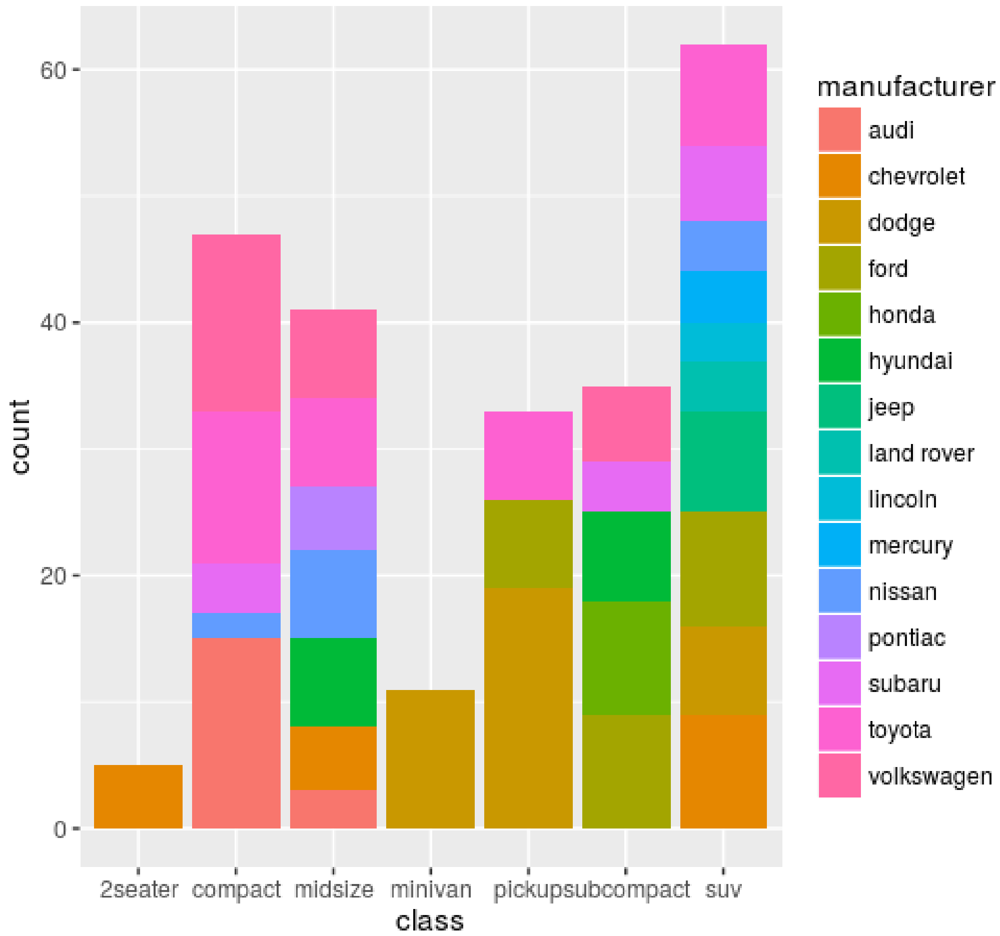

Figure 3.8:

b. Represent graphically with a bar graph, the distribution of manifacturerfor each class (set `position` argument of `geom_bar`).

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

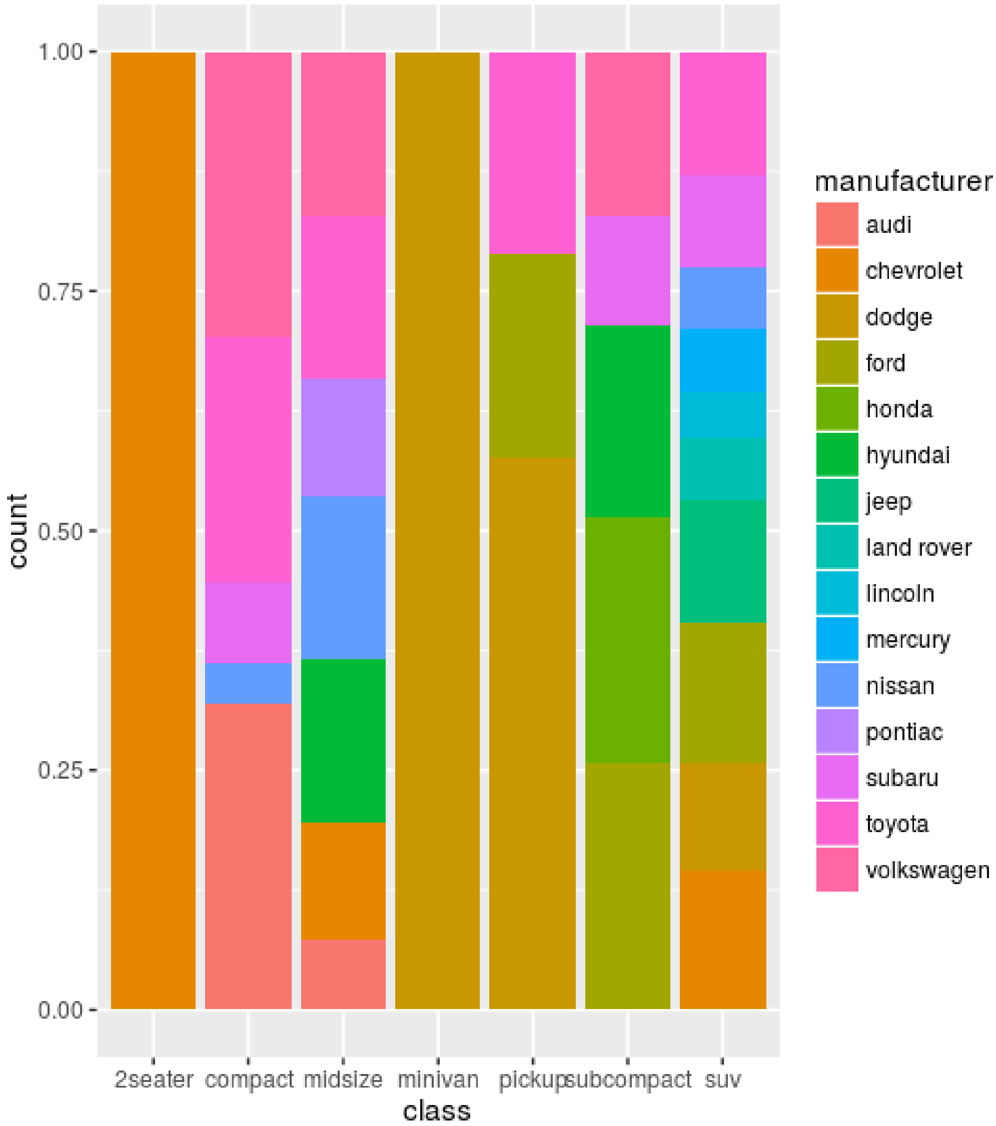

Figure 3.9:

Chapter 4

Writing R functions

4.1 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
```

4.2 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
```