Exploring data using R

Kamarul Imran Musa, Wan Nor Arifin

April 21, 2018

Contents

Chapter 1

Introduction to R

This chapter introduces readers to the basics of working with data in R. We will start with installing R in your computer and getting familiar with RStudio interface. These will be followed by the basics of handling data in R.

1.1 R and RStudio

1.1.1 Installation of R

- The latest version of R is R version 3.4.4 (2018-03-15), Someone to Lean On.
- R is available for Windows, Mac OS and Linux.
- The installation files can be downloaded from https://cran.r-project.org/.
- Users can install different versions of R in a same machine or computer.
- There is no need to uninstall if you want to upgrade the currently installed R.

1.1.2 Starting R

Double click on R icon and you should get this

You should see an R console.

1.1.3 Installation of RStudio

RStudio installation files can be downloaded from http://www.rstudio.com/. First, make sure you have RStudio successfully installed.

1.1.3.1 Starting RStudio

You can double click on RStudio icon and you will see this:

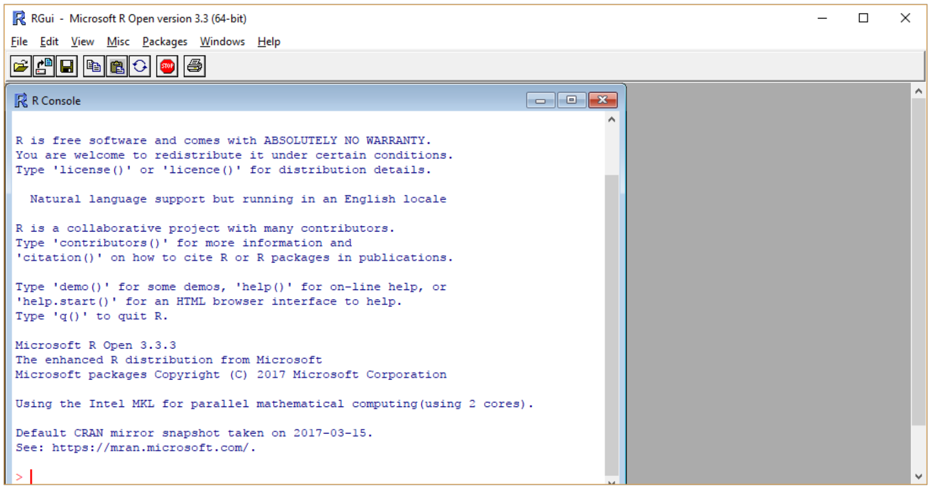


Figure 1.1: R console

1.1.3.2 Why RStudio?

- Working with R console is alright.
- But for many people, they prefer to communicate with R using a graphical user interface (GUI).
- RStudio is the popular GUI and intergrated developement environment (IDE) for R.
- Other R IDE includes Microsoft R

Check this links for more info:

1. RStudio https://www.rstudio.com/
2. Microsoft R http://blog.revolutionanalytics.com/2016/01/microsoft-r-open.html

1.1.3.3 RStudio interface

You should be able to see 4 panes in the layout. You should see that

1. Console - the lower left pane. It tells you about your R information.
2. Source - the upper left pane. It shows the active files.
3. Environment and History - the upper right pane. It shows the currently loaded data files and values, and command history.
4. Miscellaneous - the lower right pane. It contains most important tabs, which are Files, Plots, Packages, Help and Viewer. It list file names, show plots, show packages, display help document and view outputs.

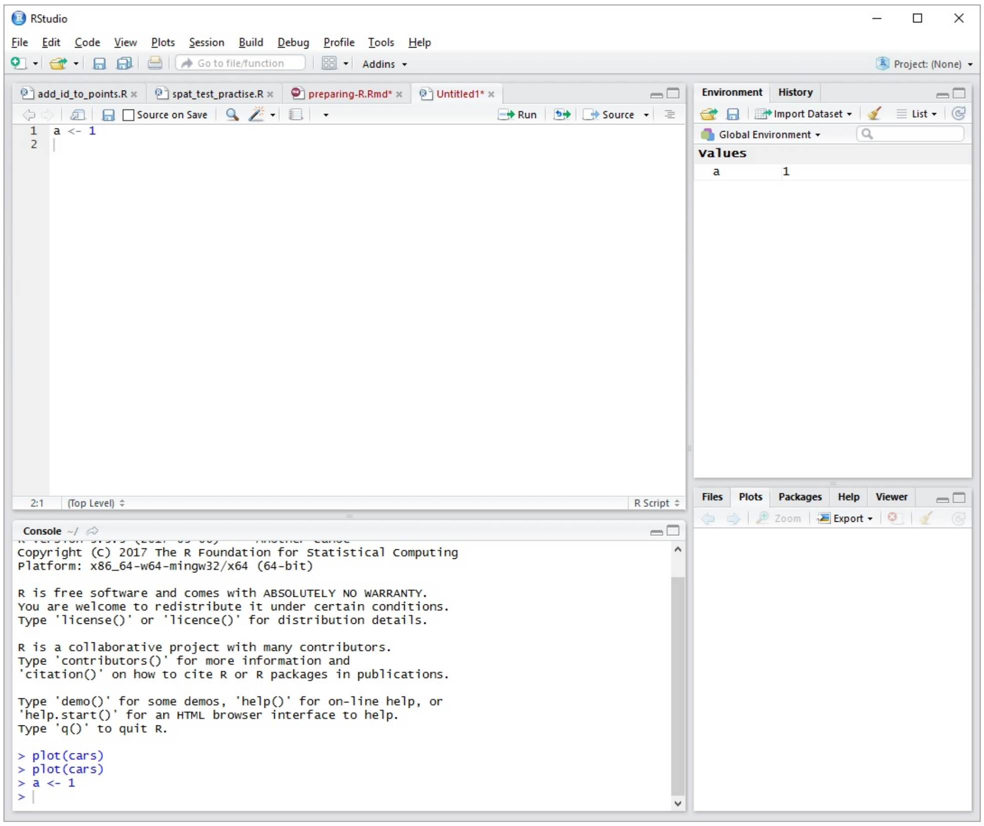


Figure 1.2: RStudio

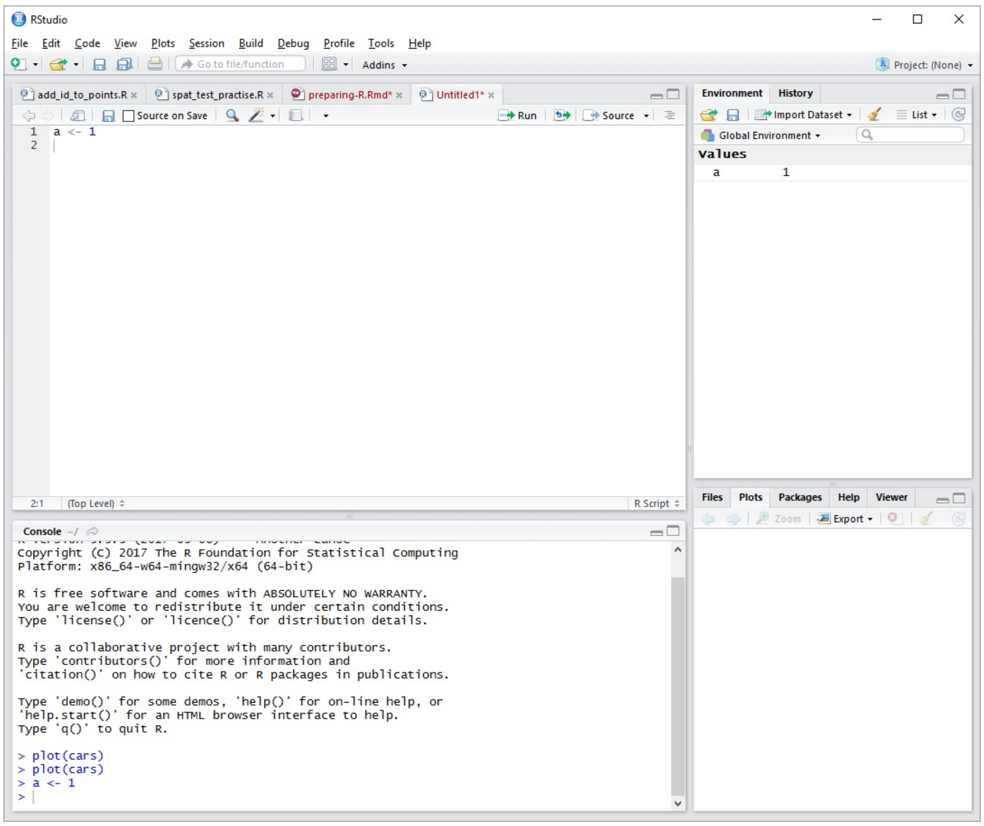


Figure 1.3: Panes in RStudio

1.2 Functions and objects

Before we start, there are a number of basics that you must know to understand the syntax in R. These are functions and objects.

1.2.1 Functions

R commands are in form of `function()`. You can think of MS Excel function. Inside a function, there will be a number of options. We will see this as we go through the examples later.

1.2.2 Objects

Object is a sort of like container. You assign an object by giving it a name on left side of `<-` or =. For the sake of consistency, we will use `<-` throughout, although = is perfectly fine (some might argue about this though). # - variable, data (data frame, matrix, list)

```
x <- 1
y = 2
z <- x + y
z  # type object name, you'll get the value
```

```
## [1] 3
```

But later you will see that = is used to set values for options of a function, i.e. inside the bracket after the function's name. For example, `function(option1 = value, option2 = value, ...)`. Thus, some prefer using `<-` to avoid confusion with = for options.

1.3 Working with packages

1.3.1 About packages

R uses packages to perform its tasks.

There are two common packages:

1. `base` packages
2. `user-contributed` packages

- The base packages come with the installation of R
- The base package provides basic but adequate functions to perform many standard data management, visualization and analysis.
- However, user needs to install user-contributed packages if they need to perform functions (tasks) not available in the base package
- User-contributed packages allow users to perform more advanced and more complicated functions
- There are more than 10200 packages as of March 2017

For a complete list of packages, see https://cran.r-project.org/web/packages/

1.3.2 Package installation

You can install user-contributed packages through:

1. Internet (to cran)
2. Github packages
3. Local zip files

We will learn to install a few small packages.

Basically, a function to install a package will look like this

```
install.packages("package.name")
```

To install a package, saya `car` 1. put your cursor in the CONSOLE pane 2. type the codes below

```
install.packages("car")
```

3. press Ctrl + ENTER

1.3.3 Loading packages

Basically, to utilize a package, it has to be loaded using `library()` function,

```
library("package.name")
```

For example, we load the newly installed `car` package

```
library("car")
```

1.4 Working directory

In general, R reads and saves data and other files into a working directory. Therefore, a user must create or specify the working directory to work with R. This is a good practice.

A working directory:

1. stores all the outputs such as the plots, html files, pdf files
2. contains your data

Creating a working directory is a simple BUT an important step.

Unfortunately, many users do not pay attention to this and forget to set it. So, remember, this is a very important step to work in R.

1.4.1 Setting a working directory

To set your working directory:

1. Go back to RStudio's Miscellaneous pane.
2. In the Files tab, click ...
3. Navigate to the folder containing your data or any folder you want to work in.
4. Click *More*
5. Click *Set as working directory*

or simply use `setwd` function to do so.

```
setwd("path to your folder")
```

for example in Windows

```
setwd("C:/myfolder")
```

or in Mac OS/Linux

```
setwd("~/myfolder")
```

1.5 Data management

This section is concerned with reading data from dataset and displaying data.

1.5.1 Reading data set

Easiest is to read .csv file,

```
data <- read.csv("cholest.csv")
```

For SPSS and STATA files, we need `foreign` package,

```
library("foreign")
data <- read.spss("cholest.sav", as.data.frame = TRUE)
data <- read.dta("cholest.dta", convert.factors = TRUE)
```

For Excel file, we need `readxl` package,

```
library("readxl")
data <- read_excel("cholest.xlsx", sheet = 1)
```

1.5.2 Viewing data set

Easy, just type the name,

```
data
```

Nicer, using `View()`

```
View(data)
```

View only the first six observations,

```
head(data)
```

```
##   chol age exercise sex categ
## 1  6.5  38        6   1     0
## 2  6.6  35        5   1     0
## 3  6.8  39        6   1     0
## 4  6.8  36        5   1     0
## 5  6.9  31        4   1     0
## 6  7.0  38        4   1     0
```

and the last six observations,

```
tail(data)
```

```
##    chol age exercise sex categ
## 75  9.4  45        4   0     2
## 76  9.5  52        4   0     2
## 77  9.6  35        4   0     2
## 78  9.8  43        3   0     2
## 79  9.9  47        3   0     2
## 80 10.0  44        3   0     2
```

Important tasks

```
dim(data)
str(data)
names(data)
```

1.5.3 Exporting data set from R

You can also export data into various formats using similar packages.

For example,

1. to export data into a *comma separated version* (.csv) file, we can use `write.csv` function.
2. to export data into stata format, we can use `write.dta` function

```
write.csv(data, 'data.csv')
write.dta(data, 'data.dta')
```

1.6 More about data management

In this section, we will deal with more advanced data management (subsetting, recoding and creating new variables) and direct data entry (especially useful for tables).

Let say we use `cholest.csv`,

```r
data <- read.csv("cholest.csv")
dim(data)
```

```
## [1] 80  5
```

```r
str(data)
```

```
## 'data.frame':    80 obs. of  5 variables:
##  $ chol    : num  6.5 6.6 6.8 6.8 6.9 7 7 7.2 7.2 7.2 ...
##  $ age     : int  38 35 39 36 31 38 33 36 40 34 ...
##  $ exercise: int  6 5 6 5 4 4 5 5 4 6 ...
##  $ sex     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ categ   : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
names(data)
```

```
## [1] "chol"     "age"      "exercise" "sex"      "categ"
```

```r
head(data); tail(data)
```

```
##   chol age exercise sex categ
## 1  6.5  38        6   1     0
## 2  6.6  35        5   1     0
## 3  6.8  39        6   1     0
## 4  6.8  36        5   1     0
## 5  6.9  31        4   1     0
## 6  7.0  38        4   1     0
```

```
##    chol age exercise sex categ
## 75  9.4  45        4   0     2
## 76  9.5  52        4   0     2
## 77  9.6  35        4   0     2
## 78  9.8  43        3   0     2
## 79  9.9  47        3   0     2
## 80 10.0  44        3   0     2
```

1.6.1 Subsetting

Subsetting means "selecting parts of data". It allows selecting only a number of variables (columns) or observations (rows) from a dataframe. There are ways to do that.

1.6.1.1 Selecting a column (variable) or a row (observation)

Let say, to select `age`

```r
data$age
```

```
##  [1] 38 35 39 36 31 38 33 36 40 34 38 40 40 28 37 38 49 29 40 38 34 46 42
## [24] 38 32 43 42 40 38 39 39 39 35 38 40 38 45 36 31 34 44 35 40 37 33 46
## [47] 42 40 45 42 45 38 34 44 39 38 39 47 41 44 30 48 47 42 42 49 31 38 38
```

```
## [70] 48 34 45 45 36 45 52 35 43 47 44
```

to select the 7th observation,

```
data[7, ]
```

```
##   chol age exercise sex categ
## 7    7  33        5   1     0
```

1.6.1.2 Selecting columns

Let us create a new data frame with only `chol`, `age` and `sex` as the variables

```
data_col <- subset(data, select = c("chol", "age", "sex"))
str(data_col)
```

```
## 'data.frame':    80 obs. of  3 variables:
##  $ chol: num  6.5 6.6 6.8 6.8 6.9 7 7 7.2 7.2 7.2 ...
##  $ age : int  38 35 39 36 31 38 33 36 40 34 ...
##  $ sex : int  1 1 1 1 1 1 1 1 1 1 ...
```

alternatively, we can use square brackets

```
data_col <- data[ , c("chol", "age", "sex")]
str(data_col)
```

```
## 'data.frame':    80 obs. of  3 variables:
##  $ chol: num  6.5 6.6 6.8 6.8 6.9 7 7 7.2 7.2 7.2 ...
##  $ age : int  38 35 39 36 31 38 33 36 40 34 ...
##  $ sex : int  1 1 1 1 1 1 1 1 1 1 ...
```

you may even select by the column numbers

```
data_col <- data[ , c(1:2, 4)]
str(data_col)
```

```
## 'data.frame':    80 obs. of  3 variables:
##  $ chol: num  6.5 6.6 6.8 6.8 6.9 7 7 7.2 7.2 7.2 ...
##  $ age : int  38 35 39 36 31 38 33 36 40 34 ...
##  $ sex : int  1 1 1 1 1 1 1 1 1 1 ...
```

selecting column 1 to 2, and column 4.

1.6.1.3 Selecting rows

To select 7th to 14th observations,

```
data_row <- data[7:14, ]
data_row
```

```
##    chol age exercise sex categ
## 7   7.0  33        5   1     0
```

```
## 8    7.2  36         5   1     0
## 9    7.2  40         4   1     0
## 10   7.2  34         6   1     0
## 11   7.3  38         6   1     0
## 12   7.3  40         5   1     0
## 13   7.3  40         4   1     0
## 14   7.3  28         5   1     0
```

Practically, we want to choose observations based on certain criteria, for example those aged > 35 year old,

```
data_row <- subset(data, age > 35)
str(data_row)
```

```
## 'data.frame':    62 obs. of  5 variables:
##  $ chol    : num  6.5 6.8 6.8 7 7.2 7.2 7.3 7.3 7.3 7.3 ...
##  $ age     : int  38 39 36 38 36 40 38 40 40 37 ...
##  $ exercise: int  6 6 5 4 5 4 6 5 4 5 ...
##  $ sex     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ categ   : int  0 0 0 0 0 0 0 0 0 0 ...
```

alternatively, we can use square brackets,

```
data_row <- data[data$age > 35, ]
str(data_row)
```

```
## 'data.frame':    62 obs. of  5 variables:
##  $ chol    : num  6.5 6.8 6.8 7 7.2 7.2 7.3 7.3 7.3 7.3 ...
##  $ age     : int  38 39 36 38 36 40 38 40 40 37 ...
##  $ exercise: int  6 6 5 4 5 4 6 5 4 5 ...
##  $ sex     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ categ   : int  0 0 0 0 0 0 0 0 0 0 ...
```

1.6.1.4 Select rows and columns together

Select those aged > 35, and `chol`, `age`, `sex` variables,

```
data_rc <- subset(data, age > 35 & sex == 1, select = c("chol", "age", "sex"))
str(data_rc)
```

```
## 'data.frame':    29 obs. of  3 variables:
##  $ chol: num  6.5 6.8 6.8 7 7.2 7.2 7.3 7.3 7.3 7.3 ...
##  $ age : int  38 39 36 38 36 40 38 40 40 37 ...
##  $ sex : int  1 1 1 1 1 1 1 1 1 1 ...
```

1.6.1.5 Creating a new variable

For example, create age in months,

```
data$age_month <- data$age * 12
data$age_month
```

```
##  [1] 456 420 468 432 372 456 396 432 480 408 456 480 480 336 444 456 588
## [18] 348 480 456 408 552 504 456 384 516 504 480 456 468 468 468 420 456
## [35] 480 456 540 432 372 408 528 420 480 444 396 552 504 480 540 504 540
## [52] 456 408 528 468 456 468 564 492 528 360 576 564 504 504 588 372 456
## [69] 456 576 408 540 540 432 540 624 420 516 564 528
```

1.6.2 Recoding

1.6.3 Categorize into new variables

1.6.3.1 From a numerical variable

```
data$age_cat <- cut(data$age, breaks = c(-Inf,40,50,Inf),
                    labels = c("< 40", "40-49", "> 50"))
table(data$age_cat)
```

```
##
##  < 40 40-49  > 50
##    51    28     1
```

```
str(data$age_cat)
```

```
##  Factor w/ 3 levels "< 40","40-49",..: 1 1 1 1 1 1 1 1 1 1 ...
```

1.6.3.2 From a categorical variable

Using `age_cat` variable,

```
levels(data$age_cat)
```

```
## [1] "< 40"  "40-49" "> 50"
```

```
table(data$age_cat)
```

```
##
##  < 40 40-49  > 50
##    51    28     1
```

Only one observation labeled as `> 50`. We want to combine `40-49` with `> 50`.

```
library(car)
data$age_cat1 <- recode(data$age_cat, "c('40-49','> 50') = '40 & above'")
table(data$age_cat1)  # combined
```

```
##
##       < 40 40 & above
```

```
##          51          29
```

1.6.4 Direct data entry

We may also enter short data directly using `read.table`.

For example, a standard data frame,

ID	Group	BMI
1	Fat	30
2	Fat	31
3	Fat	32
4	Thin	20
5	Thin	19
6	Thin	18

```
data_frame <- read.table(header = TRUE, text = "
ID Group BMI
1 Fat 30
2 Fat 31
3 Fat 32
4 Thin 20
5 Thin 19
6 Thin 18
")
str(data_frame)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ ID   : int  1 2 3 4 5 6
##  $ Group: Factor w/ 2 levels "Fat","Thin": 1 1 1 2 2 2
##  $ BMI  : int  30 31 32 20 19 18
```

```
data_frame
```

```
##   ID Group BMI
## 1  1   Fat  30
## 2  2   Fat  31
## 3  3   Fat  32
## 4  4  Thin  20
## 5  5  Thin  19
## 6  6  Thin  18
```

or a table,

	Cancer	No Cancer
Smoker	80	10
Non-smoker	5	100

```
data_table <- read.table(header = FALSE, text = "
80 10
5 100
                        ")
colnames(data_table) <- cancer <- c("Cancer", "No Cancer")
rownames(data_table) <- c("Smoker", "Non-smoker")
str(data_table)  # still a data frame, but laid out in form of a table.
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ Cancer   : int  80 5
##  $ No Cancer: int  10 100
```

```
data_table
```

```
##            Cancer No Cancer
## Smoker         80        10
## Non-smoker      5       100
```

The numbers are separated by space. We set the row and column names by `rownames` and `colnames` respectively.

1.7 Summary

In this chapter, we learned some basics in using R effectively. In the next chapter, we are going to learn about how to explore the variables by means of basic descriptive statistics.

Chapter 2

Descriptive statistics

In this chapter, we will go through a number of R functions for basic statistics. The focus will be on the results that are presented in form of numbers in text or tables (textual). We will mostly use the builtin functions (from R standard library). Extra packages will be introduced whenever necessary.

2.1 Preliminaries

In this part, we are going to use the functions as applied to a variable. For this purpose, we are going to use builtin datasets in R. You can view the available datasets by

```
data()
```

```
## Data sets in package 'datasets':

## AirPassengers                      Monthly Airline Passenger Numbers 1949-1960
## BJsales                            Sales Data with Leading Indicator
## BJsales.lead (BJsales)             Sales Data with Leading Indicator
## BOD                                Biochemical Oxygen Demand
## CO2                                Carbon Dioxide Uptake in Grass Plants
## ...
```

We can view any dataset description by appending "?" to the dataset name. For example,

```
?chickwts
```

We will start by using `chickwts` dataset that contains both numerical (`weight`) and categorical (`feed`) variables. We can view the first six observations,

```
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
```

```
## 6    168 horsebean
```

the last six observations,

```
tail(chickwts)
```

```
##    weight   feed
## 66    352 casein
## 67    359 casein
## 68    216 casein
## 69    222 casein
## 70    283 casein
## 71    332 casein
```

and the dimension of the data (row and column).

```
dim(chickwts)
```

```
## [1] 71  2
```

Here we have 71 rows (71 subjects) and two columns (two variables).

Next, view the names of the variables,

```
names(chickwts)
```

```
## [1] "weight" "feed"
```

and view the details of the data,

```
str(chickwts)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

which shows that `weight` is a numerical variable and `feed` is a factor, i.e. a categorical variable. `feed` consists of six categories or levels.

We can view the levels in `feed`,

```
levels(chickwts$feed)
```

```
## [1] "casein"    "horsebean" "linseed"   "meatmeal"  "soybean"   "sunflower"
```

2.2 One variable

2.2.1 A numerical variable

A numberical variable is described by a number of descriptive statistics below.

To judge the central tendency of the `weight` variable, we obtain its mean,

```
mean(chickwts$weight)
```

```
## [1] 261.3099
```

and median,

```r
median(chickwts$weight)
```

```
## [1] 258
```

To judge its spread and variability, we can view its minimum, maximum and range

```r
min(chickwts$weight)
```

```
## [1] 108
```

```r
max(chickwts$weight)
```

```
## [1] 423
```

```r
range(chickwts$weight)
```

```
## [1] 108 423
```

and obtain its standard deviation (SD)

```r
sd(chickwts$weight)
```

```
## [1] 78.0737
```

variance,

```r
var(chickwts$weight)
```

```
## [1] 6095.503
```

quantile,

```r
quantile(chickwts$weight)
```

```
##    0%   25%   50%   75%  100% 
## 108.0 204.5 258.0 323.5 423.0
```

and interquartile range (IQR)

```r
IQR(chickwts$weight)
```

```
## [1] 119
```

There are nine types of quantile algorithms in R (for `quantile` and `IQR`), the default being type 7. You may change this to type 6 (Minitab and SPSS),

```r
quantile(chickwts$weight, type = 6)
```

```
##   0%  25%  50%  75% 100% 
##  108  203  258  325  423
```

```r
IQR(chickwts$weight, type = 6)
```

```
## [1] 122
```

In addition to SD and IQR, we can obtain its median absolute deviation (MAD),

```
mad(chickwts$weight)
```

```
## [1] 91.9212
```

It is actually simpler to obtain most these in a single command,

```
summary(chickwts$weight)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   108.0   204.5   258.0   261.3   323.5   423.0
```

even simpler, obtain all of the statistics using `describe` in the `psych` package

```
install.packages("psych")
```

```
library(psych)
describe(chickwts$weight)
```

```
##    vars  n   mean    sd median trimmed   mad min max range  skew kurtosis
## X1    1 71 261.31 78.07    258     261 91.92 108 423   315 -0.01    -0.97
##      se
## X1 9.27
```

2.2.2 A categorical variable

A categorical variable is described by its count, proportion and percentage by categories.

We obtain the count of the `feed` variable,

```
summary(chickwts$feed)
```

```
##    casein horsebean   linseed  meatmeal   soybean sunflower
##        12        10        12        11        14        12
```

```
table(chickwts$feed)
```

```
##
##    casein horsebean   linseed  meatmeal   soybean sunflower
##        12        10        12        11        14        12
```

both `summary` and `table` give the same result.

`prop.table` gives the proportion of the result from the count.

```
prop.table(table(chickwts$feed))
```

```
##
##    casein horsebean   linseed  meatmeal   soybean sunflower
## 0.1690141 0.1408451 0.1690141 0.1549296 0.1971831 0.1690141
```

the result can be easily turned into percentage,

```
prop.table(table(chickwts$feed))*100
```

```
##
##    casein horsebean   linseed  meatmeal   soybean sunflower
##  16.90141  14.08451  16.90141  15.49296  19.71831  16.90141
```

To view the count and the percentage together, we can use `cbind`,

```
cbind(n = table(chickwts$feed), "%" = prop.table(table(chickwts$feed))*100)
```

```
##            n        %
## casein    12 16.90141
## horsebean 10 14.08451
## linseed   12 16.90141
## meatmeal  11 15.49296
## soybean   14 19.71831
## sunflower 12 16.90141
```

We need the quotation marks " " around the percentage sign %, because % also serves as a mathematical operator in R.

2.3 Two variables and more

Just now, we viewed all the statistics as applied to a variable. In this part, we are going to view the statistics on a number of variables. This includes viewing a group of numerical variables or categorical variables, or a mixture of numerical and categorical variables. This is relevant in a sense that, most of the time, we want to view everything in one go (e.g. the statistics of all items in a questionnaire), compare the means of several groups and obtain cross-tabulation of categorical variables.

2.3.1 Numerical variables

Let us use `women` dataset,

```
head(women)
```

```
##   height weight
## 1     58    115
## 2     59    117
## 3     60    120
## 4     61    123
## 5     62    126
## 6     63    129
```

```
names(women)
```

```
## [1] "height" "weight"
```

```
str(women)
```

```
## 'data.frame':    15 obs. of  2 variables:
##  $ height: num  58 59 60 61 62 63 64 65 66 67 ...
##  $ weight: num  115 117 120 123 126 129 132 135 139 142 ...
```

which consists of `weight` and `height` numerical variables.

The variables can be easily viewed together by `summary`,

```
summary(women)
```

```
##      height         weight
##  Min.   :58.0   Min.   :115.0
##  1st Qu.:61.5   1st Qu.:124.5
##  Median :65.0   Median :135.0
##  Mean   :65.0   Mean   :136.7
##  3rd Qu.:68.5   3rd Qu.:148.0
##  Max.   :72.0   Max.   :164.0
```

even better using `describe` (`psych`),

```
describe(women)
```

```
##        vars  n   mean    sd median trimmed   mad min max range skew
## height    1 15  65.00  4.47     65   65.00  5.93  58  72    14 0.00
## weight    2 15 136.73 15.50    135  136.31 17.79 115 164    49 0.23
##        kurtosis   se
## height    -1.44 1.15
## weight    -1.34 4.00
```

2.3.2 Categorical variables

Let us use `infert` dataset,

```
head(infert)
```

```
##   education age parity induced case spontaneous stratum pooled.stratum
## 1    0-5yrs  26      6       1    1           2       1              3
## 2    0-5yrs  42      1       1    1           0       2              1
## 3    0-5yrs  39      6       2    1           0       3              4
## 4    0-5yrs  34      4       2    1           0       4              2
## 5   6-11yrs  35      3       1    1           1       5             32
## 6   6-11yrs  36      4       2    1           1       6             36
```

```
names(infert)
```

```
## [1] "education"      "age"            "parity"         "induced"
## [5] "case"           "spontaneous"    "stratum"        "pooled.stratum"
```

```
str(infert)
```

```
## 'data.frame':    248 obs. of  8 variables:
##  $ education     : Factor w/ 3 levels "0-5yrs","6-11yrs",..: 1 1 1 1 2 2 2 2 2 2 ...
##  $ age           : num  26 42 39 34 35 36 23 32 21 28 ...
##  $ parity        : num  6 1 6 4 3 4 1 2 1 2 ...
##  $ induced       : num  1 1 2 2 1 2 0 0 0 0 ...
##  $ case          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ spontaneous   : num  2 0 0 0 1 1 0 0 1 0 ...
##  $ stratum       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ pooled.stratum: num  3 1 4 2 32 36 6 22 5 19 ...
```

We notice that `induced`, `case` and `spontaneous` are not yet set as categorical variables, thus we need to `factor` the variables. We view the value labels in the dataset description,

```
?infert
```

We label the values in the variables according to the description as

```
infert$induced <- factor(infert$induced, levels = 0:2, labels = c("0", "1", "2 or more"))
infert$case <- factor(infert$case, levels = 0:1, labels = c("control", "case"))
infert$spontaneous <- factor(infert$spontaneous, levels = 0:2, labels = c("0", "1", "2 or more")
str(infert)
```

```
## 'data.frame':    248 obs. of  8 variables:
##  $ education     : Factor w/ 3 levels "0-5yrs","6-11yrs",..: 1 1 1 1 2 2 2 2 2 2 ...
##  $ age           : num  26 42 39 34 35 36 23 32 21 28 ...
##  $ parity        : num  6 1 6 4 3 4 1 2 1 2 ...
##  $ induced       : Factor w/ 3 levels "0","1","2 or more": 2 2 3 3 2 3 1 1 1 1 ...
##  $ case          : Factor w/ 2 levels "control","case": 2 2 2 2 2 2 2 2 2 2 ...
##  $ spontaneous   : Factor w/ 3 levels "0","1","2 or more": 3 1 1 1 2 2 1 1 2 1 ...
##  $ stratum       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ pooled.stratum: num  3 1 4 2 32 36 6 22 5 19 ...
```

and we now all these variables are turned into factors.

Again, the variables can be easily viewed together by `summary`,

```
summary(infert[c("education", "induced", "case", "spontaneous")])
```

```
##    education        induced          case         spontaneous
##  0-5yrs : 12   0        :143   control:165   0        :141
##  6-11yrs:120   1        : 68   case   : 83   1        : 71
##  12+ yrs:116   2 or more: 37                 2 or more: 36
```

We do not use `table` here in form of `table(infert[c("education", "induced", "case", "spontaneous")])` because `table` used in this form will give us 3-way cross-tabulation instead of count per categories. Cross-tabulation of categorical variables will be covered later.

To obtain the proportion and percentage results, we have to use `lapply`,

```
lapply(infert[c("education", "induced", "case", "spontaneous")],
       function(x) summary(x)/length(x))
```

```
## $education
```

```
##    0-5yrs   6-11yrs   12+ yrs
## 0.0483871 0.4838710 0.4677419
##
## $induced
##         0         1 2 or more
## 0.5766129 0.2741935 0.1491935
##
## $case
##   control      case
## 0.6653226 0.3346774
##
## $spontaneous
##         0         1 2 or more
## 0.5685484 0.2862903 0.1451613
```

```
lapply(infert[c("education", "induced", "case", "spontaneous")],
       function(x) summary(x)/length(x)*100)
```

```
## $education
##   0-5yrs  6-11yrs  12+ yrs
##  4.83871 48.38710 46.77419
##
## $induced
##         0         1 2 or more
##  57.66129  27.41935  14.91935
##
## $case
##  control     case
## 66.53226 33.46774
##
## $spontaneous
##         0         1 2 or more
##  56.85484  28.62903  14.51613
```

because we need `lappy` to obtain the values for each of the variables. `lappy` goes through each variable and performs this particular part,

```
function(x) summary(x)/length(x)
```

`function(x)` is needed to specify some extra operations to any basic function in R, in our case `summary(x)` divided by `length(x)`, in which the summary results (the counts) are divided by the number of subjects (`length(x)` gives us the "length" of our dataset).

Now, since we already learned about `lapply`, we may also obtain the same results by using `summary` (within `lapply`), `table` and `prop.table`.

```
lapply(infert[c("education", "induced", "case", "spontaneous")], summary)
```

```
## $education
##  0-5yrs 6-11yrs 12+ yrs
##      12     120     116
```

```
##
## $induced
##         0         1 2 or more
##       143        68        37
##
## $case
## control    case
##     165      83
##
## $spontaneous
##         0         1 2 or more
##       141        71        36
```

```
lapply(infert[c("education", "induced", "case", "spontaneous")], table)
```

```
## $education
##
##  0-5yrs 6-11yrs 12+ yrs
##      12     120     116
##
## $induced
##
##         0         1 2 or more
##       143        68        37
##
## $case
##
## control    case
##     165      83
##
## $spontaneous
##
##         0         1 2 or more
##       141        71        36
```

```
lapply(infert[c("education", "induced", "case", "spontaneous")],
       function(x) prop.table(table(x)))
```

```
## $education
## x
##    0-5yrs   6-11yrs   12+ yrs
## 0.0483871 0.4838710 0.4677419
##
## $induced
## x
##         0         1 2 or more
## 0.5766129 0.2741935 0.1491935
##
## $case
```

```
## x
##   control      case
## 0.6653226 0.3346774
##
## $spontaneous
## x
##         0         1 2 or more
## 0.5685484 0.2862903 0.1451613
```

```
lapply(infert[c("education", "induced", "case", "spontaneous")],
       function(x) prop.table(table(x))*100)
```

```
## $education
## x
##   0-5yrs  6-11yrs  12+ yrs
##  4.83871 48.38710 46.77419
##
## $induced
## x
##         0         1 2 or more
##  57.66129  27.41935  14.91935
##
## $case
## x
##  control     case
## 66.53226 33.46774
##
## $spontaneous
## x
##         0         1 2 or more
##  56.85484  28.62903  14.51613
```

Notice here, whenever we do not need to specify extra operations on a basic function, e.g. `summary` and `table`, all we need to write after the comma in `lapply` is the basic function without `function(x)` and `(x)`.

2.4 Groups and cross-tabulations

We intentionally went through the descriptive statistics of a variable, followed by a number of variables of the same type. This will give you the basics in dealing with the variables. Most commonly, the variables are described by groups or in form cross-tabulated counts/percentages.

2.4.1 By groups

To obtain all the descriptive statistics by group, we can use `by` with the relevant functions. Let say we want to obtain the statistics by case and control (`case`). We start with numerical variables

```
by(infert[c("age", "parity")], infert$case, summary)
```

```
## infert$case: control
##       age            parity
##  Min.   :21.00   Min.   :1.000
##  1st Qu.:28.00   1st Qu.:1.000
##  Median :31.00   Median :2.000
##  Mean   :31.49   Mean   :2.085
##  3rd Qu.:35.00   3rd Qu.:3.000
##  Max.   :44.00   Max.   :6.000
## --------------------------------------------------------
## infert$case: case
##       age            parity
##  Min.   :21.00   Min.   :1.000
##  1st Qu.:28.00   1st Qu.:1.000
##  Median :31.00   Median :2.000
##  Mean   :31.53   Mean   :2.108
##  3rd Qu.:35.50   3rd Qu.:3.000
##  Max.   :44.00   Max.   :6.000
```

```
by(infert[c("age", "parity")], infert$case, describe)
```

```
## infert$case: control
##        vars   n  mean   sd median trimmed  mad min max range skew kurtosis
## age       1 165 31.49 5.25     31   31.34 5.93  21  44    23 0.23    -0.72
## parity    2 165  2.08 1.24      2    1.88 1.48   1   6     5 1.32     1.42
##          se
## age    0.41
## parity 0.10
## --------------------------------------------------------
## infert$case: case
##        vars  n  mean   sd median trimmed  mad min max range skew kurtosis
## age       1 83 31.53 5.28     31   31.39 5.93  21  44    23 0.21    -0.77
## parity    2 83  2.11 1.28      2    1.90 1.48   1   6     5 1.32     1.34
##          se
## age    0.58
## parity 0.14
```

We can also use `describeBy`, which is an the extension of `describe` in the `psych` package.

```
describeBy(infert[c("age", "parity")], group = infert$case)
```

```
##
##  Descriptive statistics by group
## group: control
##        vars   n  mean   sd median trimmed  mad min max range skew kurtosis
## age       1 165 31.49 5.25     31   31.34 5.93  21  44    23 0.23    -0.72
## parity    2 165  2.08 1.24      2    1.88 1.48   1   6     5 1.32     1.42
##          se
```

```
## age    0.41
## parity 0.10
## --------------------------------------------------------
## group: case
##        vars  n  mean   sd median trimmed  mad min max range skew kurtosis
## age       1 83 31.53 5.28     31   31.39 5.93  21  44    23 0.21    -0.77
## parity    2 83  2.11 1.28      2    1.90 1.48   1   6     5 1.32     1.34
##          se
## age    0.58
## parity 0.14
```

which gives us an identical result.

If you want to obtain results using the basic functions (i.e. `mean`, `median`, `quantile`, `IQR` and `mad`), you need to use `lappy` within `by`, because they could not handle many variables, for example for `mean` and `IQR`,

```
by(infert[c("age", "parity")], infert$case, function(x) lapply(x, mean))
```

```
## infert$case: control
## $age
## [1] 31.49091
##
## $parity
## [1] 2.084848
##
## --------------------------------------------------------
## infert$case: case
## $age
## [1] 31.53012
##
## $parity
## [1] 2.108434
```

```
by(infert[c("age", "parity")], infert$case, function(x) lapply(x, IQR))
```

```
## infert$case: control
## $age
## [1] 7
##
## $parity
## [1] 2
##
## --------------------------------------------------------
## infert$case: case
## $age
## [1] 7.5
##
## $parity
## [1] 2
```

For categorical variables, using summary

```
by(infert[c("education", "induced", "spontaneous")], infert$case, summary)
```

```
## infert$case: control
##    education       induced      spontaneous
##  0-5yrs : 8   0        :96   0        :113
##  6-11yrs:80   1        :45   1        : 40
##  12+ yrs:77   2 or more:24   2 or more: 12
## --------------------------------------------------------
## infert$case: case
##    education       induced      spontaneous
##  0-5yrs : 4   0        :47   0        :28
##  6-11yrs:40   1        :23   1        :31
##  12+ yrs:39   2 or more:13   2 or more:24
```

```
by(infert[c("education", "induced", "spontaneous")], infert$case,
   function(x) lapply(x, function(x) summary(x)/length(x)))
```

```
## infert$case: control
## $education
##     0-5yrs    6-11yrs    12+ yrs
## 0.04848485 0.48484848 0.46666667
##
## $induced
##         0         1 2 or more
## 0.5818182 0.2727273 0.1454545
##
## $spontaneous
##          0          1  2 or more
## 0.68484848 0.24242424 0.07272727
##
## --------------------------------------------------------
## infert$case: case
## $education
##     0-5yrs    6-11yrs    12+ yrs
## 0.04819277 0.48192771 0.46987952
##
## $induced
##         0         1 2 or more
## 0.5662651 0.2771084 0.1566265
##
## $spontaneous
##         0         1 2 or more
## 0.3373494 0.3734940 0.2891566
```

```
by(infert[c("education", "induced", "spontaneous")], infert$case,
   function(x) lapply(x, function(x) summary(x)/length(x)*100))
```

```
## infert$case: control
```

```
## $education
##    0-5yrs   6-11yrs   12+ yrs
##  4.848485 48.484848 46.666667
##
## $induced
##         0         1 2 or more
##  58.18182  27.27273  14.54545
##
## $spontaneous
##         0         1 2 or more
## 68.484848 24.242424  7.272727
##
## --------------------------------------------------------
## infert$case: case
## $education
##    0-5yrs   6-11yrs   12+ yrs
##  4.819277 48.192771 46.987952
##
## $induced
##         0         1 2 or more
##  56.62651  27.71084  15.66265
##
## $spontaneous
##         0         1 2 or more
##  33.73494  37.34940  28.91566
```

or by using table

```r
by(infert[c("education", "induced", "spontaneous")], infert$case,
   function(x) lapply(x, table))
```

```
## infert$case: control
## $education
##
##  0-5yrs 6-11yrs 12+ yrs
##       8      80      77
##
## $induced
##
##         0         1 2 or more
##        96        45        24
##
## $spontaneous
##
##         0         1 2 or more
##       113        40        12
##
## --------------------------------------------------------
## infert$case: case
```

```
## $education
##
##  0-5yrs 6-11yrs 12+ yrs
##       4      40      39
##
## $induced
##
##         0         1 2 or more
##        47        23        13
##
## $spontaneous
##
##         0         1 2 or more
##        28        31        24
```

```
by(infert[c("education", "induced", "spontaneous")], infert$case,
   function(x) lapply(x, function(x) prop.table(table(x))))
```

```
## infert$case: control
## $education
## x
##     0-5yrs    6-11yrs    12+ yrs
## 0.04848485 0.48484848 0.46666667
##
## $induced
## x
##         0         1 2 or more
## 0.5818182 0.2727273 0.1454545
##
## $spontaneous
## x
##          0          1  2 or more
## 0.68484848 0.24242424 0.07272727
##
## --------------------------------------------------------
## infert$case: case
## $education
## x
##     0-5yrs    6-11yrs    12+ yrs
## 0.04819277 0.48192771 0.46987952
##
## $induced
## x
##         0         1 2 or more
## 0.5662651 0.2771084 0.1566265
##
## $spontaneous
## x
```

```
##         0         1 2 or more
## 0.3373494 0.3734940 0.2891566
```

```
by(infert[c("education", "induced", "spontaneous")], infert$case,
   function(x) lapply(x, function(x) prop.table(table(x))*100))
```

```
## infert$case: control
## $education
## x
##    0-5yrs   6-11yrs   12+ yrs
##  4.848485 48.484848 46.666667
##
## $induced
## x
##         0         1 2 or more
##  58.18182  27.27273  14.54545
##
## $spontaneous
## x
##         0         1 2 or more
## 68.484848 24.242424  7.272727
##
## --------------------------------------------------------
## infert$case: case
## $education
## x
##    0-5yrs   6-11yrs   12+ yrs
##  4.819277 48.192771 46.987952
##
## $induced
## x
##         0         1 2 or more
##  56.62651  27.71084  15.66265
##
## $spontaneous
## x
##         0         1 2 or more
##  33.73494  37.34940  28.91566
```

Please note that simply replacing `table` for `summary` as in `by(infert[c("education", "induced", "spontaneous")], infert$case, table)` will not work as intended. `education` will be nested in `induced`, which is nested in `spontaneous`, listed by `case` instead. And yes, to obtain the proportions and percentages, it gets slightly more complicated as we have to specify `function` twice in `by`.

2.4.2 Cross-tabulation

As long as the categorical variables are already `factor`ed properly, there should not be a problem to obtain the cross-tabulation tables. For example between `education` and `case`,

```r
table(infert$education, infert$case)
```

```
##
##           control case
##   0-5yrs        8    4
##   6-11yrs      80   40
##   12+ yrs      77   39
```

We may also include row and column headers, just like `cbind`,

```r
table(education = infert$education, case = infert$case)
```

```
##          case
## education control case
##   0-5yrs        8    4
##   6-11yrs      80   40
##   12+ yrs      77   39
```

Since we are familiar with the powerful `lappy`, we can use it to get cross-tabulation of all of the factors with `case` status,

```r
lapply(infert[c("education", "induced", "spontaneous")], function(x) table(x, infert$case))
```

```
## $education
##
## x         control case
##   0-5yrs        8    4
##   6-11yrs      80   40
##   12+ yrs      77   39
##
## $induced
##
## x           control case
##   0              96   47
##   1              45   23
##   2 or more      24   13
##
## $spontaneous
##
## x           control case
##   0             113   28
##   1              40   31
##   2 or more      12   24
```

We may also view subgroup counts (nesting). Here, the cross-tabulation of `education` and `case` is nested within `induced`

```r
table(infert$education, infert$case, infert$induced)
```

```
## , ,  = 0
##
```

```
##
##            control case
##   0-5yrs         4    0
##   6-11yrs       57   21
##   12+ yrs       35   26
##
## , ,  = 1
##
##
##            control case
##   0-5yrs         0    2
##   6-11yrs       16   11
##   12+ yrs       29   10
##
## , ,  = 2 or more
##
##
##            control case
##   0-5yrs         4    2
##   6-11yrs        7    8
##   12+ yrs       13    3
```

which will look nicer if we apply `by`

```r
by(infert[c("education", "case")], infert$induced, table)
```

```
## infert$induced: 0
##          case
## education control case
##   0-5yrs         4    0
##   6-11yrs       57   21
##   12+ yrs       35   26
## --------------------------------------------------------
## infert$induced: 1
##          case
## education control case
##   0-5yrs         0    2
##   6-11yrs       16   11
##   12+ yrs       29   10
## --------------------------------------------------------
## infert$induced: 2 or more
##          case
## education control case
##   0-5yrs         4    2
##   6-11yrs        7    8
##   12+ yrs       13    3
```

2.5 Summary

In this chapter, we learned about how to handle numerical and categorical variables and obtain the basic and relevant statistics. In the next chapter, we are going to learn about how to explore the variables visually in form of the relevant graphs and plots.

Chapter 3

Visual exploration

3.1 Introduction to visualization

Data visualization or data visualisation is viewed by many disciplines as a modern equivalent of visual communication. It involves the creation and study of the visual representation of data.

This means "information that has been abstracted in some schematic form, including attributes or variables for the units of information".

For complete references, read these sources:

1. https://en.m.wikipedia.org/wiki/Data_visualization
2. https://en.m.wikipedia.org/wiki/Michael_Friendly

3.1.1 History of data visualization

In his 1983 book *The Visual Display of Quantitative Information* (**ref needed**), the author Edward Tufte defines *graphical displays* and the principles for effective graphical displays. The book defines "excellence in statistical graphics consists of complex ideas communicated with clarity, precision and efficiency".

3.1.2 Processes and objectives of visualization

Visualization is the process of representing data graphically and interacting with these representations. The main objective is to gain insight into the data (http://researcher.watson.ibm.com/researcher/view_group.php?id=143)

3.1.3 What makes good graphics

You may require these to make good graphics:

1. Data

2. Substance rather than about method, graphic design, technology of graphic production or something else
3. No distortion to what the data has to say
4. Presence of many numbers in a small space
5. Coherence for large data sets
6. Encourage the eye to compare different pieces of data
7. Reveal the data at several levels of detail, from a broad overview to the fine structure
8. Serve a reasonably clear purpose: description, exploration, tabulation or decoration
9. Be closely integrated with the statistical and verbal descriptions of a data set.

3.2 Graphics packages in R

There are a number of graphics packages in R. A few of the packages are aimed to perform tasks related with graphs. Some provide graphics for certain analyses.

The popular general graphics packages in R include:

1. `graphics`
2. `ggplot2`
3. `lattice`

Some examples of other more specific packages aimed to run graphics for certain analyses include:

1. `survminer::ggsurvlot` to plot survival probability
2. `sjPlot` to plot mixed models results

3.3 Preliminaries

We will be using a dataset named `cholest.dta` which is in Stata format.

```
library(foreign)
cholest <- read.dta("cholest.dta")
head(cholest)
```

```
##   chol age exercise  sex categ
## 1  6.5  38        6 male Grp A
## 2  6.6  35        5 male Grp A
## 3  6.8  39        6 male Grp A
## 4  6.8  36        5 male Grp A
## 5  6.9  31        4 male Grp A
## 6  7.0  38        4 male Grp A
```

The data can be summarized as:

```
summary(cholest)
```

```
##       chol            age           exercise          sex        categ
##  Min.   : 6.50   Min.   :28.00   Min.   :2.000   female:40   Grp A:25
##  1st Qu.: 7.60   1st Qu.:36.00   1st Qu.:4.000   male  :40   Grp B:33
```

```
##  Median : 8.30   Median :39.00   Median :4.000               Grp C:22
##  Mean   : 8.23   Mean   :39.48   Mean   :4.225
##  3rd Qu.: 8.80   3rd Qu.:43.25   3rd Qu.:5.000
##  Max.   :10.00   Max.   :52.00   Max.   :6.000
```

From variable `sex`, we will create a variable named `male` and label female for 0 and 1 for male)

```
cholest$male <- factor(cholest$sex, labels = c('female', 'male'))
```

3.4 Questions to ask before plotting graphs

You must ask yourselves these questions:

1. Which variable or variables do I want to plot?
2. What is (or are) the type of that variable?

- Are they factor (categorical) variables ?
- Are they numerical variables?

3. Am I going to plot

- a single variable?
- two variables together?
- three variables together?

3.5 Using the `graphics` package

3.5.1 One variable: Plotting a categorical variable

For categorical variable, we can plot a `barchart` to display the frequencies of the data.

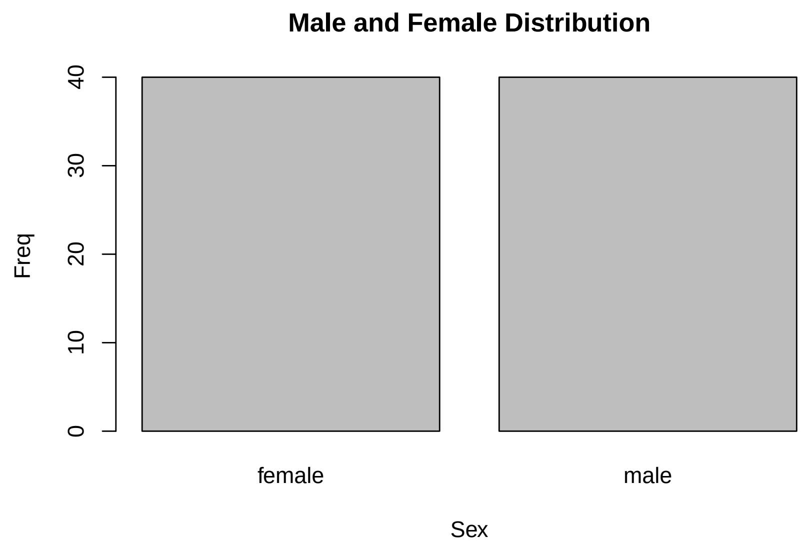
Create a frequency table and name as `count`:

```
counts <- table(cholest$male)
counts
```

```
##
## female   male
##     40     40
```

Now, plot the frequencies for the `counts` object created above

```
barplot(counts, main="Male and Female Distribution",
    xlab = "Sex", ylab = "Freq")
```



3.5.2 One variable: Plotting a numerical variable

A common graphic for numerical variable is a histogram.

Histogram

We create histograms with the function `hist(x)`. In the function,

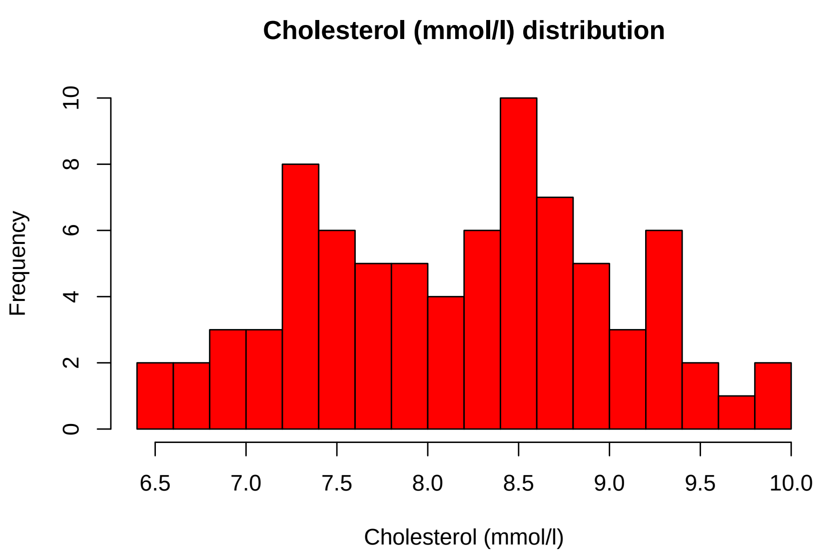
1. the argument `x` is a numeric vector of values to be plotted
2. the argument option `freq=FALSE` plots probability densities instead of frequencies.
3. the argument option `breaks =` controls the number of bins.

```
hist(cholest$chol)
```

Now, inside the `hist()` function, we will

1. set the `col =` argument to `red`
2. set the argument for bin to 12 bins `breaks = 14`
3. label the x-axis as "Cholesterol (mmol/l)"
4. the title is set in `main = 'title of plot'`

```
hist(cholest$chol, breaks = 14, col = "red",
     main = "Cholesterol (mmol/l) distribution", xlab = "Cholesterol (mmol/l)")
```



Kernel density plot

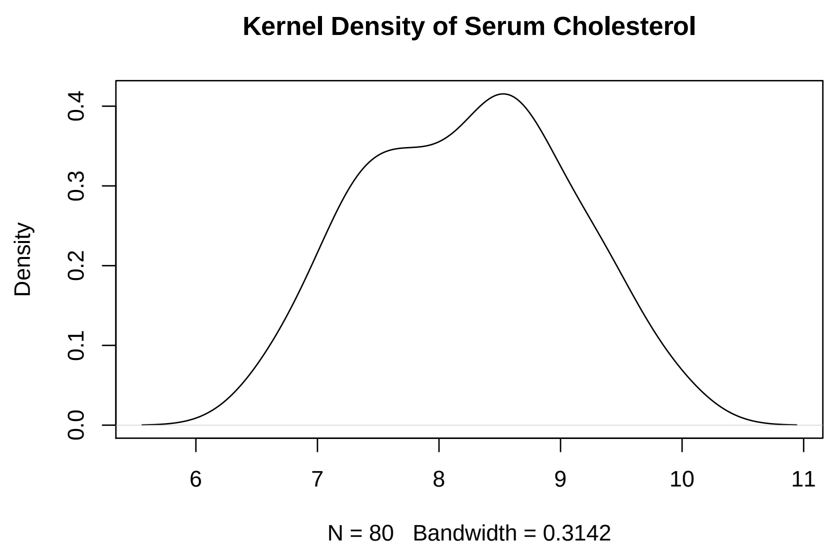
Kernal density plots are usually a much more effective way to view the distribution of a variable.

This can be done using `plot(density(x))`. In the function, the argument for x is a numeric vector.

Below, we

1. create a density plot and named it as `d.plot`. We do not consider missing value by setting the `na.rm = TRUE`
2. next, we plot `d.plot`

```
d.plot <- density(cholest$chol, na.rm = TRUE) # returns the density data
plot(d.plot, main = "Kernel Density of Serum Cholesterol") # plots the results
```



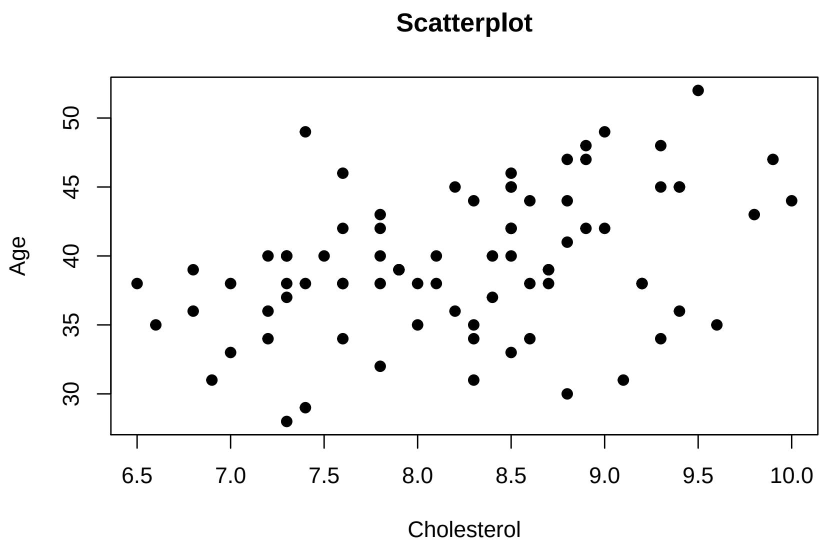
3.5.3 Two variables: Plotting scatter plot

We can plot two numerical variables simultenously. From such plot, we can see the association or relationship between the two variables.

Scatter plot

Scatter plot is one of the most common plots to display the association between 2 numerical variables.

```
plot(cholest$chol, cholest$age, main = "Scatterplot",
    xlab = "Cholesterol", ylab = "Age", pch=19)
```



You can always personalize the graphical parameters such as parameters for *fonts, colours, lines* and *symbols*. You can find the details in the `graphics` documentation. In addition, this website summarizes the parameters in a very nice way: http://www.statmethods.net/advgraphs/parameters.html

3.6 Using the `ggplot2` package

The official website for ggplot2 is here http://ggplot2.org/. In their own words, the package is described as

ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

3.6.1 One variable: Plotting a numerical variable

Plot distribution of values of a numerical variable.

Histogram

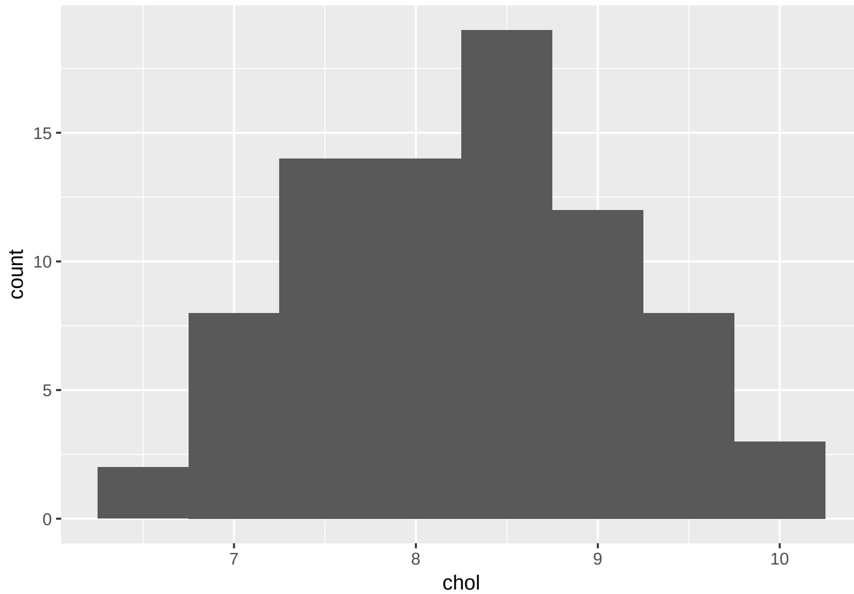
Load the `ggplot2` package,

```
library(ggplot2)
```

In `ggplot2`,

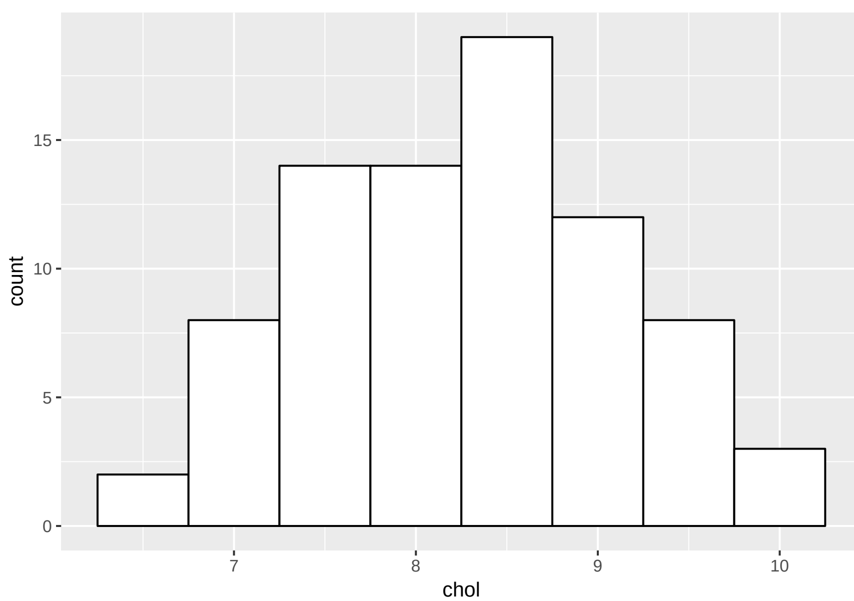
1. type `ggplot(data = X)` function to choose the dataset .
2. the `aes()` for variable or variables to be plotted.
3. then we use `geom_X` to specify the geometric (X) form of the plot.

```
myplot <- ggplot(data = cholest, aes(x = chol))
myplot + geom_histogram(binwidth = 0.5)
```



ggplot2 has lots of flexibility and personalization. For example, we can set the line color and fill color, the theme, the size, the symbols etc.

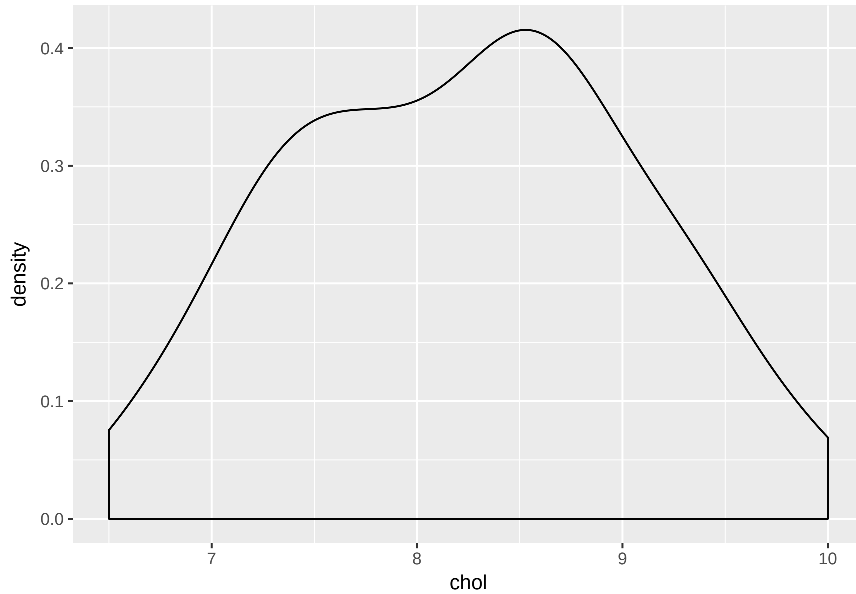
```
ggplot(cholest, aes(x = chol)) + geom_histogram(binwidth = 0.5,
                                                colour = "black", fill = "white")
```



Density curve

Density is useful to examine the distribution of observations.

```
ggplot(data = subset(cholest, !is.na(chol)), aes(x = chol)) + geom_density()
```



Combining the histogram and the density curve

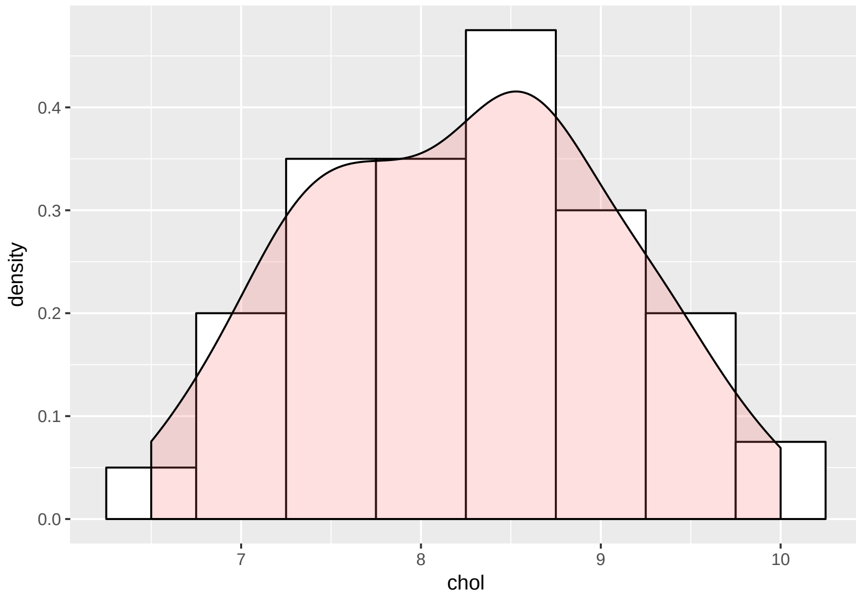
`ggplot2` allows plot to be displayed together. We can combine multiple plots in one single plot by overlaying multiple plots on one another.

Here, we will

1. create a histogram plot
2. create a density curve plot
3. overlay both (the density curve + the histogram).

To do this we need to specify a histogram with density instead of count on y-axis

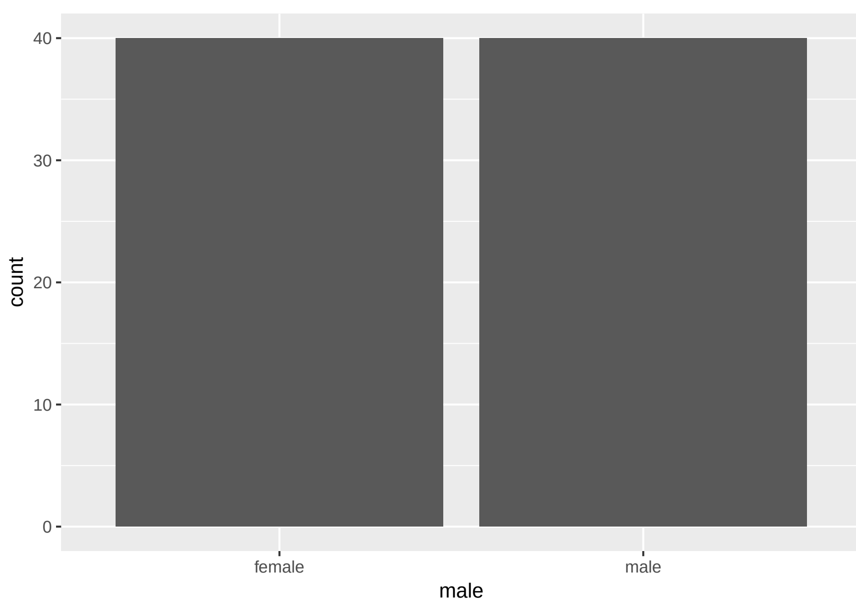
```
ggplot(data = subset(cholest, !is.na(chol)), aes(x = chol)) +
        geom_histogram(aes(y = ..density..), binwidth = 0.5,
                   colour = "black", fill = "white") +
    geom_density(alpha = .2, fill = "#FF6666")
```



3.6.2 One variable: Plotting a categorical variable

Now, let us create a basic barchart using `ggplot2::geom_bar()`

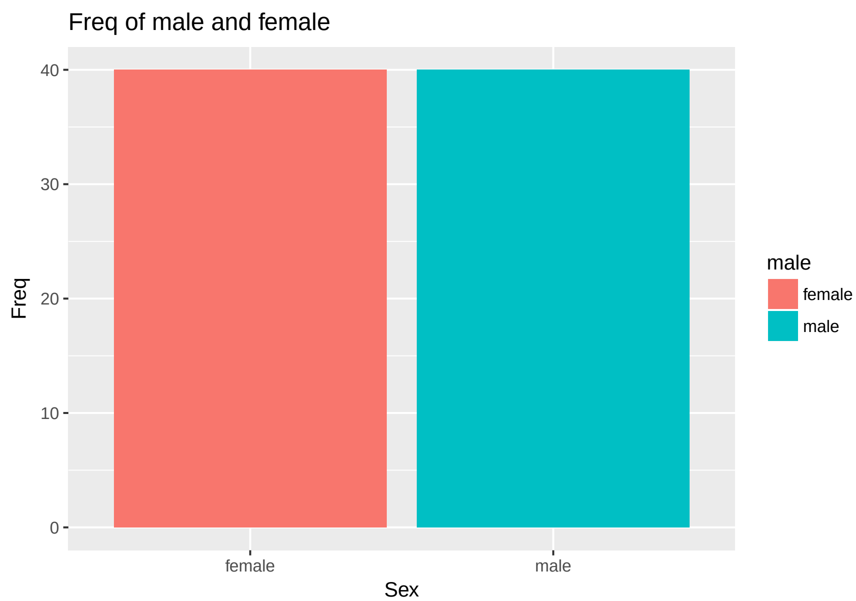
```
sex_bar <- ggplot(data = cholest, aes(male))
sex_bar + geom_bar()
```



The barchart looks OK, but we want to personalize it more - make it prettier and more presentable:

1. add labels to x and y axes `xlab()` and `ylab()`
2. add the title `ggtitle()`

```
ggplot(data = cholest, mapping = aes(male, fill = male)) +
  geom_bar() + xlab('Sex') + ylab('Freq') +
  ggtitle('Freq of male and female')
```



In addition, there is an excellent resource from this website on ggplot2: http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/

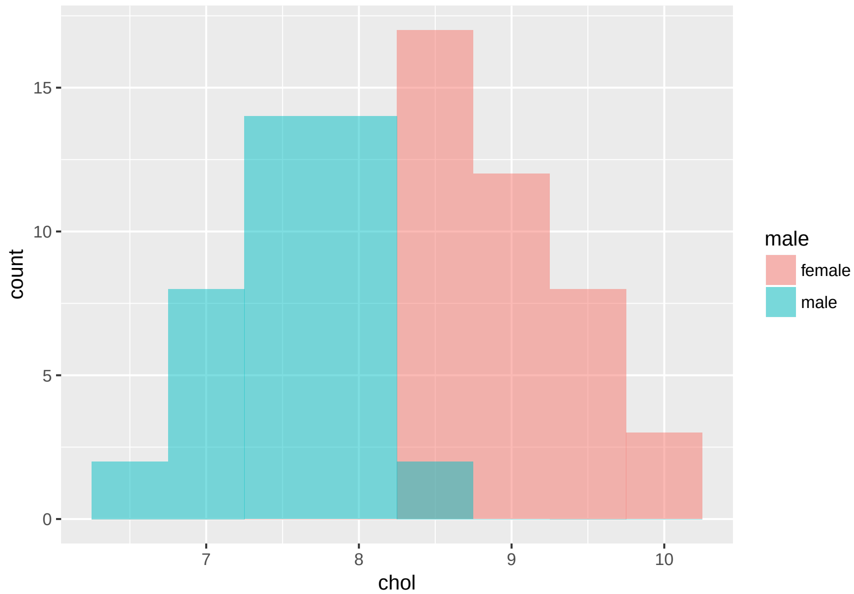
3.6.3 Two variables: Plotting a numerical and a categorical variable

Now, examine the distribution of a numerical variable (`rating`) in two groups (A and B) of the variable `cond` by

1. overlaying two histograms
2. interleaving two histograms
3. overlaying two density curve

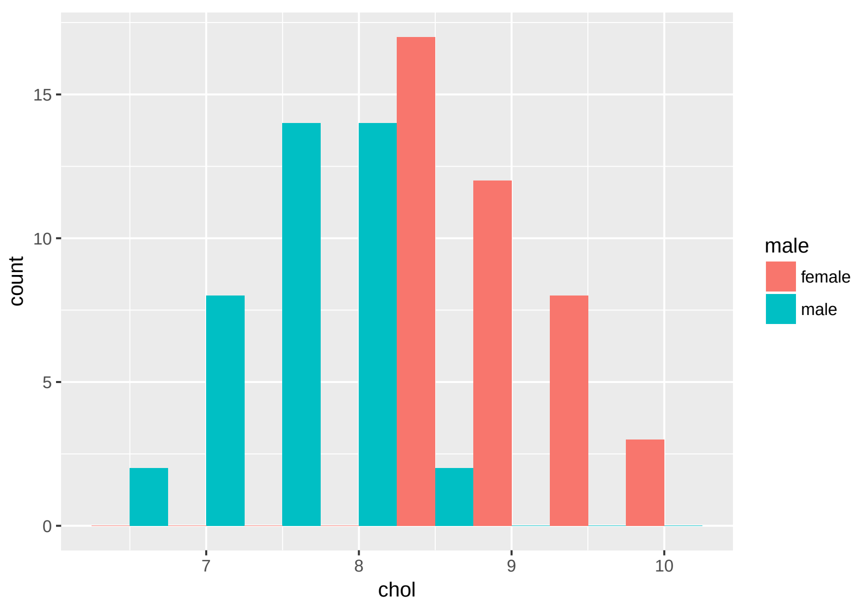
Overlaying histograms

```
ggplot(cholest, aes(x = chol, fill = male)) +
    geom_histogram(binwidth = .5, alpha = .5, position = "identity")
```



Interleaving histograms

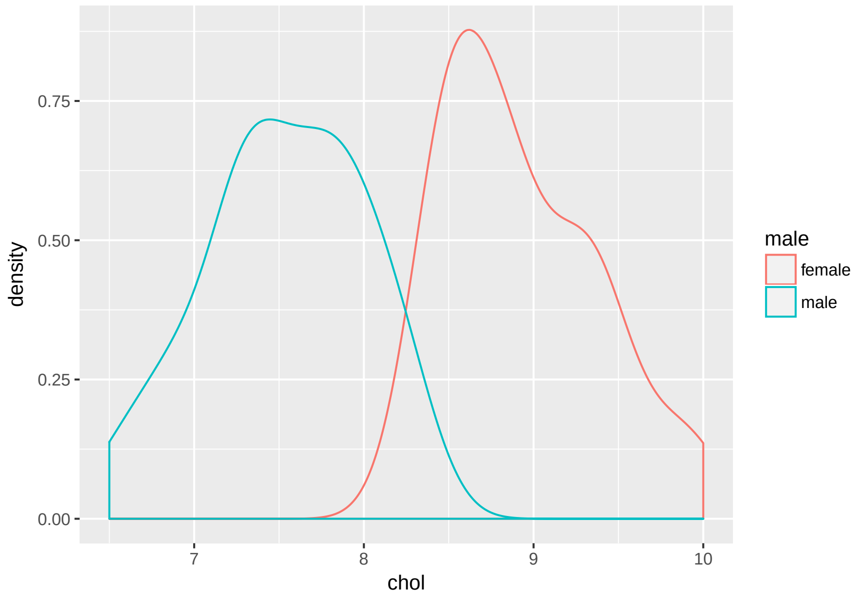
```
ggplot(cholest, aes(x = chol, fill = male)) +
    geom_histogram(binwidth = .5, position = "dodge")
```



Overlaying density plots

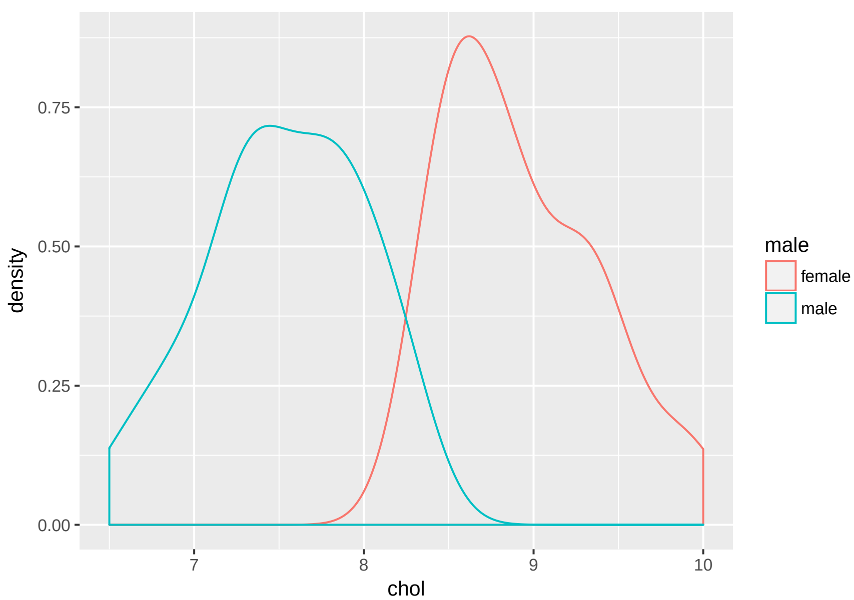
Full transparent

```
ggplot(cholest, aes(x = chol, colour = male)) + geom_density()
```



Now, try set the transparency at 30%

```
# Density plots with semi-transparent fill
ggplot(cholest, aes(x = chol, colour = male)) + geom_density(alpha = .3)
```



Using facets

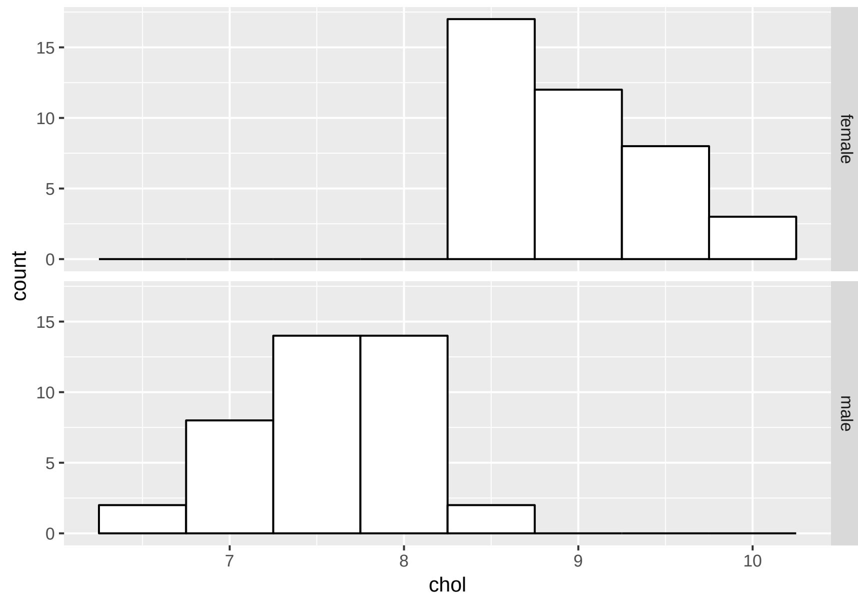
We use `facet_grid()` to split the plot.

There are two types of facetting the plot:

1. Vertical facet
2. Horizontal facet

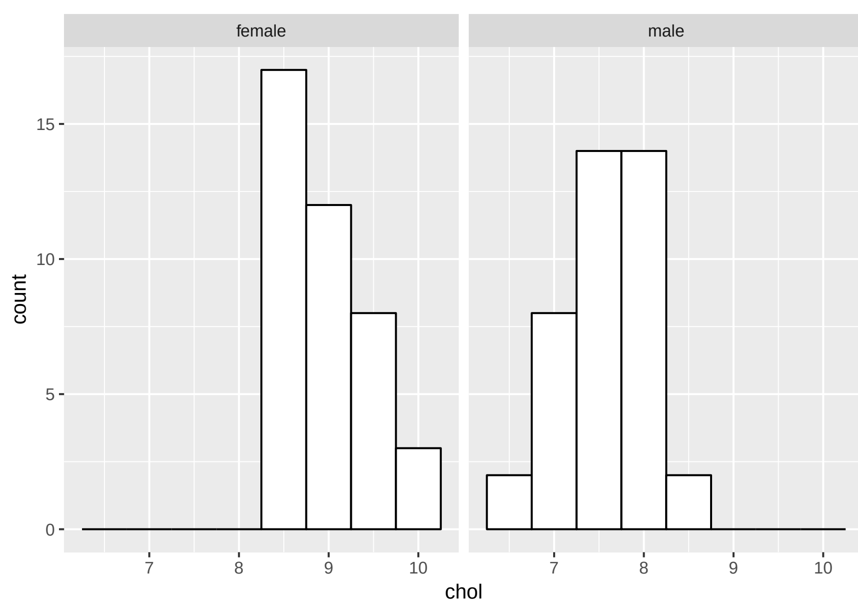
The vertical facets

```
ggplot(data = subset(cholest, !is.na(chol)), aes(x = chol)) + geom_histogram(binwidth = .5,
                                                        colour = "black", fill = "white") +
  facet_grid(male ~ .)
```



The horizontal facets

```
ggplot(data = subset(cholest, !is.na(chol)), aes(x = chol)) + geom_histogram(binwidth = .5,
                                                        colour = "black", fill = "white") +
    facet_grid(. ~ male)
```



3.6.4 Saving plots in ggplot2

This will save the last plot as .png and .jpg formats,

```
ggsave("myhistogram.png", width = 5, height = 5)
ggsave("myhistogram.jpg", width = 5, height = 5)
```

3.7 Using the lattice package

lattice package can create beautiful plots too. A very useful vignette for lattice package can be found here http://lattice.r-forge.r-project.org/Vignettes/src/lattice-intro/lattice-intro.pdf

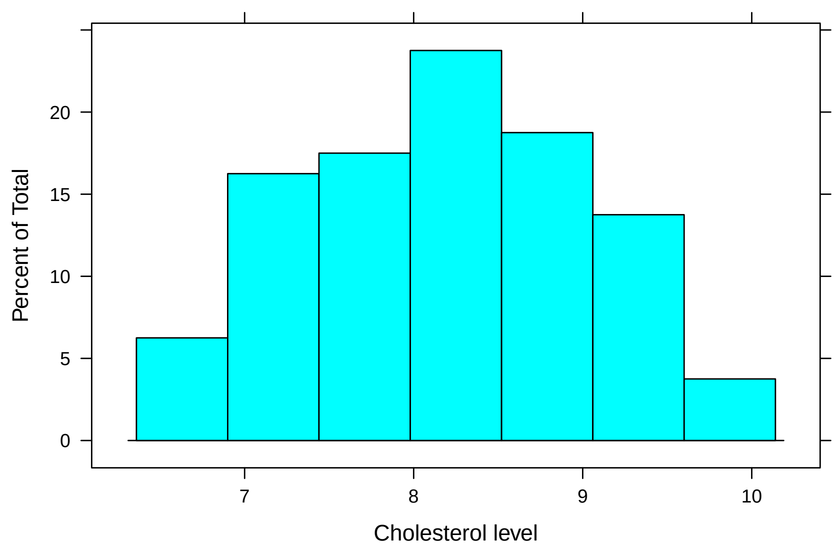
3.7.1 Loading the lattice package

```
library(lattice)
```

3.7.2 One numerical variable: Plotting a histogram

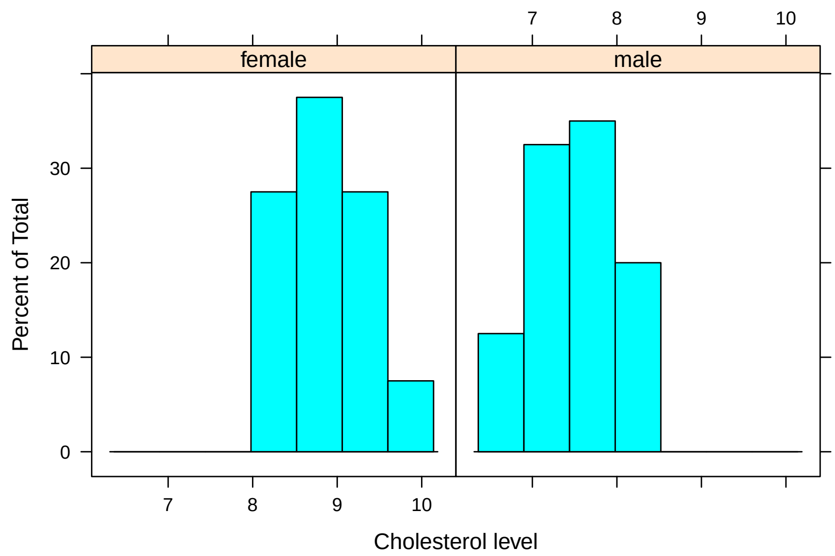
Plot a histogram for variable chol and label the x axis

```
histogram(~ chol, data = cholest, xlab = 'Cholesterol level')
```



3.7.3 One numerical and one categorical variable: Plotting histograms

```
histogram(~ chol | male, data = cholest,
          xlab = 'Cholesterol level' )
```

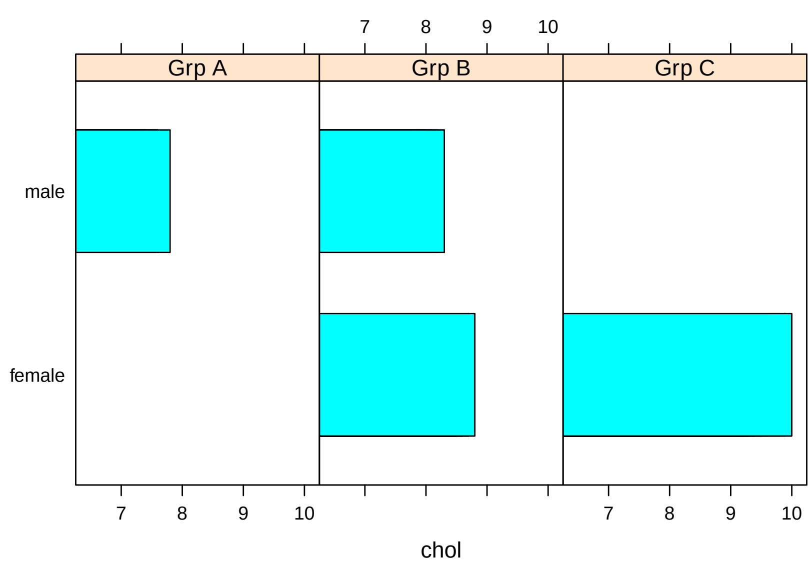


3.7.4 One categorical variable: Plotting a barchart

We use `barchart()` and set

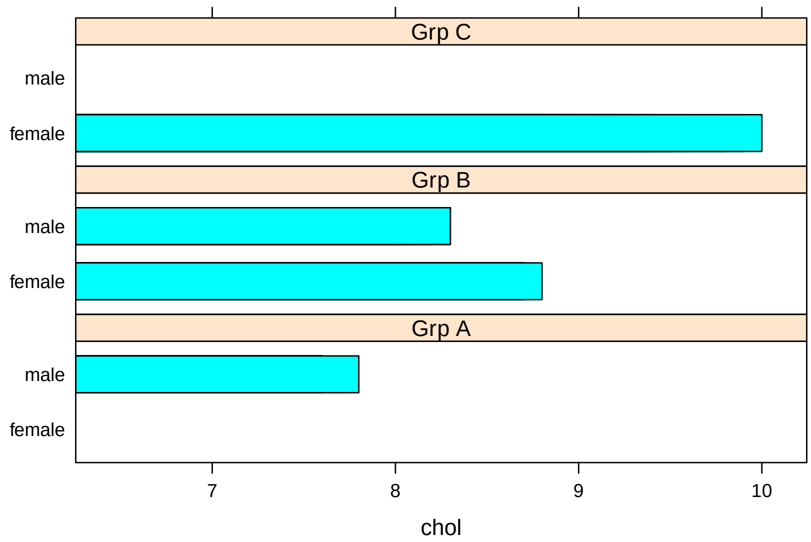
1. dependent variable = `male`
2. independent variable = `chol`
3. 4 bar charts in 4 columns and 1 row

```
barchart(male ~ chol | factor(categ), data = cholest, layout = c(3, 1))
```



Now we change the variables for the x and y axes and also the column and row arrangements - vertical plots, that is 1 column and 4 rows

```r
barchart(male ~ chol | factor(categ), data = cholest, layout = c(1, 3))
```



3.8 Summary

Chapter 4

Reporting results

In this chapter, we will learn how to combine bits and parts of your results into custom-made texts, labels and tables. This will be useful for reporting and summarizing your results for publication, as well as labeling specific axes on your plots. There are a number of ways to achieve this. We will do this by basic functions and packages.

4.1 `cbind` and `data.frame`

4.2 `paste` and `paste0`

4.3 `kable`

4.4 `stargazer`

Chapter 5

Practicals

In this chapter, using a number of additional data sets, we will conduct full exploration of the data from data description, descriptive statistics, plots and reporting.

5.1 data 1

5.2 data 2

5.3 data 3

5.4 Summary

[summary here]

Chapter 6

References

R Core Team. (2017). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Boca Raton, Florida: Chapman; Hall/CRC. Retrieved from http://yihui.name/knitr/