Exploring data using R

Kamarul Imran Musa, Wan Nor Arifin

Draft updated 30 October 2018

Contents

Preface

This is a book on data exploration using R. The focus of this book is mainly on the basics of R, data entry and management, descriptive statistics and graphical exploration of the data.

We did not cover basic statistical analyses, for examples t-test and chi-squared test, to focus on the basics in data exploration. By limiting the scope to data exploration, we can cover this aspect of handling data in greater details.

We also include a chapter on combining outputs for data presentation and also some packages that provide nice looking ready-made tables. To end this book, we provide more examples using a number of selected datasets, covering the data exploration skills from the preceding chapters.

All in all, we hope you enjoy this book!

Kamarul Imran Musa, Wan Nor Arifin

Chapter 1

Introduction to R

This chapter introduces the basics of getting started with R. We start with installing R and RStudio, some basics about R syntax, dealing with packages and setting up the working directory.

1.1 R

1.1.1 Installing R

The latest version of R is R version 3.5.1 (2018-07-02), Feather Spray. R can run on Windows OS, Mac OS and Linux distribution.

You need to download the R installation files from https://cran.r-project.org/. And you can install many versions of R in one single machine. There is no need to uninstall if you want to upgrade the currently installed R. The size of installtion files as of today 2018-10-30 is about 80 megabytes.

The links to install R for

1. Windows is https://cran.r-project.org/bin/windows/. Then click `base` subdirectories
2. Mac OS is https://cran.r-project.org/bin/macosx/
3. Linux is https://cran.r-project.org/bin/linux/

1.1.2 Starting R

You can start R software like starting any other software. In Windows, double click on R icon on the Start page and you should get this:

If you can see the R GUI, you are good to go. In the figure, I am using the Microsoft R Open version 3.3.3. In your case, the R will be shown as just R version R version 3.5.1 (2018-07-02), Feather Spray.

1.2 RStudio

1.2.1 Installing RStudio

We encourage you to install RStudio in your machine. In the RStudio website, the company describes RStudio as follows:

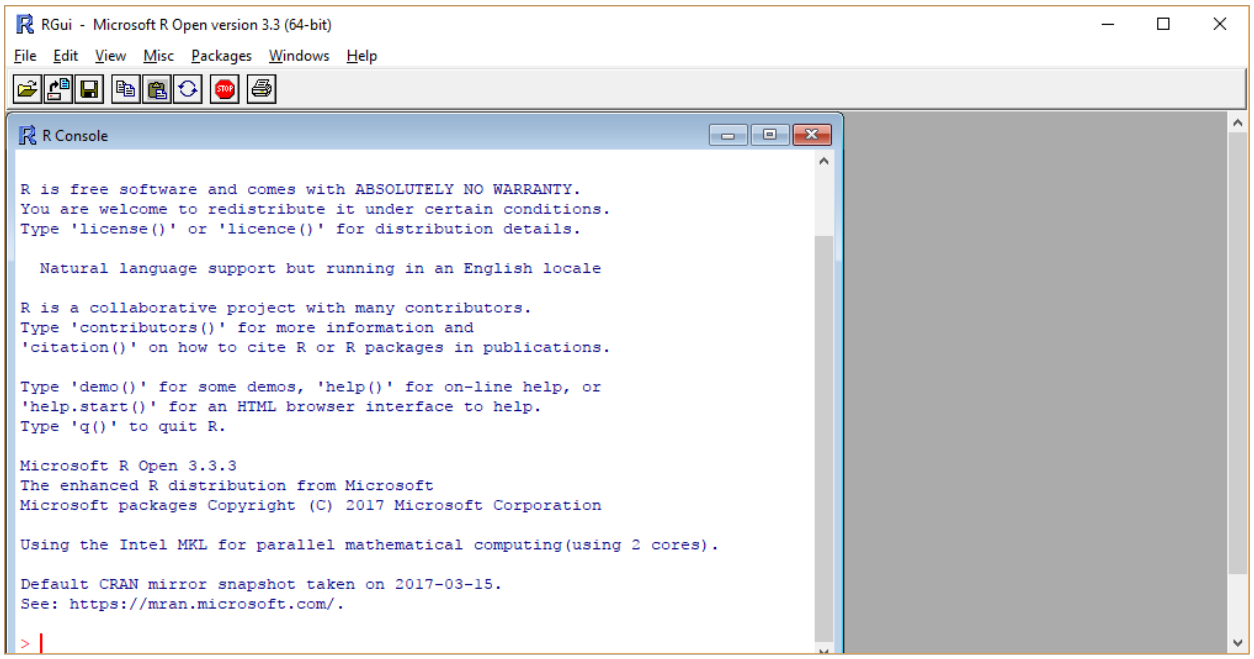


Figure 1.1: R console

> *RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.*

The full information about RStudio can be found here https://www.rstudio.com/products/rstudio/. RStudio is available in open source and commercial editions. It runs on the desktop with Windows OS, Mac OS, and Linux OS. It can also run in a browser connected to RStudio Server or RStudio Server Pro.

RStudio installation files can be downloaded from http://www.rstudio.com/products/rstudio/download/. Take note that it is recommended that you, firstly install R before trying to install RStudio. In the download link, choose the Free RStudio Desktop and click the download button. From there, you can a list of downloadable RStudio depending on the supported platforms.

From there, download the installation files. Once the download has finished, follow the simple instructions.

1.2.2 Starting RStudio

You can double click on RStudio icon in the menu or your start page on your computer desktop and you will see the RStudio interface. Take note of the R version in the RStudio Console, most probably on the right hand side of the computer screen.

1.2.3 Why RStudio?

Based on our experience and the experience of others, we feel working with RStudio helps new users learn R quicker in the beginning.

The green R GUI is way too intimidating to new users especially to those with no experience with programming language. What we want to say is that, working with R console is alright, but for majority of new users, they prefer to communicate with R using a RStudio.

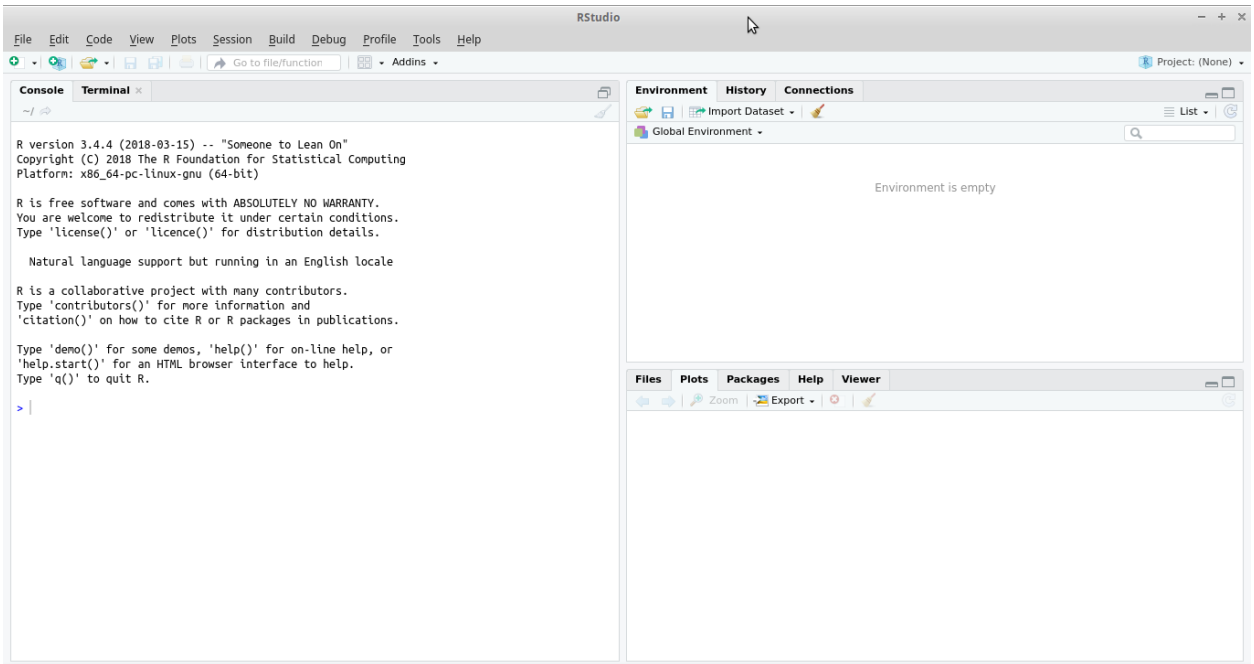


Figure 1.2: RStudio

RStudio is more than a GUI. RStudio is intergrated developement environment (IDE) for R. Other R IDEs includes Microsoft R Open. To learn more about R IDE and GUI, check out these links:

1. RStudio at https://www.rstudio.com/
2. Microsoft R Open at https://mran.microsoft.com/open
3. Tinn-R at https://sourceforge.net/p/tinn-r/wiki/Home/

1.2.4 RStudio interface

You should be able to see 3 (Figure 1.2) or 4 panes (Figure 1.3) in the layout. There are:

1. Console pane - on the left side of your computer screen. It tells you about your R, when you first start RStudio.
2. Source pane - on the upper left side of your computer screen. This will show the R script, R markdown files and other active files. The first time you start RStudio, this pane is not shown.
3. Environment and History panes - on the upper right side of your computer screen. It is where you can see the objects created by R, the codes that you have run and the connections to data sources such as databases.
4. Miscellaneous - on the lower right screen of your computer. It contains smaller tabs, Files, Plots, Packages, Help and Viewer. This tabs can list file names, show plots, show packages, display help document and view outputs.

1.2.5 Entering the codes in RStudio

To start using R, you need to open up an R script. In RStudio, click **File > New File > R Script**, or click on the icon  and choose **R Script** from the dropdown menu. You may also type the shortcut **Ctrl+Shift+N**. You should be able to get RStudio interface similar to Figure 1.3 before (four panes view).

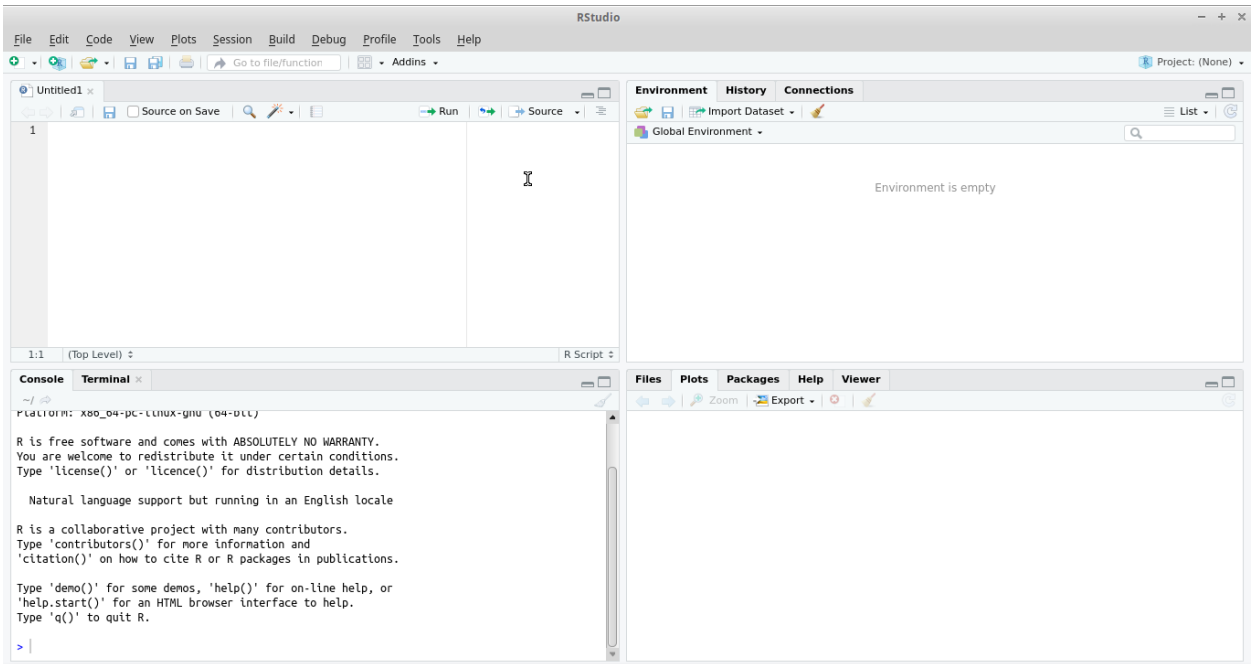


Figure 1.3: Panes in RStudio

Each line of code can be run by placing the cursor on any specific line, followed by **Ctrl+Enter**. You may also run several lines of codes by highlighting the lines followed by **Ctrl+Enter**. And, you may even run a small part in a line by selecting the part only, and **Ctrl+Enter**.

Later, you will notice some codes have hashes `#` in the codes (for starter, in the next section). `#` marks specific parts of codes as comments. These parts will not be run by R for analysis. R will recognize the lines/parts marked with `#` as comments. This is very important in any programming language. Because may end up having to deal with long lines of codes, it will be helpful to comment parts of codes to describe what the codes do. This is very advantageous to us, because we can nicely document our codes and describe whatever we do during the analysis. Think of hashtag for the social media! `#` can be used to comment the whole line or for a short commetn at the end of a line.

1.3 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.

1.3.1 Functions

R commands are in the form of `function(argument = value, argument = value, ...)`. If you are familiar with MS Excel, think of MS Excel **functions**.

Inside a function `function()`, there will be a number of **arguments**. For each argument, you may need to provide the **values**. We will see this as we go through examples later.

1.3.2 Objects

Object is like a 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).

```
# try these three lines of codes
x <- 1
y = 2
z <- x + y  # sum up x and y
```

Type the object name `x`, `y`, and `z`, you'll get the value,

```
x
```

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

```
y
```

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

```
z
```

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

Now you will notice that the `=` symbol is used to set the value (or parameter) for the argument of a function, i.e. inside the bracket after the function's name. For example, `function(argument1 = value, argument2 = value, ...)`. Thus, some people prefer using `<-` to avoid confusion with `=` for setting the values for the arguments.

1.4 Packages

R run on packages. In each package, there are function. This can be represented as **package::function()**. This packages will be installed in your home directory. To know where this directory is located, type this /home/wnarifin/R/x86_64-pc-linux-gnu-library/3.5, /usr/local/lib/R/site-library, /usr/lib/R/site-library, /usr/lib/R/library

There are two packages in R:

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

1.4.1 `base` packages

The base packages come with the installation of R. They provide basic but adequate functions to perform many standard data management, visualization and analysis.

1.4.2 `user-contributed` packages

However, in many situations, user needs to install user-contributed packages to deal with their data. These user-contributed packages are necessary to perform tasks that are not available in the base packages.

User-contributed packages allow users to perform more advanced and more complicated functions and they are contributed by R users all over the world. There are more than 12000 packages as of April 2018

For a complete list of packages, see https://cran.r-project.org/web/packages/. The packages name can be found here https://cran.r-project.org/web/packages/available_packages_by_name.html. CRAN Task

Views (https://cran.r-project.org/web/views/) aggregated all the packages according to their main tasks for examples packages to deal with:

1. *Clinical Trials*: Clinical Trial Design, Monitoring and Analysis https://cran.r-project.org/web/views/ClinicalTrials.html
2. *MetaAnalysis*: https://cran.r-project.org/web/views/MetaAnalysis.html
3. *SocialSciences*: Statistics for Social Sciences https://cran.r-project.org/web/views/SocialSciences.html
4. *Spatial*: Analysis of Spatial data https://cran.r-project.org/web/views/Spatial.html

1.4.3 Installing packages

You can install user-contributed packages through:

1. Internet (from CRAN repository).
2. Local .zip or tar.gz files.
3. Github packages.

We will now learn to install a package. You must have an active internet connection.

Using function

Basically, a function to install a package looks like this:

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

To install a package, say `car`,

1. put your cursor in the CONSOLE pane
2. type the codes below

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

3. press Ctrl + ENTER

You can also install multiple packages, for example `car` and `plyr`,

```
install.packages(c("car", "plyr"))
```

Using Packages tab

It is easier to install in RStudio. Click on **Packages > Install**. You can install many packages in one go, with the package names separated by space or comma.

In addition, you can click on **Install from:** dropdown menu and install from downloaded files (.zip, .tar.gz) by selecting **Package Archive File**.

1.4.4 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")
```

1.5 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.

1.5.1 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")
```

1.6 Getting help

We can easily access the documentation for any package or function by appending `?` before its name, for example, for help on `car` package,

```
?car
```

or for help on `mean()` function,

```
?mean
```

If you are not very sure of the exact name of the function, you may also search the documentation by keywords, for example to search functions that can obtain `mean`,

```
??mean
```

1.7 Summary

In this chapter, we installed R and R Studio. We also learned a little bit about functions and objects. We should also be able to install and load packages. Lastly, we set up the working directory.

In the next chapter, we are going to learn about loading datasets into R, managing the loaded data and also some basics on direct data entry.

Chapter 2

Data management

In this chapter, we will learn how to deal with data in R. We will learn how load, view and export data. We will also learn about selecting subsamples from the data and editing the data (creating new variables, recoding). The basics of direct data entry for tables will also be introduced.

2.1 Reading, viewing and exporting data

2.1.1 The datasets

For the purpose of doing analysis in this chapter and the rest of this book, you can download the datasets from https://wnarifin.github.io/.

2.1.2 Reading dataset

The easiest way to read a dataset into R is from .csv file,

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

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

```
library("foreign")
data <- read.spss("cholest.sav", to.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)
```

2.1.3 Viewing dataset

This is very easy in R, just type the name,

```
data
```

For a nicer view of the dataset, using `View()`

```
View(data)
```

We can also 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
```

We can view the dimension of the data (row and column),

```
dim(data)
```

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

Here, we have 80 rows (observations) and 5 columns (variables).

Next, view the names of the five variables,

```
names(data)
```

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

Using `str`, in one go we can view these details of the data,

```
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 ...
```

2.1.4 Exporting dataset from R

You can also export data into various formats using these 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')
```

2.2 Built-in datasets in R

R also have a number of built-in datasets (some are also provided by loaded packages). The datasets are often used for teaching purposes in learning new statistical analyses. 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 use `chickwts`, `women` and `infert` datasets in the next chapter.

2.3 Data structure

To completely understand the output from `str()` of `data`, there are several basics that we must understand; the variable types and the containers.

2.3.1 Variable types

Again, from

```
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 ...
```

you will notice `num` and `Factor`. These represent the variable types:

- num = numerical variable.
- Factor = categorical variable.

Each column/variable in R is a vector, which is a collection of values of the same type. You can create the vectors as follows (pay attention to the variable type):

```
data_num <- c(1,2,3,4,5); str(data_num)
```

```
##  num [1:5] 1 2 3 4 5
```

```
data_cat <- factor( c("M", "F", "M", "F", "M") ); str(data_cat)
```

```
##  Factor w/ 2 levels "F","M": 2 1 2 1 2
```

`c()` function is used to combine several values together as a vector. You may use `;` to write two lines of short codes into one line.

There are several more types of vectors, but knowing these two are sufficient for starter. You can view other types from help,

```
?typeof
```

2.3.2 Containers

Based on `str(data)`, we notice that our `data` is a data frame (`data.frame`). Basic containers that we usually use are data frame, list and matrix. These can be easily understood by examples below (utilizing our recently created vectors `data_num` and `data_cat`).

Data frame

```
data.frame(data_num, data_cat)
```

```
##   data_num data_cat
## 1        1        M
## 2        2        F
## 3        3        M
## 4        4        F
## 5        5        M
```

```
data_frame <- data.frame(data_num, data_cat); str(data_frame)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ data_num: num  1 2 3 4 5
##  $ data_cat: Factor w/ 2 levels "F","M": 2 1 2 1 2
```

List

```
list(data_num, data_cat)
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] M F M F M
## Levels: F M
```

```
data_list <- list(data_num, data_cat); str(data_list)
```

```
## List of 2
##  $ : num [1:5] 1 2 3 4 5
##  $ : Factor w/ 2 levels "F","M": 2 1 2 1 2
```

Matrix

```
matrix(data = c(data_num, data_cat), nrow = 5, ncol = 2)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    2    1
```

```
## [3,]    3    2
## [4,]    4    1
## [5,]    5    2
```

```
data_matrix <- matrix(data = c(data_num, data_cat), nrow = 5,
                      ncol = 2)
data_matrix
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    2    1
## [3,]    3    2
## [4,]    4    1
## [5,]    5    2
```

```
str(data_matrix)  # shown as numerical for both
```

```
##  num [1:5, 1:2] 1 2 3 4 5 2 1 2 1 2
```

When vectors are combined in a matrix, the factor will be turned into numeric. Matrix can only contain one type of data only. Contrast this to list.

You may have a look at array `?array` and table `?table`.

2.4 Subsetting

Subsetting means "selecting parts of data". It allows selecting only a number of variables (columns) or observations (rows) from a data frame. There are ways to do that. Basically, we can use

- `$` sign.
- `[ , ]` square brackets.
- `subset()`.

Let us use `cholest.sav` dataset,

```
library(foreign)  # to use `read.spss`
data <- read.spss("cholest.sav", to.data.frame = TRUE)
```

```
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     : num  38 35 39 36 31 38 33 36 40 34 ...
##  $ exercise: num  6 5 6 5 4 4 5 5 4 6 ...
##  $ sex     : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
##  $ categ   : Factor w/ 3 levels "Grp A","Grp B",..: 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "variable.labels")= Named chr  "cholesterol in mmol/L" "age in year"
##  "duration of exercise (hours/week)" "" ...
##  ..- attr(*, "names")= chr  "chol" "age" "exercise" "sex" ...
##  - attr(*, "codepage")= int 65001
```

```
head(data)
```

```
##   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
```

```
tail(data)
```

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

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

Let say, to select `age`, which is the second variable, first using `$`

```
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
```

and column/variable number,

```
data[ , 2]
```

```
##  [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
```

and using the name within `[]`,

```
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
```

Please keep in mind, the name is case sensitive. Thus make sure the spelling and capitalization are correct.

Then, to select the seventh observation,

```
data[7, ]
```

```
##   chol age exercise  sex categ
## 7    7  33        5 male Grp A
```

We can also choose a specific combination of row and column, let say the 73rd and `age`,

```
data[73, 2]
```

```
## [1] 45
```

```
data[73, "age"]
```

```
## [1] 45
```

2.4.2 Selecting columns and rows

Let us select `chol`, `age` and `sex`. We can use also square brackets

```
data[ , c("chol", "age", "sex")]
```

```
##    chol age  sex
## 1   6.5  38 male
## 2   6.6  35 male
## 3   6.8  39 male
## 4   6.8  36 male
## 5   6.9  31 male
## 6   7.0  38 male
## 7   7.0  33 male
## 8   7.2  36 male
## 9   7.2  40 male
## 10  7.2  34 male
```

```
## ... some data omitted.
```

or the column numbers

```
data[ , c(1:2, 4)]
```

```
##    chol age  sex
## 1   6.5  38 male
## 2   6.6  35 male
## 3   6.8  39 male
## 4   6.8  36 male
## 5   6.9  31 male
## 6   7.0  38 male
## 7   7.0  33 male
## 8   7.2  36 male
## 9   7.2  40 male
## 10  7.2  34 male
```

```
## ... some data omitted.
```

```
data[ , c(1, 2, 4)]
```

```
##    chol age  sex
## 1   6.5  38 male
## 2   6.6  35 male
## 3   6.8  39 male
## 4   6.8  36 male
## 5   6.9  31 male
## 6   7.0  38 male
## 7   7.0  33 male
## 8   7.2  36 male
## 9   7.2  40 male
## 10  7.2  34 male
```

```
## ... some data omitted.
```

selecting column 1 to 2, and column 4. Note the use of `c()` function here. It is used to combine the numbers. R needs this to let it know we want to view all these columns together. `:` means `to`. Here `1:2` means from `1 to 2`.

To select seventh to 14th observations,

```
data[7:14, ]
```

```
##    chol age exercise  sex categ
## 7   7.0  33        5 male Grp A
## 8   7.2  36        5 male Grp A
## 9   7.2  40        4 male Grp A
## 10  7.2  34        6 male Grp A
## 11  7.3  38        6 male Grp A
## 12  7.3  40        5 male Grp A
## 13  7.3  40        4 male Grp A
## 14  7.3  28        5 male Grp A
```

Then, we want to view specific combination of rows and columns. In the example below, it can be done in several ways in R.

```
data[7:14, c(2, 4)]
```

```
##    age  sex
## 7   33 male
## 8   36 male
## 9   40 male
## 10  34 male
## 11  38 male
## 12  40 male
## 13  40 male
## 14  28 male
```

```
data[7:14, c("chol", "age")]
```

```
##    chol age
## 7   7.0  33
## 8   7.2  36
## 9   7.2  40
## 10  7.2  34
## 11  7.3  38
## 12  7.3  40
## 13  7.3  40
## 14  7.3  28
```

```
data[c(1:2, 7:14), c(2, 4)]
```

```
##    age  sex
## 1   38 male
## 2   35 male
## 7   33 male
## 8   36 male
## 9   40 male
## 10  34 male
## 11  38 male
## 12  40 male
## 13  40 male
## 14  28 male
```

```
data[c(1:2, 7:14), c("chol", "age")]
```

```
##    chol age
## 1   6.5  38
```

```
## 2   6.6  35
## 7   7.0  33
## 8   7.2  36
## 9   7.2  40
## 10  7.2  34
## 11  7.3  38
## 12  7.3  40
## 13  7.3  40
## 14  7.3  28
```

Quite intrestingly, not only you can select specific rows/columns, you can also exclude them! For example, to select the rest of the variables, except `age`,

```
data[ , -2]
```

```
##    chol exercise  sex categ
## 1   6.5        6 male Grp A
## 2   6.6        5 male Grp A
## 3   6.8        6 male Grp A
## 4   6.8        5 male Grp A
## 5   6.9        4 male Grp A
## 6   7.0        4 male Grp A
## 7   7.0        5 male Grp A
## 8   7.2        5 male Grp A
## 9   7.2        4 male Grp A
## 10  7.2        6 male Grp A
```

```
## ... some data omitted.
```

using `-` sign before the column number. Please note that this syntax is only possible with column/row numbers, you cannot use it in reference to the names.

You can try the following,

```
data[-c(1:35, 40:75), ]
```

```
##    chol age exercise    sex categ
## 36  8.1  38        4   male Grp B
## 37  8.2  45        6   male Grp B
## 38  8.2  36        4   male Grp B
## 39  8.3  31        4   male Grp B
## 76  9.5  52        4 female Grp C
## 77  9.6  35        4 female Grp C
## 78  9.8  43        3 female Grp C
## 79  9.9  47        3 female Grp C
## 80 10.0  44        3 female Grp C
```

```
data[-c(1:35, 40:75), -c(1:2, 4)]
```

```
##    exercise categ
## 36        4 Grp B
## 37        6 Grp B
## 38        4 Grp B
## 39        4 Grp B
## 76        4 Grp C
## 77        4 Grp C
## 78        3 Grp C
## 79        3 Grp C
```

```
## 80        3 Grp C
```

to explude the specific rows and columns. This is a very important and neat syntax whenever we are want to exclude some observations or variables for during the analysis.

2.4.3 Selecting based on logical expressions

Practically, we want to c4oose observations based on certain criteria, for example those aged more than 35 year old, only females subjects and so on. This is easy with `subset()`, for example `age > 45`,

```
subset(data, age > 45)
```

```
##    chol age exercise    sex categ
## 17  7.4  49        5   male Grp A
## 22  7.6  46        4   male Grp A
## 46  8.5  46        4 female Grp B
## 58  8.8  47        3 female Grp B
## 62  8.9  48        3 female Grp C
## 63  8.9  47        4 female Grp C
## 66  9.0  49        3 female Grp C
## 70  9.3  48        3 female Grp C
## 76  9.5  52        4 female Grp C
## 79  9.9  47        3 female Grp C
```

```
subset(data, sex == "female")
```

```
##    chol age exercise    sex categ
## 41  8.3  44        4 female Grp B
## 42  8.3  35        5 female Grp B
## 43  8.4  40        4 female Grp B
## 44  8.4  37        6 female Grp B
## 45  8.5  33        4 female Grp B
## 46  8.5  46        4 female Grp B
## 47  8.5  42        5 female Grp B
## 48  8.5  40        4 female Grp B
## 49  8.5  45        4 female Grp B
## 50  8.5  42        5 female Grp B
```

```
## ... some data omitted.
```

alternatively, we can use square brackets with a number of variants for the same subset,

```
data[data$age > 45, ]
data[data[ , "age"] > 45, ]
data[data[ , 2] > 45, ]
```

```
##    chol age exercise    sex categ
## 41  8.3  44        4 female Grp B
## 42  8.3  35        5 female Grp B
## 43  8.4  40        4 female Grp B
## 44  8.4  37        6 female Grp B
## 45  8.5  33        4 female Grp B
## 46  8.5  46        4 female Grp B
## 47  8.5  42        5 female Grp B
## 48  8.5  40        4 female Grp B
## 49  8.5  45        4 female Grp B
## 50  8.5  42        5 female Grp B
```

```
## ... some data omitted.
```

but the syntax it quite messy; we have to repeat `data` twice here, and the sysntax is difficult to grasp. But knowing the syntax is useful, just in case subset doesn't work.

Logical expressions are several, which are - `==` equal. - `>=` more than or equal. - `<=` less than or equal. - `>` more than. - `<` less than. - `!=` not equal.

You can play around with these expressions by changing `>` to these expressions in the example above.

`subset()` has this simple syntax, `subset(data, subset/row_expression, select = column)`. It can be used to select specific variables, for example,

```
subset(data, select = c("chol", "age", "sex"))
subset(data, select = c(chol, age, sex))
```

```
##    chol age  sex
## 1   6.5  38 male
## 2   6.6  35 male
## 3   6.8  39 male
## 4   6.8  36 male
## 5   6.9  31 male
## 6   7.0  38 male
## 7   7.0  33 male
## 8   7.2  36 male
## 9   7.2  40 male
## 10  7.2  34 male
```

```
## ... some data omitted.
```

by omitting the `subset =` argument. Notice that you don't even need to quote names using `" "` in `select =` argument parameter.

Interestingly, using `subset()`, you can apply `:` sign to names,

```
subset(data, select = chol:sex)
```

```
##    chol age exercise  sex
## 1   6.5  38        6 male
## 2   6.6  35        5 male
## 3   6.8  39        6 male
## 4   6.8  36        5 male
## 5   6.9  31        4 male
## 6   7.0  38        4 male
## 7   7.0  33        5 male
## 8   7.2  36        5 male
## 9   7.2  40        4 male
## 10  7.2  34        6 male
```

```
## ... some data omitted.
```

Now let us try this, select those aged more or equal to 45, and `age`, `sex` variables,

```
subset(data, age >= 45, select = c(age, sex))
```

```
##    age    sex
## 17  49   male
## 22  46   male
## 37  45   male
## 46  46 female
## 49  45 female
```

```
## 51  45 female
## 58  47 female
## 62  48 female
## 63  47 female
## 66  49 female
## 70  48 female
## 72  45 female
## 73  45 female
## 75  45 female
## 76  52 female
## 79  47 female
```

Then try with a combination of expressions to select rows, for example those aged less or equal to 35 and/or female, and `chol`, `age` and `sex` variables,

```r
subset(data, age <= 35 & sex == "female", select = c(age, sex))
```

```
##    age    sex
## 42  35 female
## 45  33 female
## 53  34 female
## 61  30 female
## 67  31 female
## 71  34 female
## 77  35 female
```

```r
subset(data, age <= 35 | sex == "female", select = c(age, sex))
```

```
##    age    sex
## 2   35   male
## 5   31   male
## 7   33   male
## 10  34   male
## 14  28   male
## 18  29   male
## 21  34   male
## 25  32   male
## 33  35   male
## 39  31   male
## 40  34   male
## 41  44 female
## 42  35 female
## 43  40 female
## 44  37 female
## 45  33 female
## 46  46 female
## 47  42 female
## 48  40 female
## 49  45 female
## 50  42 female
## 51  45 female
## 52  38 female
## 53  34 female
## 54  44 female
## 55  39 female
## 56  38 female
```

```
## 57  39 female
## 58  47 female
## 59  41 female
## 60  44 female
## 61  30 female
## 62  48 female
## 63  47 female
## 64  42 female
## 65  42 female
## 66  49 female
## 67  31 female
## 68  38 female
## 69  38 female
## 70  48 female
## 71  34 female
## 72  45 female
## 73  45 female
## 74  36 female
## 75  45 female
## 76  52 female
## 77  35 female
## 78  43 female
## 79  47 female
## 80  44 female
```

Notice we used `&` for AND and `|` for OR in between the expressions.

Run `levels()` to remind us the available factor levels for `sex`,

```
levels(data$sex)
```

```
## [1] "female" "male"
```

For the sake of completeness, you can try the following codes using `[ , ]` and `$` in place of `subset()`,

```
data[data$age <=35 & data$sex == "female", c("age", "sex")]
```

```
##    age    sex
## 42  35 female
## 45  33 female
## 53  34 female
## 61  30 female
## 67  31 female
## 71  34 female
## 77  35 female
```

```
data[data$age <=35 | data$sex == "female", c("age", "sex")]
```

```
##    age    sex
## 2   35   male
## 5   31   male
## 7   33   male
## 10  34   male
## 14  28   male
## 18  29   male
## 21  34   male
## 25  32   male
## 33  35   male
```

```
## 39  31   male
## 40  34   male
## 41  44 female
## 42  35 female
## 43  40 female
## 44  37 female
## 45  33 female
## 46  46 female
## 47  42 female
## 48  40 female
## 49  45 female
## 50  42 female
## 51  45 female
## 52  38 female
## 53  34 female
## 54  44 female
## 55  39 female
## 56  38 female
## 57  39 female
## 58  47 female
## 59  41 female
## 60  44 female
## 61  30 female
## 62  48 female
## 63  47 female
## 64  42 female
## 65  42 female
## 66  49 female
## 67  31 female
## 68  38 female
## 69  38 female
## 70  48 female
## 71  34 female
## 72  45 female
## 73  45 female
## 74  36 female
## 75  45 female
## 76  52 female
## 77  35 female
## 78  43 female
## 79  47 female
## 80  44 female
```

```
data[data$age <=35 & data$sex == "female", ]$age  # view `age` only
```

```
## [1] 35 33 34 30 31 34 35
```

```
# using [ , ] and $ combination.
```

Actually, the most important reason why we bother with subsetting is that we can easy assign a subset of the dataset to a new data object. This will make our analysis easier when we deal with large datasets. For example,

```
data_short <- data[1:20, c("age", "sex")]
data_short
```

```
##    age  sex
## 1   38 male
## 2   35 male
## 3   39 male
## 4   36 male
## 5   31 male
## 6   38 male
## 7   33 male
## 8   36 male
## 9   40 male
## 10  34 male
## 11  38 male
## 12  40 male
## 13  40 male
## 14  28 male
## 15  37 male
## 16  38 male
## 17  49 male
## 18  29 male
## 19  40 male
## 20  38 male
```

```
( data_short <- data[1:20, c("age", "sex")] )
```

```
##    age  sex
## 1   38 male
## 2   35 male
## 3   39 male
## 4   36 male
## 5   31 male
## 6   38 male
## 7   33 male
## 8   36 male
## 9   40 male
## 10  34 male
## 11  38 male
## 12  40 male
## 13  40 male
## 14  28 male
## 15  37 male
## 16  38 male
## 17  49 male
## 18  29 male
## 19  40 male
## 20  38 male
```

```
str(data_short)
```

```
## 'data.frame':    20 obs. of  2 variables:
##  $ age: num  38 35 39 36 31 38 33 36 40 34 ...
##  $ sex: Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
```

There's a new trick here. If we want to view the assigned data in one step, include `( )` the assignment codes in between the round brackets.

2.5 Sorting data

At times, we want to view the data in ascending or descending order, especially for numerical variables. Let us start with `sort()`. `sort()` is used on a vector, for example here the vector of `age`,

```
sort(data$age)  # values in ascending order
```

```
##  [1] 28 29 30 31 31 31 32 33 33 34 34 34 34 34 35 35 35 35 36 36 36 36 37
## [24] 37 38 38 38 38 38 38 38 38 38 38 38 38 38 39 39 39 39 39 39 40 40 40
## [47] 40 40 40 40 40 41 42 42 42 42 42 42 43 43 44 44 44 44 45 45 45 45 45
## [70] 45 46 46 47 47 47 48 48 49 49 52
```

```
sort(data$age, decreasing = TRUE)  # values in descending order
```

```
##  [1] 52 49 49 48 48 47 47 47 46 46 45 45 45 45 45 45 44 44 44 44 43 43 42
## [24] 42 42 42 42 42 41 40 40 40 40 40 40 40 40 39 39 39 39 39 39 38 38 38
## [47] 38 38 38 38 38 38 38 38 38 38 37 37 36 36 36 36 35 35 35 35 34 34 34
## [70] 34 34 33 33 32 31 31 31 30 29 28
```

Next, `order()` is used on data frame. `order()` gives the ordering index in ascending order. This can be used to provide the row number whenever we use `[ , ]` to subset the data. Here we order by `age`,

```
order(data$age)  # gives the index in ascending order
data[order(data$age), ]  # rows follow the index
```

```
##  [1] 14 18 61  5 39 67 25  7 45 10 21 40 53 71  2 33 42 77  4  8 38 74 15
## [24] 44  1  6 11 16 20 24 29 34 36 52 56 68 69  3 30 31 32 55 57  9 12 13
## [47] 19 28 35 43 48 59 23 27 47 50 64 65 26 78 41 54 60 80 37 49 51 72 73
## [70] 75 22 46 58 63 79 62 70 17 66 76
```

```
##    chol age exercise    sex categ
## 14  7.3  28        5   male Grp A
## 18  7.4  29        5   male Grp A
## 61  8.8  30        3 female Grp C
## 5   6.9  31        4   male Grp A
## 39  8.3  31        4   male Grp B
## 67  9.1  31        2 female Grp C
## 25  7.8  32        5   male Grp A
## 7   7.0  33        5   male Grp A
## 45  8.5  33        4 female Grp B
## 10  7.2  34        6   male Grp A
```

```
## ... some data omitted.
```

```
data[order(data$age, decreasing = TRUE), ]  # descending order
```

```
##    chol age exercise    sex categ
## 76  9.5  52        4 female Grp C
## 17  7.4  49        5   male Grp A
## 66  9.0  49        3 female Grp C
## 62  8.9  48        3 female Grp C
## 70  9.3  48        3 female Grp C
## 58  8.8  47        3 female Grp B
## 63  8.9  47        4 female Grp C
## 79  9.9  47        3 female Grp C
## 22  7.6  46        4   male Grp A
## 46  8.5  46        4 female Grp B
```

```
## ... some data omitted.
```

Now, we want to order by `age` and `exercise`. The ordering starts from the last variable in the list (`data$age`) to the first variable in the list (`data$exercise`),

```r
order(data$exercise, data$age)  # order by age, then exercise
data[order(data$exercise, data$age), ]  # ascending order
```

```
##  [1] 67 61 53 33 68 69 55 57 19 78 80 72 73 58 79 62 70 66  5 39 45 71 77
## [24] 38 74  6 16 24 34 36 56  9 13 43 48 59 64 65 26 41 54 60 49 51 75 22
## [47] 46 63 76 14 18 25  7 21 40  2 42  4  8 15 20 29 52 30 31 32 12 28 35
## [70] 23 27 47 50 17 10 44  1 11  3 37
```

```
##    chol age exercise    sex categ
## 67  9.1  31        2 female Grp C
## 61  8.8  30        3 female Grp C
## 53  8.6  34        3 female Grp B
## 33  8.0  35        3   male Grp B
## 68  9.2  38        3 female Grp C
## 69  9.2  38        3 female Grp C
## 55  8.7  39        3 female Grp B
## 57  8.7  39        3 female Grp B
## 19  7.5  40        3   male Grp A
## 78  9.8  43        3 female Grp C
```

```
## ... some data omitted.
```

```r
data[order(data$exercise, data$age, decreasing = TRUE), ]  # descending order
```

```
##    chol age exercise    sex categ
## 37  8.2  45        6   male Grp B
## 3   6.8  39        6   male Grp A
## 1   6.5  38        6   male Grp A
## 11  7.3  38        6   male Grp A
## 44  8.4  37        6 female Grp B
## 10  7.2  34        6   male Grp A
## 17  7.4  49        5   male Grp A
## 23  7.6  42        5   male Grp A
## 27  7.8  42        5   male Grp B
## 47  8.5  42        5 female Grp B
```

```
## ... some data omitted.
```

```r
data[order(data$exercise, data$age, decreasing = c(TRUE, FALSE)), ]
# age ascending order, exercise descending order
```

```
##    chol age exercise    sex categ
## 10  7.2  34        6   male Grp A
## 44  8.4  37        6 female Grp B
## 1   6.5  38        6   male Grp A
## 11  7.3  38        6   male Grp A
## 3   6.8  39        6   male Grp A
## 37  8.2  45        6   male Grp B
## 14  7.3  28        5   male Grp A
## 18  7.4  29        5   male Grp A
## 25  7.8  32        5   male Grp A
## 7   7.0  33        5   male Grp A
```

```
## ... some data omitted.
```

Now, we will use `arrange()` from `plyr` package. Make sure you installed `plyr` beforehand. `arrange()` has a simpler syntax, `arrange(dataset, variables)`, and can be applied easily as shown below,

```
library(plyr)
arrange(data, exercise, age)  # all ascending
```

```
##    chol age exercise    sex categ
## 1   9.1  31        2 female Grp C
## 2   8.8  30        3 female Grp C
## 3   8.6  34        3 female Grp B
## 4   8.0  35        3   male Grp B
## 5   9.2  38        3 female Grp C
## 6   9.2  38        3 female Grp C
## 7   8.7  39        3 female Grp B
## 8   8.7  39        3 female Grp B
## 9   7.5  40        3   male Grp A
## 10  9.8  43        3 female Grp C

## ... some data omitted.
```

```
arrange(data, desc(exercise), age)  # age ascending order,
# exercise descending order
```

```
##    chol age exercise    sex categ
## 1   7.2  34        6   male Grp A
## 2   8.4  37        6 female Grp B
## 3   6.5  38        6   male Grp A
## 4   7.3  38        6   male Grp A
## 5   6.8  39        6   male Grp A
## 6   8.2  45        6   male Grp B
## 7   7.3  28        5   male Grp A
## 8   7.4  29        5   male Grp A
## 9   7.8  32        5   male Grp A
## 10  7.0  33        5   male Grp A

## ... some data omitted.
```

2.6 Editing data

Using the same dataset `cholest.sav`, we want to add new variables based on the existing variables.

2.6.1 Creating a new variable

It is easy to create a new variable in R. We only have to decide on a name for the new variable, and the include it with `$name` to the data frame.

For example, to create age in months `age_month`, we multiply the existing variable age in years `age` by 12, then assign the values to 'data$age_month as follows,

```
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
```

2.6.2 Recoding into new variables

From a numerical variable

From the numerical `age` variable, let say we want to break into three groups: less than 40, 40-50 and more than 50.

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

What is meant by `breaks = c(-Inf, 40, 50, Inf)` here is "from minus infinity to below 40, between 40 to 50, from above 50 to infinity".

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

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

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

```
##  Factor w/ 3 levels "< 40","40-50",..: 1 1 1 1 1 1 1 1 1 1 ...
```

From a categorical variable

Using the recently created `age_cat` variable,

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

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

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

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

Only one observation is labeled as `> 50`. We want to combine `40-50` with `> 50`. Make sure you installed `car` package to use `recode()` function in the codes below.

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

Pay attention to the use of `""` and `''` in `recode()`.

```
levels(data$age_cat1)
```

```
## [1] "< 40"       "40 & above"
```

```
table(data$age_cat1)  # combined
```

```
##
##       < 40 40 & above
##         51         29
```

2.6.3 Removing variables and observations

You can easily remove variables and observations by using subsetting method above. Here we want to consider another approach to do this in R. For example you only need to remove one variable, let say `age_month`, you can assign the variable to `NULL`,

```
data$age_month <- NULL
names(data)
```

```
## [1] "chol"     "age"      "exercise" "sex"      "categ"    "age_cat"
## [7] "age_cat1"
```

then let say we want to remove `exercise` and `categ`,

```
data[c("exercise", "categ")] <- NULL
names(data)
```

```
## [1] "chol"     "age"      "sex"      "age_cat"  "age_cat1"
```

We can also easily select which subjects we want to keep or remove in the data object. Let say we want to remove subject number 20, 39 and 71 from our data frame, we assign `NA` (not available) to the data belonging to these observations,

```
dim(data)
```

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

```
data[c(20, 39, 71), ] <- NA
```

then we use `na.omit()` to remove the observations,

```
data <- na.omit(data)
dim(data)
```

```
## [1] 77  5
```

We will learn more about handling missing observations (`NA`) below.

But this approach of using `NA` is not as good as subsetting, which is simpler,

```
dim(data)
```

```
## [1] 77  5
```

```
data <- data[-c(20, 39, 71), ]
dim(data)
```

```
## [1] 74  5
```

2.7 Direct data entry

We can enter short data directly using `read.table`. This is very useful whenever we want to analyze data from tables, for example those obtained from research articles, and also data provided in textbooks.

For example, you can easily create a standard data frame, consisting of patient's ID, group and BMI for six patients,

ID	Group	BMI
1	Fat	30
2	Fat	31

ID	Group	BMI
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
```

Recall what you learned in **Containers** section, we combined numerical `num_data` and factor `cat_data` vectors into a data frame namely `data_frame` right? As you might have guessed, you can also create the data frame by combining the vectors,

```
ID <- 1:6
Group <- c("Fat", "Fat", "Fat", "Thin", "Thin", "Thin")
BMI <- c(30, 31, 32, 20, 19, 18)
data_frame <- data.frame(ID, Group, BMI)
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  : num  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
```

However, we find this approach less intuitive because we have to enter by variables/vectors. Take note that we used a shortcut here to generate numbers from one to six for `ID`. So instead of `ID <- c(1, 2, 3, 4, 5, 6)`, we can just write `ID <- 1:6`.

You can also easily enter tabulated data in R, as shown below,

	Cancer	No Cancer
Smoker	80	10
Non-smoker	5	100

```
data_table <- read.table(header = FALSE, text = "
80 10
5 100
                        ")
colnames(data_table) <- c("Cancer", "No Cancer")
rownames(data_table) <- c("Smoker", "Non-smoker")
str(data_table)  # data_table is a data frame
```

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

The numbers are separated by space. We set the row and column names by `rownames` and `colnames` respectively. This will create a data frame.

However, to create a proper table in R, we need a few more steps. We need to convert the data frame to a matrix. Remember, a matrix container will force the data to be of the same type only. Then, we will turn the matrix into a proper table. The steps are shown below,

```
data_table <- as.matrix(data_table) # convert data_table to matrix
data_table <- as.table(data_table) # then to table
str(data_table)  # data_table is now a table
```

```
##  'table' int [1:2, 1:2] 80 5 10 100
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2] "Smoker" "Non-smoker"
##   ..$ : chr [1:2] "Cancer" "No Cancer"
```

```
data_table
```

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

2.8 Miscellaneous

Next, we will go through a number of additional important data management skills.

2.8.1 Sums of the existing variables

Among the most important functions in R are related to obtaining sums. Here, we load `mtf.csv` dataset (Arifin & Yusoff, 2017). The dataset consists of two multiple true-false questions, with five independent statements each. The correct answers are awarded one (coded as 1) mark each, and incorrect answers are awarded zero (coded as 0) mark each.

```
mtf <- read.csv("mtf.csv")
```

The basic `sum()` works on a vector, e.g. `mtf$Q1A`,

```
sum(mtf$Q1A)
```

```
## [1] 111
```

It gives the total number of correct answers for question 1A.

Next, we obtain the total number of correct answers for all respondents, i.e. by rows. Hence we use `rowSums()`,

```
rowSums(mtf)
```

```
##   [1]  3  5  5  7  8  8  6  7  7  8  8  8  8  8  5 10  5  4  4  7  9  1  9
##  [24]  7  5  4  7  3  7  0  6  5  6  7  7  8 10  7  8  6  6  8  9 10  3  6
##  [47]  6  9  6  6  5  7  8  6 10  4  8  9  8  9  9  5 10  5  5  9  5  6  7
##  [70]  5  6  6  9  3  7 10  5  6  7  7  7  8  6  9  7  6  6  9  6  7  7 10
##  [93]  0  5 10  9  6  6  3  7 10  3  6  9  6  6  7  8  8  6  6  6  8  7  9
## [116]  7  4  8  8  9  8  8  5  8  6 10  7  6  5  3  6  7  7  6  9  6  8  7
## [139]  7  7  6  5  5  8 10  8  8  6  5  6  8  7  6 10  6  7  6  6  4  2
```

While `sum()` gives use the total per vector, we can easily obtain for all questions (columns) by `colSums()`,

```
colSums(mtf)
```

```
## Q1A Q1B Q1C Q1D Q1E Q2A Q2B Q2C Q2D Q2E
## 111 119 100  95 134 120 117 105  84  83
```

We can also easily create new variables `total_mark` and `percent` as follows,

```
mtf$total_mark <- rowSums(mtf[ , 1:10])
mtf$percent <- (mtf$total_mark/10)*100
head(mtf)
```

```
##   Q1A Q1B Q1C Q1D Q1E Q2A Q2B Q2C Q2D Q2E total_mark percent
## 1   1   0   0   0   0   0   1   1   0   0          3      30
## 2   1   0   0   0   1   0   0   1   1   1          5      50
## 3   0   1   0   0   1   1   0   1   1   0          5      50
## 4   1   1   0   1   1   0   1   0   1   1          7      70
## 5   1   1   1   0   1   1   1   1   1   0          8      80
## 6   0   1   1   1   1   0   1   1   1   1          8      80
```

2.8.2 Handling missing observations (`NA`/not available)

We start by generating a data frame with `NA` and " " (empty entry),

```
data_na <- read.table(header = T, sep = ",", text = "
ID, age, gender
8110, 20, M
8110, 20, M
1627, 30,
1234, 23, F
4567, , F
4567, 12, F
")  # we use comma separated values in this example
str(data_na); data_na
```

```
## 'data.frame':    6 obs. of  3 variables:
```

```
##  $ ID    : int  8110 8110 1627 1234 4567 4567
##  $ age   : int  20 20 30 23 NA 12
##  $ gender: Factor w/ 3 levels " "," F"," M": 3 3 1 2 2 2
```

```
##     ID age gender
## 1 8110  20      M
## 2 8110  20      M
## 3 1627  30
## 4 1234  23      F
## 5 4567  NA      F
## 6 4567  12      F
```

There will be a `NA` in `age` and " " category in gender,

```
summary(data_na)  # NA in age, " " category in gender
```

```
##        ID            age     gender
##  Min.   :1234   Min.   :12     :1
##  1st Qu.:2362   1st Qu.:20    F:3
##  Median :4567   Median :20    M:2
##  Mean   :4702   Mean   :21
##  3rd Qu.:7224   3rd Qu.:23
##  Max.   :8110   Max.   :30
##                 NA's   :1
```

We now verify whether there is `NA` in the data frame,

```
anyNA(data_na)
```

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

`TRUE`, yes there is a NA, it is located in,

```
is.na(data_na)
```

```
##         ID   age gender
## [1,] FALSE FALSE  FALSE
## [2,] FALSE FALSE  FALSE
## [3,] FALSE FALSE  FALSE
## [4,] FALSE FALSE  FALSE
## [5,] FALSE  TRUE  FALSE
## [6,] FALSE FALSE  FALSE
```

and you notice here " " is treated as a category, not `NA` for categorical variable `gender`.

First we omit the observation containing `NA`,

```
dim(data_na)  # 6 observations
```

```
## [1] 6 3
```

```
data_na_clean <- na.omit(data_na)
dim(data_na)  # 5 observations
```

```
## [1] 6 3
```

```
summary(data_na_clean)
```

```
##        ID            age     gender
##  Min.   :1234   Min.   :12     :1
##  1st Qu.:1627   1st Qu.:20    F:2
```

```
##  Median :4567   Median :20    M:2
##  Mean   :4730   Mean   :21
##  3rd Qu.:8110   3rd Qu.:23
##  Max.   :8110   Max.   :30
```

Now we handle " " by excluding the observation containing empty gender information,

```
data_na_cleaner <- data_na_clean[data_na_clean$gender != " ", ]
data_na_cleaner
```

```
##     ID age gender
## 1 8110  20      M
## 2 8110  20      M
## 4 1234  23      F
## 6 4567  12      F
```

2.8.3 Handling duplicates

Let say we have this data:

```
duplicate <- read.table(header = T, text = "
ID age gender
8110 20 M
8110 20 M
1627 30 M
1234 23 F
4567 12 F
4567 12 F
")
str(duplicate); duplicate
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ ID    : int  8110 8110 1627 1234 4567 4567
##  $ age   : int  20 20 30 23 12 12
##  $ gender: Factor w/ 2 levels "F","M": 2 2 2 1 1 1
```

```
##     ID age gender
## 1 8110  20      M
## 2 8110  20      M
## 3 1627  30      M
## 4 1234  23      F
## 5 4567  12      F
## 6 4567  12      F
```

We use and `anyDuplicated()` and `duplicated()`, functions in `base` R,

```
anyDuplicated(duplicate)  # 2 duplicates
```

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

and we found two duplicates.

We check for duplicated `ID`,

```
dupli <- duplicate[duplicated(duplicate), "ID"]
dupli
```

```
## [1] 8110 4567
```

view the duplicated entries,

```
duplicate[duplicate$ID == dupli, ]
```

```
##     ID age gender
## 1 8110  20      M
## 6 4567  12      F
```

and view entries minus the duplicated ones by,

```
duplicate[duplicate$ID != dupli, ]
```

```
##     ID age gender
## 2 8110  20      M
## 3 1627  30      M
## 4 1234  23      F
## 5 4567  12      F
```

or this way,

```
duplicate[!duplicated(duplicate), ]
```

```
##     ID age gender
## 1 8110  20      M
## 3 1627  30      M
## 4 1234  23      F
## 5 4567  12      F
```

Then you can easily keep data frame with the unduplicated entries,

```
noduplicate <- duplicate[data$ID != dupli, ]
```

2.9 Summary

In this chapter, we learned how to handle data in R, which is very flexible. We learned how load, view and export data. We also learned how select subsamples from the data, and how to edit the data (creating new variables, recoding). Then we learned some basics on direct data entry for tables.

In the next chapter, we are going to learn about how to explore the variables by means of basic descriptive statistics.

Chapter 3

Descriptive statistics

In this chapter, we will learn how to obtain a number of important descriptive statistics. The statistics will be obtained based on the variable types and groups. We will also learn how to perform cross-tabulation.

3.1 One variable

We will start by using `chickwts` dataset that contains both numerical (`weight`) and categorical (`feed`) variables. We 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
```

Next, 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 ...
```

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

3.1.1 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,

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

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

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

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

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

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

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

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

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

and obtain its standard deviation (SD)

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

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

variance,

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

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

quantile,

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

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

and interquartile range (IQR)

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

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

There are nine types of quantile algorithms in R (for `quantile()` and `IQR`), t()he 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),

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

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

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

```r
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

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

```r
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
```

3.1.2 A categorical variable

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

We obtain the count of the `feed` variable,

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

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

```r
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.

```r
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.

3.2 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.

3.2.1 Numerical variables

Let us use `women` dataset and explore the dataset,

```
head(women)
```

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

```
tail(women)
```

```
##    height weight
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

```r
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`,

```r
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` package),

```r
library(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
```

3.2.2 Categorical variables

Let us use `infert` dataset,

```r
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
```

```r
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. `lapply()` 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 the variable).

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
```

```r
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
```

```r
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)`.

3.3 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.

3.3.1 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

```r
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
```

```r
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()`

```r
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
```

```r
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
```

```r
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
```

```r
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()`.

3.3.2 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`

```
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
```

3.4 Customizing text outputs

Text outputs will look nicer by combining every bits and parts of the outputs into custom-made texts and tables. There are a number of ways to achieve this. We will do this by utilizing basic functions.

We again use `cholest.sav` from previous chapter. Load the dataset as `cholest`,

```
library(foreign)
cholest <- read.spss("cholest.sav", to.data.frame = TRUE)
```

```
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     : num  38 35 39 36 31 38 33 36 40 34 ...
##  $ exercise: num  6 5 6 5 4 4 5 5 4 6 ...
##  $ sex     : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
##  $ categ   : Factor w/ 3 levels "Grp A","Grp B",..: 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "variable.labels")= Named chr  "cholesterol in mmol/L" "age in year"
##  "duration of exercise (hours/week)" "" ...
##  ..- attr(*, "names")= chr  "chol" "age" "exercise" "sex" ...
##  - attr(*, "codepage")= int 65001
```

3.4.1 cbind and rbind

We were introduced to `cbind()` earlier in this chapter. We will further use `cbind()` to customize our outputs. In addition, we will use its sibling, `rbind()`.

Let say we want to view mean, standard deviation (SD) and sample size (n) together,

```
mean(cholest$age)
```

```
## [1] 39.475
```

```
sd(cholest$age)
```

```
## [1] 5.128661
```

```
length(cholest$age)
```

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

First utilize the basic `cbind()`,

```
cbind(mean = mean(cholest$age), sd = sd(cholest$age),
      n = length(cholest$age))
```

```
##        mean       sd  n
## [1,] 39.475 5.128661 80
```

and then we can give it a proper row name,

```
chol_c <- cbind(mean = mean(cholest$age), sd = sd(cholest$age),
                n = length(cholest$age))
rownames(chol_c) <- "Cholestrol"
chol_c
```

```
##              mean       sd  n
## Cholestrol 39.475 5.128661 80
```

Compare `cbind()` with `rbind()`. `rbind()` combines the values by row, while `cbind()` combines the values by column. Thus, you can customize the outputs based on your preference.

```
rbind(mean = mean(cholest$age), sd = sd(cholest$age),
      n = length(cholest$age))
```

```
##           [,1]
## mean 39.475000
## sd    5.128661
## n    80.000000
```

```
chol_r = rbind(mean = mean(cholest$age), sd = sd(cholest$age),
               n = length(cholest$age))
colnames(chol_r) <- "Cholestrol"
chol_r
```

```
##      Cholestrol
## mean  39.475000
## sd     5.128661
## n     80.000000
```

Now we can add in `lappy()` to come up with vectors of mean and SD for the selected variables. `cbind()` and `rbind()` can also combine vectors,

```r
mean_cholest <- lapply(cholest[, c("chol", "age", "exercise")], mean)
sd_cholest <- lapply(cholest[, c("chol", "age", "exercise")], sd)
cbind(mean = mean_cholest, SD = sd_cholest,
      n = lengths(cholest[, c("chol", "age", "exercise")]))
```

```
##          mean   SD        n
## chol     8.23   0.8386849 80
## age      39.475 5.128661  80
## exercise 4.225  0.9136794 80
```

```r
rbind(mean = mean_cholest, SD = sd_cholest,
      n = lengths(cholest[, c("chol", "age", "exercise")]))
```

```
##      chol      age      exercise
## mean 8.23      39.475   4.225
## SD   0.8386849 5.128661 0.9136794
## n    80        80       80
```

Now, we can edit the variable names to make the results more presentable,

```r
names(mean_cholest) <- c("Cholestrol", "Age", "Exercise")
cbind(mean = mean_cholest, SD = sd_cholest,
      n = lengths(cholest[, c("chol", "age", "exercise")]))
```

```
##            mean   SD        n
## Cholestrol 8.23   0.8386849 80
## Age        39.475 5.128661  80
## Exercise   4.225  0.9136794 80
```

```r
rbind(mean = mean_cholest, SD = sd_cholest,
      n = lengths(cholest[, c("chol", "age", "exercise")]))
```

```
##      Cholestrol Age      Exercise
## mean 8.23       39.475   4.225
## SD   0.8386849  5.128661 0.9136794
## n    80         80       80
```

Now, let us try `cbind()` and `rbind()` on categorical variables, `sex` and `categ`,

```r
count_cholest <- sapply(cholest[c("sex", "categ")], summary)
count_cholest
```

```
## $sex
## female   male
##     40     40
##
## $categ
## Grp A Grp B Grp C
##    25    33    22
```

```r
perc_cholest <- sapply(cholest[c("sex", "categ")], function(x) summary(x)/length(x)*100)
perc_cholest
```

```
## $sex
## female   male
##     50     50
##
## $categ
```

```
## Grp A Grp B Grp C
## 31.25 41.25 27.50
```

then we list down by variables,

```
list(Sex = cbind(n = count_cholest$sex, "%" = perc_cholest$sex),
     Category = cbind(n = count_cholest$categ,
                      "%" = perc_cholest$categ))
```

```
## $Sex
##         n  %
## female 40 50
## male   40 50
##
## $Category
##        n     %
## Grp A 25 31.25
## Grp B 33 41.25
## Grp C 22 27.50
```

3.4.2 data.frame() and matrix()

These two functions work like `cbind()`. They are very handy to present results to look like a nice table.

Using `data.frame()`,

```
data.frame(mean = mean_cholest, SD = sd_cholest,
      n = lengths(cholest[, c("chol", "age", "exercise")]))
```

```
##          mean.Cholestrol mean.Age mean.Exercise   SD.chol   SD.age
## chol                8.23   39.475         4.225 0.8386849 5.128661
## age                 8.23   39.475         4.225 0.8386849 5.128661
## exercise            8.23   39.475         4.225 0.8386849 5.128661
##          SD.exercise  n
## chol       0.9136794 80
## age        0.9136794 80
## exercise   0.9136794 80
```

Using `matrix()`,

```
matrix(c(mean_cholest, sd_cholest,
         n = lengths(cholest[, c("chol", "age", "exercise")])),
       nrow = 3, ncol = 3,
       dimnames = list(names(cholest[, c("chol", "age", "exercise")]),
                       c("mean", "SD", "n")))
```

```
##          mean   SD        n
## chol     8.23   0.8386849 80
## age      39.475 5.128661  80
## exercise 4.225  0.9136794 80
```

3.4.3 paste0()

We can also use the `table()` and `paste0()` as follows,

```
tab_categ = table(Category = cholest$categ)
per_categ = prop.table(tab_categ)*100
cell_categ = paste0(tab_categ, " (", per_categ, "%)")
tab_per_categ = tab_categ  # just to set the dimension of `tab_per_categ`
tab_per_categ[] = cell_categ[]
tab_per_categ
```

```
## Category
##       Grp A       Grp B       Grp C
## 25 (31.25%) 33 (41.25%)  22 (27.5%)
```

In another example for cross-tabulation,

```
tab = table(Category = cholest$categ, Gender = cholest$sex); tab  # count
```

```
##         Gender
## Category female male
##    Grp A      0   25
##    Grp B     18   15
##    Grp C     22    0
```

```
per = prop.table(table(Category = cholest$categ, Gender = cholest$sex))*100
per  # %
```

```
##         Gender
## Category female  male
##    Grp A   0.00 31.25
##    Grp B  22.50 18.75
##    Grp C  27.50  0.00
```

```
cbind(tab, per)
```

```
##       female male female  male
## Grp A      0   25    0.0 31.25
## Grp B     18   15   22.5 18.75
## Grp C     22    0   27.5  0.00
```

```
addmargins(tab)  # marginal counts
```

```
##         Gender
## Category female male Sum
##    Grp A      0   25  25
##    Grp B     18   15  33
##    Grp C     22    0  22
##    Sum       40   40  80
```

```
# nicer view
cell = paste0(tab, " (", per, "%)")
str(tab)
```

```
##  'table' int [1:3, 1:2] 0 18 22 25 15 0
##  - attr(*, "dimnames")=List of 2
##   ..$ Category: chr [1:3] "Grp A" "Grp B" "Grp C"
##   ..$ Gender  : chr [1:2] "female" "male"
```

```
tab1 = tab
tab1[] = cell[]
tab1
```

```
##         Gender
## Category female     male
##    Grp A 0 (0%)     25 (31.25%)
##    Grp B 18 (22.5%) 15 (18.75%)
##    Grp C 22 (27.5%) 0 (0%)
```

```r
ftable(tab1)  # nicer 'flat' view
```

```
##          Gender      female        male
## Category
## Grp A           0 (0%)      25 (31.25%)
## Grp B           18 (22.5%)  15 (18.75%)
## Grp C           22 (27.5%)  0 (0%)
```

3.4.4 cat()

Lastly, `cat()` can be used to write combine relevant outputs in text format.

```r
cat("For cholestrol, the mean was ", round(mean(cholest$chol), 2),
    " (SD = ", round(sd(cholest$chol)), ") in a sample of ",
    length(cholest$chol), " subjects.", sep = "")
```

```
## For cholestrol, the mean was 8.23 (SD = 1) in a sample of 80 subjects.
```

3.5 Summary

In this chapter, we learned about how to handle numerical and categorical variables and obtain the basic and relevant descriptive statistics. We also learned how to combine outputs into custom made tables and texts.

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 4

Visual exploration

In this chapter, we will learn how to explore and understand the data by generating graphs. We will first use the built-in functions to come up with the graphs. Then, we will go through a number of powerful packages to generate visually pleasant graphs to summarize the data.

4.1 Introduction to visualization

Data visualization is essentially "information that has been abstracted in some schematic form, including attributes or variables for the units of information" (Friendly, 2009).

For further reading, you may read these sources:

1. Wikipedia entry on data visualization (https://en.m.wikipedia.org/wiki/Data_visualization).
2. Milestones in the history of thematic cartography, statistical graphics, and data visualization (http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf).

4.1.1 History of data visualization

In his 1983 book *The Visual Display of Quantitative Information* (Tufte, 1983), 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".

4.1.2 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)

4.1.3 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.

4.2 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. `lattice`.
3. `ggplot2`.

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

1. `ggsurvlot()` in `survminer` package to plot survival probability.
2. `sjPlot` package to plot mixed models results.

4.3 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?

4.4 Using the `graphics` package

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

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

```
str(cholest)
```

```
'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     : num  38 35 39 36 31 38 33 36 40 34 ...
 $ exercise: num  6 5 6 5 4 4 5 5 4 6 ...
 $ sex     : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
 $ categ   : Factor w/ 3 levels "Grp A","Grp B",..: 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "datalabel")= chr ""
 - attr(*, "time.stamp")= chr ""
```

```
 - attr(*, "formats")= chr  "%10.0g" "%10.0g" "%10.0g" "%10.0g" ...
 - attr(*, "types")= int  255 255 255 255 255
 - attr(*, "val.labels")= chr  "" "" "" "sex" ...
 - attr(*, "var.labels")= chr  "cholesterol in mmol/L" "age in year"
   "duration of exercise (hours/week)" "" ...
 - attr(*, "version")= int 8
 - attr(*, "label.table")=List of 2
  ..$ sex  : Named int  0 1
  .. ..- attr(*, "names")= chr  "female" "male"
  ..$ categ: Named int  0 1 2
  .. ..- attr(*, "names")= chr  "Grp A" "Grp B" "Grp C"
```

```r
head(cholest); tail(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

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

```r
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
```

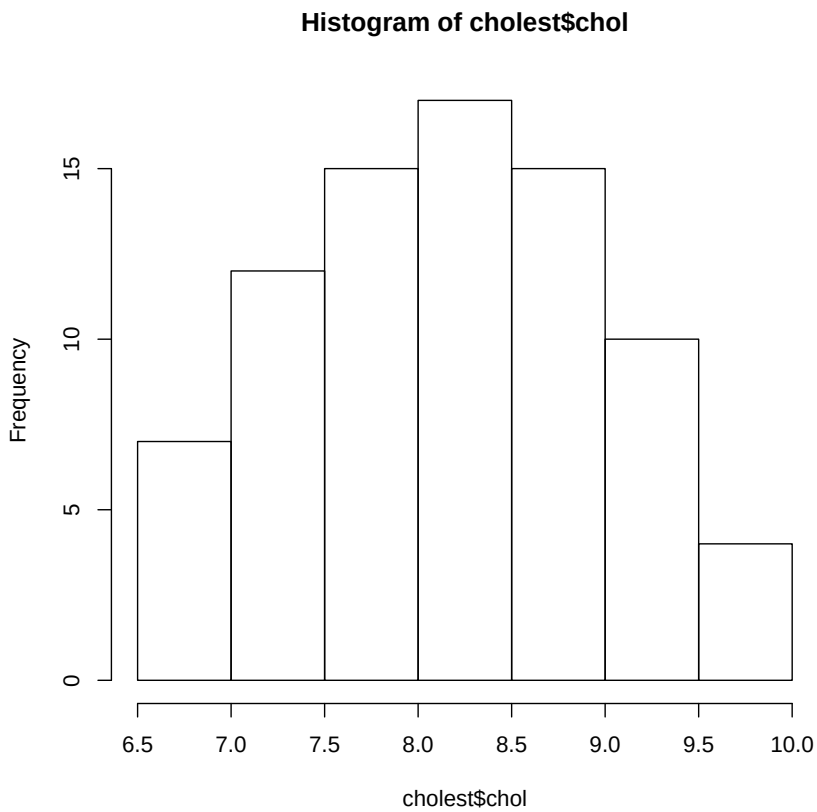
Histogram

We create histograms with `hist(x, breaks, freq)` function. 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.

The basic one can be run with `x` alone, in our case `cholest$chol`,

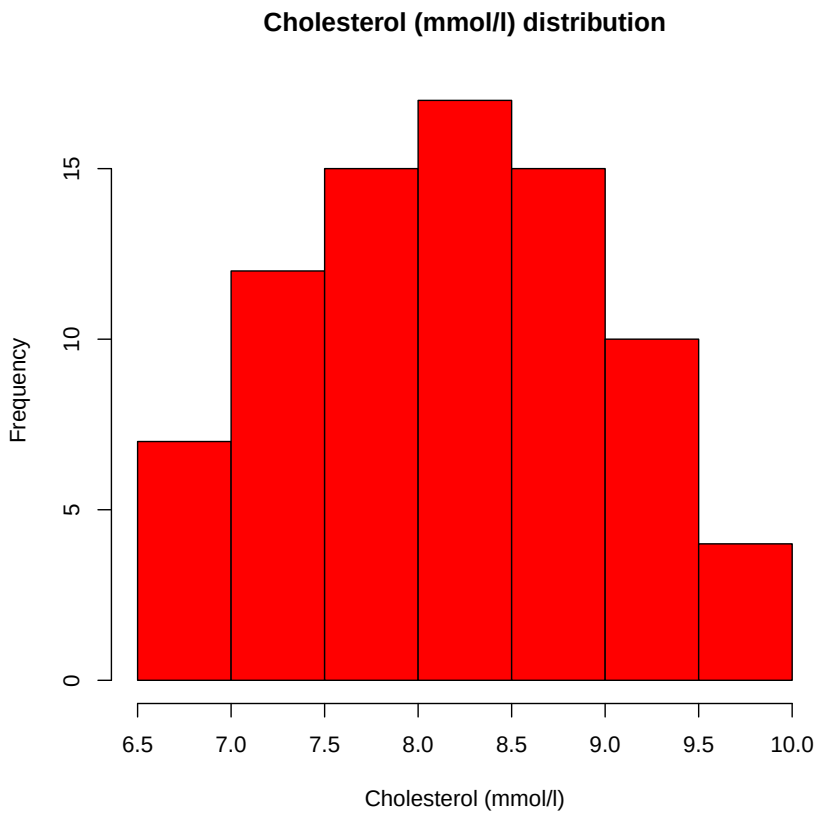
```r
hist(cholest$chol)
```



Now, to refine the `hist()` function, we will:

1. set the color `col =` argument to `red`,
2. set the argument for the number of bins to 8 bins `breaks = 10`,
3. label the x-axis using `xlab = "label"`,
4. the plot title is set by `main = "title of plot"`.

```
hist(cholest$chol, breaks = 10, col = "red",
     main = "Cholesterol (mmol/l) distribution", xlab = "Cholesterol (mmol/l)")
```



Kernel density plot

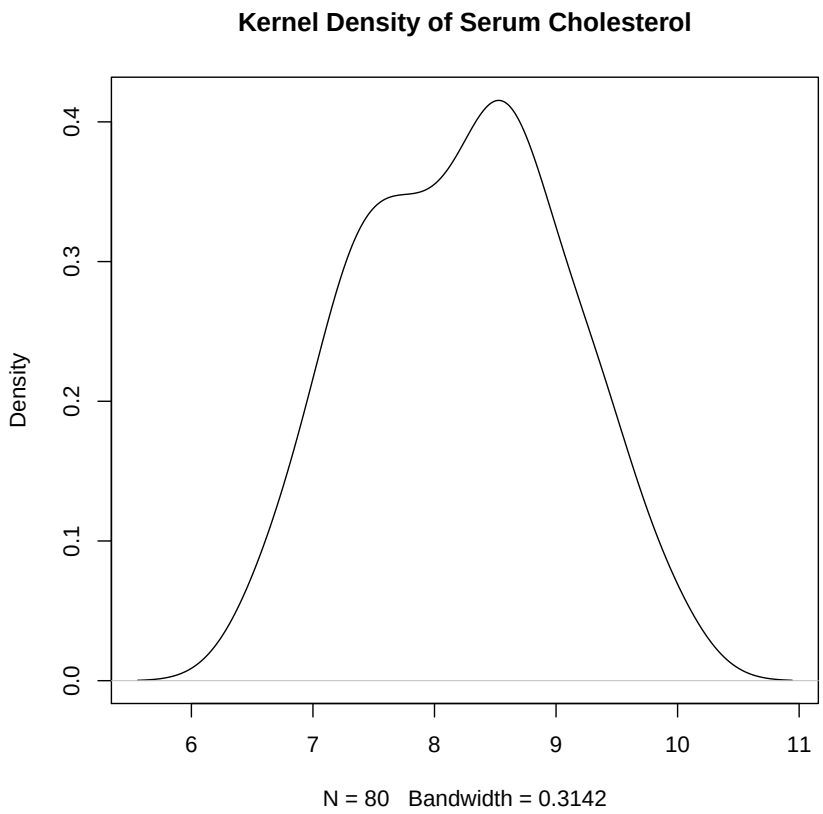
Kernel density plots are usually a much more effective way to view the distribution of a numerical variable.

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

Below, we

1. create the density data and named it as `d.plot`,
2. next, we plot `d.plot`.

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



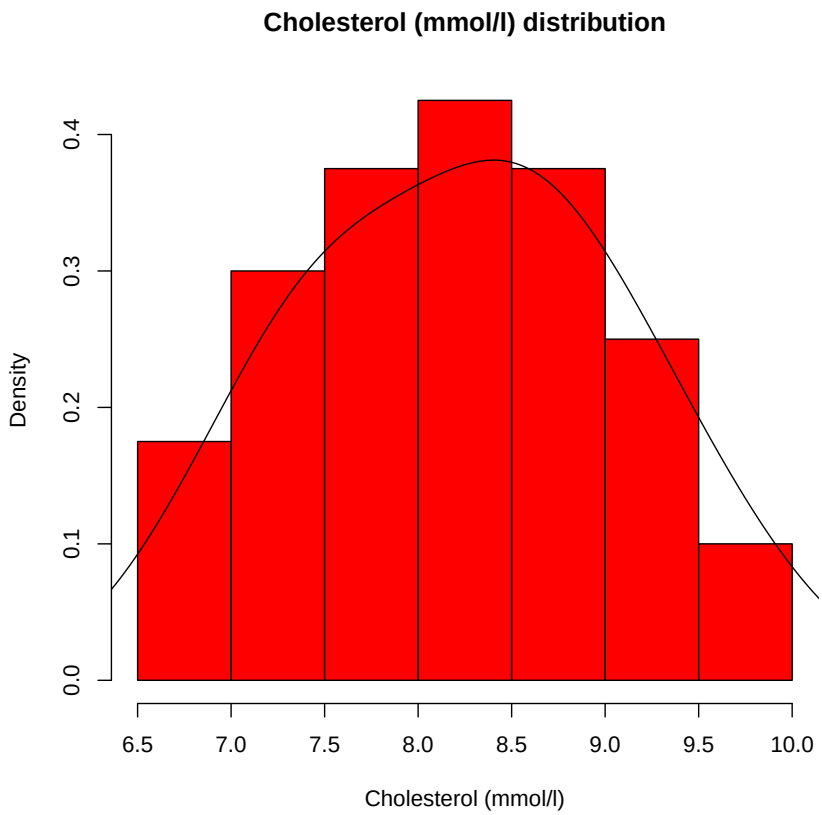
`plot()` is a generic function for X-Y axes plotting. It accepts data frame and density objects and choose suitable plot automatically. You can view the details about `?plot` in the help.

Combining the histogram and density curve

We can combine these plots in one single plot. Here, we will

1. plot the histogram with density (instead of frequency),
2. overlay the density curve on top of the histogram. To do that we need to use `lines()` in place of `plot()`. `plot()` will create a new plot, but `lines()` will overlay line(s) on top of any plot.

```
hist(cholest$chol, breaks = 10, freq = FALSE, col = "red",
     main = "Cholesterol (mmol/l) distribution", xlab = "Cholesterol (mmol/l)")
lines(density(cholest$chol, adjust = 1.5))
```

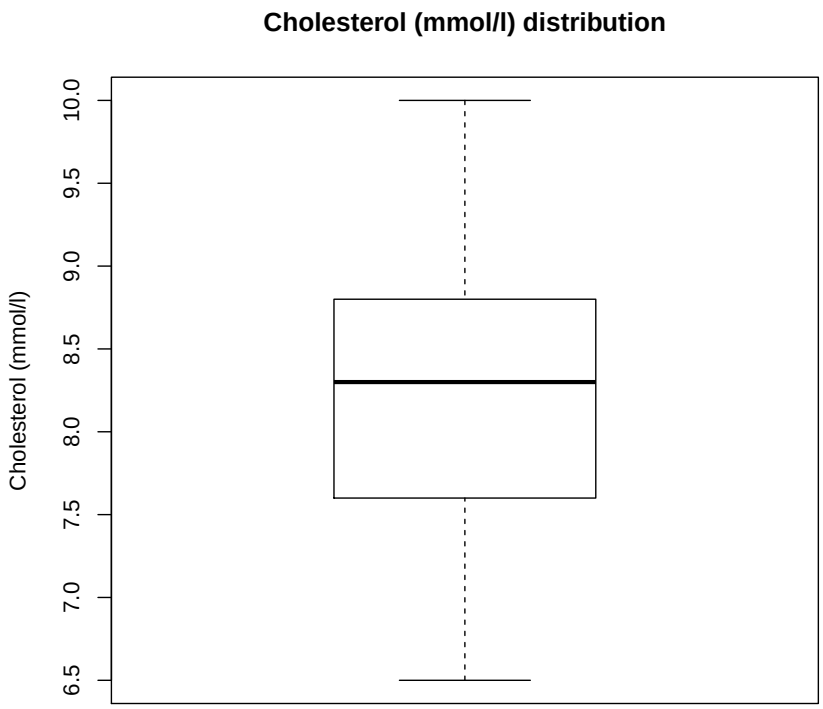


Notice that you can `adjust =` the density bandwidth relative to the default bandwidth. Here we use `adjust = 1.5`.

Box-and-whisker plot

We can easily obtain box-and-whisker plot using `boxplot()`,

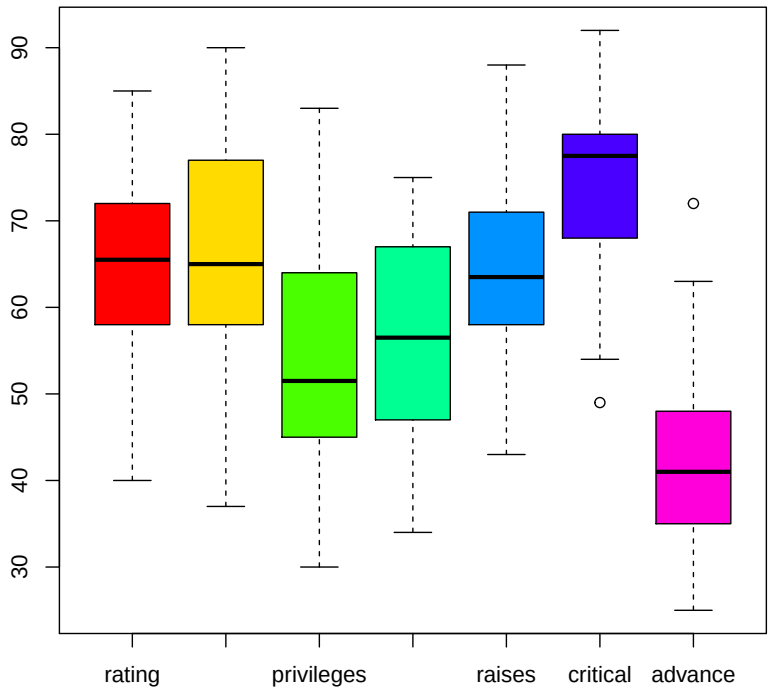
```
boxplot(cholest$chol, main = "Cholesterol (mmol/l) distribution",
        ylab = "Cholesterol (mmol/l)")
```



Here we include `ylab`, which stands for y-axis label.

`boxplot()` can easily handle many variables (of the same scale), for example we use `attitude` dataset,

```
boxplot(attitude, col = rainbow(7))
```



We leave it to you to discover what `rainbow()` does.

4.4.1 Plotting relationship between numerical variables

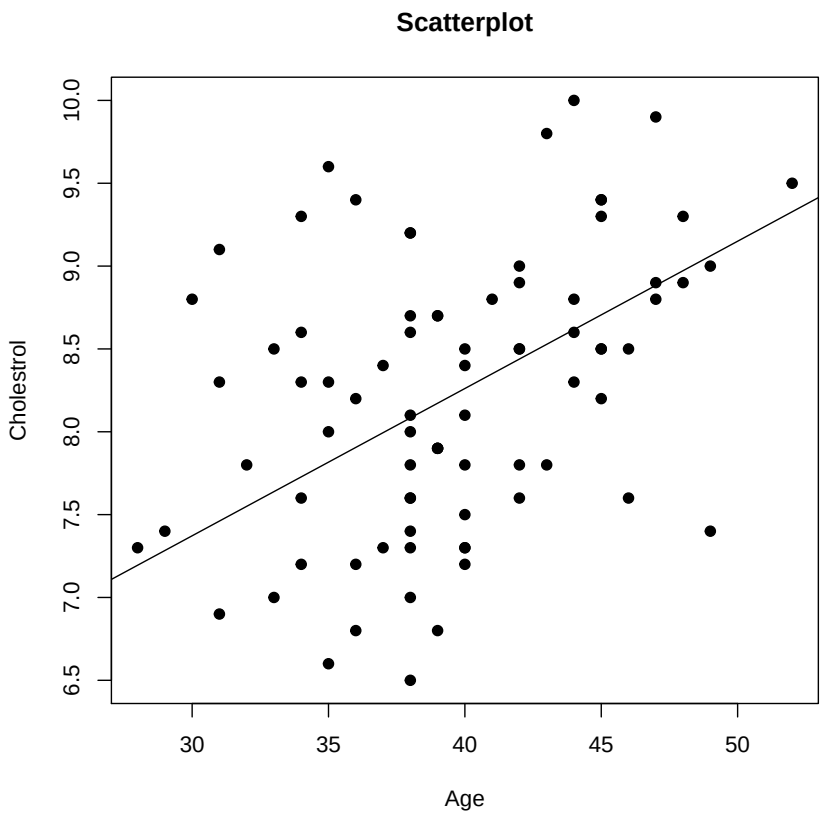
We can plot two numerical variables simultenously. From such plot, we can see the association or relationship between the two variables.

Scatter plot

Scatter plot is one of the most common plots to display the association between 2 numerical variables. The function is basically specified as `plot(x, y)`.

Now we plot `age` on x-axis and `chol` on y-axis,

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



Here we include a new argument, plotting character `pch`. Here we use `19` (see help `?points`). We can included a regression line, by combining `abline()` and `line()`. `abline()` gives you the straight line, while `line()` feeds the data of robust line fitting to `abline()`.

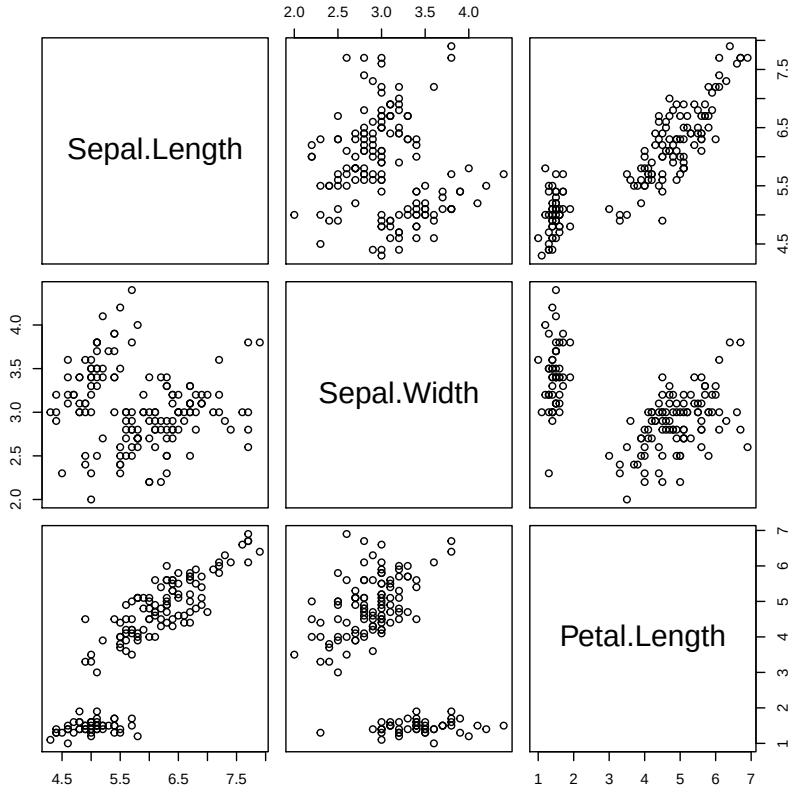
You can always personalize the graphical parameters such as parameters for `fonts, colours, lines` and `symbols`. You can find the details in the `graphics` package documentation and help `?par`. In addition, this website summarizes the parameters in a very nice way: http://www.statmethods.net/advgraphs/parameters.html

We can also plot a number of scatter plots simultaneously to explore the relationship between several numerical variables, for example using `iris` data set,

```
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 ...
```

```
plot(iris[1:3])
```

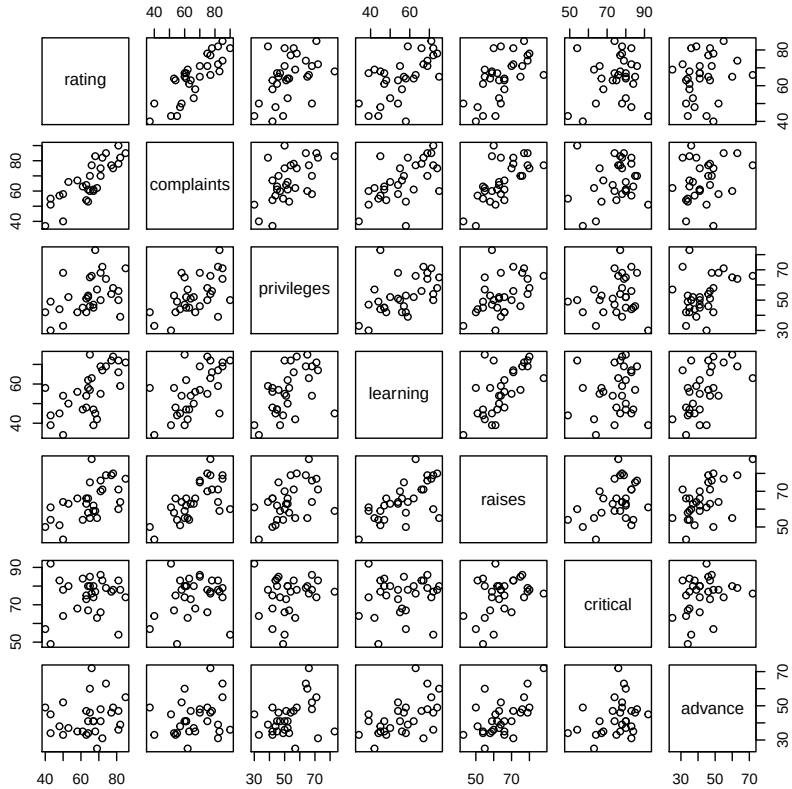


and `attitude` data set,

```
str(attitude)
```

```
## 'data.frame':    30 obs. of  7 variables:
##  $ rating    : num  43 63 71 61 81 43 58 71 72 67 ...
##  $ complaints: num  51 64 70 63 78 55 67 75 82 61 ...
##  $ privileges: num  30 51 68 45 56 49 42 50 72 45 ...
##  $ learning  : num  39 54 69 47 66 44 56 55 67 47 ...
##  $ raises    : num  61 63 76 54 71 54 66 70 71 62 ...
##  $ critical  : num  92 73 86 84 83 49 68 66 83 80 ...
##  $ advance   : num  45 47 48 35 47 34 35 41 31 41 ...
```

```
plot(attitude)
```



4.4.2 Categorical variables

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

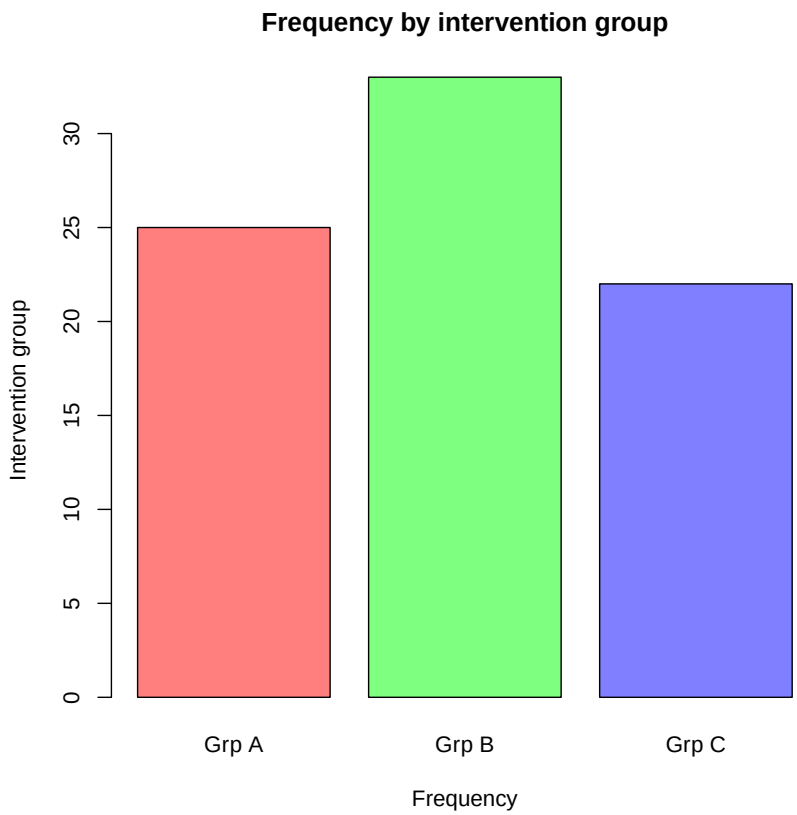
Create a frequency table of intervention groups `categ` and name it as `counts`:

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

```
##
## Grp A Grp B Grp C
##    25    33    22
```

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

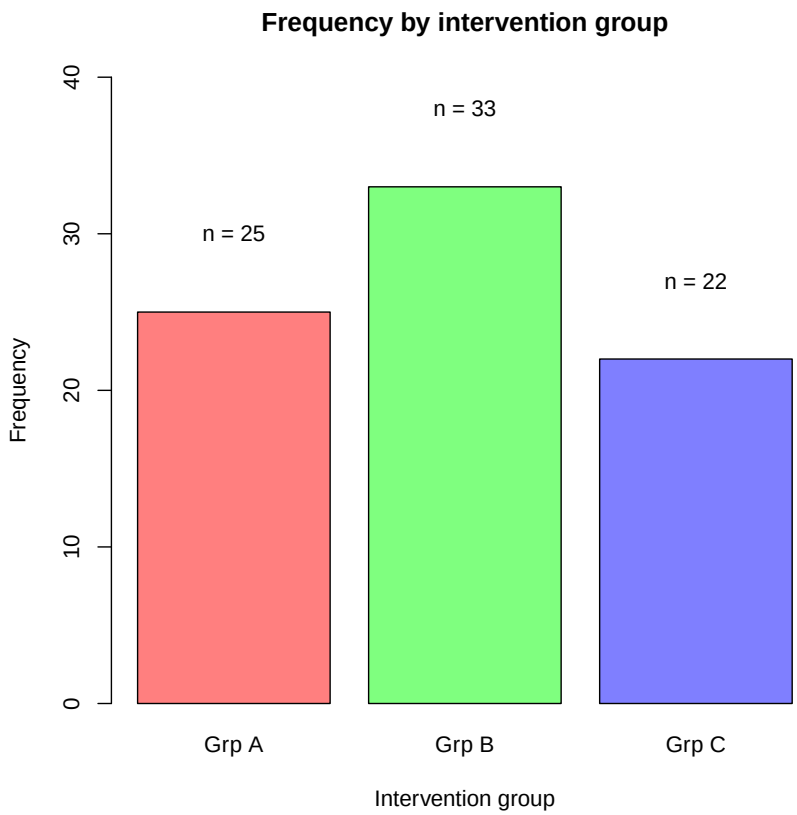
```
barplot(counts, main="Frequency by intervention group",
        ylab = "Intervention group", xlab = "Frequency",
        col = rainbow(3, alpha = 0.5))
```



Here we give `rainbow()` a bit twist by adding `alpha = 0.5` argument and value.

We can make the plot look nicer by adding sample sizes to the bars,

```
barplot(counts, main="Frequency by intervention group",
         xlab = "Intervention group", ylab = "Frequency",
         col = rainbow(3, alpha = 0.5), ylim = c(0, 40)) -> bplot_setting
text(bplot_setting, counts + 5, paste0("n = ", counts))
```



Notice `->` assignment sign, which is just the reverse of `<-` sign we are used to. We can also write the object name on the right hand side of the assignment arrangement. Here we intentionally do so to emphasize the `barplot()` codes. `bplot_setting` gives `text()` the x coordinates, and `counts + 5` gives it the y coordinates.

To make things more complicated (a.k.a more interesting in R), we plot a stacked barchart. We need age categories to demonstrate a nice looking stacked bacrchart, and we create `age_cat`,

```
# group `age` into `age_cat` = `< 35`, `35-45`, `> 45`
cholest$age_cat <- cut(cholest$age,
                       breaks = c(-Inf, 35, 45, Inf),
                       labels = c("<35", "35-45", ">45"))
```

Then we obtain the cross-tabulated counts between `sex` and `age_cat`,

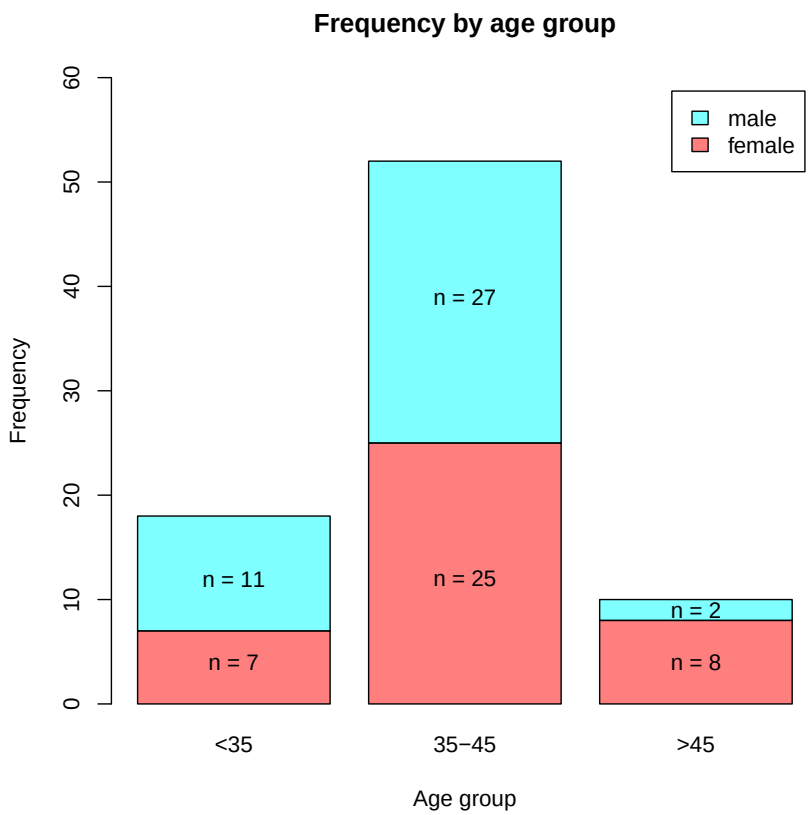
```
cross <- table(cholest$sex, cholest$age_cat)
addmargins(cross)  # just to get an overview of the height of the bars
```

```
##
##          <35 35-45 >45 Sum
##   female   7    25   8  40
##   male    11    27   2  40
##   Sum     18    52  10  80
```

Plot our nice stacked barchart,

```
barplot(cross, main = "Frequency by age group",
        xlab = "Age group", ylab = "Frequency",
        col = rainbow(2, alpha = 0.5), ylim = c(0, 60),
        legend = rownames(cross)) -> bplot_setting
```

```
text(rep(bplot_setting, each = 2), c(4, 12, 12, 39, 4, 9),
     paste0("n = ", cross))  # adjust y coordinates to your liking
```



Note how we use `rep()` to repeat x coordinates twice for each age category.

4.4.3 Saving plots in R

We can save the generated plots. In RStudio, under **Plots** tab, yiou can click on the **Export** button to save the plots as image of PDF. Alternatively, we can automaticaly save the plots (without viewing the plots). The examples below will save the plot as image and PDF formats.

Here we save as an image `.png`,

```
png(file = "hist.png")
hist(cholest$chol)
dev.off()
```

```
## pdf
##   2
```

`png()` opens creates an empty file namely `airmiles.png`, while `dev.off()` closes the file and save whatever plot you have in between these two lines (limited to the last one if you specified several plots).

You can also specify the `width =` and `height =` of the image. View the help for `?png`. The help also lists functions for the rest of image formats such as `bmp, jpeg` and `tiff`.

Now, we can also save as PDF,

```
pdf("plots.pdf")
hist(cholest$chol, freq = FALSE)
lines(density(cholest$chol))
barplot(table(cholest$sex))
plot(cholest$chol, cholest$age)
dev.off()
```

```
## pdf
##   2
```

The advantage of saving as PDF is because we can save many plots in a single PDF file as demonstrated here. The quality of the saved plots is also very good as compared to saving as images.

4.5 Using the `lattice` package

`lattice` package can create beautiful plots too. Its main emphasis is on the visualization of multivariate data, thus it is very useful for plotting multiple plots, for example histograms of questionnaire items. It is also easy to visualize the data by groups in `lattice` as we will show in examples below. A very useful introduction to `lattice` package by the package developer can be found here http://lattice.r-forge.r-project.org/Vignettes/src/lattice-intro/lattice-intro.pdf.

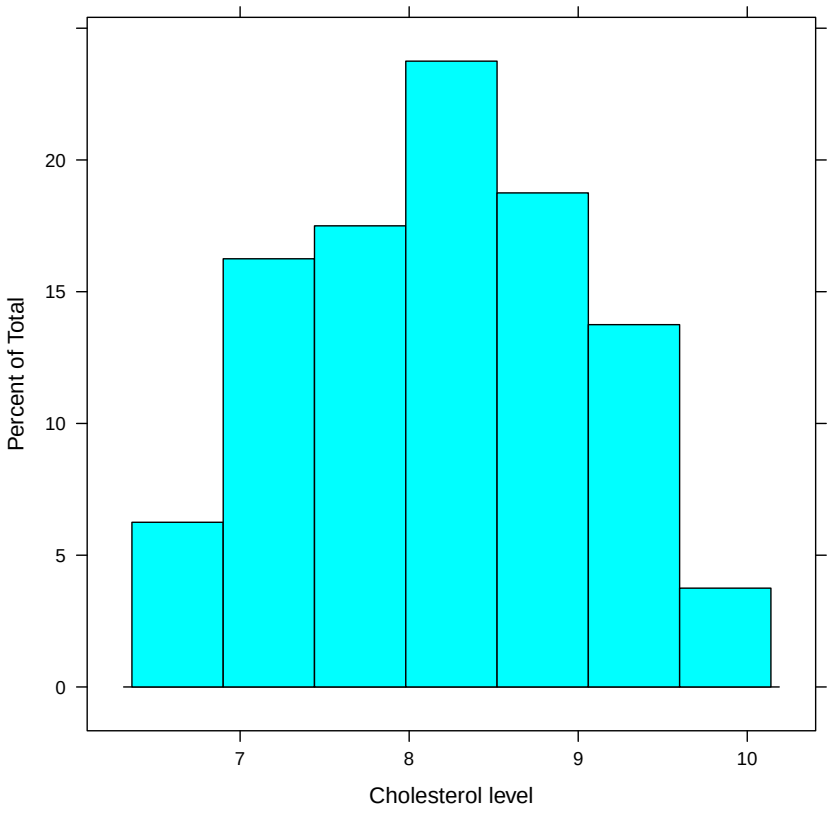
4.5.1 Histogram, density and box-and-whisker plots

Load the `lattice` package,

```
library(lattice)
```

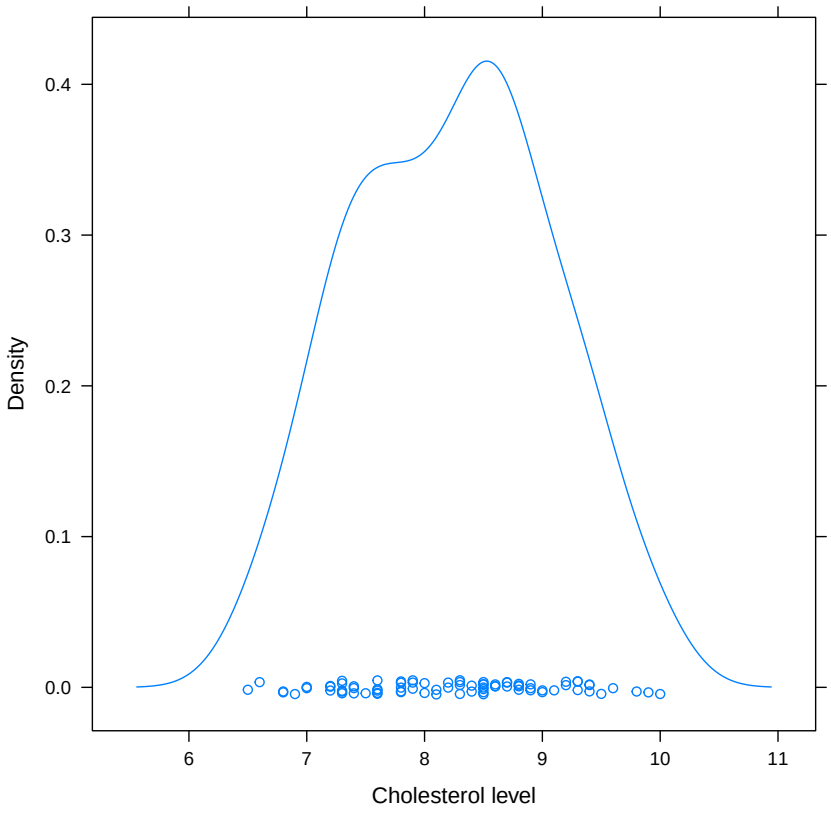
Plot a histogram for variable `chol` and label the x-axis

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



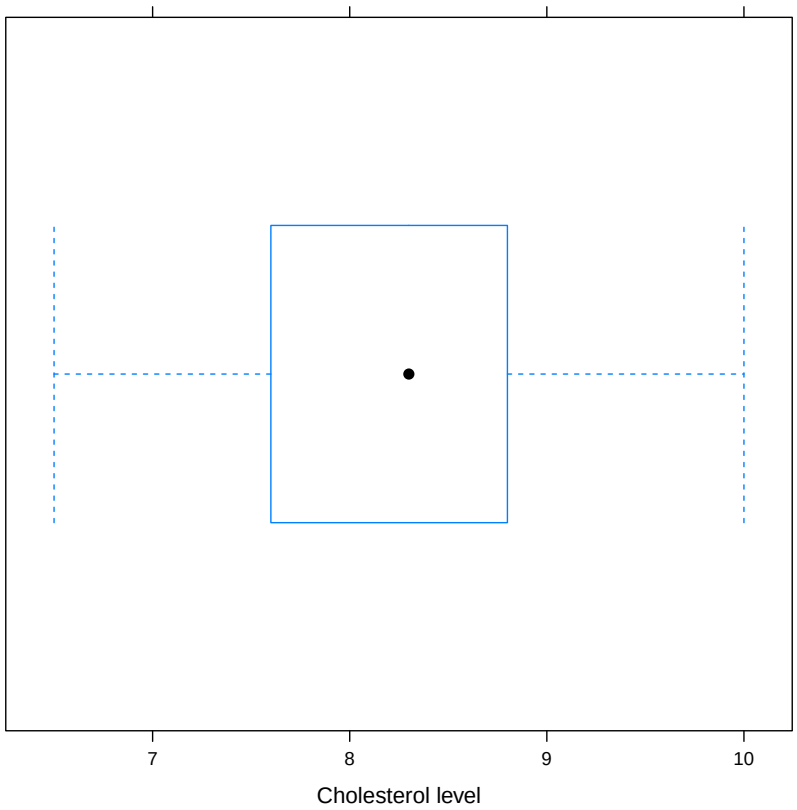
Now we plot a density plot for variable `chol`,

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



followed by a box-and-whisker plot for the variable,

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

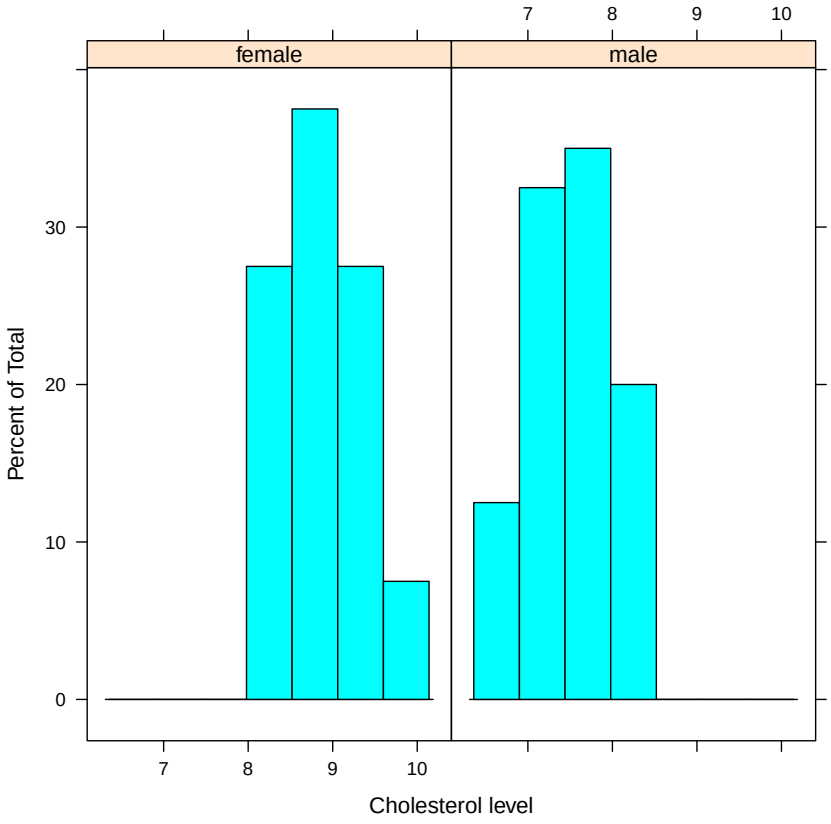


4.5.2 Histogram, density and box-and-whisker plots by group

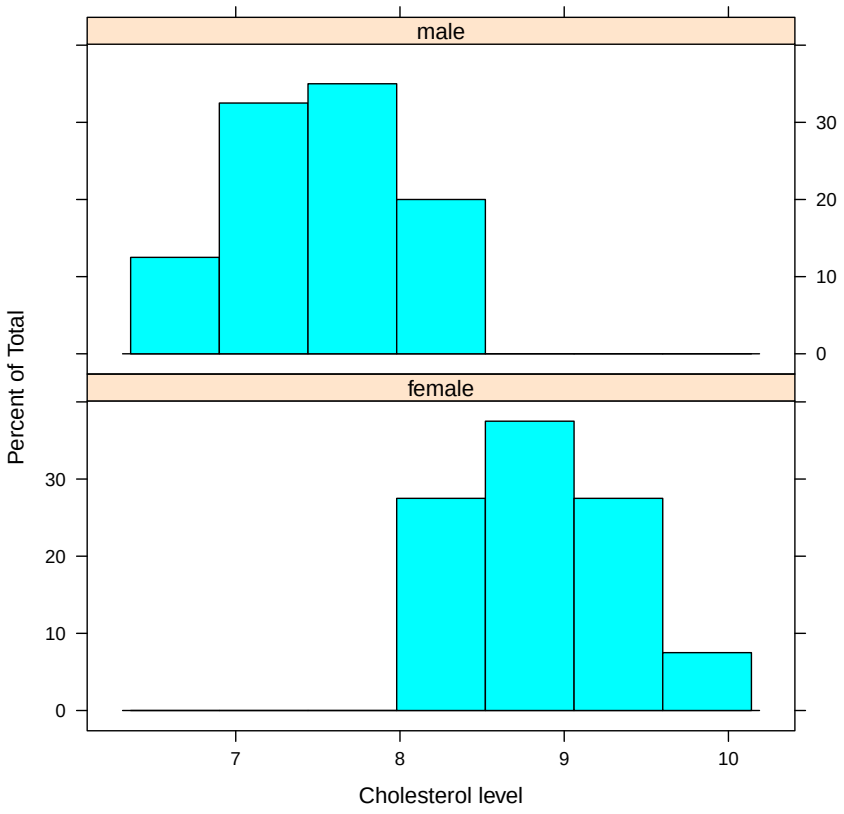
It is easy to plot these plots by group in `lattice`, making it a quick data visualization package.

Histograms,

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



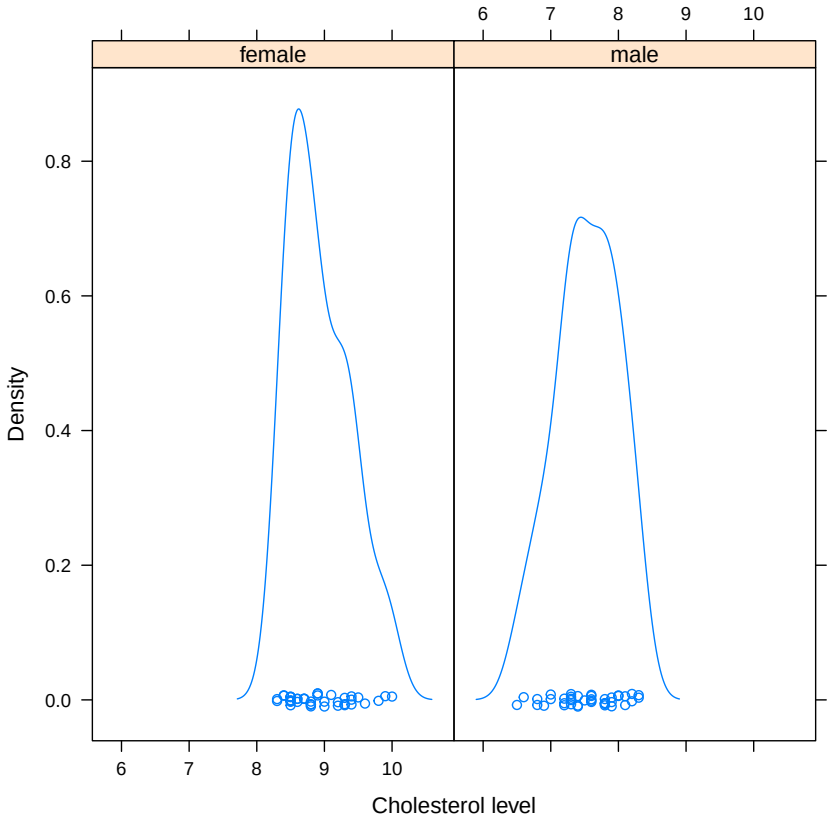

```
histogram(~ chol | sex, data = cholest, layout = c(1, 2), xlab = 'Cholesterol level')
```



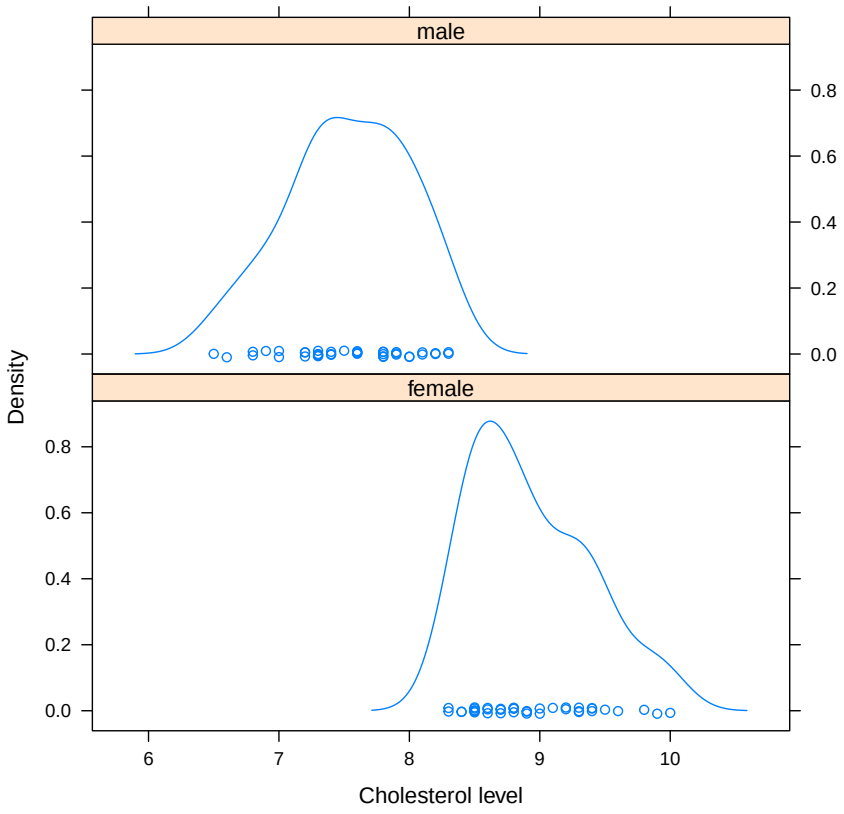
Here, we use `layout = c(1, 2)` for `lattice` argument. This means "1" column (over the x-axis) and "2" rows (along the y-axis).

Density plots,

```
densityplot(~ chol | sex, data = cholest, xlab = 'Cholesterol level')
```

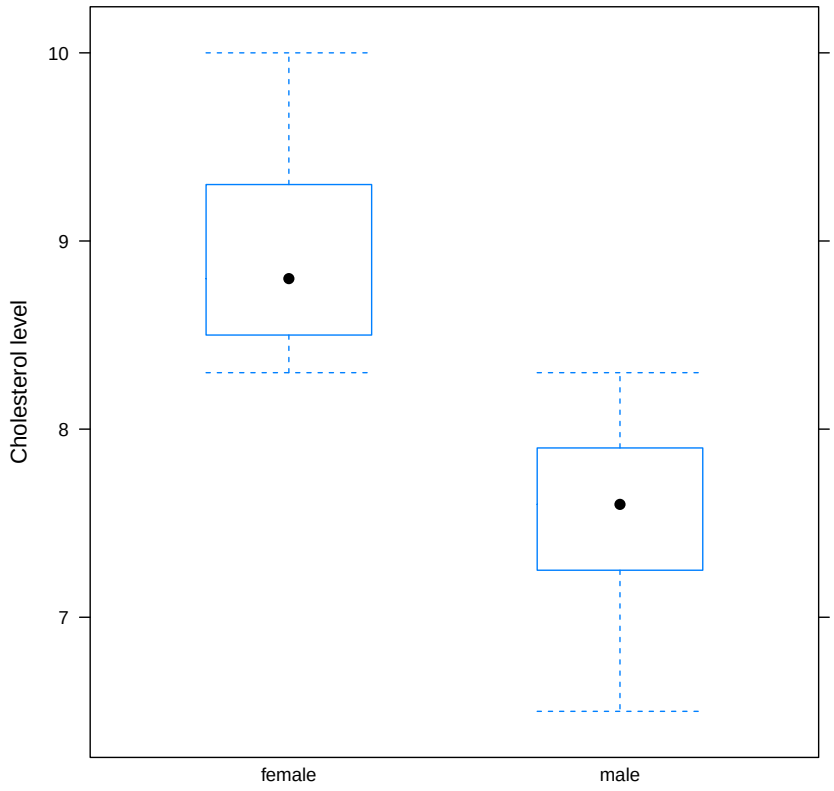



```
densityplot(~ chol | sex, data = cholest, layout = c(1, 2), xlab = 'Cholesterol level')
```

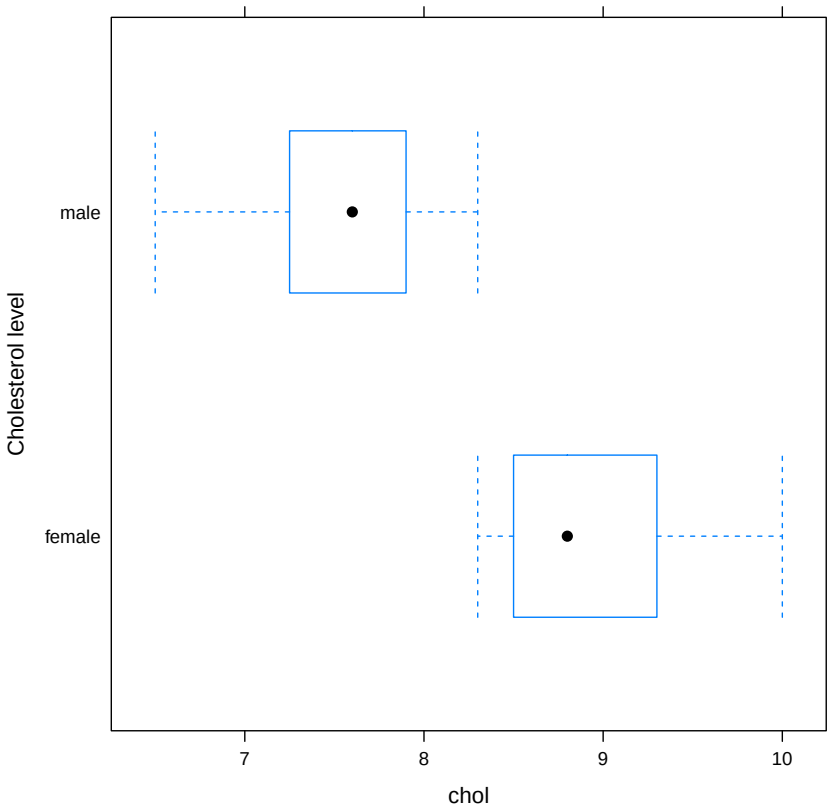


Boxplots,

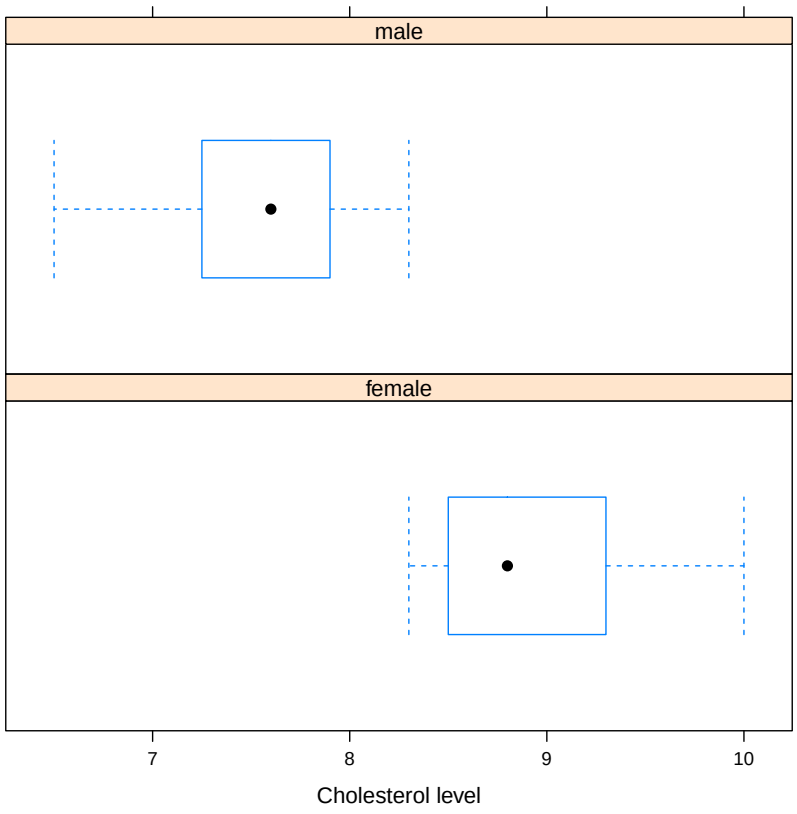
```
bwplot(chol ~ sex, data = cholest, ylab = 'Cholesterol level')
```




```
bwplot(sex ~ chol, data = cholest, ylab = 'Cholesterol level')  # note the change in x-y axis.
```

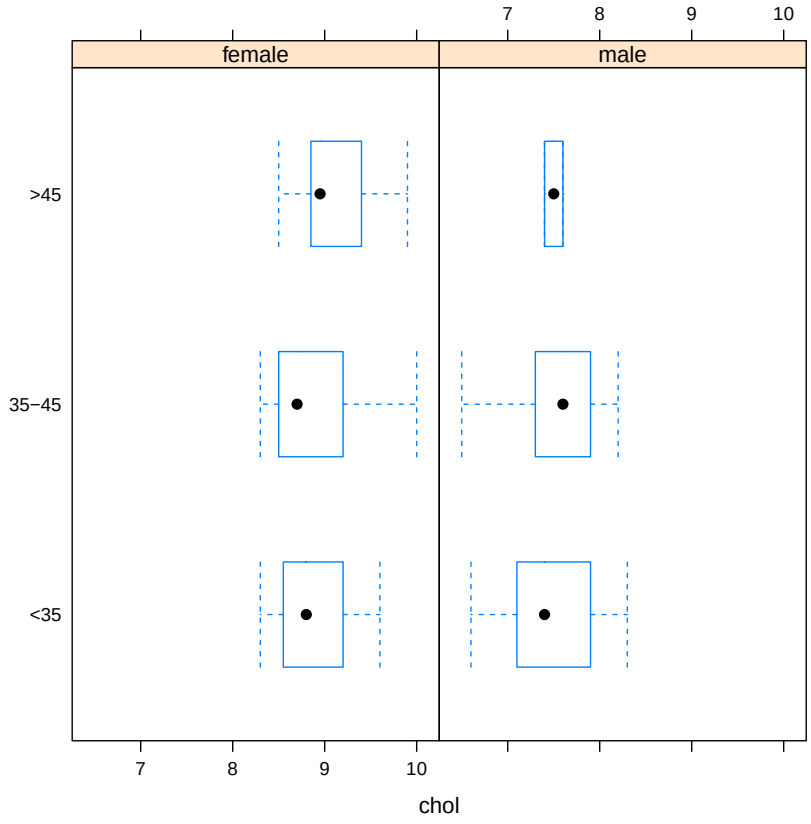



```
bwplot(~ chol | sex, data = cholest, layout = c(1, 2), xlab = 'Cholesterol level')
```



Then we add an extra grouping layer (`age_cat`) to the boxplots. Remember that we created `age_cat` in the previous section and added it to `cholest` data frame.

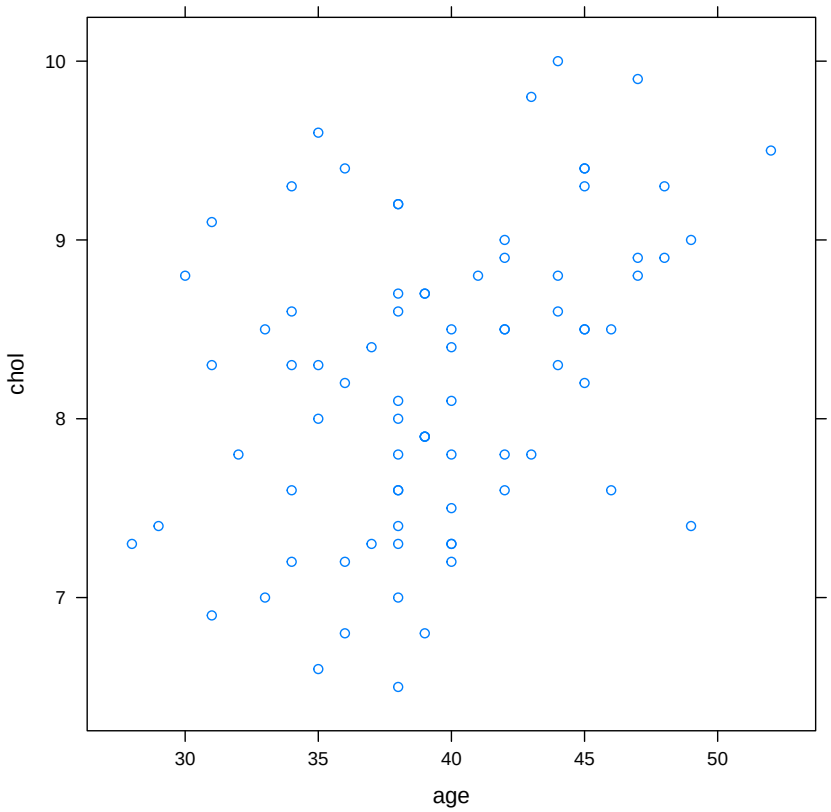
```
bwplot(age_cat ~ chol | sex, data = cholest, layout = c(2, 1))
```



4.5.3 Scatter plot

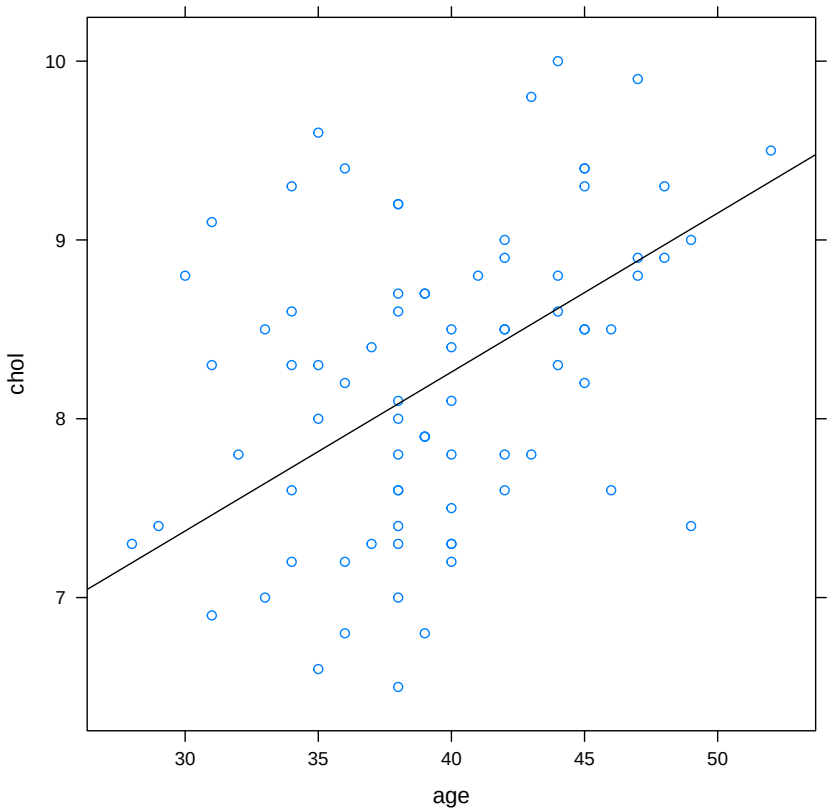
We can also plot scatter plot easily in `lattice`,

```
xyplot(chol ~ age, data = cholest)
```



However, to add the line is a bit tricky as shown below,

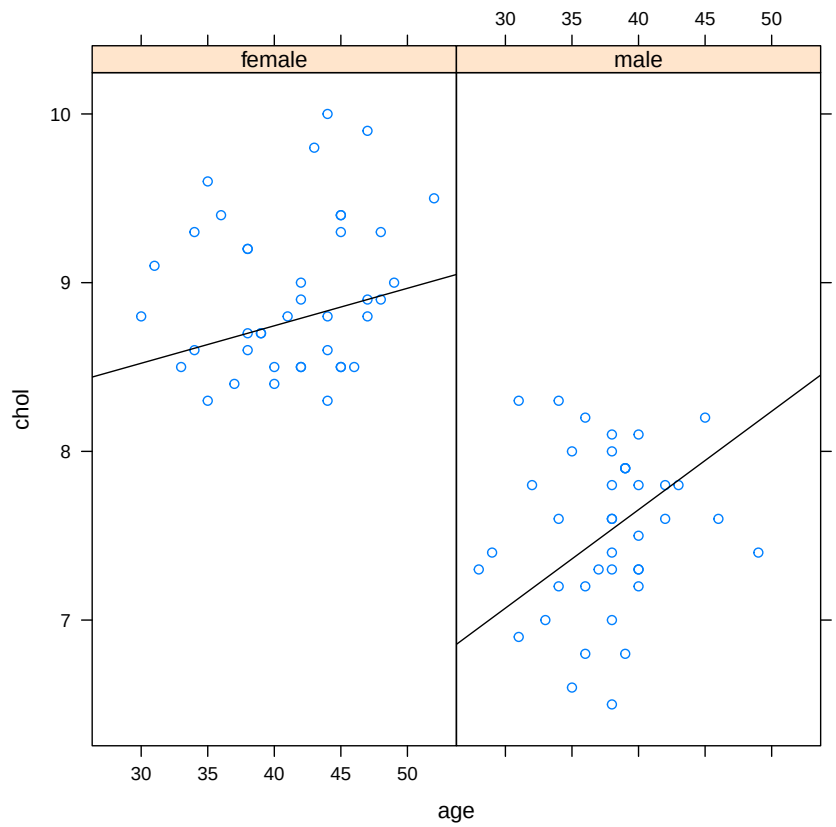
```
xyplot(chol ~ age, data = cholest,
       panel = function(x, y) {
         panel.xyplot(x, y)
         panel.abline(line(x, y))
         })
```



We find it easier to do this by `graphics` package.

Despite this slight "trickiness", it is relatively easily to obtain scatter plots by group,

```
xyplot(chol ~ age | sex, data = cholest,
       panel = function(x, y) {
         panel.xyplot(x, y)
         panel.abline(line(x, y))
       })
```



4.5.4 Barchart

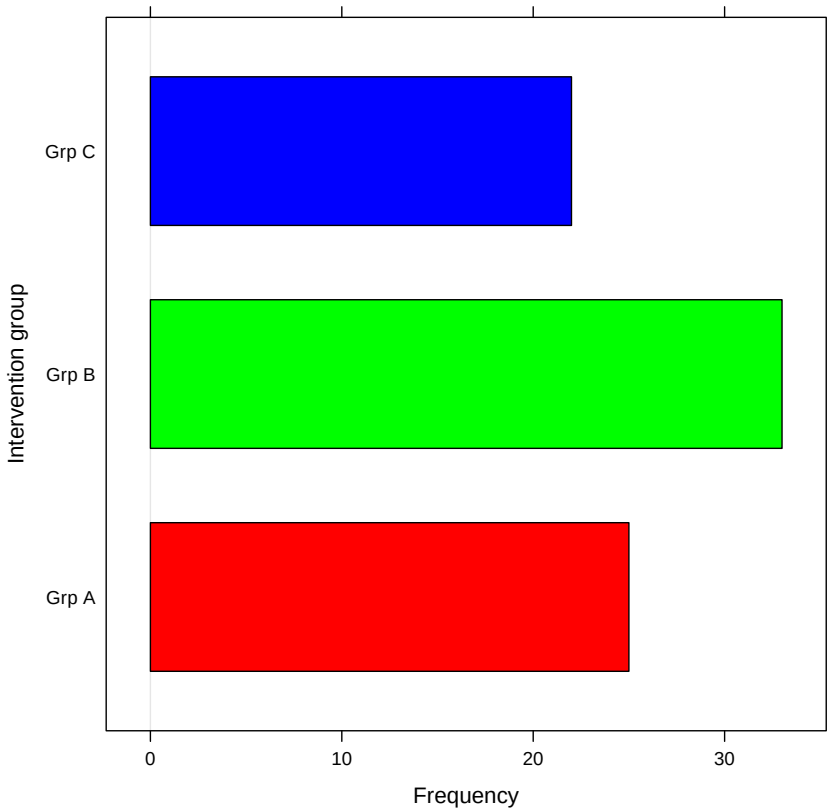
For categorical variables, we can easily plot barcharts in `lattice`. We generate the count per group for the categorical variable, for example `categ`:

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

```
##
## Grp A Grp B Grp C
##    25    33    22
```

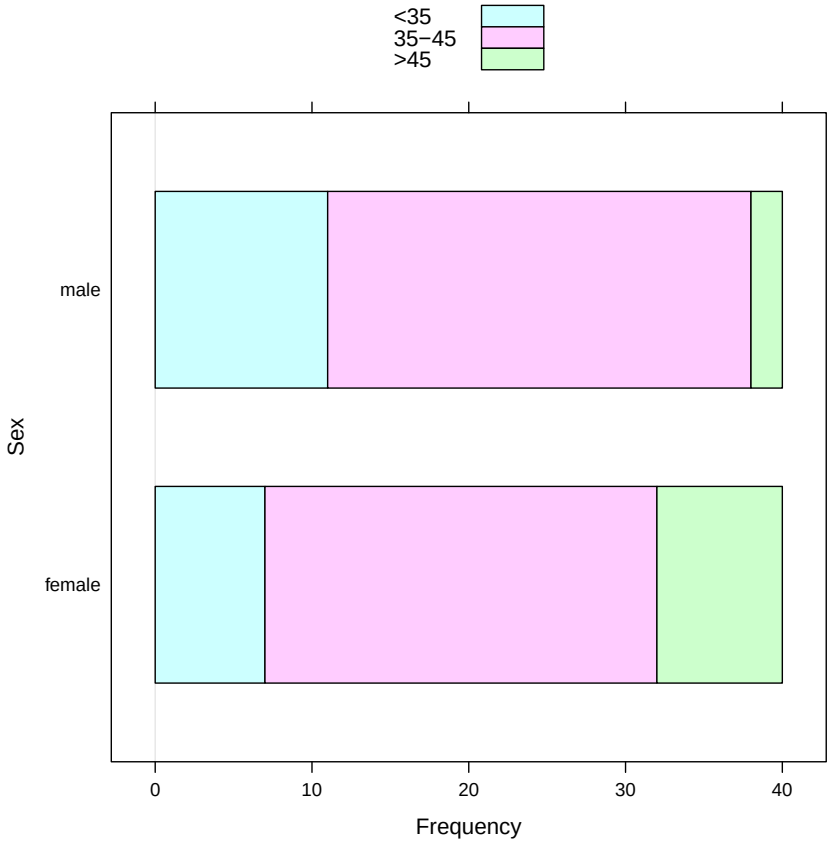
Now, plot the frequencies for the `counts` object,

```
barchart(counts, ylab = "Intervention group", xlab = "Frequency",
         col = rainbow(3))
```

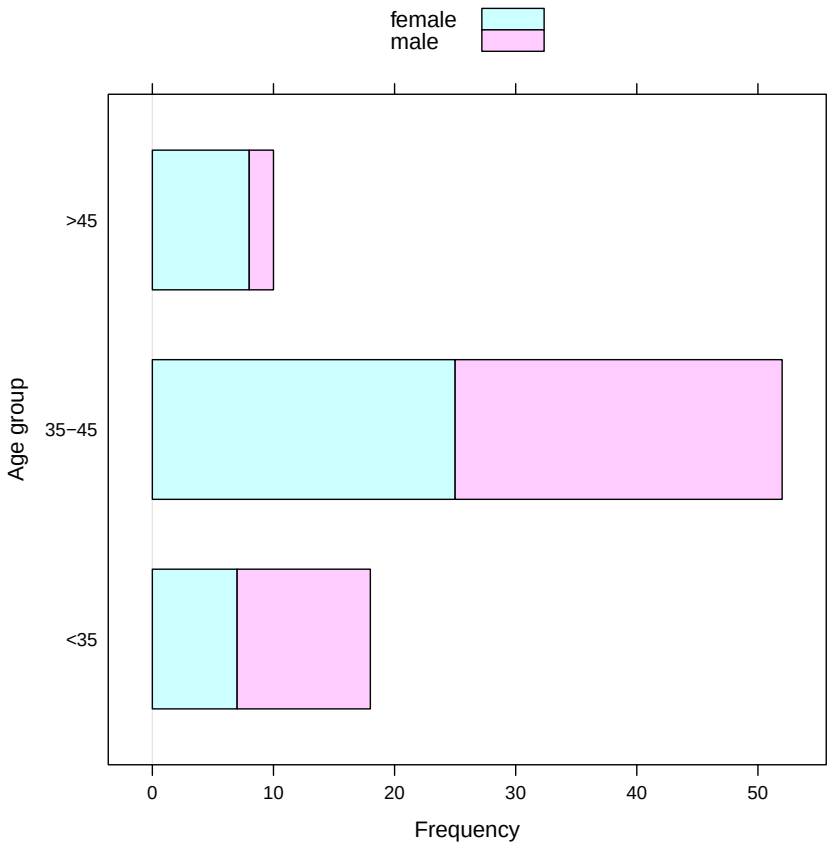


We can also have stacked barchart based on counts from cross-tabulation of `sex` and `age_cat`,

```
cross <- table(cholest$sex, cholest$age_cat)
barchart(cross, auto.key = T, ylab = "Sex", xlab = "Frequency")
```




```
barchart(t(cross), auto.key = T, ylab = "Age group", xlab = "Frequency")
```



`auto.key` automatically gives us the legend. This is a special feature of `lattice`. With `auto.key` it is better to leave the color choice to the function. We also use `t()` transpose function. Because `barchart()` treat counts from `cross` object by row instead of by column, we need to transpose the arrangement of the row and column to replicate `barplot()` behavior (i.e. in the previous section).

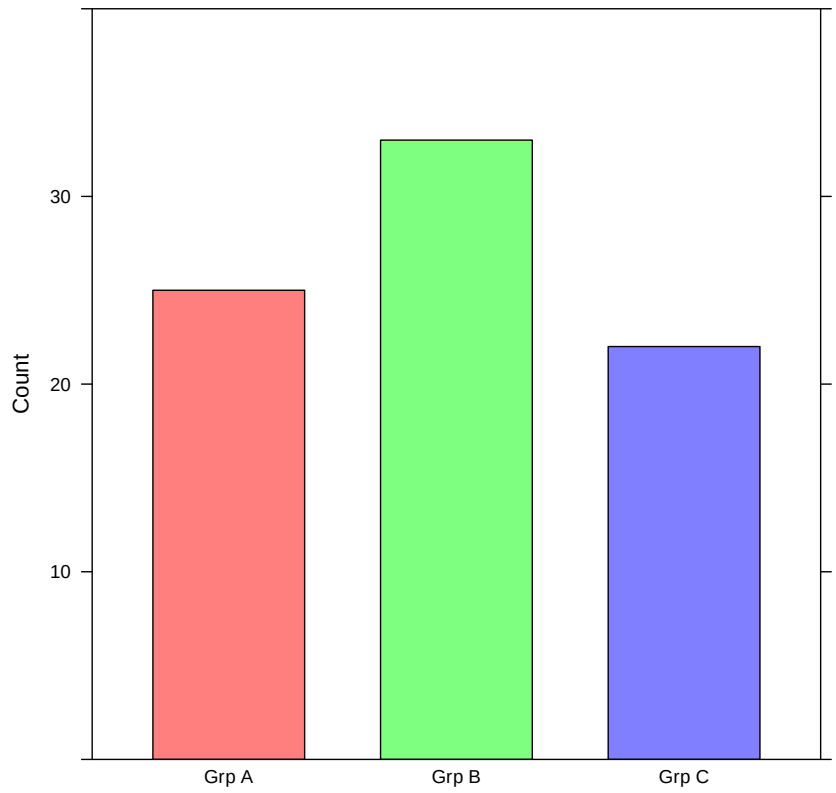
For a more flexible chart setting, convert the count table to a data frame,

```
counts_df <- as.data.frame(counts)
colnames(counts_df) <- c("Category", "Count")  # set the column names
counts_df
```

```
##   Category Count
## 1    Grp A    25
## 2    Grp B    33
## 3    Grp C    22
```

Then, we can plot with `Category` as x-axis and `Count` on y-axis,

```
barchart(Count ~ Category, data = counts_df,
         col = rainbow(3, alpha = 0.5), ylim = c(0, 40))
```



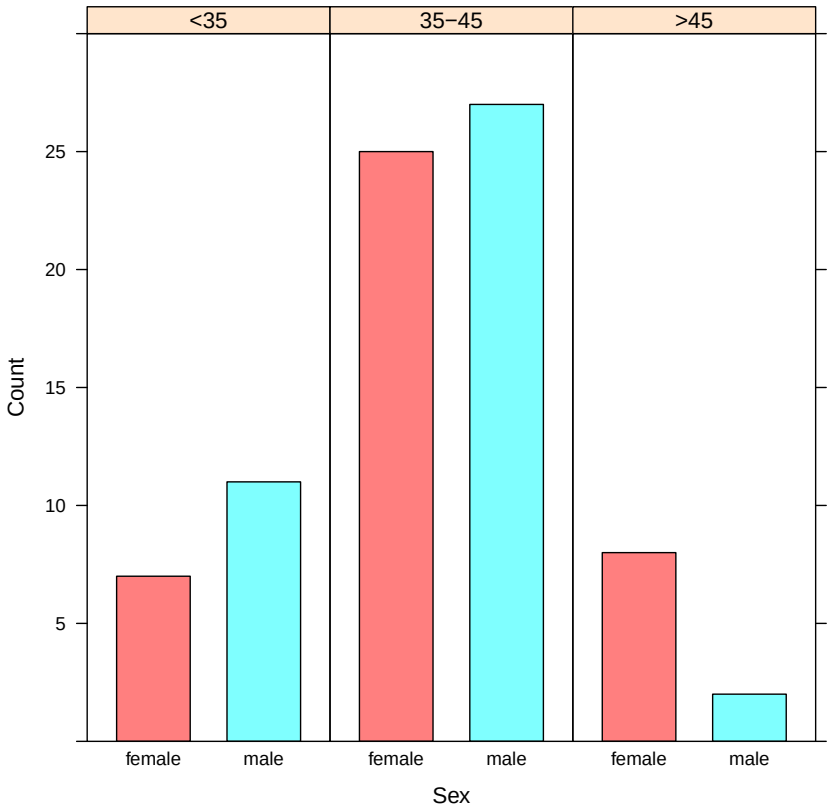
We can also plot barcharts by group using our cross-tabulated counts, `cross` object. Convert the table format into a data frame,

```
cross_df <- as.data.frame(cross)  # save as data frame
colnames(cross_df) <- c("Sex", "Age_Group", "Count")  # give proper names
cross_df
```

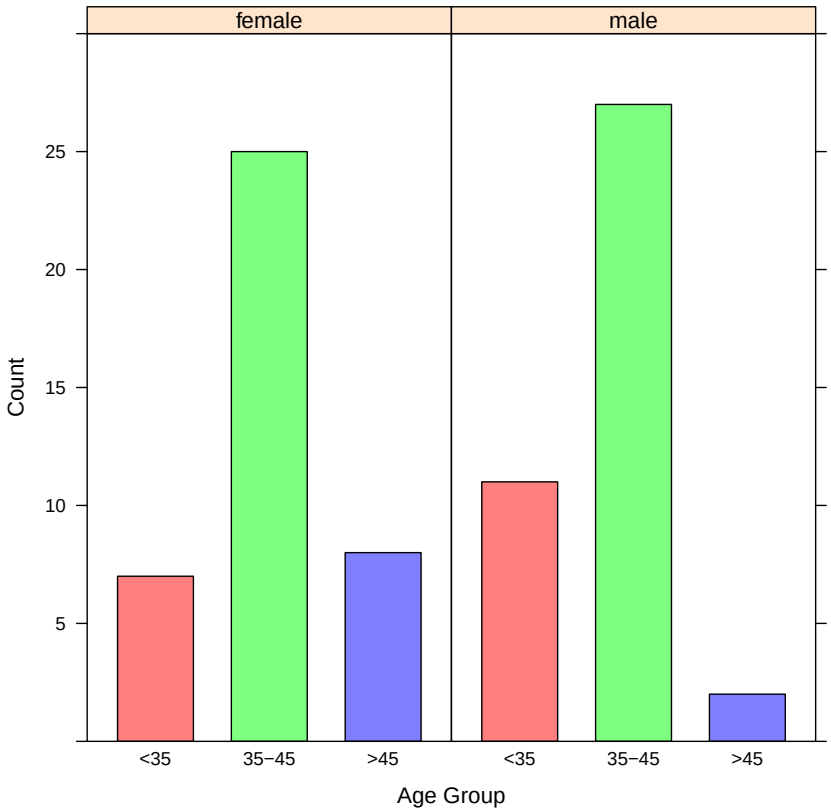
```
##      Sex Age_Group Count
## 1 female       <35     7
## 2   male       <35    11
## 3 female     35-45    25
## 4   male     35-45    27
## 5 female       >45     8
## 6   male       >45     2
```

Then, plot the barcharts,

```
barchart(Count ~ Sex | Age_Group, data = cross_df,
         ylim = c(0, 30), col = rainbow(2, alpha = 0.5),
         xlab = "Sex", layout = c(3, 1))
```




```
barchart(Count ~ Age_Group | Sex, data = cross_df,
         ylim = c(0, 30), col = rainbow(3, alpha = 0.5),
         xlab = "Age Group", layout = c(2, 1))
```



4.5.5 Histograms and box-and-whisker plots

The beauty of `lattice` is in the visualization of multivariate data. We use `attitude` questionaire data set to demonstrate this point.

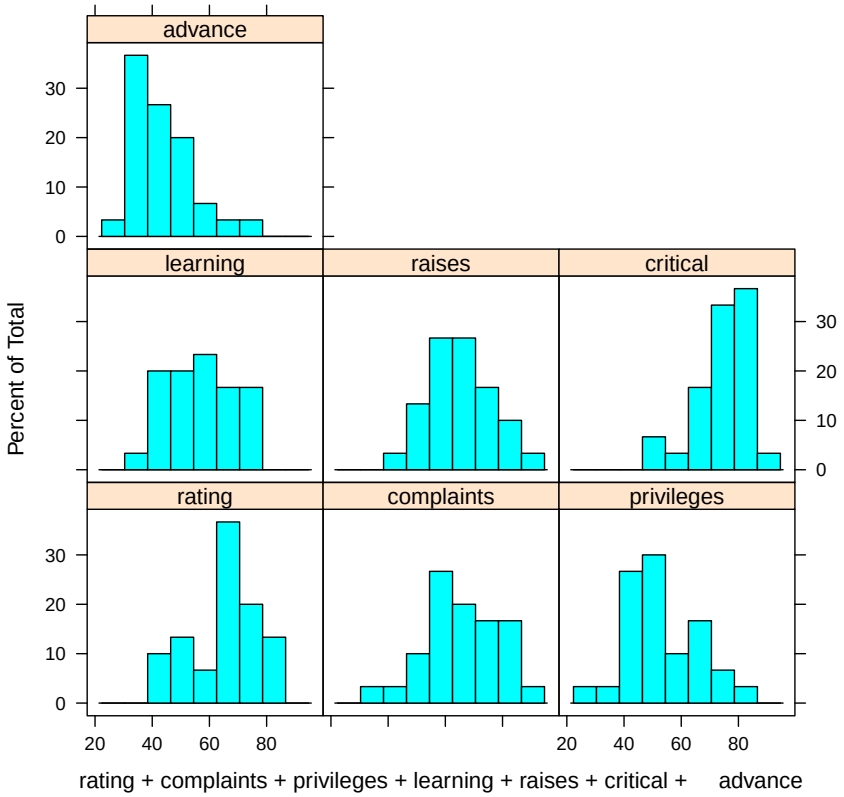
Obtain the list of variables, separated by " + ". This is easily done by `cat()`,

```
cat(names(attitude), sep = " + ")
```

```
## rating + complaints + privileges + learning + raises + critical + advance
```

Plot histograms. Copy-paste from our output of `cat` just now,

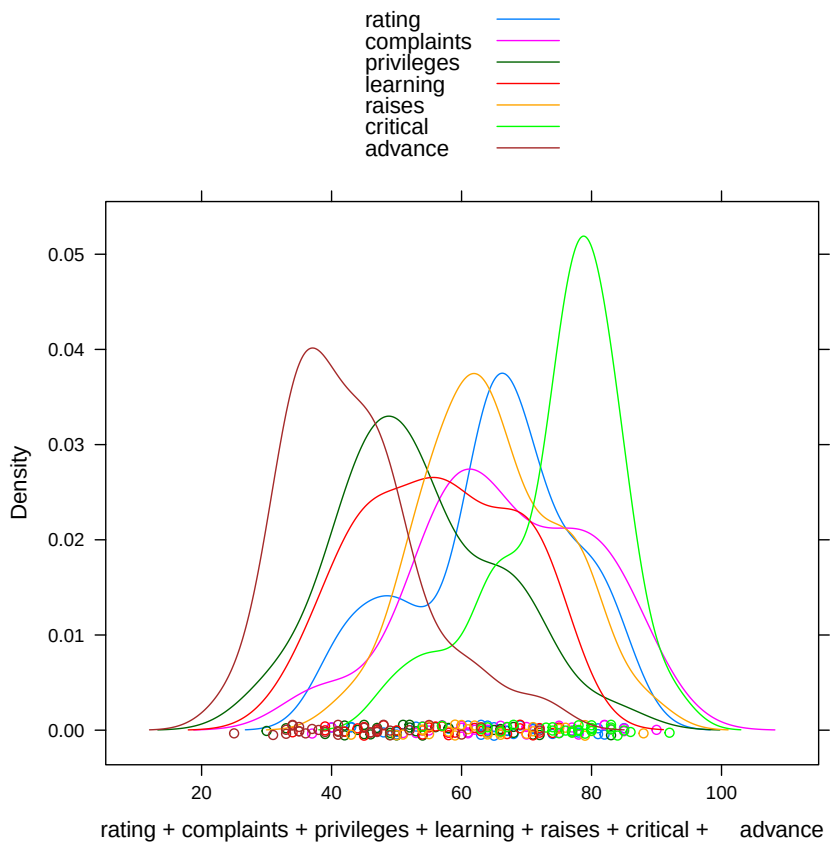
```
histogram(~ rating + complaints + privileges + learning + raises + critical
          + advance, data = attitude)
```



Multiple histogram using `lattice` is meant for numerical variables with same scales, in our case here, percentages.

Plot density plots,

```
densityplot(~ rating + complaints + privileges + learning + raises + critical
            + advance, data = attitude, auto.key = T)
```



However, it is not practical to obtain box-and-whisker plots for variables using `bwplot()`, because `lattice` requires outcome and group variables to plot (i.e. as `bwplot(group ~ numerical)`)

4.6 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.*

4.6.1 One variable: Plotting a numerical variable

Plot distribution of values of a numerical variable.

Histogram

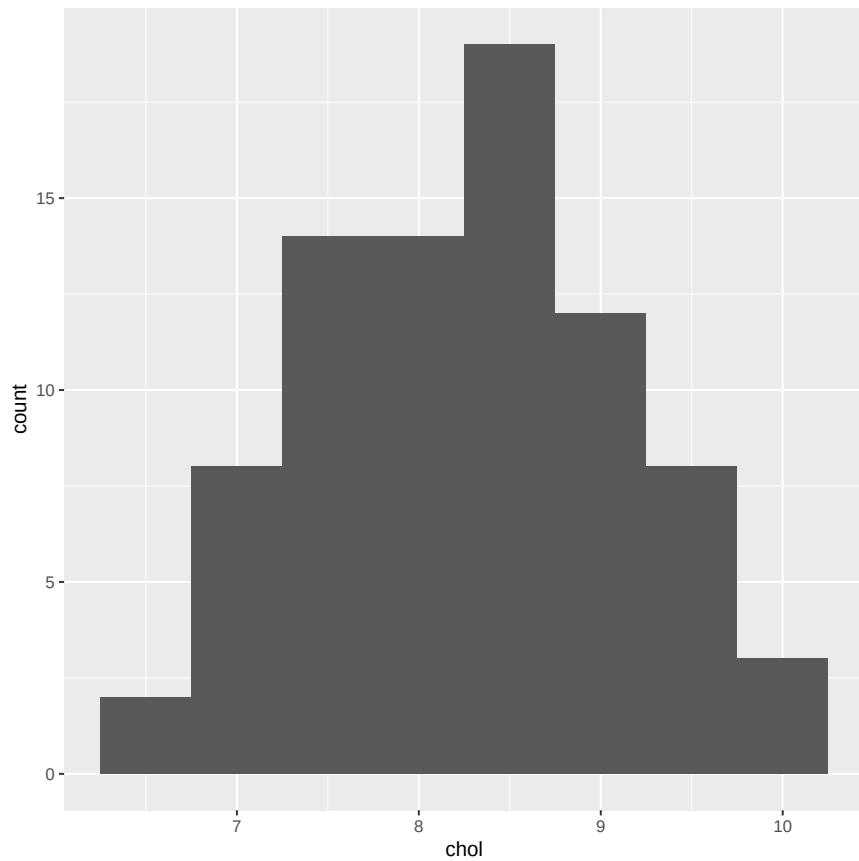
Load the `ggplot2` package,

```
library(ggplot2)
```

In `ggplot2`,

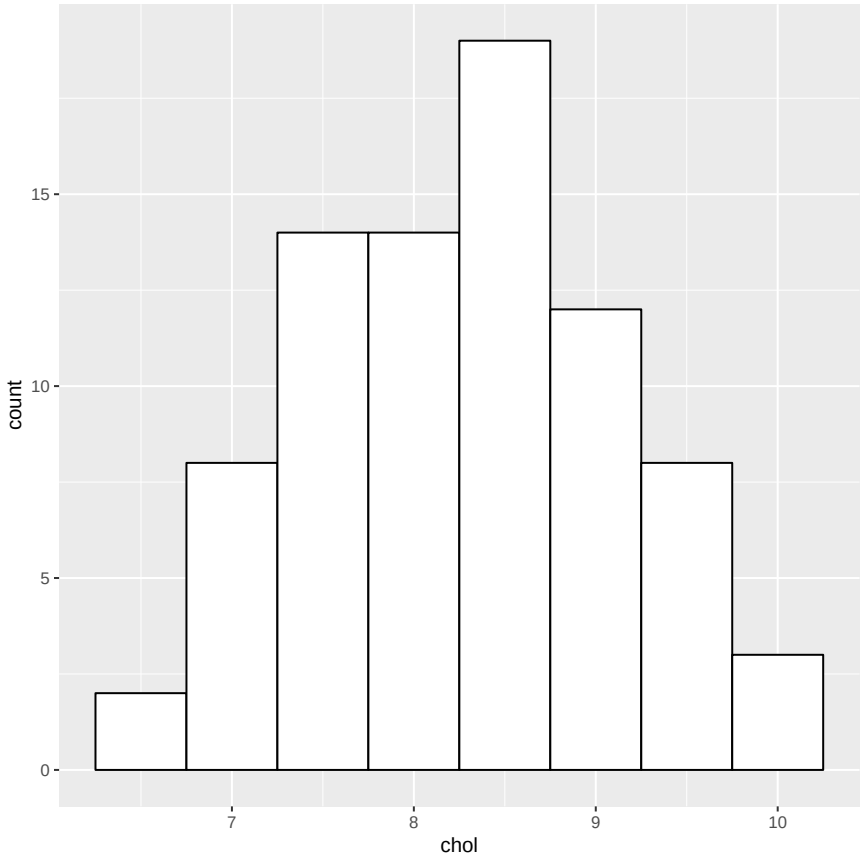
1. type `ggplot(data = X)` function to choose the dataset,
2. the `aes()` for variable or variables to be plotted,
3. then we use `geom_X` to specify the geometric (X) form of the plot.

```
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.

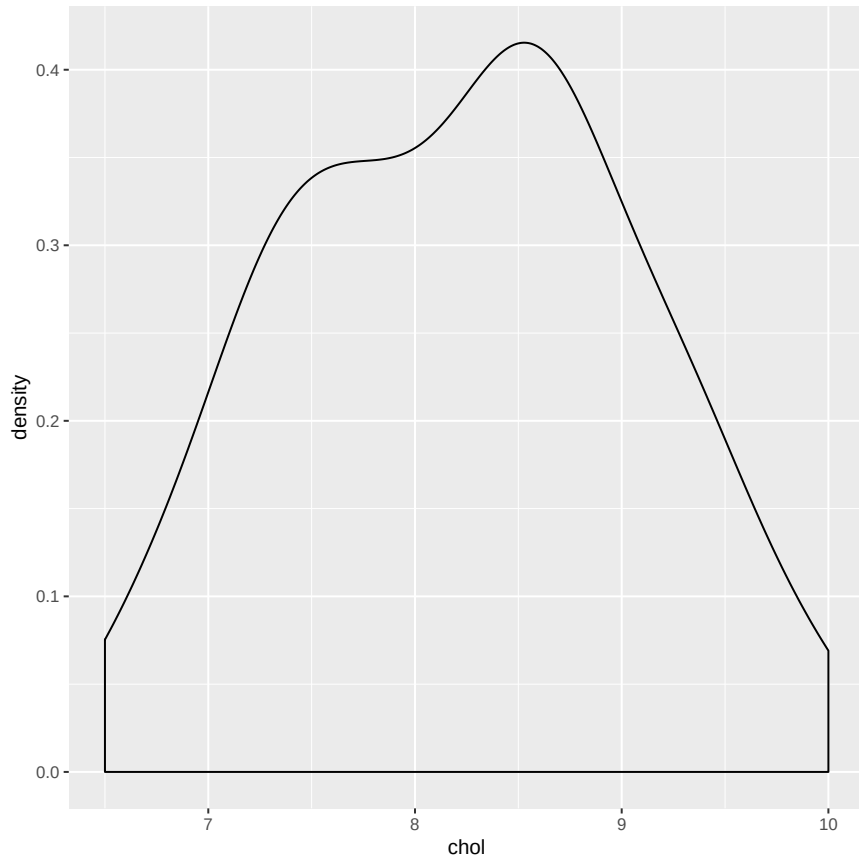
```
ggplot(cholest, aes(x = chol)) + geom_histogram(binwidth = 0.5,
                                                colour = "black", fill = "white")
```



Density curve

Density is useful to examine the distribution of observations.

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



Combining the histogram and the density curve

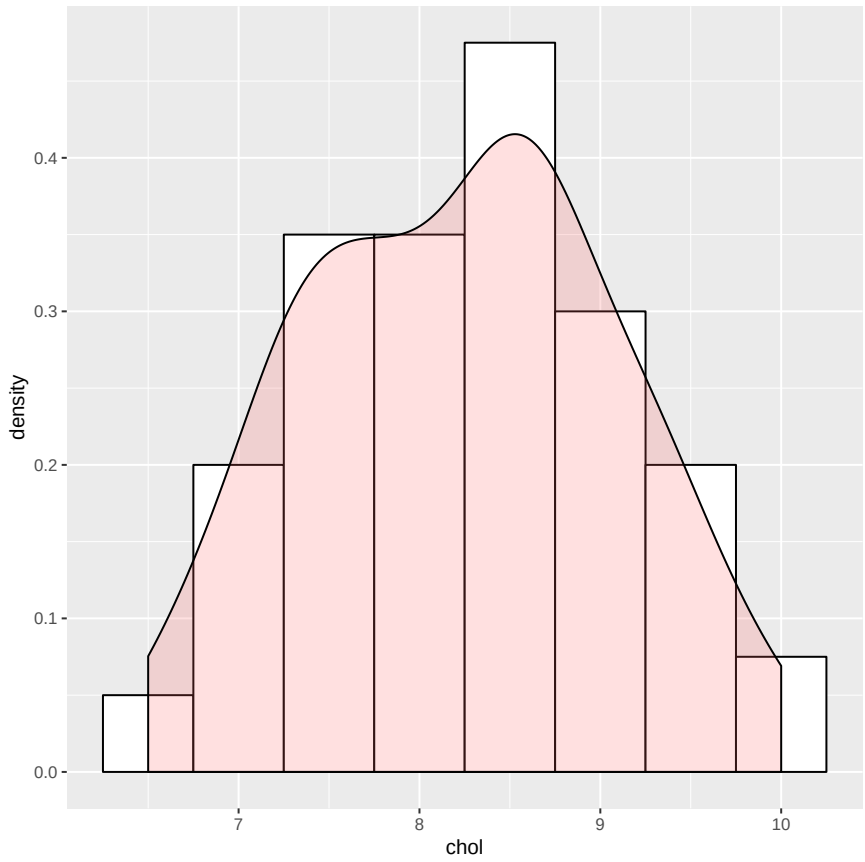
`ggplot2` allows plot to be displayed together. We can combine multiple plots in one single plot by overlaying multiple plots on one another.

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

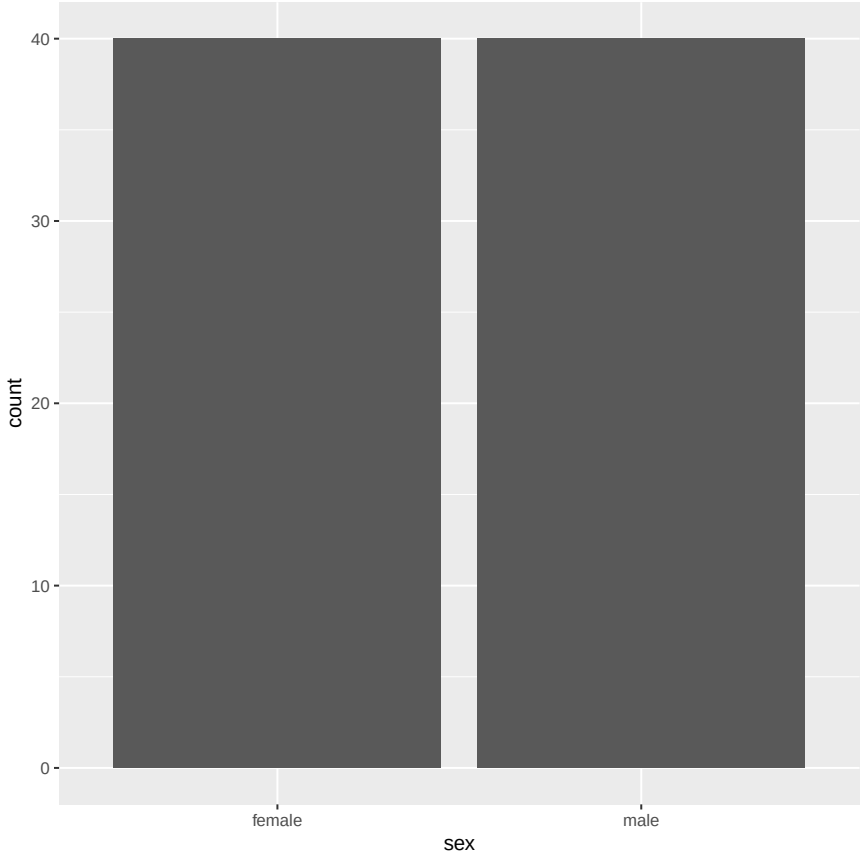
```
ggplot(data = cholest, aes(x = chol)) +
  geom_histogram(aes(y = ..density..), binwidth = 0.5, colour = "black", fill = "white") +
  geom_density(alpha = .2, fill = "#FF6666")
```



4.6.2 One variable: Plotting a categorical variable

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

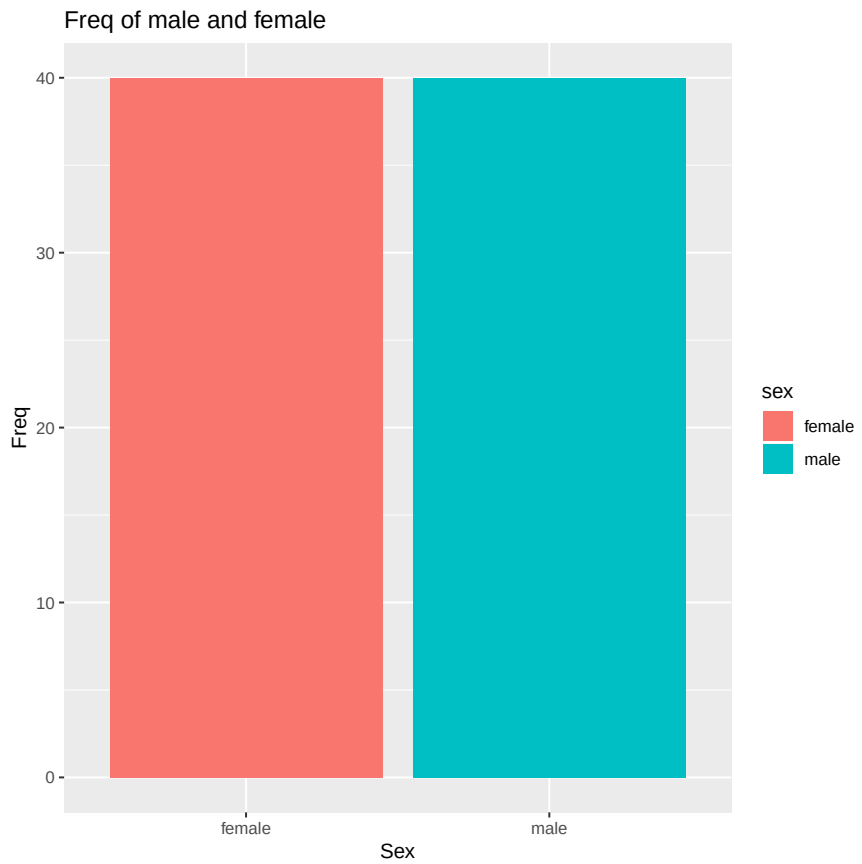
```
sex_bar <- ggplot(data = cholest, aes(sex))
sex_bar + geom_bar()
```



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(sex, fill = sex)) +
  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)/

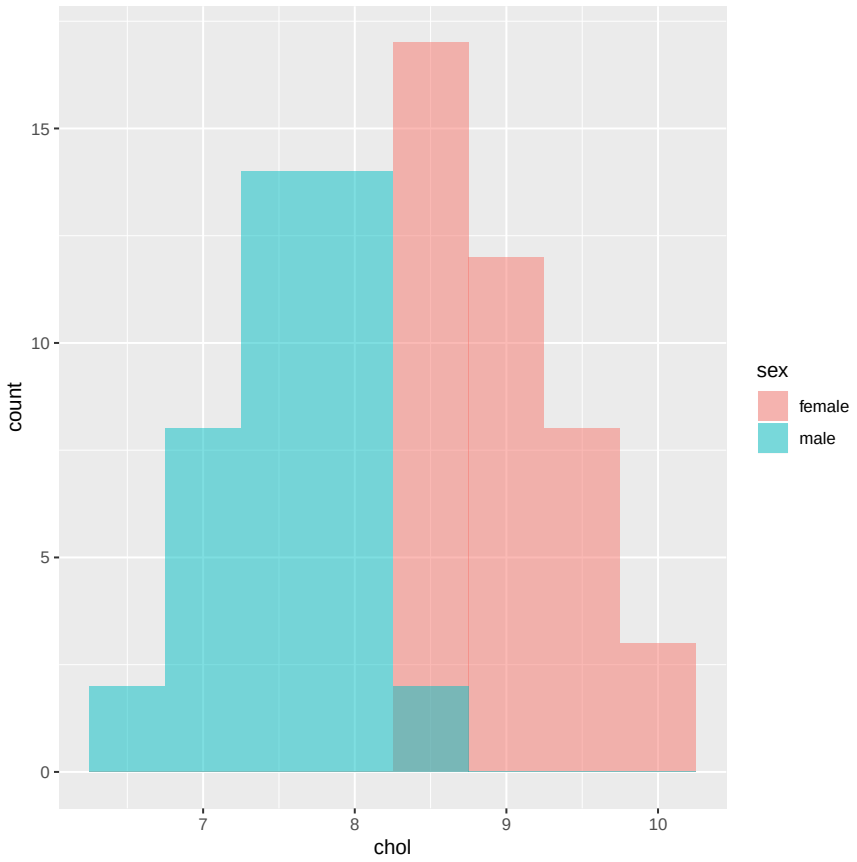
4.6.3 Two variables: Plotting a numerical and a categorical variable

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.

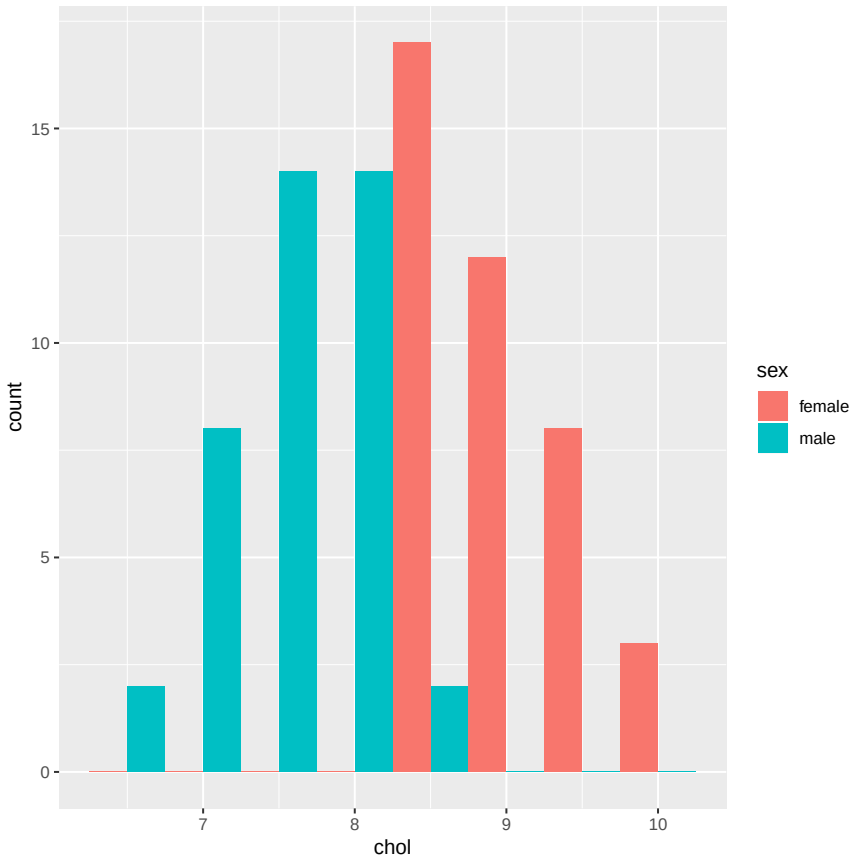
Overlaying histograms

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



Interleaving histograms

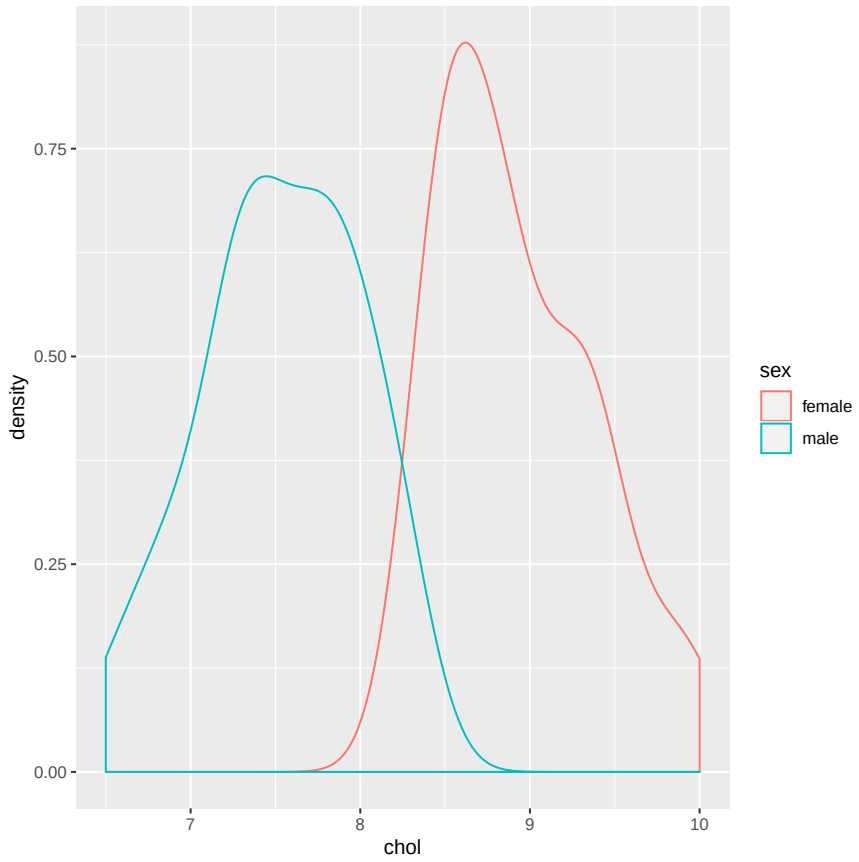
```
ggplot(cholest, aes(x = chol, fill = sex)) +
    geom_histogram(binwidth = .5, position = "dodge")
```



Overlaying density plots

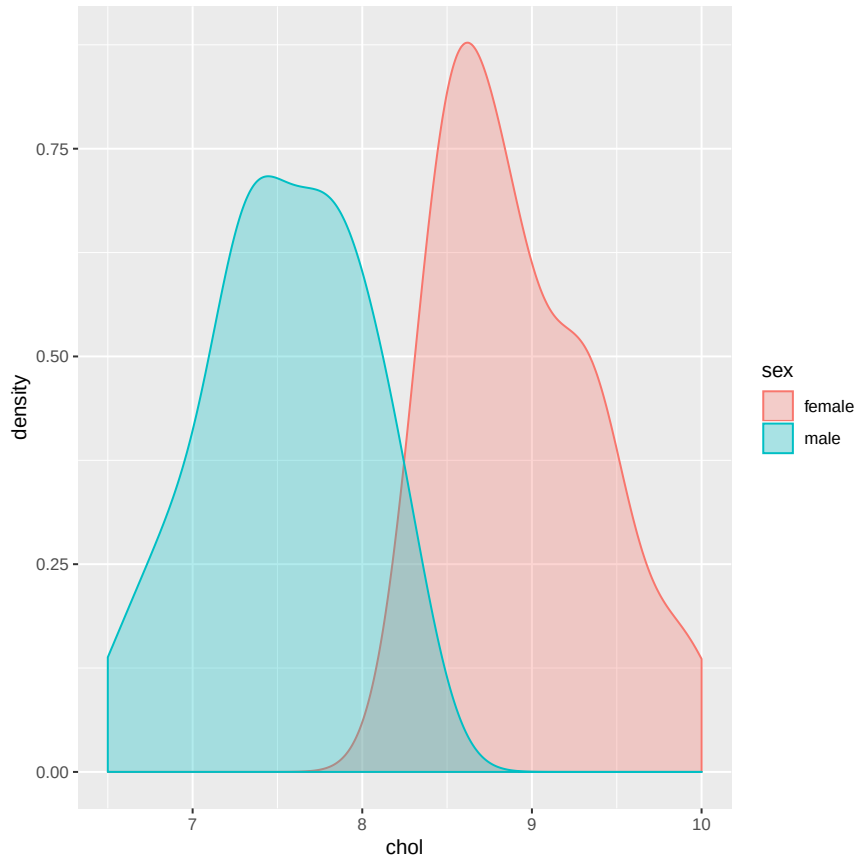
Full transparent

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



Now, try set the transparency at 30%

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



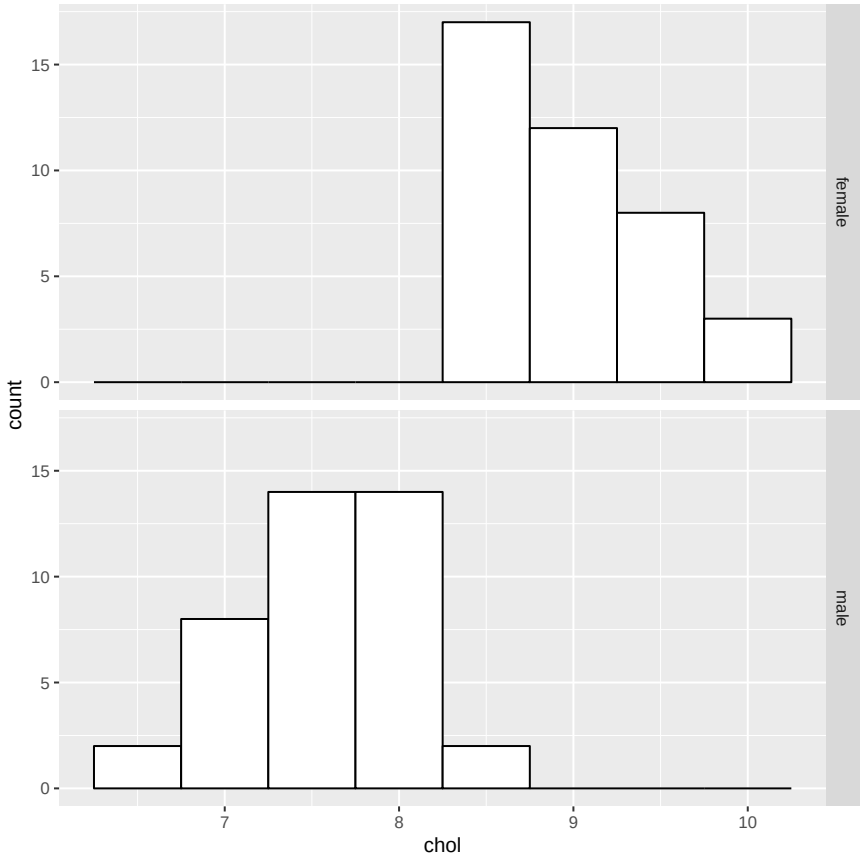
Using facets

We use `facet_grid()` to split the plot. There are two types of facetting the plot:

1. Vertical facet.
2. Horizontal facet.

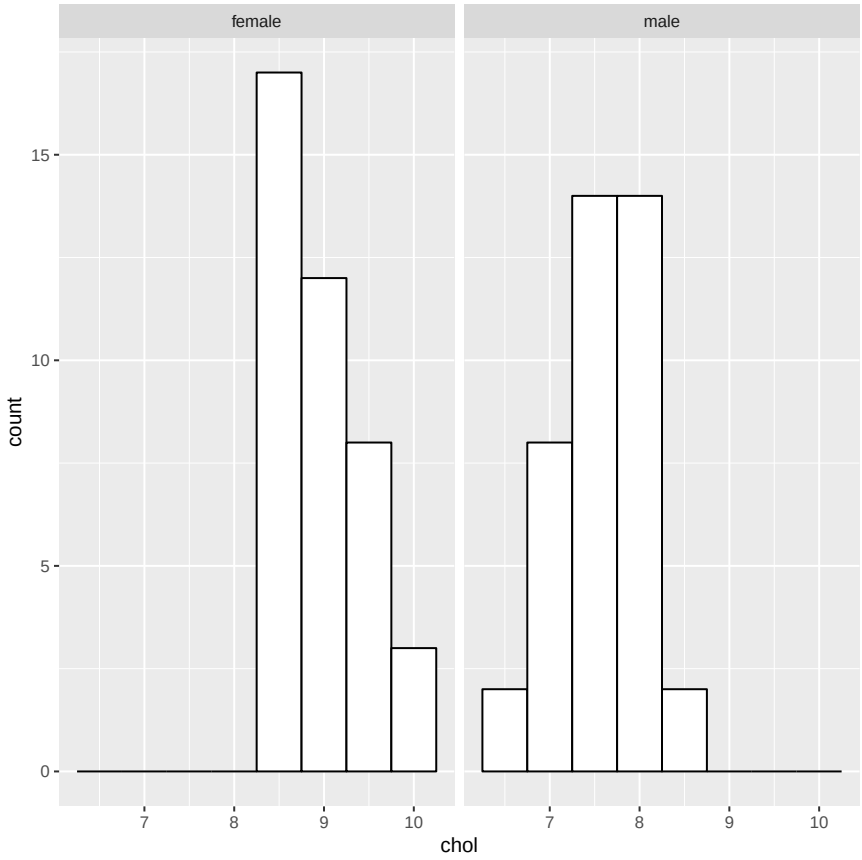
The vertical facets

```
ggplot(data = cholest, aes(x = chol)) +
  geom_histogram(binwidth = .5, colour = "black", fill = "white") +
  facet_grid(sex ~ .)
```



The horizontal facets

```
ggplot(data = cholest, aes(x = chol)) +
  geom_histogram(binwidth = .5, colour = "black", fill = "white") +
  facet_grid(. ~ sex)
```



4.6.4 Saving plots in ggplot2

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

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

4.7 Summary

In this chapter, we learned to plot graphs in R, using the built-in functions and additional packages. We also learned how powerful R can be to generate visually beautiful graphs and how customizable the graphs are.

In the next chapter, we will learn how to combine outputs into custom-made texts, labels and tables. This will be useful in reporting and summarizing your results for publication, and labeling axes on your plots.

References

Arifin, W. N., & Yusoff, M. S. B. (2017). Item response theory for medical educationists. *Education in Medicine Journal*, *9*(9), 69–81.

Fox, J., Weisberg, S., & Price, B. (2018). *Car: Companion to applied regression.* Retrieved from https://CRAN.R-project.org/package=car

Friendly, M. (2009). Milestones in the history of thematic cartography, statistical graphics, and data visualization. Retrieved from http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf

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/

R Core Team. (2018). *Foreign: Read data stored by 'minitab', 's', 'sas', 'spss', 'stata', 'systat', 'weka', 'dBase', ...* Retrieved from https://CRAN.R-project.org/package=foreign

Revelle, W. (2018). *Psych: Procedures for psychological, psychometric, and personality research.* Retrieved from https://CRAN.R-project.org/package=psych

Sarkar, D. (2017). *Lattice: Trellis graphics for r.* Retrieved from https://CRAN.R-project.org/package=lattice

Tufte, E. (1983). *The visual display of quantitative information.* Graphics Press. Retrieved from https://books.google.com.my/books?id=BHazAAAAIAAJ

Wickham, H. (2016). *Plyr: Tools for splitting, applying and combining data.* Retrieved from https://CRAN.R-project.org/package=plyr

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., & Woo, K. (2018). *Ggplot2: Create elegant data visualisations using the grammar of graphics.* Retrieved from https://CRAN.R-project.org/package=ggplot2

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