

# Applied Methods in Statistics

*Thore Egeland and Raju Rimal*

*Year: 2018*



# Chapter 1

## Practical Information

Exercises in this site is relevant for Stat340 course which discusses some of the applied topics in statistics. For the exercises, we will use open source R statistical software along with RStudio, an integrated development environment for R. We advise student to install the latest version of R and RStudio on their laptop computer. In addition, we will use few packages in R which we will discuss during the exercise period. Students are highly encouraged to complete these exercises on their own and also participate in Group Exercises. Follow the link below to install R and RStudio.

**Install R and Install RStudio**

**See:** Lecture and Exercise Plan and Reference Books

## Lecture and Exercise Plan

Week	Topics	Exercises
Week 6 (Feb. 05)	Overview, R and R Studio	Getting Started
Week 7 (Feb. 12)	Regression Analysis	Exercise 1
Week 8 (Feb. 19)	Analysis of Variance	Exercise 2
Week 9 (Feb. 26)	Principal Component Analysis	Exercise 3
Week 10 (Mar. 05)	Multivariate Statistics (PCA, PCR)	Exercise 4
Week 11 (Mar. 12)	Cluster Analysis	Exercise 5
Week 12 (Mar. 19)	Classification	Exercise 6
Week 14 - 15 (Apr. 2, Apr. 9)	Generalized Linear Models	Exercise 7-8
Week 16 (Apr. 16)	Random Effects Models	Exercise 9
Week 17 (Apr. 23)	Mixed Effects Models	Exercise 10

**See:** Practical Information | Reference Books

## Reference Books

**See:** Practical Information | Lecture and Exercise Plan

# Chapter 2

## Getting Started

In this section, we will dive into R and RStudio and get used to with it to some extent. We will continue learning about R and RStudio as we go on. R is an open source programming language basically used in data analysis. In R there are many packages that are created for specific purposes and they have made R rich and powerful. In this course, apart from default R package (that is installed and already loaded), we will use few other packages which we will install and load as we go through our exercises. We can use following command to install a package. Below, a car package is used as an example:

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

To load the package we use library function as,

```
library(car)
```

Following screenshot help you to install package using RStudio IDE,

### Exercise 1: Create New Project

Creating a project allows us to organize the files and related materials during our study. File => New Project opens a window to create new project. It will be easier to access all the resources, if all the scripts and datasets are within a main folder, i.e. the project folder.

The examples in this exercise uses the project folder as the main folder. Although it is not necessary, but throughout the exercises we will use `_data` folder as a folder containing all

the data that we are using in the course.

## Exercise 2: Importing data in R

The usual data sources can be a text file in txt or csv format or spreadsheet(excel) file in xls orxlsx. Data and R-objects can also be imported from rds, rda or rdata. Below, we will discuss these in detail. In addition you can also find some animated images showing how we can import data in RStudio for each of these file formats.

### Import txt or csv

Base R-package has `read.table` and `read.csv` for importing a text or comma separated file (csv) files. Download `bodydata.txt` and to import it in R as,

```
bodydata <- read.table("_data/bodydata.txt", header = TRUE)
```

Here the argument `header` is `TRUE` if the data has header in its first row. The argument `sep` takes `\t`, `,` or `;` based on if the columns in the text data are tab-separated, comma-separated or separated by semi-colons. If the decimal values in the data are represented by `,`, the `dec` argument takes the value `,`. For further help see: `?read.table`

### Import Microsoft Excel spreadsheet

An R-package `readxl` helps to import excel file. If it is not installed, you should install it as,

```
install.packages("readxl")
```

Download `bodydata.xlsx` from canvas and load it to R as,

```
library(readxl)
bodydata <- read_excel("_data/bodydata.xlsx", sheet = 1)
```

For further help and arguments on this function load the library as `library(readxl)` and see: `?read_excel` or `?read_xlsx`.

## Load rdata, rda or rds

One can save data, models and other R-objects in rdata or rds format. In order to load rdata, we can use load function. Download bodydata.rdata from canvas and load it to R.

```
load("_data/bodydata.Rdata")
```

## Reading data from clipboard (“pasting” copied data into an R object)

You can import data in clipboard in R. For example the data you copied in Excel or Word files.

```
bodydata <- read.table(file = "clipboard", header = TRUE)
```

In Mac, you need to do,

```
bodydata <- read.table(file = pipe("pbpaste"), header = TRUE)
```

This allows us to get the data from anywhere just after they are copied.

After every import in above examples, we have saved our imported data in bodydata variable. This an R-object that holds any kind of data-structures such as matrix, data.frame, list or fitted models. We can find these R-objects in “Environment” tab in RStudio.

## Exercise 3: Exporting data to a file

To export an r-object to a file, write.table function is used. For example, we can export the bodydata table we just imported as a text file named bodydata-export.txt as,

```
write.table(bodydata, file = "_data/bodydata-export.txt",  
            row.names = FALSE, quote = FALSE)
```

We can also use write.csv function to export the data in csv format. The txt and csv file format only holds data in tabular structures. Sometimes we need to save other R-objects such as fitted models or list for which we can use Rdata format. We can save our bodydata objects in Rdata format as,

```
save(bodydata, file = "_data/bodydata-export.rdata")
```

This will export bodydata object to a file named bodydata-export.rdata in \_data folder in your project directory.

## Exercise 4: Data Structure in R

The dataset we imported in Exercise 2: Importing data in R is a data frame. DataFrame is a structure that R uses to keep the data in that particular format. If you do `class(bodydata)` for the data we have imported before, we can see `data.frame` as its class. There are other data structures in R. Some basic structure that R uses are discussed below:

### Vector

A vector is a one-dimensional object where you can store elements of different modes such as “logical” (TRUE or FALSE), “integer”, “numeric”, “character” etc. All elements of a vector must be of same mode. For example,

```
x <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
y <- c("TRUE", "FALSE", "Not Sure")
z <- c(2, 3, 5, 6, 10)
```

Here, x, y and z are of class logical, character and numeric respectively. Although in vector y we have TRUE and FALSE they are in character format. The function `c` is used to define a vector. However functions that are used to create sequences also gives us a vector. For example,

```
(a_sequence <- seq(from = 0, to = 10, by = 2))
```

```
[1] 0 2 4 6 8 10
```

```
(b_sequence <- 1:10)
```

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

Here both `a_sequence` and `b_sequence` are vector. Give special attention to the way we have created the sequence of numbers. It will be useful in many situations in future



exercises.

## Matrix

A matrix is a two dimensional structure with row and column. As this is an extension of vector structure, matrix must have elements of same mode as in a vector. For example:

```
(a_matrix <- matrix(1:25, nrow = 5, ncol = 5))
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	6	11	16	21
[2,]	2	7	12	17	22
[3,]	3	8	13	18	23
[4,]	4	9	14	19	24
[5,]	5	10	15	20	25

```
(b_matrix <- diag(1:5))
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	0	0	0	0
[2,]	0	2	0	0	0
[3,]	0	0	3	0	0
[4,]	0	0	0	4	0
[5,]	0	0	0	0	5

Here, `a_matrix` is created from a vector of sequence of 1 to 25 in 5 rows and 5 columns. We can also define a diagonal matrix as `b_matrix` with numbers from 1 to 5 in its diagonal.

## Array

An array is an extension of Matrix structure in three or more dimension. We can define an array as,

```
(an_array <- array(1:24, dim = c(2, 4, 3)))
```

```
, , 1
```

	[,1]	[,2]	[,3]	[,4]
--	------	------	------	------

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

```
, , 2
```

```
      [,1] [,2] [,3] [,4]
[1,]  9   11   13   15
[2,] 10   12   14   16
```

```
, , 3
```

```
      [,1] [,2] [,3] [,4]
[1,] 17   19   21   23
[2,] 18   20   22   24
```

## List

All the above structure we discussed require that the the elements in them to be of same mode such as numeric, character and logical. Sometimes it is necessary to keep objects of different modes in same place. List is a structure that helps in such situation. A list can contain list, matrix, vector, numeric or any other data structure as its elements. For example:

```
a_list <- list(
  a_matrix = matrix(1:6, nrow = 2, ncol = 3),
  a_vector = 2:7,
  a_list = list(a = 1, b = 3:6),
  a_logical = c(TRUE, FALSE, TRUE, NA)
)
a_list
```

```
$a_matrix
      [,1] [,2] [,3]
[1,]  1    3    5
[2,]  2    4    6
```

```
$a_vector
[1] 2 3 4 5 6 7
```

```
$a_list
$a_list$a
[1] 1
```

```
$a_list$b
[1] 3 4 5 6
```

```
$a_logical
[1] TRUE FALSE TRUE NA
```

In above example, `a_list` contains a matrix, a numeric vector, a list and a logical vector.

## Data Frame

Data Frame is a list kept in tabular structure. Every column of a data frame has a name assigned to it. The `bodydata` dataset we have imported is an example of data frame. Data frame is the most used data structure to keep data in tabular format. Lets create a data frame:

```
a_dataframe <- data.frame(
  character = c("a", "b", "c"),
  numeric = 1:3,
  logical = c(TRUE, FALSE, NA)
)
a_dataframe
```

```
  character numeric logical
1         a         1    TRUE
2         b         2   FALSE
3         c         3     NA
```

Every column of a `data.frame` is a vector. Different columns of a data frame can contain element of different modes. For example: the first column can be a character vector while

the second column can be a numeric vector as in the example above.

## Exercise 5: Exploring the data

### Structure of an R-object

The first command you need to learn is `str` function in order to explore any object in R. Lets apply this to our `bodydata`,

```
str(bodydata)
```

```
'data.frame':  407 obs. of  4 variables:
 $ Weight      : num  65.6 80.7 72.6 78.8 74.8 86.4 78.4 62 81.6 76.6 ...
 $ Height      : num  174 194 186 187 182 ...
 $ Age         : num   21 28 23 22 21 26 27 23 21 23 ...
 $ Circumference: num  71.5 83.2 77.8 80 82.5 82 76.8 68.5 77.5 81.9 ...
```

This output shows us that `bodydata` is a `data.frame` with 407 rows and 4 numeric variables - Weight, Height, Age, Circumference.

### Accessing elements from R-objects

Different data structure have different way of accessing elements from them.

#### Extracting elements from vector, matrix and array

For vector, matrix and array we can use `[` for accessing their elements. Lets create a vector, a matrix and an array as follows,

```
a_vector <- c("one", "two", "three", "four", "five")
a_matrix <- matrix(1:24, nrow = 3, ncol = 8)
an_array <- array(1:24, dim = c(2, 3, 4))
```

**Extracting element at position 3 to 5 in `a_vector`** `a_vector[3:5]` will give three, four, five, the elements at position index 3, 4, and 5. In R, position index starts from 1.

**Extracting element in rows 2, 3 and columns 2, 4, 6, 8 from a\_matrix** This is a two dimensional structure, we give row-index and column-index inside [ operator separated by comma as,

```
a_matrix[c(2, 3), c(2, 4, 6, 8)]
```

	[,1]	[,2]	[,3]	[,4]
[1,]	5	11	17	23
[2,]	6	12	18	24

We can also write this as,

```
a_matrix[2:3, seq(from = 2, to = 8, by = 2)]
```

	[,1]	[,2]	[,3]	[,4]
[1,]	5	11	17	23
[2,]	6	12	18	24

Here seq(from = 2, to = 8, by = 2) create sequence even integer from 2 to 8 which is used as column index for extracting elements from a\_matrix.

**Extracting first element of an\_array:** Here an\_array is an array structure of dimension three. So, we have to use three index vector inside [ operator in order to extract element from it. For instance an\_array[1, 1, 1] gives 1 as its first element of the array.

In all these structures we can only supply index of one or more dimension. For example, a\_matrix[1:2,] where we have only mentioned the row index, will give elements in *first* and *second* row from all columns. i.e.

```
a_matrix[1:2, ]
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
[1,]	1	4	7	10	13	16	19	22
[2,]	2	5	8	11	14	17	20	23

### Extracting elements from data.frame and list

Lets create a data.frame and a list as,

```

a_dataframe <- data.frame(
  fertilizer = c("Low", "Low", "High", "High"),
  yield = c(12.5, 13.1, 15.3, 16.2)
)
a_list <- list(
  facebook = data.frame(
    name = c("Gareth", "Raju", "Marek", "Franchisco"),
    has_profile = c(TRUE, TRUE, FALSE, TRUE)
  ),
  twitter = c("@gareth", "@raju", "@marek", "franchisco")
)

```

**Extracting third and fourth row of fertilizer from a\_dataframe** Same as extracting elements as matrix as discussed above we can use row and column index as `a_dataframe[3:4, 1]`. We have used 1 in place of column index since fertilizer is in first column. We can also use name instead as `a_dataframe[3:4, "fertilizer"]`.

```
a_dataframe[3:4, "fertilizer"]
```

```

[1] High High
Levels: High Low

```

**Extracting first element of a\_list** We can use `[[` for extracting elements from `a_list`. For example `a_list[[1]]` will give the first element of the list. Here in our list we have two elements with names `facebook` and `twitter`. So, we can also use their names as `a_list[["facebook"]]` which is not possible if they do not have any name.

```
a_list[["facebook"]]
```

	name	has_profile
1	Gareth	TRUE
2	Raju	TRUE
3	Marek	FALSE
4	Franchisco	TRUE

We can also use `$` operator to extract elements from named list and a data frame. For example, `bodydata$Weight` extracts `Weight` variable from `bodydata` dataset.

## View Data in RStudio

Newer version of RStudio support viewing data in different structures. To view bodydata we have imported in Exercise 2: Importing data in R, we can use `View(bodydata)`. If you have not imported the, you need to follow the exercise and import the data first. We can also click the data in “Environment” tab to view it.

## Summary of data

We can compute basic descriptive summary statistics using `summary` function as,

```
summary(bodydata)
```

Weight	Height	Age	Circumference
Min. : 42.00	Min. :149.5	Min. :18.00	Min. : 57.90
1st Qu.: 58.45	1st Qu.:163.9	1st Qu.:23.00	1st Qu.: 67.95
Median : 68.60	Median :171.4	Median :27.00	Median : 75.60
Mean : 69.19	Mean :171.3	Mean :29.91	Mean : 76.91
3rd Qu.: 78.80	3rd Qu.:177.8	3rd Qu.:35.00	3rd Qu.: 84.30
Max. :108.60	Max. :198.1	Max. :67.00	Max. :113.20

## Dimension of data

The number of elements in a data structure like vector and list we can use `length` function. For example: if we extract `Weight` variable from `bodydata` we will get a numeric vector. The length of this vector is,

```
length(bodydata$Weight)
```

A multi-dimensional data structure like matrix, array and data frame has dimension. We can use `dim` function to find the dimension.

```
dim(bodydata)
```

```
[1] 407 4
```

Here, the first and second item refers to the number of rows and number of columns of `bodydata`. Similarly, we can use `nrow(bodydata)` and `ncol(bodydata)` to obtain these

number individually.

## Lets Practice

- 1) Take a look at the top 5 rows of bodydata
- 2) Take a look at the top 5 rows of Height and Circumference variables of bodydata
- 3) Apply summary function on Age variable of bodydata

## Exercise 6: Subsets of data and logical operators

### Logical vector and index vector

A lot of times we want to get a subset of data filtering rows or columns of a dataframe. For which we can perform logical test and get TRUE or FALSE as result. This vector of logical can then be used to subset the observations from a dataframe.

For example, Lets extract observation from bodydata with Weight greater than 80. You might be wondering why following code does not work,

```
isHeavy <- Weight > 80
```

Error in eval(expr, envir, enclos): object 'Weight' not found

But remember that, the variable Weight is a part of bodydata. We have to extract Weight from the bodydata first. In R, with and within function helps you in this respect. In the following code, with function goes inside bodydata and execute the expression Weight > 80.

```
isHeavy <- with(bodydata, Weight > 80)
```

Here the logical vector isHeavy is computed by performing a logical operation on Weight variable within bodydata. The same operation can be done as,

```
isHeavy <- bodydata$Weight > 80
```

**Take a look at this variable, what is it? :**



```
head(isHeavy)
```

```
[1] FALSE TRUE FALSE FALSE FALSE TRUE
```

Yes, it is a vector of TRUE and FALSE with same length as Weight. Here the condition has compared each element of Weight results TRUE if it is greater than 80 and FALSE if it is less than 80.

**Identify the elements** We can identify which observations that are heavy by the which() function

```
HeavyId <- which(isHeavy)
```

This will return a vector of row index for the observations that are heavy, i.e. greater than 80. So how many are heavy? To find the size of a vector we can use length function.

```
length(HeavyId)
```

```
[1] 94
```

Here, 94 observations have Weight larger than 80.

## Exercise

- 1) Identify who are taller than 180 and save this logical vector as an object called isTall.

```
isTall <- with(bodydata, Height > 180)
```

- 2) How many observations have height taller than 180?

```
TallId <- which(isTall)  
length(TallId)
```

```
[1] 76
```

- 3) How many observations are both tall and heavy? Here, you can use length function as above to find how many person are taller than 180.

```
isBoth <- isHeavy * isTall
```

**How is this computation done?** Here `isHeavy` and `isTall` contains `TRUE` and `FALSE`. The multiplication of logical operator results a logical vector with `TRUE` only if both the vectors are `TRUE` else `FALSE`.

**Alternatively :**

```
isBoth <- which(isHeavy & isTall)
```

The `&` operator result `TRUE` if both `isHeavy` and `isTall` are `TRUE` else, `FALSE` which is same as previous.

## Subsetting data frame

### Example 1

Lets create a subset of the data called `bodydataTallAndHeavy` containing only the observations for tall and heavy persons as defined by `isBoth`.

```
bodydataTallAndHeavy <- bodydata[isBoth, ]
```

For other logical tests see help file `?Comparison`

### Example 2

Lets create a random subset of 50 observations. For this we first sample 50 row index randomly from all rows in `bodydata`. The `sample` function is used for the purpose. In the following code, `nrow(bodydata)` return the number of rows in `bodydata`. The `sample` function takes two argument `x` which can be a vector or a integer and `size` which is the size of the sample to be drawn.

```
idx <- sample(x = nrow(bodydata), size = 50)
```

Here, 50 rows are sampled from the total number of rows and the index of the selected rows are saved on vector `idx`.

Using this vector we can select the observations in `bodydata` to create a new data set called `bodydataRandom` as,

```
bodydataRandom <- bodydata[idx, ]
```

Here is the first five rows of bodydataRandom dataset.

```
head(bodydataRandom, n = 5)
```

	Weight	Height	Age	Circumference
105	55.5	168.9	20	68.0
355	59.8	164.3	23	71.0
157	67.7	175.3	24	75.0
346	66.8	179.8	24	66.0
367	54.0	163.2	18	60.2

## Exercise

Create a subset of dataset bodydata including the observation with Age larger an 55 and Circumference larger than 80. Save this dataset named subdata.

```
idx <- with(bodydata, Age > 55 & Circumference > 80)
subdata <- bodydata[idx, ]
subdata
```

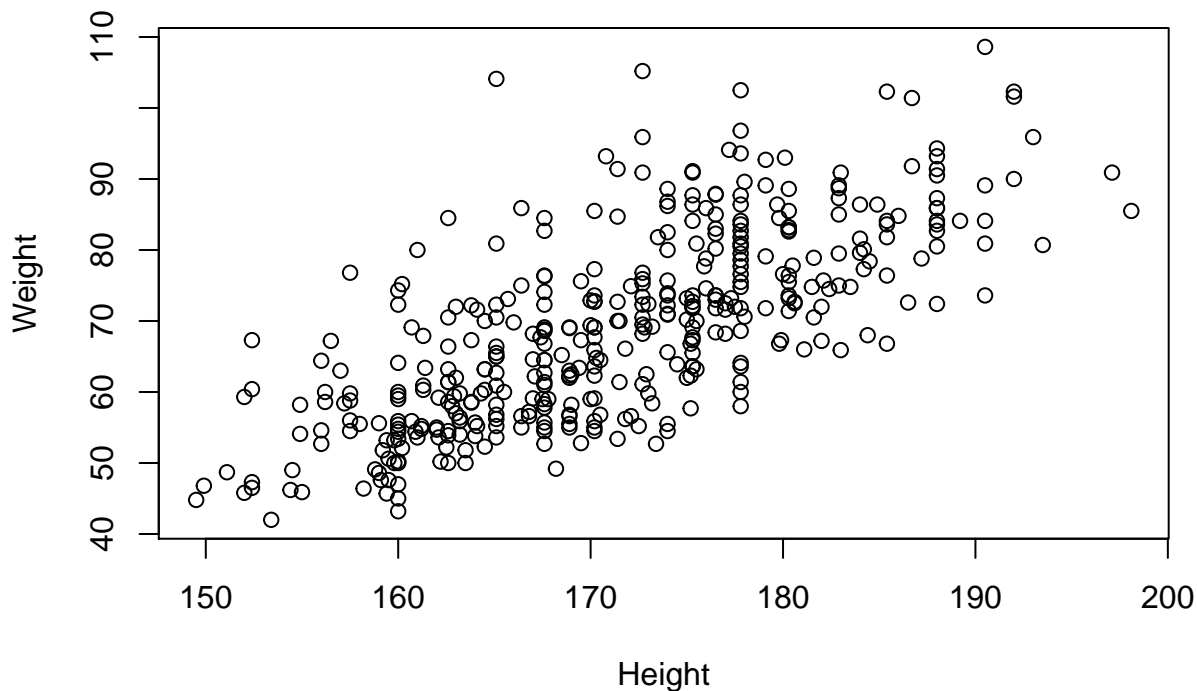
	Weight	Height	Age	Circumference
136	76.4	185.4	62	94.8
189	73.6	175.3	60	90.5
207	66.8	167.6	62	81.5
231	80.0	174.0	65	98.6

For those who are interested in playing more with data, have a look at <http://r4ds.had.co.nz/transform.html>

## Exercise 7: Graphics

Plot the heights versus the weights for all observations in `bodydata`.

```
with(bodydata, plot(x = Height, y = Weight))
```



### Spice up the plot

Check out the presentation for lesson 1 to see how to spice up the plot

### Explore the `?par` help file

Use the `isBoth` variable to create a color vector

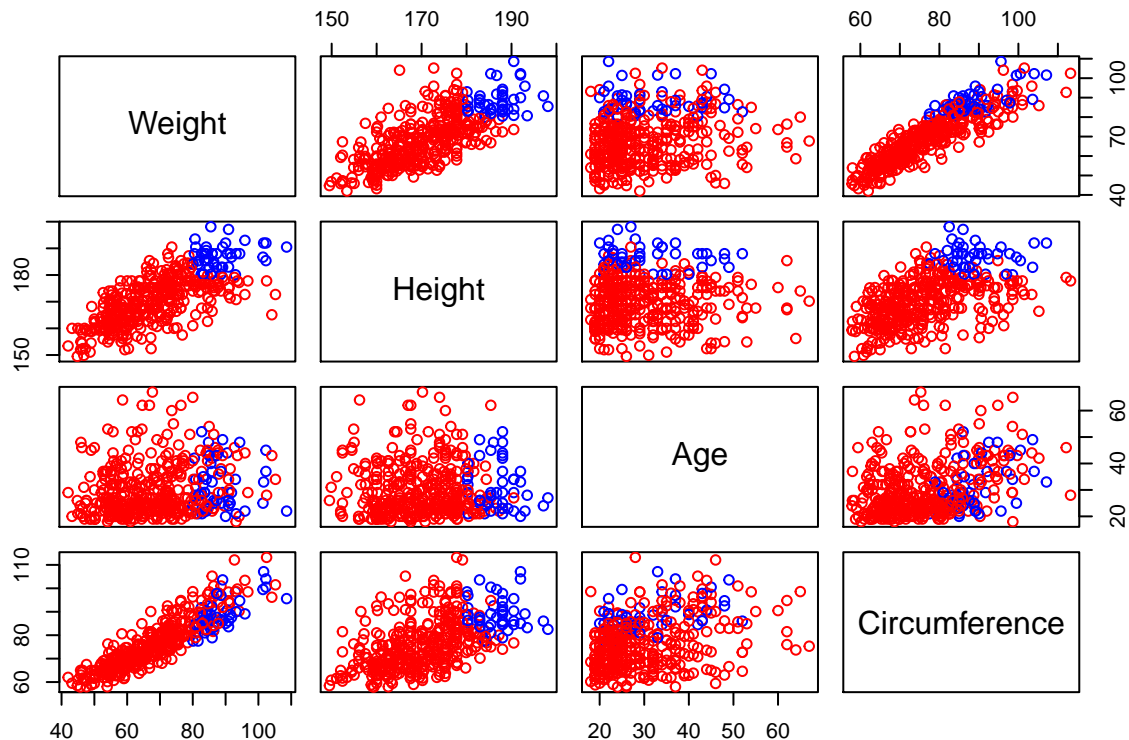
```
mycolors <- ifelse((isHeavy & isTall), "blue", "red")
```

Here, `isHeavy & isTall` returns a logical vector. The `ifelse` function returns blue if TRUE and red if FALSE for each element of the logical vector. The colors are then used in the plot so that all the Heavy and Tall person will be colored "blue" and rest as "red".

Use “mycolors” in the `col` argument of the `plot` function to mark the tall and heavy individuals

**Plot all variables against each other**

```
pairs(bodydata, col = mycolors)
```



**Which variables seem to be most correlated to each other?**

Here, Weight and Circumference seems to have highest correlation.

**Which variables are least correlated to each other?**

Age and Height variables seems to have least correlation.

Check by,

```
cor(bodydata)
```

	Weight	Height	Age	Circumference
Weight	1.0000000	0.72080269	0.18701340	0.8994638
Height	0.7208027	1.00000000	0.04822479	0.5447980
Age	0.1870134	0.04822479	1.00000000	0.3547390
Circumference	0.8994638	0.54479800	0.35473898	1.0000000

This returns the correlation matrix for the variables, and the guess made earlier true. Further, check out the help file for `pairs`, and the examples at the end. Try to make a pairs plot with scatter plots with smoothed lines in the lower left triangle, histograms on the diagonal, and correlation numbers in the upper right triangle.

Lets first create a function which create histogram. The function will later be used in the `pairs` function to create its diagonal plots.

```
panel.hist <- function(x, ...) {
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}
```

Now, create a function that will display correlation on pairs plot.

```
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...) {
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y))
  txt <- format(c(r, 0.123456789), digits = digits)[1]
  txt <- paste0(prefix, txt)
  if (missing(cex.cor)) cex.cor <- 0.8 / strwidth(txt)
  text(0.5, 0.5, txt, cex = cex.cor * r)
}
```

Now, the above functions are implemented on the pairs plot,

```
pairs(bodydata,
      lower.panel = panel.smooth,
```

```
upper.panel = panel.cor,
diag.panel = panel.hist)
```



Here the `panel.smooth` deals with the smooth line on the lower panel of pairs plot.

**Note::** Chapter 5 of the R book contains numerous examples of graphics. **Note::** For those interested in playing around with plots in R checkout: <http://r4ds.had.co.nz/data-visualisation.html>