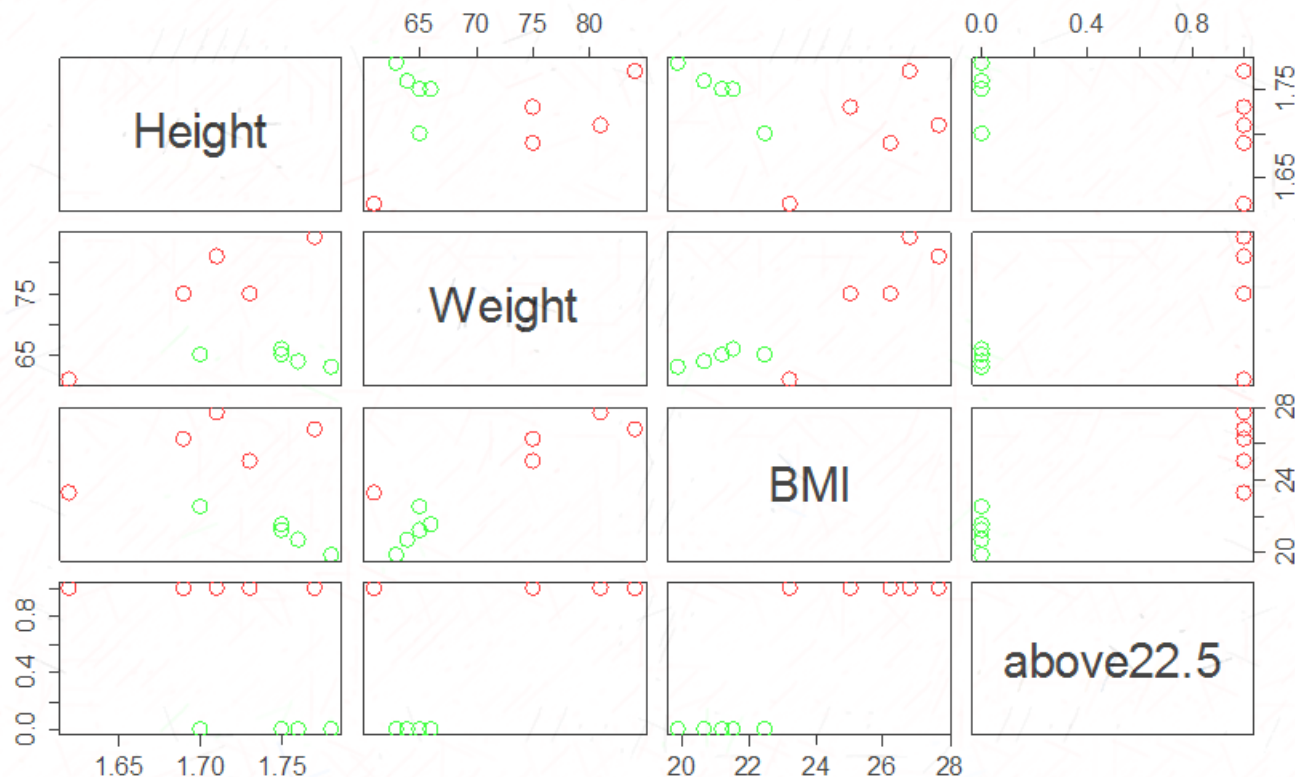


Introduction to Computing for Economics and Management

Lecture 8: Graphics



Acknowledgement

- These slides are adapted from Bert Arnrich's R lecture.



Previous lectures

- In previous lectures we already used graphics
 - Scatter plot of person's height and weight
 - Adding a line to a plot
 - Barplot of word frequencies
 - Plot of random numbers
- We first recapitulate the previous plots
- Next we learn more about graphics

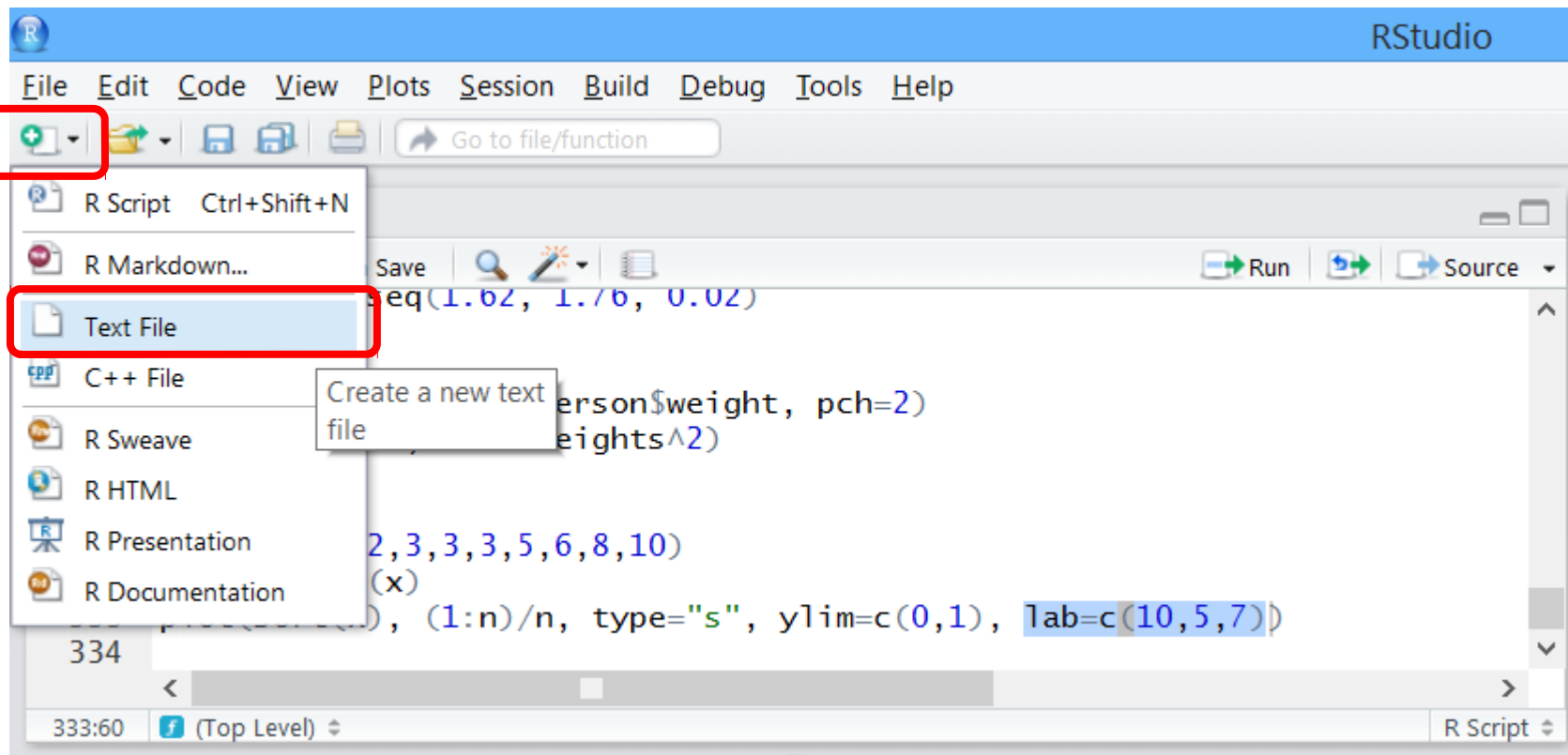
Recap: data import

So far, we have entered our data into R

```
> person.height <- c(Can=1.70, Cem=1.75,  
Hande=1.62)
```

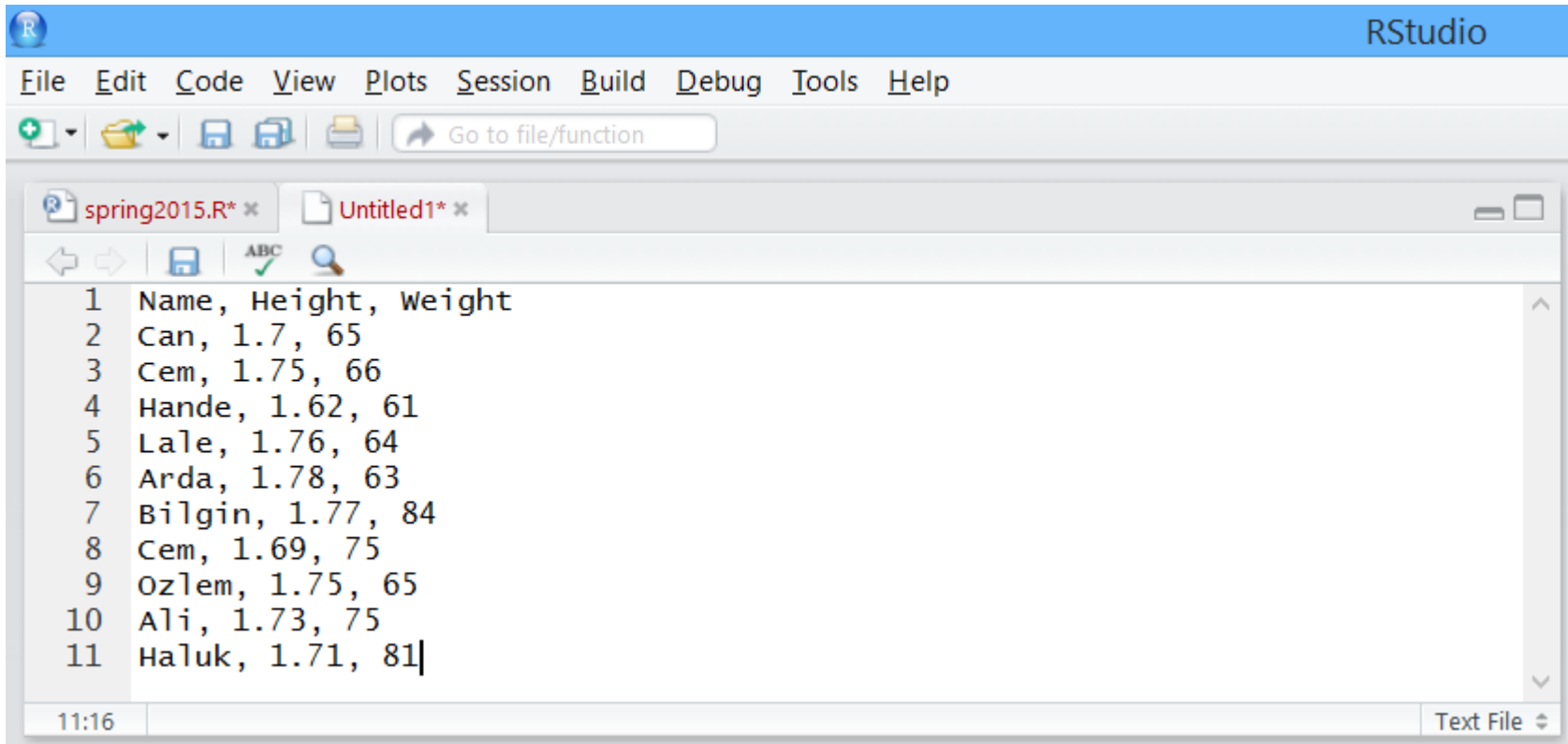
- In practice, data is usually stored in data bases or files and we import it from there
- We prepare a file which contains our data and import the file content into R

Recap: data file



- As a first step we create a new text file using RStudio or an alternative text editor like Notepad++

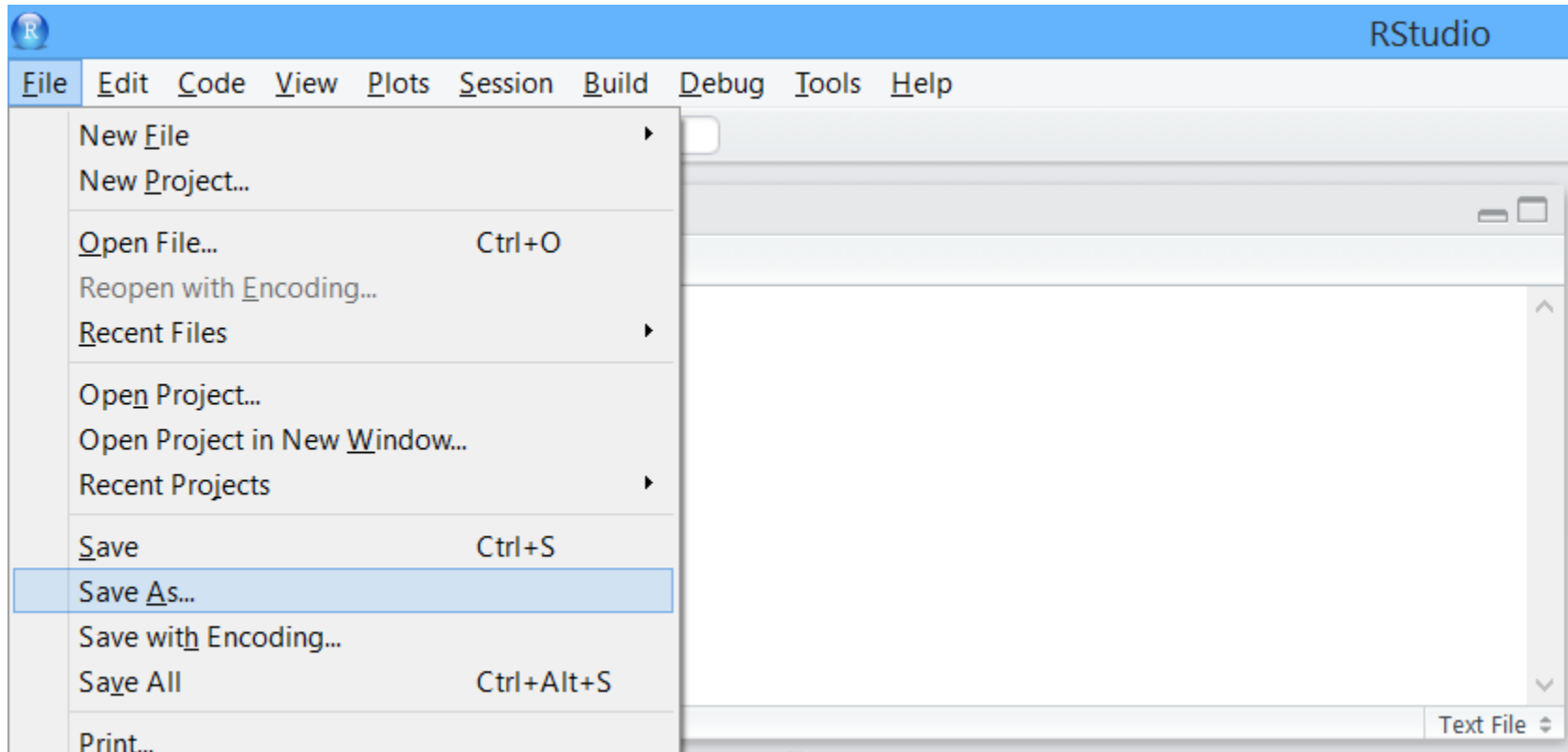
Recap: data file



```
1 Name, Height, Weight
2 Can, 1.7, 65
3 Cem, 1.75, 66
4 Hande, 1.62, 61
5 Lale, 1.76, 64
6 Arda, 1.78, 63
7 Bilgin, 1.77, 84
8 Cem, 1.69, 75
9 Ozlem, 1.75, 65
10 Ali, 1.73, 75
11 Haluk, 1.71, 81
```

- Next, we fill the text file with our data:
 - In the first line we write the file header
 - In the remaining lines we write our data entries
 - We separate each entry by a comma

Recap: data file



- Finally we save the file

Recap: data import with `read.table`

The function `read.table` is the most convenient way to read in a rectangular grid of data from a text file

```
> help(read.table)
read.table(file, header = FALSE, sep = "", quote = "\"'",
  dec = ".", row.names, col.names,
  as.is = !stringsAsFactors,
  na.strings = "NA", colClasses = NA, nrows = -1,
  skip = 0, check.names = TRUE,
  fill = !blank.lines.skip,
  strip.white = FALSE, blank.lines.skip = TRUE,
  comment.char = "#",
  allowEscapes = FALSE, flush = FALSE,
  stringsAsFactors = default.stringsAsFactors(),
  fileEncoding = "", encoding = "unknown", text)
```

Recap: arguments of `read.table`

`file`

- Name of the file which the data are to be read from
- Each row of the table appears as one line of the file
- If it does not contain an absolute path, the file name is relative to the current working directory
- Can also be a complete URL

Recap: arguments of `read.table`

header

- Logical value indicating whether the file contains the names of the variables as its first line
- If header information is available in the file, it will be used for variable names

Recap: arguments of `read.table`

`sep`

- Field separator character
- Values on each line of the file are separated by this character
- Default value `sep = ""` means that the separator is 'white space': one or more spaces, tabs, newlines or carriage returns

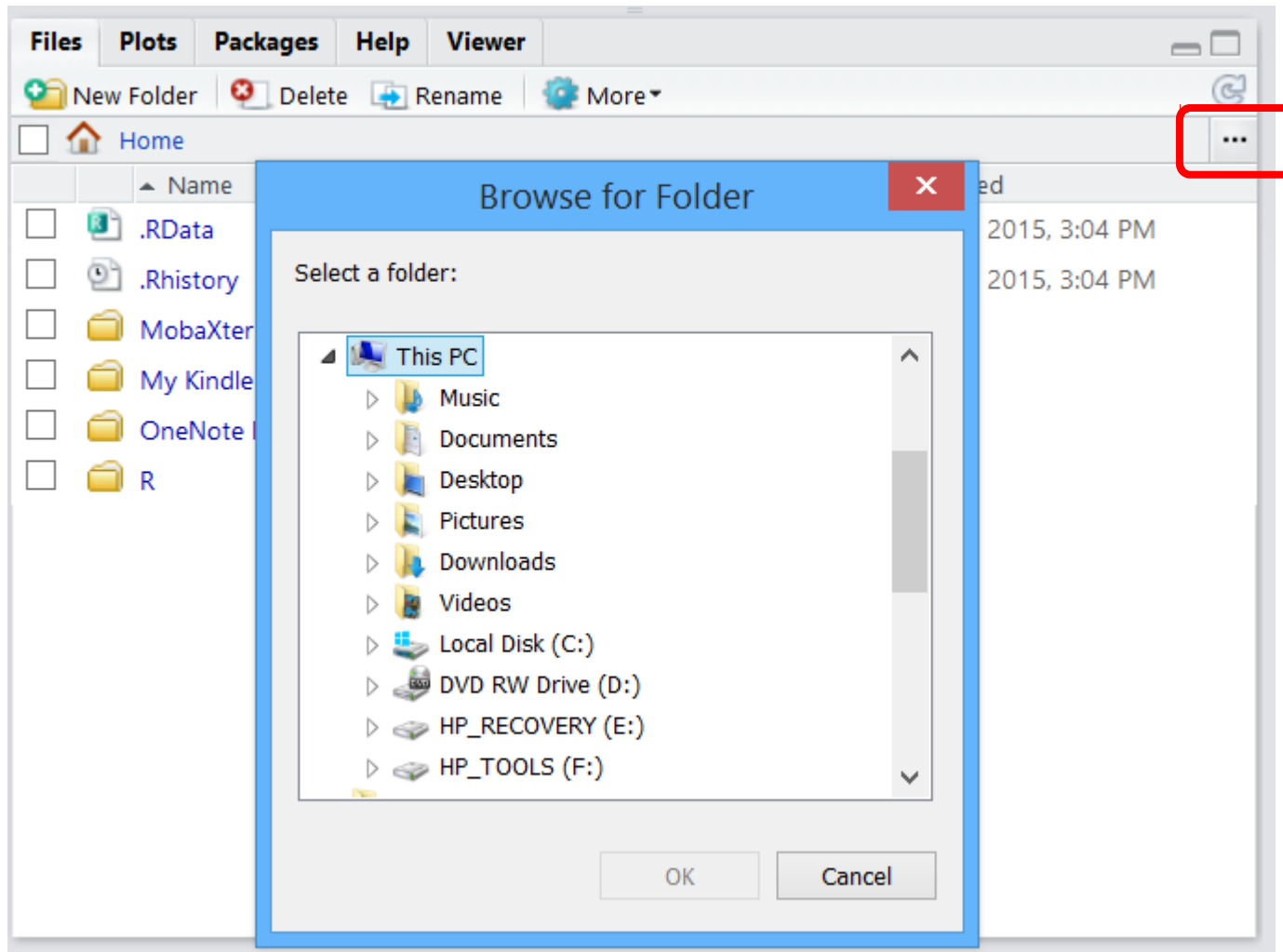
Recap: working directory

Before the actual import, we need to check the current working directory to make sure which path to use when importing the data file

```
> getwd()  
[1] "C:/Users/Bert/Documents"
```

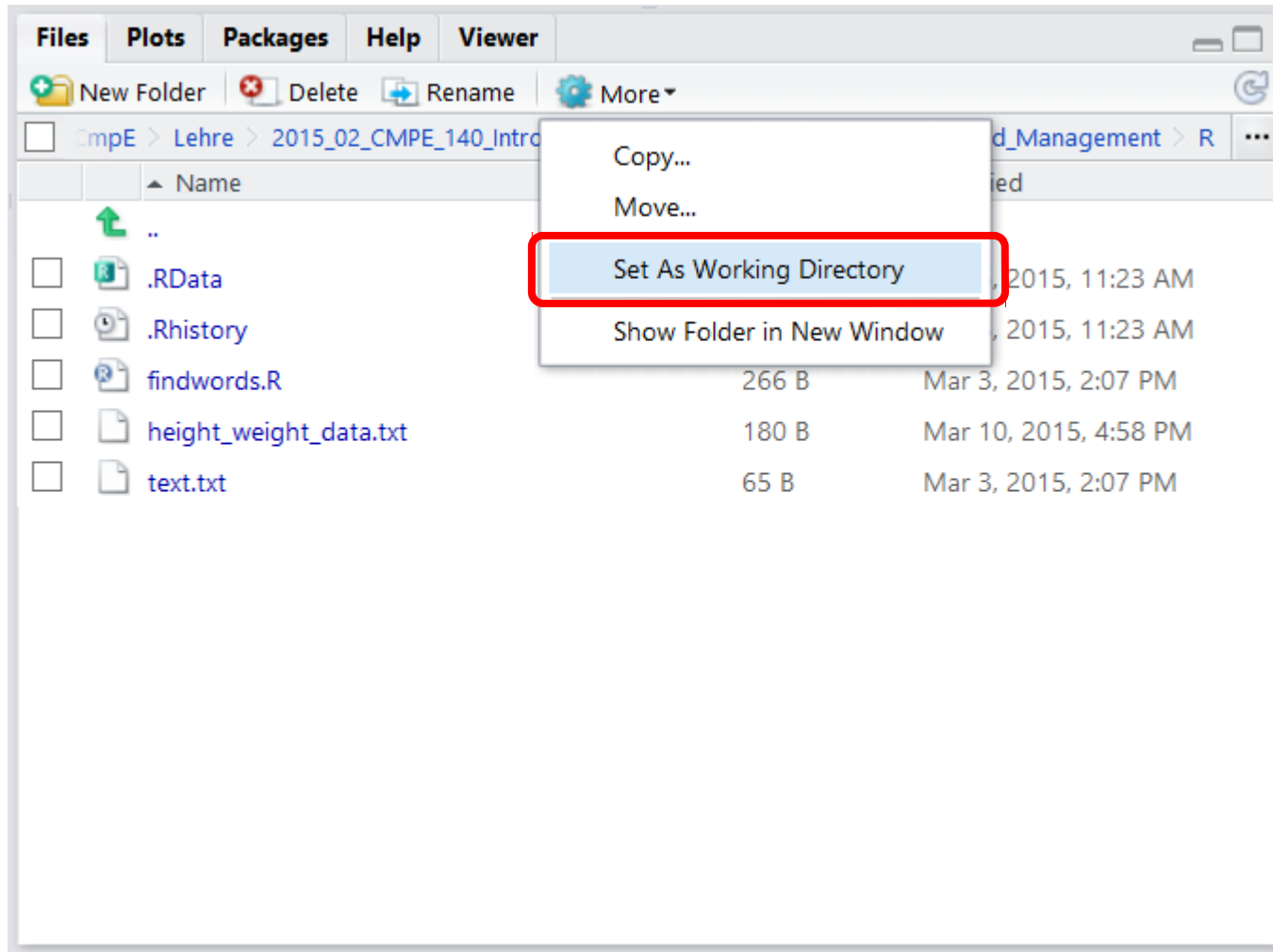
- We change working directory to the path where our data file is located in order to simplify data import

Recap: change working directory in RStudio



- In the files tab we select the “...” item and browse to the folder in which we have stored the text file

Recap: change working directory in RStudio



- In the menu “More”, we select “Set As Working Directory”

Recap: data import

Now, we import the data into the data frame `person.data` by using the function `read.table`

```
> person.data <- read.table(header=TRUE,  
"height_weight_data.txt", sep=",")
```

```
> person.data
```

	Name	Height	Weight
1	Can	1.70	65
2	Cem	1.75	66
3	Hande	1.62	61
4	Lale	1.76	64
5	Arda	1.78	63
6	Bilgin	1.77	84
7	Cem	1.69	75
8	Ozlem	1.75	65
9	Ali	1.73	75
10	Haluk	1.71	81

Recap: data modifications

We add a new column BMI like we did before

```
> person.data$BMI <- person.data$Weight /  
person.data$Height^2
```

```
> person.data
```

	Name	Height	Weight	BMI
1	Can	1.70	65	22.49135
2	Cem	1.75	66	21.55102
3	Hande	1.62	61	23.24341
4	Lale	1.76	64	20.66116
5	Arda	1.78	63	19.88385
6	Bilgin	1.77	84	26.81222
7	Cem	1.69	75	26.25958
8	Ozlem	1.75	65	21.22449
9	Ali	1.73	75	25.05931
10	Haluk	1.71	81	27.70083

Recap: data modifications

We can change the values of a column by reassigning the column with the new values, e.g. rounding BMI

```
> person.data$BMI <- round(person.data$BMI, 2)
```

```
> person.data
```

	Name	Height	Weight	BMI
1	Can	1.70	65	22.49
2	Cem	1.75	66	21.55
3	Hande	1.62	61	23.24
4	Lale	1.76	64	20.66
5	Arda	1.78	63	19.88
6	Bilgin	1.77	84	26.81
7	Cem	1.69	75	26.26
8	Ozlem	1.75	65	21.22
9	Ali	1.73	75	25.06
10	Haluk	1.71	81	27.70

Recap: data modifications

- When creating new columns, we can make use of functions to compute the values of a new column
- Let's recapitulate the `ifelse()` function

`ifelse(test, yes, no)` returns a vector which is created from selected elements from the vectors `yes` and `no`: `yes[i]` is selected if `test[i]` is true and `no[i]` is selected if `test[i]` is false

```
> ifelse(person.height > 1.7, "tall", "small")
      Can      Cem      Hande
"small"  "tall"  "small"
```

Recap: data modifications

Let's use `ifelse()` to create a new column which indicates whether BMI is above 22.5

```
> person.data$above22.5 <- ifelse(person.data$BMI>22.5, T ,F)
```

```
> person.data
```

	Name	Height	Weight	BMI	above22.5
1	Can	1.70	65	22.49	FALSE
2	Cem	1.75	66	21.55	FALSE
3	Hande	1.62	61	23.24	TRUE
4	Lale	1.76	64	20.66	FALSE
5	Arda	1.78	63	19.88	FALSE
6	Bilgin	1.77	84	26.81	TRUE
7	Cem	1.69	75	26.26	TRUE
8	Ozlem	1.75	65	21.22	FALSE
9	Ali	1.73	75	25.06	TRUE
10	Haluk	1.71	81	27.70	TRUE

Recap: scatter plot of person height and weight

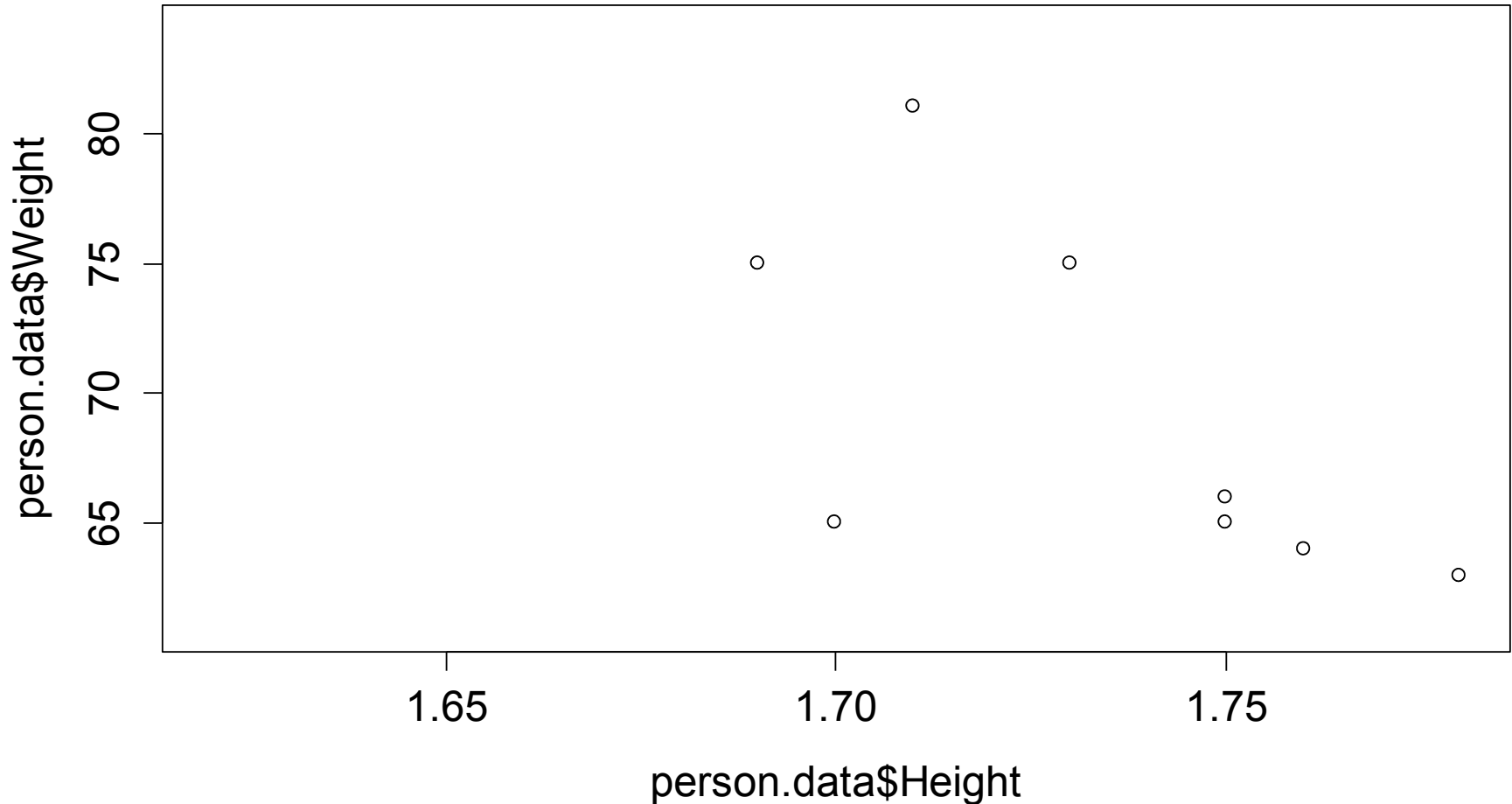


- Beside numeric summary statistics, we use graphics for data exploration is plotting
- R provides us many powerful tools for plotting

We create a simple scatter plot by plotting height on the x-axis and weight on the y-axis

```
> plot(person.data$Height, person.data$Weight)
```

Recap: scatter plot of person height and weight



Recap: scatter plot of person height and weight

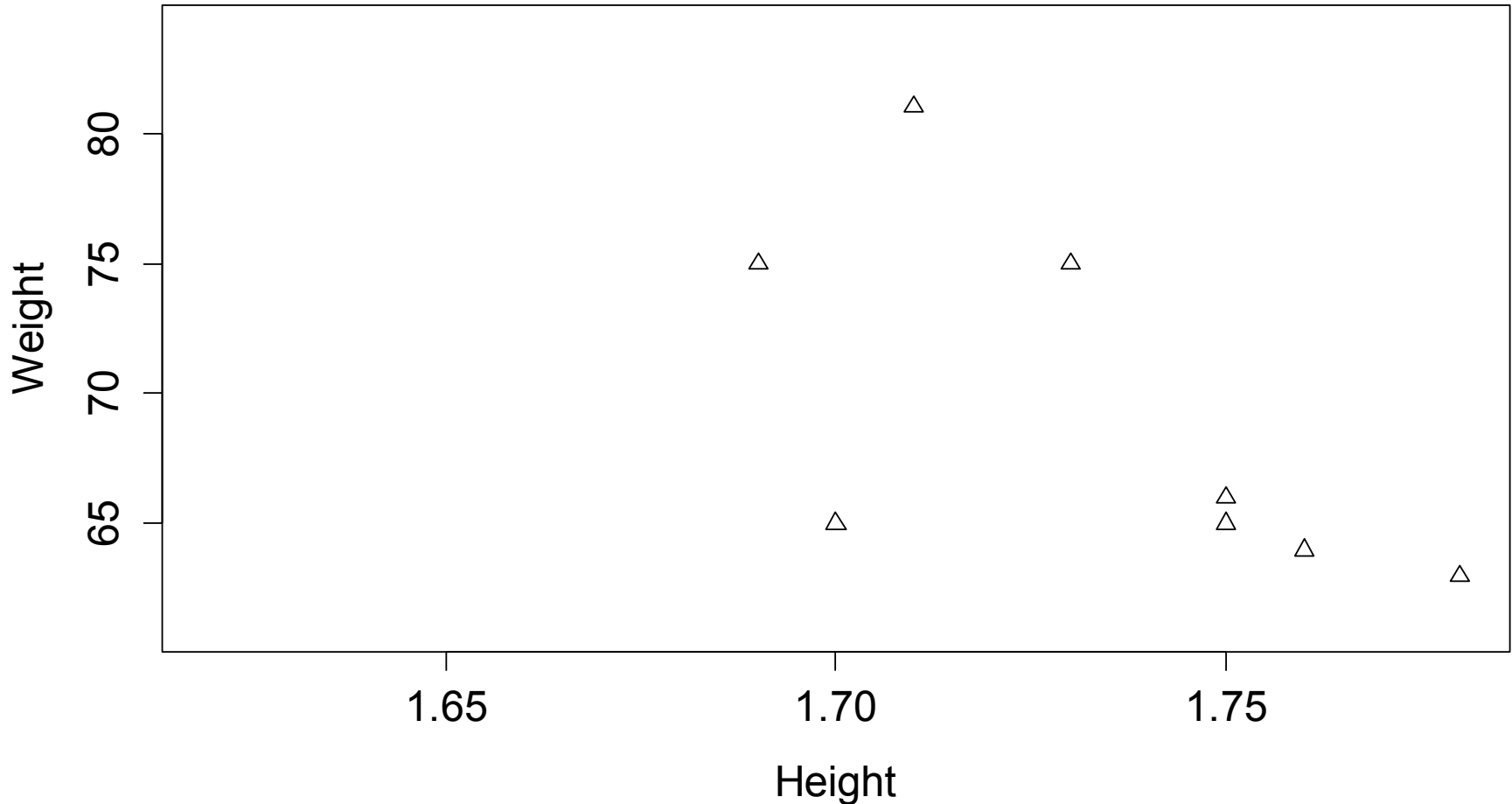


- From the previous plot we observe that the values of Height are plotted versus the corresponding values of Weight
- Similar to other functions, we can provide additional parameters to the plot function

With `pch=2` we change the point type (from circles to triangles) and beside that we provide axis labels

```
> plot(person.data$Height, person.data$Weight,  
pch=2, xlab="Height", ylab="Weight")
```

Recap: scatter plot of person height and weight



Recap: scatter plot of person height and weight



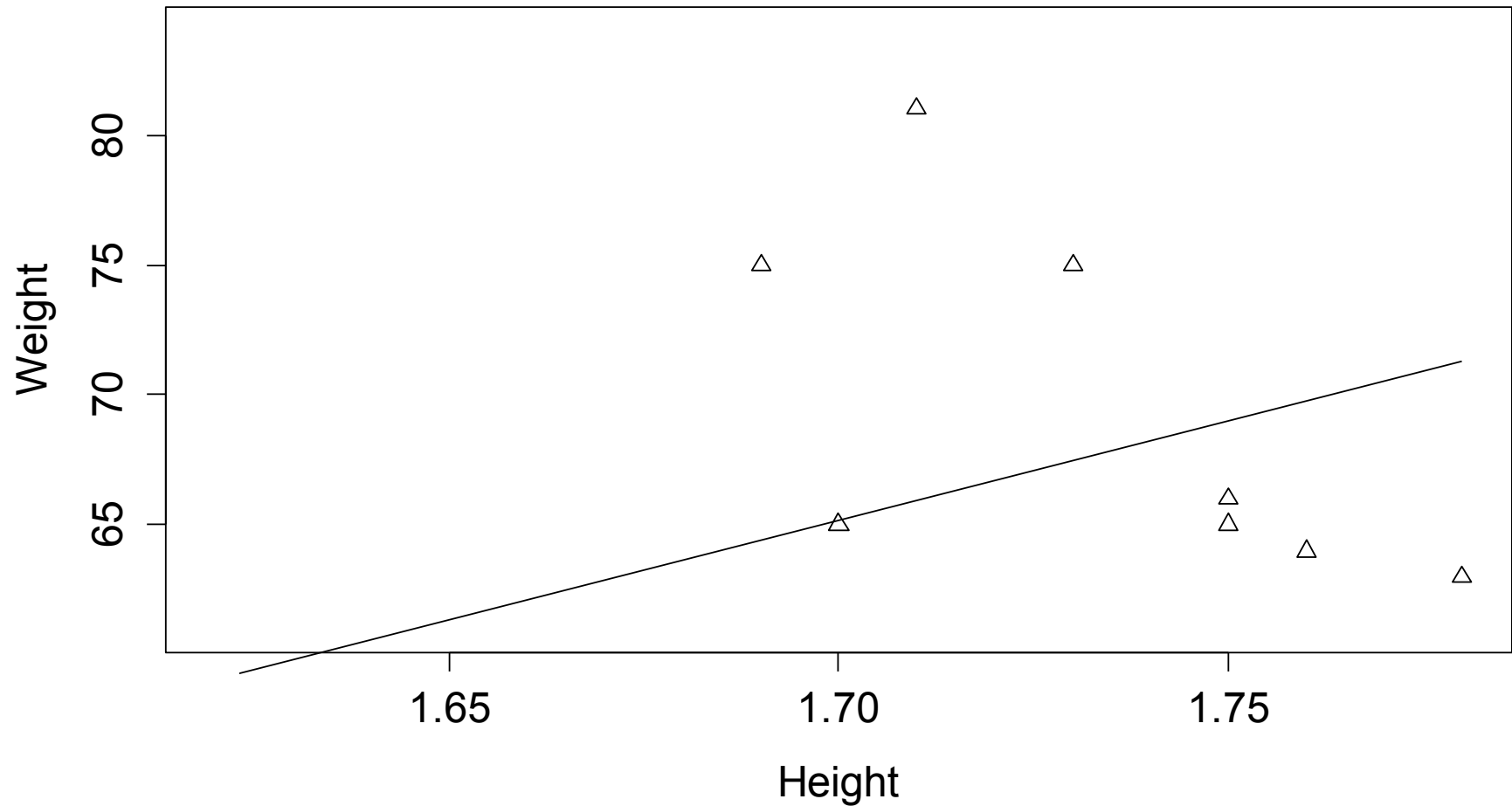
From the summery statistic of Height we learn the minimum and maximum numbers

```
> summary(person.data$Height)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.620  1.702   1.740   1.726   1.758   1.780
```

We draw the line from the minimum height (1.62) to the maximum height (1.78) using $22.5 \times \text{Height}^2$ for the y values of the line

```
> lines(c(1.62, 1.78), 22.5*c(1.62, 1.78)^2)
```

Recap: scatter plot of person height and weight



- Data points above the line represents persons having a BMI above 22.5

Recap: word list

- Web search and other types of textual data mining are of great interest
- Let's assume we have a collection of text documents
- Whenever we search for some term, we would like to retrieve those documents in which our search term appears most often
- Our first goal is to determine which words are in a text and at which location in the text each word occurs

Recap: word list

Let's consider this sentence as our text example:

- a text consists of a word and another word and so on and so forth
- For each word we need to obtain the location in the text:
 - **a** 1 5
 - **text** 2
 - **consists** 3
 - **of** 4
 - **word** 6 9
 - **and** 7 10 13
 - **another** 8
 - **so** 11 14
 - **on** 12
 - **forth** 15

Recap: function `findwords`

```
## finds locations of each word in file
findwords <- function(file, sort.by.freq = F)
{
  # fill word.vec from data in file
  word.vec <- scan(file, "")

  # initialize word list
  word.list <- list()

  # iterate through word vector
  for(i in 1:length(word.vec))
  {
    # store current word in variable word
    word <- word.vec[i]
    # add current word to word.list
    word.list[[word]] <- c(word.list[[word]], i)
  }
  # sort by word frequency or else sort alphabetically
  if(sort.by.freq) {return(word.list[order(sapply(word.list, length),
decreasing = T))]} else {return(word.list[sort(names(word.list))])}
}
```

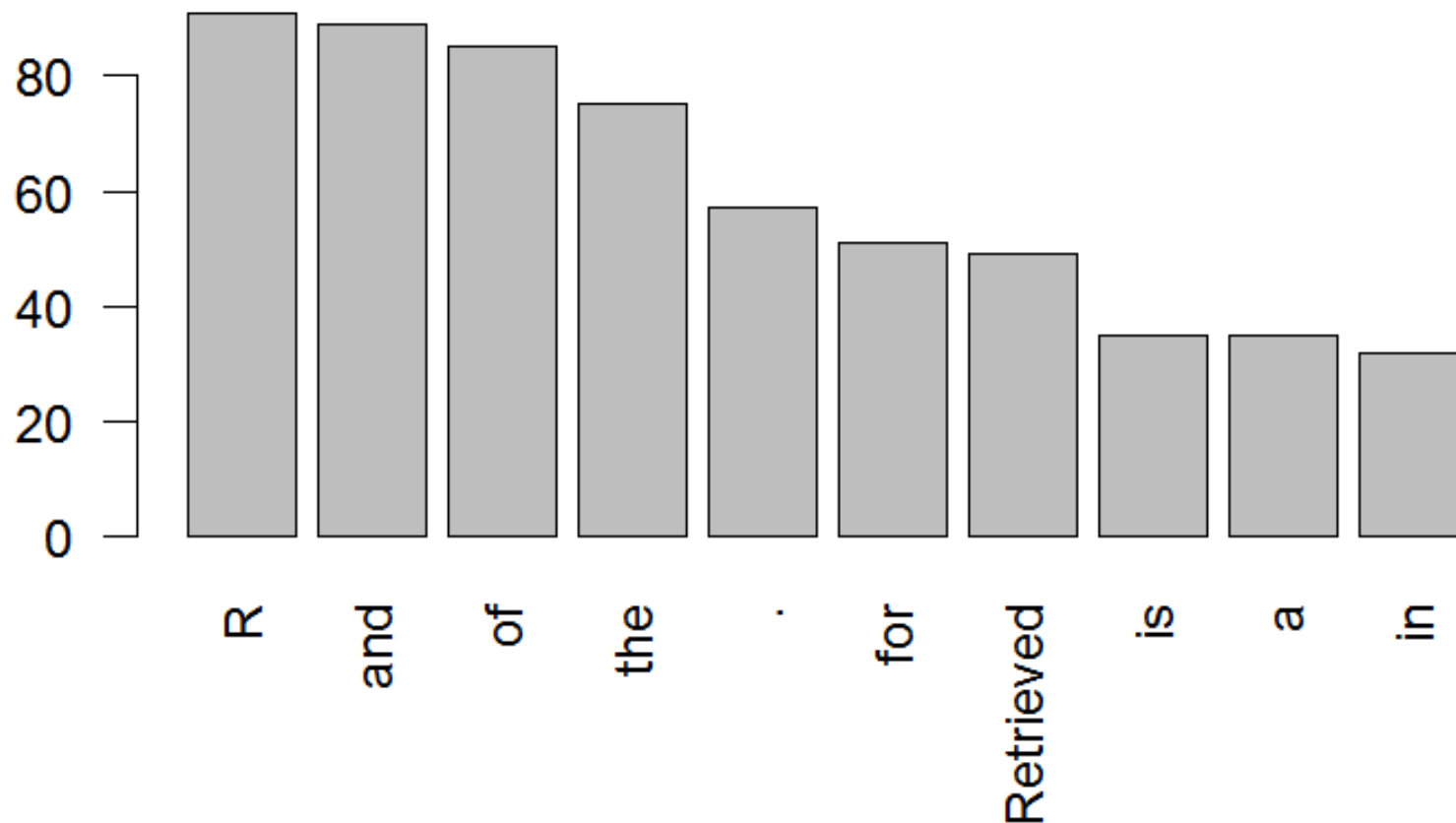
Recap: word frequency in Wikipedia

- Since the for loop allows us to iterate through large texts, let's apply our function to Wikipedia
- We select the Wikipedia article about the R programming language
- We apply findwords and we create a barplot of the 10 most frequent words

Recap: word frequency in Wikipedia

```
> word.list <- findwords("R_wikipedia.txt",  
sort.by.freq=T)  
Read 3395 items  
  
> word.freq <- sapply(word.list, length)  
  
> barplot(word.freq[1:10], las=2)
```

Recap: word frequency in Wikipedia



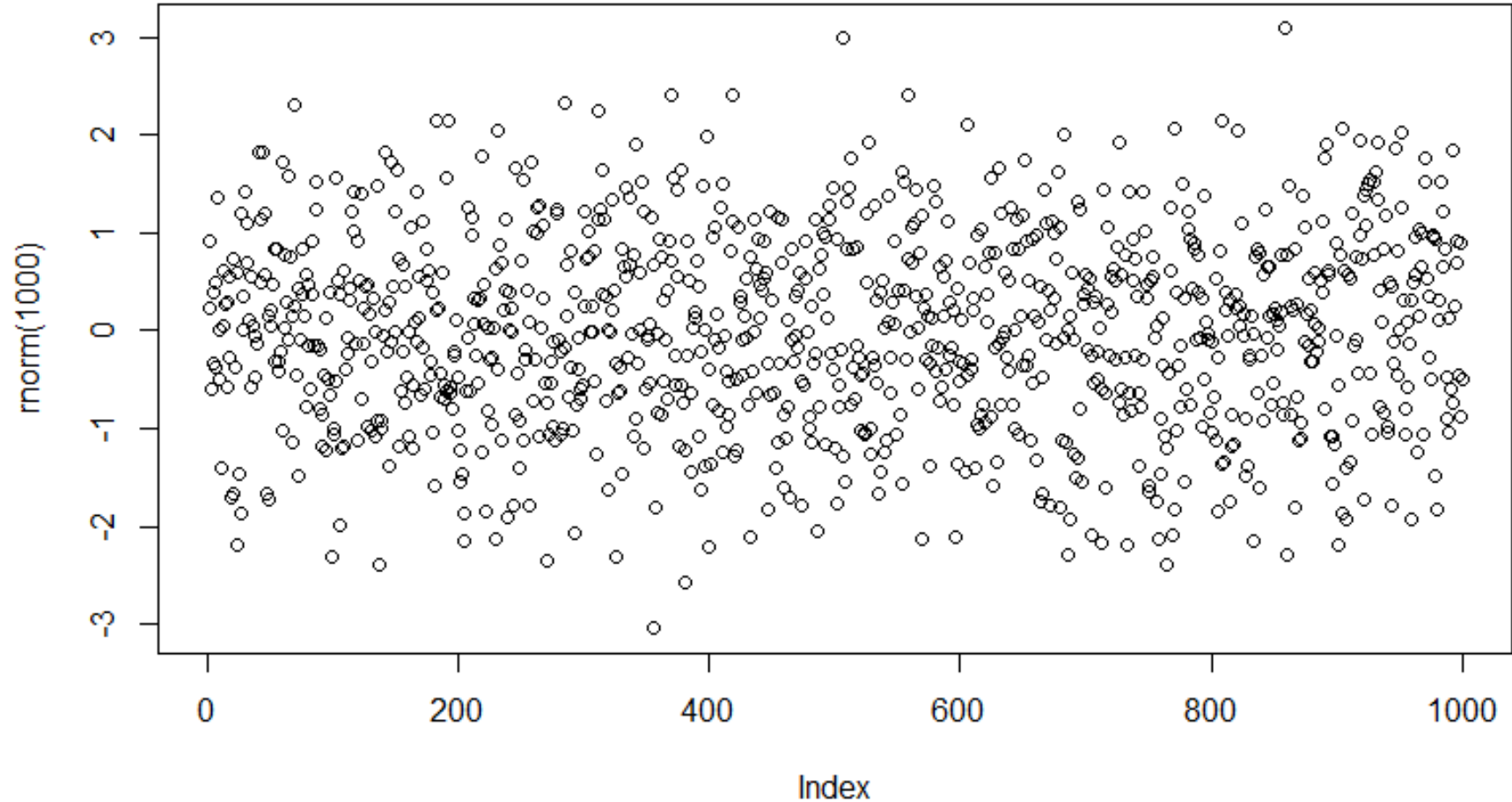
Recap: random numbers with the while loop

- We often use the while-loop when the number of iterations is not known beforehand
- For example, we want to generate 1000 positive random numbers
- We use the `rnorm` function which generates normal distributed random numbers with mean 0

Let's start with generating 1000 normal distributed random numbers and plotting them

```
> plot(rnorm(1000))
```

Recap: random numbers with the while loop



Recap: random numbers with the while loop

```
### generate n positive random numbers
positive.rnd.numbers <- function(n=1000)
{
  i <- 1
  rnd.vec <- vector()
  while(TRUE)
  {
    # generate a single random number
    rnd.number <- rnorm(1)
    # if random number is positive add it to vector rnd.vec
    if(rnd.number > 0)
    {
      rnd.vec <- c(rnd.vec, rnd.number)
      i <- i + 1
    }
    # return rnd.vec after n positive random number
    if(i > n) {return(rnd.vec)}
  }
}
```

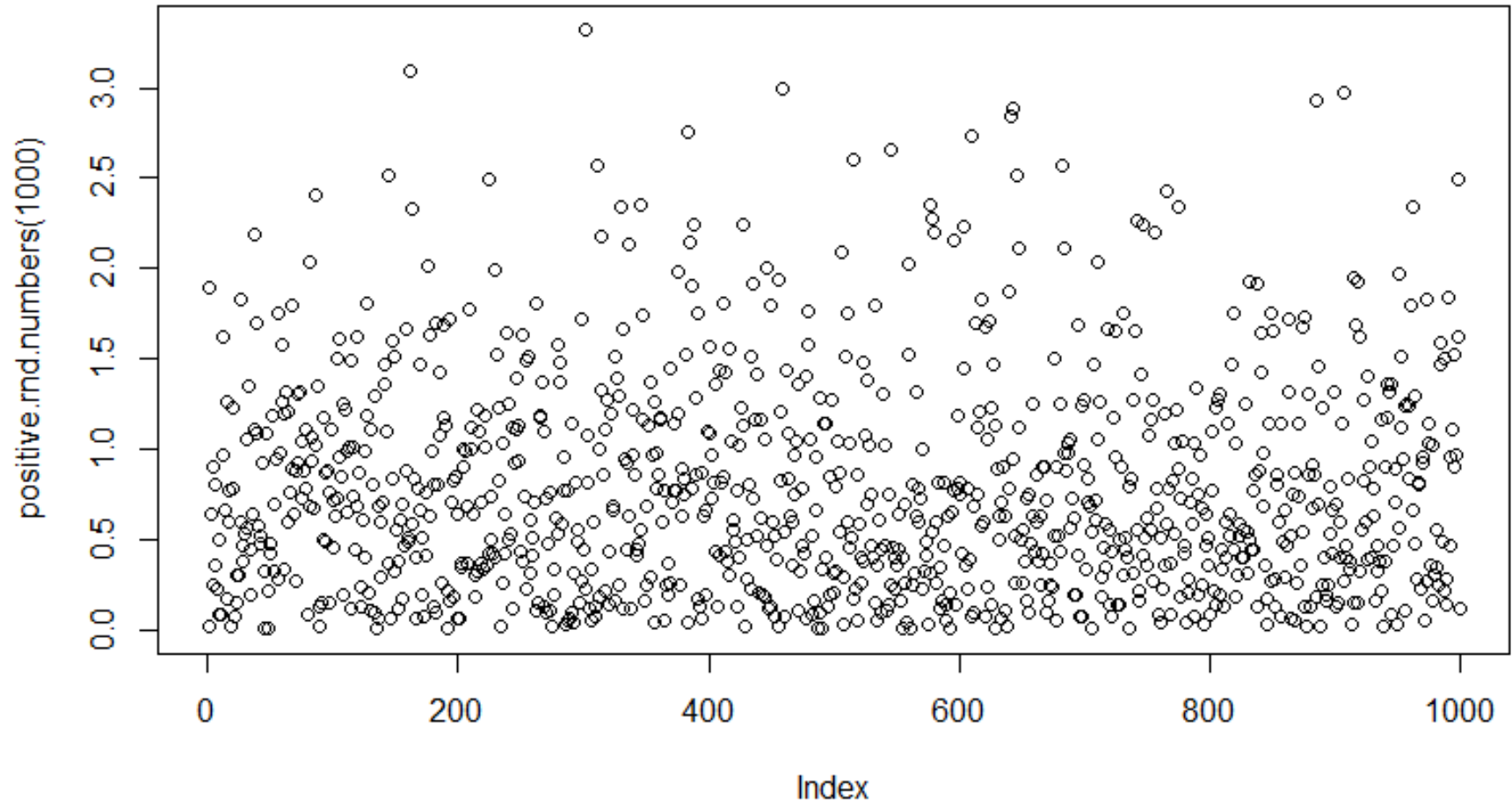
Recap: random numbers with the while loop

- In the function we use `return` to exit the loop since we need the vector as result

Let's test our function by plotting

```
> plot(positive.rnd.numbers(1000))
```

Recap: random numbers with the while loop



Today

- Scatter plot
- Histogram
- Figure array
- Example: Analysis of the lecture with these

Scatter plots of data frames

- Let's come back to our person height and weight example
- So far we created a simple scatter plot by plotting height on the x-axis and weight on the y-axis
- Often we have more than two dimensions and we would like to observe pairwise correlations between them

In our example we have 3 numeric and one Boolean dimension

Scatter plots of data frames

```
> person.data
```

	Name	Height	Weight	BMI	above22.5
1	Can	1.70	65	22.49	FALSE
2	Cem	1.75	66	21.55	FALSE
3	Hande	1.62	61	23.24	TRUE
4	Lale	1.76	64	20.66	FALSE
5	Arda	1.78	63	19.88	FALSE
6	Bilgin	1.77	84	26.81	TRUE
7	Cem	1.69	75	26.26	TRUE
8	Ozlem	1.75	65	21.22	FALSE
9	Ali	1.73	75	25.06	TRUE
10	Haluk	1.71	81	27.70	TRUE

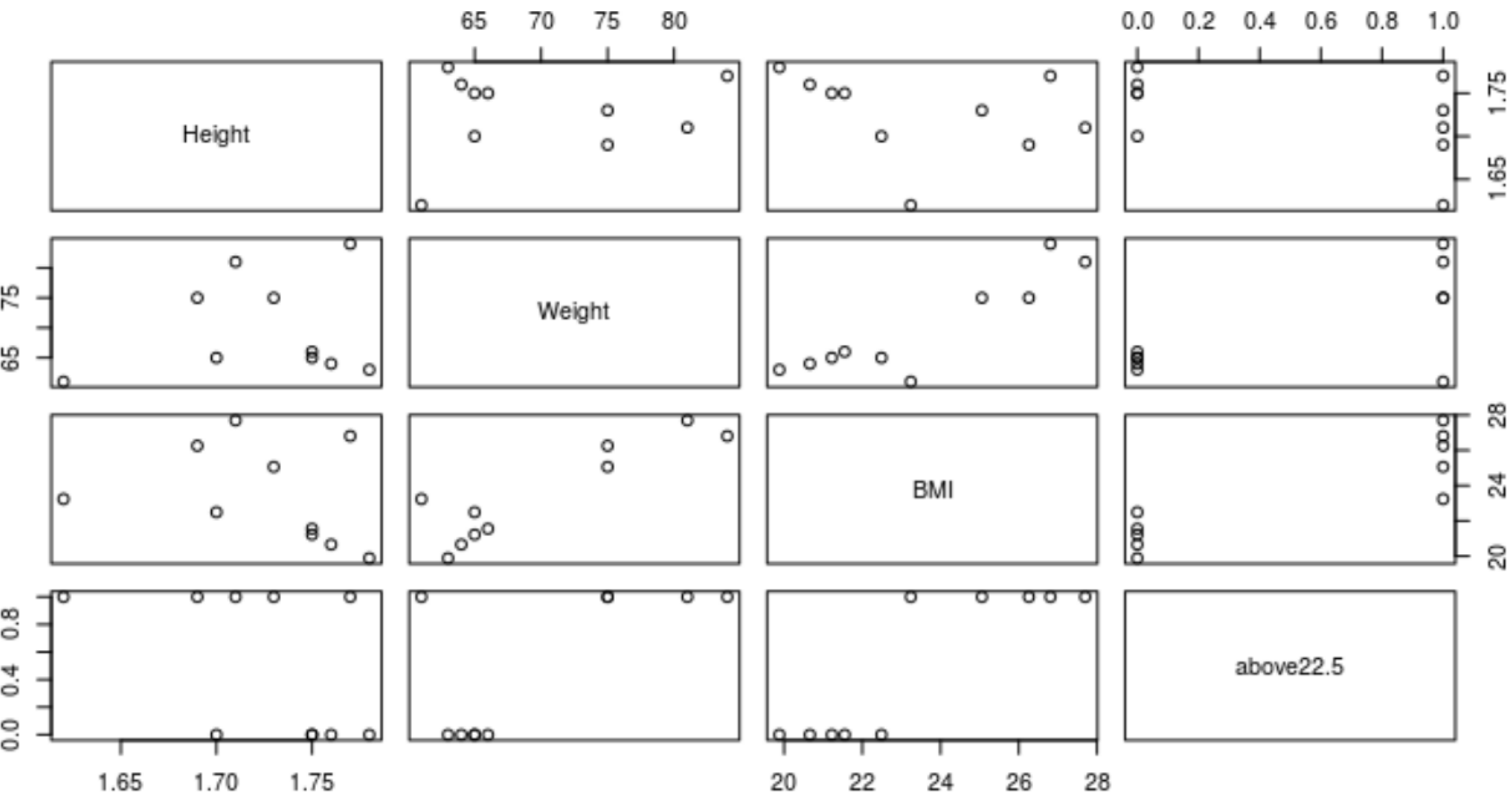
Scatter plots of data frames

- We can plot each dimension of a data frame against each other by providing the data frame as argument to the `plot()` function

In our example, we plot the numeric and Boolean columns 2, 3, 4 and 5

```
> plot(person.data[, 2:5])
```

Scatter plots of data frames



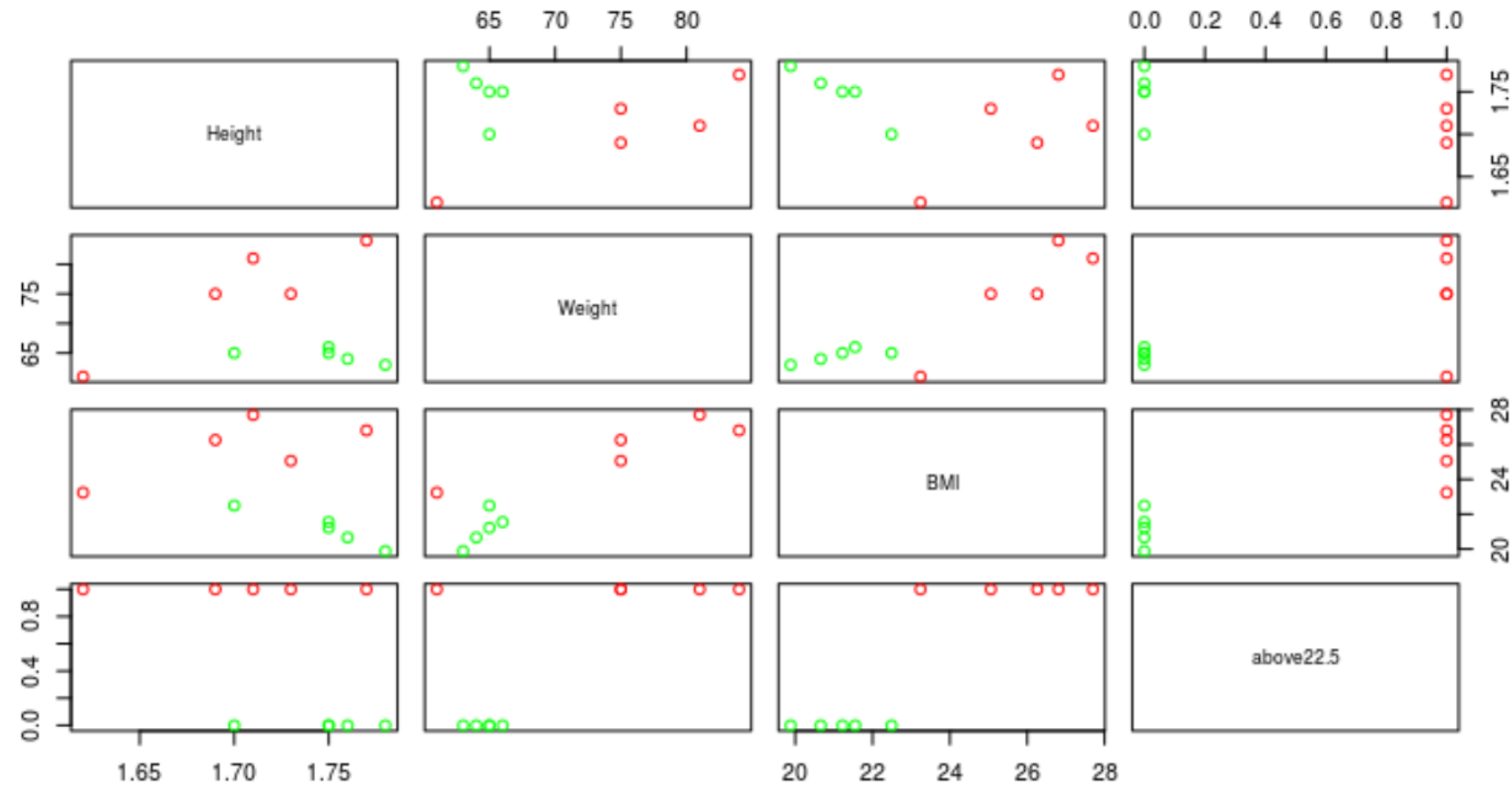
Scatter plots of data frames

- In the previous plot we can observe how the numeric dimensions `Height`, `Weight`, and `BMI` are related to each other
- In order to better observe the relation between the Boolean group variable `above22.5` we use a color coding
- We can specify a color coding with the argument `col`

We use `ifelse` to specify color red for `above22.5=TRUE` and color green otherwise

```
> plot(person.data[,2:5],  
col=ifelse(person.data$above22.5, "red", "green"))
```

Scatter plots of data frames



Scatter plots of data frames

```
help("plot")  
plot(x, y, ...)  
Arguments
```

```
...
```

Many methods will accept the following arguments:

`type`

```
"p" for points,  
"l" for lines,  
"b" for both,  
"c" for the lines part alone of "b",  
"o" for both 'overplotted',  
"h" for 'histogram' like (or 'high-density') vertical  
lines,  
"s" for stair steps,  
"S" for other steps, see 'Details' below,  
"n" for no plotting.
```

Scatter plots of data frames

```
help("plot")  
plot(x, y, ...)  
Arguments  
...
```

main

an overall title for the plot: see title.

sub

a sub title for the plot: see title.

xlab

a title for the x axis: see title.

ylab

a title for the y axis: see title.

asp

the y/x aspect ratio, see plot.window.

Histogram

- Previously we have seen two types of plotting
 - Plot raw data like person's height/weight or random numbers
 - Plot number of observations in the word list barplot
- A histogram is similar to a barplot since we visualize how many observations fall within specified divisions called “bins”
- In the word list barplot the bins were given by the words itself
- In a histogram we usually have to create the bins

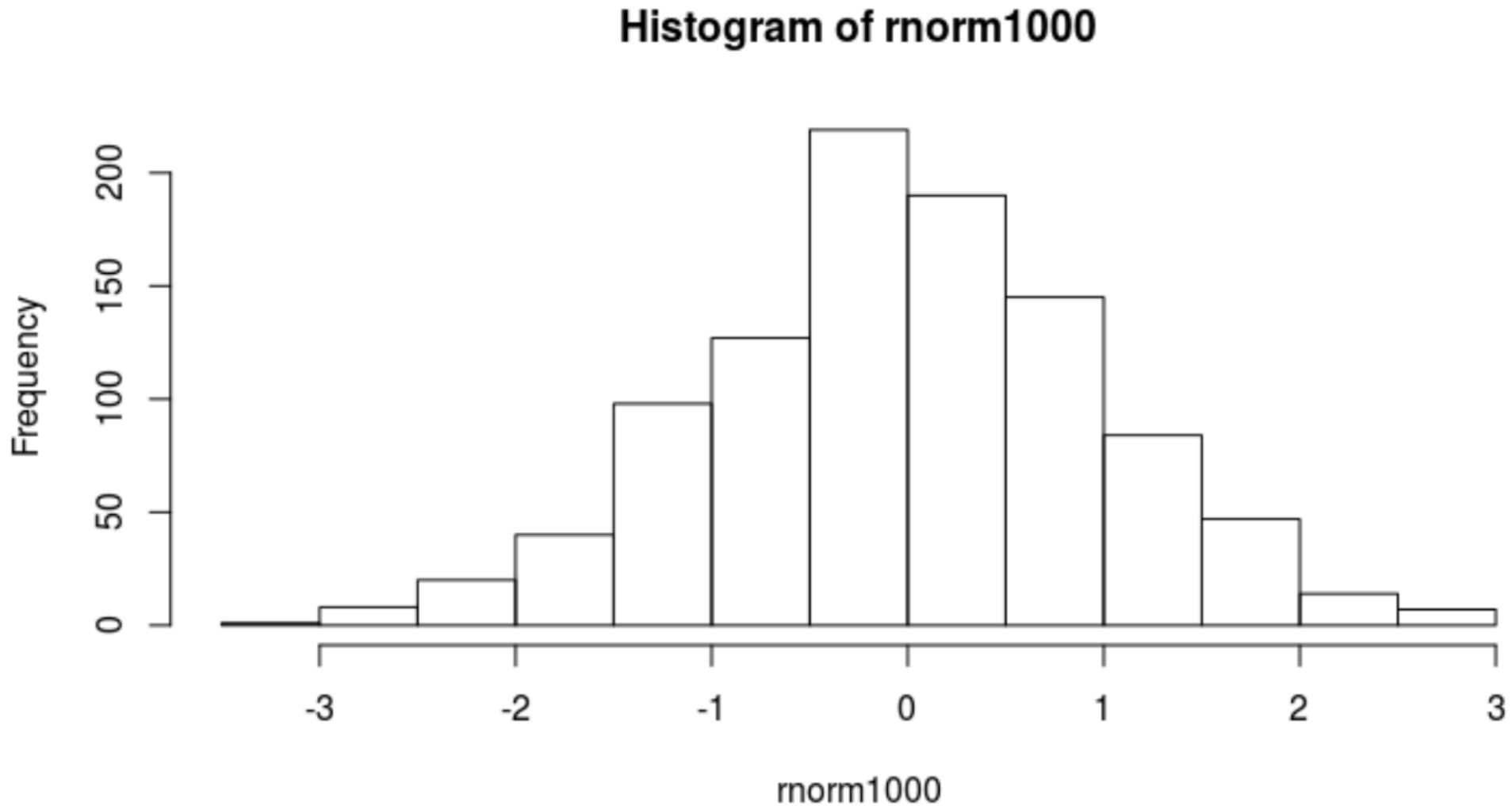
Histogram

- Let's create a histogram of normal distributed random numbers with mean 0
- In R we simply call the `hist` function

`hist` creates the bins automatically, counts the number of observations that fall within the bins and plots the result

```
> rnorm1000 <- rnorm(1000)
> hist(rnorm1000)
```

Histogram



Histogram

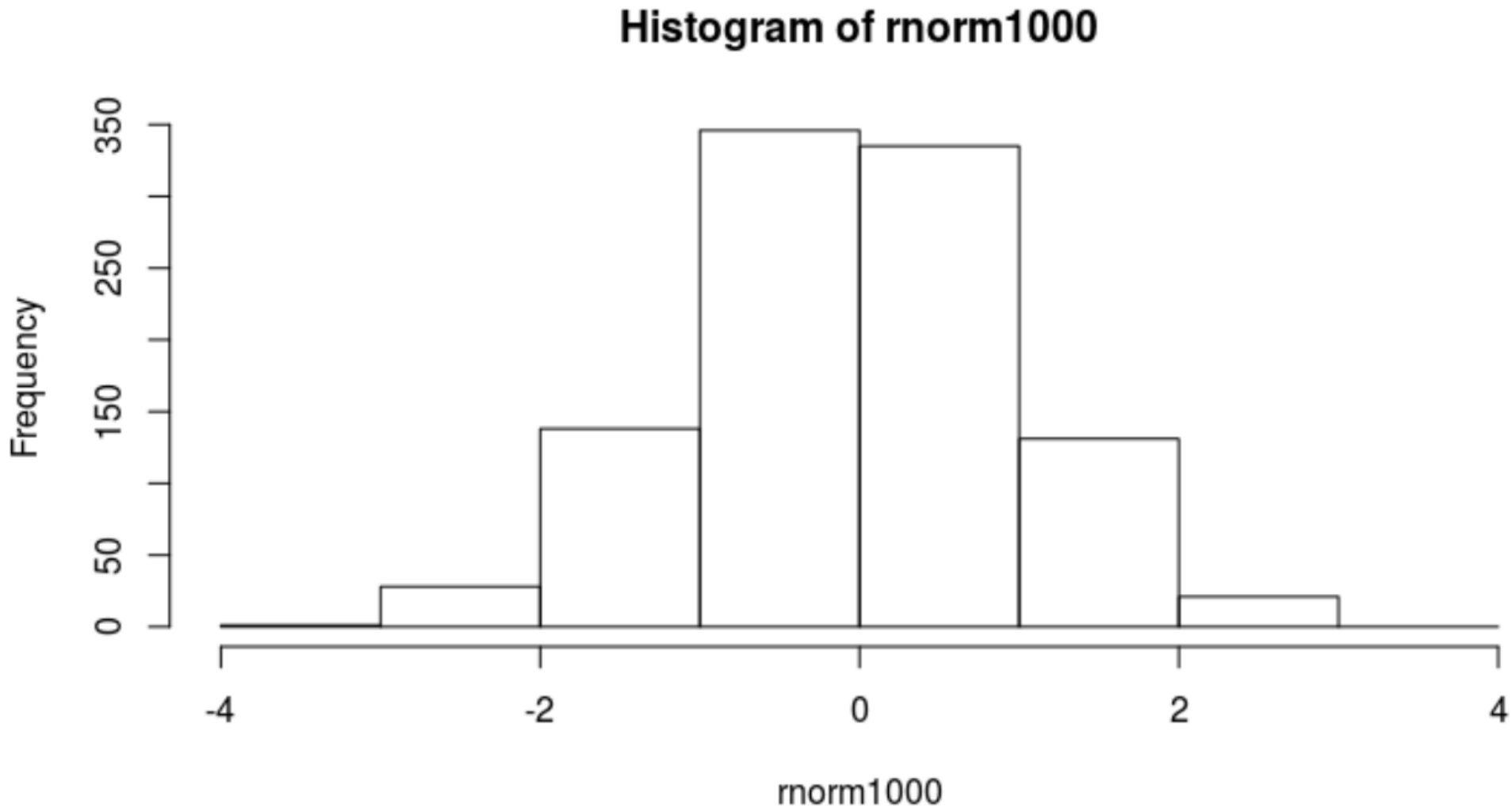
- In the previous plot we observe that bins of width 0.5 were automatically created
- Important to note that the number of bins influences the appearance of a histogram

We can modify how the bins are created by providing a vector `breaks`

- Let's try broader and smaller bins

```
> hist(rnorm1000, breaks=seq(-4, 4, 1))  
> hist(rnorm1000, breaks=seq(-4, 4, 0.25))
```

Histogram



Histogram



Histogram

- Let's now create a histogram of positive random numbers

We use our function `positive.rnd.numbers` from previous lecture to create positive random numbers

```
> hist(positive.rnd.numbers())
```

Histogram

Histogram of positive.rnd.numbers()

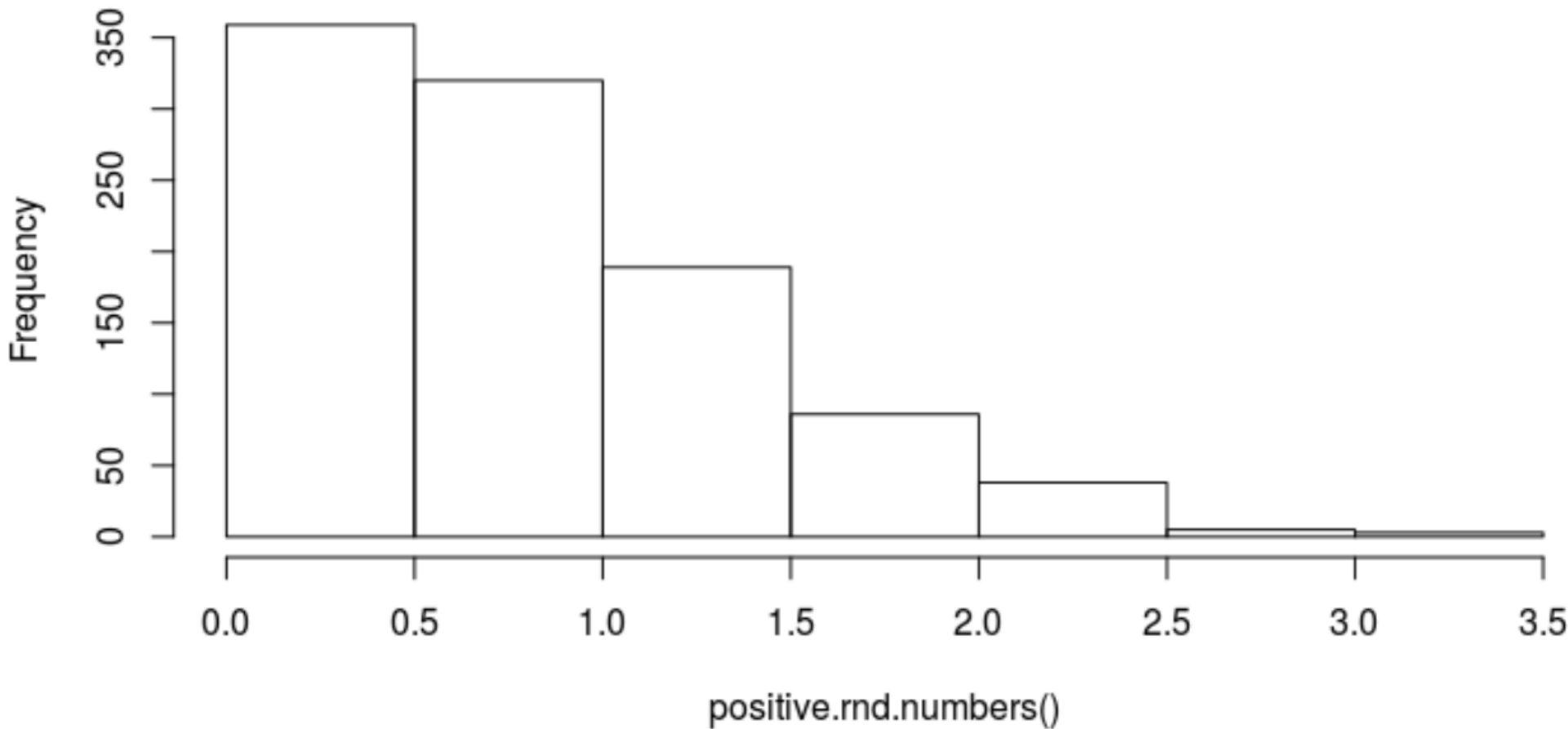


Figure array

- Sometimes we need to place several figures in the same plot
- For example, we would like to have the plot of the normal distributed random numbers and the corresponding histogram in one plot
- We use `par` to specify how several figures will be drawn in an number of rows by number of columns array
 - `par(mfrow=c(nr, nc))` specifies that figures will be drawn in an `nr`-by-`nc` array **by rows**
 - `par(mfcol=c(nr, nc))` specifies that figures will be drawn in an `nr`-by-`nc` array **by columns**

Figure array

We place the plot of the normal distributed random numbers and the corresponding histogram side by side

```
> rnorm1000 <- rnorm(1000)
> par(mfrow=c(1,2))
> plot(rnorm1000)
> hist(rnorm1000)
> par(mfrow=c(1,1))
```

- The last line resets `mfrow` to its default value

Figure array

We place the plot of the normal distributed random numbers and the corresponding histogram side by side

```
> rnorm1000 <- rnorm(1000)
> par(mfrow=c(1,2))
> plot(rnorm1000)
>
```

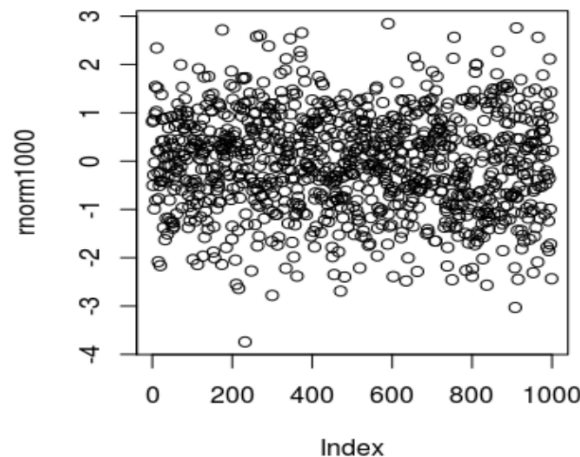


Figure array

We place the plot of the normal distributed random numbers and the corresponding histogram side by side

```
> rnorm1000 <- rnorm(1000)
> par(mfrow=c(1,2))
> plot(rnorm1000)
> hist(rnorm1000)
```

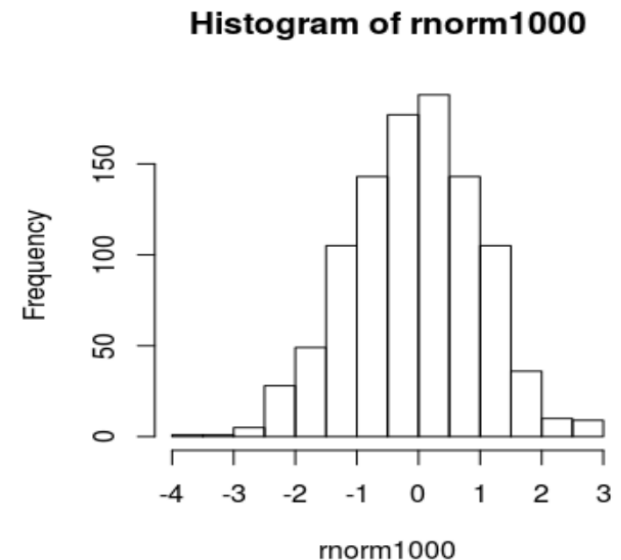
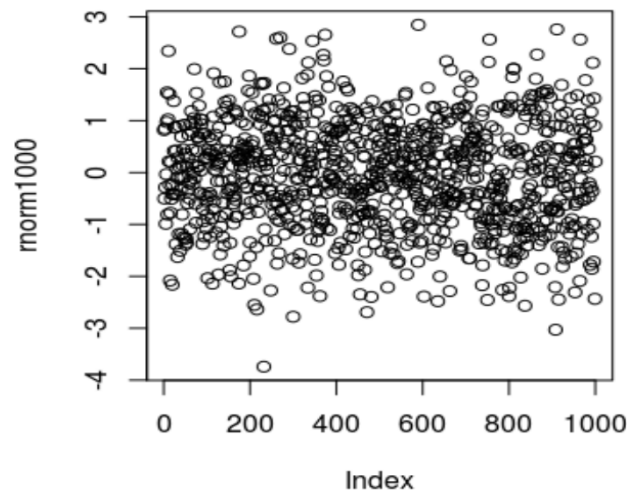
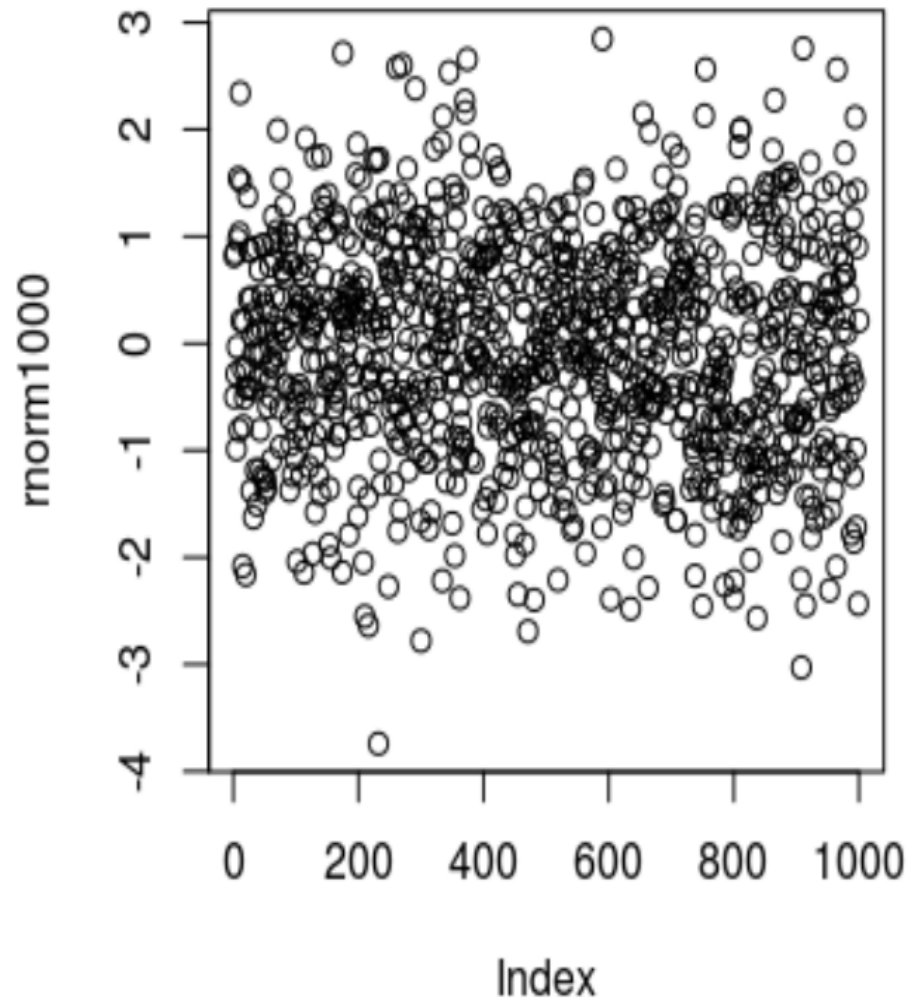


Figure array



Histogram of rnorm1000

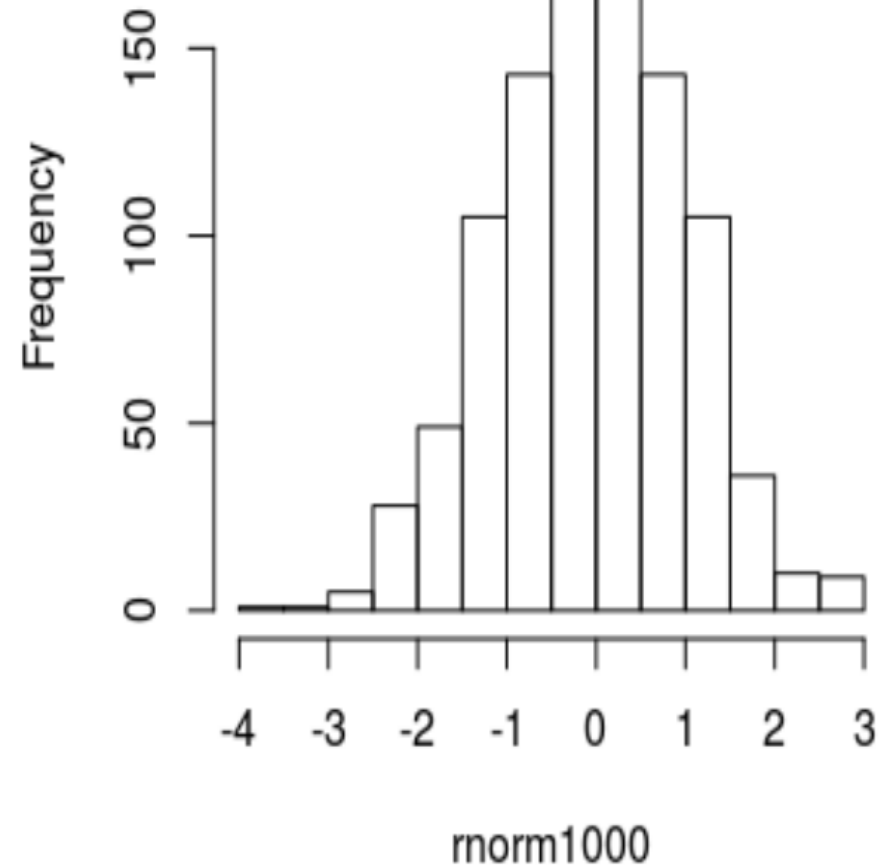


Figure array

We place the plot of the normal distributed random numbers and the corresponding histogram side by side

```
> rnorm1000 <- rnorm(1000)
> par(mfrow=c(1,2))
> plot(rnorm1000)
> hist(rnorm1000)
> par(mfrow=c(1,1))
```

- The last line resets `mfrow` to its default value

Figure array

In a 2x2 figure array we plot normal distributed random numbers, positive random numbers and their corresponding histograms

```
> rnorm1000 <- rnorm(1000)
> pos.rnorm1000 <- positive.rnd.numbers(1000)
> par(mfrow=c(2,2))
> plot(rnorm1000)
> hist(rnorm1000)
> plot(pos.rnorm1000)
> hist(pos.rnorm1000)
> par(mfrow=c(1,1))
```

Figure array

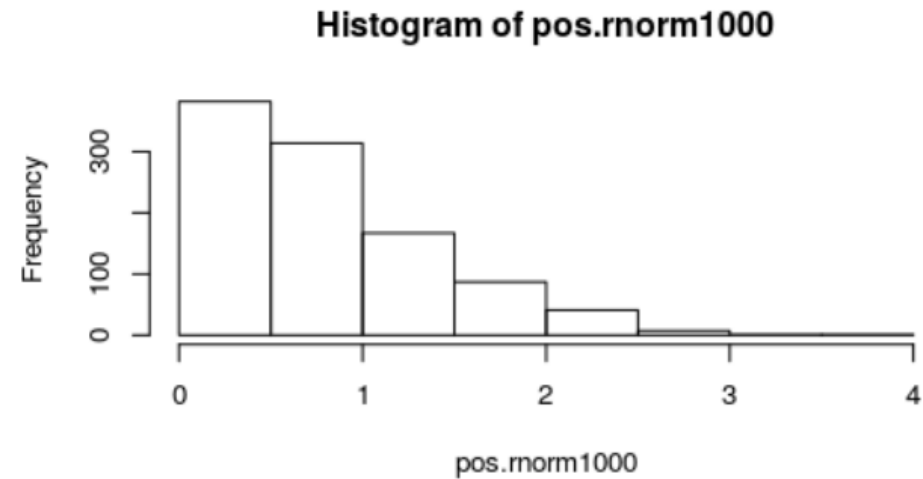
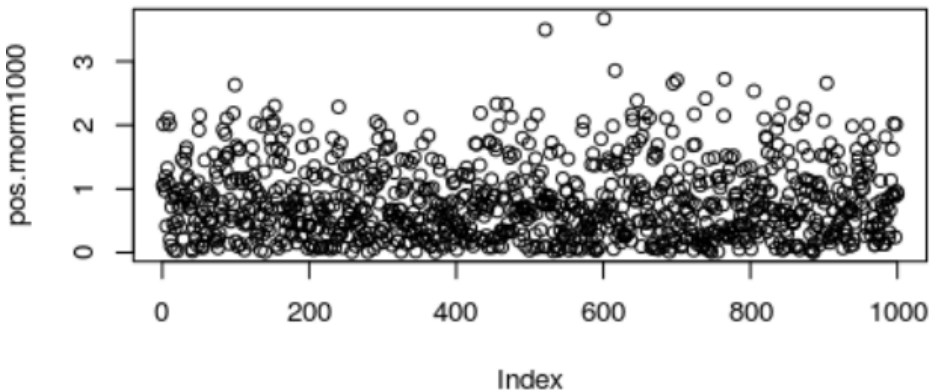
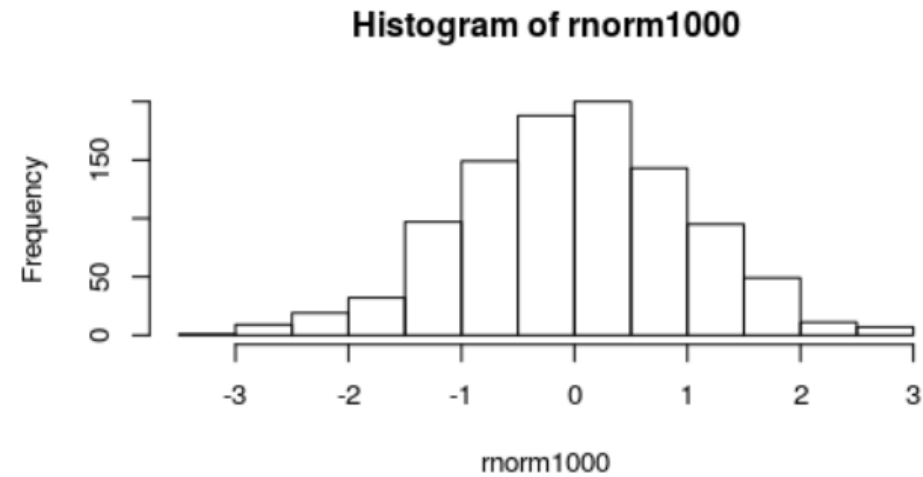
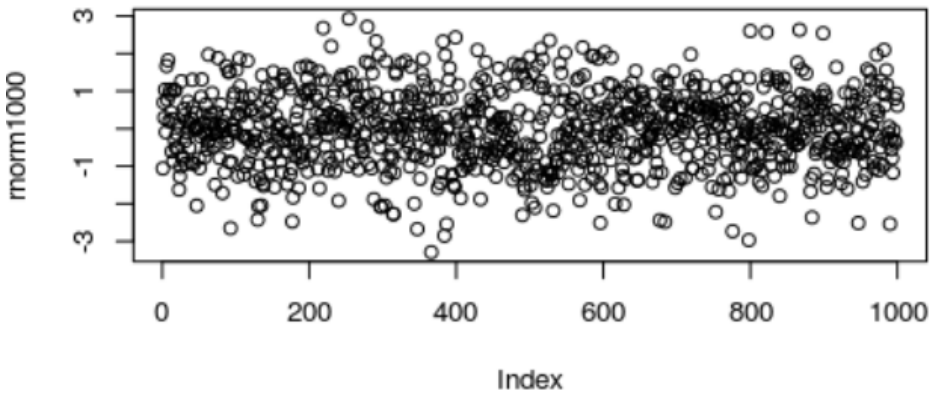


Figure array

- In the previous plot we might have difficulties to compare both distributions because axis limits and bins are chosen automatically

In order to avoid confusion, we better enforce identical axis limits and bins for both groups

```
> par(mfrow=c(2,2))  
> plot(rnorm1000, ylim=c(-4,4))  
> hist(rnorm1000, breaks=seq(-4,4,1))  
> plot(pos.rnorm1000, ylim=c(-4,4))  
> hist(pos.rnorm1000, breaks=seq(-4,4,1))  
> par(mfrow=c(1,1))
```

Figure array

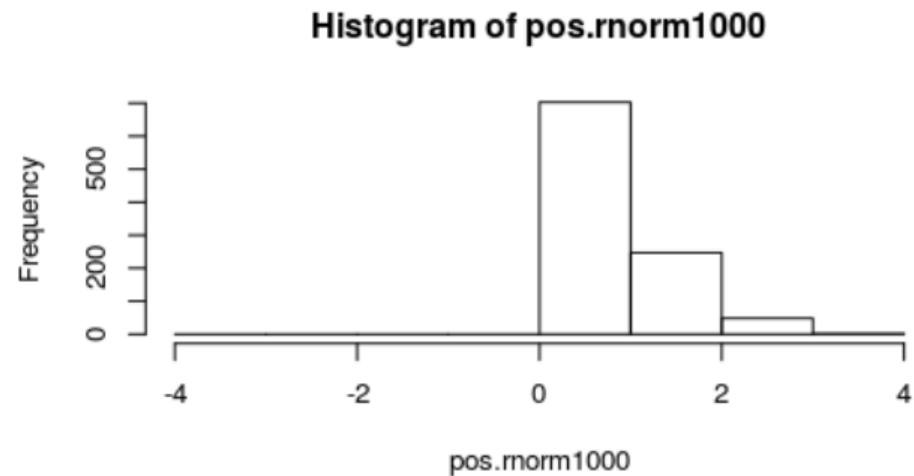
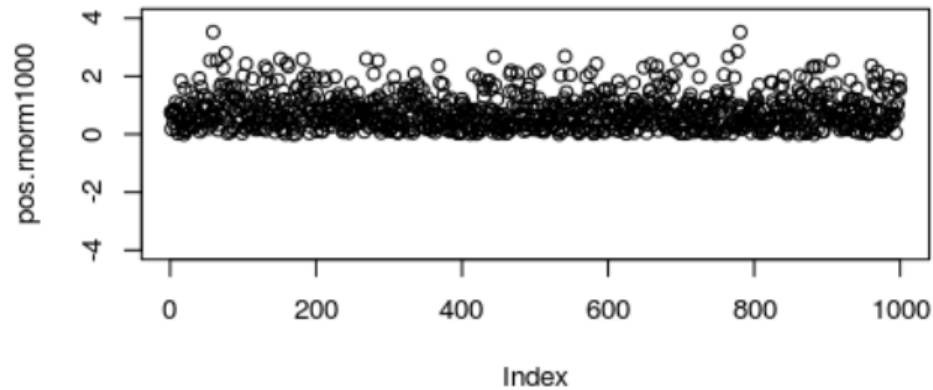
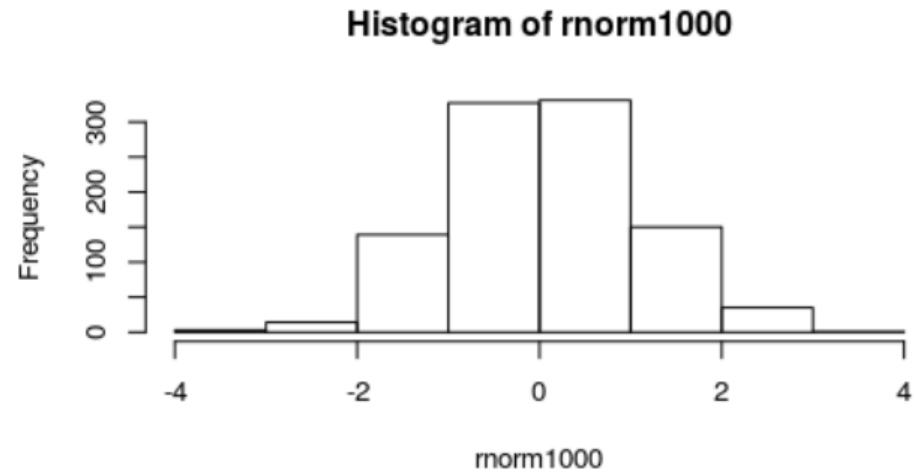
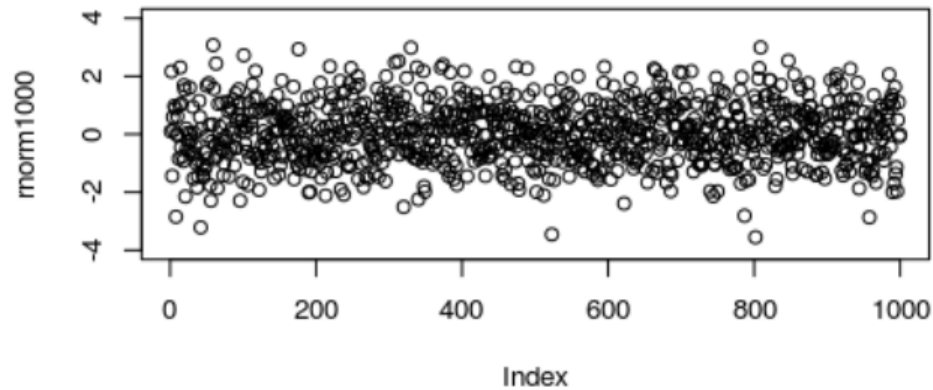


Figure array

We place the scatter plot of person's weight and height beside a histogram of their BMI

```
> par(mfrow=c(1,2))  
> plot(person.data$Height, person.data$Weight, pch=2)  
> lines(c(1.62, 1.78), 22.5*c(1.62, 1.78)^2)  
> hist(person.data$BMI, breaks=seq(18.5, 28.5, 2))  
> par(mfrow=c(1,1))
```

Figure array

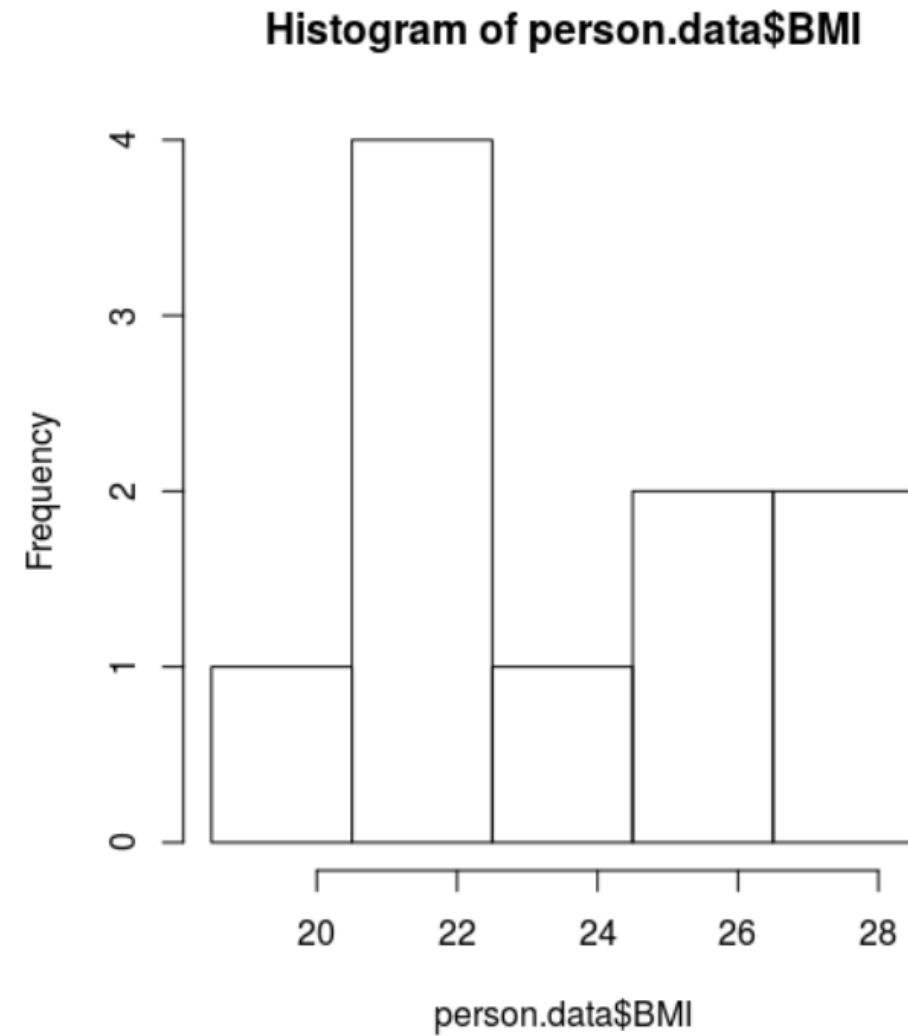
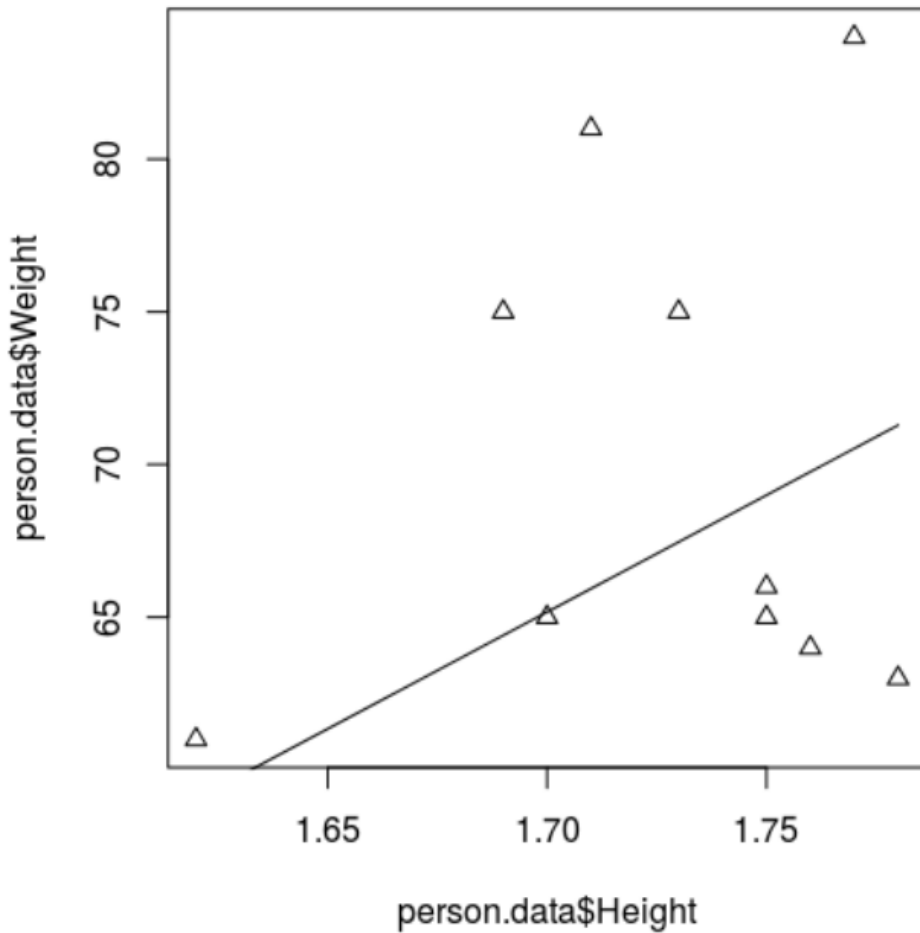


Figure array exercise: mid-semester analysis of grades



```
# save google doc as csv file
grades <- read.table("grades.csv", sep=",", header=T)
grades <- grades[1:length(grades$id)-1,]
summary(grades)
plot(grades[,c(6,7,10)], col=ifelse(grades$Attendance>80,
"red", "green"))
```

Figure array exercise: mid-semester analysis of grades

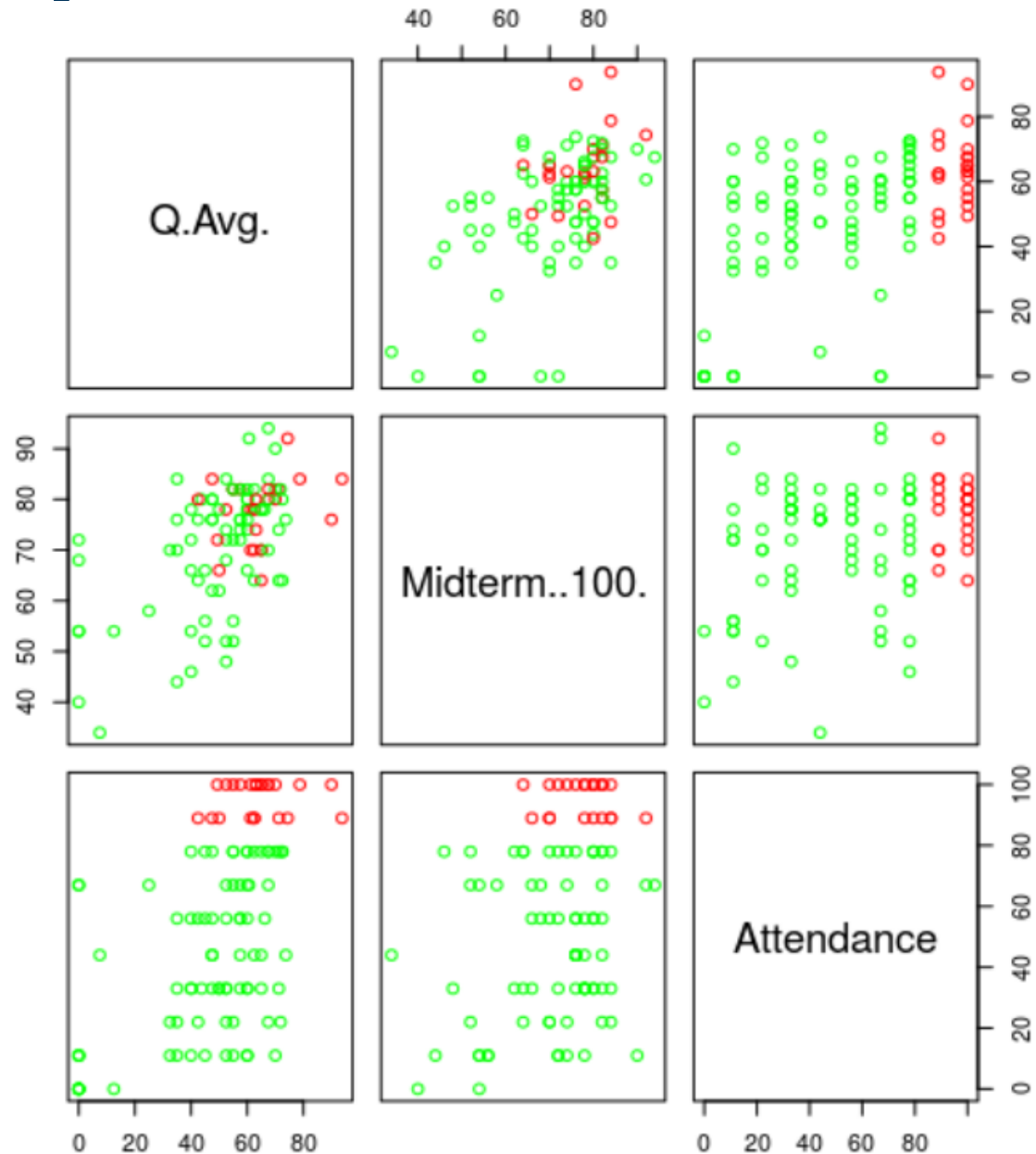


Figure array exercise: mid-semester analysis of grades



```
# Check out the grades for people with attendance > X%
par(mfrow=c(1,1))
plot.new()
par(mfrow=c(2,2))
plot(grades$Q4.20., col=ifelse(grades$Attendance.last>=50
,"red","green"), main='last-attendance>50%')
plot(grades$Q4.20., col=ifelse(grades$Attendance>=50,"red
","green"), main='all-attendance>50%')
plot(grades$Q.Avg., col=ifelse(grades$Attendance.last>=70
,"red","green"), main='last-attendance>70%')
plot(grades$Q.Avg., col=ifelse(grades$Attendance>=70,"red
","green"), main='all-attendance>70%')
```

Figure array exercise: mid-semester analysis of grades

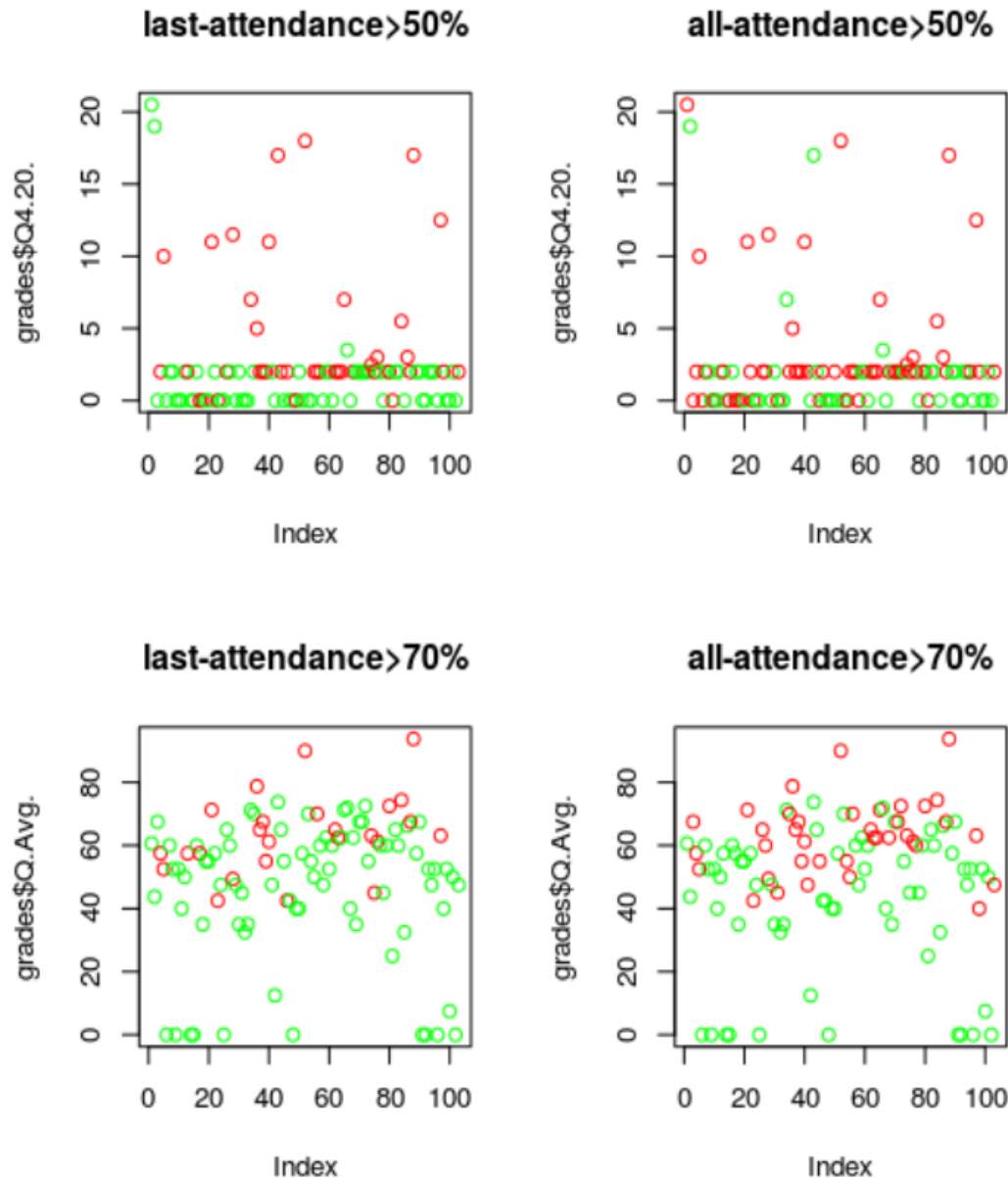


Figure array exercise: mid-semester analysis of grades

```
# for each attendance level, plot mean grade of quiz and
# midterm
par(mfrow=c(1,2))
for (column in c(6:7)) {
  indices <- c()
  means<-c()
  for (x in seq(0,100,10)) {
    m = mean(grades[grades$Attendance>=x,column])
    means <- c(means,m)
    indices <- c(indices,x)
  }
  plot(indices,means,main=colnames(grades)[column],
        xlab="% of attendance")
}
```

Figure array exercise: mid-semester analysis of grades

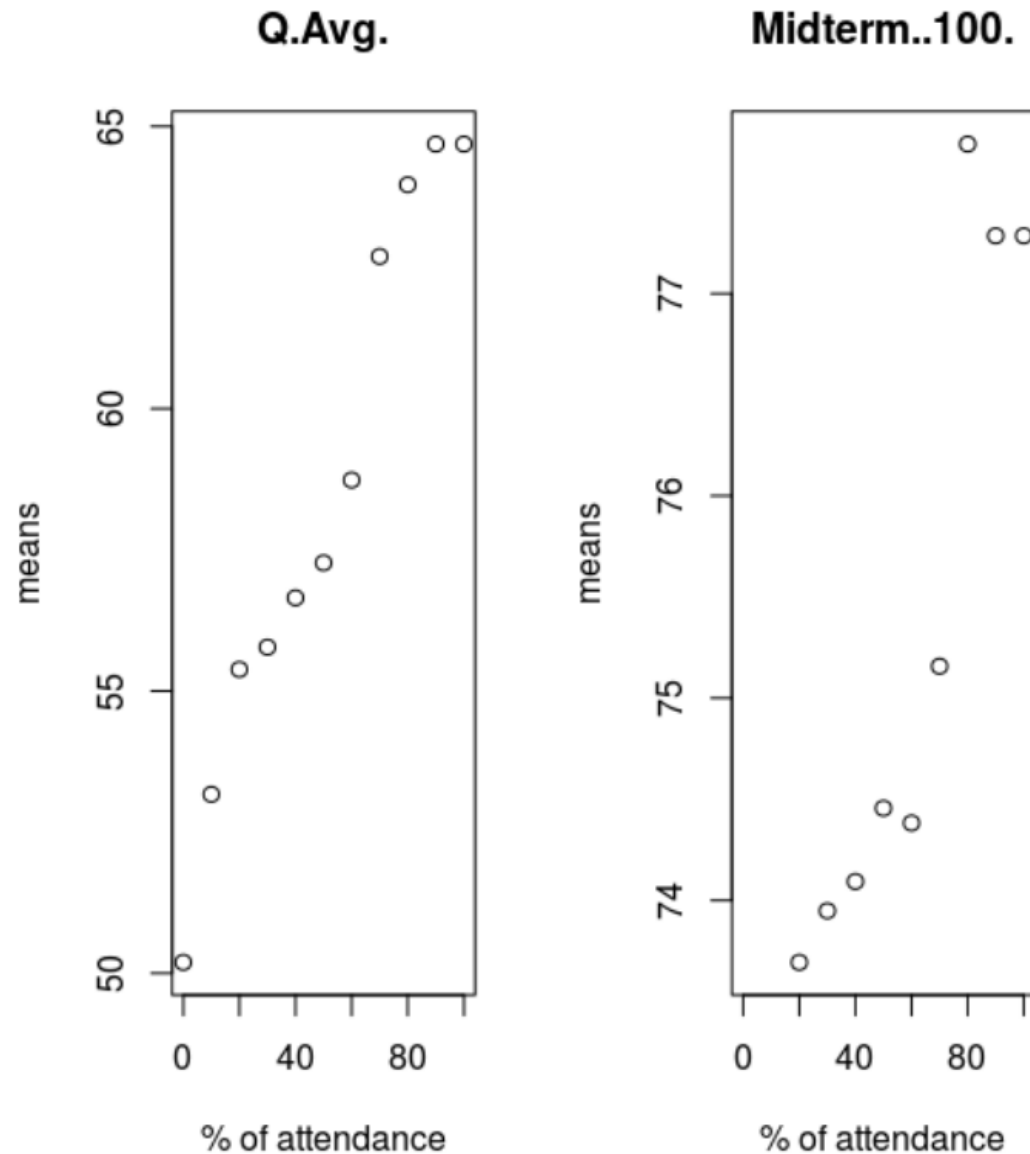


Figure array exercise: mid-semester analysis of grades

```
# for each attendance level, plot mean grade of quiz and
# midterm
par(mfrow=c(1,2))
for (column in c(1:length(grades))) {
  cNames=colnames(grades)
  if (cNames[column]=="Q.Avg." || # or
      cNames[column]=="Midterm..100." ) {
    means<-c()
    indices <- c()
    for (x in seq(0,100,10)) {
      m = mean(grades[grades$Attendance>=x,column])
      means <- c(means,m)
      indices <- c(indices,x)
    }
    plot(indices,means,main=colnames(grades)
          [column],xlab="% of attendance")
  }
}
```

Figure array exercise: mid-semester analysis of grades

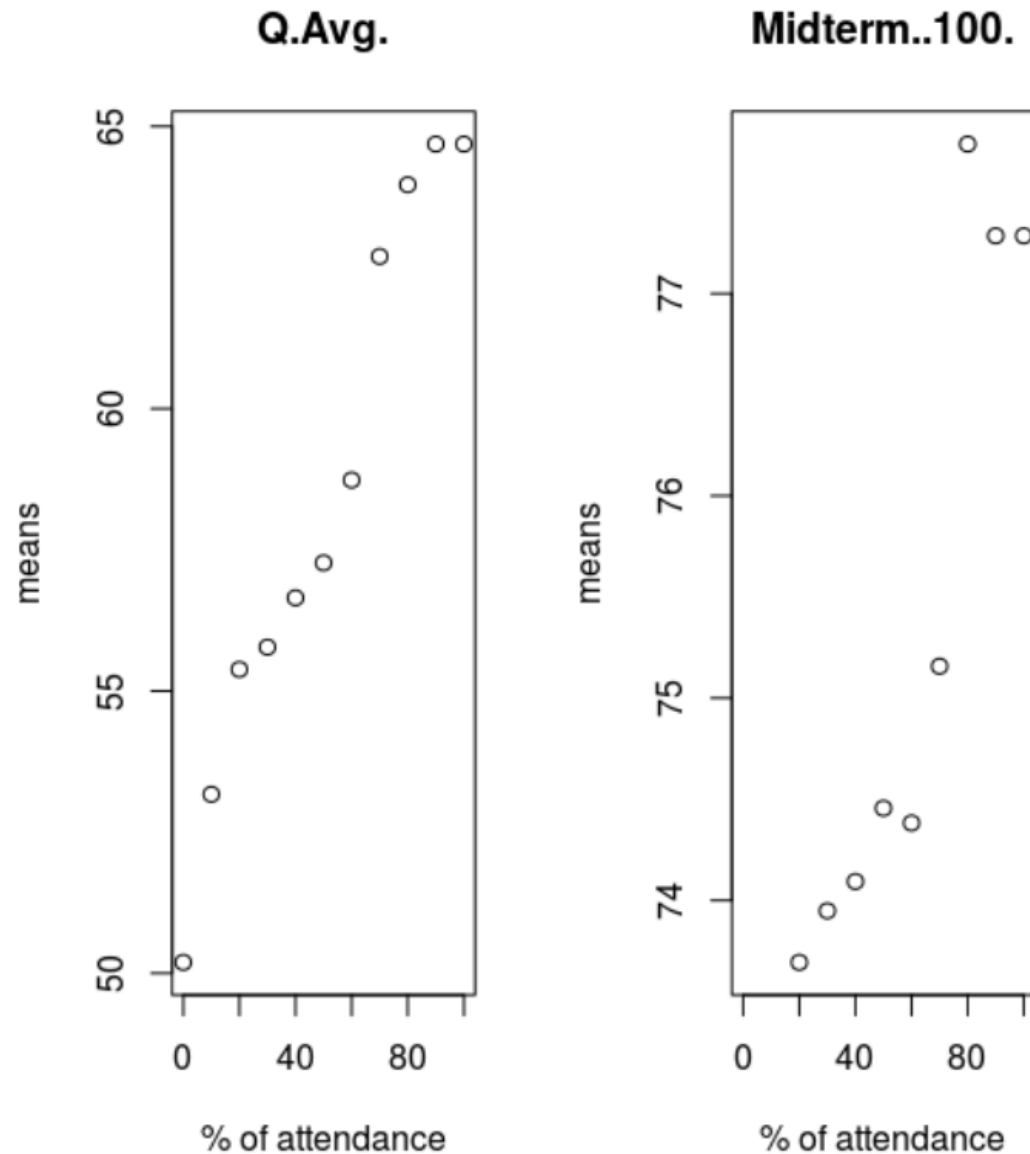
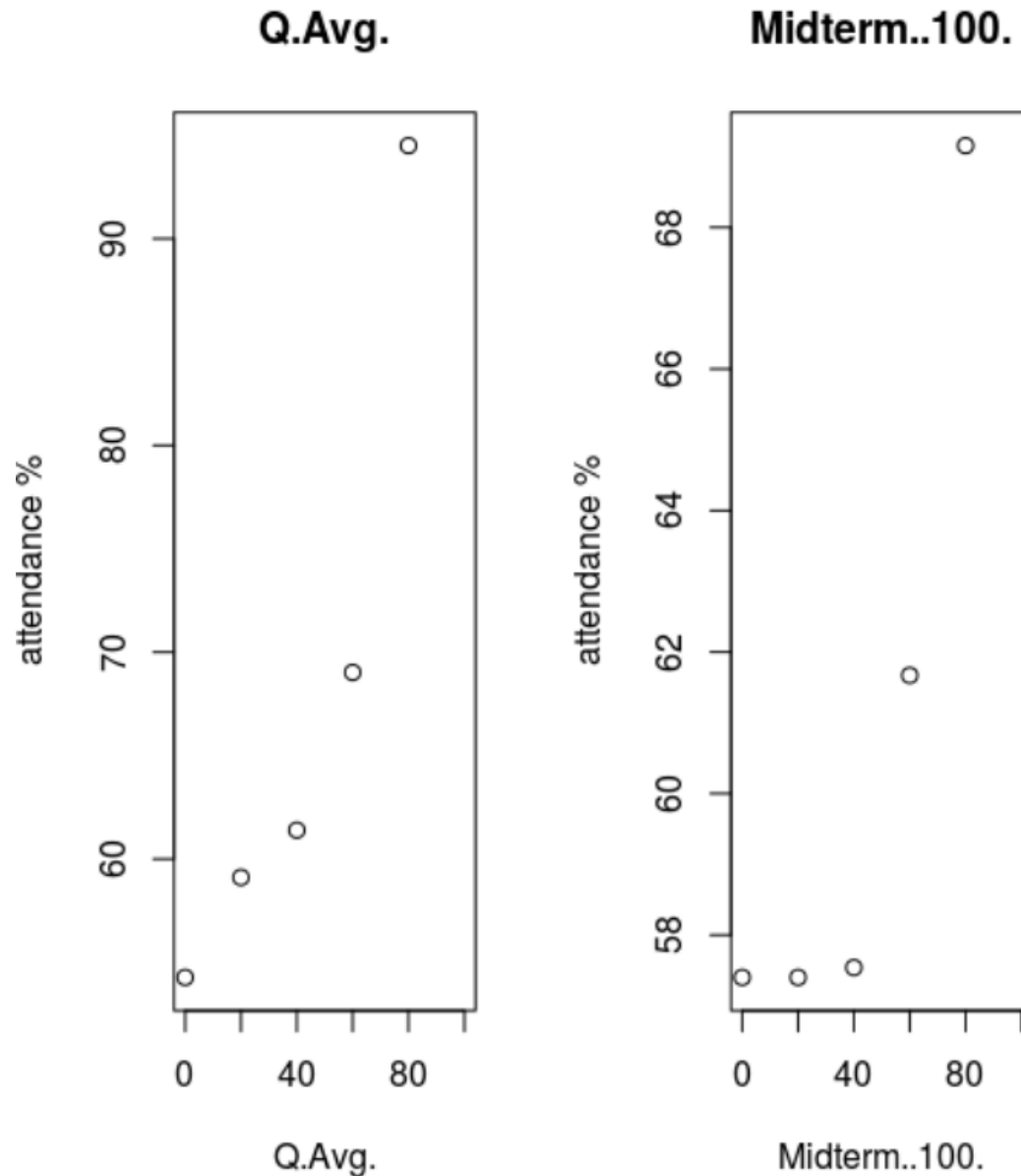


Figure array exercise: mid-semester analysis of grades



```
# for each grade level (0%, 20%, etc), plot attendance
plot.new()
par(mfrow=c(1,2))
for (column in c(6:7)) { #cols for quiz and midterm
  means<-c()
  indices <- c()
  for (x in seq(0,100,20)) { #grade levels
    m = mean(grades$Attendance[grades[,column]>=x],
             na.rm=TRUE)
    means <- c(means,m)
    indices <- c(indices,x)
  }
  plot(indices,means,main=colnames(grades)[column],
        xlab=colnames(grades[column]),ylab='attendance %')
}
```

Figure array exercise: mid-semester analysis of grades



Homework



1. In previous homework, you have collected body height and weight from 10 of your friends. Now, please get the data collections from 2 other students of this course. Create a data frame which consists of your data collection and the data collections from the two other students.
2. Create a scatter plot of the data frame.
3. Plot histograms of body height, body weight and BMI.
4. Create a figure array and place a scatter plot of weight and height beside a histogram of the BMI.
5. Repeat the final analysis yourself.