# Introduction to Computing for Economics and Management

Lecture 10: Input and Output

# Previous lectures

- In previous lectures we already used several input and output operations, e.g.
  - Import data from a file into a data frame using `read.table()`
  - Read a text from a file using `scan()`
  - Get user input from command line to play a quiz using `readline()`
  - Graphics: scatter plot, bar plot, histogram, figure arrays, boxplot, stripcharts, pie charts

- We will first recapitulate how we previously used the input and output operations

- Next, we learn more about input and output operations

# 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,
```

# 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

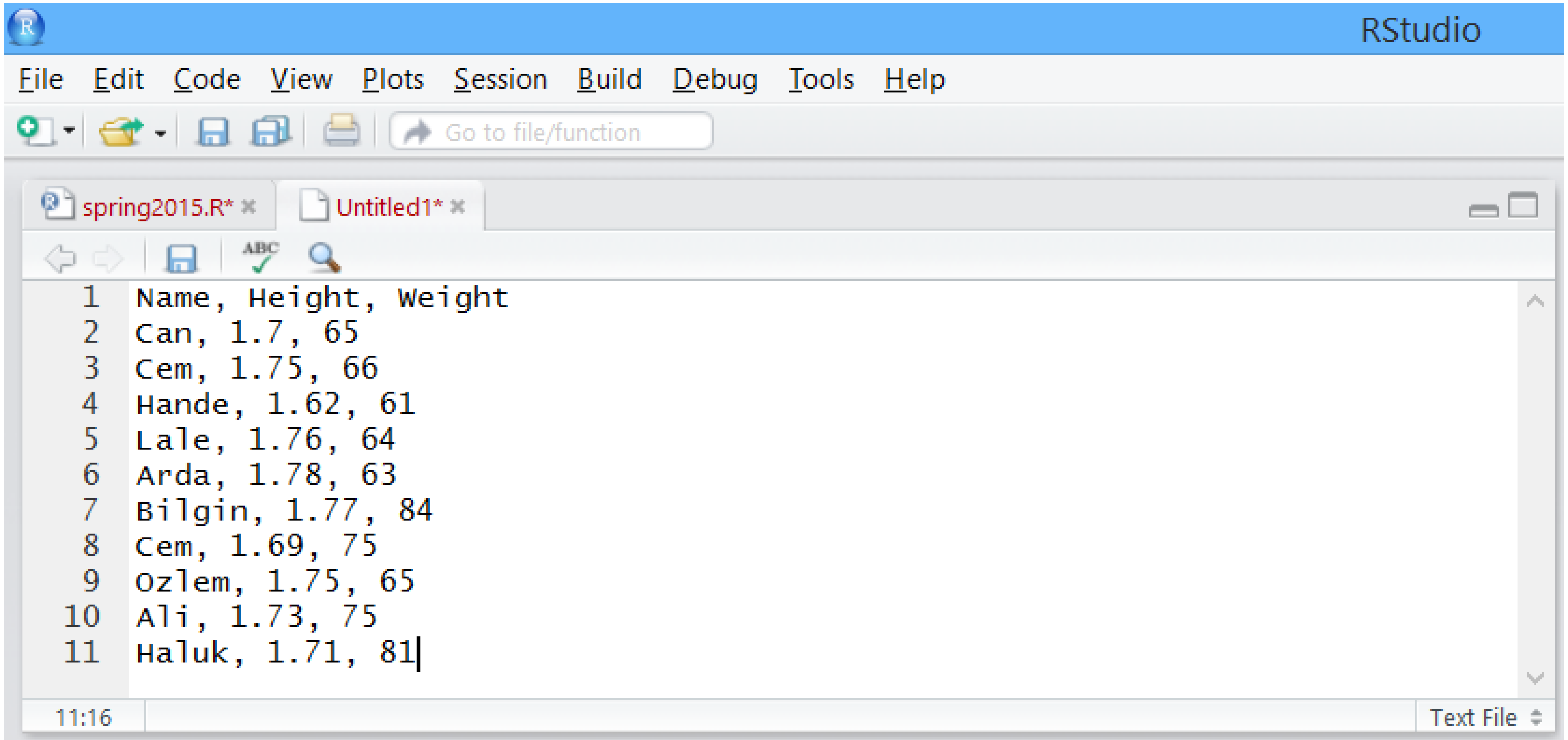
- If header information is available in the file, it will be used for variable names

# 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: data file




- 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: working directory

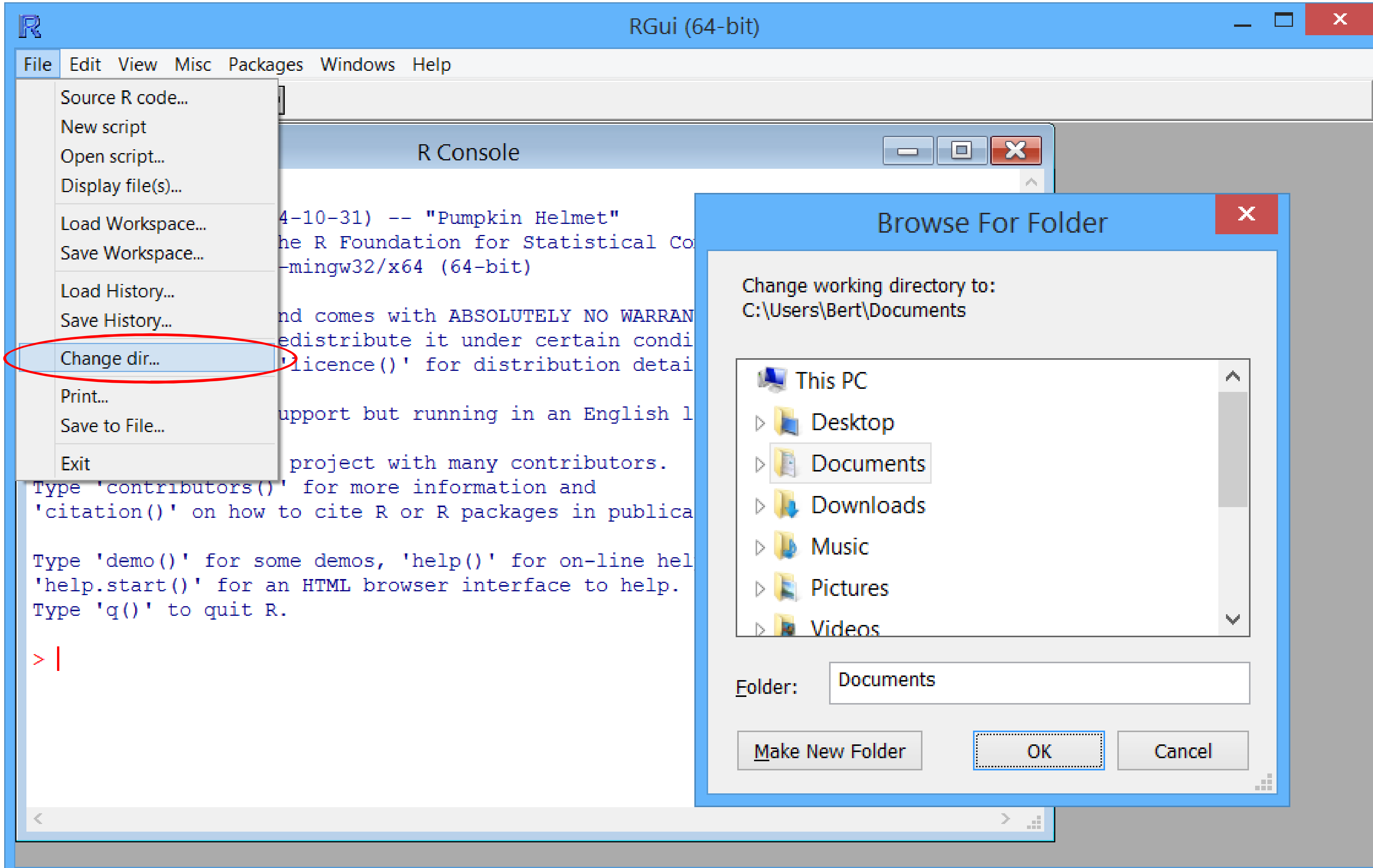
Before the actual import, we need to check the current working directory to make sure which path to use when importing the data file

```
> getwd()
```

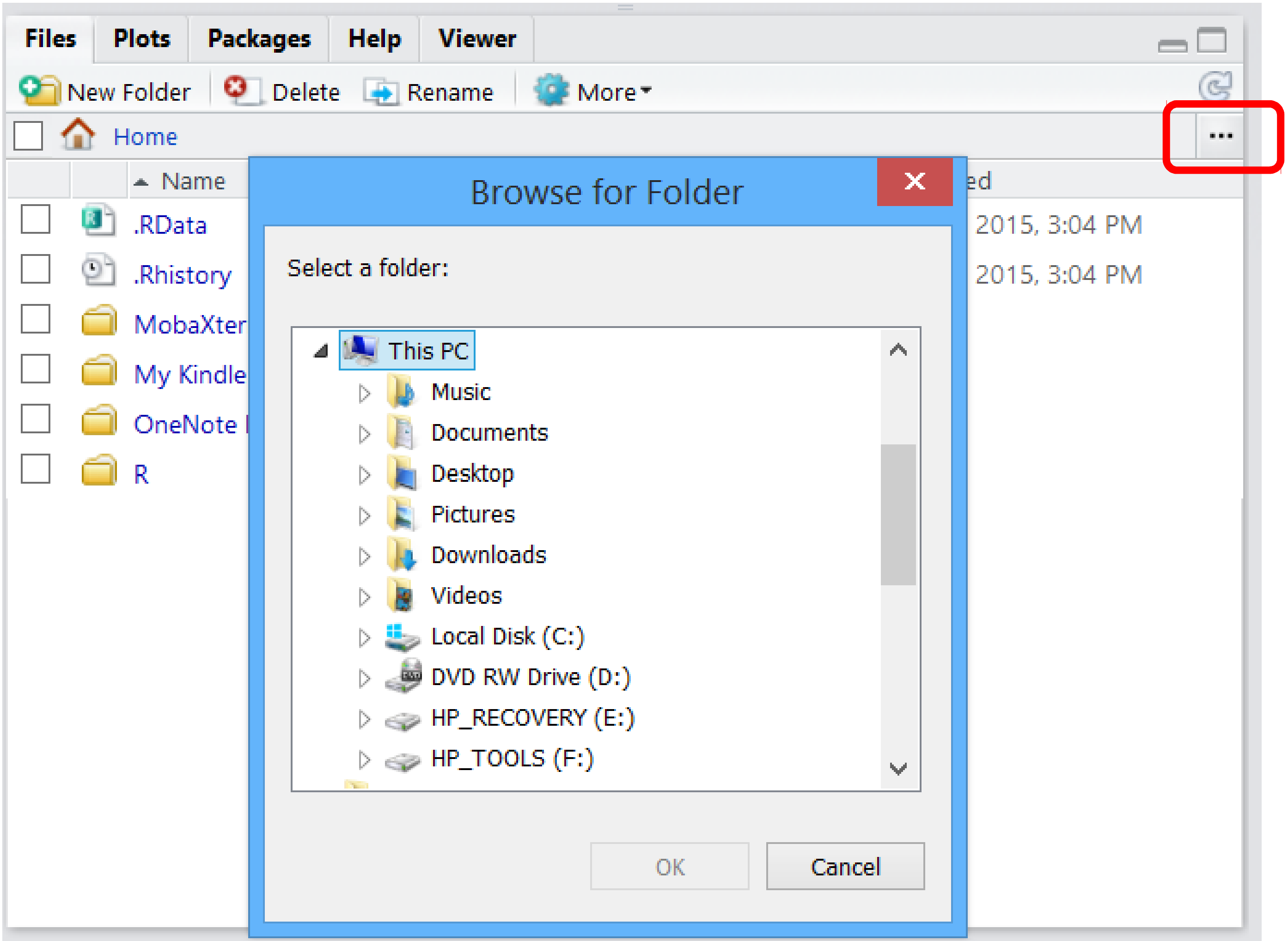
- `[1] "C:/Users/Bert/Documents"`

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

# Recap: change working directory in R

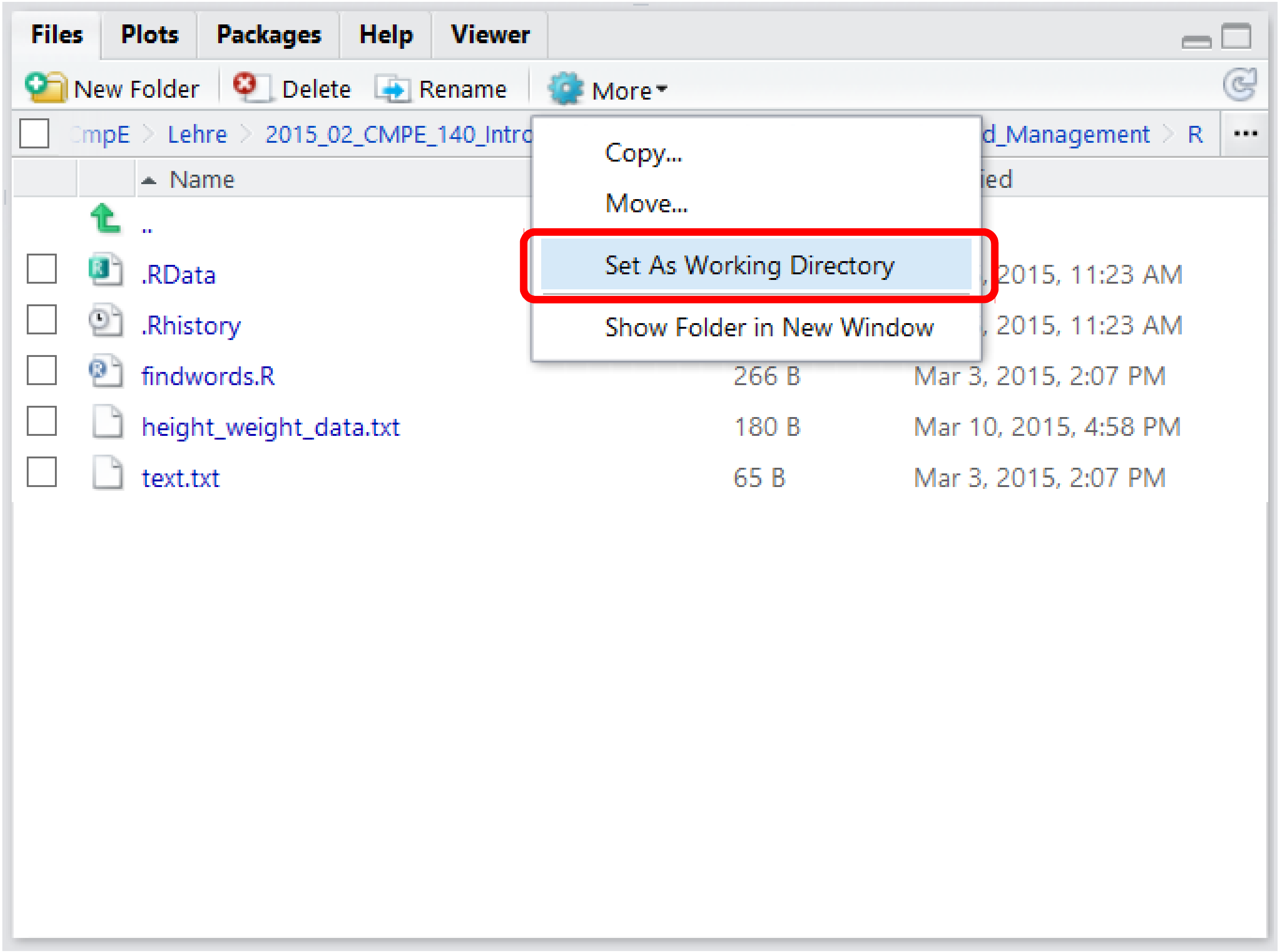


# Recap: change working directory in RStudio




- 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: word list

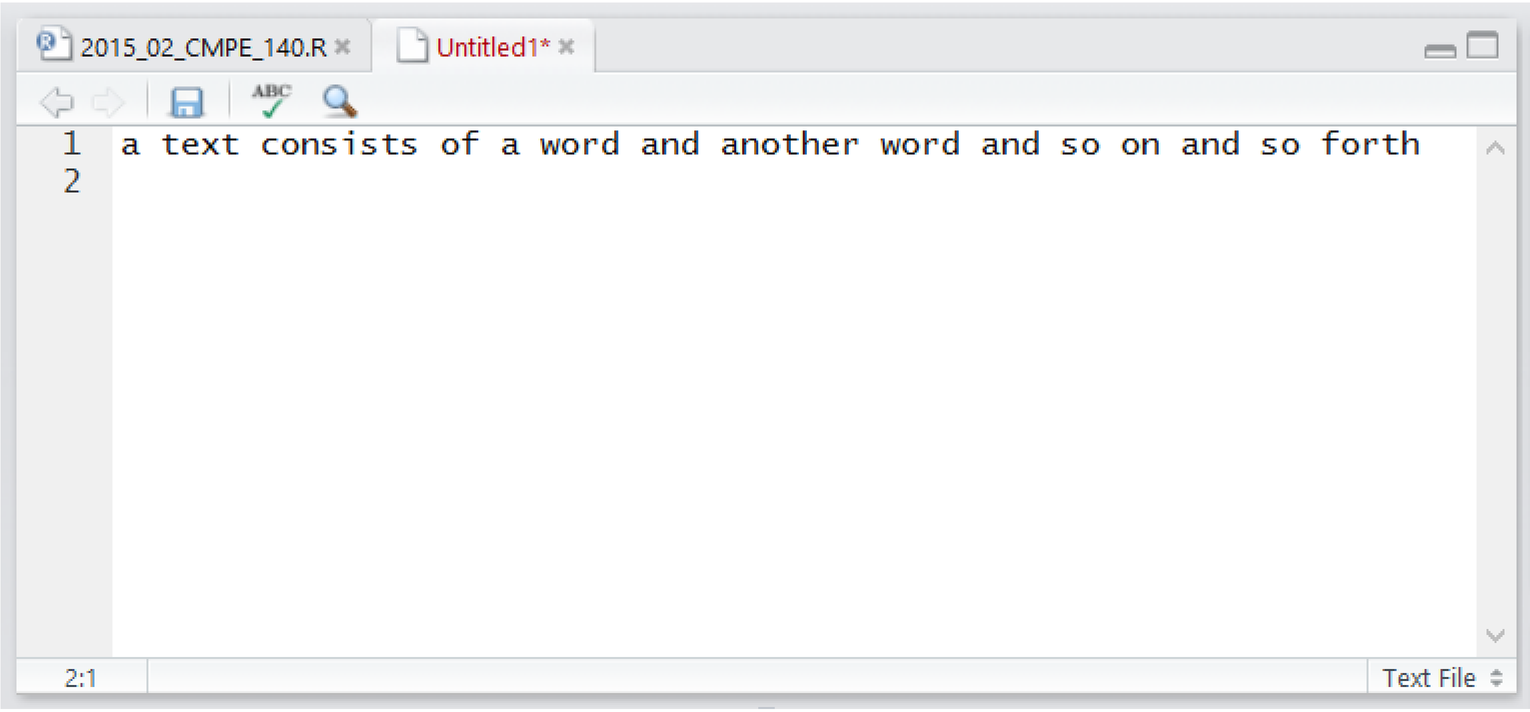
- Our goal is to determine at which location in the text each word occurs

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: import text data from file




- First, we write our text into a file
- In the following we import the text data to determine at which location in the text each word occurs

# Recap: import text data from file

We read text data from a file into a vector

```
> word.vec <- scan("text.txt", what="")
Read 15 items

> word.vec
 [1] "a"        "text"     "consists" "of"
 [5] "a"        "word"     "and"      "another"
 [9] "word"     "and"      "so"       "on"
[13] "and"      "so"       "forth"
```

- The second argument is a short form of `what=""` to indicate that we intend to import text data
- Similar like in `read.table()` the separator between items is ‘white space’ by default: one or more spaces, tabs, newlines or carriage returns

# 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: function `findwords`

```
> findwords("text.txt", sort.by.freq=T)
Read 15 items
$and
[1]  7 10 13

$a
[1] 1 5

$word
[1] 6 9

$so
[1] 11 14

$text
[1] 2
...
```

# Recap: plotting word frequencies

From the resulting list of word positions returned by `findwords` we can easily calculate the word frequencies using `sapply`

```
> word.list <- findwords("text.txt",
sort.by.freq=T)
Read 15 items

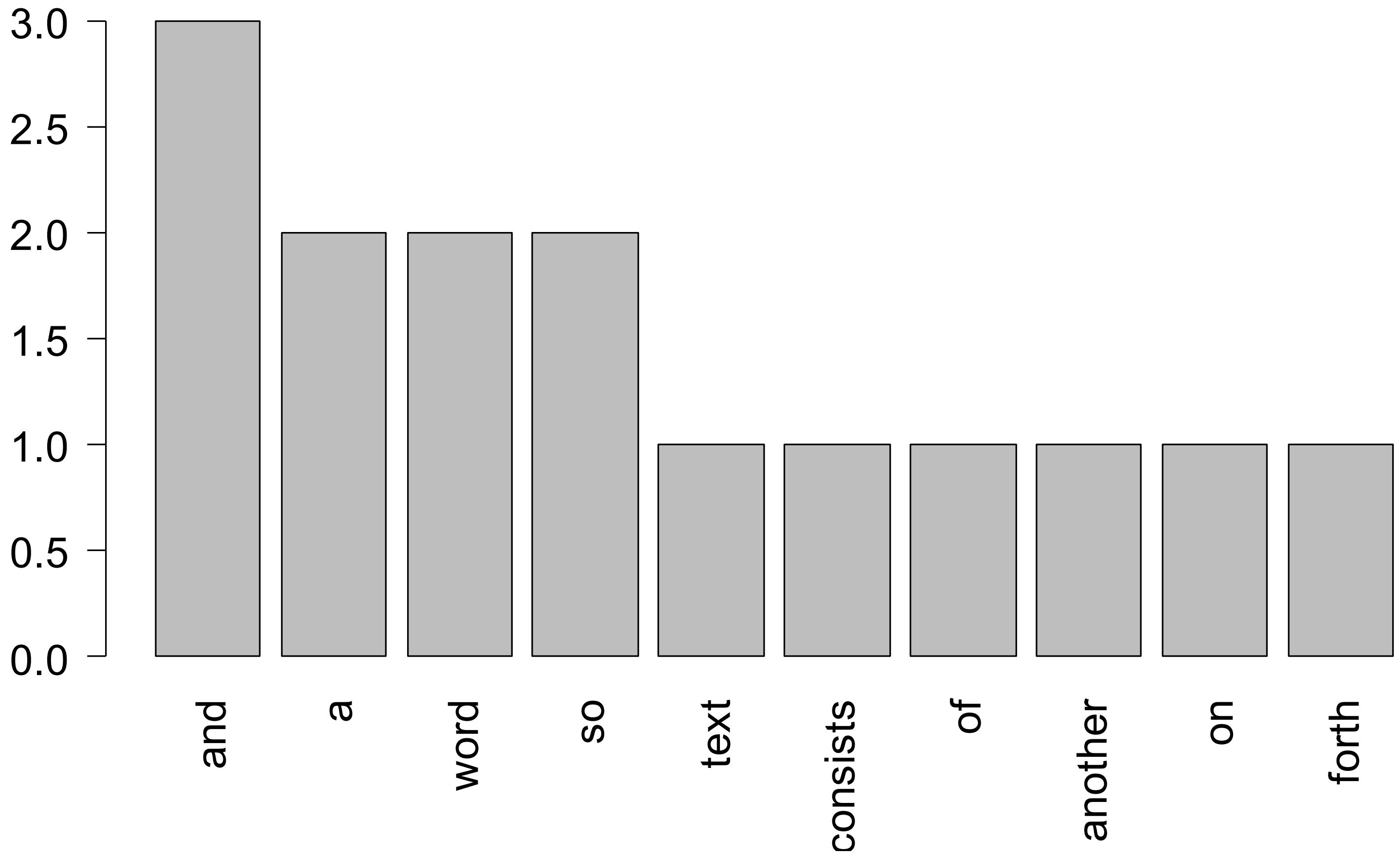
> word.freq <- sapply(word.list, length)

> word.freq
     and        a    word      so    text
       3        2       2       2       1
consists       of another      on   forth
       1        1       1       1       1
```

# Recap: plotting word frequencies

We create a barplot of the word frequencies

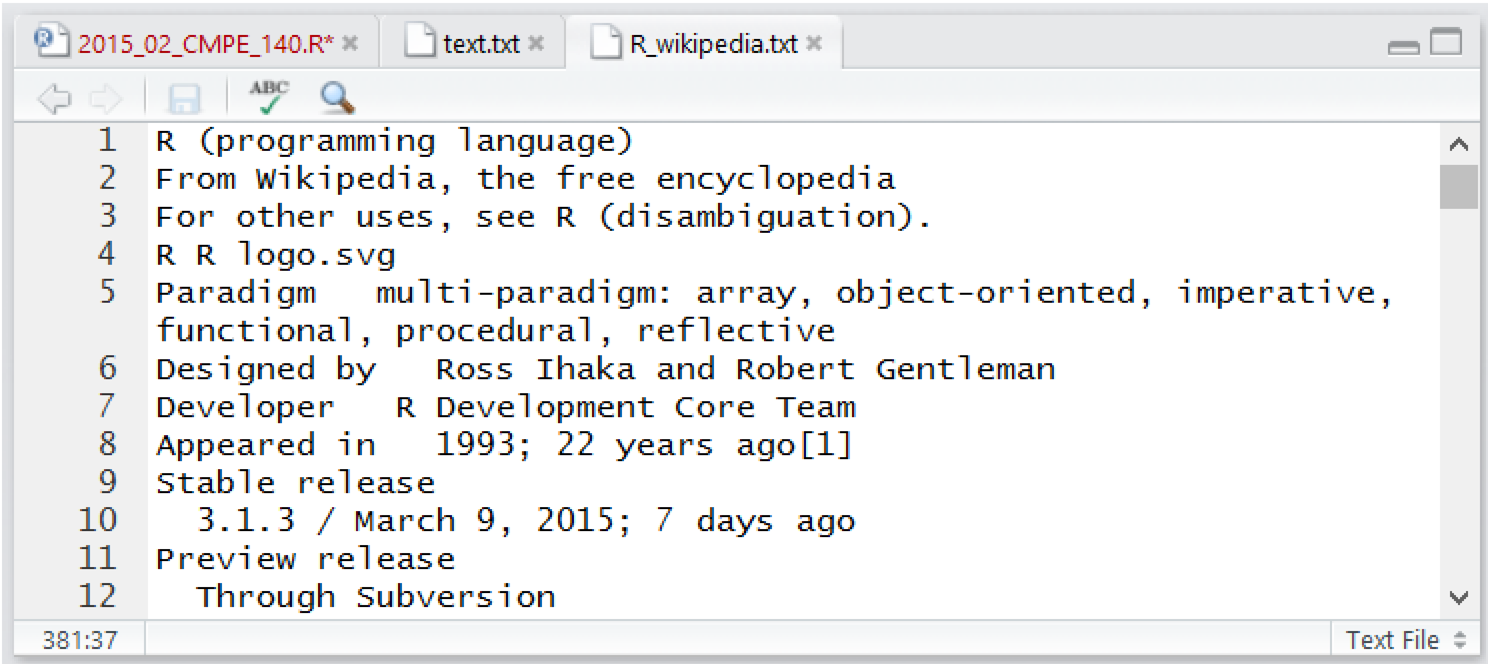
- `> barplot(word.freq, las=2)`



# 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 copy the article in a text editor
- We apply `findwords` and we create a barplot of the 10 most frequent words

# Recap: word frequency in Wikipedia


2015_02_CMPE_140.R*
text.txt
R_wikipedia.txt

```
R (programming language)
From Wikipedia, the free encyclopedia
For other uses, see R (disambiguation).
R R logo.svg
Paradigm   multi-paradigm: array, object-oriented, imperative, functional, procedural, reflective
Designed by   Ross Ihaka and Robert Gentleman
Developer   R Development Core Team
Appeared in   1993; 22 years ago[1]
Stable release
  3.1.3 / March 9, 2015; 7 days ago
Preview release
  Through Subversion
```

381:37
Text File

- We copy the Wikipedia article about R programming language and save it as `R_wikipedia.txt`

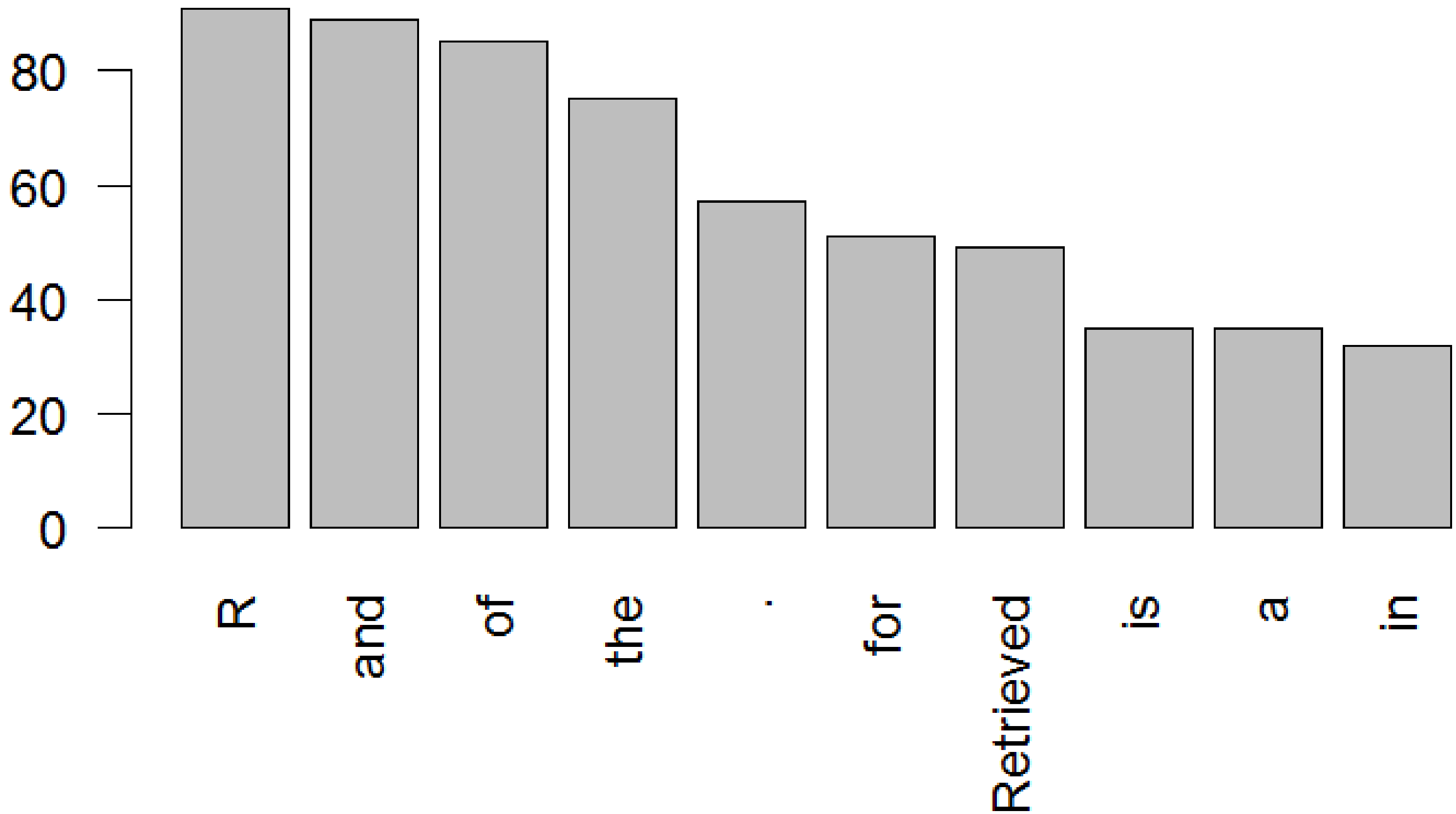
# Recap: word frequency in Wikipedia

```
> 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: quiz with the while loop

- We use `readline` to get user input from command line

We use the `while` loop to ask the same question again and again until we get the right answer

```
quiz <- function()
{
  answer <- 0
  while(answer != 143)
  {
    answer <- readline("How many students are registered
for this course? ")
    answer <- as.numeric(answer)
  }
  print("Congratulations, 143 is the right number.")
}
```

# Recap: quiz with the while loop

```
> quiz()
How many students are registered for this course? 50
How many students are registered for this course? 100
How many students are registered for this course? 200
How many students are registered for this course?
```

- We better provide some help for solving the quiz …

# Recap: quiz with the while loop

```
quiz <- function()
{
  answer <- 0
  while(answer != 143)
  {
    answer <- readline("How many students are registered for 
this course? ")
    answer <- as.numeric(answer)

    if(answer < 143) {print("No, more students.")}
    if(answer > 143) {print("No, less students.")}
  }
  print("Congratulations, 143 is the right number.")
}
```

# Recap: quiz with the while loop

```
> quiz()
How many students are registered for this course? 50
[1] "No, more students."
How many students are registered for this course? 100
[1] "No, more students."
How many students are registered for this course? 200
[1] "No, less students."
How many students are registered for this course? 143
[1] "Congratulations, 143 is the right number."
```

# Program today

- Read data with the the `scan()` function
- Read data from the Internet
- Export Graphics
- Write data to files

# Read data with scan

We already know the `scan` function from reading text data from a file into a vector

- `> word.vec <- scan("text.txt", what="")`

- Usually we use `scan` to read the entire content of a file into a vector

- We have learned in a previous lecture that we need to be careful when dealing with vectors and mixed modes

In the following we learn how `scan` behaves when confronted with mixed modes

# Read data with scan

Suppose we have files named f1.txt, f2.txt, f3.txt, and f4.txt with the following contents:

f1.txt
```
123
4 5
6
```

f2.txt
```
123
4.2 5
6
```

f3.txt
```
abc
de f
g
```

f4.txt
```
abc
123 6
y
```

Let's check how `scan` behaves when reading those files.

# Read data with scan

f1.txt

```
123
4 5
6
```

```
> scan("f1.txt")
Read 4 items
[1] 123   4   5   6
```

We get a 4-dimensional vector of integers as expected.

# Read data with scan

f2.txt

```
123
4.2 5
6
```

```
> scan("f2.txt")
Read 4 items
[1] 123.0   4.2   5.0   6.0
```

We get a 4-dimensional vector of floating-point numbers since with `4.2` we havd a non-integer number in our file.

# Read data with scan

f3.txt

```
abc
de f
g
```

```
> scan("f3.txt")
Error in scan(file, what, nmax, sep, dec,
quote, skip, nlines, na.strings,  :
  scan() expected 'a real', got 'abc'
```

We get an error message since `scan` was expecting numbers and not characters.

# Read data with scan

f3.txt

```
abc
de f
g
```

```
> scan("f3.txt", what="")
Read 4 items
[1] "abc" "de"  "f"   "g"
```

The type of the `what` argument gives the type of data to be read. If we specify `what=""` then we get a 4-dimensional vector of characters.

# Read data with scan

f4.txt

```
abc
123 6
y
```

```
> scan("f4.txt", what="")
Read 4 items
[1] "abc" "123" "6"    "y"
```

We get a 4-dimensional vector of characters.

# Read data with scan

- By default, `scan()` assumes that the items of the vector are separated by whitespace, which includes blanks, carriage return/line feeds, and horizontal tabs.
- We can use the optional `sep` argument for other situations.
- For example, we can set `sep` to the newline character `\n` to read in each line as a string

# Read data with scan

f3.txt

```
abc
de f
g
```

```
> scan("f3.txt", what="", sep="\n")
Read 3 items
[1] "abc"  "de f" "g"
```

Each line of the file is now a vector element and thus `de` and `f` are assigned together.

# Read data with scan

We can use `scan()` to read from the keyboard by specifying an empty string instead of the filename

```
> v <- scan("")
1: 1
2: 2 3 4
5: 17
6:
Read 5 items

> v
```

- `[1]  1  2  3  4 17`

# Read from the keyboard

If we want to read in a single line from the keyboard, `readline()` is a good choice

```
> readline()
This line will be stored in a single string
[1] "This line will be stored in a single 
string"
```

We can call `readline()` with an optional prompt

```
> readline("Please enter your line: ")
Please enter your line: Here is my line
[1] "Here is my line"
```

# Read into a matrix

- There is no direct way of reading from a file into a matrix

- We can use `scan()` in combination with `matrix()` to create a matrix from data in a file

Let's assume we have the following data stored in the file `height_weight.txt`

```
1.7 65
1.75 66
1.62 61
1.76 64
1.78 63
```

# Read into a matrix

We use `scan()` to read in the matrix row by row

```
> matrix(scan("height_weight.txt"), ncol=2,
byrow=TRUE)
Read 10 items
      [,1] [,2]
[1,] 1.70   65
[2,] 1.75   66
[3,] 1.62   61
[4,] 1.76   64
[5,] 1.78   63
```

# Read into a matrix

As alternative we can use `read.table()`, which returns a data frame, and then convert via `as.matrix()`

```
> as.matrix(read.table("height_weight.txt"))
       V1 V2
[1,] 1.70 65
[2,] 1.75 66
[3,] 1.62 61
[4,] 1.76 64
[5,] 1.78 63
```

# Reading text files

We have already learned how to read text data from a file into a vector

```
> word.vec <- scan("text.txt", "")
Read 15 items

> word.vec
 [1] "a"          "text"       "consists" "of"
 [5] "a"          "word"       "and"      "another"
 [9] "word"       "and"        "so"       "on"
```

- ```
  [13] "and"        "so"         "forth"
  ```
  ```

# Reading text files

- In text files, lines are usually separated by the newline character `\n`

We can use `readLines()` to read in a text into a vector of strings

```
> readLines("f3.txt")
```

- `[1] "abc"  "de f" "g"`

We observe that the result is equivalent to using `scan()` and setting `sep` to the newline character `\n`

```
> scan("f3.txt", what="", sep="\n")
Read 3 items
```

- `[1] "abc"  "de f" "g"`

# Accessing files from the Internet

- Certain input functions, such as `read.table()` and `scan()`, can read data from web URLs as well

- Reading data from the Internet is of particular interest when dealing with real-time data like stock exchange

- In the following we will access files from
  - Economics
  - The UCI Machine Learning Repository
  - Project Gutenberg

# Economics datasets

http://data.worldbank.org/topic/financial-sector

# The UCI Machine Learning Repository

- The UCI Machine Learning Repository is a collection of databases that are used by the machine learning community for the empirical analysis of machine learning algorithms

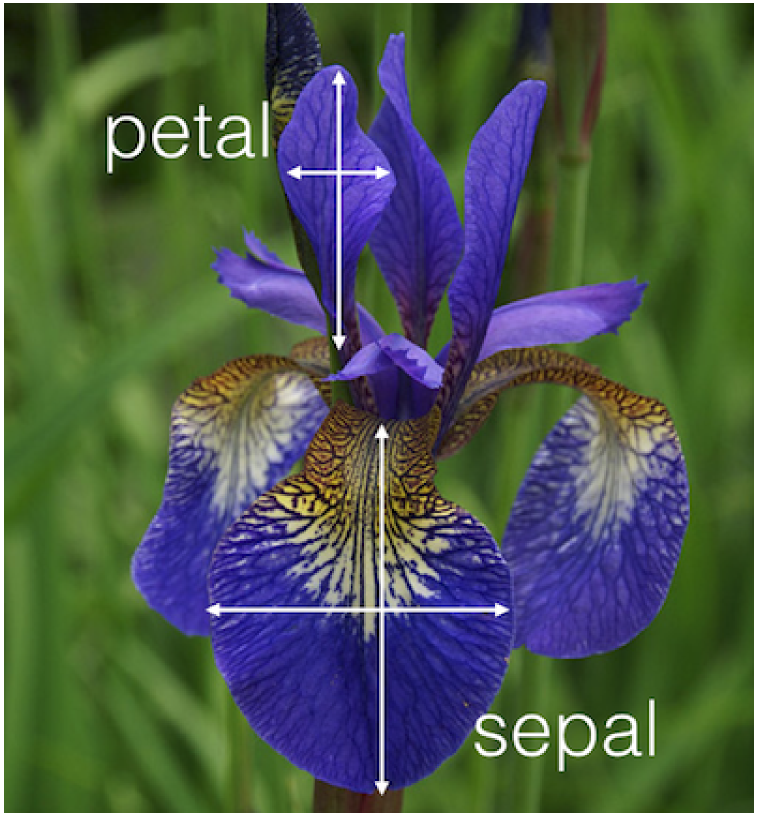

- Currently, 311 datasets are available, see http://archive.ics.uci.edu/ml/datasets.html
  - Diabetes, dermatology, hearth disease
  - Mushroom, iris, mammals
  - Faces, etc.

# Iris flower data set

- We will work with the most famous dataset from the repository: the Iris flower data set

- The dataset consists of 4 dimensions:  length and width of the sepals and petals

- The dataset was introduced by Sir Ronald Fisher in 1936



# Iris flower data set

- Data were obtained from three different species of the Iris flower:
  - Iris setosa
  - Iris versicolor
  - Iris virginica
- For each of the three species 50 samples are available
- In total 150 measurement are available

*Iris setosa*

*Iris versicolor*

*Iris virginica*

# Iris flower data set

- URL of the Iris flower data set: http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

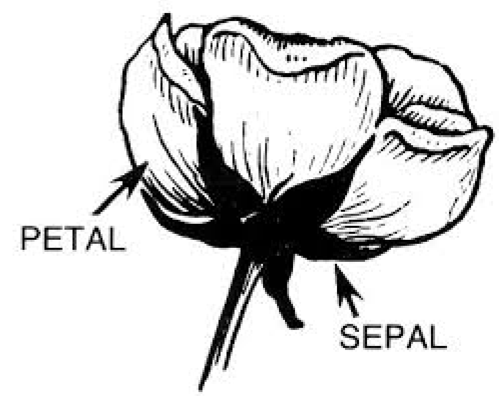
We read the data from the Internet source using `read.table()` by providing the URL instead of a file name

- 
```
> iris.flower <-
read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data",
sep=",")
```

# Iris flower data set

We can now work with the Internet data like with every other data frame

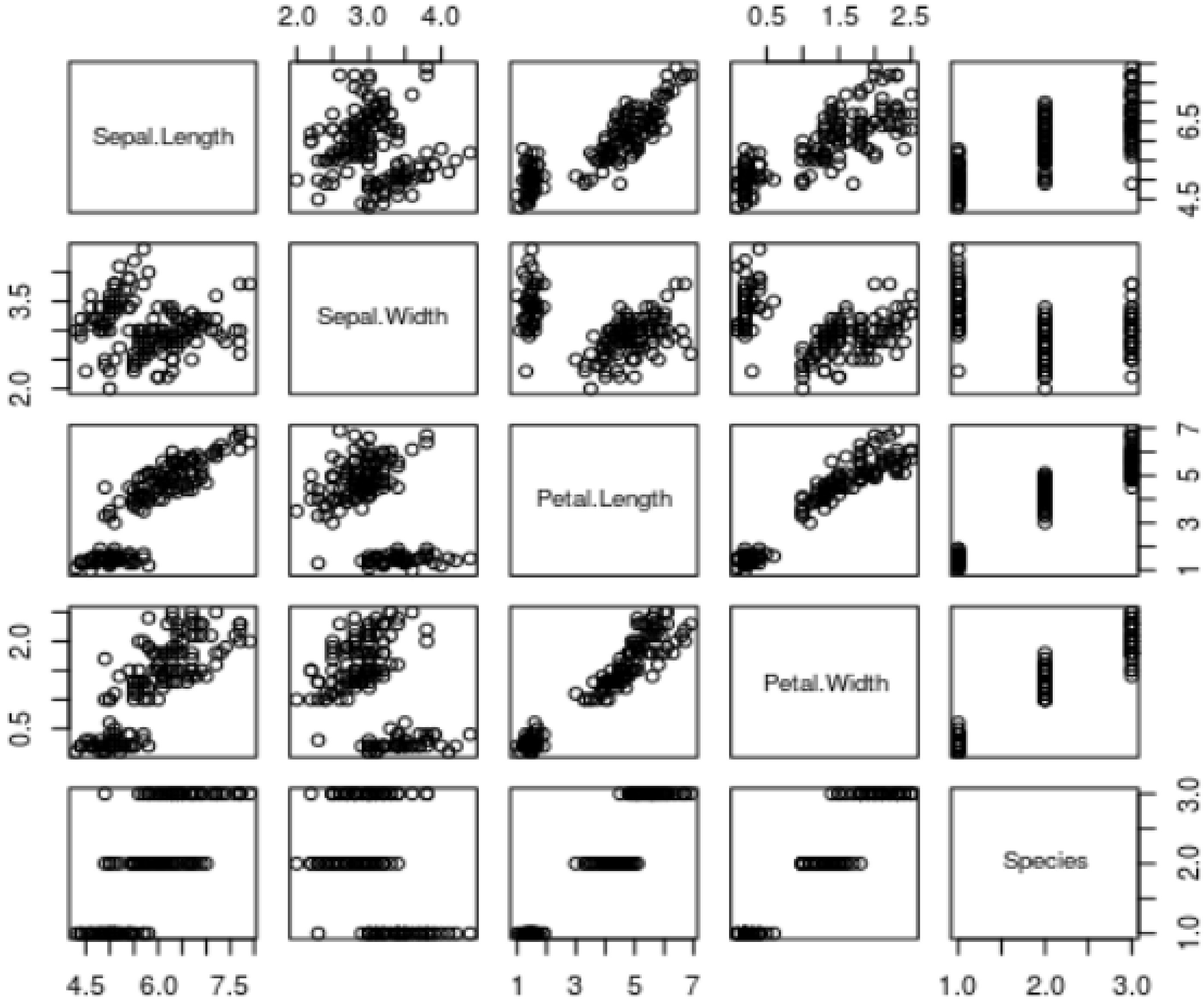
```
> head(iris.flower)
   V1  V2  V3  V4          V5
1 5.1 3.5 1.4 0.2 Iris-setosa
2 4.9 3.0 1.4 0.2 Iris-setosa
3 4.7 3.2 1.3 0.2 Iris-setosa
4 4.6 3.1 1.5 0.2 Iris-setosa
5 5.0 3.6 1.4 0.2 Iris-setosa
6 5.4 3.9 1.7 0.4 Iris-setosa
```




```
> colnames(iris.flower) <- c("Sepal.Length",
"Sepal.Width", "Petal.Length", "Petal.Width", "Species")
```

- `> plot(iris.flower)`

# Iris flower data set



# Iris flower data set

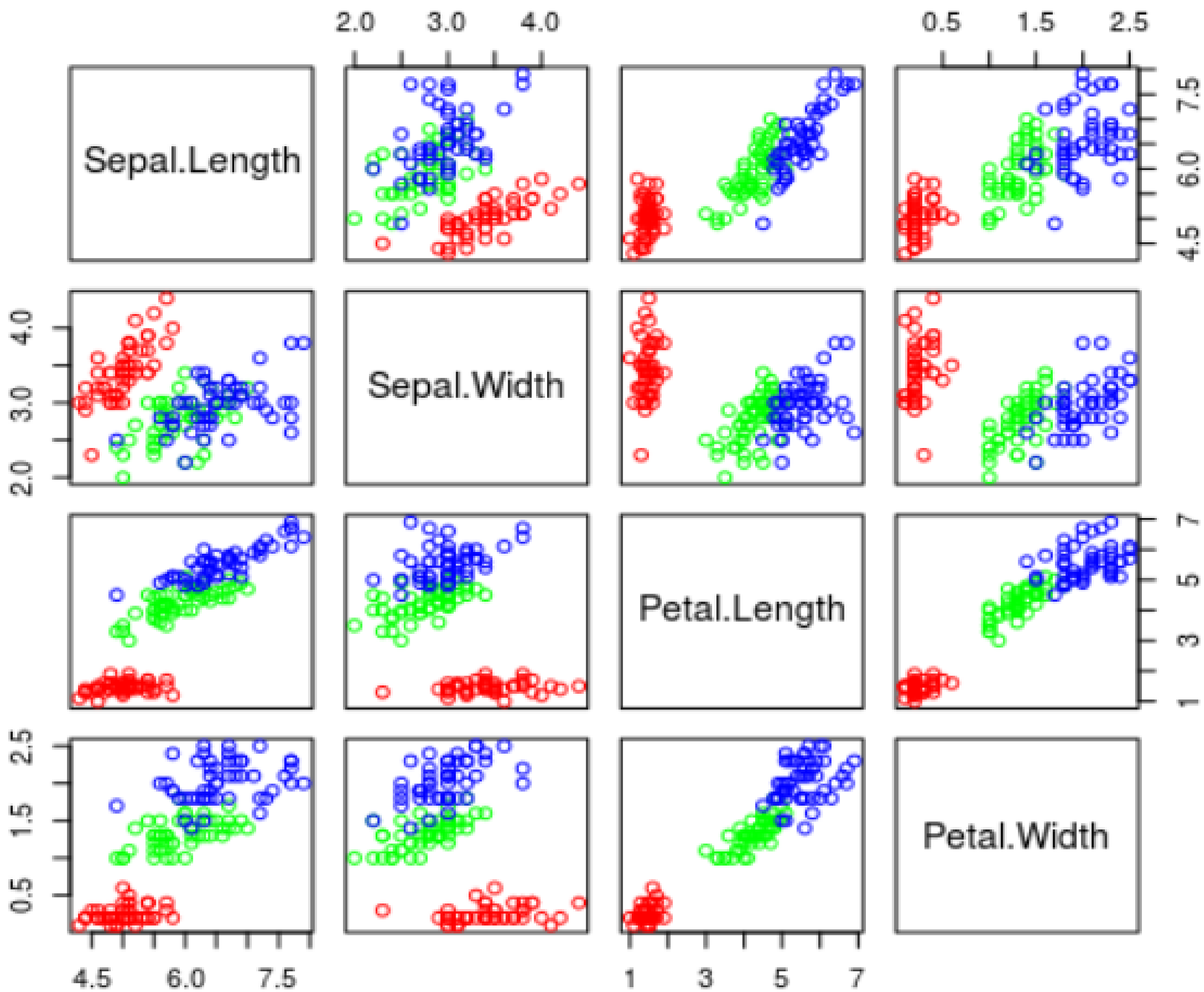
In order to color the data points according to their species, we add a new column color and specify the color information

```
> iris.flower$color <- ifelse(iris.flower$Species ==
"Iris-setosa", "red", ifelse(iris.flower$Species ==
"Iris-versicolor", "green", "blue"))
```

We plot again and use the color information for coloring the data points

```
> plot(iris.flower[,1:4], col=iris.flower$color)
```

# Iris flower data set



# Iris flower data set

- From the previous plot we can observe the most important characteristics of the Iris flower data set

- The red data points which contains samples from Iris-setosa are clearly separable from the two other species colored with green in red

For example, if we filter the data points which have petal length smaller than 2.5 we get only Iris-setosa

```
> iris.flower[iris.flower$Petal.Length < 2.5,]
```

- The two other species are not linearly separable from each other

# Project Gutenberg

- Offers over 46000 free ebooks

- Available formats are HTML, EPUB, Kindle, PDF, and plain text

- We make use of our function `findwords()` to analyze the most frequent words in some ebooks

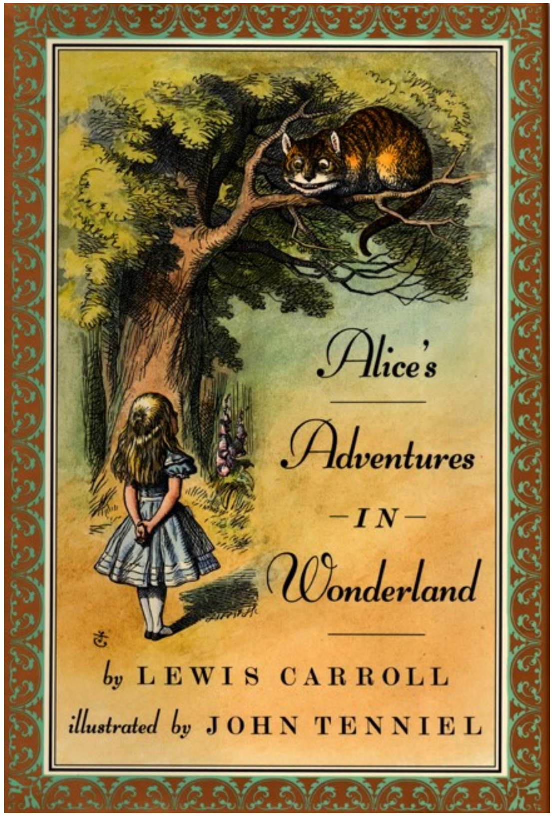
# Project Gutenberg

- We start with *Alice's Adventures in Wonderland* by Lewis Carroll

- According the Project Gutenberg, the book belongs to the most popular ebooks

- We can download various formats of this book from [http://www.gutenberg.org/ebooks/11](http://www.gutenberg.org/ebooks/11)

- Plain text is available from http://www.gutenberg.org/files/11/11-0.txt



# Project Gutenberg

When we previously analyzed the word frequencies of the Wikipedia article about R, we first created a text file and called `findwords()` with it

- `> word.list <- findwords("R_wikipedia.txt", sort.by.freq=T)`

In fact, we don't need to create a text file since the `scan()` function inside `findwords()` can read an URL directly

```
> word.vec <-
scan("http://www.gutenberg.org/files/11/11-
0.txt", "")
Read 24386 items
```

# Project Gutenberg

- Checking if you are a web-bot
- Therefore it is not possible to directly read with `scan()` function

- File → Save Page As → Alice.txt in R/ directory

# Project Gutenberg

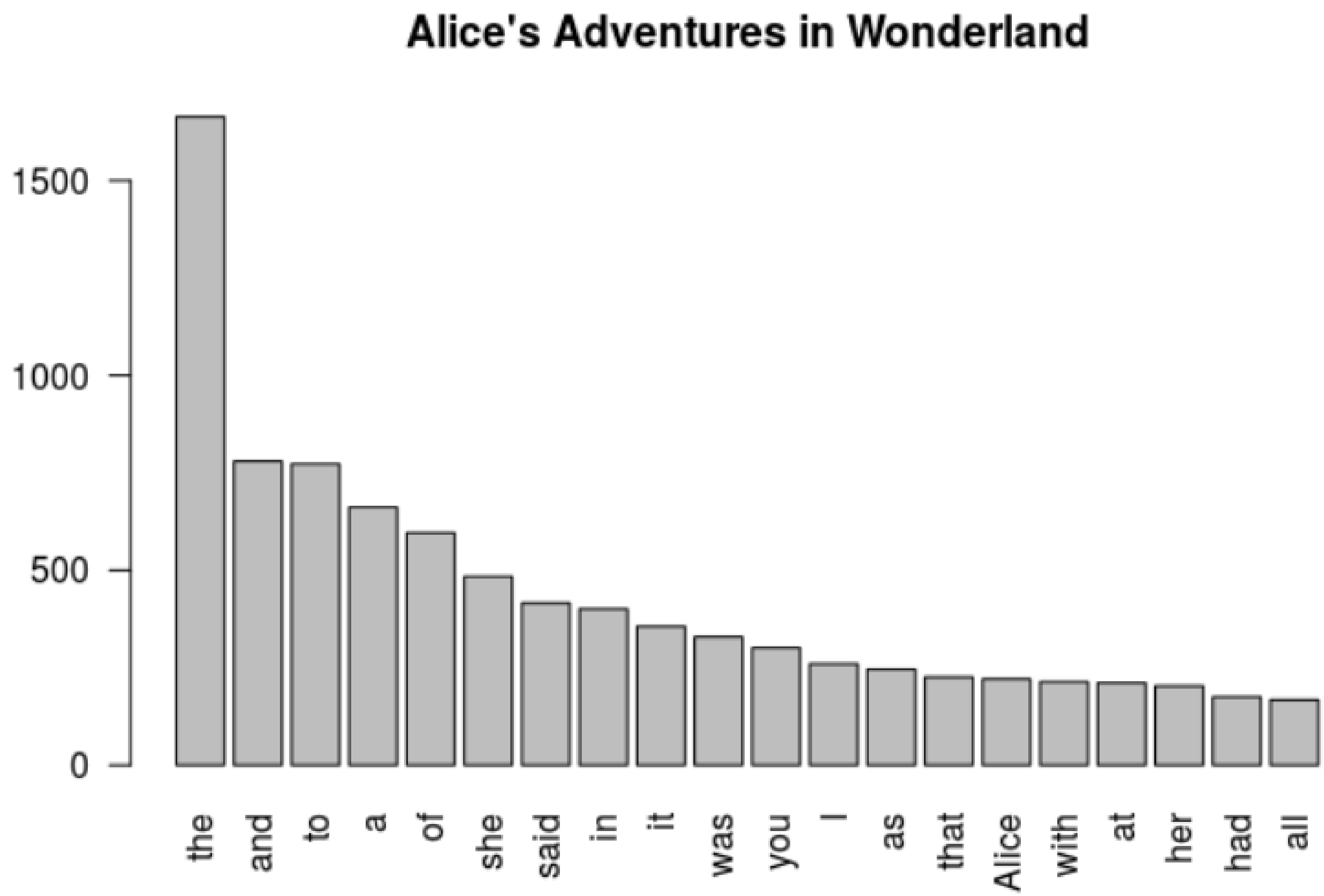
As before, we first create the word list with `findwords()`, next we compute the word frequencies and finally we plot the result as a barplot

```
> word.list <- findwords("Alice.txt", sort.by.freq=T)
Read 24386 items

> word.freq <- sapply(word.list, length)

> barplot(word.freq[1:20], las=2, main="Alice's
Adventures in Wonderland")
```

# Project Gutenberg



# Project Gutenberg

As a second book we analyze word frequencies from *The Adventures of Tom Sawyer* by Mark Twain

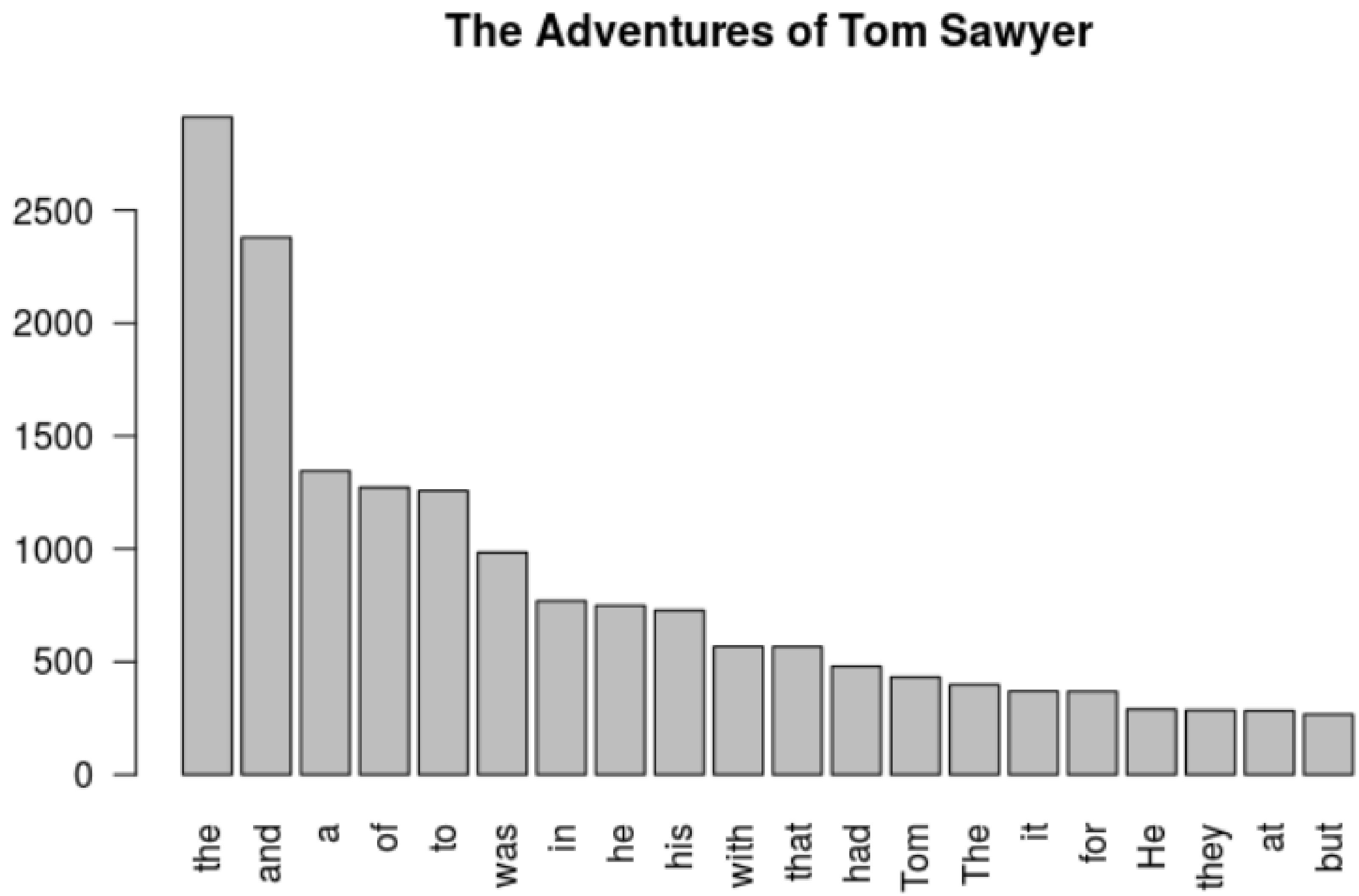
http://www.gutenberg.org/cache/epub/74/pg74.txt

```
> word.list <- findwords("TomSawyer.txt",
sort.by.freq=T)
Read 53582 items

> word.freq <- sapply(word.list, length)

> barplot(word.freq[1:20], las=2, main="The Adventures
of Tom Sawyer")
```

# Project Gutenberg



# Project Gutenberg

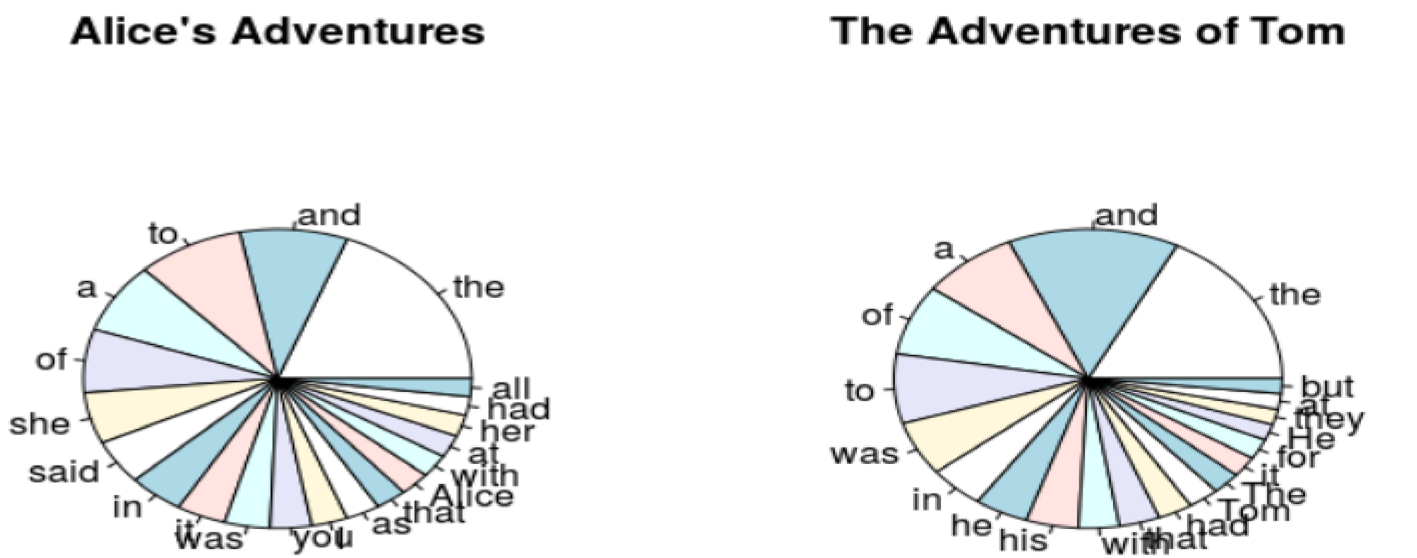
- In both bar plots we observe frequent English words like “the”, “and”, “a”, etc.

- Both plots differ obviously in the name of the main character: “Alice” vs. “Tom”

- We can also realize that in the first book the main character is female since we observe “she” and “her” while in the second book we observe “he” and “his”

# Compare book contents

```
word.list.alice <- findwords("Alice.txt", sort.by.freq=T)
word.freq.alice <- sapply(word.list.alice, length)
word.list.sawyer <- findwords("TomSawyer.txt", sort.by.freq=T)
word.freq.sawyer <- sapply(word.list.sawyer, length)
par(mfrow=c(1,2))
pie(word.freq.alice[1:20], main="Alice's Adventures")
pie(word.freq.sawyer[1:20], main="The Adventures of Tom")
par(mfrow=c(1,1))
```



# Compare book contents, diff

```
# a function that takes returns the frequencies that only
# appear in one vector.
freq.alice=word.freq.alice
freq.sawyer=word.freq.sawyer
only.alice <- getDiffFreq(freq.alice[1:30],freq.sawyer[1:30])
only.sawyer <- getDiffFreq(freq.sawyer[1:30],freq.alice[1:30])
par(mfrow=c(1,2))
pie(only.alice, main="Alice's Adventures")
pie(only.sawyer, main="The Adventures of Tom")
par(mfrow=c(1,1))

getDiffFreq <- function (vec1, vec2){

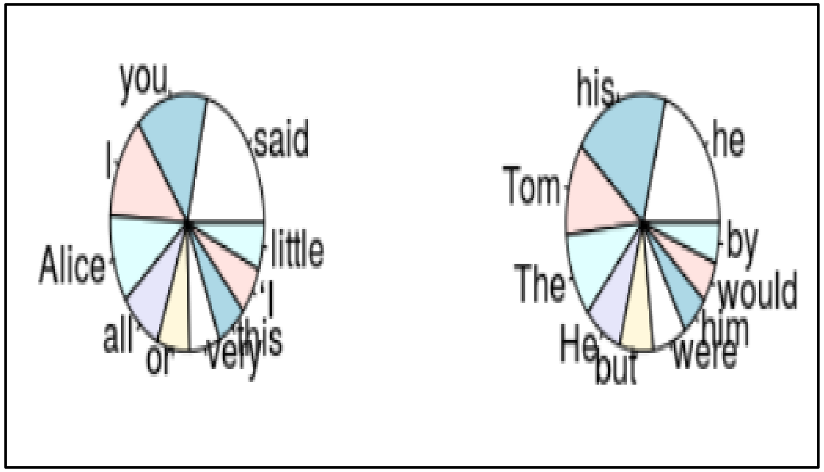
  return (uniqueVec1)
}
```

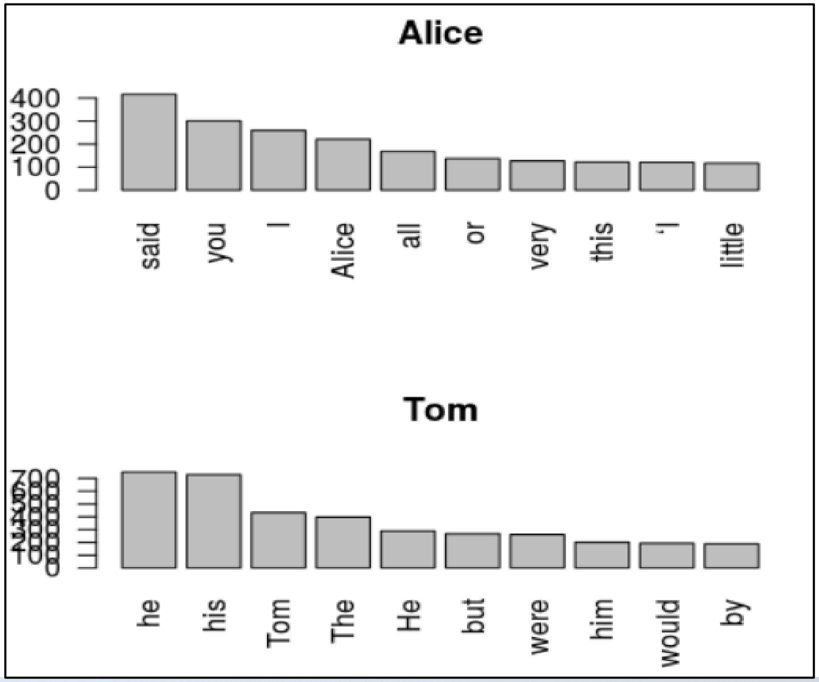
# Compare book contents, diff

```
GetDiffFreq <- function (vec1, vec2){
  uniqueVec1 <- c()
  For (
  return (uniqueVec1)
}
```

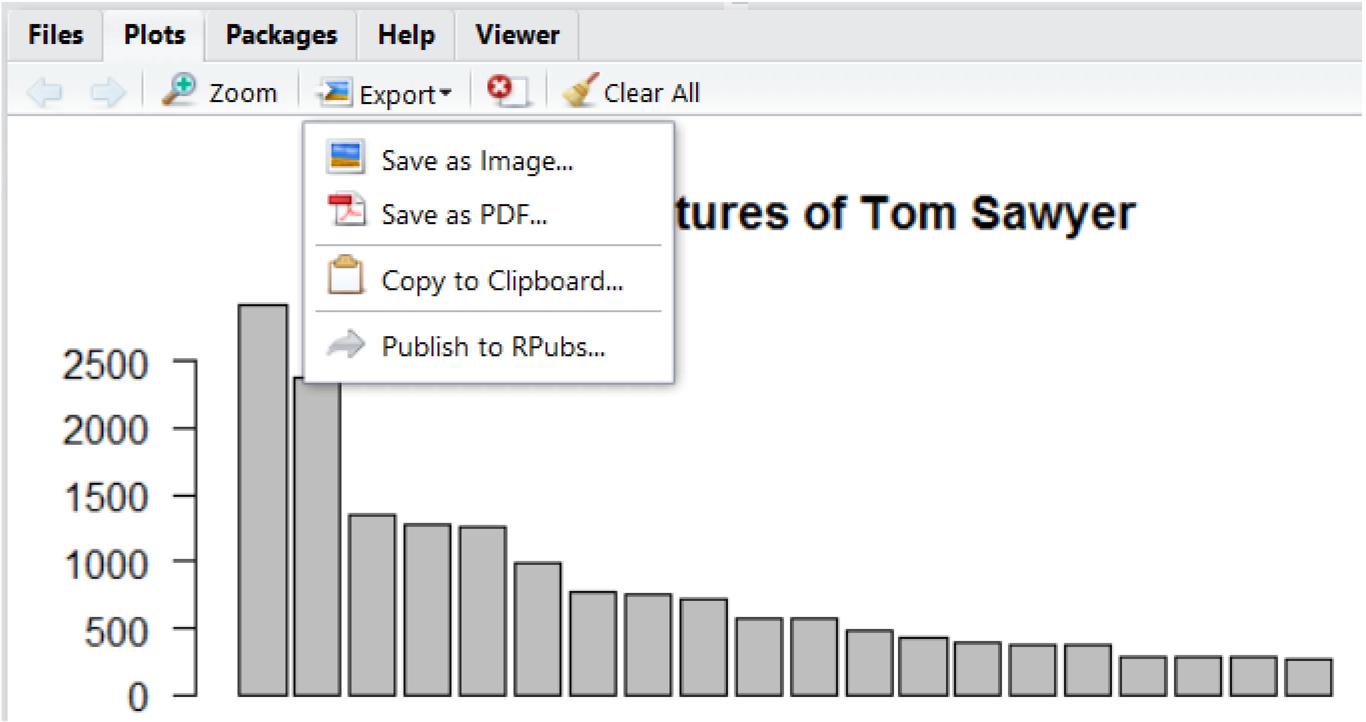
# Compare book contents, diff

```
getDiffFreq <- function(vec1, vec2){
  retVec <- c()
  for (i in 1:length(vec1)){
    vec1.word  <- names(vec1[i])
    existInVec2 <- FALSE
    for (j in 1:length(vec2)){
      if (names(vec2[j])==vec1.word){
        existInVec2 <- TRUE
        break
      }
    }
    if (existInVec2==FALSE){
      retVec <- c(retVec,vec1[i])
    }
  }
  return (retVec)
}
freq.alice=word.freq.alice
freq.sawyer=word.freq.sawyer
new.alice <- getDiffFreq(freq.alice[1:30],freq.sawyer[1:30])
new.sawyer <- getDiffFreq(freq.sawyer[1:30],freq.alice[1:30])
par(mfrow=c(1,2))
pie(new.alice, main="Alice's Adventures")
pie(new.sawyer, main="The Adventures of Tom")
par(mfrow=c(1,1))
par(mfrow=c(2,1))
barplot(new.alice, las=2,main="Alice")
barplot(new.sawyer, las=2,main="Tom")
par(mfrow=c(1,1))
```



# Export Graphics




- In RStudio the Export menu provides ways to export your graphics in various formats

# Writing to a file

- So far we have imported data, processed the data and visualized the results

- A convenient way to save data to a file is `write.table()`

- It works similar to `read.table()`

- `write.table()` writes a data frame into a file

# Recap: read.table

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
```

# Writing to a file

We compute the BMI

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

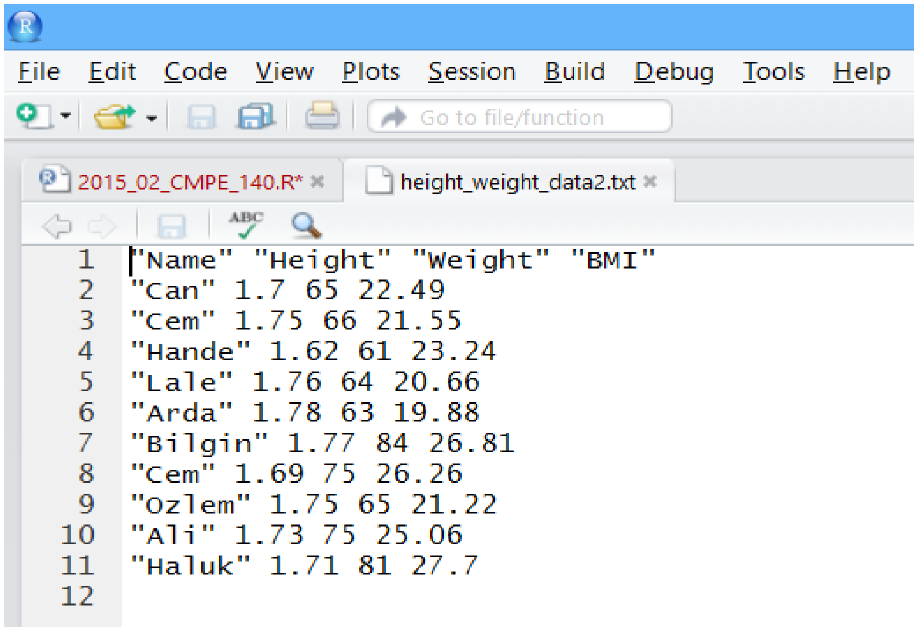
# Writing to a file

We store the extended data frame into the file `height_weight_data2.txt`

- `> write.table(person.data, "height_weight_data2.txt")`
- The file will be saved in your current working directory

# Writing to a file

- We check the file contents by opening the file in a text editor




```
"Name" "Height" "Weight" "BMI"
"Can" 1.7 65 22.49
"Cem" 1.75 66 21.55
"Hande" 1.62 61 23.24
"Lale" 1.76 64 20.66
"Arda" 1.78 63 19.88
"Bilgin" 1.77 84 26.81
"Cem" 1.69 75 26.26
"Ozlem" 1.75 65 21.22
"Ali" 1.73 75 25.06
"Haluk" 1.71 81 27.7
```

- We observe that a space is used by default to separate the entries

# Writing to a file

We open the file with `read.table` and check the contents

```
> person.data2 <- read.table(header=TRUE,
"height_weight_data2.txt", sep="")

> person.data2
     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
```

# Homework

1. Create a bar plot and a pie chart of the most frequent words from your favorite ebook

2. Export bar plot and pie chart as pdf

3. Use the data on body height and weight from previous homework. Compute BMI and store the resulting data frame in a file.