R Basics

R is a software package that can be used for statistical computing and for the creation of data visualisations. It is also a programming language. This language can be used exclusively within the R application, however. R can be used both to create data about texts,[1] and to analyse the resulting data sets. In the DTDP course, however, the data sets will be created using Perl programs. R will only be used for analyses and for visualisations of these data. First, some general information will be given about programming in R.

Variables

Like any other programming language, the R language makes use of variables. Variable names in R can consist of any combination of alphanumerical characters, underscores and dots. The only restriction is that the first character can never be a number, and that the second character cannot be a number when the first character is a dot. `Data1` and `.Data2` are both valid names, but `.3rdVariable` is not. Unlike Perl, R does not use special characters to indicate explicitly that a particular string is in fact a variable.

The assignment operator in R is the "smaller than" sign, combined with a hyphen: <-.

R uses various variable types. The DTDP course only makes use of vectors and of data frames.

Vector

A vector is a list of values. In a vector, all the values are of the same data type (e.g. a string or a number).

Vectors can be created using the concatenation function, `c()`. As arguments for this function, you should provide all the values that should be in the vector, separated by a comma. Vectors can also be created by providing a range of numerical values. Within this range, the first and the last item in the range need to be separated by a colon.

```
x <- 1:30

y <- c( 4, 5, 3, 7)
```

When vectors are created, using either one of these two methods, R automatically creates a numerical index for each item. The first value has index 1. Individual items in a vector can be accessed using the index of this item, supplied within square brackets directly following the vector name. It is also possible to retrieve multiple items simultaneously by providing a range of indexes.

```
x[1:3]
```

[1] A detailed explanation of the ways in which aspects of texts may be quantified can be found in Mathew Jockers, *Text Analysis with R for Students of Literature* (New York: Springer 2014).

```
y[5]
```

A vector always has a certain a length. This term refers to the number of items contained in the vector. The length of a vector can be found via the `length()` function:

```
x <- 5:10

y <- length(x)

## variable y has value 5
```

Data frame

A data frame is a collection of vectors. All of these vectors are of the same length. Data frames can be created using the data.frame() function, as follows:

```
x <- 1:20
y <- 21:40

df <- data.frame(x,y)
```

A data frame can be compared to a regular table in a database. The vectors in a data frame are similar to the columns in such a table. In the code above, the two vectors x and y both contain twenty items. The function `data.frame()` will create a data frame with two columns and with twenty rows. The row names will be created automatically and will consist simply of numbers. The column names will be the names of the two vectors.

To view the separate vectors (or columns) in a data frame, you can use the $ operator. More specifically, you need to give the name of the data frame, followed by a dollar sign, and the name of the column that you want to see. In the data frame that was created using the code above, the column that is labelled “y” can be accessed using the following syntax.

```
df$y
```

The names of the rows and the columns can be requested using the following functions.

```
rownames(df)
#this will print a list ranging
# from 1 to 20, i.e. all the rows

colnames(df)
# this will print ‘x’ and ‘y’
```

While, as has been discussed, data frames can be created using the `data.frame()` function, data frames, in practice, are usually created by reading in CSV files.

Reading data into R

R generally works with data formatted Comma Separated Value (csv) format. Below, you can find an example of a csv file.

```
type,token,ratio
SecretAgent.txt,8915,90078,0.0989697817447101
HeartOfDarkness.txt,5699,38545,0.147853158645739
```

If you save this file under the name ratio.csv, you can read this file into R, using the read.csv() function:

```
ttr <- read.csv( "ratio.csv" , header = TRUE) ;
```

The read.csv() function will create a data frame. The header = TRUE parameter will have the effect that the values on the first line and the first values of all other lines will be treated as column names and row names. As you can see in the example above, the first line in the CSV file has three values, while all the lines that follow have four values. The values on the first lines will be used as column names. The values on the first 'column' on the subsequent lines are not explicitly labelled, but these values will be used as row names. R transforms the CSV file into a format that may be visualised as follows:

	type	token	ratio
SecretAgent.txt	8915	90078	0.0989697817447101
HeartOfDarkness.txt	5699	38545	0.147853158645739

To receive some general information about the data frame, you can use the str() function or the summary() function. The str() function gives information about the total number of rows and columns, and it also gives information on the data types of each vector in the data frame.

```
str(ttr)

## Output:

'data.frame': 2 obs. of  3 variables:
 $ type : int  8915 5699
 $ token: int  90078 38545
 $ ratio: num  0.099 0.148
```

To identify individual items in a data frame, you need to use square brackets. You need to provide two values: (1) the row number and (2) the column number. The two values need to be separated by a comma. When one of the values is omitted, it will be assumed that a whole row or a whole column needs to be selected. The code below contains a number of examples.

```
## Value on the first row, in the first column
df[1,1]

## The entire first row
df[1,]

## All the values in the second column
df[,2]

## The first fifteen rows and ALL of the columns
df[1:15,]
```

Basic calculations

If a vector x contains numbers, the following basic operations can be performed on them:

`mean( x )`	The average value of the vector
`max(x)`	The maximum value of the vector
`min(x)`	The minimum value of the value
`sd(x)`	Standard deviation
`summary(x)`	Prints the mean, the median, max, min and the quantiles.
`sort(x)`	This function sorts the vector numerically. By default, the function sorts in an ascending way. To sort in a descending way, use the parameter decreasing = TRUE
`sum()`	Adds all the values in the vector.

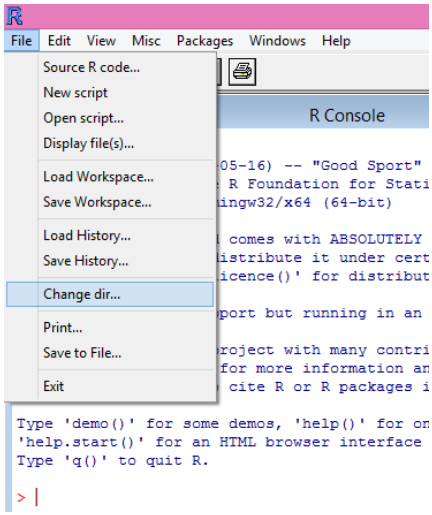
To order a data frame in its entirety, you can use the order() function. As a first parameter to this function, you need to provide the column on which you want to sort. The second parameter can be left empty. By default, the function uses an ascending order. To sort in a descending way, place a minus sign in front of the name of the vector.

```
sorted <- data[order( -data$num_words ),]
```

Running R code

R code can be created and edited in a regular text editor such as NotePad++. To execute the code, follow the steps below.

First, set your working directory. This, more specifically, is the folder on your computer which contains the file that you want to run. You can do this by choosing “File” > “Change dir” and by navigating to your working directory.



Next, choose “File” > “Source R Code”. This will open a new window which displays all the files in the working directory. Select the file that you want to execute and click on “Open”. On the command prompt in R, this will firstly print the source() function, mentioning the file that you have selected. The output of the program will also be shown.

Ggplot

Within R, several code libraries can be used for the creation of graphics. Ggplot is probably the most advanced and the most versatile package. The current version is ggplot2. To work with ggplot2, the following packages need to be installed:

```
install.packages( "ggplot2", dependencies = TRUE ) ;
install.packages( "gridExtra", dependencies = TRUE ) ;
install.packages( "reshape", , dependencies = TRUE ) ;
install.packages( "labeling" , dependencies = TRUE ) ;
install.packages( " RColorBrewer" , dependencies = TRUE ) ;
```

When, in a script, you want to make use of the ggplot functions, you must include the following line:

```
library(ggplot2)
```

GGplot is a technical implementation of Leland Wilkinson’s generic description of data visualisations, as given in the monograph *The Grammar of Graphics*. In the book, Wilkinson notes that a graphic is “a mapping from data to aesthetic attributes (colour, shapes, size) of geometric objects (points, lines

bars)".[2] To create a diagram in ggplot, data values firstly needs to be associated with specific aesthetic primitives. This can be accomplished by making use of `aes()` function within the general `ggplot()` function. The first parameter of `ggplot()` is the name of the data frame whose values needs to be visualised. The second parameter, `aes()`, connects specific columns of this data frame to aesthetic attributes. The following options may be used:

Aesthetic attributes:

x	The first variable to be visualised. In a scatter plot, this is the variable which is shown on the x-axis
Y	The second variable to be visualised. In a scatter plot, this is the variable which is shown on the y-axis
col	The colour of a geometrical object. This parameter is generally used to specify the colour of the points in a scatter plot.
fill	The fill colour of a geometrical object. This parameter can be used in combination with barplots.
shape	The shape of the points in a scatter plot.
size	The size of the points in a scatter plot.
label	Text labels for the points in a scatter plot.

These aesthetic properties can be used in combination with geometrical objects. Examples include the following:

Geometrical objects:

geom_bar	A bar chart.
geom_point	A scatter plot.
geom_freqpoly	A "frequency polygram". This function generally creates a line chart, on the basis of the frequencies of a particular variable.
geom_boxplot	A boxplot.
geom_text	Text labels for the points in a scatter plot.

When the plot is printed, the geometric object is drawn on a specific coordinate system, using the aesthetic attributes that have been specified.

A number of additional functions can be used to manipulate the general appearance of the plot.

`coord_flip()`	This function inverses the x-axis and the y-axis. In other words, this changes the orientation of the diagram.
`scale_x_continuous()`	Change the values of "ticks" on the x axis. Using "breaks", followed by the name of a column has the effect that all the values of the corresponding vector will be shown.
`scale_y_continuous()`	Change the values of "ticks" on the y axis.
`xlab()`	Text to be shown underneath the x-axis

[2] Leland Wilkinson, The grammar of graphics (New York: Springer 2005).

`ylab()`	Text to be shown next to the y-axis
`xlim()`	Specifies the range of value to be shown on the x-axis. xlim() and ylim() demands two values: the first value and the last value in the range.
`ylim()`	Specifies the range of value to be shown on the y-axis

These appearance of these various geometrical objects can be further refined using a number of parameters. The tables below discuss some of the parameters which can be used inside geom_bar() and geom_text ().

geom_bar	Stat	Value "bin" is the default option. This results in a count of all the occurrences of the various unique values. When you use stat = "bin", you only need to supply one variable as input in the aes() function. Alternatively, you can use the value "identity". In this case, you need to supply to variables as input ("X" AND "Y" in the aes() function). The first variable specifies the labels to be used for the various bars, and the second variable specific the lengths of these bars.
	Colour	This parameter specifies the colour of the outline. Inside geom_bar, you can to supply a colour in hexadecimal format.
	Fill	This parameter specifies the fill colour of the bar. Inside geom_bar, you can to supply a colour in hexadecimal format.
	width	This parameter specifies the width of the bars.

geom_text	col	The colours of the text labels
	hjust and vjust	By default, the text labels are shown on the points. Using different values for hjust and vjust, you can shift the labels to position above or below the point, or to the left or to the right of a point.
	size	The font size of the labels.

These different functions and parameters are illustrated in the examples below.

The example in listing 1 makes use of a csv file named Elzevier*.csv*. This file contains brief descriptions of books published by Elsevier. The data are based on an export from the *Short Title Catalogue of the Netherlands*.[3]

With ggplot, a bar plot can be created using the following code:

[3] http://www.kb.nl/organisatie/onderzoek-expertise/informatie-infrastructuur-diensten-voor-bibliotheken/short-title-catalogue-netherlands-stcn

```
1.      library("ggplot2") ;
2.
3.      stcn <- read.csv("stcn.csv" , header = TRUE ) ;
4.
5.      plot <- ggplot( stcn , aes( x=subject1 )) plot +
6.      geom_bar( ) + coord_flip( ) ;
7.
8.      print (plot)
9.
```

Listing 1.

Note that, once the name of the dataframe has been declared as the first parameter of ggplot, this data frame does no longer need to be mention within the aes() function. In this latter function, the column names suffice.

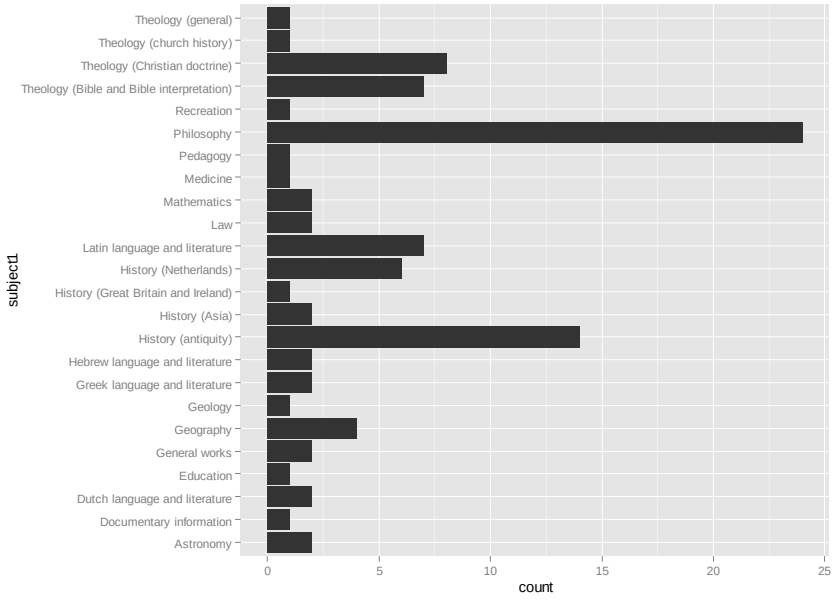


Image 1. Output of listing 1

One of the attractive aspects of ggplot is that it enables you to make a clever use of so-called facet variables. These are essentially variables which assign specific categories to the records in the data frame. If the data frame contains such a facet variable, this variable can be used to determine some of the other aesthetic attributes of the graph. Listing 2 is very similar to listing 1. The only difference is that a "fill" parameter has been added. In the code below, it is specified that the colours of the bar must be determined by the languages.

```
1.      library("ggplot2") ;
2.      library("RColorBrewer") ;
3.
4.      stcn <- read.csv("stcn.csv" , header = TRUE ) ;
5.      cols <- c( brewer.pal( 9 , "Set1") , brewer.pal( 8 ,
```

```
6.      "Accent") , brewer.pal( 8 , "Set2") , brewer.pal( 12 ,
7.      "Set3")  ) ;
8.
9.      plot <- ggplot( stcn , aes( x=subject1 , fill =
10.     language ) ) + geom_bar( ) + coord_flip( ) +
11.     scale_fill_manual(values = cols) + xlab("Subjects") ;
        print (plot) ;
```

Listing 2.

The instructions on lines 4-7 are used to create a colour palette to be used on the bars. The first parameter of the brewer.pal function specifies the number of colour to be generated. In all cases, the maximum number of colour in these palettes have been used. The command which begins on line 4 creates a long palette with 37 colour in total.

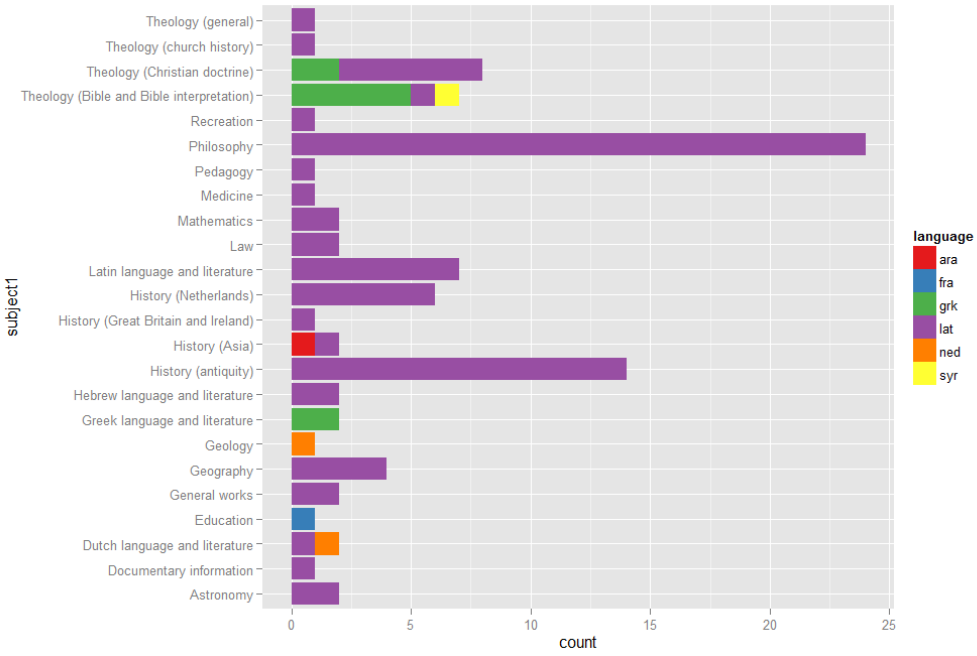


Image 2. Output of listing 2

Listing 3 makes use of a csv file which contains data about the number of types, the number of tokens and the number of sentences. These columns have the names types, tokens and sentences respectively. The titles of these different texts have been used as row names. In R, the row names of a data frame can be obtained using the rownames() function. The first few lines of this sample file look as follows:

```
nrSentences,nouns,adjectives,types,adverbs,tokens,modalVerbs
ARoomWithAView,3520,8727,4537,6967,4742,67066,1317
Emma,4710,20568,12430,7391,12636,160414,4203
```

Some of the data in this set can be represented in scatter plot using the code in listing 3.

```
1.      library("ggplot2")
2.
3.      d <- read.csv("data.csv" , header = TRUE)
4.
5.      p <- ggplot( d , aes( x = ( types/tokens) , y = (
6.      tokens/nrSentences  ) , label = rownames(d) )  ) +
7.      geom_point( size = 6 ) + geom_text( col = "black" ,
8.      hjust = -0.2 , size = 4 ) + xlab("Type-token ratio") +
9.      ylab( "Average length of sentences" )
10.
11.     print(p)
12.
```

Listing 3

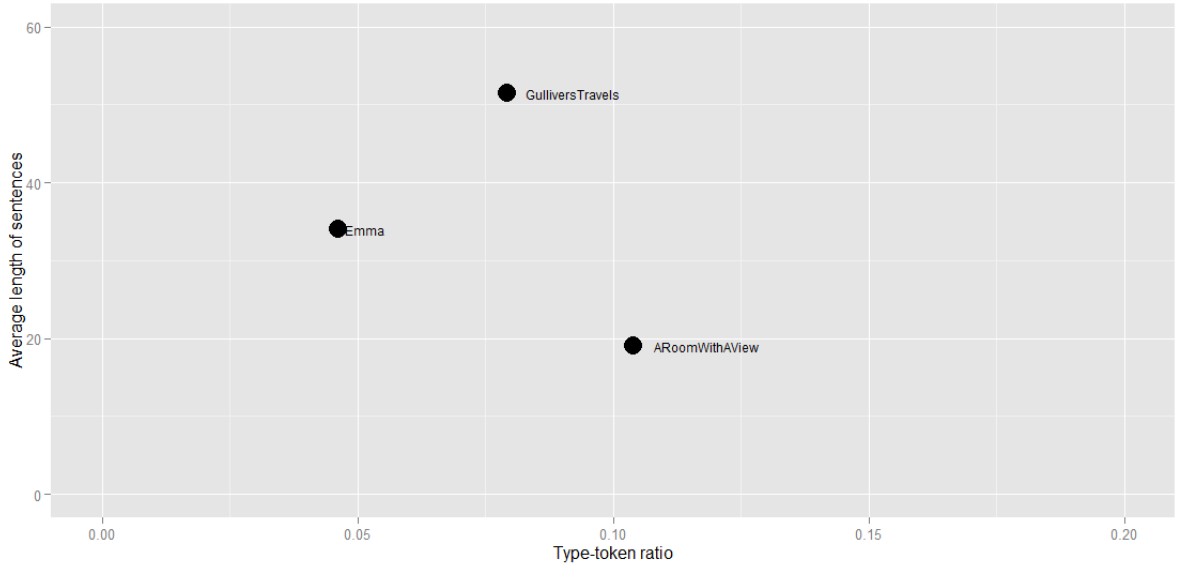


Image 3. Output of listing 3

You can combine different plots in one diagram by making use of the grid.arrange() function. To make use of this function you need to install the "gridExtra" library. Listing 4 shows you how you can use this function.

```
1.      setwd("C:\\Users\\Peter\\Documents\\BDMS\\DTDP\\W62016")
2.      ;
3.
4.      library("ggplot2")
```

```
5.    library("gridExtra")
6.
7.
8.    d <- read.csv("data.csv") ;
9.
10.   p1 <- ggplot( d , aes( x = rownames(d) , y = (
11.   nouns/tokens)  ) ) + geom_bar( stat = "identity" )
12.
13.   p2 <- ggplot( d , aes( x = rownames(d) , y = (
14.   modalVerbs/tokens)  )  ) + geom_bar( stat = "identity" )
15.
16.   grid.arrange(p1, p2 , ncol=2)
```

Listing 4.

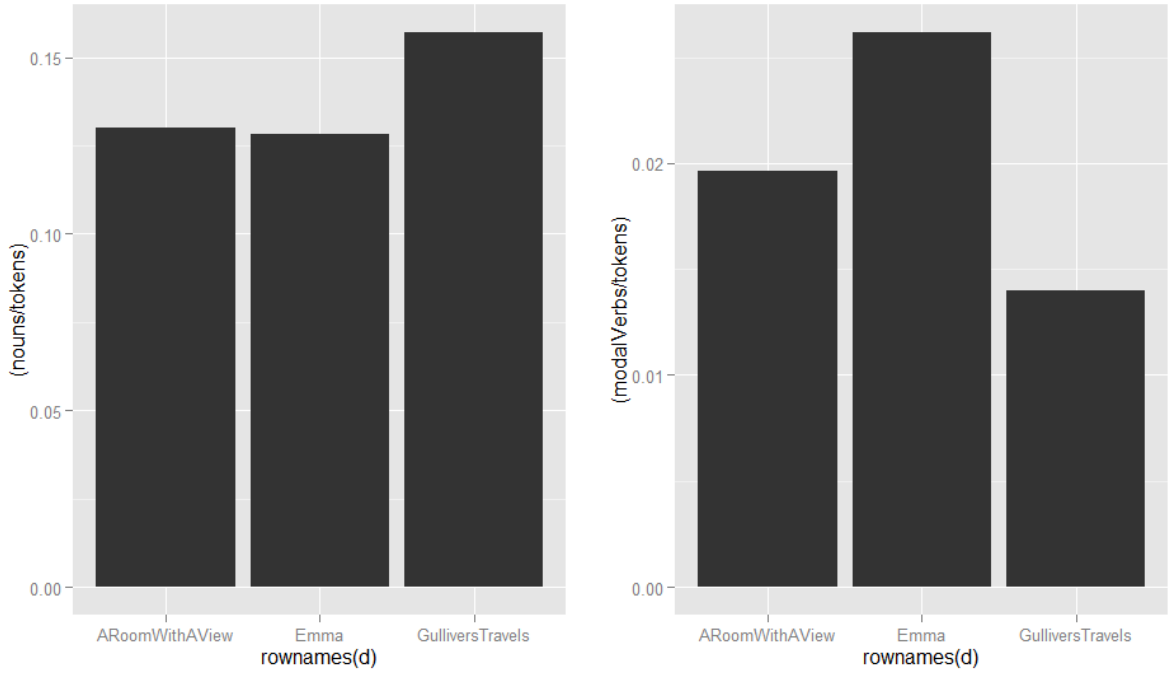


Image 4. Output of listing 4.

Dendrogram

A term-document matrix is a file in which the frequencies are listed of a selected number of terms. In this case, counts are shown of the 25 most frequent terms. These frequencies can be used to calculate overall differences and similarities between texts.

Such clusters can be rendered visually via a dendrogram. In this type of diagram diagram, the texts which are most similar with respect to vocabulary form a single branch, and texts which display fewer similarities do not form a union until a much later stage. As such, the method provides a highly intuitive method of clarifying the differences and the similarities between texts.

A simple dendrogram can be created out of a term-document matrix using the following code:

```
1.      tdm <- read.csv("tdm.csv" , header = TRUE )
2.
3.      dg = hclust(dist( tdm ))
4.      plot(as.dendrogram( dg ) )
```

Listing 10.

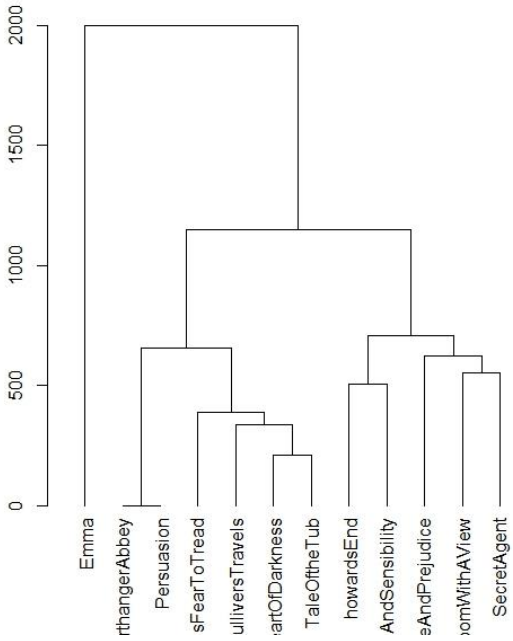


Image 8. Output of listing 10

A dendrogram with a slightly different appearance can be created when you install package “ape”, using the following command:

```
install.packages("ape", dependencies = TRUE )
```

The following code can be used:

```
library("ape" )

cols = c("seagreen", "#4ECDC4", "#1B676B", "#FF6B6B",
"#C44D58")

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

```
dg = hclust(dist( tdm ))

clus5 = cutree(dg, 5)
op = par(bg="ivory")
plot(as.phylo(dg), type = "fan", tip.color = cols[clus5] )
```

Listing 11.

Principal Component Analysis

A PCA diagram can be created and visualised using the following code:

```
file <- read.csv("tdm.csv", head=TRUE )

p <- prcomp( file , center = TRUE, scale. = TRUE)

scores <- data.frame( pca$x[,1:3])

p <- ggplot(  scores , aes( x = PC1 , y = PC2 , label =
rownames(scores) ) ) + geom_point() + geom_text()
```

If necessary, the readability of the diagram can be improved by supplying different values for the xlim and the ylim parameters.