〽️

Advanced Modelling and Simulation PS 7

Reviewed	☐
Files & media	
Multi-select	AMAS
Status	Not started
Type	Exercises

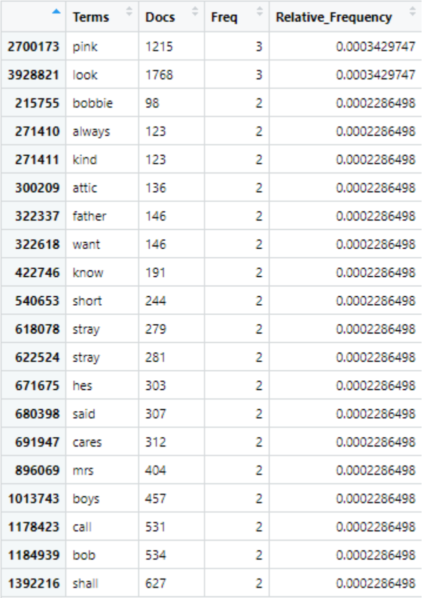
Group 2:

- Endri Lohja 29251
- Ajkana Hima 31033
- Orlando Rexhaj 31034
- Artem Khoripiakov 31024

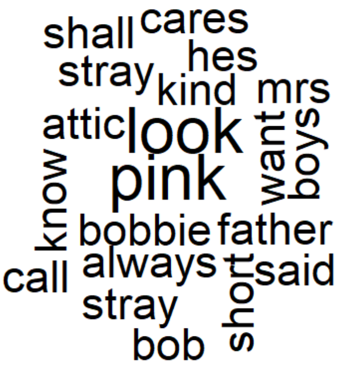
Exercise 1: Text mining with R

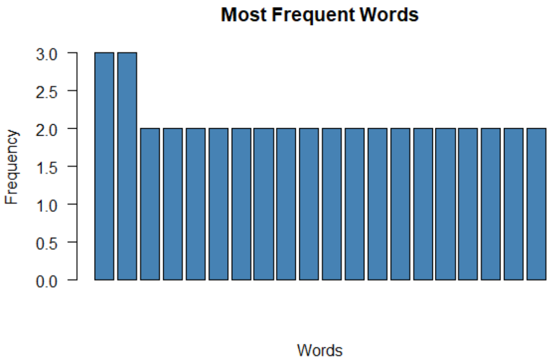
The following is the code in R for text mining, we selected a random book on Gutenberg website and used the libraries tm, SnowballC and worcloud. Using the tm_map function we transformed the text multiple times, as shown in the code, and at the end we were able to get clean text, ready for usage.

```
install.packages(c("tm"))
install.packages(c("SnowballC", "wordcloud"))
# Step 1: Load the required packages
library(tm)
library(SnowballC)
library(wordcloud)


# Step 2: Load the text file using Corpus()
text_file <- system.file("extdata", "C:\\Users\\endri\\amas\\lec7\\book.txt" ,package = "tm")
corpus <- Corpus(VectorSource(readLines("C:\\Users\\endri\\amas\\lec7\\book.txt")))

# Step 3: Preprocess the text data
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stripWhitespace)

# Step 4: Create term-document matrix
tdm <- TermDocumentMatrix(corpus)

# Step 5: Get word frequencies
freq_table <- as.data.frame(as.table(tdm))
freq_table <- freq_table[order(freq_table$Freq, decreasing = TRUE), ]
freq_table$Relative_Frequency <- freq_table$Freq / sum(freq_table$Freq)

# Step 6: Sort the frequency table
freq_table <- freq_table[order(freq_table$Freq, decreasing = TRUE), ]

# Step 7: Select top 20 words
top_words <- freq_table[1:20, ]

# Step 8: Generate bar plot for top 20 words with improved aesthetics
barplot(top_words$Freq, las = 2, names.arg = top_words$Var1, main = "Most Frequent Words",
        xlab = "Words", ylab = "Frequency", col = "steelblue", border = "black")

# Step 9: Write frequency table to CSV file
write.csv(freq_table, "word_frequencies.csv", row.names = FALSE)

# Step 10: Generate word cloud
wordcloud(words = top_words$Var1, freq = top_words$Freq, min.freq = 100, random.order = FALSE)

print(top_words)
words <- c("pink", "look", "bobbie")
freq <- c(1215, 1768, 98)

wordcloud(words = top_words$Terms, freq = top_words$Freq, min.freq = 100, random.order = FALSE)
```



	Terms	Docs	Freq	Relative_Frequency
2700173	pink	1215	3	0.0003429747
3928821	look	1768	3	0.0003429747
215755	bobbie	98	2	0.0002286498
271410	always	123	2	0.0002286498
271411	kind	123	2	0.0002286498
300209	attic	136	2	0.0002286498
322337	father	146	2	0.0002286498
322618	want	146	2	0.0002286498
422746	know	191	2	0.0002286498
540653	short	244	2	0.0002286498
618078	stray	279	2	0.0002286498
622524	stray	281	2	0.0002286498
671675	hes	303	2	0.0002286498
680398	said	307	2	0.0002286498
691947	cares	312	2	0.0002286498
896069	mrs	404	2	0.0002286498
1013743	boys	457	2	0.0002286498
1178423	call	531	2	0.0002286498
1184939	bob	534	2	0.0002286498
1392216	shall	627	2	0.0002286498



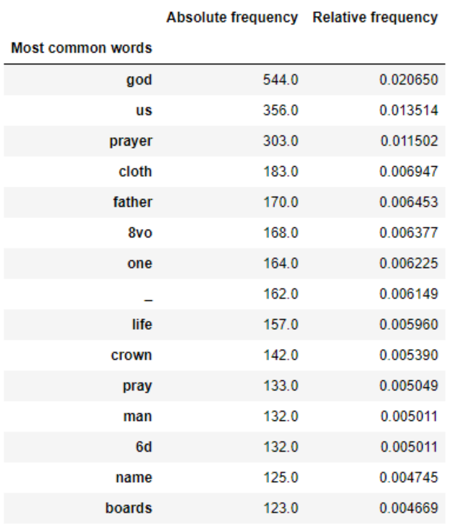


Exercise 2: Text mining with Python

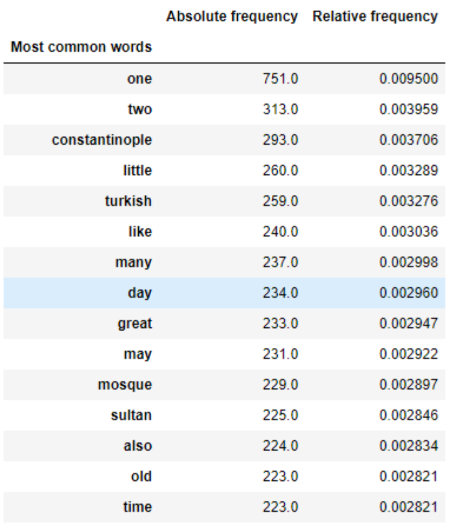
The following code gets two different texts, which were downloaded from Gutenberg website for free, and compares them in terms of most used words. From the original code the books are different and also the variable names have been changed to b1 (referring to book 1) and b2 (referring to book 2). This script also outputs a CSV, comparing two books.

```
import codecs
import re
import copy
import collections
import numpy as np
import pandas as pd
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import WordPunctTokenizer
from __future__ import division
import matplotlib
%matplotlib inline

nltk.download('stopwords')
from nltk.corpus import stopwords

with codecs.open(r'C:/Users/endri/amas/lec7/comp1.txt', "r", encoding="utf-8") as f:
    text_1 = f.read()

with codecs.open(r'C:/Users/endri/amas/lec7/comp2.txt', "r", encoding="utf-8") as f:
    text_2 = f.read()

esw = stopwords.words('english')
```

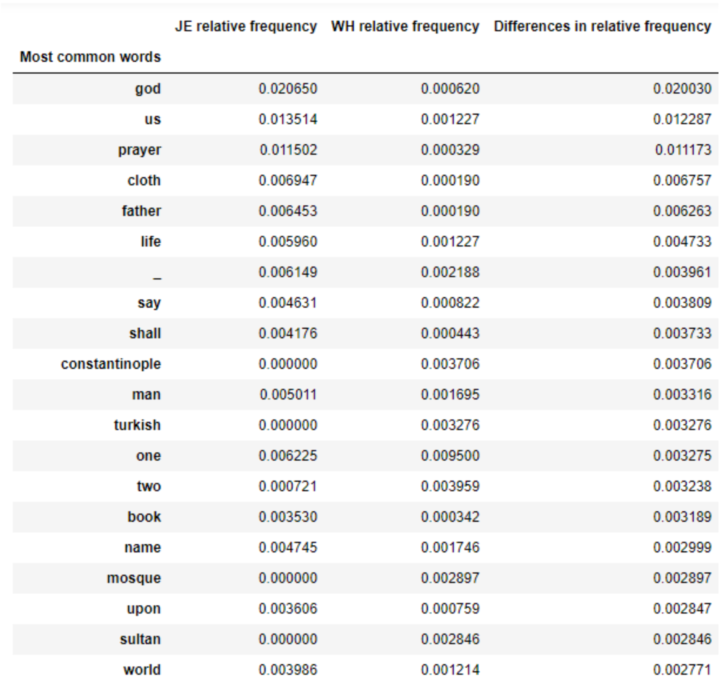
```
esw.append("would")
word_pattern = re.compile("^\w+$")

def get_text_counter(text):
    tokens = WordPunctTokenizer().tokenize(PorterStemmer().stem(text))
    tokens = list(map(lambda x: x.lower(), tokens))
    tokens = [token for token in tokens if re.match(word_pattern, token) and token not in esw]
    return collections.Counter(tokens), len(tokens)

def make_df(counter, size):
    abs_freq = np.array([el[1] for el in counter])
    rel_freq = abs_freq / size
    index = [el[0] for el in counter]
    df = pd.DataFrame(data=np.array([abs_freq, rel_freq]).T, index=index, columns=["Absolute frequency", "Relative frequency"])
    df.index.name = "Most common words"
    return df

b1_counter, b1_size = get_text_counter(text_1)
make_df(b1_counter.most_common(15), b1_size)
b1_df = make_df(b1_counter.most_common(1000), b1_size)
b1_df.to_csv("b1_1000.csv")

b2_counter, b2_size = get_text_counter(text_2)
make_df(b2_counter.most_common(15), b2_size)
b2_df = make_df(b2_counter.most_common(1000), b2_size)
b2_df.to_csv("b2_1000.csv")

all_counter = b2_counter + b1_counter
all_df = make_df(b2_counter.most_common(1000), 1)
most_common_words = all_df.index.values
df_data = []

for word in most_common_words:
    b1_c = b1_counter.get(word, 0) / b1_size
    b2_c = b2_counter.get(word, 0) / b2_size
    d = abs(b1_c - b2_c)
    df_data.append([b1_c, b2_c, d])

diff_df = pd.DataFrame(data=df_data, index=most_common_words, columns=["B1 relative frequency", "B2 relative frequency", "Differences in relative frequency"])
diff_df.index.name = "Most common words"
diff_df.sort_values("Differences in relative frequency", ascending=False, inplace=True)
diff_df.head(20)
diff_df.to_csv("dist_b1b2.csv")
```



	Absolute frequency	Relative frequency
Most common words		
god	544.0	0.020650
us	356.0	0.013514
prayer	303.0	0.011502
cloth	183.0	0.006947
father	170.0	0.006453
8vo	168.0	0.006377
one	164.0	0.006225
_	162.0	0.006149
life	157.0	0.005960
crown	142.0	0.005390
pray	133.0	0.005049
man	132.0	0.005011
6d	132.0	0.005011
name	125.0	0.004745
boards	123.0	0.004669



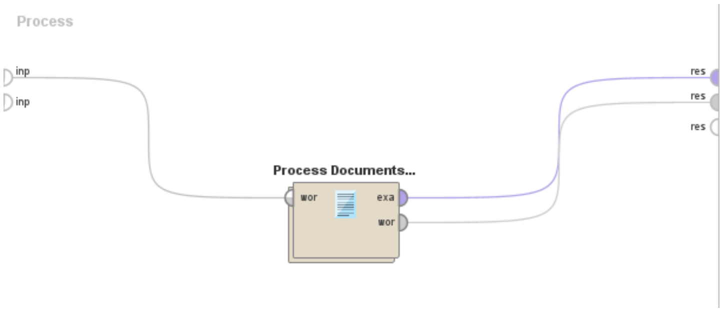
	Absolute frequency	Relative frequency
Most common words		
one	751.0	0.009500
two	313.0	0.003959
constantinople	293.0	0.003706
little	260.0	0.003289
turkish	259.0	0.003276
like	240.0	0.003036
many	237.0	0.002998
day	234.0	0.002960
great	233.0	0.002947
may	231.0	0.002922
mosque	229.0	0.002897
sultan	225.0	0.002846
also	224.0	0.002834
old	223.0	0.002821
time	223.0	0.002821

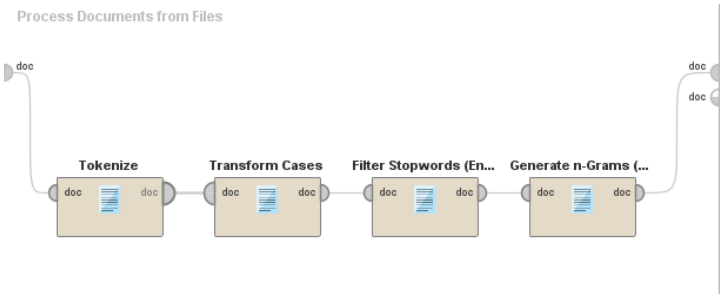


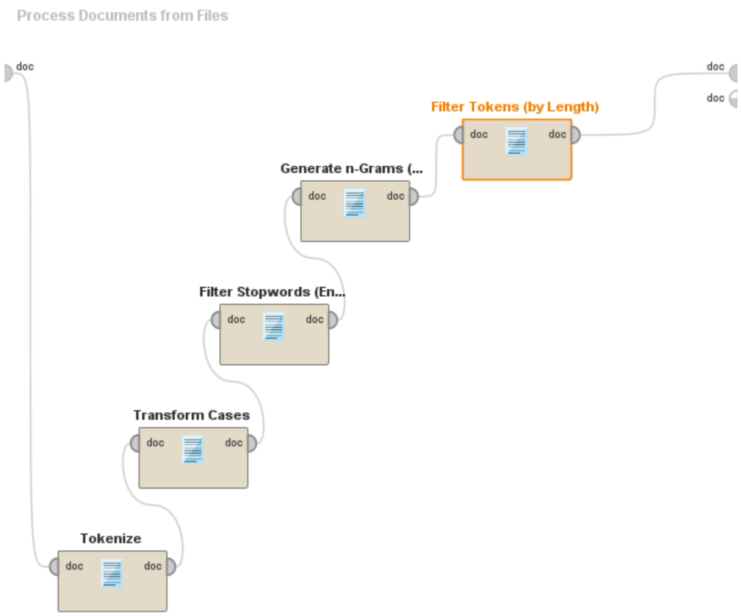
	JE relative frequency	WH relative frequency	Differences in relative frequency
Most common words			
god	0.020650	0.000620	0.020030
us	0.013514	0.001227	0.012287
prayer	0.011502	0.000329	0.011173
cloth	0.006947	0.000190	0.006757
father	0.006453	0.000190	0.006263
life	0.005960	0.001227	0.004733
_	0.006149	0.002188	0.003961
say	0.004631	0.000822	0.003809
shall	0.004176	0.000443	0.003733
constantinople	0.000000	0.003706	0.003706
man	0.005011	0.001695	0.003316
turkish	0.000000	0.003276	0.003276
one	0.006225	0.009500	0.003275
two	0.000721	0.003959	0.003238
book	0.003530	0.000342	0.003189
name	0.004745	0.001746	0.002999
mosque	0.000000	0.002897	0.002897
upon	0.003606	0.000759	0.002847
sultan	0.000000	0.002846	0.002846
world	0.003986	0.001214	0.002771

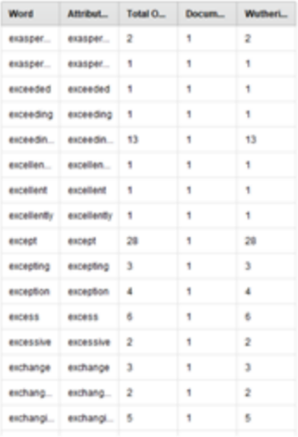
Exercise 3: Text mining with Rapid Orange

1. Install the necessary extensions: "Aylien Text Analysis" and "Text Processing".
2. Create a new repository.
3. Add the "Process Documents from Files" operator from the "Text Processing" extension to the process design.
4. Configure the "Process Documents from Files" operator to specify the location of the text file(s) you want to analyze.
5. Add the "Tokenize" operator to break down words into smaller tokens.
6. Connect the output of the "Tokenize" operator to the input of the "Transform Cases" operator to transform the character cases in the document.
7. Connect the output of the "Transform Cases" operator to obtain the results, which include the word list and example set.
8. Add the "Filter Stopwords (English)" operator to remove English stopwords from the document.
9. Add the "Generate n-Grams (Terms)" operator to generate word groups for better analysis.
10. Add the "Filter Tokens (by length)" operator to filter words based on their character length.
11. Configure the "Filter Tokens (by length)" operator to specify the maximum and minimum character lengths for the words.
12. Run the process to analyze the sentiment of the text.

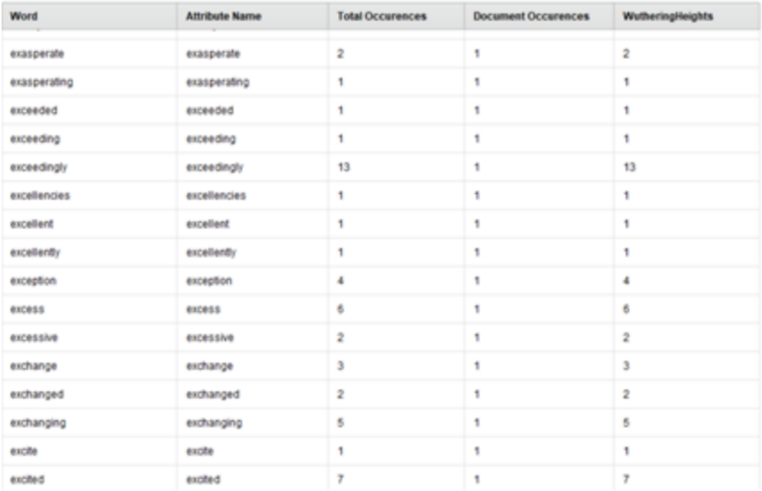








Word	Attribut...	Total O...	Docum...	Wutheri...
exasper...	exasper...	2	1	2
exasper...	exasper...	1	1	1
exceeded	exceeded	1	1	1
exceeding	exceeding	1	1	1
exceedin...	exceedin...	13	1	13
excellen...	excellen...	1	1	1
excellent	excellent	1	1	1
excellently	excellently	1	1	1
except	except	28	1	28
excepting	excepting	3	1	3
exception	exception	4	1	4
excess	excess	6	1	6
excessive	excessive	2	1	2
exchange	exchange	3	1	3
exchang...	exchang...	2	1	2
exchangi...	exchangi...	5	1	5



Word	Attribute Name	Total Occurences	Document Occurences	WutheringHeights
exasperate	exasperate	2	1	2
exasperating	exasperating	1	1	1
exceeded	exceeded	1	1	1
exceeding	exceeding	1	1	1
exceedingly	exceedingly	13	1	13
excellencies	excellencies	1	1	1
excellent	excellent	1	1	1
excellently	excellently	1	1	1
exception	exception	4	1	4
excess	excess	6	1	6
excessive	excessive	2	1	2
exchange	exchange	3	1	3
exchanged	exchanged	2	1	2
exchanging	exchanging	5	1	5
excite	excite	1	1	1
excited	excited	7	1	7