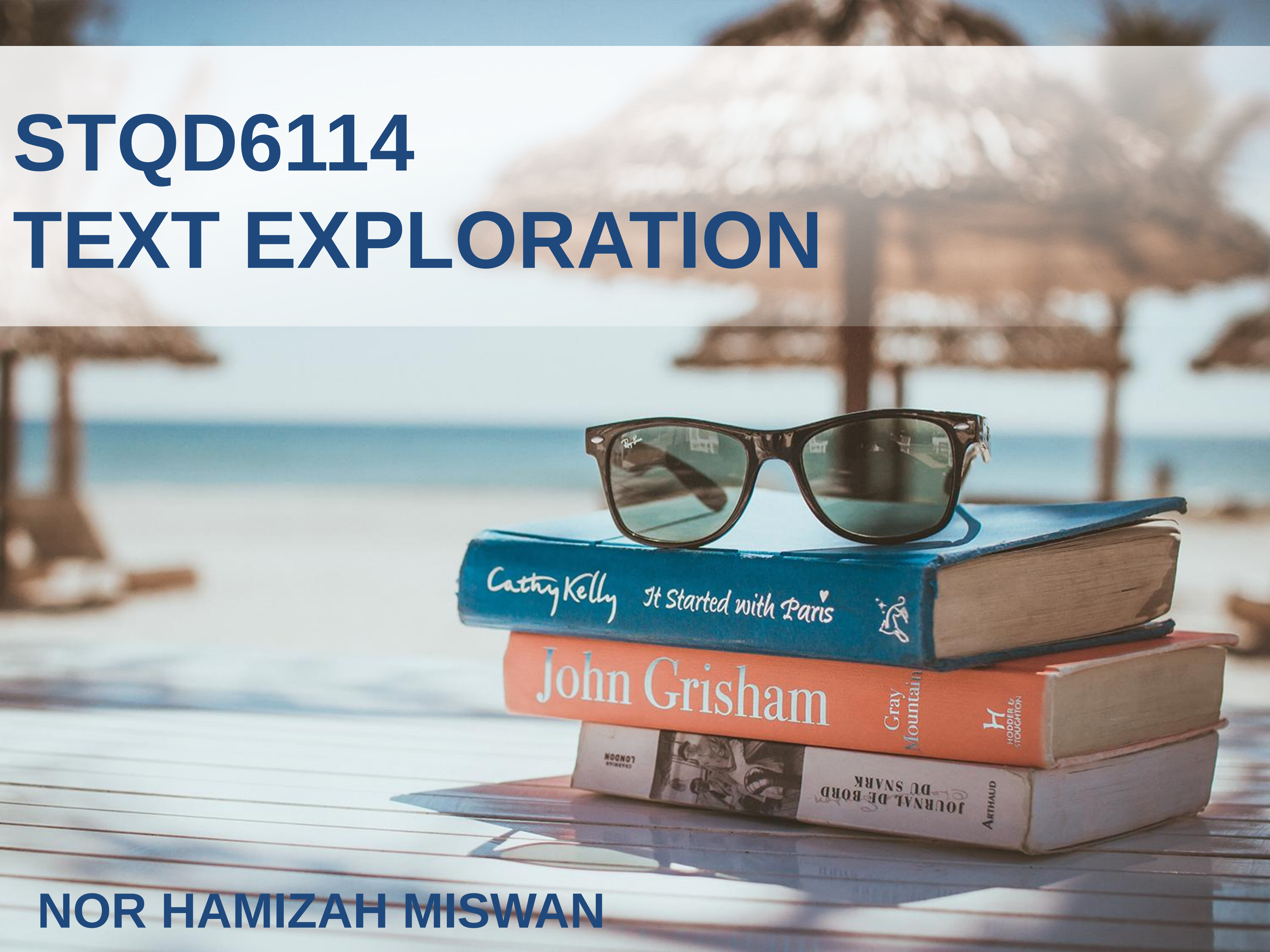
STQD6114
TEXT EXPLORATION



NOR HAMIZAH MISWAN

Regular Expression

- ❖ Regular expression, regex, in R is a sequence of characters (or even one character) that describes a certain pattern found in a text.
- ❖ Regex patterns can be as short as ‘a’.
- ❖ Regex represents a very flexible and powerful tool widely used for processing and mining unstructured text data.
- ❖ For example, they find their application in search engines, lexical analysis, spam filtering, and text editors.

Regular Expression

- Below are the main functions that search for regex matches in a character vector and then do the following:
  - **grep(), grepl()** – return the indices of strings containing a match (grep()) or a logical vector showing which strings contain a match (grepl()).
  - **regexpr(), gregexpr()** – return the index for each string where the match begins and the length of that match. While regexpr() provides this information only for the first match (from the left), gregexpr() does the same for all the matches.
  - **sub(), gsub()** – replace a detected match in each string with a specified string (sub() – only for the first match, gsub() – for all the matches).
  - **regexec()** – works like regexpr() but returns the same information also for a specified sub-expression inside the match.

Working with R regex

- Instead of using the native R functions, a more convenient and consistent way to work with R regex is to use a specialized **stringr** package of the tidyverse collection.
- In the stringr library, all the functions start with str_
- In the stringr functions, we pass in first the data and then a regex, while in the base R functions – just the opposite.

Preprocessing Text Data

- The ultimate goal is to turn a large collection of texts, a corpus, into insights that reveal important and interesting patterns in the data.
- Before we can move into the analysis of text, the unstructured nature of the data means there is a need to pre-process the raw text
    - to transform it to provide some additional structure and clean the text to make it more amenable for further analysis.
- Common practice is using the 'tm' package and create a corpus, the main structure for managing text documents, then conduct a range of text preprocessing tasks.
- These steps will allow you to transform raw text into a more structured and suitable form for analysis.

Preprocessing Text Data

Steps for pre-processing text data:

- ❖ Step 1: Create corpus
- ❖ Step 2: Cleaning raw data
- ❖ Step 3: Tokenization
- ❖ Step 4: Creating the Term-Document Matrix

CREATE CORPUS

- ❖ The tm package uses a so-called corpus as the main structure for managing text documents.
- ❖ A corpus is a collection of documents and is classified into two types based on how the corpus is stored either VCorpus or Pcorpus

Preprocessing Text Data

CLEANING RAW DATA

- ❖ The tm package has several built-in transformation functions that enable pre-processing.
- ❖ Cleaning raw data includes:
    - ➢ The transformation of words (lowering case)
        - Lowering case is helpful to reduce the dimensions by decreasing the size of the vocabulary and is weighed similarly when counting the frequency of words.
    - ➢ The removal of special characters
    - ➢ The removal of stopwords
        - Stopwords such as “the”, “an” etc. do not provide much of valuable information and can be removed from the text.
    - ➢ The removal of extra spaces/punctuation/numbers
- ❖ This procedure might include (depending on the data).

Preprocessing Text Data

TOKENIZATION

- ❖ The process of splitting text into smaller bites called tokens is called tokenization.
- ❖ Each token can then be used as an input into a machine learning algorithm as a feature. The two techniques to normalize tokens are *stemming* and *lemmatization*.

Preprocessing Text Data

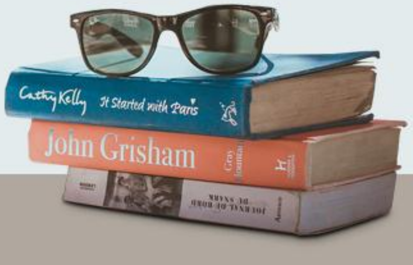
TOKENIZATION

- **Stemming**: the process of getting the root form (stem) of the word by removing and replacing suffixes
  - However, watch out for overstemming or understemming.
  - Overstemming occurs when words are over-truncated which might distort or strip the meaning of the word.
    - Example: the words “university” and “universe” may be reduced to “univers” but this implies both words mean the same which is incorrect.
  - Understemming occurs when two words are stemmed from the same root that is not of different stems.
    - Example: consider the words “data” and “datum” which have “dat” as the stem. Reducing the words to “dat” and “datu”, respectively, results in understemming.

Preprocessing Text Data

TOKENIZATION

- **Lemmatization**: the process of identifying the correct base forms of words using lexical knowledge bases.
- This overcomes the challenge of stemming where words might lose meaning and makes words more interpretable.



Preprocessing Text Data

CREATING THE TERM-DOCUMENT MATRIX

- The corpus can now be represented in the form of a Term-Document Matrix, which represents document vectors in matrix format.
- The rows of this matrix correspond to the terms in the document, columns represent the documents in the corpus and cells correspond to the weights of the terms.

Inspect and Analyze Textual Data

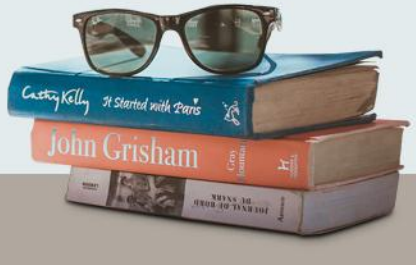
INSPECTING WORD FREQUENCIES

- After creating term-document matrix, we can start analyzing our data
- Let say we want to quickly view the terms with certain frequency, say at least 50.
- This can be done using *findFreqTerms()* for this.
- *findAssocs()* is another useful function to find associations with at least certain percentage of correlation for certain term.

Inspect and Analyze Textual Data

VISUALIZING THE DATA

- ❖ For visualizing text data, a word cloud that gives quick insights into the most frequently occurring words across documents at a glance.
- ❖ We will use the wordcloud2 package for this purpose.



APPENDIX

Without using library

```
##grep functions (without use any package)
ww<-c("statistics","estate","castrate","catalyst","Statistics")
ss<-c("I like statistics","I like bananas","Estates and statues are expensive")

#1st function - grep() -give the location of pattern
grep(pattern="stat",x=ww) #x is the document, will return the location only
grep(pattern="stat",x=ww,ignore.case=T) #ignore the capital/small letter, will return the location only
grep(pattern="stat",x=ww,ignore.case=T,value=T) #ignore the capital/small letter, return to that particular words

#2nd function - grepl() - give logical expression
grepl(pattern="stat",x=ww) #Return true/false
grepl(pattern="stat",x=ss)
```

APPENDIX

Without using library

```
#3rd function - regexpr()
#return a vector of two attributes; position of the first match and its length
#if not, it returns -1
regexpr(pattern="stat",ww)
regexpr(pattern="stat",ss)

#4th function - gregexpr()
gregexpr(pattern="stat",ss)

#5th function - regexec()
regexec(pattern="(st)(at)",ww)

#6th function - sub()
sub("stat","STAT",ww,ignore.case=T)
sub("stat","STAT",ss,ignore.case=T)

#7th function - gsub()
gsub("stat","STAT",ss,ignore.case=T)
```