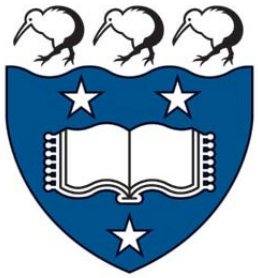
# Text Analytics

Jason Peter Cairns

Supervised by Chris Wild



Bachelor of Science (Honours)
Department of Statistics
The University of Auckland
New Zealand

# Todo list

should this be an abstract? . . . . . 9

# Acknowledgements

# Contents

**Listings** **5**

**Tables** **6**

**Figures** **7**

**1 Introduction** **9**
1.1 Intention . . . . . 9
1.2 Background: Text Analytics (incl. examples) . . . . . 9
1.2.1 common functions: sentiment, summarisation, scoring 9
1.2.2 Existing Systems . . . . . 10
1.3 Background: inZight . . . . . 10
1.3.1 What iNZight is - capabilities, popularity, etc. . . . . 10
1.3.2 how our program fits in - shiny, inzight lite etc. . . . . 10
1.4 Literature Review (existing packages in R) . . . . . 11
1.4.1 Copy over from notes, flesh out a bit . . . . . 11
1.4.2 Praise tidytext book, complain about the package . . 11
1.5 Scope of work . . . . . 11

**2 Text Analytics Prolusion** **12**
2.1 overview . . . . . 12
2.1.1 Explain broadness of term . . . . . 12
2.1.2 compile glossary from terms here . . . . . 12
2.1.3 Areas of text analytics in a data science framework . . 12
2.1.4 what we have done . . . . . 12
2.1.5 what we haven't done . . . . . 12
2.2 terms . . . . . 12
2.2.1 terms and their centrality . . . . . 14
2.2.2 generalisation: n-grams, sentences etc. . . . . 14
2.3 Historical Background . . . . . 14
2.3.1 computer science vs statistics - reflection in data science 14
2.4 Processing . . . . . 14
2.4.1 why process . . . . . 14

2.4.2 stopwords, lemmatisation etc. . . . 14
2.4.3 modelling vs db joins - more info in notes . . . 14
2.5 scores & statistics . . . 14
2.5.1 why compute scores & statistics . . . 14
2.5.2 scoring - tf-idf, word count . . . 14
2.5.3 Suggestions for further research - more on the statistics of words . . . 14
2.5.4 recount the book of John text analysis . . . 14
2.6 Sentiment . . . 14
2.6.1 why sentiment . . . 14
2.6.2 Process of sentiment . . . 14
2.6.3 sentiment modelling vs db joins . . . 14
2.6.4 our implementation and why . . . 14
2.6.5 reviews . . . 14
2.6.6 issues . . . 14
2.7 Summarisation . . . 14
2.7.1 why compute summarisations . . . 14
2.7.2 lexrank, textrank - include notes on lexrank . . . 14
2.7.3 other methods . . . 14
2.7.4 reddit bot example . . . 14
2.8 what we didn't do (yet) . . . 14
2.8.1 topic modelling . . . 14
2.8.2 Term correlation . . . 14
2.8.3 modelling based on linguistic features . . . 14
2.9 Visualisation . . . 14
2.9.1 talk about score vs structure . . . 14
2.9.2 complain about tag clouds . . . 14
2.9.3 talk about ggpage . . . 14
2.9.4 discuss our experimentations with some alternative visualisations . . . 14

**3 Program Structure & Development 15**
3.0.1 why R . . . 15
3.0.2 Why Shiny . . . 15
3.0.3 why tidyverse . . . 15
3.0.4 Git . . . 15
3.0.5 possible future: datatables, futures, etc. . . . 15
3.0.6 why functional . . . 15
3.0.7 Why lossless data . . . 15
3.1 Program Architecture . . . 15
3.1.1 Why structure it like it has been . . . 15
3.1.2 make graph of architecture . . . 15
3.1.3 Describe package and package creation . . . 15

3.1.4 following three sections copy and paste from the notes - buffing up as necessary . . . 15
3.1.5 include screenshots . . . 15
3.2 Preparation . . . 15
3.2.1 Importing . . . 16
3.2.2 Object Preparation . . . 18
3.2.3 Filtering . . . 19
3.2.4 Lemmatisation . . . 19
3.2.5 Stemming . . . 19
3.2.6 Stopwords . . . 20
3.2.7 Formatting . . . 20
3.2.8 Sectioning . . . 22
3.2.9 Grouping . . . 23
3.3 Insight . . . 24
3.3.1 Term Insight . . . 24
3.3.2 Aggregate Insight . . . 28
3.3.3 Wrapper . . . 29
3.4 Visualisation . . . 31
3.4.1 Rank . . . 31
3.4.2 Score . . . 31
3.4.3 Distribution . . . 32
3.4.4 Structure . . . 33
3.4.5 Wrapper . . . 34
3.5 Application . . . 35

**4 Conclusion** **41**
4.1 Summary . . . 41
4.1.1 summarise successes . . . 41
4.1.2 summarise failures . . . 41
4.1.3 general thoughts on the topic . . . 41
4.2 Recommendations . . . 41
4.2.1 educational potential of text analytics . . . 41
4.2.2 what else remains . . . 41

**5 Appendix** **42**

**Glossary** **69**

**Index** **69**

**Bibliography** **70**

# List of source codes


1 Import txt . . . 16
2 Import csv . . . 17
3 Import excel . . . 17
4 Import files . . . 18
5 Prepare text . . . 18
6 Manage stopwords . . . 20
7 Format data . . . 22
8 Detect and add sections . . . 23
9 Determine Term Frequencies . . . 24
10 Get bigrams functionally . . . 24
11 Concatenate walks . . . 25
12 Concatenate walks . . . 25
13 Bigram example . . . 25
14 Get n-grams . . . 26
15 Get n-grams frequencies . . . 26
16 Determine Key Words with Textrank . . . 27
17 Determine Term Sentiments . . . 27
18 Determine the Moving Average Term Sentiment . . . 28
19 Determine the term count over some aggregate . . . 28
20 Determine the Key Sections . . . 29
21 Determine the Aggregate Sentiments . . . 29
22 Insight functions wrapper . . . 30
23 Bind aggregate terms . . . 31
24 Create Bar Plot . . . 32
25 Create Word Cloud . . . 32
26 Create Histogram . . . 33
27 Create Kernel Density Estimate Plot . . . 33
28 Create Time Series Plot . . . 34
29 Create Page View Plot . . . 34
30 Create Visualisation . . . 35
31 UI and Server of Shiny Application . . . 40

# List of Tables

3.1 Primary data structure format . . . . . 18
3.2 Formatting Logic for Stopwords and Lemmatisation . . . . . 21

# List of Figures

# Chapter 1

# Introduction

## 1.1 Intention

Text Analytics serves to glean insights from a body of text. Within the broad category of text analytics, we seek to answer questions about what the text is communicating, what is felt about it, and how this information is structured. In this dissertation, we demonstrate the creation of a user-friendly program to perform text analytics functions using modern R with the Shiny web application framework. In a literate style, we illustrate top-down the structure of such a program, as well as the data structures and computational processes that have established their value for such a program.

should this be an abstract?

## 1.2 Background: Text Analytics (incl. examples)

### 1.2.1 common functions: sentiment, summarisation, scoring

Text Analytics is comprised of a variety of processes and techniques to extract information from text. The text almost always requires some initial processing. Some of the following functions have proven utility, and are expanded upon in chapter 2;

- Sentiment: In order to answer what emotions are conveyed in a text, sentiment analysis is commonly performed. The technique yields some measure of what is represented in an emotional sense by the text, with a range different methods and their associated outputs allowing for different forms of the analysis. Sentiment analysis won't pick up the subtle nuances that a human reader would, but generally gives reasonable output over the extent of a text.
- Associated Words: The meaning of a text is dependent on the struc-

ture between and within words. Looking at how words are associated, through correlation, common sequences, visualisation of sections, etc., allow for a clear high-level assessment of the associations between words. The higher level not only saves individual efforts, but can demonstrate any emergent properties inherent to a text, in a way that a direct reading won't necessarily reveal.

- Summarisation: Automation of an executive summary, or a list of key words, typically falls under the purview of summarisation. The primary aim is to rank and select the most "representative" words or sentences from a text. A few major techniques dominate, being somewhat complex in nature. The results seem to be suprisingly representative of a text.
- Feature Counts: The simplest quantitative measure is very often the most informative; from simple word counts, to selective counts of sentences within groups, counting features can reveal how much written weighting is given to various elements, aiding insight into content, structure, and sentiment simultaneously.

### 1.2.2 Existing Systems

There are several existing systems in the field of Text Analytics. The field was initially nurtured as a sub-field of Computer Science, being computationally-dependent in nature. More recently, there has been increasing statistical interest. The existing systems reflect this; most older text analytics programs were Artificial Intelligence focussed, being experimental in nature, typically composed in lisp. More recently, major statistical programs have been incorporating text analytic features, with a few smaller text-analytic specific programs appearing. SAS, SPSS, and R are all examples of major statistical processing systems, with recent additions of text analytics capabilities. An overview of R packages aiding in text analytics will be given in section 1.4.

## 1.3 Background: inZight

### 1.3.1 What iNZight is - capabilities, popularity, etc.

### 1.3.2 how our program fits in - shiny, inzight lite etc.

Our program will form part of the suite of modules extending iNZight. It provides a simple GUI interface to rapidly perform common text analyses. The primary audience are those learning the fundamentals and potential of text analysis and statistics, which could include students of the traditional text analytics fields of Statistics and Computer Science, but can and should include students of Linguistics, Communications, Law, History, and any

other text-based field. Beyond the educational aspects of the program, it is fully functional for practical use for general text analysis.

## 1.4 Literature Review (existing packages in R)

### 1.4.1 Copy over from notes, flesh out a bit

### 1.4.2 Praise tidytext book, complain about the package

## 1.5 Scope of work

While the total scope possible for text analytics is enormous, our time in creating this program is not. Thus, it is essential that we limit the scope. There are two primary areas with which we created the limitations: Text type, and analysis type.

By limiting the forms of text we work with, we can spend less effort on consideration of every single possible import and transformation case, and more time on the actual design of analysis. The simplest means with which to create the limitation exists in allowing only import of particular text files — in this case, we allow for flat `.txt` files, as well as tabular `.csv` and `.xlsx` files. What we do not provide (though by design leaving open the future possibility of inclusion) is access in-program to common text sources through their API, such as Twitter or Project Gutenberg.

Through focussing on dictionary-based, rather than model-based analyses, we have avoided much of the associated complexities. An example of this follows. It is common to categorise words based on their grammatical category, then use models that take this into account. By avoiding that (again, keeping the design flexible enough to incorporate this in the future), we have been able to get far more functionality implemented in a shorter amount of time, with the analyses still performing soundly. Additionally, we focus on the general audience, as it is typically more advanced, linguistically-trained users who would make intelligent use of such analyses.

# Chapter 2

# Text Analytics Prolusion

## 2.1 overview

Most importantly, words must be extracted, serving as the basic unit of analysis, from which more complex items may be derived.

### 2.1.1 Explain broadness of term

### 2.1.2 compile glossary from terms here

### 2.1.3 Areas of text analytics in a data science framework

### 2.1.4 what we have done

### 2.1.5 what we haven't done

## 2.2 terms

term

### 2.2.1 terms and their centrality

### 2.2.2 generalisation: n-grams, sentences etc.

## 2.3 Historical Background

### 2.3.1 computer science vs statistics - reflection in data science

## 2.4 Processing

### 2.4.1 why process

### 2.4.2 stopwords, lemmatisation etc.

### 2.4.3 modelling vs db joins - more info in notes

## 2.5 scores & statistics

### 2.5.1 why compute scores & statistics

### 2.5.2 scoring - tf-idf, word count

### 2.5.3 Suggestions for further research - more on the statistics of words

### 2.5.4 recount the book of John text analysis

## 2.6 Sentiment

### 2.6.1 why sentiment

### 2.6.2 Process of sentiment

### 2.6.3 sentiment modelling vs db joins

### 2.6.4 our implementation and why

### 2.6.5 reviews

### 2.6.6 issues

## 2.7 Summarisation

### 2.7.1 why compute summarisations

### 2.7.2 lexrank, textrank - include notes on lexrank

### 2.7.3 other methods

### 2.7.4 reddit bot example

## 2.8 what we didn't do (yet)



### 2.8.1 topic modelling

### 2.8.2 Term correlation

### 2.8.3 modelling based on linguistic features

## 2.9 Visualisation

# Chapter 3

# Program Structure & Development

#### 3.0.1 why R

#### 3.0.2 Why Shiny

#### 3.0.3 why tidyverse

#### 3.0.4 Git

#### 3.0.5 possible future: datatables, futures, etc.

#### 3.0.6 why functional

#### 3.0.7 Why lossless data

## 3.1 Program Architecture

#### 3.1.1 Why structure it like it has been

#### 3.1.2 make graph of architecture

#### 3.1.3 Describe package and package creation

#### 3.1.4 following three sections copy and paste from the notes - buffing up as necessary

#### 3.1.5 include screenshots

## 3.2 Preparation

The first step in all text analysis is to import the text data and wrangle it into a data structure suitable for statistical analysis.

### 3.2.1 Importing

Text must first be brought in from an outside source to be useful for the program. The import functions are such that all text from different files exist in dataframes of equivalent structure. The primary differences are that each row of an imported `.txt` file corresponds to a single line, whereas each row of an imported tabular file corresponds to the row of the tabular file. Importantly for tabular files, the column of the text intended for analysis must be given the header of "text" prior to import. This condition will be relaxed later.-

#### Import .txt

The following is the simple function used in the import of `.txt` files:

```
#' Import text file
#'
#' @param filepath a string indicating the relative or absolute
#'     filepath of the file to import
#'
#' @return a [tibble][tibble::tibble-package] of each row
#'   corrresponding to a line of the text file, with the column named
#'   "text"
import_txt <- function(filepath){
  paste(readLines(filepath), collapse = "\n") %>%
    tibble::tibble(text=.)
}
```

Listing 1: Import txt

#### Import .csv

CSV is a plaintext tabular format, with columns typically delimited by commas, and rows by new lines. A particular point of difference in the importation of tabular data and regular plaintext is that the text of interest for the analysis should be (as per tidy principles) in one column, with the rest being additional information that can be used for grouping or filtering. Thus, additional user input is required, in the specification of which column is the text column of interest. The following function is effectively just a wrapper around

```
#' Import csv file
#'
#' @param filepath a string indicating the relative or absolute
#'     filepath of the file to import
#'
#' @return a [tibble][tibble::tibble-package] of each row
#'   corrresponding to a line of the text file, with the column named
#'   "text"
import_csv <- function(filepath){
  readr::read_csv(filepath) ## %>%
    ## table_textcol()
}
```

Listing 2: Import csv

### Import Excel

Unfortunately, much data exists in the Microsoft Excel format, but this must be catered for. As tabular data, it is treated equivalently to csv, with a wrapper around `readr::read_excel()`

```
#' Import excel file
#'
#' @param filepath a string indicating the relative or absolute
#'     filepath of the file to import
#'
#' @return a [tibble][tibble::tibble-package] of each row
#'     corrresponding to a line of the text file, with the column
#'     named "text"
import_excel <- function(filepath){
  readxl::read_excel(filepath) ## %>%
    ## table_textcol()
}
```

Listing 3: Import excel

### Import Wrapper for Arbitrary Number of Files

To have just one function required to import files, we define two functions; one that imports any file, and one making use of it to import multiple files. The import of multiple files is no trivial task; the program must shape them in such a way that they retain identification, and fit into the same datastructure together.
The base wrapper function takes in the filename, and other relevent information, handling the importation process. It also stamps in the name of the document as a column.
The base file import is generalised to multiple files with a multiple import function: this will be our sole import function

```
#' Base case for file import
#'
#' @param filepath string filepath of file for import
#'
#' @return imported file with document id
import_base_file <- function(filepath){
  filetype <- get_filetype(filepath)
  filename <- basename(filepath)
  if (filetype == "csv"){
    imported <- import_csv(filepath)
  } else if (filetype == "xlsx" | filetype == "xls") {
    imported <- import_excel(filepath)
  } else {
    imported <- import_txt(filepath)
  }
  imported %>%
    dplyr::mutate(doc_id = filename)
```

```
}

#' Import any number of files
#'
#' @param filepaths char vector of filepaths
#'
#' @return a [tibble][tibble::tibble-package] imported files with
#'   document id
#'
#' @export
import_files <- function(filepaths){
  filepaths %>%
    purrr::map(import_base_file) %>%
    dplyr::bind_rows()
}
```

Listing 4: Import files

### 3.2.2 Object Preparation

From the imported files, we work at transforming their representations into a lossless and efficient data structure that any analysis can make use of. Our solution to the essential constraint of losslessness is to separate and ID by each word in a dataframe. To do this, we take the line ID, the sentence ID, then the word ID, producing a dataframe that takes the following form:

| line_id | sentence_id | word_id | word |
|---|---|---|---|
| 1 | 1 | 1 | the |
| 1 | 1 | 2 | quick |
| 2 | 1 | 3 | brown |

Table 3.1: Primary data structure format

The reason for the ID columns is the preservation of the structure of the text; If required, the original text can be reconstructed in entirety, sans minor punctuation differences. The following function automatically formats any data of the format returned by the initial import functions.

```
text_prep <- function(data){
    data %>%
      tidytext::unnest_tokens(output = sentence, input = text,
                              token = "sentences", to_lower = FALSE) %>%
      dplyr::mutate(sentence_id = dplyr::row_number()) %>%
      dplyr::group_by(sentence_id, add=TRUE) %>%
      dplyr::group_modify(~ {
          .x %>%
              tidytext::unnest_tokens(output = word, input = sentence,
                                      token = "words", to_lower=FALSE) %>%
              dplyr::mutate(word_id = dplyr::row_number())
      }) %>%
      ungroup_by("sentence_id")
}
```

Listing 5: Prepare text

### 3.2.3 Filtering

Filtering of text is implemented directly with the `dplyr::filter()` function, directly in the server of the shiny app. Filtering can take place multiple times throughout an analysis. The program is flexible enough such that after some initial analytics have been done in the insight layer, preparation can be returned to and the text can be filtered on based on features seen in the analytics.

### 3.2.4 Lemmatisation

Lemmatisation is effectively the process of getting words into dictionary form. It is a very complex, stochastic procedure, as natural languages don't follow consistent and clear rules all the time. Hence, models have to be used. Despite the burden, it is generally worthwhile to lemmatise words for analytics, as there are many cases of words not being considered significant, purely due to taking so many different forms relative to others. Additionally, stopwords work better when considering just the lemmatised form, rather than attempting to exhaustively cover every possible form of a word. textstem is an R package allowing for easy lemmatisation, with it's function `lemmatize_words()` transforming a vector of words into their lemmatised forms (thus being compatible with `mutate()` straight out of the box). We have the lemmatisation in this program managed completely by this single function in the server end of the shiny app. The package Udpipe was another option, but it requires downloading model files, and performs far more in depth linguistic determinations such as parts-of-speech tagging, that we don't need at this point. It is worth noting that, like stopwords, there are different dictionaries available for the lemmatisation process, but we will use the default, as testing has shown it to be the simplest to set up and just as reliable as the rest.

### 3.2.5 Stemming

Stemming is far simpler than lemmatisation, being the removal of word endings. This doesn't require as complex a model, as it is deterministic. It is not quite as effective, as the base word ending is not concatenated back on at the tail, so we are left with word stumps and morphemes. However, it may sometimes be useful when the lemmatisation model isn't working effectively, and textstem provides the capability with `stem_words()`. We have not implemented this yet, as it is not as essential to an analysis when lemmatisation is already available.

### 3.2.6 Stopwords

We make use of dictionary-form stopwords, allowing for the input of both developed lexicons as well as user input. Two functions compose stopwords in the programl `get_sw()`, which gathers user input, queries the selected lexicon and combines the two, and `determine_stopwords()` which adds a boolean `TRUE | FALSE` column to the input dataframe. The following code listing defines `get_sw()` and `determine_stopwords()`:

```
#' Gets stopwords from a default list and user-provided list
#'
#' @param lexicon a string name of a stopword list, one of "smart",
#'     "snowball", or "onix"
#'
#' @param addl user defined character vector of additional stopwords,
#'     each element being a stopword
#'
#' @return a [tibble][tibble::tibble-package] with one column named "word"
get_sw <- function(lexicon = "snowball", addl = NA){
  addl_char <- as.character(addl)
  tidytext::get_stopwords(source = lexicon) %>%
    dplyr::select(word) %>%
    dplyr::bind_rows(., tibble::tibble(word = addl_char)) %>%
    stats::na.omit() %>%
    purrr::as_vector() %>%
    tolower() %>%
    as.character()
}

#' determine stopword status
#'
#' @param .data vector of words
#'
#' @param ... arguments of get_sw
#'
#' @return a [tibble][tibble::tibble-package] equivalent to the input
#'   dataframe, with an additional stopword column
#'
#' @export
determine_stopwords <- function(.data, ...){
  sw_list <- get_sw(...)
  .data %in% sw_list
}
```

Listing 6: Manage stopwords

### 3.2.7 Formatting

The final component in preparation is to format the prepared object with the correct attributes to have formatting automated. We define a wrapper that takes all combinations of stopwords and lemmatisation options and intelligently connects them for the "insight column" in a dataframe, which the insight is performed upon. For the purpose of standard interoperability, e.g., with ggpage, we name this column "text".
At the heart of this function is an `ifexp()` that encodes the following lo-

gic involving the interaction of stopwords and lemmatisation, to enable the correct output text based on stopword and lemmatisation options;

| | Stopwords True | Stopwords False |
|---|---|---|
| Lemmatise True | Lemmatise, determine stopwords on lemmatisation, perform insight on lemmas sans stopwords | Lemmatise, perform insight on lemmas |
| Lemmatise False | Determine stopwords on original words (no lemmatisation), perform insight on words sans stopwords | Perform insight on original words |

Table 3.2: Formatting Logic for Stopwords and Lemmatisation

Based on the combination, stopword filtering and lemmatisation take place inside the function, defined as the following:

```
#' takes imported one-line-per-row data and prepares it for later analysis
#'
#' @param .data tibble with one line of text per row
#'
#' @param lemmatize boolean, whether to lemmatize or not
#'
#' @param stopwords boolean, whether to remove stopwords or not
#'
#' @param sw_lexicon string, lexicon with which to remove stopwords
#'
#' @param addl_stopwords char vector of user-supplied stopwords
#'
#' @return a [tibble][tibble::tibble-package] with one token per line,
#'   stopwords removed leaving NA values, column for analysis named
#'   "text"
#'
#' @export
format_data <- function(.data, lemmatize=TRUE, stopwords=TRUE,
                        sw_lexicon="snowball", addl_stopwords=NA){
  formatted <- .data %>%
    text_prep()

  text <- ifexp(lemmatize,
                ifexp(stopwords,
                      dplyr::mutate(formatted,
                                    lemma = tolower(textstem::lemmatize_words(word)),
                                    stopword = determine_stopwords(lemma,
                                                                   sw_lexicon,
                                                                   addl_stopwords),
                                    text = dplyr::if_else(stopword,
                                                          as.character(NA),
                                                          lemma)),
                      dplyr::mutate(formatted,
                                    lemma = tolower(textstem::lemmatize_words(word)),
                                    text = lemma)),
                ifexp(stopwords,
                      dplyr::mutate(formatted,
                                    stopword = determine_stopwords(word,
```

```
                                                                    sw_lexicon,
                                                                    addl_stopwords),
                                        text = dplyr::if_else(stopword,
                                                              as.character(NA),
                                                              word)),
                                dplyr::mutate(formatted, text = word)))
  return(text)
}
```

Listing 7: Format data

### 3.2.8 Sectioning

Plaintext, as might exist as a Gutenberg Download, differs from more complex representations in many ways, including a lack of sectioning — for example, chapters require a specific search in order to jump to them. Here, I compose a closure that searches and sections text based on a Regular Expression intended to capture a particular section. Several functions are created from that. At a later date, advanced users could be given the option to compose their own regular expressions for sectioning.

```
#' creates a search closure to section text
#'
#' @param search a string regexp for the term to seperate on, e.g. "Chapter"
#'
#' @return closure over search expression
get_search <- function(search){
  function(.data){
    .data %>%
      stringr::str_detect(search) %>%
      purrr::accumulate(sum, na.rm=TRUE)
    }
}

#' sections text based on chapters
#'
#' @param .data vector to section
#'
#' @return vector of same length as .data with chapter numbers
#'
#' @export
get_chapters <- get_search("^[\\s]*[Cc][Hh][Aa]?[Pp][Tt]([Ee][Rr])?")

#' sections text based on parts
#'
#' @param .data vector to section
#'
#' @return vector of same length as .data with part numbers
#'
#' @export
get_parts <- get_search("^[\\s]*[Pp]([Aa][Rr])?[Tt]")

#' sections text based on sections
#'
#' @param .data vector to section
#'
#' @return vector of same length as .data with section numbers
#'
```

```r
#' @export
get_sections <- get_search("^[\\s]*([Ss][Ss])|([Ss][Ee][Cc][Tt][Ii][Oo][Nn])")

#' sections text based on cantos
#'
#' @param .data vector to section
#'
#' @return vector of same length as .data with canto numbers
#'
#' @export
get_cantos <- get_search("(?i)canto (XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$")

#' sections text based on book
#'
#' @param .data vector to section
#'
#' @return vector of same length as .data with book numbers
#'
#' @export
get_books <- get_search("(?i)book$")

#' Adds section column to dataframe
#'
#' @param .data dataframe formatted as per output of prep process
#'
#' @param section_by character name of what to section over
#'
#' @return input dataframe with additional section column
#'
#' @export
section <- function(.data, section_by){
    sec_table <- list("chapter" = get_chapters,
                      "part" = get_parts,
                      "section" = get_sections,
                      "canto" = get_cantos,
                      "book" = get_books)
    .data %>%
        dplyr::mutate(!! section_by := sec_table[[section_by]](word))
}
```

Listing 8: Detect and add sections

### 3.2.9 Grouping

Grouping is an essential, killer feature of our app. The implementation is to run a `dplyr::group_by()` command in the shiny server on the prepared object, over user-specified groups, and all further insights and visualisations are performed groupwise. This allows for immediate and clear comparisons between groups.
Like filtering, after some initial analytics have been done in the insight layer, preparation can be returned to and the text can be grouped on based on the analytics.

## 3.3 Insight

### 3.3.1 Term Insight

#### Term Frequency

```r
#' Determine term frequency
#'
#' @param .data character vector of terms
#'
#' @return numeric vector of term frequencies
#'
#' @export
term_freq <- function(.data){
  .data %>%
    tibble::enframe() %>%
  dplyr::add_count(value) %>%
  dplyr::mutate(n = dplyr::if_else(is.na(value),
                    as.integer(NA),
                    n))  %>%
  dplyr::pull(n)
}
```

Listing 9: Determine Term Frequencies

#### n-Grams

```r
#' Determine bigrams
#'
#' @param .data character vector of words
#'
#' @return character vector of bigrams
#'
 get_bigram <- function(.data){
    concat_walk(.data, c(.data[-1], NA))
}
```

Listing 10: Get bigrams functionally

```r
#' concat list 1 and 2 at index, skipping NA values
#'
#' @param i numeric index to assess index at
#'
#' @param list1 list or vector for first token
#'
#' @param list2 list or vector for second token
#'
#' @return paste of list1 and list2 at index i, skipping NA's
concat_walk_i <- function(i, list1, list2){
  ifelse(length(list2) < i | is.na(list1[i]),
          as.character(NA),
  ifelse(!(is.na(list1[i]) | is.na(list2[i])),
         paste(list1[i], list2[i]),
         concat_walk_i(i,list1, list2[-1])))
}

#' concat list 1 and 2, moving past NA values
#'
```

```r
#' @param list1 list or vector for first bigram token
#'
#' @param list2 list or vector for second bigram token
#'
#' @return paste of list1 and list2, skipping NA's
concat_walk <- function(list1, list2){
    stopifnot(length(list1) == length(list2))
    sapply(seq_along(list1), concat_walk_i, list1, list2)
}
```

Listing 11: Concatenate walks

```r
#' concat list 1 and 2 at index, skipping NA values
#'
#' @param i numeric index to assess index at
#'
#' @param list1 list or vector for first token
#'
#' @param list2 list or vector for second token
#'
#' @return paste of list1 and list2 at index i, skipping NA's
concat_walk_i <- function(i, list1, list2){
  ifelse(length(list2) < i | is.na(list1[i]),
           as.character(NA),
  ifelse(!(is.na(list1[i]) | is.na(list2[i])),
         paste(list1[i], list2[i]),
         concat_walk_i(i,list1, list2[-1])))
}

#' concat list 1 and 2, moving past NA values
#'
#' @param list1 list or vector for first bigram token
#'
#' @param list2 list or vector for second bigram token
#'
#' @return paste of list1 and list2, skipping NA's
concat_walk <- function(list1, list2){
    stopifnot(length(list1) == length(list2))
    sapply(seq_along(list1), concat_walk_i, list1, list2)
}
```

Listing 12: Concatenate walks

```r
x <- c(1, 2, NA, 4, 5, NA, 7, NA)
get_bigram(x)
```

Listing 13: Bigram example

```r
#' Returns the n-grams, skipping NA values
#'
#' @param .data vector to get n-grams from
#'
#' @param n number of n-grams to attain
#'
#' @return n-gram vector without NA values
#'
#' @export
get_ngram <- function(.data, n){
    main_n <- n
    ngrams <- rep(NA_character_, length(.data))
    for (i in seq_along(.data)){
```

```
        if (i - 1 >  length(.data) - n) break
        if (is.na(.data[i])){
            next
        } else {
            ngram <- .data[i]
        }
        j <- i + 1
        while(n > 1){
            if (j >  length(.data)){
                ngram <- NA_character_
                break
            }
            if (is.na(.data[j])){
                j <- j + 1
            } else {
                ngram <- paste(ngram, .data[j])
                n <- n - 1
                j <- j + 1
            }
        }
        ngrams[i] <- ngram
        n <- main_n
    }
    return(ngrams)
}
```

Listing 14: Get n-grams

```
#' NOT FOR PRODUCTION - STILL IN TESING. Returns the count of n-grams, skipping NA values
#'
#' @param .data vector to get n-grams from
#'
#' @param n number of n-grams to attain
#'
#' @return count of each associated n-gram
#'
#' @export
ngram_freq <- function(.data, n){
  term_freq(get_ngram(.data, n))
}
```

Listing 15: Get n-grams frequencies

## Key Words

```
#' Determine textrank score for vector of words
#'
#' @param .data character vector of words
#'
#' @param summ_method method to use for summarisation: textrank or
#'     lexrank. Doesn't do anything yet
#'
#' @return vector of scores for each word
#'
#' @export
keywords_tr <- function(.data, summ_method){
  relevent <- !is.na(.data)
  tr <- textrank::textrank_keywords(.data, relevent, p=+Inf)
  score <- tr$pagerank$vector %>% tibble::enframe()
  data <- .data %>% tibble::enframe("number", "name")
```

```
  dplyr::full_join(data, score, by="name") %>%
    dplyr::pull(value)
}
```

Listing 16: Determine Key Words with Textrank

### Term Sentiment

```
#' Determine sentiment of terms
#'
#' @param .data vector of terms
#'
#' @param lexicon sentiment lexicon to use, based on the corpus
#'   provided by tidytext
#'
#' @return vector with sentiment score of each word in the vector
#'
#' @export
term_sentiment <- function(.data, lexicon="afinn"){
  data <- tibble::enframe(.data, "number", "word")
  tidytext::get_sentiments(lexicon) %>%
    dplyr::select(word, value) %>%
    dplyr::right_join(data, by="word") %>%
    dplyr::pull(value)
}
```

Listing 17: Determine Term Sentiments

### Moving Average Term Sentiment

```
#' Determine the lagged sentiment of terms
#'
#' @param .data vector of terms
#'
#' @param lexicon sentiment lexicon to use, based on the corpus
#'     provided by tidytext
#'
#' @param lag how many (inclusive) terms to compute statistic over
#'
#' @param statistic base statistic used to summarise the data, capable
#'     of taking an na.rm argument
#'
#' @return vector with lagged sentiment score of each term in the input vector
#'
#' @export
ma_term_sentiment <- function(.data, lexicon="afinn", lag = 10, statistic = mean){
    sents <- term_sentiment(.data, lexicon)
    ## lagged_sents <- rep(NA, length(sents))
    ## for (i in seq(lag, length(sents))){
    ##     lagged_sents[i] <- statistic(sents[(seq(i - lag + 1, i))], na.rm = TRUE)
    ## }
    ## lagged_sents
    c(rep(NA, lag - 1),
      sapply(seq(lag, length(sents)),
             function(i){x <- statistic(sents[(seq(i - lag + 1, i))],
                                        na.rm = TRUE)
                                        ifelse(is.nan(x),
                                               NA,
```

```
                                                x)
            }))
}
```

Listing 18: Determine the Moving Average Term Sentiment

### 3.3.2 Aggregate Insight

#### Term Count

```
#' Determine the number of terms at each aggregate level
#'
#' @param .data character vector of terms
#'
#' @param aggregate_on vector to split .data on for insight
#'
#' @return vector of number of terms for each aggregate level, same
#'   length as .data
#'
#' @export
term_count <- function(.data, aggregate_on){
  split(.data, aggregate_on) %>%
    purrr::map(function(x){rep(length(x), length(x))}) %>%
    dplyr::combine()
}
```

Listing 19: Determine the term count over some aggregate

#### Key Sections

```
#' get score for key sentences as per Lexrank
#'
#' @param .data character vector of words
#'
#' @param summ_method method to use for summarisation: textrank or
#'     lexrank. Doesn't do anything yet
#'
#' @param aggregate_on vector to aggregate .data over; ideally, sentence_id
#'
#' @return lexrank scores of aggregates
#'
#' @export
key_aggregates <- function(.data, aggregate_on, summ_method){
  ## prepare .data for lexrank
  base <-  tibble::tibble(word = !! .data, aggregate = aggregate_on)
  aggregated <- base %>%
    dplyr::group_by(aggregate) %>%
    stats::na.omit() %>%
    dplyr::summarise(sentence = paste(word, collapse = " ")) %>%
    dplyr::mutate(sentence = paste0(sentence, "."))
  ## lexrank
  lr <- aggregated %>%
    dplyr::pull(sentence) %>%
    lexRankr::lexRank(., n=length(.),removePunc = FALSE, returnTies = FALSE,
             removeNum = FALSE, toLower = FALSE, stemWords = FALSE,
             rmStopWords = FALSE, Verbose = TRUE)
  ## match lexrank output to .data
  lr %>%
```

```
    dplyr::distinct(sentence, .keep_all = TRUE) %>%
    dplyr::full_join(aggregated, by="sentence") %>%
    dplyr::full_join(base, by="aggregate") %>%
    dplyr::arrange(aggregate) %>%
    dplyr::pull(value)
}
```

Listing 20: Determine the Key Sections

### Aggregate Sentiment

```
#' Get statistics for sentiment over some group, such as sentence.
#'
#' @param .data character vector of words
#'
#' @param aggregate_on vector to aggregate .data over; ideally,
#'   sentence_id, but could be chapter, document, etc.
#'
#' @param lexicon as per term sentiment
#'
#' @param statistic function that accepts na.rm argument; e.g. mean,
#'   median, sd.
#'
#' @return sentiment of same length as input vector aggregated over the aggregate_on vector
#'
#' @export
aggregate_sentiment <- function(.data, aggregate_on, lexicon = "afinn", statistic = mean){
  tibble::enframe(.data, "nil1", "word") %>%
    dplyr::bind_cols(tibble::enframe(aggregate_on, "nil2", "aggregate")) %>%
    dplyr::select(word, aggregate) %>%
    dplyr::mutate(sentiment = term_sentiment(word, lexicon)) %>%
    dplyr::group_by(aggregate) %>%
    dplyr::mutate(aggregate_sentiment =
                    (function(.x){
                      rep(statistic(.x, na.rm = TRUE), length(.x))
                    })(sentiment)) %>%
    dplyr::pull(aggregate_sentiment)
}
```

Listing 21: Determine the Aggregate Sentiments

### Word Correlation

### Term Frequency — Inverse Document Frequency

### Topic Modelling

### 3.3.3 Wrapper

```
#' perform group-aware term operations on the data
#'
#' @param .data dataframe of terms as per output of format_data
#'
#' @param operations character vector of term operations to perform
#'
#' @param ... additional arguments to the operation - only sensible for singular operations
#'
```

```
#' @return .data with operation columns added
#'
#' @export
get_term_insight <- function(.data, operations, ...){
    opstable <- list("Term Frequency" = term_freq,
                     "n-gram Frequency" = ngram_freq,
                     "n-grams" = get_ngram,
                     "Key Words" = keywords_tr,
                     "Term Sentiment" = term_sentiment,
                     "Moving Average Term Sentiment" = ma_term_sentiment)
    ops <- opstable[operations]
    lapply(seq(length(ops)),
           function(x){
               name <- dplyr::sym(names(ops[x]))
               operation <- ops[x][[1]]
               df <- dplyr::mutate(.data,
                                   !!name := operation(text, ...))
               df[names(ops[x])]
           }) %>%
        dplyr::bind_cols(.data, .)
}

#' perform group-aware aggregate operations on the data
#'
#' @param .data dataframe of terms as per output of format_data
#'
#' @param operations character vector of operations to perform
#'
#' @param aggregate_on character name of the column to perform aggregate operations on
#'
#' @param ... additional arguments to the operation - only sensible for singular operations
#'
#' @return .data with operation columns added
#'
#' @export
get_aggregate_insight <- function(.data, operations, aggregate_on, ...){
    opstable <- list("Aggregated Term Count" = term_count,
                     "Key Sections" = key_aggregates,
                     "Aggregated Sentiment" = aggregate_sentiment,
                     "Bound Aggregates" = bind_aggregation)
    ops <- opstable[operations]
    lapply(seq(length(ops)),
           function(x){
               name <- dplyr::sym(names(ops[x]))
               operation <- ops[x][[1]]
               agg_on <- dplyr::sym(aggregate_on)
               df <- if (names(ops[x]) == "Bound Aggregates"){
                         dplyr::mutate(.data,
                                       !!name := operation(word, !! agg_on))
                     } else {
                         dplyr::mutate(.data,
                                       !!name := operation(text, !! agg_on, ...))
                     }
               df[names(ops[x])]
           }) %>%
        dplyr::bind_cols(.data, .)
}
```

Listing 22: Insight functions wrapper

```
#' bind aggregate terms together
#'
```

```r
#' @param data vector of terms
#'
#' @param aggregate_on vector of aggregations
#'
#' @return data with every aggregation bound, as in a sentence
#'
#'
bind_aggregation <- function(data, aggregate_on){
    tibble::tibble(data, agg = aggregate_on) %>%
        dplyr::group_by(agg) %>%
        dplyr::mutate(bound = paste(data, collapse = " ")) %>%
        dplyr::pull(bound)
}
```

Listing 23: Bind aggregate terms

## 3.4 Visualisation

### 3.4.1 Rank

### 3.4.2 Score

#### Bar Plot

```r
#' output a ggplot column graph of the top texts from some insight function
#'
#' @param .data a dataframe containing "text" and insight columns as
#'     per the output of the get_(term|aggregate)_insight wrapper
#'     function
#'
#' @param y symbol name of the column insight was
#'     outputted to
#'
#' @param x symbol name of column for insight labels
#'
#' @param n number of bars to display
#'
#' @param desc bool: show bars in descending order
#'
#' @export
score_barplot <- function(.data, y, n = 15,
                                x = text, desc = FALSE){
    wrap <- 50
    text <- dplyr::enquo(x)
    insight_col <- dplyr::enquo(y)
    .data %>%
        dplyr::distinct(!! text, .keep_all=TRUE) %>%
        dplyr::arrange(dplyr::desc(!! insight_col)) %>%
        dplyr::group_modify(~{.x %>% head(n)}) %>%
        dplyr::ungroup() %>%
        dplyr::mutate(text = forcats::fct_reorder(shorten(!! text, wrap),
                                                  !! insight_col,
                                                  .desc = desc)) %>%
        ggplot2::ggplot(ggplot2::aes(x = text)) +
        ggplot2::geom_col(ggplot2::aes(y = !! insight_col)) +
        ggplot2::coord_flip()
}

```

```r
#' Shorten some text up to n characters
#'
#' @param .data character vector
#'
#' @param n wrap length of text
#'
#' @return shortened form of .data
#'
shorten <- function(.data, n){
    ifelse(nchar(.data) > n,
           paste(substr(.data, 1, n), "...", sep = ""),
           .data)
}
```

Listing 24: Create Bar Plot

### Word Cloud

```r
#' output a ggplot wordcloud graph of the top texts from some insight function
#'
#' @param .data a dataframe containing "text" and insight columns as
#'     per the output of the get_(term|aggregate)_insight wrapper
#'     function
#'
#' @param y symbol name of the column insight was
#'     outputted to
#'
#' @param x symbol name of column for insight labels
#'
#' @param n number of words to display
#'
#' @param shape character: shape of the wordcloud
#'
#' @export
score_wordcloud <- function(.data, y, n = 15,
                            x = text, shape = "circle"){
    text <- dplyr::enquo(x)
    insight_col <- dplyr::enquo(y)
    .data %>%
        dplyr::distinct(!! text, .keep_all=TRUE) %>%
        dplyr::arrange(dplyr::desc(!! insight_col)) %>%
        dplyr::group_modify(~{.x %>% head(n)}) %>%
        dplyr::ungroup() %>%
        ggplot2::ggplot(ggplot2::aes(label = stringr::str_wrap(!! text, 30), size = !! insight_col)) +
        ggwordcloud::geom_text_wordcloud(shape = shape, rm_outside = TRUE) +
        ggplot2::scale_size_area(max_size = 24)
}
```

Listing 25: Create Word Cloud

## 3.4.3 Distribution

### Histogram

```r
#' output a histogram of the distribution of some function of words
#'
#' @param .data the standard dataframe, modified so the last column
#'     is the output of some insight function (eg. output from
```

```
#'     term_freq)
#'
#' @param col_name symbol name of the column insight was
#'     performed on
dist_hist <- function(.data, col_name){
    q_col_name <- dplyr::enquo(col_name)
    .data %>%
        ggplot2::ggplot(ggplot2::aes(x = !! q_col_name)) +
        ggplot2::geom_histogram()
}
```

Listing 26: Create Histogram

### Density

```
#' output a histogram of the distribution of some function of words
#'
#' @param .data the standard dataframe, modified so the last column
#'     is the output of some insight function (eg. output from
#'     term_freq)
#'
#' @param col_name symbol name of the column insight was
#'     performed on
dist_density <- function(.data, col_name){
    q_col_name <- dplyr::enquo(col_name)
    .data %>%
        ggplot2::ggplot(ggplot2::aes(x = !! q_col_name)) +
        ggplot2::geom_density()
}
```

Listing 27: Create Kernel Density Estimate Plot

### Box Plot

## 3.4.4 Structure

### Time Series

```
#' output a ggplot time series plot of some insight function
#'
#' @param .data a dataframe containing "text" and insight columns as
#'     per the output of the get_(term|aggregate)_insight wrapper
#'     function
#'
#' @param y symbol name of the column insight was
#'     outputted to
#'
#' @export
  struct_time_series <- function(.data, y){
    q_y <- dplyr::enquo(y)
    .data %>%
        dplyr::filter(!is.na(!! q_y)) %>%
        dplyr::mutate(term = seq_along(!! q_y)) %>%
        ggplot2::ggplot(ggplot2::aes(term, !! q_y)) +
        ggplot2::geom_line(na.rm = TRUE)
}
```

Listing 28: Create Time Series Plot

### Page View

```
#' Colours a ggpage based on an insight function
#'
#' @param .data a dataframe containing "word" and insight columns as
#'     per the output of the get_(term|aggregate)_insight wrapper
#'     function
#'
#' @param col_name symbol name of the insight column intended to
#'     colour plot
#'
#' @param num_terms the number of terms to visualise
#'
#' @param term_index which term to start the visualisation from
#'
#' @param palette determine coloration of palette (not yet implemented)
#'
#' @return ggplot object as per ggpage
#'
#' @export
struct_pageview <- function(.data, col_name, num_terms, term_index, palette){
    end <- min(nrow(.data), term_index + num_terms)
    q_col_name <- dplyr::enquo(col_name)
    .data[seq(term_index, end),] %>%
        dplyr::pull(word) %>%
        ggpage::ggpage_build() %>%
        dplyr::bind_cols(.data[seq(term_index, end),]) %>%
        ggpage::ggpage_plot(ggplot2::aes(fill = !! q_col_name)) ## +
}
```

Listing 29: Create Page View Plot

### 3.4.5 Wrapper

```
#' create a group-aware visualisation
#'
#' @param .data the standard dataframe, modified so the last column
#'     is the output of some insight function (eg. output from
#'     term_freq)
#'
#' @param vis character name of visualisation function
#'
#' @param col character name of the column to get insight from
#'
#' @param facet_by character name of the column to facet by
#'
#' @param scale_fixed force scales to be fixed in a facet
#'
#' @param ... additional arguments to the visualisation
#'
#' @export
get_vis <- function(.data, vis, col, facet_by="", scale_fixed = TRUE, ...){
    vistable <- list("Page View" = struct_pageview,
                     "Time Series" = struct_time_series,
```

```
                    "Bar" = score_barplot,
                    "Density" = dist_density,
                    "Histogram" = dist_hist,
                    "Word Cloud" = score_wordcloud)
    y <- dplyr::sym(col)
    chart <- vistable[[vis]](.data, !! y, ...)
    if (shiny::isTruthy(facet_by)){
        facet_name <- dplyr::sym(facet_by)
        q_facet_name <- dplyr::enquo(facet_name)
        return(chart + ggplot2::facet_wrap(ggplot2::vars(!! q_facet_name),
                                           scales = ifelse(vis == "struct_pageview" | scale_fixed,
                                                           "fixed",
                                                           "free")))

    } else {
        return(chart)
    }
}
```

Listing 30: Create Visualisation

## 3.5 Application

```
library(shiny)
library(inzightta)
library(rlang)

ui <- navbarPage("iNZight Text Analytics",
                 tabPanel("Processing",
                          sidebarLayout(
                              sidebarPanel(
                                  tags$p("Import"),
                                  fileInput("file1", "Choose File(s)",
                                            multiple = TRUE,
                                            accept = c("text/csv",
                                                       "text/comma-separated-values,text/plain",
                                                       ".csv", ".xlsx", ".xls")),
                                  tags$hr(),
                                  tags$p("Process"),
                                  checkboxInput("lemmatise", "Lemmatise"),
                                  uiOutput("sw_lexicon"),
                                  checkboxInput("stopwords", "Stopwords"),
                                  actionButton("prep_button", "Prepare Text"),
                                  selectInput("section_by", "Section By",
                                              list("", "chapter", "part", "section", "canto", "book")),
                                  uiOutput("vars_to_filter"),
                                  textInput("filter_pred", "value to match", "")),
                              mainPanel(
                                  tableOutput("table")))),
                 tabPanel("Visualisation",
                          sidebarLayout(
                              sidebarPanel(selectInput("what_vis",
                                                       "Select what you want to Visualise",
                                                       list("Term Frequency",
                                                            "n-gram Frequency",
                                                            "Key Words",
                                                            "Term Sentiment",
                                                            "Moving Average Term Sentiment",
                                                            "Aggregated Term Count",
```

```
                                                        "Key Sections",
                                                        "Aggregated Sentiment")),
                                            uiOutput("group_by"),
                                            uiOutput("insight_options"),
                                            uiOutput("vis_options"),
                                            uiOutput("vis_facet_by"),
                                            actionButton("plot_button", "PLOT!!!")),
                                mainPanel(
                                    plotOutput("plot"))))
                  )

server <- function(input, output) {
    imported <- reactive({
        inzightta::import_files(input$file1$datapath)})
    prepped <- eventReactive(input$prep_button, {
        imported() %>%
            format_data(input$lemmatise, input$stopwords, input$sw_lexicon, NA)})
    sectioned <- reactive({
        data <- prepped()
        if (isTruthy(input$section_by)){
            data <- data %>%
                section(input$section_by)}
        data})
    filtered <- reactive({
        data <- sectioned()
        if (isTruthy(input$filter_var) &
            isTruthy(input$filter_pred)){
            data <- data %>%
                dplyr::filter(!! dplyr::sym(input$filter_var) == input$filter_pred)}
        data})
    grouped <- reactive({
        data <- filtered()
        if (isTruthy(input$group_var)){
            data <- data %>%
                dplyr::group_by(!! dplyr::sym(input$group_var))}
        data})
    output$table <- renderTable({
        filtered() %>% head(300)})
    output$sw_lexicon <- renderUI(selectInput("sw_lexicon", "Select the Stopword Lexicon",
                                              stopwords::stopwords_getsources()))
    output$vars_to_filter <- renderUI(selectInput("filter_var",
                                                  "select which column to apply filtering to",
                                                  c("", names(sectioned())) %||% c("")))
    output$group_by <- renderUI(selectInput("group_var",
                                            "select which columns to group on",
                                            c("", names(filtered())) %||% c("")))
    output$insight_options <- renderUI({
        switch(input$what_vis,
               "Term Frequency" = selectInput("vis_type",
                                              "Select how to Visualise it",
                                              list("Bar",
                                                   "Word Cloud",
                                                   "Page View",
                                                   "Time Series",
                                                   "Density",
                                                   "Histogram")),
               "n-gram Frequency" = tagList(selectInput("vis_type",
                                                        "Select how to Visualise it",
                                                        list("Bar",
                                                             "Word Cloud",
                                                             "Page View",
                                                             "Time Series",
```

```
                                                  "Density",
                                                  "Histogram")),
                                    sliderInput("n_gram",
                                                "n-gram count",
                                                2, 8, 2)),
                "Key Words" = tagList(selectInput("vis_type",
                                                  "Select how to Visualise it",
                                                  list("Bar",
                                                       "Word Cloud",
                                                       "Page View",
                                                       "Time Series",
                                                       "Density",
                                                       "Histogram")),
                                      selectInput("summ_method",
                                                  "Method of summary generation",
                                                  list("TextRank", "LexRank"))),
                "Term Sentiment" = tagList(selectInput("vis_type",
                                                       "Select how to Visualise it",
                                                       list("Page View",
                                                            "Word Cloud",
                                                            "Time Series",
                                                            "Bar",
                                                            "Density",
                                                            "Histogram")),
                                           selectInput("sent_lex",
                                                       "Lexicon for Sentiment Dictionary",
                                                       list("afinn", "bing",
                                                            "loughran", "nrc"))),
                "Moving Average Term Sentiment" = tagList(selectInput("vis_type",
                                                                      "Select how to Visualise it",
                                                                      list("Time Series",
                                                                           "Word Cloud",
                                                                           "Page View",
                                                                           "Bar",
                                                                           "Density",
                                                                           "Histogram")),
                                                          sliderInput("term_sent_lag",
                                                                      "Lag Length for Calculation of Moving Averag
                                                                      3,500,50),
                                                          selectInput("sent_lex",
                                                                      "Lexicon for Sentiment Dictionary",
                                                                      list("afinn", "bing",
                                                                           "loughran", "nrc"))),
                "Aggregated Term Count" = tagList(selectInput("vis_type",
                                                              "Select how to Visualise it",
                                                              list("Bar",
                                                                   "Word Cloud",
                                                                   "Page View",
                                                                   "Time Series",
                                                                   "Density",
                                                                   "Histogram")),
                                                  selectInput("agg_var",
                                                              "Select which variable to aggregate on",
                                                              c("", names(grouped())) %||% c("")),
                "Key Sections" = tagList(selectInput("vis_type",
                                                     "Select how to Visualise it",
                                                     list("Bar",
                                                          "Word Cloud",
                                                          "Page View",
                                                          "Time Series",
                                                          "Density",
                                                          "Histogram")),
```

```
                                        selectInput("summ_method",
                                                    "Method of summary generation",
                                                    list("TextRank", "LexRank")),
                                        selectInput("agg_var",
                                                    "Select which variable to aggregate on",
                                                    c("", names(grouped())) %||% c("")),
                "Aggregated Sentiment" = tagList(selectInput("vis_type",
                                                             "Select how to Visualise it",
                                                             list("Page View",
                                                                  "Word Cloud",
                                                                  "Time Series",
                                                                  "Bar",
                                                                  "Density",
                                                                  "Histogram")),
                                                 selectInput("sent_lex",
                                                             "Lexicon for Sentiment Dictionary",
                                                             list("afinn", "bing",
                                                                  "loughran", "nrc")),
                                                 selectInput("agg_var",
                                                             "Select which variable to aggregate on",
                                                             c("", names(grouped())) %||% c(""))))})
    insighted <- reactive({
        switch(input$what_vis,
                "Term Frequency" = get_term_insight(grouped(),
                                                    input$what_vis),
                "n-gram Frequency" = get_term_insight(grouped(),
                                                      c("n-grams", "n-gram Frequency"),
                                                      input$n_gram),
                "Key Words" = get_term_insight(grouped(),
                                               input$what_vis,
                                               input$summ_method),
                "Term Sentiment" = get_term_insight(grouped(),
                                                    input$what_vis,
                                                    input$sent_lex),
                "Moving Average Term Sentiment" = get_term_insight(grouped(),
                                                                   input$what_vis,
                                                                   input$sent_lex,
                                                                   input$term_sent_lag),
                "Aggregated Term Count" = get_aggregate_insight(grouped(),
                                                                c("Bound Aggregates", input$what_vis),
                                                                input$agg_var),
                "Key Sections" = get_aggregate_insight(grouped(),
                                                       c("Bound Aggregates", input$what_vis),
                                                       input$agg_var,
                                                       input$summ_method),
                "Aggregated Sentiment" = get_aggregate_insight(grouped(),
                                                               c("Bound Aggregates", input$what_vis),
                                                               input$agg_var,
                                                               input$sent_lex))})
    output$vis_options <- renderUI({
        switch(input$vis_type,
                "Word Cloud" = tagList(sliderInput("num_terms",
                                                   "Select the number of terms to visualise",
                                                   3, 50, 15),
                                       selectInput("wordcloud_shape",
                                                   "Select the shape of the wordcloud",
                                                   list("circle",
                                                        "cardioid",
                                                        "diamond",
                                                        "square",
                                                        "triangle-forward",
                                                        "triangle-upright",
```

```
                                                    "pentagon",
                                                    "star"))),
                "Page View" = tagList(sliderInput("num_terms",
                                                  "Select the number of terms to visualise",
                                                  3, 400, 100),
                                      sliderInput("term_index",
                                                  "Select the point to begin visualisation from",
                                                  1, nrow(insighted()), 1),
                                      selectInput("palette",
                                                  "Select the colour palette type",
                                                  list("Sequential", "Diverging"))),
                "Bar" = tagList(sliderInput("num_terms", "Select the number of terms to visualise",
                                            2,50,15))})
    output$vis_facet_by <- renderUI(tagList(selectInput("vis_facet",
                                                        "select which variable to facet on",
                                                        c("", names(grouped())) %||% c("")),
                                            checkboxInput("scale_fixed", "Scale Fixed", value=TRUE)))
    visualisation <- reactive({
        switch(input$vis_type,
                "Word Cloud" = switch(input$what_vis,
                                      "n-gram Frequency" = get_vis(
                                          insighted(),
                                          input$vis_type,
                                          input$what_vis,
                                          input$vis_facet,
                                          input$scale_fixed,
                                          input$num_terms,
                                          x = `n-grams`,
                                          shape = input$wordcloud_shape),
                                      "Aggregated Term Count" =,
                                      "Key Sections" =,
                                      "Aggregated Sentiment" = get_vis(
                                          insighted(),
                                          input$vis_type,
                                          input$what_vis,
                                          input$vis_facet,
                                          input$scale_fixed,
                                          input$num_terms,
                                          x = `Bound Aggregates`,
                                          shape = input$wordcloud_shape),
                                      get_vis(insighted(),
                                              input$vis_type,
                                              input$what_vis,
                                              input$vis_facet,
                                              input$scale_fixed,
                                              input$num_terms,
                                              shape = input$wordcloud_shape)),
                "Page View" = get_vis(insighted(), input$vis_type,
                                      input$what_vis,
                                      input$vis_facet,
                                      input$scale_fixed,
                                      input$num_terms,
                                      input$term_index,
                                      palette = input$palette),
                "Time Series" = get_vis(insighted(), input$vis_type,
                                        input$what_vis,
                                        input$vis_facet,
                                        input$scale_fixed),
                "Bar" = switch(input$what_vis,
                               "n-gram Frequency" = get_vis(insighted(),
                                                            input$vis_type,
                                                            input$what_vis,
```

```
                                                    input$vis_facet,
                                                    input$scale_fixed,
                                                    input$num_terms,
                                                    x = `n-grams`),
                                "Aggregated Term Count" =,
                                "Key Sections" =,
                                "Aggregated Sentiment" = get_vis(insighted(),
                                                                 input$vis_type,
                                                                 input$what_vis,
                                                                 input$vis_facet,
                                                                 input$scale_fixed,
                                                                 input$num_terms,
                                                                 x = `Bound Aggregates`),
                                get_vis(insighted(), input$vis_type,
                                        input$what_vis,
                                        input$vis_facet,
                                        input$scale_fixed,
                                        input$num_terms)),
                "Density" = get_vis(insighted(), input$vis_type,
                                    input$what_vis, input$vis_facet,
                                    input$scale_fixed),
                "Histogram" = get_vis(insighted(), input$vis_type,
                                      input$what_vis,
                                      input$vis_facet,
                                      input$scale_fixed))})
    output$plot <- renderPlot({
        visualisation()})
}

# Create Shiny app ----
shinyApp(ui, server)
```

Listing 31: UI and Server of Shiny Application

# Chapter 4

# Conclusion

## 4.1 Summary

### 4.1.1 summarise successes

### 4.1.2 summarise failures

### 4.1.3 general thoughts on the topic

## 4.2 Recommendations

### 4.2.1 educational potential of text analytics

### 4.2.2 what else remains

# Chapter 5

# Appendix

The following pages are a copy of the documentation for the R package created as a part of this dissertation. They were automatically generated through the Roxygen2 system.

# Package 'inzightta'

September 24, 2019

**Title** iNZight Text Analytics

**Version** 0.0.0.9000

**Description** Provides text analytics functions for the importation, analysis, and visualisation of text. This package is designed specifically for output in the shiny program, with the analytical functions all working well with dplyr tools.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Imports** readr, tibble, stringr, dplyr, readxl, purrr, tidytext, textstem, magrittr, stats, textrank, lexRankr, ggpage, ggplot2, forcats, shiny, ggwordcloud

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Author** Jason Cairns [aut, cre]

**Maintainer** Jason Cairns <jcai849@aucklanduni.ac.nz>

# R topics documented:

aggregate_sentiment . . . 3
bind_aggregation . . . 3
concat_walk . . . 4
concat_walk_i . . . 4
determine_stopwords . . . 5
dist_density . . . 5
dist_hist . . . 6
format_data . . . 6
get_aggregate_insight . . . 7
get_bigram . . . 7
get_books . . . 8
get_cantos . . . 8
get_chapters . . . 9
get_filetype . . . 9
get_ngram . . . 10
get_parts . . . 10
get_search . . . 11
get_sections . . . 11
get_sw . . . 12
get_term_insight . . . 12
get_valid_input . . . 13
get_vis . . . 13
ifexp . . . 14
import_base_file . . . 14
import_csv . . . 15
import_excel . . . 15
import_files . . . 16
import_txt . . . 16
keywords_tr . . . 17
key_aggregates . . . 17
ma_term_sentiment . . . 18
ngram_freq . . . 18
score_barplot . . . 19
score_wordcloud . . . 19
section . . . 20
shorten . . . 20
struct_pageview . . . 21
struct_time_series . . . 21
table_textcol . . . 22
term_cooccurance . . . 22
term_corr . . . 23
term_count . . . 23
term_freq . . . 24
term_sentiment . . . 24
ungroup_by . . . 25

**Index** **26**

## aggregate_sentiment *Get statistics for sentiment over some group, such as sentence.*

### Description

Get statistics for sentiment over some group, such as sentence.

### Usage

```
aggregate_sentiment(.data, aggregate_on, lexicon = "afinn",
  statistic = mean)
```

### Arguments

| | |
|---|---|
| .data | character vector of words |
| aggregate_on | vector to aggregate .data over; ideally, sentence_id, but could be chapter, document, etc. |
| lexicon | as per term sentiment |
| statistic | function that accepts na.rm argument; e.g. mean, median, sd. |

### Value

sentiment of same length as input vector aggregated over the aggregate_on vector

## bind_aggregation *bind aggregate terms together*

### Description

bind aggregate terms together

### Usage

```
bind_aggregation(data, aggregate_on)
```

### Arguments

| | |
|---|---|
| data | vector of terms |
| aggregate_on | vector of aggregations |

### Value

data with every aggregation bound, as in a sentence

## concat_walk *concat list 1 and 2, moving past NA values*

### Description

concat list 1 and 2, moving past NA values

### Usage

```
concat_walk(list1, list2)
```

### Arguments

| | |
|---|---|
| list1 | list or vector for first bigram token |
| list2 | list or vector for second bigram token |

### Value

paste of list1 and list2, skipping NA's

## concat_walk_i *concat list 1 and 2 at index, skipping NA values*

### Description

concat list 1 and 2 at index, skipping NA values

### Usage

```
concat_walk_i(i, list1, list2)
```

### Arguments

| | |
|---|---|
| i | numeric index to assess index at |
| list1 | list or vector for first token |
| list2 | list or vector for second token |

### Value

paste of list1 and list2 at index i, skipping NA's

## determine_stopwords *determine stopword status*

### Description

determine stopword status

### Usage

```
determine_stopwords(.data, ...)
```

### Arguments

| | |
|---|---|
| .data | vector of words |
| ... | arguments of get_sw |

### Value

a [tibble][tibble::tibble-package] equivalent to the input dataframe, with an additional stopword column

## dist_density *output a histogram of the distribution of some function of words*

### Description

output a histogram of the distribution of some function of words

### Usage

```
dist_density(.data, col_name)
```

### Arguments

| | |
|---|---|
| .data | the standard dataframe, modified so the last column is the output of some insight function (eg. output from term_freq) |
| col_name | symbol name of the column insight was performed on |

## dist_hist *output a histogram of the distribution of some function of words*

### Description

output a histogram of the distribution of some function of words

### Usage

```
dist_hist(.data, col_name)
```

### Arguments

| | |
|---|---|
| .data | the standard dataframe, modified so the last column is the output of some insight function (eg. output from term_freq) |
| col_name | symbol name of the column insight was performed on |

## format_data *takes imported one-line-per-row data and prepares it for later analysis*

### Description

takes imported one-line-per-row data and prepares it for later analysis

### Usage

```
format_data(.data, lemmatize = TRUE, stopwords = TRUE,
  sw_lexicon = "snowball", addl_stopwords = NA)
```

### Arguments

| | |
|---|---|
| .data | tibble with one line of text per row |
| lemmatize | boolean, whether to lemmatize or not |
| stopwords | boolean, whether to remove stopwords or not |
| sw_lexicon | string, lexicon with which to remove stopwords |
| addl_stopwords | char vector of user-supplied stopwords |

### Value

a [tibble][tibble::tibble-package] with one token per line, stopwords removed leaving NA values, column for analysis named "text"

## get_aggregate_insight *perform group-aware aggregate operations on the data*

### Description

perform group-aware aggregate operations on the data

### Usage

```
get_aggregate_insight(.data, operations, aggregate_on, ...)
```

### Arguments

| | |
|---|---|
| .data | dataframe of terms as per output of format_data |
| operations | character vector of operations to perform |
| aggregate_on | character name of the column to perform aggregate operations on |
| ... | additional arguments to the operation - only sensible for singular operations |

### Value

.data with operation columns added

## get_bigram *Determine bigrams*

### Description

Determine bigrams

### Usage

```
get_bigram(.data)
```

### Arguments

| | |
|---|---|
| .data | character vector of words |

### Value

character vector of bigrams

## get_books *sections text based on book*

### Description

sections text based on book

### Usage

```
get_books(.data)
```

### Arguments

| | |
|---|---|
| .data | vector to section |

### Value

vector of same length as .data with book numbers

## get_cantos *sections text based on cantos*

### Description

sections text based on cantos

### Usage

```
get_cantos(.data)
```

### Arguments

| | |
|---|---|
| .data | vector to section |

### Value

vector of same length as .data with canto numbers

## get_chapters *sections text based on chapters*

### Description

sections text based on chapters

### Usage

```
get_chapters(.data)
```

### Arguments

| | |
|---|---|
| .data | vector to section |

### Value

vector of same length as .data with chapter numbers

## get_filetype *Get filetype*

### Description

Get filetype

### Usage

```
get_filetype(filepath)
```

### Arguments

| | |
|---|---|
| filepath | string filepath of document |

### Value

filetype (string) - NA if no extension

## get_ngram *Returns the n-grams, skipping NA values*

### Description

Returns the n-grams, skipping NA values

### Usage

```
get_ngram(.data, n)
```

### Arguments

| | |
|---|---|
| .data | vector to get n-grams from |
| n | number of n-grams to attain |

### Value

n-gram vector without NA values

## get_parts *sections text based on parts*

### Description

sections text based on parts

### Usage

```
get_parts(.data)
```

### Arguments

| | |
|---|---|
| .data | vector to section |

### Value

vector of same length as .data with part numbers

## get_search *creates a search closure to section text*

### Description

creates a search closure to section text

### Usage

```
get_search(search)
```

### Arguments

| | |
|---|---|
| search | a string regexp for the term to seperate on, e.g. "Chapter" |

### Value

closure over search expression

## get_sections *sections text based on sections*

### Description

sections text based on sections

### Usage

```
get_sections(.data)
```

### Arguments

| | |
|---|---|
| .data | vector to section |

### Value

vector of same length as .data with section numbers

## get_sw *Gets stopwords from a default list and user-provided list*

### Description

Gets stopwords from a default list and user-provided list

### Usage

```
get_sw(lexicon = "snowball", addl = NA)
```

### Arguments

| | |
|---|---|
| lexicon | a string name of a stopword list, one of "smart", "snowball", or "onix" |
| addl | user defined character vector of additional stopwords, each element being a stopword |

### Value

a [tibble][tibble::tibble-package] with one column named "word"

## get_term_insight *perform group-aware term operations on the data*

### Description

perform group-aware term operations on the data

### Usage

```
get_term_insight(.data, operations, ...)
```

### Arguments

| | |
|---|---|
| .data | dataframe of terms as per output of format_data |
| operations | character vector of term operations to perform |
| ... | additional arguments to the operation - only sensible for singular operations |

### Value

.data with operation columns added

## get_valid_input *helper function to get valid input (recursively)*

### Description

helper function to get valid input (recursively)

### Usage

```
get_valid_input(options, init = TRUE)
```

### Arguments

| | |
|---|---|
| options | vector of options that valid input should be drawn from |
| init | whether this is the initial attempt, used only as recursive information |

### Value

readline output that exists in the vector of options

## get_vis *create a group-aware visualisation*

### Description

create a group-aware visualisation

### Usage

```
get_vis(.data, vis, col, facet_by = "", scale_fixed = TRUE, ...)
```

### Arguments

| | |
|---|---|
| .data | the standard dataframe, modified so the last column is the output of some insight function (eg. output from term_freq) |
| vis | character name of visualisation function |
| col | character name of the column to get insight from |
| facet_by | character name of the column to facet by |
| scale_fixed | force scales to be fixed in a facet |
| ... | additional arguments to the visualisation |

`ifexp` *scheme-like if expression, without restriction of returning same-size table of .test, as ifelse() does*

## Description

scheme-like if expression, without restriction of returning same-size table of .test, as ifelse() does

## Usage

```
ifexp(.test, true, false)
```

## Arguments

| | |
|---|---|
| `.test` | predicate to test |
| `true` | expression to return if .test evals to TRUE |
| `false` | expression to return if .test evals to TRUE |

## Value

either true or false

`import_base_file` *Base case for file import*

## Description

Base case for file import

## Usage

```
import_base_file(filepath)
```

## Arguments

| | |
|---|---|
| `filepath` | string filepath of file for import |

## Value

imported file with document id

## import_csv *Import csv file*

### Description

Import csv file

### Usage

```
import_csv(filepath)
```

### Arguments

| | |
|---|---|
| filepath | a string indicating the relative or absolute filepath of the file to import |

### Value

a [tibble][tibble::tibble-package] of each row corrresponding to a line of the text file, with the column named "text"

## import_excel *Import excel file*

### Description

Import excel file

### Usage

```
import_excel(filepath)
```

### Arguments

| | |
|---|---|
| filepath | a string indicating the relative or absolute filepath of the file to import |

### Value

a [tibble][tibble::tibble-package] of each row corrresponding to a line of the text file, with the column named "text"

## import_files *Import any number of files*

### Description

Import any number of files

### Usage

```
import_files(filepaths)
```

### Arguments

| | |
|---|---|
| filepaths | char vector of filepaths |

### Value

a [tibble][tibble::tibble-package] imported files with document id

## import_txt *Import text file*

### Description

Import text file

### Usage

```
import_txt(filepath)
```

### Arguments

| | |
|---|---|
| filepath | a string indicating the relative or absolute filepath of the file to import |

### Value

a [tibble][tibble::tibble-package] of each row corrresponding to a line of the text file, with the column named "text"

## keywords_tr *Determine textrank score for vector of words*

### Description

Determine textrank score for vector of words

### Usage

```
keywords_tr(.data, summ_method)
```

### Arguments

| | |
|---|---|
| .data | character vector of words |
| summ_method | method to use for summarisation: textrank or lexrank. Doesn't do anything yet |

### Value

vector of scores for each word

## key_aggregates *get score for key sentences as per Lexrank*

### Description

get score for key sentences as per Lexrank

### Usage

```
key_aggregates(.data, aggregate_on, summ_method)
```

### Arguments

| | |
|---|---|
| .data | character vector of words |
| aggregate_on | vector to aggregate .data over; ideally, sentence_id |
| summ_method | method to use for summarisation: textrank or lexrank. Doesn't do anything yet |

### Value

lexrank scores of aggregates

## ma_term_sentiment *Determine the lagged sentiment of terms*

### Description

Determine the lagged sentiment of terms

### Usage

```
ma_term_sentiment(.data, lexicon = "afinn", lag = 10,
  statistic = mean)
```

### Arguments

| | |
|---|---|
| .data | vector of terms |
| lexicon | sentiment lexicon to use, based on the corpus provided by tidytext |
| lag | how many (inclusive) terms to compute statistic over |
| statistic | base statistic used to summarise the data, capable of taking an na.rm argument |

### Value

vector with lagged sentiment score of each term in the input vector

## ngram_freq *NOT FOR PRODUCTION - STILL IN TESING. Returns the count of n-grams, skipping NA values*

### Description

NOT FOR PRODUCTION - STILL IN TESING. Returns the count of n-grams, skipping NA values

### Usage

```
ngram_freq(.data, n)
```

### Arguments

| | |
|---|---|
| .data | vector to get n-grams from |
| n | number of n-grams to attain |

### Value

count of each associated n-gram

## score_barplot

*output a ggplot column graph of the top texts from some insight function*

### Description

output a ggplot column graph of the top texts from some insight function

### Usage

```
score_barplot(.data, y, n = 15, x = text, desc = FALSE)
```

### Arguments

| | |
|---|---|
| .data | a dataframe containing "text" and insight columns as per the output of the get_(termlaggregate)_insight wrapper function |
| y | symbol name of the column insight was outputted to |
| n | number of bars to display |
| x | symbol name of column for insight labels |
| desc | bool: show bars in descending order |

## score_wordcloud

*output a ggplot wordcloud graph of the top texts from some insight function*

### Description

output a ggplot wordcloud graph of the top texts from some insight function

### Usage

```
score_wordcloud(.data, y, n = 15, x = text, shape = "circle")
```

### Arguments

| | |
|---|---|
| .data | a dataframe containing "text" and insight columns as per the output of the get_(termlaggregate)_insight wrapper function |
| y | symbol name of the column insight was outputted to |
| n | number of words to display |
| x | symbol name of column for insight labels |
| shape | character: shape of the wordcloud |

## section *Adds section column to dataframe*

### Description

Adds section column to dataframe

### Usage

```
section(.data, section_by)
```

### Arguments

| | |
|---|---|
| .data | dataframe formatted as per output of prep process |
| section_by | character name of what to section over |

### Value

input dataframe with additional section column

## shorten *Shorten some text up to n characters*

### Description

Shorten some text up to n characters

### Usage

```
shorten(.data, n)
```

### Arguments

| | |
|---|---|
| .data | character vector |
| n | wrap length of text |

### Value

shortened form of .data

## struct_pageview *Colours a ggpage based on an insight function*

### Description

Colours a ggpage based on an insight function

### Usage

```
struct_pageview(.data, col_name, num_terms, term_index, palette)
```

### Arguments

| | |
|---|---|
| .data | a dataframe containing "word" and insight columns as per the output of the get_(term\|aggregate)_insight wrapper function |
| col_name | symbol name of the insight column intended to colour plot |
| num_terms | the number of terms to visualise |
| term_index | which term to start the visualisation from |
| palette | determine coloration of palette (not yet implemented) |

### Value

ggplot object as per ggpage

## struct_time_series *output a ggplot time series plot of some insight function*

### Description

output a ggplot time series plot of some insight function

### Usage

```
struct_time_series(.data, y)
```

### Arguments

| | |
|---|---|
| .data | a dataframe containing "text" and insight columns as per the output of the get_(term\|aggregate)_insight wrapper function |
| y | symbol name of the column insight was outputted to |

## table_textcol *Interactively determine and automatically mark the text column of a table*

### Description

Interactively determine and automatically mark the text column of a table

### Usage

```
table_textcol(data)
```

### Arguments

| | |
|---|---|
| data | dataframe with column requiring marking |

### Value

same dataframe with text column renamed to "text"

## term_cooccurance *Determine term cooccurances - extremely slow*

### Description

Determine term cooccurances - extremely slow

### Usage

```
term_cooccurance(.data, term, aggregate_on)
```

### Arguments

| | |
|---|---|
| .data | character vector of terms |
| term | character to find correlations with |
| aggregate_on | vector to aggregate .data over; ideally, sentence_id, but could be chapter, document, etc. |

### Value

numeric vector of term correlations as per phi_coef

## term_corr *Determine term correlations - extremely slow*

### Description

Determine term correlations - extremely slow

### Usage

```
term_corr(.data, term, aggregate_on)
```

### Arguments

| | |
|---|---|
| .data | character vector of terms |
| term | character to find correlations with |
| aggregate_on | vector to aggregate .data over; ideally, sentence_id, but could be chapter, document, etc. |

### Value

numeric vector of term correlations as per phi_coef

## term_count *Determine the number of terms at each aggregate level*

### Description

Determine the number of terms at each aggregate level

### Usage

```
term_count(.data, aggregate_on)
```

### Arguments

| | |
|---|---|
| .data | character vector of terms |
| aggregate_on | vector to split .data on for insight |

### Value

vector of number of terms for each aggregate level, same length as .data

## term_freq *Determine term frequency*

### Description

Determine term frequency

### Usage

```
term_freq(.data)
```

### Arguments

| | |
|---|---|
| .data | character vector of terms |

### Value

numeric vector of term frequencies

## term_sentiment *Determine sentiment of terms*

### Description

Determine sentiment of terms

### Usage

```
term_sentiment(.data, lexicon = "afinn")
```

### Arguments

| | |
|---|---|
| .data | vector of terms |
| lexicon | sentiment lexicon to use, based on the corpus provided by tidytext |

### Value

vector with sentiment score of each word in the vector

ungroup_by *helper function to ungroup for dplyr. functions equivalently to group_by() but with standard (string) evaluation*

## Description

helper function to ungroup for dplyr. functions equivalently to group_by() but with standard (string) evaluation

## Usage

```
ungroup_by(x, ...)
```

## Arguments

| | |
|---|---|
| x | tibble to perform function on |
| ... | string of groups to ungroup on |

## Value

x with ... no longer grouped upon

# Index


aggregate_sentiment, 3

bind_aggregation, 3

concat_walk, 4
concat_walk_i, 4

determine_stopwords, 5
dist_density, 5
dist_hist, 6

format_data, 6

get_aggregate_insight, 7
get_bigram, 7
get_books, 8
get_cantos, 8
get_chapters, 9
get_filetype, 9
get_ngram, 10
get_parts, 10
get_search, 11
get_sections, 11
get_sw, 12
get_term_insight, 12
get_valid_input, 13
get_vis, 13

ifexp, 14
import_base_file, 14
import_csv, 15
import_excel, 15
import_files, 16
import_txt, 16

key_aggregates, 17
keywords_tr, 17

ma_term_sentiment, 18

ngram_freq, 18

score_barplot, 19
score_wordcloud, 19
section, 20
shorten, 20
struct_pageview, 21
struct_time_series, 21

table_textcol, 22
term_cooccurance, 22
term_corr, 23
term_count, 23
term_freq, 24
term_sentiment, 24

ungroup_by, 25

# Glossary

**term** "a word or expression that has a precise meaning in some uses or is peculiar to a science, art, profession, or subject'[1] — here text analysts have capitalised on the generalisation of "term'to include subcomponents or aggregations of words. 9

# Bibliography

[1] Merriam-Webster Dictionary, ed. *Term — Definition of Term*. 17th Aug. 2019. URL: `https://www.merriam-webster.com/dictionary/term` (cit. on p. 69).