# Final Project - An Analysis of Game of Throne's Script

Colby Fry

2024-04-27

## Overview

The hit television show "Game of Thrones" is known for it's incredibly rich story and beautifully portrayed world. However, neither of those two things would be complete, without the masterfully written ensemble of characters that make up George R.R. Martin's fictional universe. A great story is only as great as those that are portrayed within that story. Between the Lannisters, Starks, Tyrells, Greyjoys, and the like, Game of Thrones is host to a sizable amount of characters and a whole lot to unpack between each. What is it exactly that makes the world of Westeros so riveting? Yes the setting, world-building, and cinematography are all top notch, but equally is that of the show's script.

The goal of this project is to analyze the entire script of the television adaption of the Song of Ice and Fire Series (Game of Thrones) across each season that aired and also, the context of the words spoken within the script by the characters. What was the overall mood of the show? Were certain seasons more uplifting than others? What words were often spoken in sequence of each other? What were the most popular words spoken? These are all questions this analysis aims to answer. Methods used to answer these questions include the usage of TF-IDF to determine the importance of words in the script, sentiment and polarity analysis with popular lexicons (nrc and bing) to uncover common moods of each Season of the show, and usage of ggplot, ipgraphs and ggraph visualizations to accompany these findings.

## Literature Review

Before I dive into my own analysis of the Game of Thrones script, I would like to review a handful of articles relevant to my project.

### Article 1

**R Bloggers - Mining Game of Thrones Script with R** Source = https://www.r-bloggers.com/2017/12/mining-game-of-thrones-scripts-with-r/

The author who performed their Game of Thrones script analysis in R originally posted their work on their personal blog, Computational Social Science. Unfortunately that blog seems to no longer exist, so I was unable to retrieve this directly from the original source. The author's approach to their analysis of the Game of Thrones script, while similar to my own at points, did take quite a few different routes. A key difference between my analysis and this individual's is that their analysis relies heavily on the usage of the "quanteda" package in R. The quanteda package is very modular and allows for easy implementation of custom functions and/or methods for text analysis, it also provides efficient data structures such as DFMs (document feature matrixes). Though it does share similarities with tidytext, in that both can be used for tokenization, topic modeling, and text classification for example. Tidytext on the other hand is meant to seamlessly integrate with the tidyverse package, following tidy data principles.

The author uses the rvest package to scrape a website (https://www.springfieldspringfield.co.uk/view_episode_scripts.php?tv-show=game-of-thrones&episode=s01e01) to retrieve the script for the show, implementing a for loop within their code to scrape the script for every episode of each season. The author then uses quanteda to convert the script into a corpus, using the corpus() function. Something I found really interesting about the quanteda package is that it has a function called kwic(), which stands for "keywords in context". What this function does, is immediately return a list of words and their immediate context. For example, they can set the kwic function to parse for the word "winter" and it will return a list of sentences where that word appears throughout each episode.

Their analysis also tokenizes the script into n-grams, up to 3 in their analysis.The script is then converted into a DFM and stop words are removed. Unlike my analysis of the script, this individual put a focus on temporal trends, such as nicknames or aliases characters may have gone by throughout the show. They provide Daenerys Targaryen as an example, they create a vector containing popular aliases of that character and then use R to parse the script for the amount of times in each season each of these aliases were referenced. The author proceeds to use many other incredibly useful functions from the quanteda package throughout their analysis, I would love to try my own analysis on another project using the quanteda package and similar methods this author used.

## Article 2

**Ekram Towsif - Textual Analysis of Game of Thrones Character Dialogue's for the Books and Tv Show** Source = https://rpubs.com/etowsif/497185

Interestingly enough, this article was written by a student who's final project for their course was to analyze the script of Game of Thrones, but in addition to that, the dialogue of the books. This student's goal was to uncover differences between the script of the television adaptation of the books, as well as to compares trends between the two. What makes this type of analysis of Game of Thrones all the more intriguing is that, the television adaptation is a complete series while the books still have yet to be finished, there are many variances towards the halfway points of each. Due to this, the author of this project had to cut down on the amount of content they were analyzing in order for the results to be a closer one to one comparison, books 1-3 and seasons 1-4.

The student who performed this analysis imported the first three books of the series using online PDFs with the pdftools package. Similar to the first article I analyzed for this project, this student scrapes a website for the script of each seasons they analyzed and implemented a for loop to gather the script foe each episode in those seasons.

Sentiment analysis was also performed in this student's project, which is something that my own project focuses on. Topic modeling, tokenization, and even igraph/ggraphs are implemented in this student's final project as well. Overall I found this student's work to be incredibly thorough and loved their approaches to how they handled running comparisons across the different source material.

## Article 3

**Idil (Kaggle) - Game of Thrones Subtitles Analysis - Text Mining** Source = https://www.kaggle.com/code/bozayidil/game-of-thrones-subtitles-analysis-text-mining

This last article was written by a German researcher on Kaggle, going by the alias of "Idil". In Idil's analysis, her primary focus was to uncover the mood of the Stark family in Game of Thrones across the entire television series, based on show's subtitles. Similar to my project, her approach to loading in the source material was by importing a CSV file of the text into R. Idil primarily uses the "bing" lexicon for her sentiment analysis, where I chose to use the "nrc" lexicon due to having calculated a higher match ratio with that lexicon to my dataset. In general, her sentiment analysis approaches are comparable to mine, though since her whole project was a sentiment analysis on one specific family in the story, the sentiment analysis is more intricate and honed in on this particular subject, the Stark Family.

## Preparatory Work

Before any analysis can be completed, the necessary tools to perform the analysis need to be loaded into the R Studio environment that this project was completed in.In addition, the script of the show needs to be imported into R Studio before any analysis can be possible.

```r
library(tidytext)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
got_script <- read_csv("Game_of_Thrones_Script.csv")
```

```
## Rows: 23911 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): Season, Episode, Episode Title, Name, Sentence
## date (1): Release Date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

## Pre-Processing

Next, the script cannot be analyzed as is. We need to perform some clean up before we can dive in and pick away at this file. The work performed, below, cleans up some of the variable names of the file and also removes common contractions from the Sentences variable of the file (script).

```r
got_script <- got_script %>%
  select('Release Date', 'Season', 'Episode Number' = Episode, 'Episode Title', Character = Name, Sente
fix.contractions <- function(doc) {
doc <- gsub("won't", "will not", doc)
doc <- gsub("can't", "can not", doc)
doc <- gsub("n't", " not", doc)
doc <- gsub("'ll", " will", doc)
doc <- gsub("'re", " are", doc)
doc <- gsub("'ve", " have", doc)
doc <- gsub("'m", " am", doc)
doc <- gsub("'d", " would", doc)
doc <- gsub("'s", "", doc)
return(doc)
}
got_script$Sentence <- sapply(got_script$Sentence, fix.contractions)
```

Following the removal of contractions, this script needs to be put into a tidy format. Here, we are tokenizing the Sentences variable so that there is only one word per row within the dataset. We are also removing common stop words (and, with, from, etc.), filtering out words with less than 2 characters, and lastly we use the gsub function to replace any white spaces or non-character values in the newly created "word" variable with an empty string value instead.

```
tidy_script <- got_script %>%
unnest_tokens("word", Sentence)%>%
anti_join(stop_words)%>%
filter(nchar(word) > 2)
```

```
## Warning: Outer names are only allowed for unnamed scalar atomic inputs

## Joining with ‘by = join_by(word)‘
```

```
tidy_script$word <- gsub("\\s+","", tidy_script$word)
tidy_script$word <- gsub("[^a-zA-Z]","", tidy_script$word)
```

## Term Frequency Distribution

The following code begins our analysis of the most frequently spoken terms in the script of Game of Thrones, by Season. Starting off, we are creating a new object called "count_script", which will simply count the number of words per season, sorted in descending order. The code below also creates a new object called "total_words". This object is essentially creating a total of most common words in the script from the tidy_script object, the two objects are then left joined together. The "n" column of the season_words object is the number of times the word was spoken in a season and the "total" column is the number of words spoken in the season as a whole.

These results are then plotted as histograms, grouped by season, to showcase the distribution of word frequencies from the script.

```
count_script <- tidy_script %>%
  count(Season, word, sort = TRUE)

total_words <- count_script %>%
  group_by(Season) %>%
  anti_join(stop_words)%>%
  filter(nchar(word) > 2) %>%
  summarise(total = sum(n))
```

```
## Joining with ‘by = join_by(word)‘
```

```
season_words <- left_join(count_script, total_words)
```

```
## Joining with ‘by = join_by(Season)‘
```

```
season_words
```

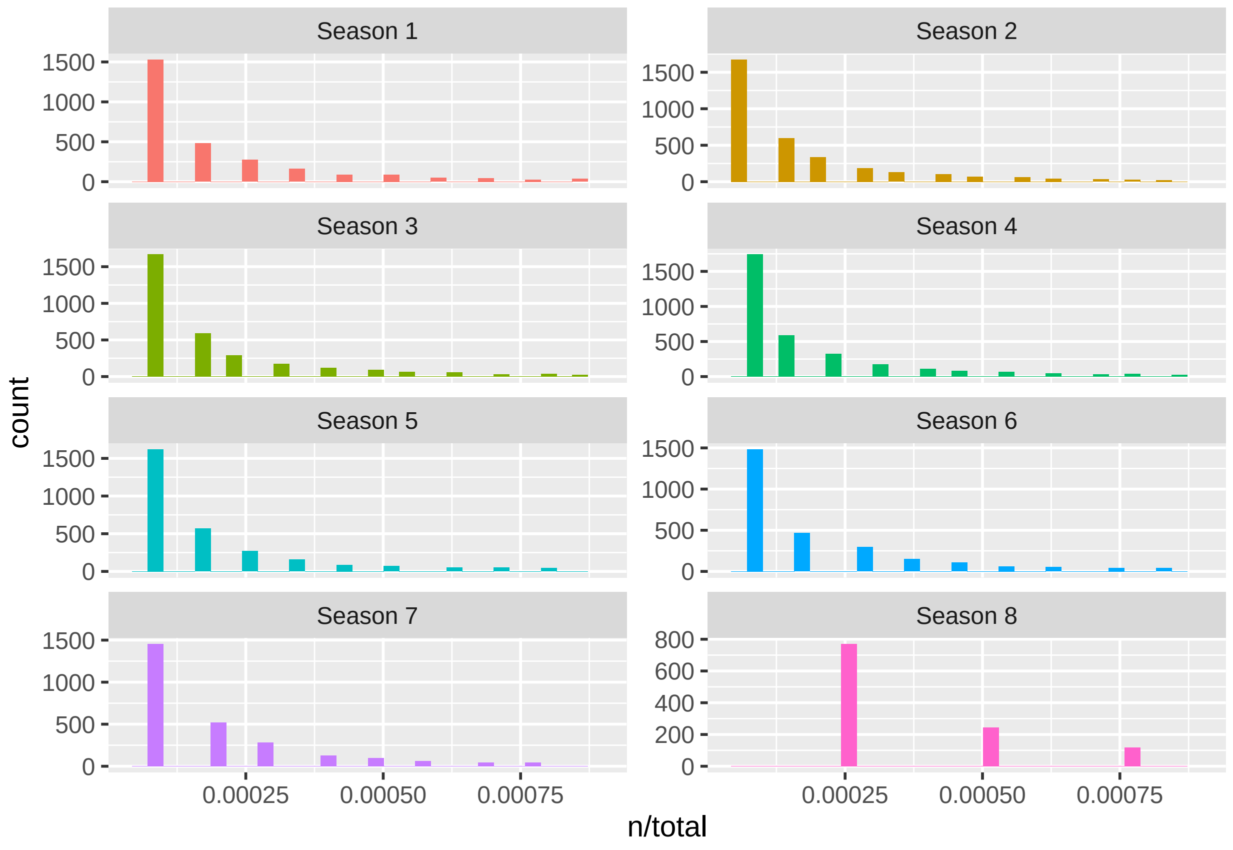
```
## # A tibble: 23,664 x 4
##    Season   word       n total
##    <chr>    <chr>  <int> <int>
##  1 Season 1 lord     239 11626
##  2 Season 2 king     223 14260
##  3 Season 1 king     212 11626
##  4 Season 2 lord     205 14260
##  5 Season 3 lord     177 12631
##  6 Season 4 lord     154 12823
##  7 Season 4 king     150 12823
##  8 Season 3 king     140 12631
##  9 Season 5 king     134 11374
## 10 Season 1 father   126 11626
## # i 23,654 more rows
```

```
#Term frequency distribution across each season of the show
ggplot(season_words, aes(n/total, fill = Season)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~Season, ncol = 2, scales = "free_y")
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```

```
## Warning: Removed 1786 rows containing non-finite outside the scale range
## (‘stat_bin()‘).
```

```
## Warning: Removed 8 rows containing missing values or values outside the scale range
## (‘geom_bar()‘).
```



## TF-IDF

We can use bind_tf_idf() to tell R to determine what the most important words were throughout each Season of Game of Thrones. We will start with the original got_script file that had been imported and tokenize the script based on words in the Sentence column. Stop Words are then removed with anti_join(), words less than 3 characters are filtered out, and we then group the results of bind_tf_idf() by Season

```
popular_tfidf_words <- got_script %>%
  unnest_tokens("word", Sentence) %>%
  anti_join(stop_words) %>%
  filter(nchar(word) > 3) %>%
  group_by(Season) %>%
  count(Season, word, sort = TRUE) %>%
  bind_tf_idf(word, Season, n)
```

```
## Warning: Outer names are only allowed for unnamed scalar atomic inputs
```

```
## Joining with ‘by = join_by(word)‘
```

```
popular_tfidf_words
```

```
## # A tibble: 22,739 x 6
## # Groups:   Season [8]
##    Season   word       n     tf   idf tf_idf
##    <chr>    <chr>  <int>  <dbl> <dbl>  <dbl>
##  1 Season 1 lord     239 0.0223     0      0
##  2 Season 2 king     223 0.0168     0      0
##  3 Season 1 king     212 0.0198     0      0
##  4 Season 2 lord     205 0.0155     0      0
##  5 Season 3 lord     177 0.0152     0      0
##  6 Season 4 lord     154 0.0129     0      0
##  7 Season 4 king     150 0.0125     0      0
##  8 Season 3 king     140 0.0120     0      0
##  9 Season 5 king     134 0.0125     0      0
## 10 Season 1 father   126 0.0118     0      0
## # i 22,729 more rows
```

The most important words from the show can be plotted, by Season. The code below plots the top 10 most important words of each season.

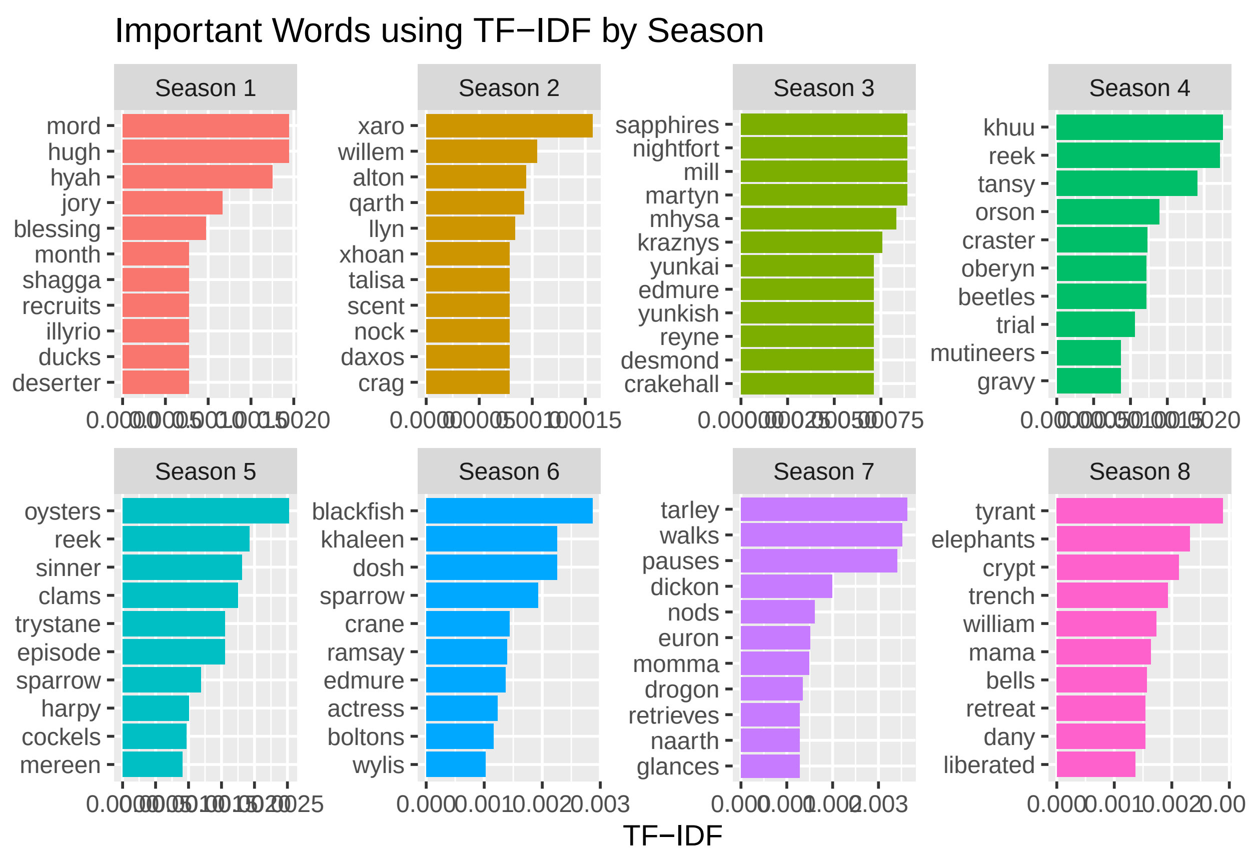
```
list <- c("Season 1", "Season 2", "Season 3", "Season 4", "Season 5", "Season 6", "Season 7", "Season 8

top_popular_tfidf_words <- popular_tfidf_words %>%
  filter(Season %in% list) %>%
  arrange(desc(tf_idf)) %>%
  mutate(word=reorder_within(word,tf_idf,Season))%>%
  top_n(10) %>%
  ggplot(aes(tf_idf, word, fill = Season) )+
  geom_col(show.legend = NULL) +
  ylab(NULL) +
  xlab("TF-IDF") +
  ggtitle("Important Words using TF-IDF by Season") +
  scale_y_reordered() +
  facet_wrap(~Season, ncol = 4, scales = "free")
```

```
## Selecting by tf_idf
```

```
top_popular_tfidf_words
```

```
## Warning: ‘show.legend‘ must be a logical vector.
```



## Match Ratio

The tidytext package of R contains three general purpose lexicons for the purpose of evaluating moods, opinions, etc in a given text dataset. Those lexicons are “AFINN”, “bing”, and “nrc”. The three lexicons are based on unigrams (single words), with each word being assigned a certain value. In the case of the “nrc” lexicon for instance, each word may be given a specific emotion it would relate to, such as the word “frighten” giving off the emotion of “fear”.

Given that we have three lexicons we can work with, we should focus primarily on the lexicon that contains the most words from our dataset (script). We can determine this by calculating the match ratio between the Game of Thrones script and of the three lexicons, using the code below. Based on the match ratio results after running the following code, we can determine that the “nrc” lexicon has the most words in common with our dataset, compared to the other two lexicons.

```
bing <- get_sentiments("bing")
nrc <- get_sentiments("nrc")
afinn <- get_sentiments("afinn")

afinn_neg_pos <- afinn %>%
mutate( sentiment = ifelse( value >= 0, "positive",
ifelse( value < 0,
"negative", value)))

afinn_neg_pos <-afinn_neg_pos %>%
select(word, sentiment)
```

```r
sentiments <-bind_rows(list(bing=bing,nrc=nrc,afinn=afinn_neg_pos),.id =
"lexicon")
new_sentiments <- sentiments %>%
group_by(lexicon) %>%
mutate(words_in_lexicon = n_distinct(word)) %>%
ungroup()

new_sentiments <- sentiments %>%
group_by(lexicon) %>%
mutate(words_in_lexicon = n_distinct(word)) %>%
ungroup()

match_ratio_got <- tidy_script %>%
mutate(words_in_script = n_distinct(word)) %>%
inner_join(new_sentiments) %>%
group_by(lexicon,words_in_script, words_in_lexicon) %>%
summarise(lex_match_words = n_distinct(word)) %>%
ungroup() %>%
mutate(total_match_words = sum(lex_match_words),
match_ratio = lex_match_words / words_in_script) %>%
select(lexicon, lex_match_words, words_in_script, match_ratio)
```

```
## Joining with `by = join_by(word)`
```

```
## Warning in inner_join(., new_sentiments): Detected an unexpected many-to-many relationship between `
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 13069 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
## `summarise()` has grouped output by 'lexicon', 'words_in_script'. You can
## override using the `.groups` argument.
```

```r
match_ratio_got
```

```
## # A tibble: 3 x 4
##   lexicon lex_match_words words_in_script match_ratio
##   <chr>             <int>           <int>       <dbl>
## 1 afinn              1065            8799       0.121
## 2 bing               1638            8799       0.186
## 3 nrc                2113            8799       0.240
```

## Sentiment Analysis

Now that we have determined which lexicon we should proceed with, we can move forward in performing our sentiment analysis of the words in the Game of Thrones script, based around the "nrc" lexicon. To begin, we will perform an inner join on the sentiments from the "nrc" lexicon, to the list of words from our "tidy_script" object, storing the results in a new object called "got_nrc".

```
got_nrc <- tidy_script %>%
  inner_join(get_sentiments("nrc"))
```

```
## Joining with ‘by = join_by(word)‘
```

```
## Warning in inner_join(., get_sentiments("nrc")): Detected an unexpected many-to-many relationship be
## i Row 1 of ‘x‘ matches multiple rows in ‘y‘.
## i Row 6283 of ‘y‘ matches multiple rows in ‘x‘.
## i If a many-to-many relationship is expected, set ‘relationship =
##   "many-to-many"‘ to silence this warning.
```

```
head(got_nrc)
```

```
## # A tibble: 6 x 7
##   ‘Release Date‘ Season   ‘Episode Number‘ ‘Episode Title‘ Character    word
##   <date>         <chr>    <chr>            <chr>           <chr>        <chr>
## 1 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce expect
## 2 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce expect
## 3 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce expect
## 4 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce expect
## 5 2011-04-17     Season 1 Episode 1        Winter is Coming royce        fright~
## 6 2011-04-17     Season 1 Episode 1        Winter is Coming royce        fright~
## # i 1 more variable: sentiment <chr>
```

Now that we have paired each word from the Game of Thrones script with a corresponding emotion from the “nrc” lexicon, we can create a visualization of the most common words from the script and categorize them by the various emotions they give off. This visualization will not be categorized by Season, as the focus here is to gather a general idea on the script’s overall themes.

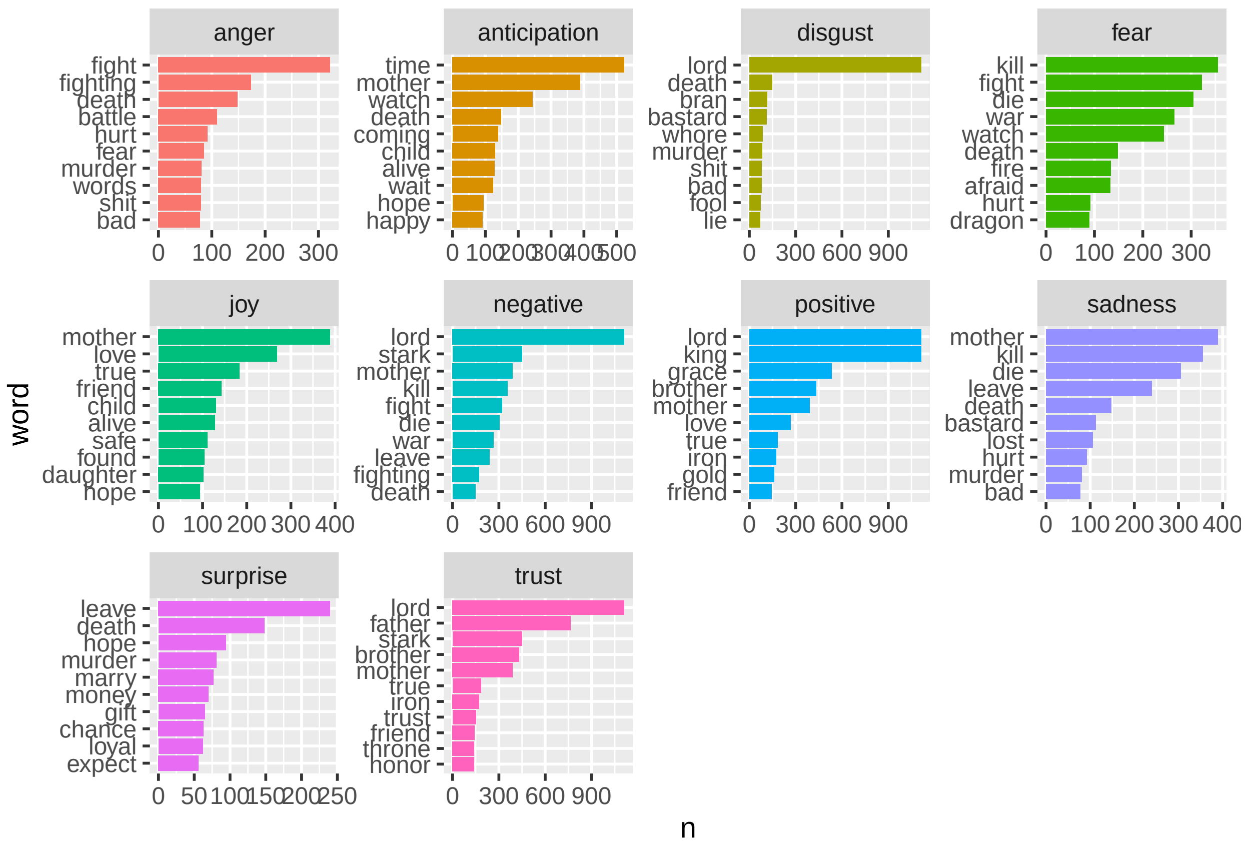
We can see that words from the script such as “kill”, “fight”, “war”, and the like all resonate with the emotion of “fear”. Depending on the context of the word’s usage within the script, you may notice the same word spread across multiple categories of emotions. The word “lord” is incredibly prevalent throughout the script in general, it should come as no surprise that depending on the context of its usage, the emotion it gives off varies. A character may express for a certain “lord” or exhibit the upmost respect for a certain “lord”, which leads to that word falling into the “positive” category.

```
tidy_script %>%
  inner_join(get_sentiments("nrc"))%>%
  count(word,sentiment,sort = TRUE)%>%
  group_by(sentiment)%>%
  top_n(10)%>%
  ungroup()%>%
  mutate(word=reorder (word,n))%>%
  ggplot(aes(word,n,fill=sentiment))+
  geom_col(show.legend = FALSE)+
  facet_wrap(~sentiment,ncol=,scales = "free")+
  coord_flip()
```

```
## Joining with ‘by = join_by(word)‘
```

```
## Warning in inner_join(., get_sentiments("nrc")): Detected an unexpected many-to-many relationship be
## i Row 1 of `x` matches multiple rows in `y`.
## i Row 6283 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.

## Selecting by n
```



## Positivity and Polarity

Game of Thrones isn't necessarily known for being all sunshine and rainbows, in fact, it's rather bleak, aggressive, and even downright depressing at points. Of course, it isn't like this the entirety of the show, there are pockets of happiness and positivity throughout its course. The next analysis dives a little further into the previous sentiment analysis where we focused on general sentiment across the script but this time around, putting a focus on each individual season and gathering insights on which seasons exhibited the most positivity. Based on the results below, we can see that seasons 8 and season 2 were the two most positive seasons. Perhaps due to season 2 still being very early on in the series and with season 8 being the final season, the light at the end of the tunnel can finally be viewed.

```r
got_sentiment_season <- got_nrc %>%
  group_by(Season) %>%
  mutate(Season_total = n()) %>%
  ungroup()

got_sentiment_season %>%
  count(Season, sentiment, Season_total) %>%
  mutate(percent = n / Season_total * 100) %>%
  filter(sentiment == "positive") %>%
  arrange(desc(percent))
```

```
## # A tibble: 8 x 5
##   Season   sentiment Season_total     n percent
##   <chr>    <chr>            <int> <int>   <dbl>
## 1 Season 8 positive          3005   648    21.6
## 2 Season 2 positive         11789  2482    21.1
## 3 Season 7 positive          6862  1437    20.9
## 4 Season 5 positive          9754  2036    20.9
## 5 Season 1 positive          9830  2009    20.4
## 6 Season 4 positive         10415  2120    20.4
## 7 Season 3 positive         10670  2167    20.3
## 8 Season 6 positive          8801  1686    19.2
```

While the "nrc" lexicon contained the most word matches when compared to the script, the "bing" lexicon is better suited for determining what words fall strictly into the "positive" and "negative" tonal categories. We can create what is called a "Word Cloud" to visualize the words in the script and whether they fall into the "positive" or "negative" category. The code below will only pull in 100 words, so we will be focusing on the top 100 words from the script that contribute to either of these two categories.

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(reshape2)
```

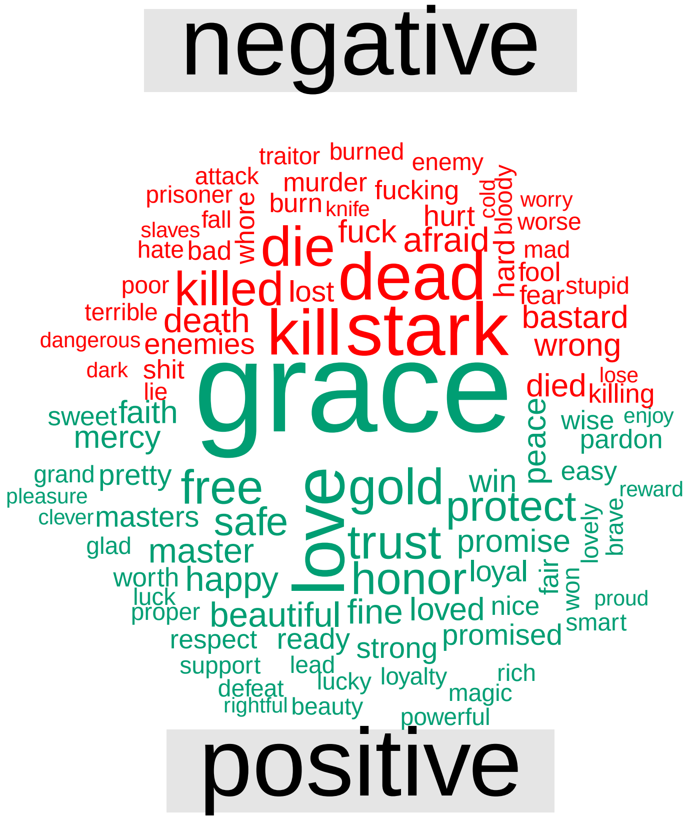
```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
my_colors <- c("#E69F00", "#56B4E9", "#009E73", "#CC79A7", "#D55E00",
"#FF0000")

tidy_script %>%
inner_join(get_sentiments("bing")) %>%
count(word, sentiment, sort = TRUE) %>%
acast(word ~ sentiment, value.var = "n", fill = 0) %>%
comparison.cloud(colors=c(my_colors[6],my_colors[3]),max.words = 100)
```

```
## Joining with `by = join_by(word)`
```



## Common Word Pairs

Thus far we have focused on analyzing the individual word frequencies and their emotional categorization across each season. Our next analysis will put more of a focus on the characters, starting with the most commonly spoken pairs of words in the script. We can begin this analysis by taking the original "got_script" that imported our script into and use it to create a new object called "bigrams". Prior to this point, we had tokenized the script into unigrams (one word per row), however this next analysis will focus on bigrams (two pairs of words, per row).

The code below tokenizes the words from the "Sentence" column into bigrams. Once this has been completed, a new object called "bigrams_separated" is created to split the two consecutive pairs of words per row, into their own unique columns, "word1" and "word2". A little extra cleanup is performed after this action to removed any stop words from this list of words and the list of word pairs are then counted and sorted in descending order by frequency of appearance in the script together.

```
bigrams <- got_script %>%
  unnest_tokens(bigrams, Sentence, token = "ngrams", n = 2) %>%
  filter(!is.na(bigrams))
```

```
## Warning: Outer names are only allowed for unnamed scalar atomic inputs
```

```
bigrams
```

```
## # A tibble: 273,513 x 6
##    ‘Release Date‘ Season   ‘Episode Number‘ ‘Episode Title‘ Character   bigrams
##    <date>         <chr>    <chr>            <chr>           <chr>       <chr>
##  1 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ what do
##  2 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ do you
##  3 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ you ex~
##  4 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ expect~
##  5 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ they a~
##  6 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ are sa~
##  7 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ savage~
##  8 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ one lot
##  9 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ lot st~
## 10 2011-04-17     Season 1 Episode 1        Winter is Coming waymar roy~ steals~
## # i 273,503 more rows
```

```
bigrams_separated <- bigrams %>%
  separate(bigrams, c("word1", "word2"), sep = " ")


bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

bigrams_filtered
```

```
## # A tibble: 18,453 x 7
##    ‘Release Date‘ Season  ‘Episode Number‘ ‘Episode Title‘ Character word1 word2
##    <date>         <chr>   <chr>            <chr>           <chr>     <chr> <chr>
##  1 2011-04-17     Season~ Episode 1        Winter is Comi~ waymar r~ lot   stea~
##  2 2011-04-17     Season~ Episode 1        Winter is Comi~ royce     dead  frig~
##  3 2011-04-17     Season~ Episode 1        Winter is Comi~ royce     south run
##  4 2011-04-17     Season~ Episode 1        Winter is Comi~ royce     moved camp
##  5 2011-04-17     Season~ Episode 1        Winter is Comi~ jon snow  fath~ watc~
##  6 2011-04-17     Season~ Episode 1        Winter is Comi~ eddard s~ prac~ bran
##  7 2011-04-17     Season~ Episode 1        Winter is Comi~ robb sta~ bow   arm
##  8 2011-04-17     Season~ Episode 1        Winter is Comi~ jonrobb   quick bran
##  9 2011-04-17     Season~ Episode 1        Winter is Comi~ jonrobb   bran  fast~
## 10 2011-04-17     Season~ Episode 1        Winter is Comi~ cassel    lord  stark
## # i 18,443 more rows
```

```
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts
```

```
## # A tibble: 12,537 x 3
##    word1    word2         n
##    <chr>    <chr>     <int>
##  1 king     landing     194
##  2 jon      snow        127
##  3 night    watch       122
##  4 castle   black        91
##  5 lord     commander    84
##  6 ned      stark        71
##  7 iron     throne       63
##  8 casterly rock         57
##  9 lord     baelish      51
## 10 ser      davos        51
## # i 12,527 more rows
```

## igraph

Now that the most common word pairs across each season have been established, we can use the “igraph” package to better analyze this network of word pairings.

```r
library(igraph)
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:lubridate':
##
##     %--%, union

## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union

## The following objects are masked from 'package:purrr':
##
##     compose, simplify

## The following object is masked from 'package:tidyr':
##
##     crossing

## The following object is masked from 'package:tibble':
##
##     as_data_frame

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```r
bigram_graph <- bigram_counts %>%
    filter(n > 20) %>%
    graph_from_data_frame((directed = FALSE))

bigram_graph
```

```
## IGRAPH b443d75 UN-- 66 50 --
## + attr: name (v/c), n (e/n)
## + edges from b443d75 (vertex names):
##  [1] king    --landing   jon     --snow      night   --watch
##  [4] castle  --black     lord    --commander ned     --stark
##  [7] iron    --throne    casterly--rock      lord    --baelish
## [10] ser     --davos     robb    --stark     lady    --stark
## [13] lady    --sansa     lord    --stark     tywin   --lannister
## [16] iron    --islands   lord    --tyrion    king    --mad
## [19] white   --walkers   ser     --jaime     jaime   --lannister
## [22] stannis --baratheon ser     --loras     sansa   --stark
## + ... omitted several edges
```

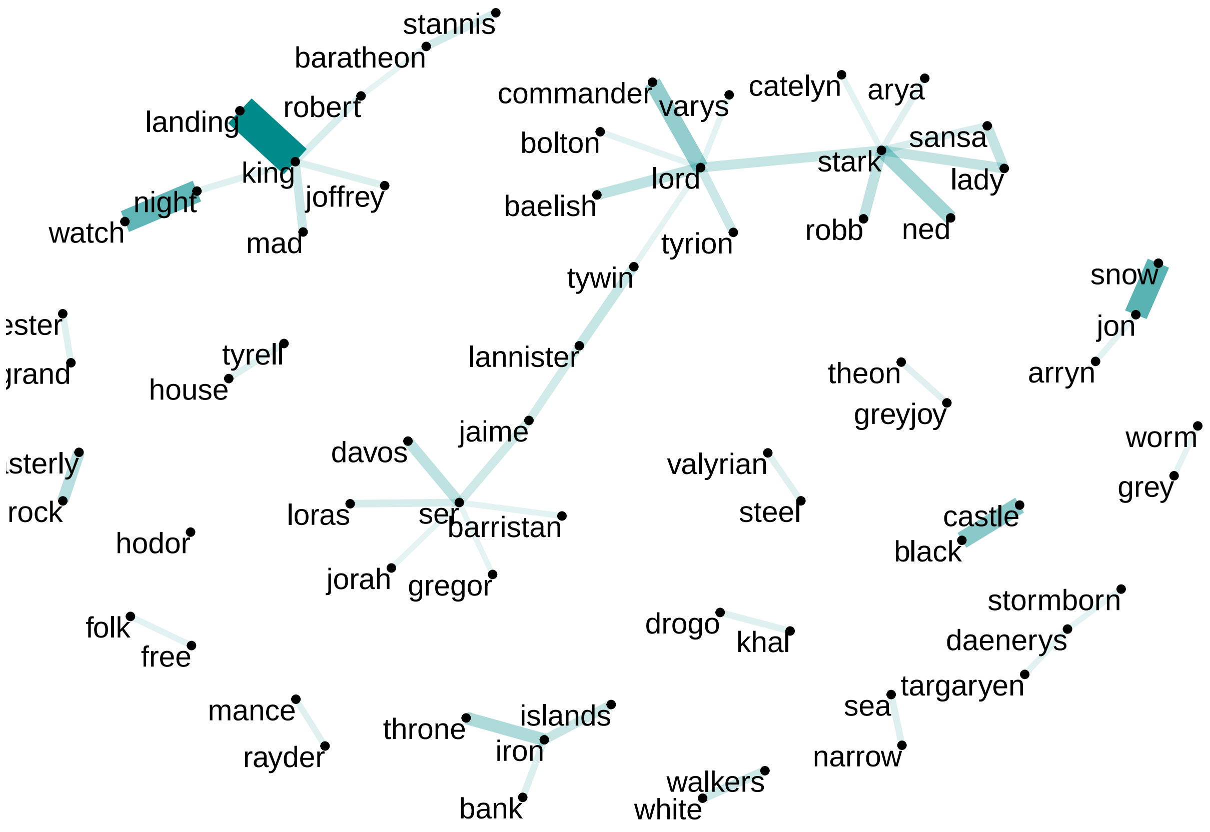
We can then use the "ggraph" to visualize these common word pairings. We use the argument "geom_edge_link" to highlight the frequencies of certain word pairings, the darker the link between two words is, means a more prominent pairing.

```r
library(ggraph)

set.seed(2017)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n), show.legend = FALSE,edge_colour = "cyan4") +
  geom_node_point(size = 1) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```

```
## Warning: The ‘trans‘ argument of ‘continuous_scale()‘ is deprecated as of ggplot2 3.5.0.
## i Please use the ‘transform‘ argument instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



```r
pairwise_script <- got_script %>%
  unnest_tokens(word, Sentence, token = "words")
```

```
## Warning: Outer names are only allowed for unnamed scalar atomic inputs
```

```r
pairwise_script
```

```
## # A tibble: 297,424 x 6
##    `Release Date` Season   `Episode Number` `Episode Title`  Character    word
##    <date>         <chr>    <chr>            <chr>            <chr>        <chr>
##  1 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce what
##  2 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce do
##  3 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce you
##  4 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce expect
##  5 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce they
##  6 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce are
##  7 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce savag~
##  8 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce one
##  9 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce lot
## 10 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce steals
## # i 297,414 more rows
```

```r
pairwise_filtered <- pairwise_script %>%
  anti_join(stop_words) %>%
  filter(nchar(word) >= 3)
```

```
## Joining with ‘by = join_by(word)‘
```

```
pairwise_filtered$word <- gsub("\\s+","", pairwise_filtered$word)
pairwise_filtered$word <- gsub("[^a-zA-Z]","", pairwise_filtered$word)

pairwise_filtered
```

```
## # A tibble: 87,875 x 6
##    ‘Release Date‘ Season   ‘Episode Number‘ ‘Episode Title‘  Character    word
##    <date>         <chr>    <chr>            <chr>            <chr>        <chr>
##  1 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce expect
##  2 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce savag~
##  3 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce lot
##  4 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce steals
##  5 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce goat
##  6 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce lot
##  7 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce rippi~
##  8 2011-04-17     Season 1 Episode 1        Winter is Coming waymar royce pieces
##  9 2011-04-17     Season 1 Episode 1        Winter is Coming will         wildl~
## 10 2011-04-17     Season 1 Episode 1        Winter is Coming will         life
## # i 87,865 more rows
```

```
pairwise_counts <- pairwise_filtered %>%
  count(word, sort = TRUE)

pairwise_counts
```

```
## # A tibble: 8,799 x 2
##    word        n
##    <chr>   <int>
##  1 lord     1112
##  2 king     1111
##  3 father    766
##  4 grace     534
##  5 time      523
##  6 lady      499
##  7 queen     476
##  8 stark     450
##  9 brother   431
## 10 north     421
## # i 8,789 more rows
```

Next, we will proceed to count the total number of times two words appear together consecutively, within the script of Game of Thrones, using the pairwise_count() function. We can see that the most common pairings included king and lord, father and lord, king and father, etc.

```
library(widyr)

word_pairs <- pairwise_filtered %>%
  pairwise_count(word, Character, sort = TRUE)

word_pairs
```

```
## # A tibble: 18,436,340 x 3
##    item1  item2      n
##    <chr>  <chr>  <dbl>
##  1 king   lord     120
##  2 lord   king     120
##  3 lord   father   100
##  4 father lord     100
##  5 king   father    95
##  6 father king      95
##  7 time   lord      94
##  8 lord   time      94
##  9 night  lord      92
## 10 lord   night     92
## # i 18,436,330 more rows
```

To paint a picture of which words are most commonly spoken together in sequence of one another, we can use graph_from_data_frame() to create a cluster-like visualization, words shown in groups with line links between each other appear in the script together.

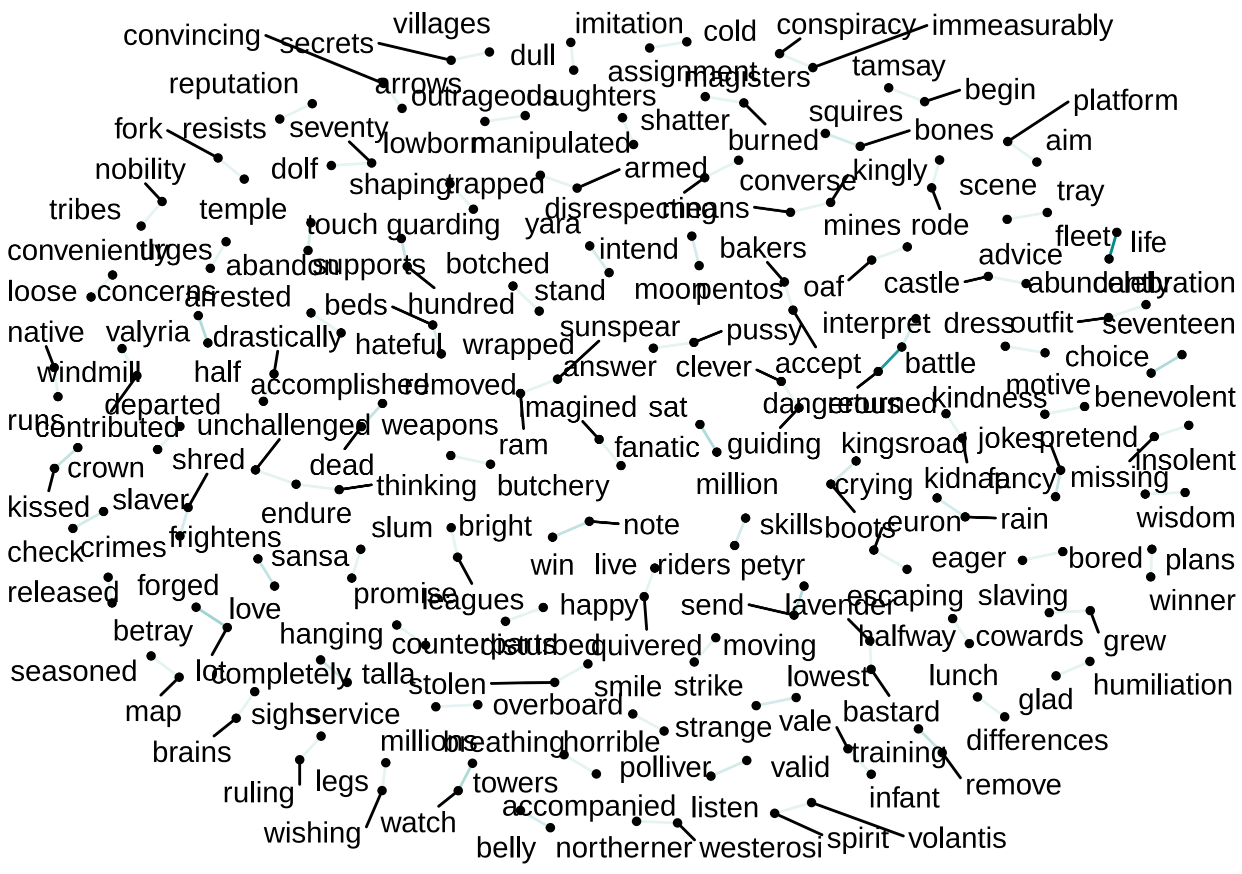
```r
library(igraph)
library(ggraph)
library(dplyr)

set.seed(2016)

filtered_word_pairs <- word_pairs %>%
  sample_n(100) %>%
  filter(n() >= 60)

# Create graph
graph <- graph_from_data_frame(filtered_word_pairs)

# Plot graph
ggraph(graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), edge_colour = "cyan4", show.legend = FALSE) +
  geom_node_point(size = 1) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```



## Final Report

There's a ton you can explore when it comes to analyzing the script of a television show and many alternative approaches that can be implemented in one's analysis. In this analysis, we explored the entirety of the script from the Game of Thrones television adaptation across each season of the show. The types of analyses performed throughout this project included Term Frequency Distribution across each season, TF-IDF of the most important words per season, Sentiment analysis of the words used within each season, and common words spoken by characters during the show and the most commonly used pairs of words.

Based on the results of these analyses it can be concluded that although certain Seasons of Game of Thrones were slightly more upbeat than others based on terminology used in the Season, the overall consensus is that each Season of Game of Thrones primarily radiates negative emotion. It should be evident at this point that the most commonly used word throughout the entire show was "Lord", though it is used in a wide variety of contexts, most commonly used when a character is required to formally address a male of an important family of higher status. TF-IDF helped us determine the most important words per season, which can give us a better idea of what may have been going on in a given season. Using Season 4 as an example, "Reek" is a very important word. This would be due to Theon Greyjoy's capture, torture, and imprisonment by Ramsey Bolton, where he was consistently demeaned and addressed by his new "name", Reek. This was an incredibly important moment in the series and was an arc shown regularly throughout the Fourth Season. Using Season Eight as another example, "Tyrant" is a key word for that season. This word was prominent as Daenerys Targaryen had fallen into madness, living up to her ancestor the Mad King, and wrecking havoc across the Seven Kingdoms throughout the final season.

Sentiment Analysis helped make observations of key words associated with certain moods throughout the show, such as positivity, negativity, sadness, joy, etc. It was also found that the polarity of each season was overwhelmingly negative, with some seasons leaning far more into that negativity than others.

Lastly, we were able to determine the most common word pairings and most commonly spoken words by each character throughout the show. Lord, king, father, and grace being the words that saturate most of the script. Word pairings such as “King, Lord” or “Lord, Father” being the most prominent pairs of words spoken by characters. Using tools such as ggraph and igraphs, we were able to visualize networks of these common words, their relevance to each other, and their overall importance to the show’s dialogue.