DATA607 Assignment 10

Alexander Simon

2024-03-31

Contents

0. Packages

In addition to the packages used in the textbook portion of this assignment, I used the SentimentAnalysis, RColorBrewer, maps, and plotly packages. If needed, you can install them using the command(s) below.

```
install.packages("SentimentAnalysis")
install.packages("RColorBrewer")
install.packages("maps")
install.packages("plotly")
```

1. Introduction

Sentiment analysis is a technique to understand the attitudes and opinions from text. In this assignment, I first implement the base code described in *Text Mining with R*, chapter 2. Then I extend these methods to analyze Yelp customer reviews using the 3 sentiment lexicons from chapter 2 (AFINN, Bing, and NRC) along with a psychosocial sentiment lexicon called the General Inquirer (see 3.3. GI sentiment lexicon for details). Finally, I combined the customer reviews with Yelp business location data and explored how geospatial analysis could be used with sentiment analysis to inform business decisions.

2. Implementing the base code

2.1. Data

Get AFINN sentiment lexicon[1]

```r
get_sentiments("afinn")
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # i 2,467 more rows
```

Get Bing sentiment lexicon[2]

```r
get_sentiments("bing")
```

```
## # A tibble: 6,786 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faces     negative
##  2 abnormal    negative
##  3 abolish     negative
##  4 abominable  negative
##  5 abominably  negative
##  6 abominate   negative
##  7 abomination negative
```

[1] Finn Årup Nielsen. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages 718 in CEUR Workshop Proceedings* 93-98. 2011 May. http://arxiv.org/abs/1103.2903.

[2] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2004)*, Seattle, Washington, USA, Aug 22-25, 2004. https://www.cs.uic.edu/~liub/publications/kdd04-revSummary.pdf

```
##   8 abort       negative
##   9 aborted     negative
##  10 aborts      negative
## # i 6,776 more rows
```

Get NRC sentiment lexicon[3]

```
get_sentiments("nrc")
```

```
## # A tibble: 13,872 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abacus      trust
##  2 abandon     fear
##  3 abandon     negative
##  4 abandon     sadness
##  5 abandoned   anger
##  6 abandoned   fear
##  7 abandoned   negative
##  8 abandoned   sadness
##  9 abandonment anger
## 10 abandonment fear
## # i 13,862 more rows
```

2.2. Analysis

2.2.1. Joyful words in *Emma* First, tidy the text

```
tidy_books <- austen_books() %>%
  group_by(book) %>%
  mutate(
    linenumber = row_number (),
    chapter = cumsum (str_detect (text, regex("^chapter [\\divxlc]", ignore_case = TRUE)))
  ) %>%
  ungroup() %>%
  unnest_tokens(word, text)
```

Filter the text needed for sentiment analysis—the joyful words in the NRC lexicon and the books for text from Emma. From these, determine the most common joyful words in *Emma*.

```
nrc_joy <- get_sentiments ("nrc") %>%
  filter(sentiment == "joy")

tidy_books %>%
  filter(book == "Emma") %>%
  inner_join(nrc_joy, by = join_by("word")) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 301 x 2
```

[3] Saif Mohammad and Peter Turney, Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence,* 29 (3), 436-465, 2013. https://arxiv.org/pdf/1308.6297.pdf

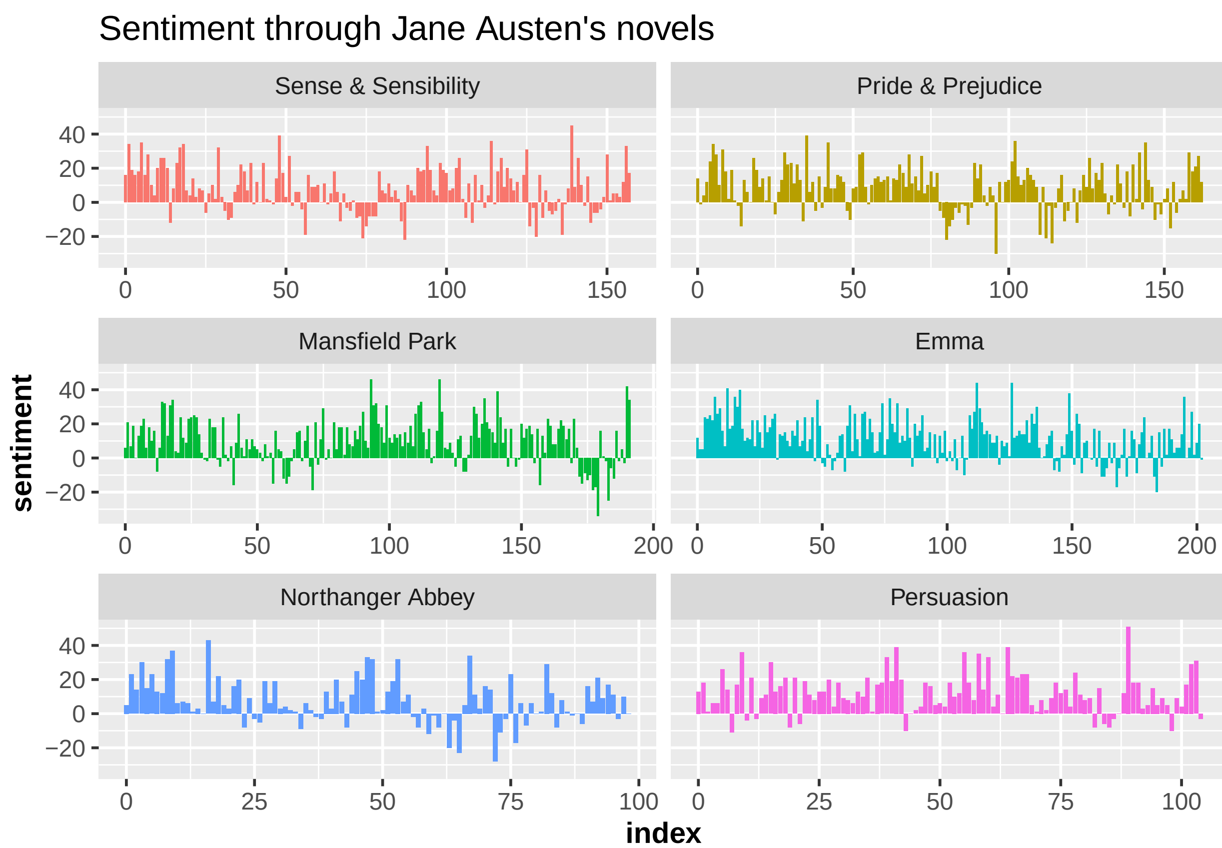
```
##    word          n
##    <chr>     <int>
##  1 good        359
##  2 friend      166
##  3 hope        143
##  4 happy       125
##  5 love        117
##  6 deal         92
##  7 found        92
##  8 present      89
##  9 kind         82
## 10 happiness    76
## # i 291 more rows
```

2.2.2. Change in sentiment across Jane Austen's novels First, determine the sentiment score for each word. Then count the number of positive and negative sentiment words in each book and calculate a net sentiment score (ie, the difference between positive and negative scores).

```
jane_austen_sentiment <- tidy_books %>%
  inner_join(get_sentiments("bing"), by = join_by("word")) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
  mutate(
    sentiment = positive - negative
  )
```

Plot sentiment scores across each novel (more specifically, across the index of 80-line sections of text).

```
ggplot(jane_austen_sentiment, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x") +
  theme(axis.title = element_text(face = "bold")) +
  ggtitle("Sentiment through Jane Austen's novels")
```



2.2.3. Change in sentiment across *Pride & Prejudice* with different lexicons First filter the words of interest

```
pride_prejudice <- tidy_books %>%
  filter(book == "Pride & Prejudice")

pride_prejudice
```

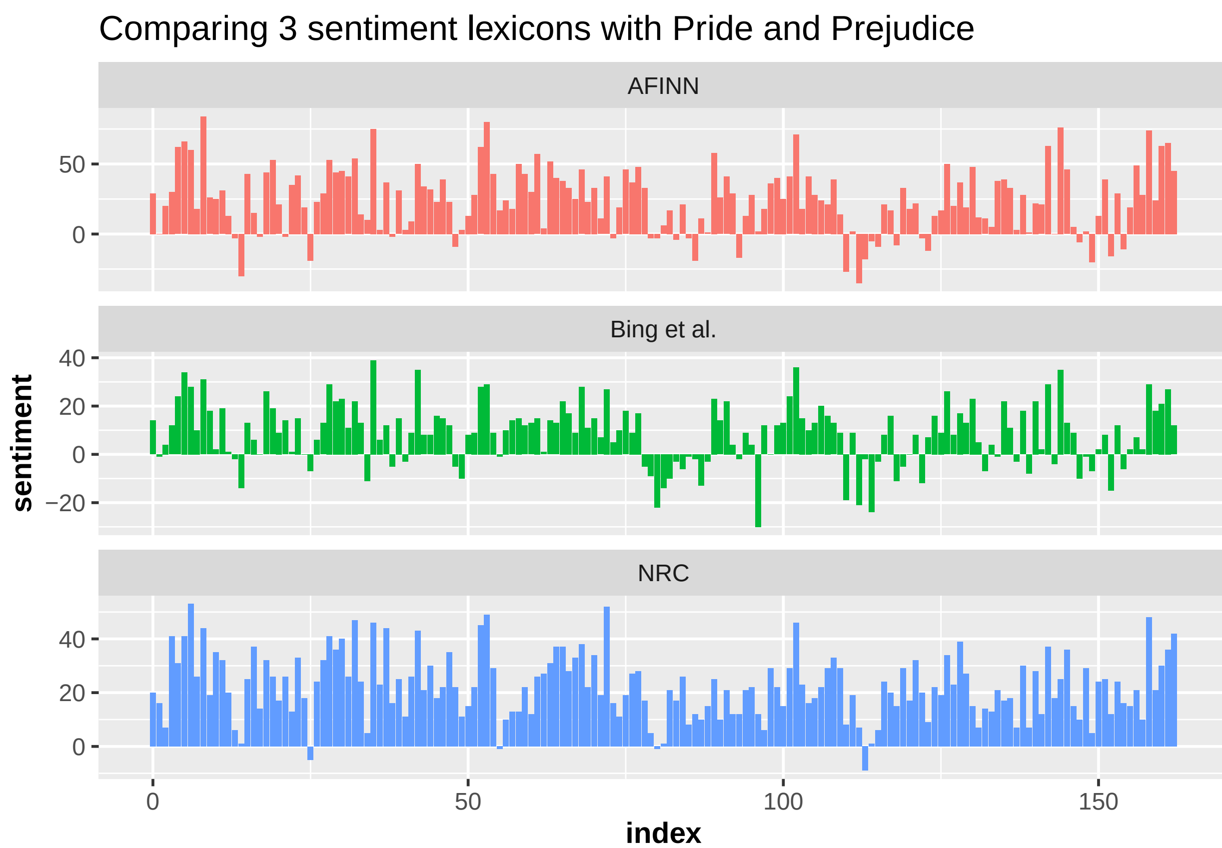
```
## # A tibble: 122,204 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Pride & Prejudice            1       0 pride
##  2 Pride & Prejudice            1       0 and
##  3 Pride & Prejudice            1       0 prejudice
##  4 Pride & Prejudice            3       0 by
##  5 Pride & Prejudice            3       0 jane
##  6 Pride & Prejudice            3       0 austen
##  7 Pride & Prejudice            7       1 chapter
##  8 Pride & Prejudice            7       1 1
##  9 Pride & Prejudice           10       1 it
## 10 Pride & Prejudice           10       1 is
## # i 122,194 more rows
```

Then calculate the sentiment scores with the different lexicons

```r
afinn <- pride_prejudice %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index = linenumber %/% 80) %>%
  summarise(
    sentiment = sum(value)
  ) %>%
  mutate(
    method = "AFINN"
  )

bing_and_nrc <- bind_rows(
  pride_prejudice %>%
    inner_join(get_sentiments("bing")) %>%
    mutate(
      method = "Bing et al."
    ),
  pride_prejudice %>%
    inner_join(get_sentiments("nrc") %>%
                 filter(sentiment %in% c("positive", "negative"))
    ) %>%
    mutate(
      method = "NRC"
    )) %>%
  count(method, index = linenumber %/% 80, sentiment) %>%
  pivot_wider(names_from = sentiment,
              values_from = n,
              values_fill = 0) %>%
  mutate(
    sentiment = positive - negative
  )
```

Bind the results and visualize them

```r
bind_rows(afinn, bing_and_nrc) %>%
  ggplot(aes(index, sentiment, fill = method)) +
    geom_col(show.legend = FALSE) +
    facet_wrap(~method, ncol = 1, scales = "free_y") +
    theme(axis.title = element_text(face = "bold")) +
    ggtitle("Comparing 3 sentiment lexicons with Pride and Prejudice")
```



2.2.4. Number of positive and negative words in different lexicons NRC

Note: The n's are slightly different from those shown in the book (see commented lines below). I assume this is because the lexicon has been updated since the time that the book was written.

```
get_sentiments("nrc") %>%
  filter(sentiment %in% c("positive", "negative")) %>%
  count(sentiment)
```

```
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative   3316
## 2 positive   2308
```

```
#> 1 negative   3324
#> 2 positive   2312
```

Bing

```
get_sentiments("bing") %>%
  count(sentiment)
```

```
## # A tibble: 2 x 2
```

```
##   sentiment     n
##   <chr>     <int>
## 1 negative   4781
## 2 positive   2005
```

```
#> 1 negative   4781
#> 2 positive   2005
```

2.2.5. Most common positive and negative words in the Bing lexicon

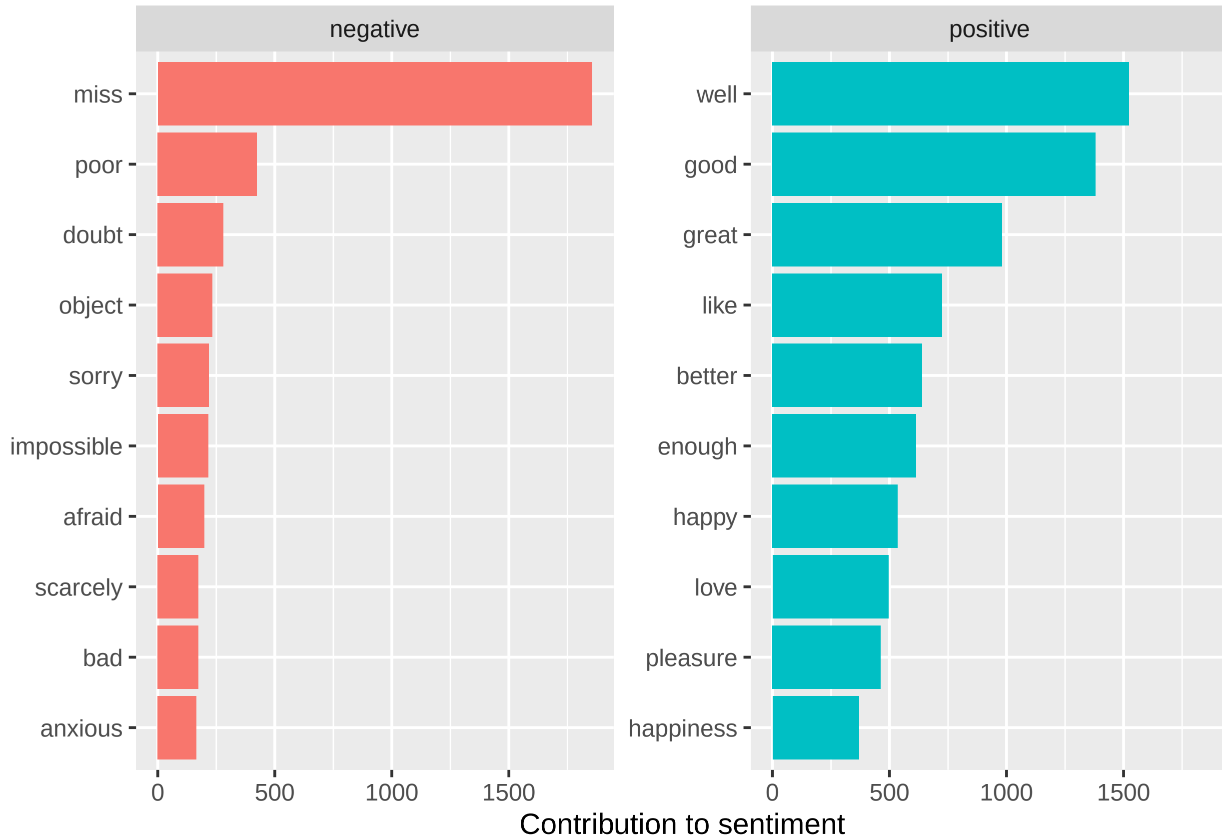
Calculate word counts

```
bing_word_counts <- tidy_books %>%
  inner_join(get_sentiments("bing"), by = join_by("word")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

bing_word_counts
```

```
## # A tibble: 2,585 x 3
##    word     sentiment     n
##    <chr>    <chr>     <int>
##  1 miss     negative   1855
##  2 well     positive   1523
##  3 good     positive   1380
##  4 great    positive    981
##  5 like     positive    725
##  6 better   positive    639
##  7 enough   positive    613
##  8 happy    positive    534
##  9 love     positive    495
## 10 pleasure positive    462
## # i 2,575 more rows
```

Visual comparison

```
bing_word_counts %>%
  group_by(sentiment) %>%
  slice_max(n, n = 10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(x = "Contribution to sentiment", y = NULL)
```



2.2.6. Custom stop words Jane Austen used "miss" as a title, not a negative emotion. To prevent the analysis from being skewed by this word, we can add it to a custom stop word list:

```
custom_stop_words <- bind_rows(tibble(word = c("miss"), lexicon = c("custom")),
                               stop_words)
custom_stop_words
```

```
## # A tibble: 1,150 x 2
##    word        lexicon
##    <chr>       <chr>
##  1 miss        custom
##  2 a           SMART
##  3 a's         SMART
##  4 able        SMART
##  5 about       SMART
##  6 above       SMART
##  7 according   SMART
##  8 accordingly SMART
##  9 across      SMART
## 10 actually    SMART
## # i 1,140 more rows
```

2.2.7. Wordclouds Visualize the most common words in Jane Austen's novels

```r
tidy_books %>%
  anti_join(stop_words, by = join_by("word")) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```

Tag the positive and negative words using the Bing lexicon, then visualize the most common ones with a wordcloud.

```r
tidy_books %>%
  inner_join(get_sentiments("bing"), by = join_by("word")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("gray20", "gray80"), max.words = 100)
```

2.2.8. Looking at units beyond words An example of tokenizing into sentences

```
p_and_p_sentences <- tibble(text = prideprejudice) %>%
  unnest_tokens(sentence, text, token = "sentences")

p_and_p_sentences$sentence[2]
```

```
## [1] "by jane austen"
```

An example of splitting tokens using a regex pattern to divide Jane Austen's novels by chapter

```
austen_chapters <- austen_books() %>%
  group_by(book) %>%
  unnest_tokens(chapter, text, token = "regex", pattern = "Chapter|CHAPTER [\\dIVXLC]") %>%
  ungroup()

austen_chapters %>%
  group_by(book) %>%
  summarise(chapters = n())
```

```
## # A tibble: 6 x 2
##   book                chapters
##   <fct>                  <int>
## 1 Sense & Sensibility       51
```

```
## 2 Pride & Prejudice         62
## 3 Mansfield Park            49
## 4 Emma                      56
## 5 Northanger Abbey          32
## 6 Persuasion                25
```

2.2.9. Sentiment analysis by chapter

Get negative words in Bing lexicon

```r
bingnegative <- get_sentiments("bing") %>%
  filter(sentiment == "negative")
```

Create a dataframe of the number of words in each chapter

```r
wordcounts <- tidy_books %>%
  group_by(book, chapter) %>%
  summarize(words = n())
```

Calculate proportion of negative words in each chapter

```r
tidy_books %>%
  semi_join(bingnegative, by = join_by("word")) %>%
  group_by(book, chapter) %>%
  summarize(negativewords = n()) %>%
  left_join(wordcounts, by = c("book", "chapter")) %>%
  mutate(
    # I added round() so output matches the book
    ratio = round(negativewords/words, 4)
  ) %>%
  filter(chapter != 0) %>%
  slice_max(ratio, n = 1) %>%
  ungroup()
```

```
## # A tibble: 6 x 5
##   book                chapter negativewords words  ratio
##   <fct>                 <int>         <int> <int>  <dbl>
## 1 Sense & Sensibility      43           161  3405 0.0473
## 2 Pride & Prejudice        34           111  2104 0.0528
## 3 Mansfield Park           46           173  3685 0.0469
## 4 Emma                     15           151  3340 0.0452
## 5 Northanger Abbey         21           149  2982 0.05
## 6 Persuasion                4            62  1807 0.0343
```

3. Extending to another text source and sentiment lexicon

3.1. Text source

I obtained customer reviews from the Yelp Open Dataset, which contains nearly 7 million reviews of 150,000 businesses. Considering the size of the raw data (>5 Gb), GitHub storage limits (generally 100 Mb, but up to 2 Gb with large file storage), and that R loads all data in memory, I decided that 50,000 reviews would be sufficient and manageable for analysis.

I extracted the first 50,000 lines from the source file on the command line.

```
head -50000 yelp_reviews.json > yelp_reviews50K.json
```

3.2. Text data transformations

3.2.1. Command-line transformations The raw data from Yelp was supposedly in JSON format; however, it failed a JSON validator, which flagged multiple root elements and missing commas between objects. I corrected these issues on the command line.

```
# add comma to end of each line
sed "s/$/,/g" yelp_reviews50K.json > yelp_reviews50K2.json
# remove the last one
# 2 characters are truncated because there is also a \n at the end of the line
truncate -s -2 yelp_reviews50K2.json
# add a top-level root element called "reviews"
# prepend
sed -i.old '1s;^;{ "reviews": [\n;' yelp_reviews50K2.json
# postpend
echo "] }" >> yelp_reviews50K2.json
```

3.2.2. Convert JSON data to dataframe After validating the corrected JSON data, I pushed the file to my GitHub repository and then read it into R.

```
yelp_reviews <- fromJSON("https://github.com/alexandersimon1/Data607/raw/main/Assignment10/yelp_reviews5
```

The data structure is shown below.

```
str(yelp_reviews)
```

```
## List of 1
##  $ reviews:'data.frame': 50000 obs. of  9 variables:
##   ..$ review_id  : chr [1:50000] "KU_O5udG6zpxOg-VcAEodg" "BiTunyQ73aT9WBnpR9DZGw" "saUsX_uimxRlCVr6
##   ..$ user_id    : chr [1:50000] "mh_-eMZ6K5RLWhZyISBhwA" "OyoGAe7OKpv6SyGZT5g77Q" "8g_iMtfSiwikVnbP
##   ..$ business_id: chr [1:50000] "XQfwVwDr-v0ZS3_CbbE5Xw" "7ATYjTIgM3jUlt4UM3IypQ" "YjUWPpI6HXG530lw
##   ..$ stars      : num [1:50000] 3 5 3 5 4 1 5 5 3 3 ...
##   ..$ useful     : int [1:50000] 0 1 0 1 1 1 0 2 1 0 ...
##   ..$ funny      : int [1:50000] 0 0 0 0 0 2 2 0 1 0 ...
##   ..$ cool       : int [1:50000] 0 1 0 1 1 1 0 0 0 0 ...
##   ..$ text       : chr [1:50000] "If you decide to eat here, just be aware it is going to take about
##   ..$ date       : chr [1:50000] "2018-07-07 22:09:11" "2012-01-03 15:28:18" "2014-02-05 20:30:30" "
```

I binded the columns into a dataframe, selected the relevant columns, and converted the date column to a date type.

```
yelp_reviews_df <- bind_cols(yelp_reviews[[1]])
yelp_reviews_df <- yelp_reviews_df %>%
  select(review_id, business_id, text, date, review_stars = stars) %>%
  mutate(
    date = as.Date(date)
  )
```

3.2.3. Transformations for text mining I removed numbers, replaced hyphen/dashes with spaces, and stripped extra white space. Note that `unnest_tokens()` will take care of punctuation and change case to lowercase.

```
yelp_reviews_df <- yelp_reviews_df %>%
  mutate(
    text = str_replace_all(text, "-", " "),
    text = str_replace_all(text, "\\d", ""),
    text = str_replace_all(text, "\\s{2,}", " ")
  )
```

Then I tokenized the text and removed stop words.

```
review_words <- yelp_reviews_df %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")
```

3.3. GI sentiment lexicon

In addition to the AFINN, Bing, and NRC lexicons used in chapter 2 of *Text Mining with R,* I used the "General Inquirer" (GI) dictionary, which includes positive and negative words from the Harvard IV-4 psychosocial dictionary and Lasswell value dictionary.[4] The GI dictionary is available in the `SentimentAnalysis` R package.

Load the dictionary

```
gi_dict <- loadDictionaryGI()
```

The dictionary is structured as a list of 2 character vectors of positive and negative words.

```
str(gi_dict)
```

```
## List of 2
##  $ positiveWords: chr [1:1316] "abid" "abil" "abl" "abound" ...
##  $ negativeWords: chr [1:1746] "abandon" "abat" "abdic" "abhor" ...
##  - attr(*, "class")= chr "SentimentDictionaryBinary"
```

I converted each list to a dataframe similar to the other three sentiment lexicons.

```
gi_positive <- tibble(word = gi_dict[["positiveWords"]], sentiment = c("positive"))
gi_negative <- tibble(word = gi_dict[["negativeWords"]], sentiment = c("negative"))
gi_all <- bind_rows(gi_positive, gi_negative)
```

3.3.1. Number of positive and negative words in GI lexicon vs other lexicons To include AFINN in the comparison, I first defined positive and negative sentiments.

[4] Dunphy, DC (1974). Harvard IV-4 Dictionary General Inquirer project. Sydney: University of New South Wales.

```
afinn_sentiments <- get_sentiments("afinn") %>%
  mutate(
    sentiment = case_when(value < 0 ~ "negative",
                          value == 0 ~ "neutral",
                          value > 0 ~ "positive")
  )

afinn_positive <- afinn_sentiments %>%
  filter(sentiment == "positive")

afinn_negative <- afinn_sentiments %>%
  filter(sentiment == "negative")

afinn_all <- bind_rows(afinn_positive, afinn_negative)
```

Then I counted the number of positive and negative sentiments in each lexicon.

```
# AFINN
afinn_counts <- afinn_all %>%
  count(sentiment) %>%
  mutate(
    prop = round(n / sum(n), 2)
  )

# Bing
bing_counts <- get_sentiments("bing") %>%
  count(sentiment) %>%
  mutate(
    prop = round(n / sum(n), 2)
  )

# NRC
nrc_counts <- get_sentiments("nrc") %>%
  filter(sentiment %in% c("positive", "negative")) %>%
  count(sentiment) %>%
  mutate(
    prop = round(n / sum(n), 2)
  )

# GI
gi_counts <- gi_all %>%
  count(sentiment) %>%
  mutate(
    prop = round(n / sum(n), 2)
  )
```

All the lexicons have more negative words than positive words. The proportions of negative and positive words in the GI lexicon is most similar to the NRC lexicon.

```
lexicon_counts <- bind_cols(afinn_counts, bing_counts$n, bing_counts$prop,
                            nrc_counts$n, nrc_counts$prop,
                            gi_counts$n, gi_counts$prop)
```

```r
colnames(lexicon_counts) <- c("sentiment", "afinn", "prop", "bing", "prop", "nrc", "prop", "gi", "prop")
lexicon_counts
```

```
## # A tibble: 2 x 9
##   sentiment afinn  prop  bing  prop   nrc  prop    gi  prop
##   <chr>     <int> <dbl> <int> <dbl> <int> <dbl> <int> <dbl>
## 1 negative   1598  0.65  4781   0.7  3316  0.59  1746  0.57
## 2 positive    878  0.35  2005   0.3  2308  0.41  1316  0.43
```

3.4. Sentiment analysis

Please note that, due to differences between books and customer reviews, this section is not an exact duplicate of the analyses in *Text Mining with R*, chapter 2 (Section 2. Implementing the base code). However, I have tried to perform similar analyses.

3.4.1. Most common words in Yelp reviews First, I counted the most common words in the Yelp reviews. Not surprisingly, the most frequent words are related to topics that one would expect in reviews of businesses (eg, time, service, staff). The most frequent word "food" suggests that most of the Yelp reviews are about restaurants.

```r
review_words %>%
  count(word, sort = TRUE) %>%
  slice_head(n = 10)
```

```
##          word     n
## 1        food 27464
## 2     service 16352
## 3        time 15084
## 4        nice  9276
## 5       staff  8208
## 6        love  7964
## 7    friendly  7813
## 8   delicious  7583
## 9  restaurant  7398
## 10    chicken  7158
```

Word cloud of the most common words

```r
set.seed(1234)
word_counts <- review_words %>%
  count(word)

wordcloud(words = word_counts$word, freq = word_counts$n, min.freq = 50,
          max.words = 50, random.order = FALSE, rot.per = 0,
          colors = brewer.pal(5, "Dark2"))
```

3.4.2. Comparison of most common positive and negative words in Yelp reviews using different lexicons I created a function to get the 10 most common words in the Yelp reviews using a specified lexicon and sentiment category (positive or negative). This function reduces the amount of code repetition for the analyses.

```
get_top10_words <- function(lexicon, category, words) {
# This function returns a dataframe of the 10 most common words in a dataframe of words,
# given a particular lexicon (string) and sentiment category (string)

# First perform the inner join
# The AFINN and GI lexicons are already separated into positive and negative sentiments
  if (lexicon == "afinn" | lexicon == "gi") {
    lexicon_sentiment <- paste(lexicon, category, sep = "_")
    top_words <- words %>%
      inner_join(eval(parse(text = lexicon_sentiment)), by = "word")
  } else {
# For Bing and NRC lexicons, filter the desired sentiment before inner join
  top_words <- words %>%
    inner_join(get_sentiments(lexicon) %>% filter(sentiment == category), by = "word")
  }

# Then get the 10 most common words and rename the lexicon column
  top_words <- top_words %>%
    count(word, sort = TRUE) %>%
    slice_head(n = 10) %>%
```

```r
    # Helpful vignette on embracing arguments and name injection
    # https://cran.r-project.org/web/packages/dplyr/vignettes/programming.html
    rename({{lexicon}} := word)

  return(top_words)
}
```

The dataframe below shows that the 10 most common positive words in the Yelp reviews vary depending on the sentiment lexicon. However, there are some similarities among the top 3 words for each lexicon. "Nice" was the most common positive word with AFINN, NRC, and GI lexicons. "Love" was the second most common positive word for all lexicons. Similarly, "friendly" was the third most common positive word with the AFINN, NRC, and Bing lexicons.

```r
top10_positive_words <- bind_cols(get_top10_words("afinn", "positive", review_words),
                                  get_top10_words("bing", "positive", review_words),
                                  get_top10_words("nrc", "positive", review_words),
                                  get_top10_words("gi", "positive", review_words))

# rename word count columns
# create sequence of indexes of columns to be renamed (ie, 2, 4, 6, 8)
col_indexes <- seq(2, ncol(top10_positive_words), 2)
# rename each column in the sequence as 'n'
colnames(top10_positive_words)[col_indexes] <- rep(c("n"), 4)

top10_positive_words
```

```
##        afinn    n      bing    n       nrc     n      gi    n
## 1       nice 9276      nice 9276      food 27464    nice 9276
## 2       love 7964      love 7964      love  7964    love 7964
## 3   friendly 7813  friendly 7813  friendly  7813   fresh 5204
## 4    amazing 6268 delicious 7583 delicious  7583   worth 3585
## 5     pretty 5571   amazing 6268    pretty  5571   clean 3387
## 6      fresh 5204    pretty 5571 recommend  4936    home 3374
## 7  recommend 4936     fresh 5204       eat  4632   super 3209
## 8  excellent 3874 recommend 4936    dinner  3876 perfect 3014
## 9    awesome 3824 excellent 3874 excellent  3874   sweet 2961
## 10     worth 3585   awesome 3824      beer  3805    free 2732
```

The dataframe below shows that the 10 most common negative words in the Yelp reviews also vary depending on the sentiment lexicon. In general, the variability between lexicons is greater than that for the 10 most common positive words, which suggests that the lexicons are more similar to each other with respect to positive words than negative words.

"Bad" was a highly ranked negative word with all lexicons—#1 with AFINN and Bing, #2 with NRC, and #4 with GI. "Disappointed" was the second most common negative word with the AFINN lexicon and the third most common with the NRC and Bing lexicons, but it did not rank in the top 10 with the GI lexicon. "Wait" was the most common negative word with the NRC lexicon and the second most common with the GI lexicon.

In general, the negative words from the NRC and Bing lexicons make the most sense. On the other hand, it is unclear why some of the words from the GI lexicon are considered negative, such as "home" and "spot". Similarly, a few of the AFINN words, such as "pay" and "cut" are not clearly negative. This may suggest that the NRC and Bing lexicons are more appropriate for sentiment analysis of the Yelp reviews.

```r
top10_negative_words <- bind_cols(get_top10_words("afinn", "negative", review_words),
                                  get_top10_words("bing", "negative", review_words),
                                  get_top10_words("nrc", "negative", review_words),
                                  get_top10_words("gi", "negative", review_words))

# rename word count columns
# create sequence of indexes of columns to be renamed (ie, 2, 4, 6, 8)
col_indexes <- seq(2, ncol(top10_negative_words), 2)
# rename each column in the sequence as 'n'
colnames(top10_negative_words)[col_indexes] <- rep(c("n"), 4)

top10_negative_words
```

```
##           afinn    n         bing    n          nrc    n    gi    n
## 1           bad 3603          bad 3603         wait 5459   bar 5880
## 2  disappointed 2109        fried 3044          bad 3603  wait 5459
## 3          hard 2031 disappointed 2109 disappointed 2109   bit 3975
## 4          stop 1928         hard 2031         cold 1754   bad 3603
## 5         wrong 1538         cold 1754        wrong 1538   hot 3469
## 6           pay 1513        wrong 1538        leave 1341  home 3374
## 7           cut 1373         slow 1205         yelp 1282  spot 2791
## 8         leave 1341    expensive 1154         late 1256   fun 2253
## 9       stopped 1301        cheap 1119        cheap 1119 front 2047
## 10     terrible 1090     terrible 1090     terrible 1090  hard 2031
```

3.4.3. Comparison word clouds

AFINN

```r
review_words %>%
  inner_join(afinn_all, by = "word") %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("darkred", "blue"), max.words = 50)
```

Bing

```r
review_words %>%
  inner_join(get_sentiments("bing"), by = "word") %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("darkred", "blue"), max.words = 50)
```

NRC

```
review_words %>%
  inner_join(get_sentiments("nrc") %>%
             filter(sentiment %in% c("positive", "negative")), by = "word") %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("darkred", "blue"), max.words = 50)
```

GI

```
review_words %>%
  inner_join(gi_all, by = "word") %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("darkred", "blue"), max.words = 50)
```

3.4.4. Comparing sentiment lexicons I further examined the differences between the four lexicons by comparing the net sentiments over time. In *Text Mining with R* chapter 2, the AFINN analysis was performed using the sum of the sentiment scores. However, this may inflate the total sentiment scores, so I used the binary reclassification of the AFINN lexicon that I defined in the previous section instead. This way, all four lexicons are compared with binary positive/negative sentiments.

```
all_lexicons <- bind_rows(
  # AFINN
  review_words %>%
    inner_join(afinn_all, by = "word") %>%
    mutate(method = "AFINN"),
  # GI
  review_words %>%
    inner_join(gi_all, by = "word") %>%
    mutate(method = "GI"),
  # Bing
```

```r
  review_words %>%
    inner_join(get_sentiments("bing"), by = "word") %>%
    mutate(method = "Bing"),
  # NRC
  review_words %>%
    inner_join(get_sentiments("nrc") %>%
                 filter(sentiment %in% c("positive", "negative")), by = "word") %>%
    mutate(method = "NRC")) %>%
  # I defined the index as the number of days since January 1, 2005
  # due to ggplot issues with overcrowded date labels
  mutate(
    index = as.integer(date - as.Date("2005-01-01"))
  ) %>%
  count(method, index, sentiment) %>%
  pivot_wider(names_from = sentiment,
              values_from = n,
              values_fill = 0) %>%
  mutate(net_sentiment = positive - negative)
```

Because the range of the net sentiment values is skewed toward large positive values, I applied a log-modulus transformation, which helps spread the magnitude of the values while preserving their sign, to improve the plots below.[5]

```r
sprintf("Net sentiment values range from %.3f to %.3f",
        min(all_lexicons$net_sentiment), max(all_lexicons$net_sentiment))
```

```
## [1] "Net sentiment values range from -26.000 to 600.000"
```

After the transformation, the magnitude of the positive net sentiment values are more similar to the negative values.

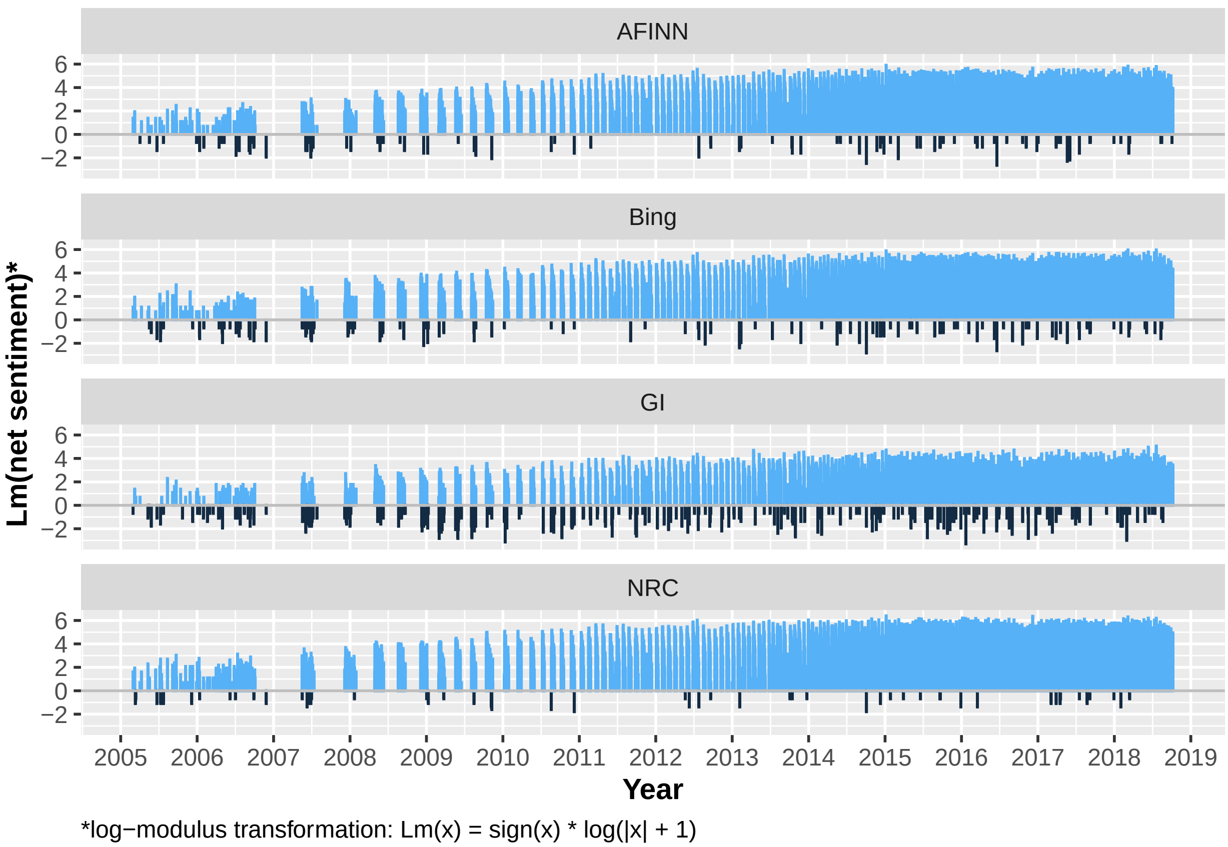
```r
all_lexicons <- all_lexicons %>%
  mutate(
    # log1p(x) computes log(1+x)
    # https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/log
    net_sentiment = sign(net_sentiment) * log1p(abs(net_sentiment))
  )

sprintf("Net sentiment values range from %.3f to %.3f",
        min(all_lexicons$net_sentiment), max(all_lexicons$net_sentiment))
```

```
## [1] "Net sentiment values range from -3.296 to 6.399"
```

Overall, the net sentiment values over time are similar for all lexicons, particularly for the reviews with positive values. The main differences appear to be in the reviews with negative net sentiments, which is similar to the findings from the analysis of the most common positive and negative words in the reviews with the four lexicons (Section 3.4.2. Comparison of most common positive and negative words in Yelp reviews using different lexicons). Focusing on these reviews, it can be seen that the GI lexicon results in the most reviews with negative net sentiments. In contrast, the NRC lexicon has the fewest negative net sentiments. The AFINN and Bing lexicons are intermediate.

[5] $L(x) = sign(x) + log(|x| + 1)$ https://blogs.sas.com/content/iml/2014/07/14/log-transformation-of-pos-neg.html

```r
# Color positive and negative values differently to enhance contrast
ggplot(all_lexicons, aes(index, net_sentiment, color = sign(net_sentiment))) +
    geom_col(show.legend = FALSE) +
    geom_hline(yintercept = 0, color = "gray") +
    xlab("Year") + ylab("Lm(net sentiment)*") +
    labs(caption = "*log-modulus transformation: Lm(x) = sign(x) * log(|x| + 1)") +
    theme(axis.title = element_text(face = "bold"),
          plot.caption = element_text(hjust = 0)) +
    scale_x_continuous(breaks = seq(0, 5110, by = 365),
                       labels = c("2005", "2006", "2007", "2008", "2009", "2010", "2011", "2012",
                                  "2013", "2014", "2015", "2016", "2017", "2018", "2019")) +
    facet_wrap(~ method, ncol = 1)
```



Combining these results with those from section 3.4.2, which suggested that the NRC and Bing lexicons identified the most intuitive negative words, I think the Bing lexicon is the best lexicon to perform sentiment analysis of the Yelp reviews.

4. Exploratory analyses

In addition to customer reviews, the Yelp Open dataset includes business data, such as the name, address, and latitude/longitude coordinates. I was very interested in merging these datasets to perform a combination of geospatial and sentiment analysis as a demonstration of how these data could be used to inform business intelligence and decision-making.

4.1. Command-line transformations of Yelp business dataset

Similar to the review JSON data, I corrected the JSON format of the business data on the command line.

```
# add comma to end of each line
sed "s/$/,/g" yelp_business.json > yelp_business2.json
# remove the last one
# 2 characters are truncated because there is also a \n at the end of the line
truncate -s -2 yelp_business2.json
# add a top-level root element called "reviews"
# prepend
sed -i.old '1s;^;{ "businessess": [\n;' yelp_business2.json
# postpend
echo "] }" >> yelp_business2.json
```

4.2. Convert JSON data to dataframe

I saved the corrected data file to my GitHub repository and then read it into R.

```
yelp_businesses <- fromJSON("https://github.com/alexandersimon1/Data607/raw/main/Assignment10/yelp_busin
```

The data structure is shown below.

```
str(yelp_businesses)
```

```
## List of 1
##  $ businessess:'data.frame': 150346 obs. of  58 variables:
##   ..$ business_id                       : chr [1:150346] "Pns2l4eNsfO8kk83dixA6A" "mpf3x-BjTdTEA3
##   ..$ name                              : chr [1:150346] "Abby Rappoport, LAC, CMQ" "The UPS Stor
##   ..$ address                           : chr [1:150346] "1616 Chapala St, Ste 2" "87 Grasso Plaz
##   ..$ city                              : chr [1:150346] "Santa Barbara" "Affton" "Tucson" "Phila
##   ..$ state                             : chr [1:150346] "CA" "MO" "AZ" "PA" ...
##   ..$ postal_code                       : chr [1:150346] "93101" "63123" "85711" "19107" ...
##   ..$ latitude                          : num [1:150346] 34.4 38.6 32.2 40 40.3 ...
##   ..$ longitude                         : num [1:150346] -119.7 -90.3 -110.9 -75.2 -75.5 ...
##   ..$ stars                             : num [1:150346] 5 3 3.5 4 4.5 2 2.5 3.5 3 1.5 ...
##   ..$ review_count                      : int [1:150346] 7 15 22 80 13 6 13 5 19 10 ...
##   ..$ is_open                           : int [1:150346] 0 1 0 1 1 1 1 1 0 1 ...
##   ..$ categories                        : chr [1:150346] "Doctors, Traditional Chinese Medicine, 
##   ..$ attributes.ByAppointmentOnly      : chr [1:150346] "True" NA "False" "False" ...
##   ..$ attributes.BusinessAcceptsCreditCards: chr [1:150346] NA "True" "True" "False" ...
##   ..$ attributes.BikeParking            : chr [1:150346] NA NA "True" "True" ...
##   ..$ attributes.RestaurantsPriceRange2 : chr [1:150346] NA NA "2" "1" ...
##   ..$ attributes.CoatCheck              : chr [1:150346] NA NA "False" NA ...
##   ..$ attributes.RestaurantsTakeOut     : chr [1:150346] NA NA "False" "True" ...
##   ..$ attributes.RestaurantsDelivery    : chr [1:150346] NA NA "False" "False" ...
##   ..$ attributes.Caters                 : chr [1:150346] NA NA "False" "True" ...
##   ..$ attributes.WiFi                   : chr [1:150346] NA NA "u'no'" "u'free'" ...
##   ..$ attributes.BusinessParking        : chr [1:150346] NA NA "{'garage': False, 'street': False
##   ..$ attributes.WheelchairAccessible   : chr [1:150346] NA NA "True" NA ...
##   ..$ attributes.HappyHour              : chr [1:150346] NA NA "False" NA ...
##   ..$ attributes.OutdoorSeating         : chr [1:150346] NA NA "False" "False" ...
```

```
##   ..$ attributes.HasTV                     : chr [1:150346] NA NA "False" NA ...
##   ..$ attributes.RestaurantsReservations   : chr [1:150346] NA NA "False" NA ...
##   ..$ attributes.DogsAllowed               : chr [1:150346] NA NA "False" NA ...
##   ..$ attributes.Alcohol                   : chr [1:150346] NA NA NA "u'none'" ...
##   ..$ attributes.GoodForKids               : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.RestaurantsAttire         : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.Ambience                  : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.RestaurantsTableService   : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.RestaurantsGoodForGroups  : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.DriveThru                 : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.NoiseLevel                : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.GoodForMeal               : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.BusinessAcceptsBitcoin    : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.Smoking                   : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.Music                     : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.GoodForDancing            : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.AcceptsInsurance          : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.BestNights                : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.BYOB                      : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.Corkage                   : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.BYOBCorkage               : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.HairSpecializesIn         : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.Open24Hours               : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.RestaurantsCounterService : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.AgesAllowed               : chr [1:150346] NA NA NA NA ...
##   ..$ attributes.DietaryRestrictions       : chr [1:150346] NA NA NA NA ...
##   ..$ hours.Monday                         : chr [1:150346] NA "0:0-0:0" "8:0-22:0" "7:0-20:0" ...
##   ..$ hours.Tuesday                        : chr [1:150346] NA "8:0-18:30" "8:0-22:0" "7:0-20:0" ...
##   ..$ hours.Wednesday                      : chr [1:150346] NA "8:0-18:30" "8:0-22:0" "7:0-20:0" ...
##   ..$ hours.Thursday                       : chr [1:150346] NA "8:0-18:30" "8:0-22:0" "7:0-20:0" ...
##   ..$ hours.Friday                         : chr [1:150346] NA "8:0-18:30" "8:0-23:0" "7:0-21:0" ...
##   ..$ hours.Saturday                       : chr [1:150346] NA "8:0-14:0" "8:0-23:0" "7:0-21:0" ...
##   ..$ hours.Sunday                         : chr [1:150346] NA NA "8:0-22:0" "7:0-21:0" ...
```

I binded the columns into a dataframe and selected the relevant columns.

```
yelp_businesses_df <- bind_cols(yelp_businesses[[1]])
yelp_businesses_df <- yelp_businesses_df %>%
  select(business_id, name, address, city, state, postal_code, latitude, longitude,
         business_stars = stars, review_count)
```

4.3. Characteristics of the Yelp business data

4.3.1. Number of businesses There are 114,117 unique business names in the dataset.

```
yelp_businesses_df %>%
  select(name) %>%
  n_distinct()
```

```
## [1] 114117
```

4.3.2. Businesses with multiple locations Starbucks had the most store locations, so I focused the geospatial analyses on this business.

```
yelp_businesses_df %>%
  count(name, sort = TRUE) %>%
  filter(n >= 200)
```

```
##                     name   n
## 1              Starbucks 724
## 2             McDonald's 703
## 3                Dunkin' 510
## 4                 Subway 459
## 5              Taco Bell 365
## 6           CVS Pharmacy 345
## 7              Walgreens 341
## 8            Burger King 338
## 9                Wendy's 331
## 10                  Wawa 307
## 11         Domino's Pizza 295
## 12          The UPS Store 281
## 13             Pizza Hut 272
## 14 Enterprise Rent-A-Car 232
```

4.4. Combine Yelp reviews and business data

```
yelp_business_reviews <- inner_join(yelp_businesses_df, yelp_reviews_df, by = "business_id")
```

4.5. Map Starbucks overall business ratings ("stars")

I mapped the business ratings ("stars") of all Starbucks locations. These ratings were included in the Yelp dataset.

4.5.1. Align nomenclature in US map dataset and Yelp business dataset First, load the US map data.

```
states <- map_data("state")
```

Since the full names of states are used in the map data whereas the Yelp business dataset uses state abbreviations, I created a named vector to map the state names to their abbreviations.

```
state_names <- c("alabama", "alaska", "arizona", "arkansas", "california", "colorado",
                 "connecticut", "delaware", "district of columbia", "florida", "georgia",
                 "hawaii", "idaho", "illinois", "indiana", "iowa", "kansas", "kentucky",
                 "louisiana", "maine", "maryland", "massachusetts", "michigan", "minnesota",
                 "mississippi", "missouri", "montana", "nebraska", "nevada", "new hampshire",
                 "new jersey", "new mexico", "new york", "north carolina", "north dakota",
                 "ohio", "oklahoma", "oregon", "pennsylvania", "rhode island", "south carolina",
                 "south dakota", "tennessee", "texas", "utah", "vermont", "virginia", "washington",
                 "west virginia", "wisconsin", "wyoming")
```

```r
state_abbreviations <- c("AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "DC", "FL", "GA", "HI",
                         "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD", "MA", "MI", "MN",
                         "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY", "NC", "ND", "OH",
                         "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA",
                         "WV", "WI", "WY")
get_state_abbreviation <- setNames(state_abbreviations, state_names)
```

I used this vector to rename the states in the map data.

```r
states <- states %>%
  rename(state = region) %>%
  mutate(
    state = unname(get_state_abbreviation[state])
  )
```

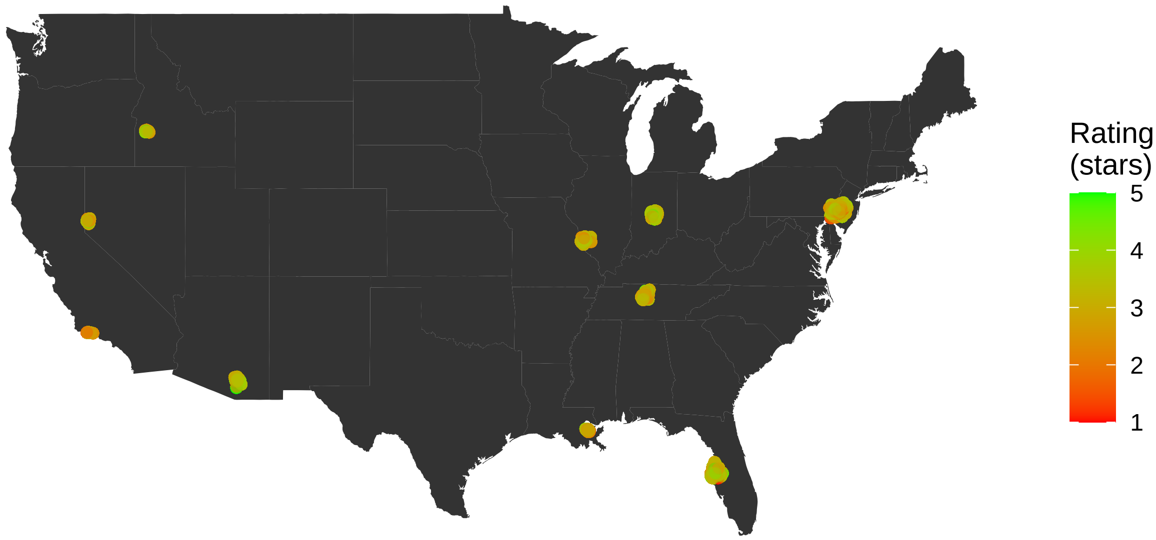
4.5.2. Combine the map and business datasets Then I combined the map data with the Starbucks business data.

```r
yelp_starbucks <- yelp_businesses_df %>%
  filter(name == "Starbucks")

starbucks_us_mapdata <- inner_join(states, yelp_starbucks, by = "state")
```

4.5.3. Create map Finally, I plotted the Starbucks store locations and colored the data points by the business ratings (number of stars). Although the data points overlap, the transparency gives a sense of the average overall rating, which appears to be 3 to 4 in most cities. Starbucks stores in Los Angeles, CA appear to have the lowest ratings nationwide.

Note that this is an interactive plot and can be panned and zoomed as desired. These operations are not instantaneous and may take a few seconds, so please be patient.

```r
# settings to remove the map grid, axes labels, and tick marks
plain_background <- theme(
  axis.text = element_blank(),
  axis.line = element_blank(),
  axis.ticks = element_blank(),
  panel.border = element_blank(),
  panel.grid = element_blank(),
  axis.title = element_blank(),
  panel.background = element_rect(fill = "white")
)

starbucks_us_map <- ggplot(states, aes(long, lat, group = group)) +
  geom_polygon() + coord_fixed(1.3) +
  geom_point(starbucks_us_mapdata,
             mapping = aes(longitude, latitude, group = group, color = business_stars), alpha = 0.4) +
  scale_color_gradient(low = "red", high = "green", name = "Rating\n(stars)") +
  plain_background

# ggplotly(starbucks_us_map)
starbucks_us_map
```



The map above shows that the greater Phliadelphia area (including the adjacent Camden, NJ area) has many Starbucks locations. Looking at this region more closely (below) reveals that there are more red data points in New Jersey, indicating that the Starbucks stores there have lower star ratings than those in Philadephia.

```
starbucks_philly_map <- ggplot(filter(states, state %in% c("NJ", "PA")),
                               aes(long, lat, group = group)) +
  geom_polygon() + coord_fixed(1.3) +
  geom_point(filter(starbucks_us_mapdata, state %in% c("NJ", "PA")),
             mapping = aes(longitude, latitude, group = group, color = business_stars),
             alpha = 0.5) +
  scale_color_gradient(low = "red", high = "green", name = "Rating\n(stars)") +
  plain_background

# ggplotly(starbucks_philly_map
starbucks_philly_map
```

4.6. Map Yelp review sentiments

Next, I wanted to map sentiments from the Yelp reviews and see how they compared with the star ratings. To do this, I calculated the average net sentiment of all reviews at each Starbucks location. Based on my findings from the analyses of the different sentiment lexicons in Section 3.4. Sentiment analysis, I selected the Bing lexicon for this analysis.

4.6.1. Filter the reviews I filtered the reviews from Starbucks stores in the Philadelphia and Camden, NJ area. Since the Yelp data are limited to those areas, I just filtered by the two states.

```
starbucks_philly_reviews <- yelp_business_reviews %>%
  filter(name == "Starbucks" & state %in% c("NJ", "PA"))
```

4.6.2. Calculate net sentiment for each review Then I calculated the net sentiment for each review in this subset.

```
calc_net_sentiment_bing <- function(text) {
  # To tokenize a string, don't need unnest_tokens, just split it up
  sentiment_words <- as_tibble(str_split_1(text, " ")) %>%
    # rename first column for anti_join
    rename(word = names(.)[1]) %>%
    anti_join(stop_words, by = "word") %>%
    inner_join(get_sentiments("bing"), by = "word")
```

```
  positive = sum(sentiment_words$sentiment == "positive")
  negative = sum(sentiment_words$sentiment == "negative")
  net_sentiment = positive - negative
  return(net_sentiment)
}

starbucks_philly_reviews <- starbucks_philly_reviews %>%
  rowwise() %>%
  mutate(
    net_sentiment = calc_net_sentiment_bing(text)
  ) %>%
  ungroup()
```

4.6.3. Calculate average net sentiment for each business location After this, I calculated the average net sentiment for each Starbucks location.

```
starbucks_philly_avg_sentiment_store <- starbucks_philly_reviews %>%
  group_by(address) %>%
  mutate(
    n_reviews = n(),
    mean_sentiment = mean(net_sentiment)
  ) %>%
  distinct(address, .keep_all = TRUE) %>%
  select(address, city, state, latitude, longitude, n_reviews, mean_sentiment) %>%
  arrange(desc(mean_sentiment))

starbucks_philly_avg_sentiment_store
```

```
## # A tibble: 13 x 7
## # Groups:   address [13]
##    address                city  state latitude longitude n_reviews mean_sentiment
##    <chr>                  <chr> <chr>    <dbl>     <dbl>     <int>          <dbl>
##  1 1839 Chestnut St       Phil~ PA        40.0     -75.2         3          2
##  2 707 Street Rd          Uppe~ PA        40.2     -75.0         4          2
##  3 214-216 Kings Hwy      Hadd~ NJ        39.9     -75.0         2          0.5
##  4 1528 Walnut St         Phil~ PA        39.9     -75.2         3          0.333
##  5 57-63 North Third St   Phil~ PA        40.0     -75.1         3          0.333
##  6 304 Greentree Rd       Sewe~ NJ        39.8     -75.1         2          0
##  7 218 East Lancaster A~ Wayne PA        40.0     -75.4         1          0
##  8 1745 South Easton Rd  Doyl~ PA        40.3     -75.1         1          0
##  9 1125 S Black Horse P~ Glou~ NJ        39.8     -75.1         2         -0.5
## 10 5 Hartford Rd          Moun~ NJ        40.0     -74.9         2         -0.5
## 11 282 Dunns Mill Rd      Bord~ NJ        40.1     -74.7         2         -0.5
## 12 480 Evesham Rd         Cher~ NJ        39.9     -75.0         1         -1
## 13 498 North Main St      Doyl~ PA        40.3     -75.1         3         -1.33
```

4.6.4. Combine the map and sentiment datasets Then I combined the map data with the sentiment data.

```r
starbucks_philly_mapdata <- inner_join(states, starbucks_philly_avg_sentiment_store, by = "state")
```

4.6.5. Create map Overall, the sentiment map agrees with the business ("star") rating—customer reviews of Starbucks stores in New Jersey have lower average net sentiment values than those in Philadelphia.

```r
starbucks_philly_map <- ggplot(filter(states, state %in% c("NJ", "PA")), aes(long, lat, group = group))
  geom_polygon() + coord_fixed(1.3) +
  geom_point(starbucks_philly_mapdata,
             mapping = aes(longitude, latitude, group = group, color = mean_sentiment), alpha = 0.5) +
  scale_color_gradient(low = "red", high = "green", name = "Mean\nsentiment") +
  plain_background

# ggplotly(starbucks_philly_map)
starbucks_philly_map
```

Based on these results (if they were more recent data), Starbucks management may want to examine the performance metrics of the Camden, NJ locations to better understand the cause and potential solutions for the lower mean sentiment scores of customer reviews.

5. Conclusions

I successfully implemented the sentiment analysis of Jane Austen's novels described in *Text Mining with R*, chapter 2 using three different sentiment lexicons (AFINN, Bing, and NRC). I applied those techniques to analyze the sentiments of Yelp customer reviews and compared the results from the three sentiment lexicons along with a fourth, the GI lexicon. The analyses suggested the Bing lexicon gave the most balanced and intuitive results for analyzing the Yelp reviews.

As an exploratory analysis, I combined the Yelp reviews with business names and locations and demonstrated how geospatial and sentiment analyses could be used together to provide insights about customer satisfaction and/or store performance. For example, among Starbucks stores in the Philadelphia + Camden, NJ area, these analyses showed Starbucks in Camden, NJ tended to have lower sentiment ratings than stores in Philadelphia. This finding was in general agreement with the geospatial analysis of the business (star) ratings.

Finally, even with 50,000 reviews, this analysis is only a fraction of the 7 million reviews in the Yelp dataset. Potential future improvements include loading and preprocessing the data in a SQL database on a server. In addition, maps of metropolitan areas with more geographic features could be used.