Lab 7

Jefferson Ratliff

Math 241, Week 9

```r
# Put all necessary libraries here
library(tidyverse)
library(tidytext)

# Ensure the textdata package is installed
if (!requireNamespace("textdata", quietly = TRUE)) {
  install.packages("textdata")
}
# Load the textdata package
library(textdata)

# Before knitting your document one last time, you will have to download the AFINN lexicon explicitly
lexicon_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
```

```r
lexicon_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
```

Due: Friday, March 29th at 5:30pm

Goals of this lab

1. Practice matching patterns with regular expressions.
2. Practice manipulating strings with `stringr`.
3. Practice tokenizing text with `tidytext`.
4. Practice looking at word frequencies.
5. Practice conducting sentiment analysis.

Problem 1: What's in a Name? (You'd Be Surprised!)

1. Load the `babynames` dataset, which contains yearly information on the frequency of baby names by sex and is provided by the US Social Security Administration. It includes all names with at least 5 uses per year per sex. In this problem, we are going to practice pattern matching!

```r
library(babynames)
data("babynames")
#?babynames
```

a. For 2000, find the ten most popular female baby names that start with the letter Z.

```r
names_2000_Z <- filter(babynames, year == 2000 & sex == "F" & substr(name, 1, 1) == "Z")

top_names_2000_Z <- top_n(names_2000_Z, 10, n)

print(top_names_2000_Z)
```

b. For 2000, find the ten most popular female baby names that contain the letter z.

```r
names_2000_with_z <- filter(babynames, year == 2000 & sex == "F" & grepl("z", name))

top_names_2000_with_z <- head(arrange(names_2000_with_z, desc(n)), 10)

print(top_names_2000_with_z)
```

```
## # A tibble: 10 x 5
##     year sex   name          n     prop
##    <dbl> <chr> <chr>     <int>    <dbl>
##  1  2000 F     Elizabeth 15094 0.00757
##  2  2000 F     Mackenzie  6348 0.00318
##  3  2000 F     Mckenzie   2526 0.00127
##  4  2000 F     Makenzie   1613 0.000809
##  5  2000 F     Jazmin     1391 0.000697
##  6  2000 F     Jazmine    1353 0.000678
## 7  2000 F     Lizbeth     817 0.000410
## 8  2000 F     Eliza       759 0.000380
## 9  2000 F     Litzy       722 0.000362
## 10 2000 F     Esperanza   499 0.000250
```

c. For 2000, find the ten most popular female baby names that end in the letter z.

```
names_2000_end_with_z <- filter(babynames, year == 2000 & sex == "F" & substr(name, nchar(name), nchar(

top_names_2000_end_with_z <- head(arrange(names_2000_end_with_z, desc(n)), 10)

print(top_names_2000_end_with_z)
```

```
## # A tibble: 10 x 5
##     year sex   name          n       prop
##    <dbl> <chr> <chr>     <int>      <dbl>
##  1  2000 F     Luz         489 0.000245
##  2  2000 F     Beatriz     357 0.000179
##  3  2000 F     Mercedez    141 0.0000707
##  4  2000 F     Maricruz     96 0.0000481
##  5  2000 F     Liz          72 0.0000361
##  6  2000 F     Inez         69 0.0000346
##  7  2000 F     Odaliz       24 0.0000120
##  8  2000 F     Marycruz     23 0.0000115
##  9  2000 F     Cruz         19 0.00000952
## 10  2000 F     Deniz        16 0.00000802
```

d. Between your three tables in 1.a - 1.c, do any of the names show up on more than one list? If so, which ones? (Yes, I know you could do this visually but use some joins!)

```
names_2000_start_with_z <- babynames %>%
  filter(year == 2000 & sex == "F" & substr(name, 1, 1) == "Z") %>%
  select(name)

names_2000_with_z <- babynames %>%
  filter(year == 2000 & sex == "F" & grepl("z", name)) %>%
  select(name)

names_2000_end_with_z <- babynames %>%
  filter(year == 2000 & sex == "F" & substr(name, nchar(name), nchar(name)) == "z") %>%
  select(name)

common_names_1_2 <- inner_join(names_2000_start_with_z, names_2000_with_z, by = "name")
common_names_1_3 <- inner_join(names_2000_start_with_z, names_2000_end_with_z, by = "name")
common_names_2_3 <- inner_join(names_2000_with_z, names_2000_end_with_z, by = "name")

all_common_names <- bind_rows(common_names_1_2, common_names_1_3, common_names_2_3) %>%
                    distinct()

# Print the common names
print(all_common_names)
```

```
## # A tibble: 41 x 1
##    name
##    <chr>
##  1 Zuzanna
##  2 Zeltzin
##  3 Zsazsa
##  4 Luz
##  5 Beatriz
##  6 Mercedez
##  7 Maricruz
##  8 Liz
##  9 Inez
## 10 Odaliz
## # i 31 more rows
```

e. Verify that none of the baby names contain a numeric (0-9) in them.

```
female_names_2000 <- babynames %>%
  filter(year == 2000 & sex == "F")

names_with_numeric <- female_names_2000 %>%
  filter(grepl("[0-9]", name))


print(names_with_numeric)
```

```
## # A tibble: 0 x 5
## # i 5 variables: year <dbl>, sex <chr>, name <chr>, n <int>, prop <dbl>
```

f. While none of the names contain 0-9, that doesn't mean they don't contain "one", "two", . . . , or "nine". Create a table that provides the number of times a baby's name contained the word "zero", the word "one", . . . the word "nine".

```
female_names_2000 <- babynames %>%
  filter(year == 2000 & sex == "F")

number_words <- c("zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine")

number_counts <- data.frame(Number = number_words, Count = numeric(length(number_words)))

for (i in 1:length(number_words)) {
  word <- number_words[i]
  counts <- female_names_2000 %>%
    filter(grepl(paste0("\\b", word, "\\b"), name)) %>%
    summarise(count = n())
  number_counts[i, "Count"] <- counts$count
}

print(number_counts)
```

```
##    Number Count
## 1    zero     0
## 2      one      0
## 3      two      0
## 4    three      0
## 5     four      0
## 6     five      0
## 7      six      0
## 8    seven      0
## 9    eight      0
## 10    nine      0
```

Notes:

- I recommend first converting all the names to lower case.
- If none of the baby's names contain the written number, there you can leave the number out of the table.
- Use `str_extract()`, not `str_extract_all()`. (We will ignore names where more than one of the words exists.)

Hint: You will have two steps that require pattern matching: 1. Subset your table to only include the rows with the desired words. 2. Add a column that contains the desired word.

g. Which written number or numbers don't show up in any of the baby names?

```
female_names_2000 <- babynames %>%
  filter(year == 2000 & sex == "F") %>%
  mutate(name = tolower(name))

number_words <- c("zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine")


number_counts <- data.frame(Number = character(), Count = integer(), stringsAsFactors = FALSE)


for (word in number_words) {

  names_with_word <- female_names_2000 %>%
    filter(str_detect(name, paste0("\\b", word, "\\b")))


names_with_word <- names_with_word %>%
    mutate(word = str_extract(name, paste0("\\b", word, "\\b"))) %>%
    distinct()  # Keep only unique names

word_count <- nrow(names_with_word)

  if (word_count > 0) {
    number_counts <- bind_rows(number_counts, data.frame(Number = word, Count = word_count))
  }
}

print(number_counts)
```

```
##   Number Count
## 1  seven     1
```

```
missing_numbers <- setdiff(number_words, number_counts$Number)
print(paste("Written number(s) not found in any baby names:", paste(missing_numbers, collapse = ", ")))
```

```
## [1] "Written number(s) not found in any baby names: zero, one, two, three, four, five, six, eight, n
```

h. Create a table that contains the names and their frequencies for the two least common written numbers.

```
female_names_2000 <- babynames %>%
  filter(year == 2000 & sex == "F") %>%
  mutate(name = tolower(name))

number_words <- c("zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine")

number_counts <- data.frame(Number = character(), Count = integer(), stringsAsFactors = FALSE)

for (word in number_words) {
  names_with_word <- female_names_2000 %>%
    filter(str_detect(name, paste0("\\b", word, "\\b")))

  names_with_word <- names_with_word %>%
    mutate(word = str_extract(name, paste0("\\b", word, "\\b"))) %>%
    distinct()

  word_count <- nrow(names_with_word)

  if (word_count > 0) {
    number_counts <- bind_rows(number_counts, data.frame(Number = word, Count = word_count))
  }
}

least_common_numbers <- number_counts %>%
  arrange(Count) %>%
  head(2)


names_least_common_numbers <- female_names_2000 %>%
  filter(str_detect(name, paste0("\\b", least_common_numbers$Number[1], "\\b|\\b", least_common_numbers$
  group_by(name) %>%
  summarise(total_count = sum(n))


print(names_least_common_numbers)
```

```
## # A tibble: 1 x 2
##   name  total_count
##   <chr>       <int>
## 1 seven          24
```

i. List out the names that contain no vowels (consider “y” to be a vowel).

```r
female_names_2000 <- babynames %>%
  filter(year == 2000 & sex == "F")

contains_no_vowels <- function(name) {
  !grepl("[aeiouyAEIOUY]", name)
}

names_no_vowels <- female_names_2000 %>%
  filter(sapply(name, contains_no_vowels))

print(names_no_vowels$name)
```

```
## [1] "Bg" "Kc" "Mc"
```

Problem 2: Tidying the "Call of the Wild"

Did you read "Call of the Wild" by Jack London? If not, read the first paragraph of its wiki page for a quick summary and then let's do some text analysis on this classic! The following code will pull the book into R using the `gutenbergr` package.

```r
library(gutenbergr)
wild <- gutenberg_download(215)
```

a. Create a tidy text dataset where you tokenize by words.

```r
wild_tokens <- wild %>%
  unnest_tokens(word, text)

head(wild_tokens)
```

```
## # A tibble: 6 x 2
##   gutenberg_id word
##          <int> <chr>
## 1          215 cover
## 2          215 the
## 3          215 call
## 4          215 of
## 5          215 the
## 6          215 wild
```

b. Find the frequency of the 20 most common words. First, remove stop words.

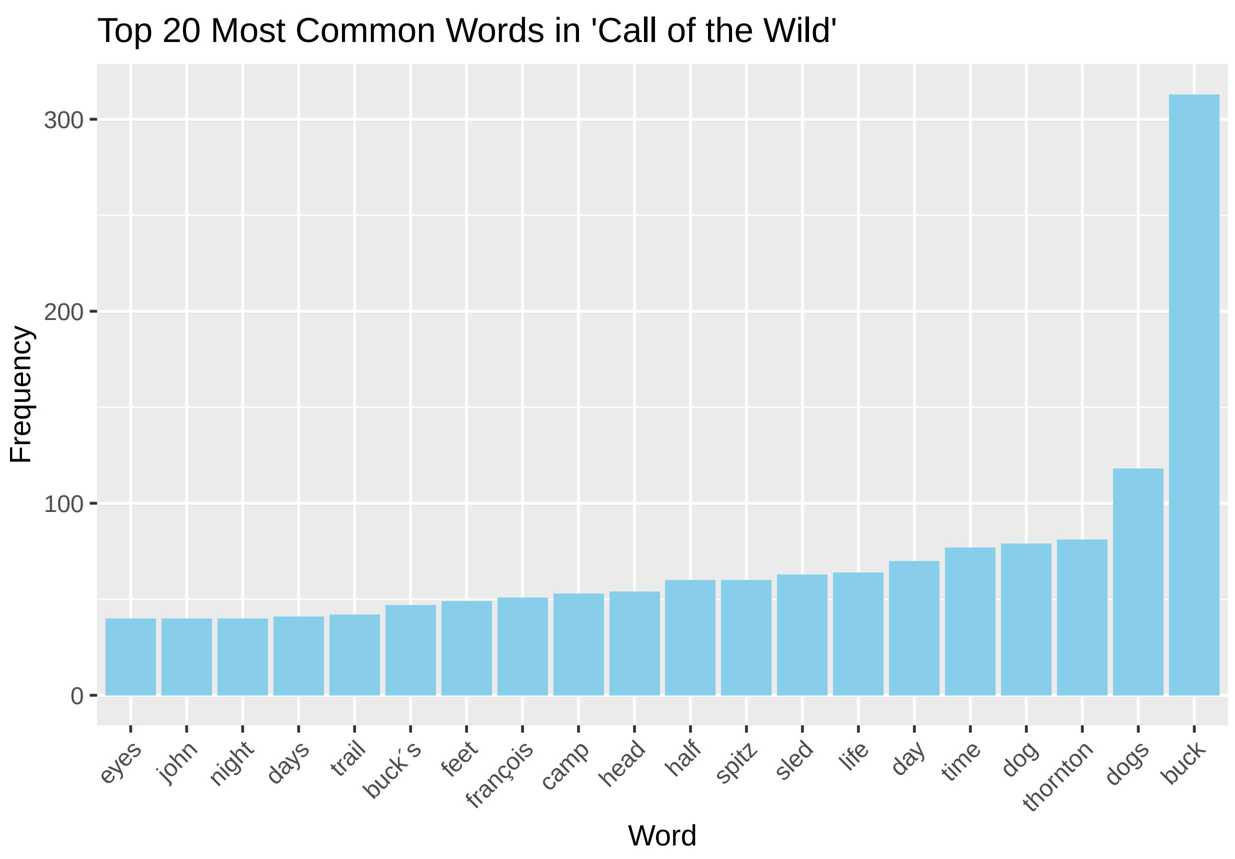
```r
wild_tokens <- wild %>%
  unnest_tokens(word, text)

wild_tokens_no_stop <- wild_tokens %>%
  anti_join(stop_words)

word_counts <- wild_tokens_no_stop %>%
  count(word, sort = TRUE)
```

```r
top_20_words <- head(word_counts, 20)

print(top_20_words)
```

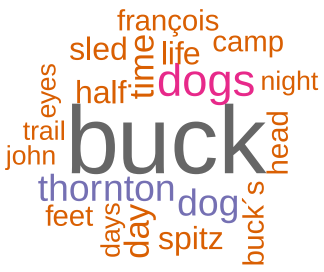
```
## # A tibble: 20 x 2
##    word         n
##    <chr>    <int>
##  1 buck       313
##  2 dogs       118
##  3 thornton    81
##  4 dog         79
##  5 time        77
##  6 day         70
##  7 life        64
##  8 sled        63
##  9 half        60
## 10 spitz       60
## 11 head        54
## 12 camp        53
## 13 françois    51
## 14 feet        49
## 15 buck's      47
## 16 trail       42
## 17 days        41
## 18 eyes        40
## 19 john        40
## 20 night       40
```

c. Create a bar graph and a word cloud of the frequencies of the 20 most common words.

```r
library(wordcloud)

bar_plot <- ggplot(top_20_words, aes(x = reorder(word, n), y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(x = "Word", y = "Frequency", title = "Top 20 Most Common Words in 'Call of the Wild'") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

print(bar_plot)
```



```
wordcloud(words = top_20_words$word, freq = top_20_words$n, scale=c(3,0.5), min.freq = 1,
          random.order=FALSE, rot.per=0.35, colors=brewer.pal(8, "Dark2"))
```



d. Explore the sentiment of the text using three of the sentiment lexicons in `tidytext`. What does your analysis say about the sentiment of the text?

Notes:

- Make sure to NOT remove stop words this time.
- `afinn` is a numeric score and should be handled differently than the categorical scores.

```r
sentiment_bing <- wild_tokens %>%
  inner_join(get_sentiments("bing"))

sentiment_afinn <- wild_tokens %>%
  inner_join(get_sentiments("afinn"))

sentiment_nrc <- wild_tokens %>%
  inner_join(get_sentiments("nrc"))


bing_score <- sum(sentiment_bing$sentiment == "positive") - sum(sentiment_bing$sentiment == "negative")
afinn_score <- sum(sentiment_afinn$value)
nrc_score <- sum(sentiment_nrc$sentiment == "positive") - sum(sentiment_nrc$sentiment == "negative")

print(paste("Bing sentiment score:", bing_score))
```

```
## [1] "Bing sentiment score: -675"
```

```
print(paste("AFINN sentiment score:", afinn_score))
```

```
## [1] "AFINN sentiment score: -591"
```

```
print(paste("NRC sentiment score:", nrc_score))
```

```
## [1] "NRC sentiment score: -266"
```

e. If you didn't do so in 2.d, compute the average sentiment score of the text using `afinn`. Which positive words had the biggest impact? Which negative words had the biggest impact?

```
sentiment_afinn %>%
  summarize(mean = mean(value))
```

```
## # A tibble: 1 x 1
##     mean
##    <dbl>
## 1 -0.322
```

```
sentiment_afinn %>%
  group_by(word) %>%
  summarize(mean = mean(value),
            n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 528 x 3
##    word      mean     n
##    <chr>    <dbl> <int>
##  1 no          -1    95
##  2 like         2    58
##  3 great        3    49
##  4 fire        -2    33
##  5 good         3    28
##  6 dead        -3    22
##  7 cried       -2    21
##  8 love         3    21
##  9 strength     2    21
## 10 broke       -1    18
## # i 518 more rows
```

f. You should have found that "no" was an important negative word in the sentiment score. To know if that really makes sense, let's turn to the raw lines of text for context. Pull out all of the lines that have the word "no" in them. Make sure to not pull out extraneous lines (e.g., a line with the word "now").

```
no_wild <- wild %>%
  filter(str_detect(text, "\\bno\\b"))
no_wild
```

```
## # A tibble: 83 x 2
##    gutenberg_id text
##           <int> <chr>
##  1          215 solitary man, no one saw them arrive at the little flag station~
##  2          215 that it was the club, but his madness knew no caution. A dozen ~
##  3          215 "He's no slouch at dog-breakin', that's wot I say," one of the ~
##  4          215 all, that he stood no chance against a man with a club. He had ~
##  5          215 in the red sweater. "And seem' it's government money, you ain't~
##  6          215 animal. The Canadian Government would be no loser, nor would its
##  7          215 while he developed no affection for them, he none the less grew
##  8          215 The other dog made no advances, nor received any; also, he did ~
##  9          215 knew no law but the law of club and fang.
## 10          215 full-grown wolf, though not half so large as she. There was no ~
## # i 73 more rows
```

g. Draw some conclusions about how "no" is used in the text.

No is used not a lot within character dialogue but more as a negative modifier to communicate the lack of a value or thing.

h. We can also look at how the sentiment of the text changes as the text progresses. Below, I have added two columns to the original dataset. Now I want you to do the following wrangling:

- Tidy the data (but don't drop stop words).
- Add the word sentiments using `bing`.
- Count the frequency of sentiments by index.
- Reshape the data to be wide with the count of the negative sentiments in one column and the positive in another, along with a column for index.
- Compute a sentiment column by subtracting the negative score from the positive.

```r
wild_time <- wild %>%
  mutate(line = row_number(), index = floor(line/45) + 1)
```

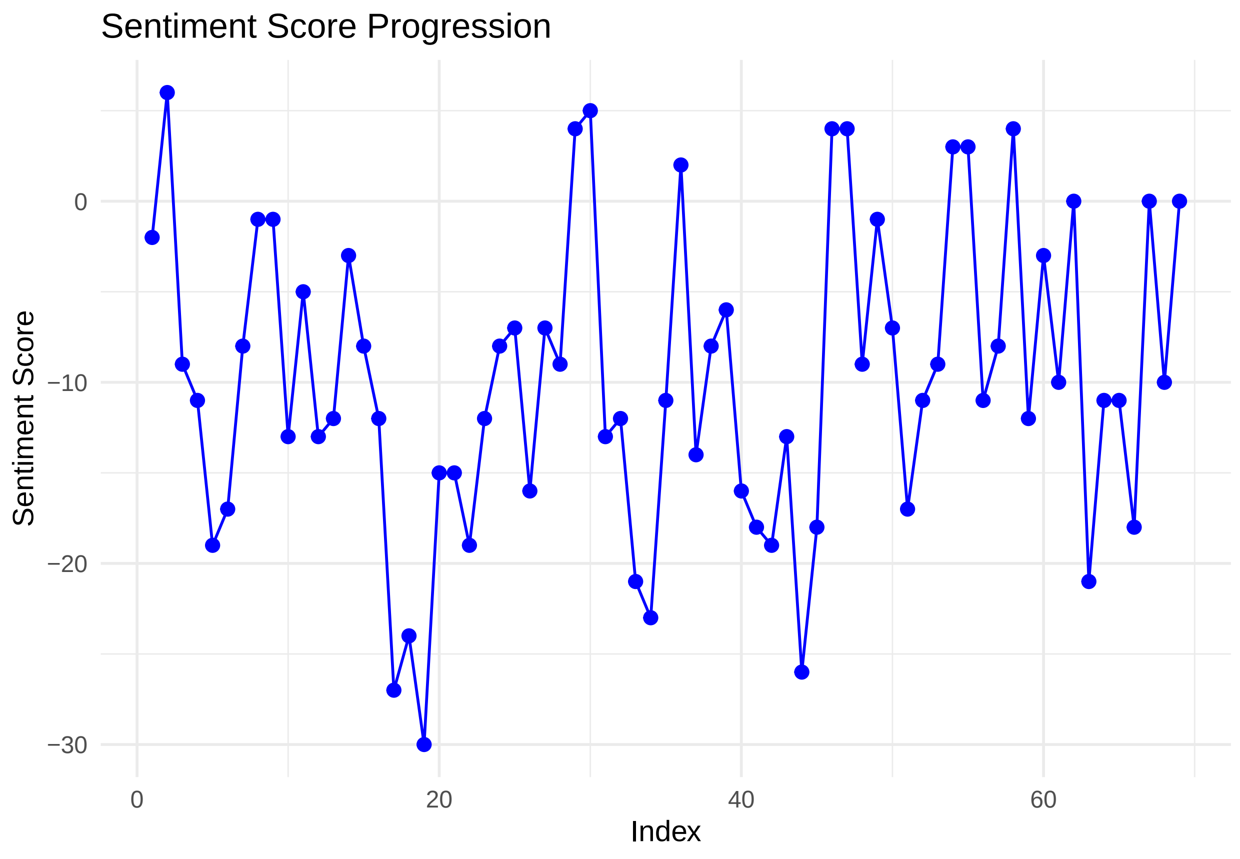
```r
#Hint: fill = 0 will insert zero instead of NA
pivot_XXX(..., values_fill = 0)
```

```r
wild_tidy <- wild_time %>%
  unnest_tokens(word, text)

wild_sentiments <- wild_tidy %>%
  inner_join(get_sentiments("bing"))

sentiment_counts <- wild_sentiments %>%
  count(index, sentiment)

sentiment_wide <- sentiment_counts %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0)

sentiment_wide <- sentiment_wide %>%
  mutate(sentiment = positive - negative)

print(sentiment_wide)
```

```
## # A tibble: 69 x 4
##    index negative positive sentiment
##    <dbl>    <int>    <int>     <int>
##  1     1        9        7        -2
##  2     2        9       15         6
##  3     3       26       17        -9
##  4     4       17        6       -11
##  5     5       29       10       -19
##  6     6       25        8       -17
##  7     7       22       14        -8
##  8     8       12       11        -1
##  9     9       20       19        -1
## 10    10       24       11       -13
## # i 59 more rows
```

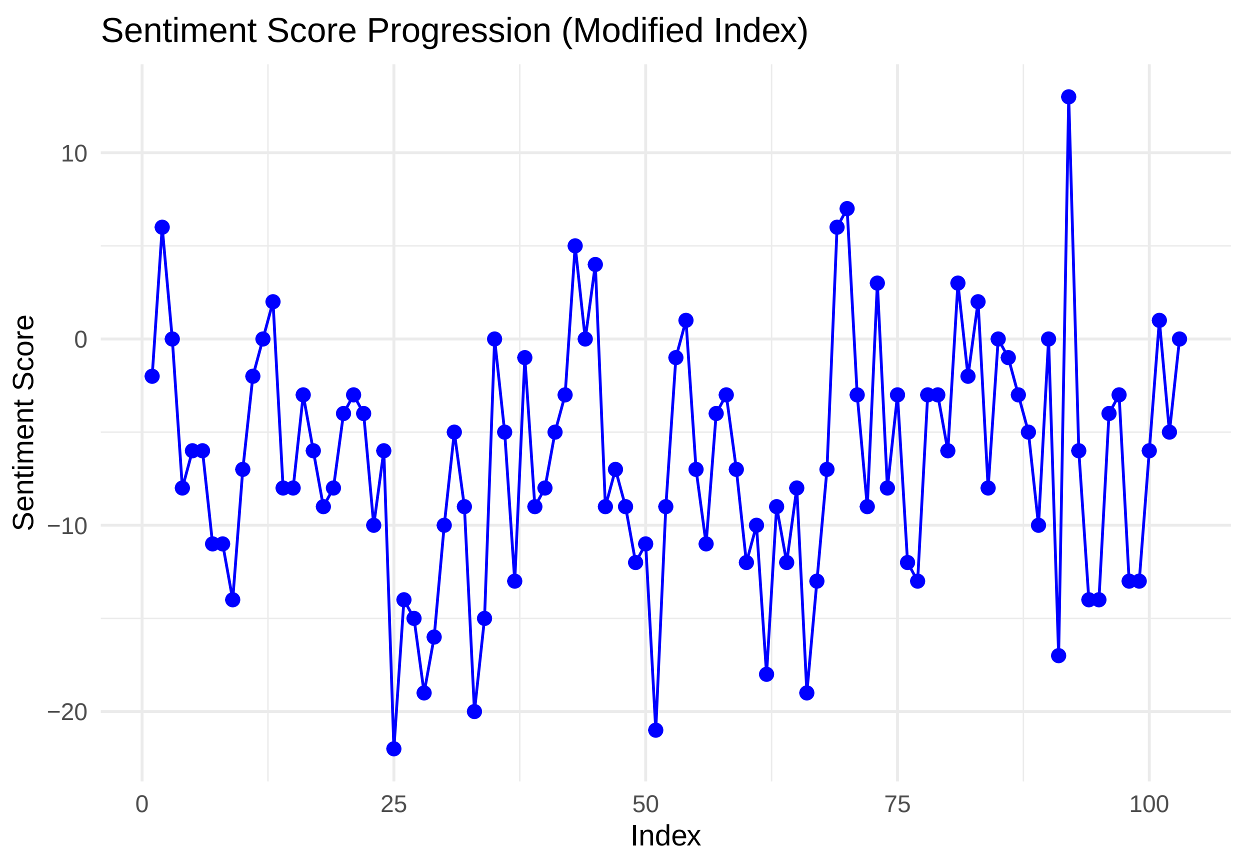
i. Create a plot of the sentiment scores as the text progresses.

```
sentiment_plot <- ggplot(sentiment_wide, aes(x = index, y = sentiment)) +
  geom_line(color = "blue") +
  geom_point(color = "blue", size = 2) +
  labs(x = "Index", y = "Sentiment Score", title = "Sentiment Score Progression") +
  theme_minimal()

# Print the plot
print(sentiment_plot)
```



j. The choice of 45 lines per chunk was pretty arbitrary. Try modifying the index value a few times and recreating the plot in i. Based on your plots, what can you conclude about the sentiment of the novel as it progresses?

```r
wild_time_modified <- wild %>%
  mutate(line = row_number(), index = floor(line/30) + 1)

wild_tidy_modified <- wild_time_modified %>%
  unnest_tokens(word, text)

wild_sentiments_modified <- wild_tidy_modified %>%
  inner_join(get_sentiments("bing"))

sentiment_counts_modified <- wild_sentiments_modified %>%
  count(index, sentiment)

sentiment_wide_modified <- sentiment_counts_modified %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0)

sentiment_wide_modified <- sentiment_wide_modified %>%
  mutate(sentiment = positive - negative)

sentiment_plot_modified <- ggplot(sentiment_wide_modified, aes(x = index, y = sentiment)) +
  geom_line(color = "blue") +
  geom_point(color = "blue", size = 2) +
  labs(x = "Index", y = "Sentiment Score", title = "Sentiment Score Progression (Modified Index)") +
  theme_minimal()

print(sentiment_plot_modified)
```



k. Let's look at the bigrams (2 consecutive words). Tokenize the text by bigrams.

```
wild_tokens <- wild %>%
  unnest_tokens(word, text)

wild_bigrams <- wild_tokens %>%
  mutate(next_word = lead(word)) %>%
  filter(!is.na(next_word)) %>%
  unite(bigram, word, next_word, sep = " ")

head(wild_bigrams)
```

```
## # A tibble: 6 x 2
##   gutenberg_id bigram
##          <int> <chr>
## 1          215 cover the
## 2          215 the call
## 3          215 call of
## 4          215 of the
## 5          215 the wild
## 6          215 wild by
```

l. Produce a sorted table that counts the frequency of each bigram and notice that stop words are still an issue.

```r
bigram_freq <- wild_bigrams %>%
  count(bigram)


bigram_freq <- bigram_freq %>%
  arrange(desc(n))
bigram_freq
```

```
## # A tibble: 20,538 x 2
##    bigram       n
##    <chr>    <int>
##  1 of the     246
##  2 in the     176
##  3 he was     131
##  4 to the     122
##  5 it was     111
##  6 and the    103
##  7 on the      86
##  8 he had      83
##  9 at the      71
## 10 into the    69
## # i 20,528 more rows
```