Lab 7

Insert Name

Math 241, Week 9

```r
#install.packages("tidytext")
#install.packages("wordcloud")
#install.packages("tm")

# Put all necessary libraries here
library(tidyverse)
library(tidytext)
library(wordcloud)
library(RColorBrewer)

# 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
afinn <- lexicon_afinn()
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!

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

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

```
# Convert names to lowercase
babynames$name <- tolower(babynames$name)

# Filter for names starting with 'z' for females in the year 2000
z_names <- babynames %>%
  filter(year == 2000) %>%
  filter(sex == "F") %>%
  filter(startsWith(name, "z")) %>%
  top_n(10)

z_names
```

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

```r
# Filter for names containing 'z'
z_containing_names <- babynames %>%
  filter(year == 2000) %>%
  filter(sex == "F") %>%
  filter(grepl("z", name)) %>%
  top_n(10)

z_containing_names
```

```
## # 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.

```r
# Filter for female names ending with 'z' for the year 2000 and select the top 10
z_end_names <- babynames %>%
  filter(year == 2000) %>%
  filter(sex == "F") %>%
  filter(endsWith(name, "z")) %>%
  top_n(10)

z_end_names
```

```
## # A tibble: 11 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
## 11  2000 F     Taiz        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!)

```r
#im gonna do this as one big code chunk


# Filter for top 10 names starting with 'z' for females in the year 2000
z_start_names <- babynames %>%
  filter(year == 2000) %>%
  filter(sex == "F") %>%
  filter(startsWith(name, "z")) %>%
  top_n(10)

# Filter for top 10 names ending with 'z' for females in the year 2000
z_end_names <- babynames %>%
  filter(year == 2000) %>%
  filter(sex == "F") %>%
  filter(endsWith(name, "z")) %>%
  top_n(10)

# Filter for top 10 names containing 'z' for females in the year 2000
z_containing_names <- babynames %>%
  filter(year == 2000) %>%
  filter(sex == "F") %>%
  filter(grepl("z", name)) %>%
  top_n(10)

# Join all three top ten lists
z_names <- inner_join(z_start_names, z_end_names, by = "name") %>%
  inner_join(z_containing_names, by = "name")

z_names
```

```
## # A tibble: 0 x 13
## # i 13 variables: year.x <dbl>, sex.x <chr>, name <chr>, n.x <int>,
## #   prop.x <dbl>, year.y <dbl>, sex.y <chr>, n.y <int>, prop.y <dbl>,
## #   year <dbl>, sex <chr>, n <int>, prop <dbl>
```

```r
# I guess theres no overlap??
```

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

```r
# Check for names containing numeric characters
names_with_numeric <- babynames[grepl("[0-9]", babynames$name), ]

# View the first few rows of the filtered dataset
head(names_with_numeric)
```

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

```r
#nope, none
```

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".

```r
# Check for names containing numbers
names_with_numeric <- babynames[grepl("one|two|three|four|five|six|seven|eight|nine", babynames$name), ]

# View the first few rows of the filtered dataset
head(names_with_numeric)
```

```
## # A tibble: 6 x 5
##    year sex   name          n      prop
##   <dbl> <chr> <chr>     <int>     <dbl>
## 1  1880 F     Ione          9 0.0000922
## 2  1880 F     Leone         7 0.0000717
## 3  1880 M     Colonel      11 0.0000929
## 4  1880 M     Jones         9 0.0000760
## 5  1880 M     Stonewall     9 0.0000760
## 6  1880 M     Lionel        8 0.0000676
```

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?

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

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

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
data("stop_words")
library(gutenbergr)
wild <- gutenberg_download(215)
```

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

```r
tidy_wild <- wild %>%
  unnest_tokens(output = word, input = text)
```

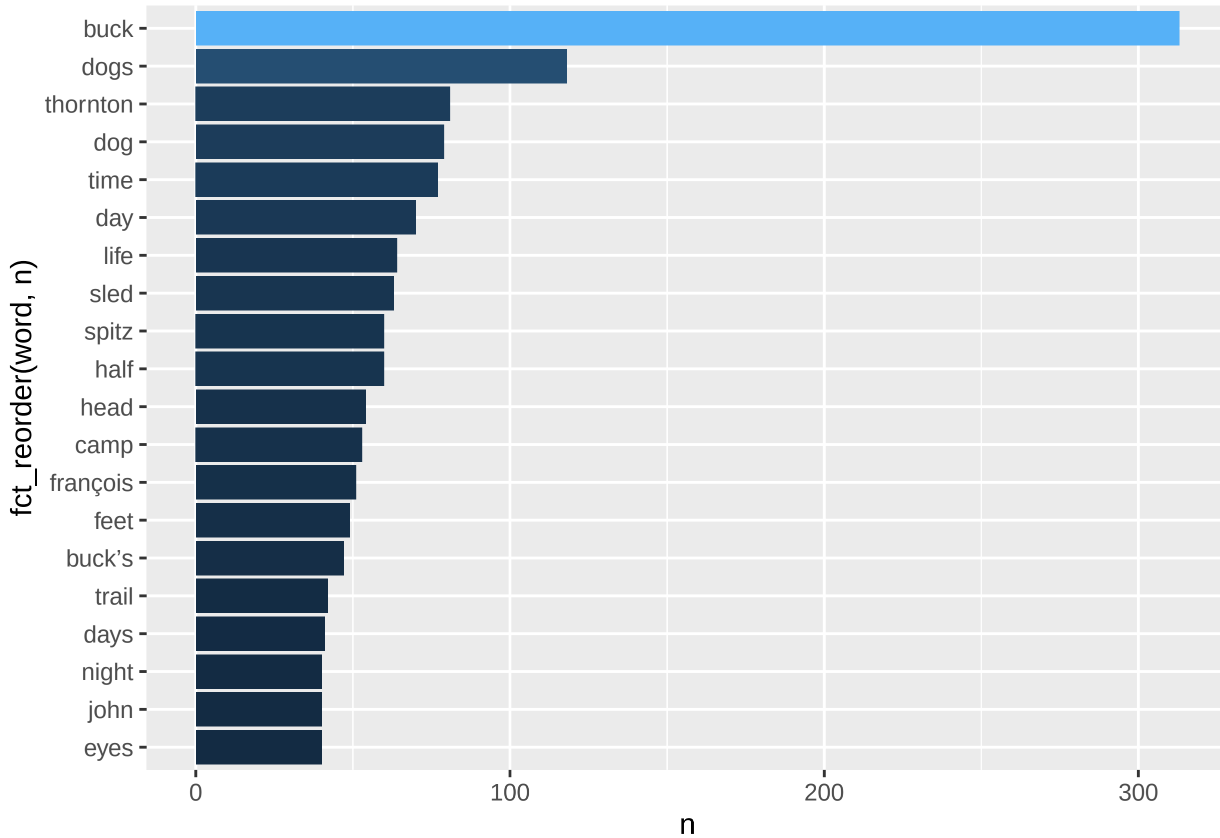
b. Find the frequency of the 20 most common words. First, remove stop words.

```r
tidy_wild_nostop <- tidy_wild %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE) %>%
  top_n(20)

tidy_wild_nostop
```

```
## # 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.

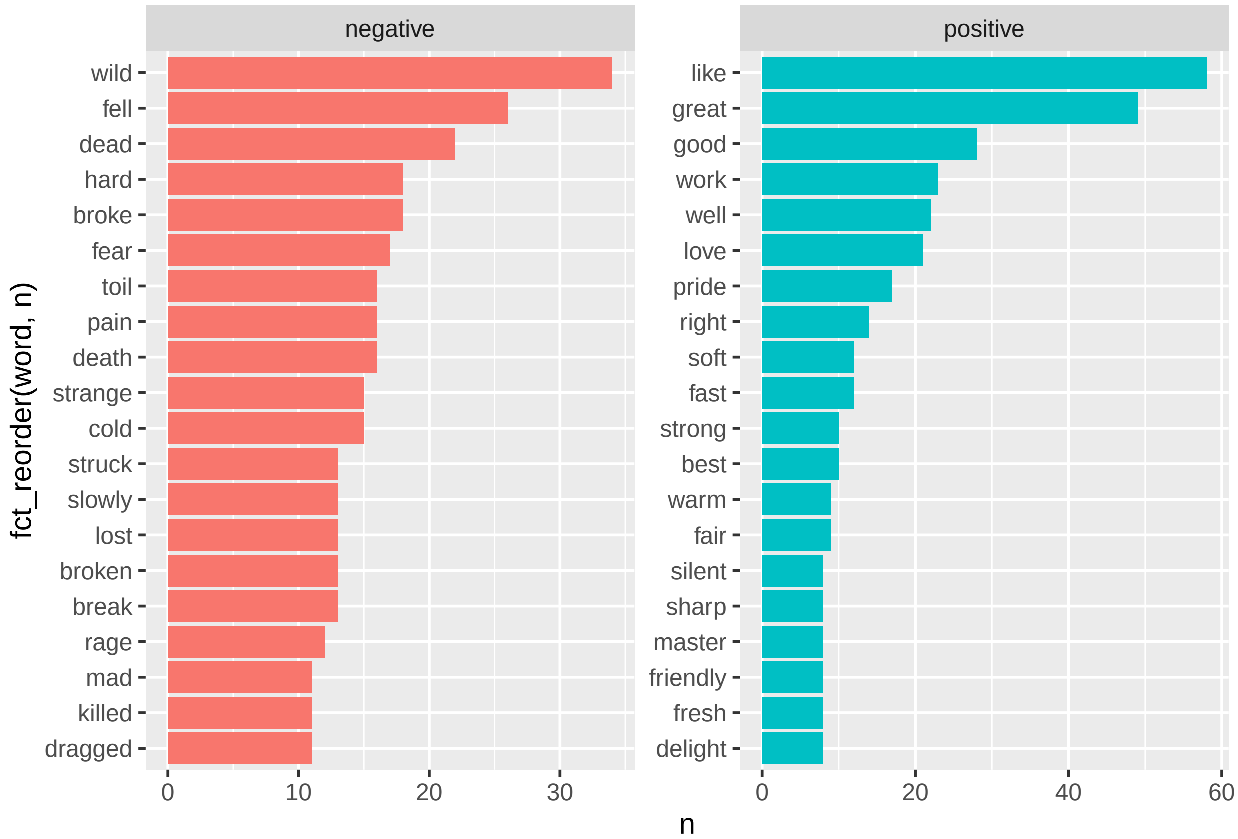
```r
# Barplot
tidy_wild_nostop %>%
  ggplot(aes(y = fct_reorder(word, n), x = n, fill = n)) +
  geom_col() +
  guides(fill = FALSE)
```




```
# Remove punctuation, numeric characters, and other non-alphabetic characters
tidy_wild_cleaned <- tidy_wild_nostop %>%
  filter(!str_detect(word, "[^A-Za-z]"))  # Keep only words containing alphabetical characters



# Wordcloud
pal <- brewer.pal(9, "Set1")

wordcloud(words = tidy_wild_cleaned$word, freq = tidy_wild_cleaned$n,
          scale = c(4, 1),
          rot.per = .5,
          colors = pal,
          random.order = FALSE
)
```

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?

```
tidy_wild %>%
  inner_join(get_sentiments("bing"), by = "word") %>%
  count(sentiment, word, sort = TRUE) %>%
  group_by(sentiment) %>%
  slice_head(n = 20) %>%
  ggplot(aes(y = fct_reorder(word, n), x = n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free")
```




```
#seems to be more negative than positive
```

Notes:

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

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?

```
#data("afinn")

# Compute sentiment scores for each word
tidy_wild_sentiment <- tidy_wild %>%
  inner_join(afinn, by = "word")%>%
  group_by(word) %>%
  summarize(avg_sentiment = mean(value, na.rm = T))

# Calculate the average sentiment score
wild_sentiment <- mean(tidy_wild_sentiment$avg_sentiment, na.rm = T)

wild_sentiment
```

```
## [1] -0.3787879
```

```r
#on average: slightly negative



# looking for the most impact words:

# Find the top ten most negative words
top_negative <- tidy_wild_sentiment %>%
  filter(avg_sentiment < 0) %>%
  arrange(avg_sentiment) %>%
  head(10)

# Find the top ten most positive words
top_positive <- tidy_wild_sentiment %>%
  filter(avg_sentiment > 0) %>%
  arrange(desc(avg_sentiment)) %>%
  head(10)

# Print the top ten most negative and positive words
print(top_negative)
```

```
## # A tibble: 10 x 2
##    word      avg_sentiment
##    <chr>             <dbl>
##  1 cock                 -5
##  2 hell                 -4
##  3 agonizing            -3
##  4 anger                -3
##  5 angry                -3
##  6 arrested             -3
##  7 bad                  -3
##  8 badly                -3
##  9 betrayed             -3
## 10 bloody               -3
```

```r
print(top_positive)
```

```
## # A tibble: 10 x 2
##    word      avg_sentiment
##    <chr>             <dbl>
##  1 miracle               4
##  2 terrific              4
##  3 triumph               4
##  4 win                   4
##  5 wonderful             4
##  6 adore                 3
##  7 affection             3
##  8 beautiful             3
##  9 best                  3
## 10 cheery                3
```

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”).

```
library(stringr)

# Split the text into sentences
wild_sentences <- str_split(tidy_wild_nostop$text, "(?<=[.!?])\\s+", simplify = TRUE)

# Find sentences containing the word "no"
wild_no_sentences <- wild_sentences[str_detect(wild_sentences, "\\bno\\b"), ]

# Print sentences containing the word "no"
print(wild_no_sentences)
```

```
## <0 x 0 matrix>
```

```
#ok, it is not finding any, I don't know whats up
```

g. Draw some conclusions about how “no” is used in the text. I can’t, for some reasons it seems to think there are no instances of “no”

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.
  - I don’t no what to do with this, I already have these two columns and I can easily make a third that combines them and just join in the sentiment data set I made with the original, I am confused by this step
- Compute a sentiment column by subtracting the negative score from the positive.

```
wild_time <- wild %>%
  mutate(line = row_number(), index = floor(line/90) + 1)
```

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

```
#tidying
tidy_wild_time <- wild_time %>%
  unnest_tokens(output = word, input = text)

#adding sentiments
tidy_wild_time_sentiments <- tidy_wild_time %>%
  inner_join(get_sentiments("bing"), by = c("word" = "word")) %>%
  group_by(index) %>%
  summarize(
    n_positive = sum(sentiment == "positive"),
    n_negative = sum(sentiment == "negative")
```

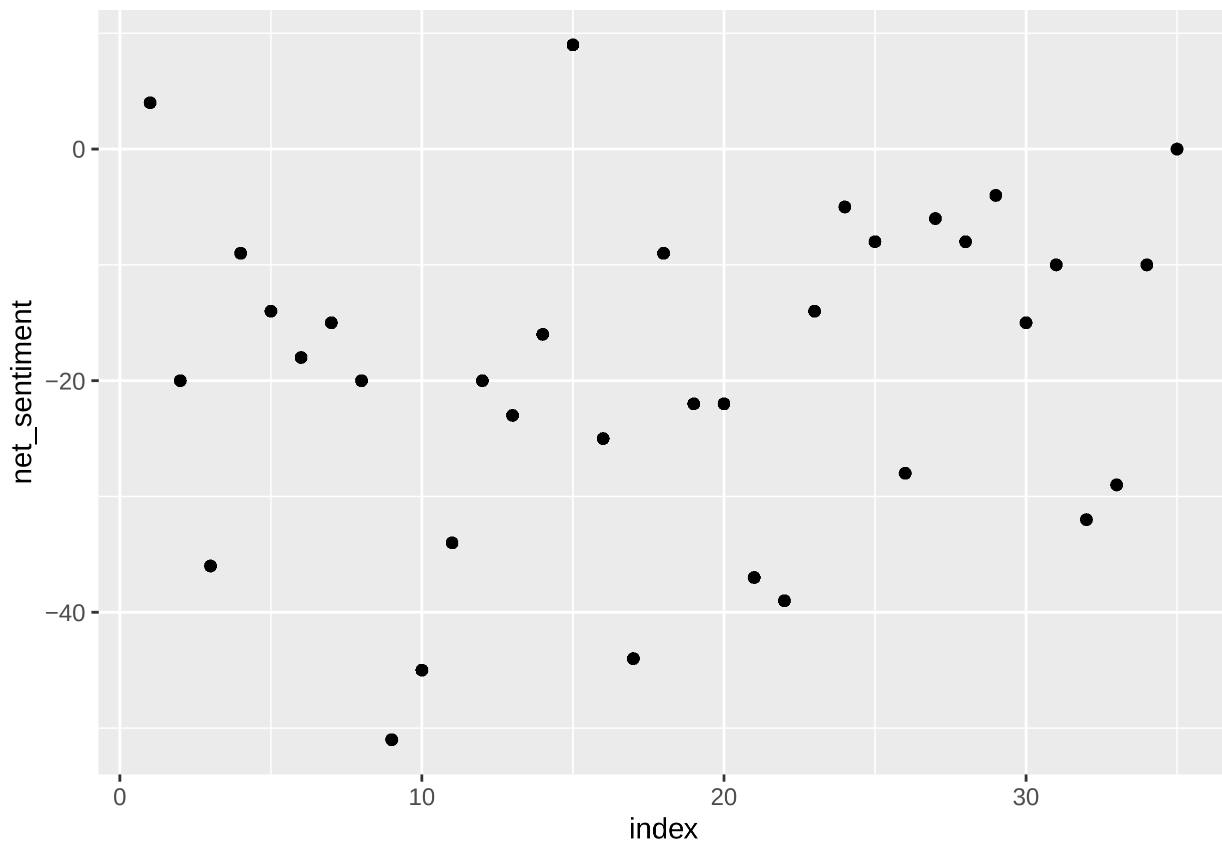
```r
  ) %>%
  mutate(
    net_sentiment = (n_positive - n_negative)
  )

# View the resulting dataset
print(tidy_wild_time_sentiments)
```

```
## # A tibble: 35 x 4
##    index n_positive n_negative net_sentiment
##    <dbl>      <int>      <int>         <int>
##  1     1         22         18             4
##  2     2         23         43           -20
##  3     3         18         54           -36
##  4     4         25         34            -9
##  5     5         30         44           -14
##  6     6         34         52           -18
##  7     7         31         46           -15
##  8     8         25         45           -20
##  9     9         17         68           -51
## 10    10         16         61           -45
## # i 25 more rows
```

```r
# Inner join tidy_wild_time and tidy_wild_time_sentiments
joined_wild_time_sentiments <- inner_join(tidy_wild_time, tidy_wild_time_sentiments, by = "index")
```

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

```r
joined_wild_time_sentiments %>%
  ggplot(
    aes(x = index, y = net_sentiment)
  ) + geom_point()
```




```
#its kinda just all over the place
```

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?

-It seems like the sentiment is kinda just all over the place not really varying with the novel's linear progresion

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

```
wild_bigrams <- wild %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  mutate(i = row_number()) %>%    # add index for later grouping
  unnest_tokens(word, bigram, drop = FALSE) %>%    # tokenize bigrams into words
  group_by(i) %>%    # group by bigram index
  filter(n() == 2) %>%    # drop bigram instances where only one word left
  summarise(bigram = unique(bigram), .groups = "drop")
```

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

```
# Count the frequency of each bigram
wild_bigram_freq <- wild_bigrams %>%
  count(bigram, sort = TRUE)
```

```r
# View the sorted table
print(wild_bigram_freq)
```

```
## # A tibble: 18,907 x 2
##    bigram       n
##    <chr>    <int>
##  1 of the     233
##  2 in the     172
##  3 he was     127
##  4 to the     116
##  5 it was     107
##  6 and the     95
##  7 on the      80
##  8 he had      77
##  9 at the      68
## 10 into the    65
## # i 18,897 more rows
```