Word Prediction: Exploratory Analysis

Data loading

Due to the size of the files, the downloading and unzipping of the data files is not included in this document. So I start with loading the data.

```
library(dplyr)
library(tidytext)
library(ggplot2)

set.seed(42)

path <- './Coursera-SwiftKey/final/en_US/'

blogs <- 'en_US.blogs.txt'
news <- 'en_US.news.txt'
twitter <- 'en_US.twitter.txt'

if (!exists('linesBlogs')) {
  connection <- file(paste0(path, blogs), "rb")
  linesBlogs <- readLines(connection, encoding="utf-8")
  close(connection)
}

if (!exists('linesNews')) {
  connection <- file(paste0(path, news), "rb")
  linesNews <- readLines(connection, encoding="utf-8")
  #lines <- readLines(connection, 1)
  close(connection)
}

if (!exists('linesTwitter')) {
  connection <- file(paste0(path, twitter), "rb")
  linesTwitter <- readLines(connection, encoding="utf-8")
  #lines <- readLines(connection, 1)
  close(connection)
}
```

Data preprocessing and cleaning

As the dataframes generated when calculating n-grams quickly become very large, I limit the exploratory analysis - as well as the latter modeling - to a subset of the data consisting of 20,000 entries from each of the three data sources.

```
# sample for testing purposes
sampleBlogs <- linesBlogs[sample(length(linesBlogs), 20000)]
sampleNews <- linesNews[sample(length(linesNews), 20000)]
sampleTwitter <- linesTwitter[sample(length(linesTwitter), 20000)]

# combine into one dataframe
text_df <- rbind(data.frame(text = sampleBlogs), data.frame(text = sampleNews), data.frame(te
xt = sampleTwitter))
text_df$text <- as.character(text_df$text)
```

There were some issues with data quality. Special characters including a number of punctuation marks were imported incorrectly. This was not so dramatic, as those were by and large eliminated or at least did not play a significant role in the analyis, except for quoation marks used e.g. in *it’s* or *you’re* which required some extra attention.

```
# remove numbers as they do not seem really helpful for prediction and are not removed automa
tically at later stage when creating ngrams
text_df$text <- gsub('[0-9]+', '', text_df$text)

# replace 'Ã¢â□¬â□¢' and 'Ã¢â□¬Å□' with single and apostrophe as apostprophe is not properl
y read
text_df$text <- gsub('Ã¢â□¬â□¢', '\'', text_df$text)
text_df$text <- gsub('Ã¢â□¬Å□', '"', text_df$text)

# remove remaining 'Ã¢' and 'Ã□'
text_df$text <- gsub('Ã¢â□¬', '"', text_df$text)
text_df$text <- gsub('[Ã¢Ã□]', '"', text_df$text)
```

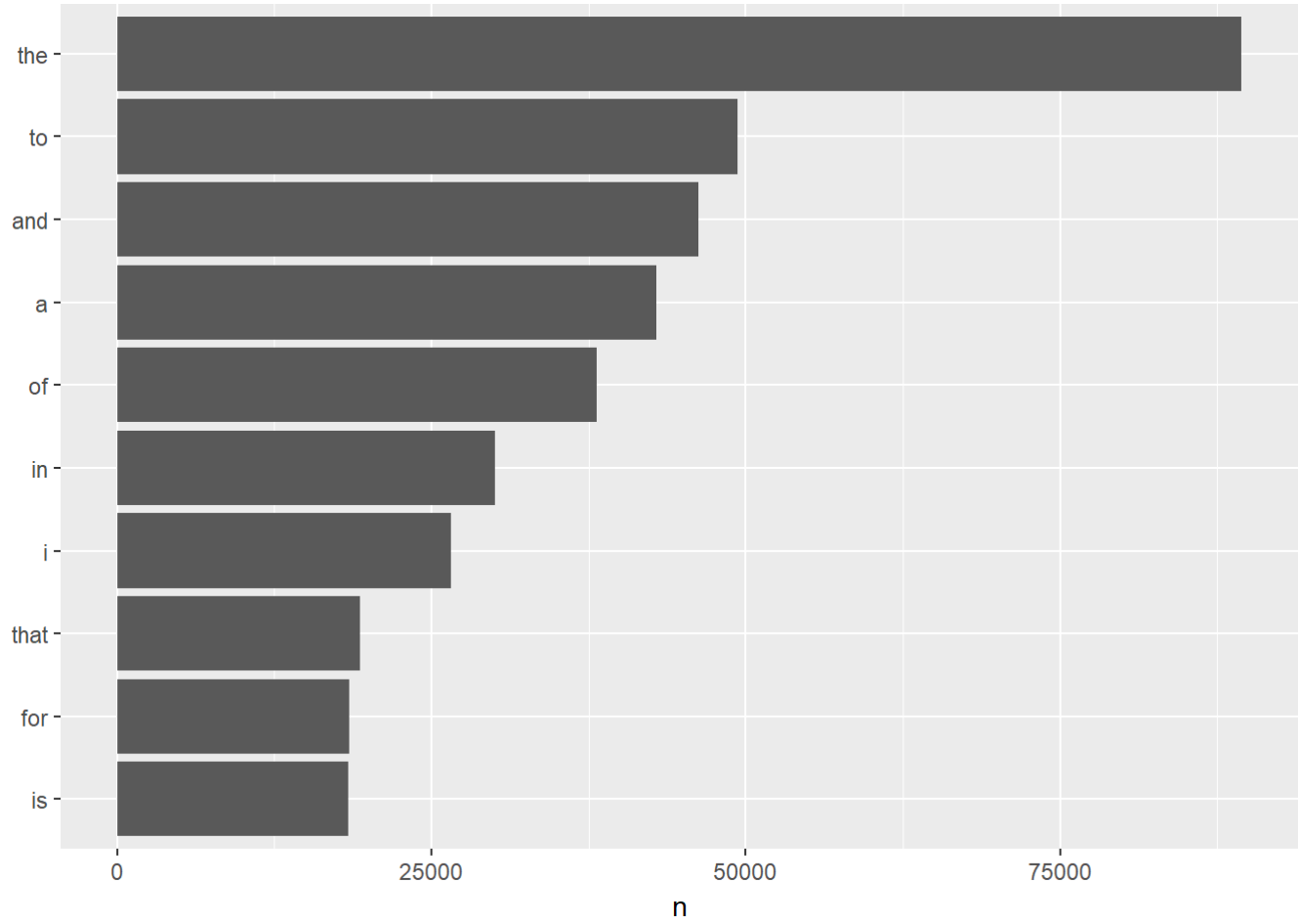
Exploratory data analysis

I then used this data to generate n-grams, starting with n = 1, i.e. just words. A plot of the most frequent words shows that prepositions, articles and simple verbs are most frequent.

```
# generate oneGrams
oneGram <- text_df %>% unnest_tokens(word, text)

# count oneGrams, select most popular 10 and display
oneGramTop10 <- oneGram %>% count(word, sort = TRUE) %>%
  top_n(10) %>% mutate(word = reorder(word, n))

# this is ordering without using dplyr
# oneGramTop10$word <- factor(oneGramTop10$word , levels = oneGramTop10$word [order(oneGramTo
p10$n)])
ggplot(aes(word, n), data = oneGramTop10) + geom_col() + xlab(NULL) + coord_flip()
```

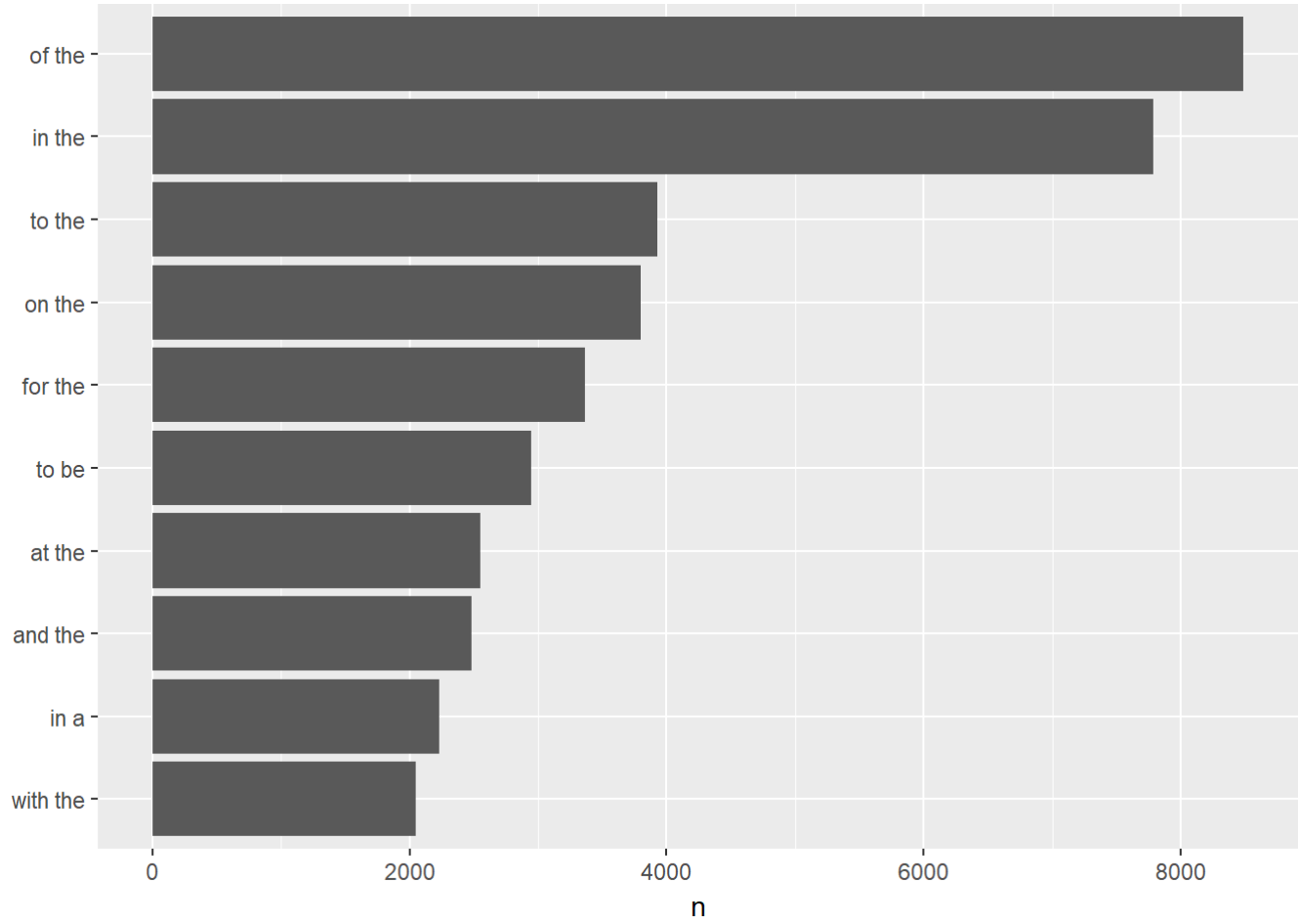


For bigrams, I also created a chart with the most common bigrams. Again, combinations of prepositions, articles and simple verbs predominate. Furthermore, I plotted a line with the log of the number of occurances which clearly shows that only very few bigrams appear more than once in the sampled texts.

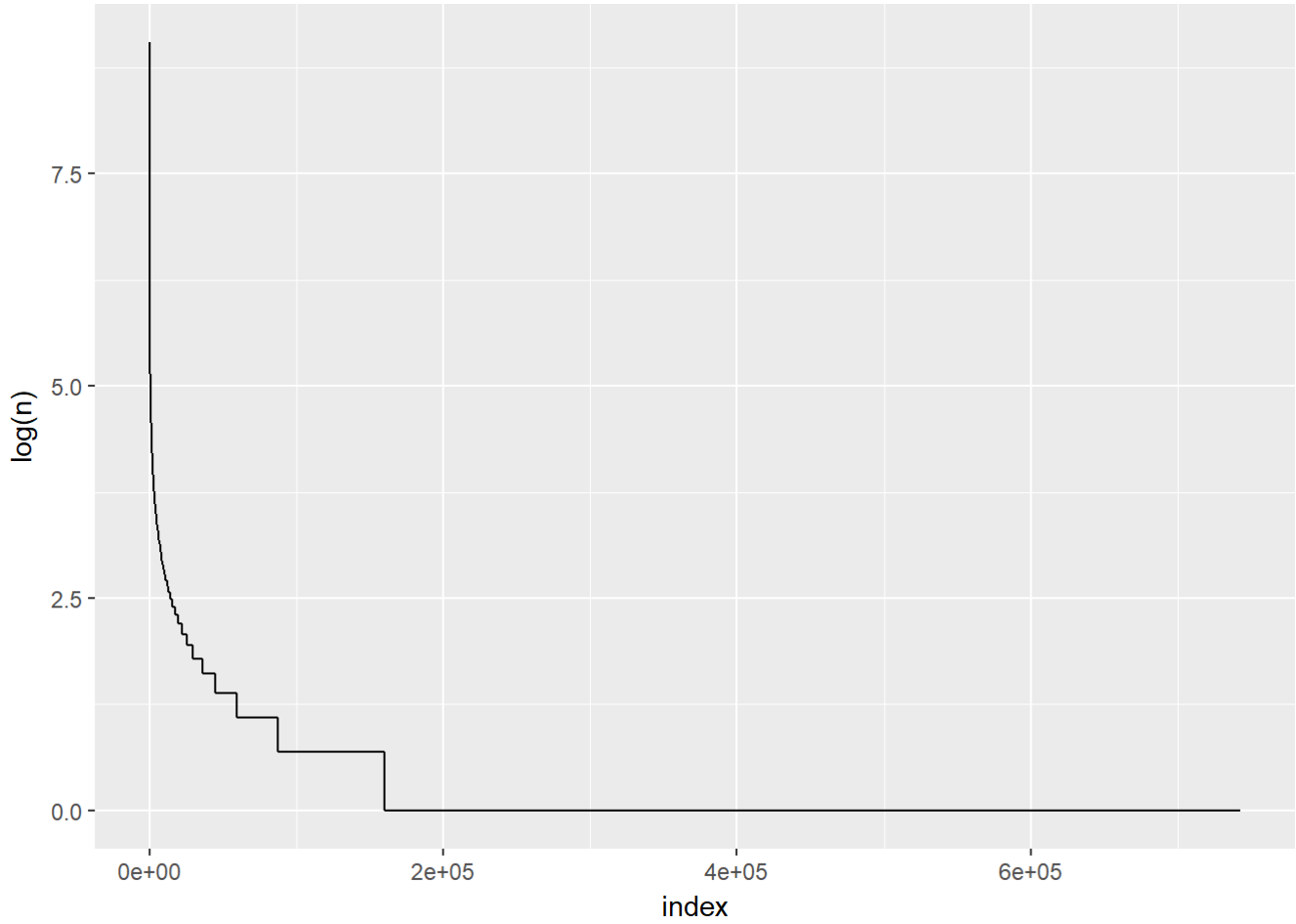
```
# generate biGrams
biGram <- text_df %>% unnest_tokens(bigram, text, token = "ngrams", n = 2)

# count biGrams, select most popular 10 and display
biGram <- biGram %>% count(bigram, sort = TRUE) %>% mutate(bigram = reorder(bigram, n))
biGramTop10 <- biGram[1:10,]

ggplot(aes(bigram, n), data = biGramTop10) + geom_col() + xlab(NULL) + coord_flip()
```




```
# display density distribution of biGrams on log scale
biGram$index <- seq(1, dim(biGram)[1])
ggplot(aes(index, log(n)), data = biGram) + geom_line()
```

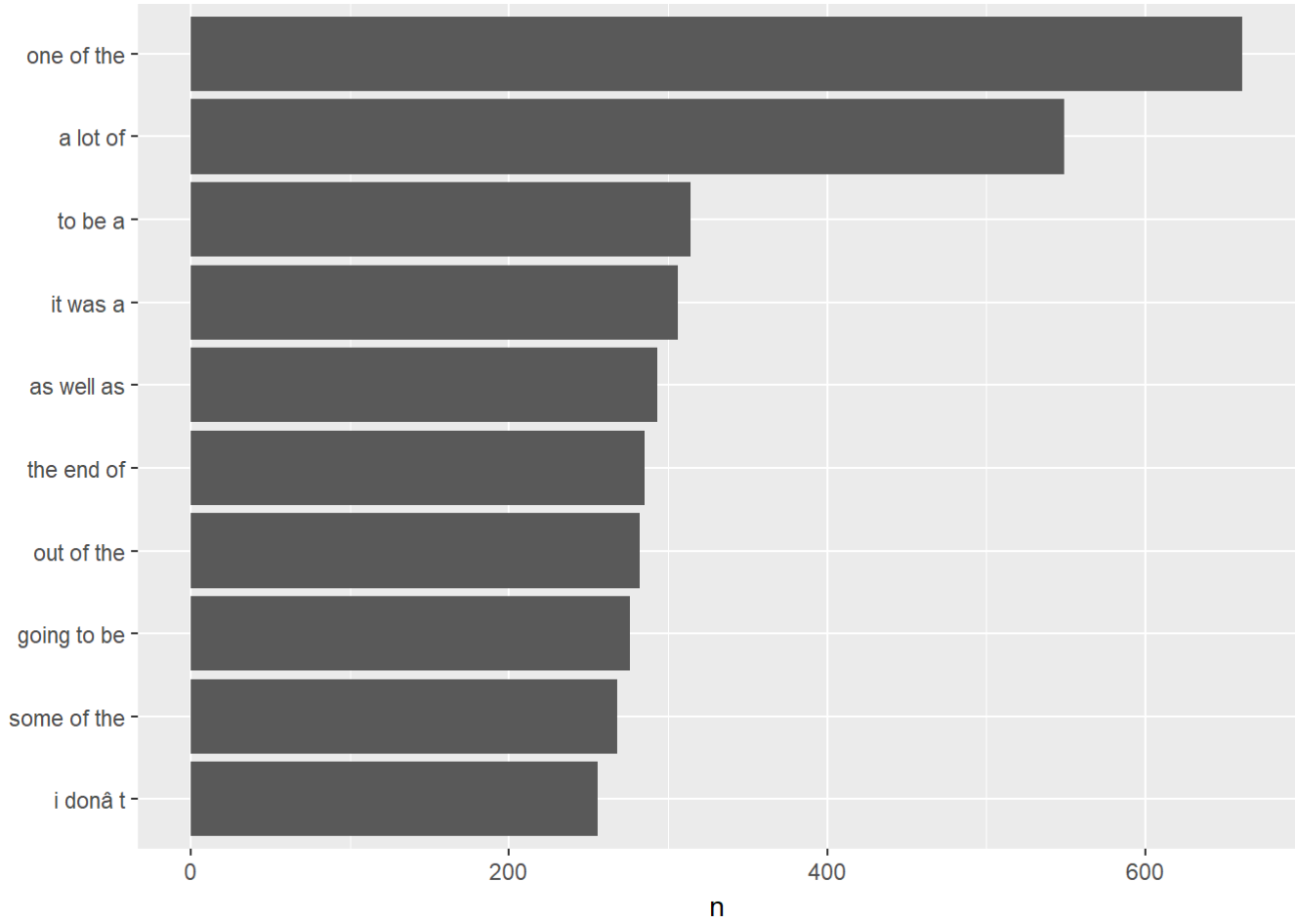


For 3-grams and 4-grams, I followed the same approach. The charts show the same pattern with even steeper declines in the log of the number of occurences as n increases.

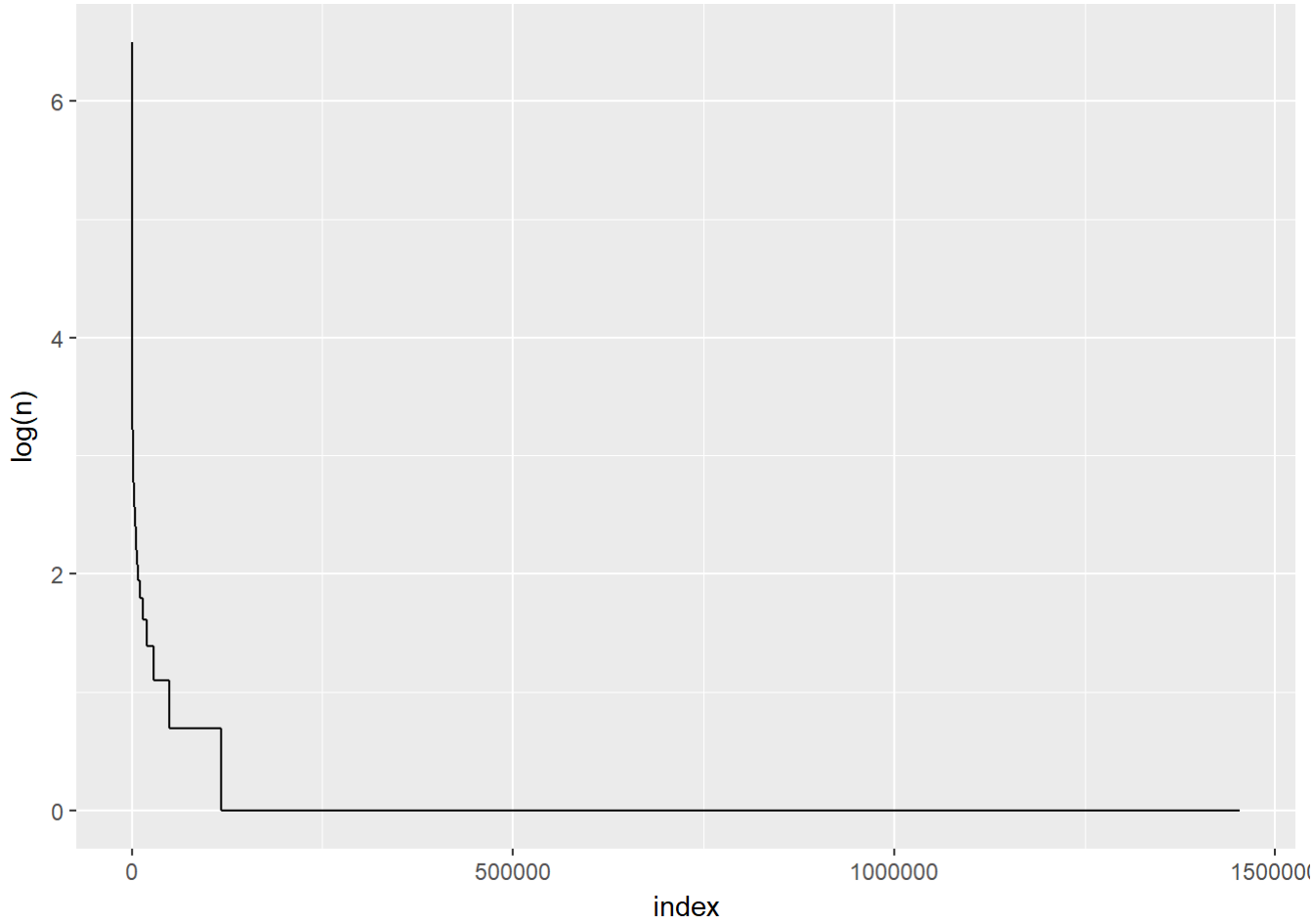
```
# generate threeGrams
threeGram <- text_df %>% unnest_tokens(threegram, text, token = "ngrams", n = 3)

# count threeGrams, select most popular 10 and display
threeGram <- threeGram %>% count(threegram, sort = TRUE) %>% mutate(threegram = reorder(three
gram, n))
threeGramTop10 <- threeGram[1:10,]

ggplot(aes(threegram, n), data = threeGramTop10) + geom_col() + xlab(NULL) + coord_flip()
```



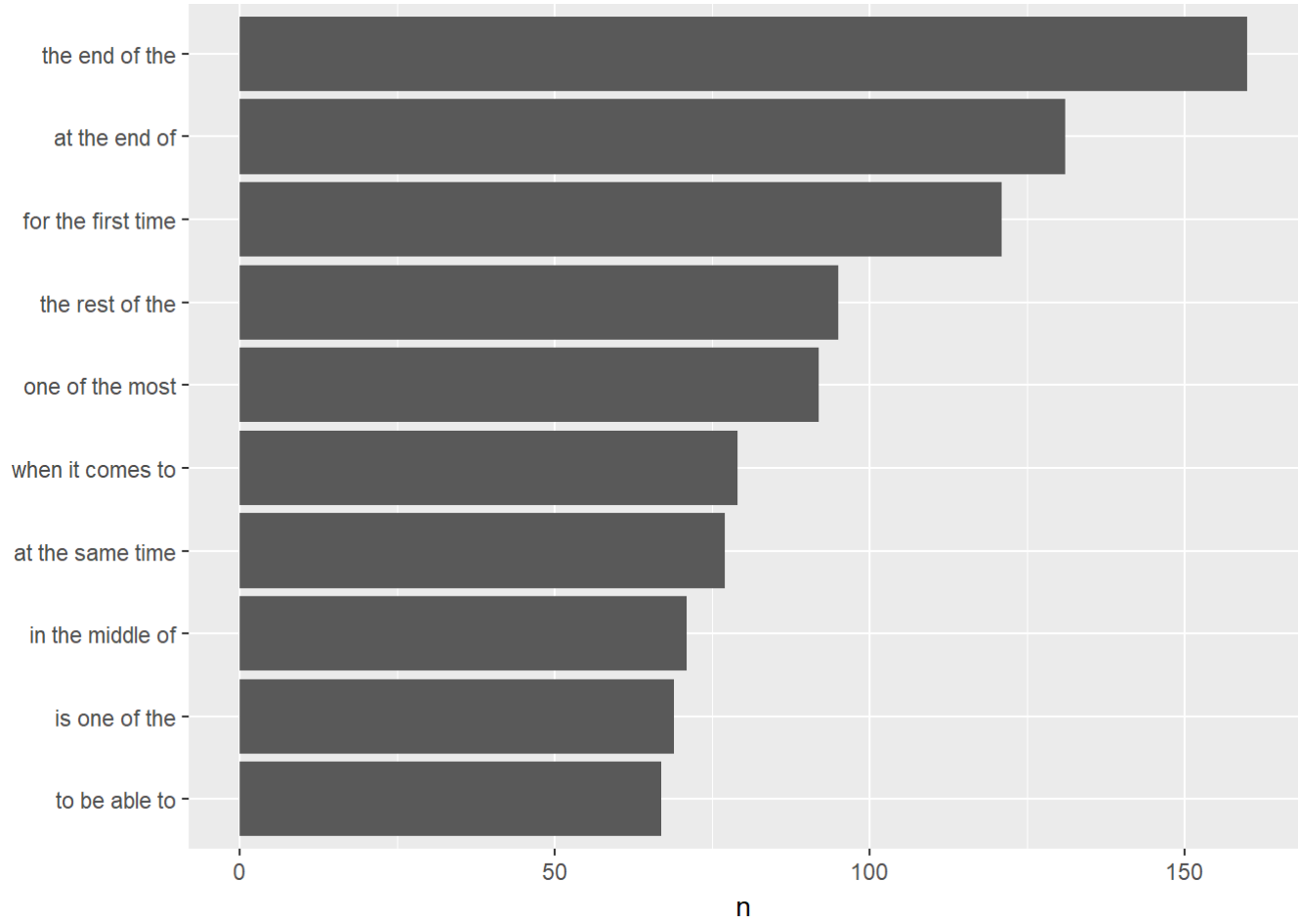

```
# display density distribution of threeGrams on log scale
threeGram$index <- seq(1, dim(threeGram)[1])
ggplot(aes(index, log(n)), data = threeGram) + geom_line()
```



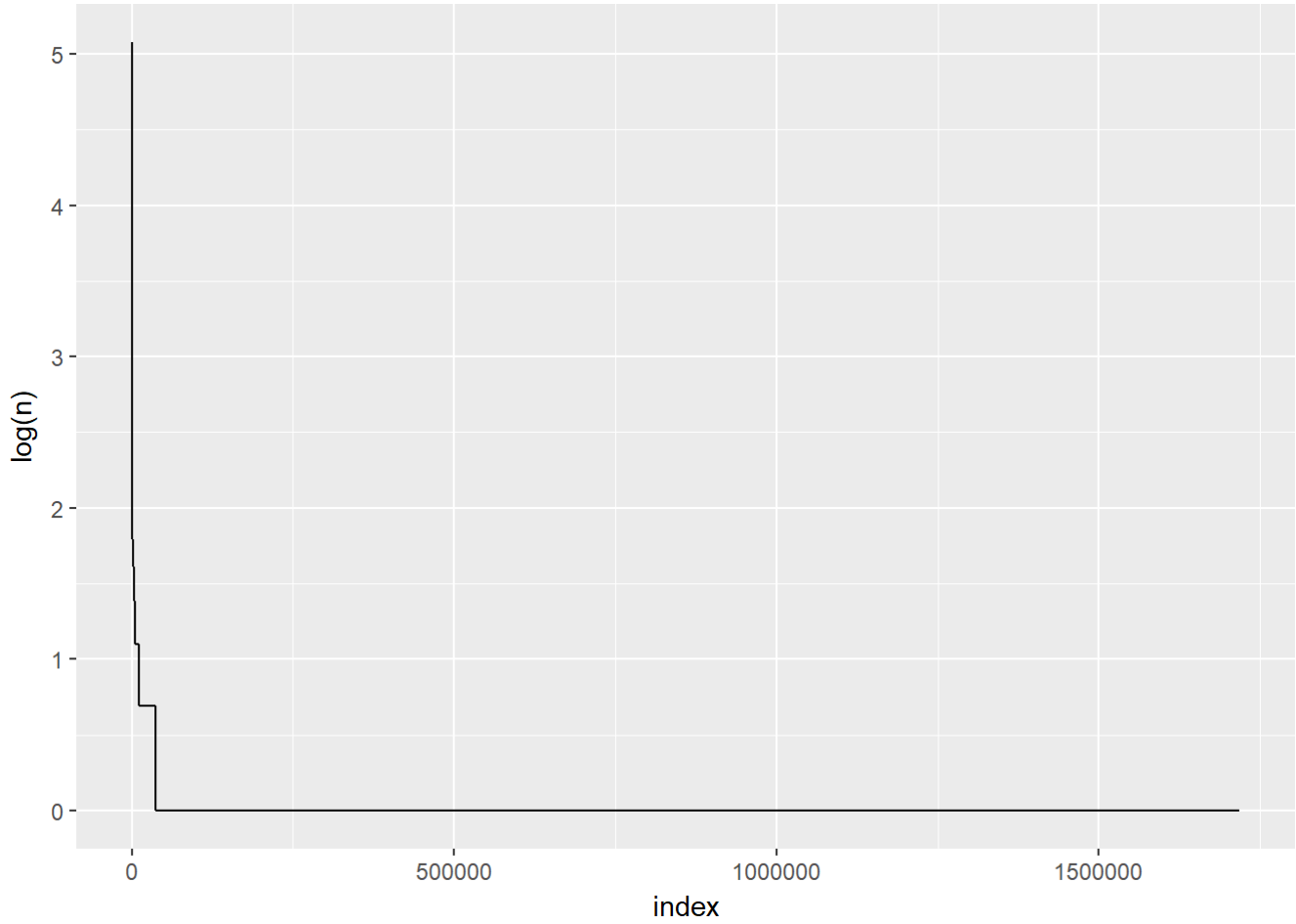

```
# generate fourGrams
fourGram <- text_df %>% unnest_tokens(fourgram, text, token = "ngrams", n = 4)

# count fourGrams, select most popular 10 and display
fourGram <- fourGram %>% count(fourgram, sort = TRUE) %>% mutate(fourgram = reorder(fourgram,
 n))
fourGramTop10 <- fourGram[1:10,]

ggplot(aes(fourgram, n), data = fourGramTop10) + geom_col() + xlab(NULL) + coord_flip()
```




```
# display density distribution of fourGrams on log scale
fourGram$index <- seq(1, dim(fourGram)[1])
ggplot(aes(index, log(n)), data = fourGram) + geom_line()
```



For the prediction, I need to combine n-grams and (n-1)-grams in order to detect what word will likely follow a given (n-1)-gram.

```
# add (n-1)-grams to n-grams
biGram$bigram <- as.character(biGram$bigram)
biGram$onegram <- sapply(biGram$bigram, function(x) paste(strsplit(x, ' ')[[1]][1]))

threeGram$threegram <- as.character(threeGram$threegram)
threeGram$bigram <- sapply(threeGram$threegram, function(x) paste(strsplit(x, ' ')[[1]][1],
                                                    strsplit(x, ' ')[[1]][2]))

fourGram$fourgram <- as.character(fourGram$fourgram)
fourGram$threegram <- sapply(fourGram$fourgram, function(x) paste(strsplit(x, ' ')[[1]][1],
                                                        strsplit(x, ' ')[[1]][2],
                                                        strsplit(x, ' ')[[1]][3]))
```

Each of the tables has several hundred thousand rows with n-grams. I choose only those entries which appear more than once, which leads to a substiantial reduction in size due to the fact that - as seen above - a lot of n-grams appear only once.

```
# find most frequent (n-1)-grams
oneGram2Relevant <- oneGramTop10 # no need for further filtering

biGram2 <- biGram %>% group_by(onegram) %>% mutate(count = n())
biGram2 <- biGram2 %>% filter(n == max(n)) %>% arrange(desc(n), onegram, bigram, n) # please
 note: this can return more than one row per (n-i)-gram
biGram2Relevant <- biGram2 %>% filter((n>1) | (count > 1))

threeGram2 <- threeGram %>% group_by(bigram) %>% mutate(count = n())
threeGram2 <- threeGram2 %>% filter(n == max(n)) %>% arrange(desc(n), bigram, threegram, n) #
 please note: this can return more than one row per (n-i)-gram
threeGram2Relevant <- threeGram2 %>% filter((n>1) | (count > 1))

fourGram2 <- fourGram %>% group_by(threegram) %>% mutate(count = n())
fourGram2 <- fourGram2 %>% filter(n == max(n)) %>% arrange(desc(n), threegram, fourgram, n) #
 please note: this can return more than one row per (n-i)-gram
fourGram2Relevant <- fourGram2 %>% filter((n>1) | (count > 1))
```

This data is then used to build the actual prediction algorithm.

Conclusion

The exploratory analysis showed three main things from my perspective:

- The dataset is very large, and with n-grams, the size of the tables tends to grow exponentially. This - at least in my approach - requires restriction the amount of data used to a subset.
- After limited data cleaning, the data proved of sufficient quality for further exploratory analysis.
- Very few n-grams appear frequently. Especially with n > 2, a lot of n-grams are already quite unique. At second glance, though, this seems not too surprising as the number of potential combinations grows. exponentially in n.