```ruby
require 'nokogiri'
require 'open-uri'
require 'lemmatizer'
require 'histogram/array'
require 'chartkick'
include Chartkick::Helper

# ruby doesn't have a native implementation of linked lists
class Node
  attr_accessor :val, :next

  def initialize(val, next_node)
      @val = val
      @next = next_node
  end
end

class LinkedList

  def initialize(val)
    @head = Node.new(val, nil)
  end

  def add(val)
    current = @head
    while current.next != nil
      current = current.next
    end
    current.next = Node.new(val, nil)
  end

  def return_list
    elements = []
    current = @head
    while current.next != nil
      elements << current
      current = current.next
    end
    elements << current
  end

  def return_values
    return_list.map{|n| n.val}
  end
end

class WordCounter
  attr_accessor :source_url, :text_source, :lem, :preprocess_storage, :wor
d_count, :articles_with_counts, :word_count_table, :graph_data

  def initialize
    @source_url =
"http://gss.uva.nl/binaries/content/assets/programmas/information-
studies/txt-for-assignment-data-science.txt?3015083536432"
```

```
    @text_source = "./txt-for-assignment-data-science.txt"
    @lem = Lemmatizer.new
    @preprocess_storage = {}
    @articles_with_counts = {}
    @word_count_table = {}
    retrieve_tokenize_and_lemmatize
    @graph_data = format_for_plotting(histogram_data_for_collection)
  end

  def retrieve_tokenize_and_lemmatize
    doc = Nokogiri::HTML(open(@text_source))
    doc.search('text').each_with_index do |link, index|
      # simple white space tokenizer with ruby regex sufficient
      tokenized_text = link.content.scan(/\w+/)
      @preprocess_storage[index] = tokenized_text.map{ |token|
@lem.lemma(token.downcase) }
    end
  end

  def count(article_tokens)
    article_tokens.each_with_object({}) do |token, article_word_count|
      article_word_count[token] ||= 0
      article_word_count[token] += 1
    end
  end

  def count_tokens_by_article
    @preprocess_storage.each do |article_id, tokenized_article|
      @articles_with_counts[article_id + 1] = count(tokenized_article)
    end
  end

  def count_tokens_by_collection
    count(@preprocess_storage.values.flatten)
  end

  def all_frequencies_descending
    count_tokens_by_collection.values.sort.reverse
  end

  def article_counts_by_word
    @articles_with_counts.each do |article_id, counts_by_word|
      counts_by_word.each do |word, count|
        if @word_count_table[word] == nil
          @word_count_table[word] = LinkedList.new([article_id, count])
        else
          @word_count_table[word].add([article_id, count])
        end
      end
    end
  end

  def histogram_data_for_collection
    collection_counts = count_tokens_by_collection.values
```

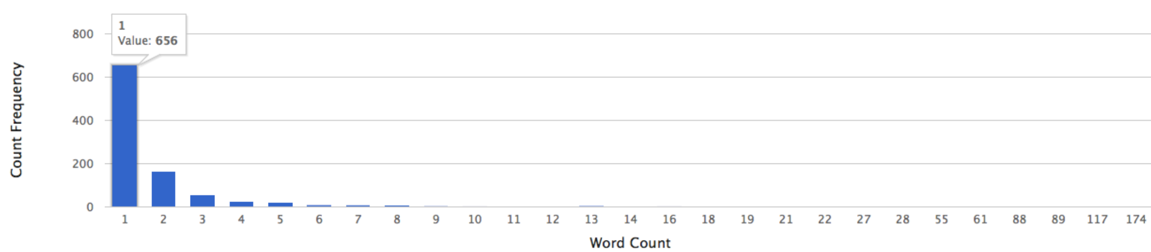
```
    (bins, freq) =
collection_counts.histogram(collection_counts.uniq.sort)
  end

  def format_for_plotting(histogram_arrays)
    frequency_count_hash = {}
    count_array = histogram_arrays[0]
    frequency_array = histogram_arrays[1]
    count_array.each_with_index do |count, index|
      frequency_count_hash[count.to_i] = frequency_array[index].to_i
    end
    frequency_count_hash
  end

  def plot_histogram
    graph_html = column_chart @graph_data, xtitle: 'Word Count', ytitle:
'Count Frequency'
    open('frequency_count_plot.html', 'w+') do |f|
      f.puts '<script src="https://www.google.com/jsapi"></script>'
      f.puts '<script src="chartkick.js"></script>'
      f.puts graph_html
    end
  end

  def count_source
    count_tokens_by_article
    article_counts_by_word
    @word_count_table
  end
end
```

word_counter.rb usage:

```
>> irb
>> require_relative 'word_counter'
>> w = WordCounter.new; nil
>> w.count_source
>> w.word_count_table.keys # shows all lexical types
>> w.word_count_table["the"] #show linked list type count per article for "the"
>> w.word_count_table["the"].return_values
>> [[1, 94], [2, 12], [3, 68]]
>> w. plot_histogram # generates frequency_count_plot.html
>> exit
```

open frequency_count_plot.html
chart.js required in same directory as frequency_count_plot.html resources here:
https://github.com/iamliamc/uva_assignment

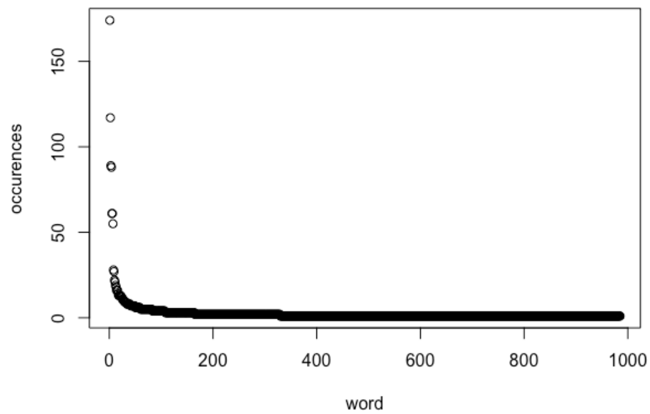
*In this analysis, only the content of the articles found within the <text> tags are included.
screen_shot: frequency_count_plot.html



Basic Features of the lexical type-token frequencies generated from word_counter.rb

```
>> irb
>> require_relative 'word_counter'
>> w = WordCounter.new; nil
>> w.graph_data
>> {1=>656, 2=>165, 3=>57, 4=>25, 5=>23, 6=>10, 7=>9, 8=>7, 9=>4, 10=>3, 11=>2,
12=>2, 13=>5, 14=>1, 16=>3, 18=>1, 19=>1, 21=>1, 22=>1, 27=>1, 28=>1, 55=>1,
61=>2, 88=>1, 89=>1, 117=>1, 174=>1}
# The first entry in this hash indicates there are 656 distinct words that occurred 1 time
>> w.all_frequencies_descending
# Outputs the frequency of each word in the corpus as an array (i.e. the integer 1, 656
times, the integer 2, 165 times etc. )
```

```
occurrences <- c(values from the ruby expression: w.all_frequencies_descending)
word <- 1:length(occurrences)
plot(occurrences ~ word)
```



> describe(occurrences)
vars 1
mean 2.61
n 985
sd 8.64
median 1
trimmed 1.39
mad 0
min 1
max 174
range 173
skew: 13.22
kurtosis 210.65
se 0.28

The mean of the data values is larger than the median, and a highly positive kurtosis indicates that the distribution has a greater number and more extreme outliers than does the normal distribution.

Outliers:
> OutVals = boxplot(occurrences, plot=FALSE)$out
174 117 89 88 61 61 55 28 27 22 21 19 18 16 16 16 14 13 13 13 13 13 12 12 11 11
10 10 10 9 9 9 9 8 8 8 8 8 8 8 7 7 7 7 7 7 7 7 7 6 6 6 6 6 6 6 6
6 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

(Standard deviation: 8.64 / mean: 2.61) = Coefficient of variation 3.31

Without additional context, a reasonable heuristic is that a coefficient of variation > 1 indicates a high relative variability.

Fitting a model:

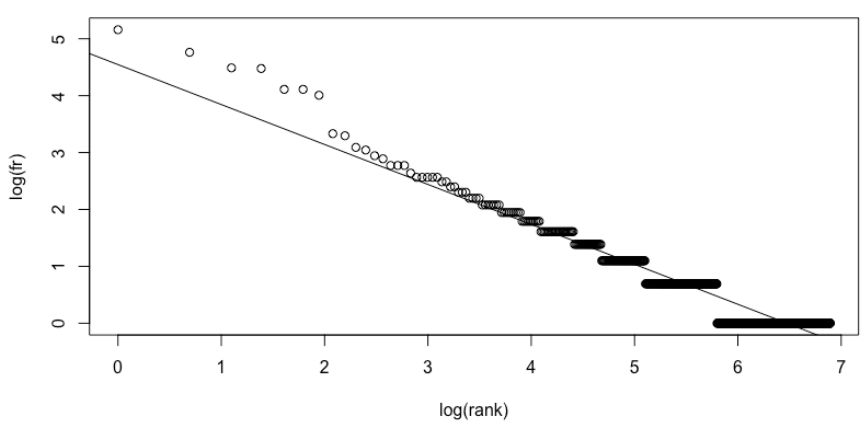
It is known that type-token analysis of natural language corpora follow a Zipfian distribution where:

> "the frequency of any word is inversely proportional to its rank in the frequency table. The most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc… Zipf's law is most easily observed by plotting the data on a log-log graph, with the axes being log (rank order) and log (frequency). The data conform to Zipf's law to the extent that the plot is linear." (https://en.wikipedia.org/wiki/Zipf%27s_law)

Question: is Zipf's Law a good way to "characterize this distribution?"

Log-Log Linear regression of the type-token frequencies from the LA Times corpus::

```
fr <- c(values from the ruby expression: w.all_frequencies_descending)
rank <- 1:length(fr)
log_fr <- log(fr)
log_rank <- log(rank)
plot(log(fr) ~ log(rank))
reg <- lm(log(fr) ~ log(rank))
abline(reg, untf=F)
reg
summary(reg)
plot(reg)
```

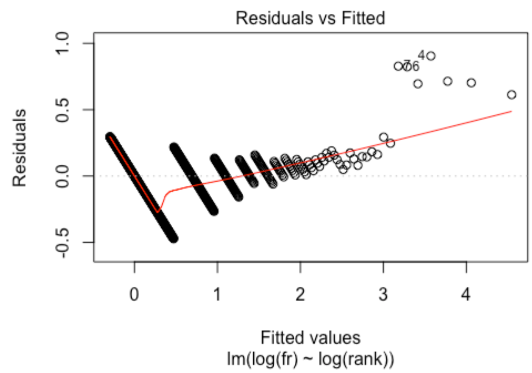


Coefficients:
(Intercept)   log(rank)
4.5461   -0.7025

Residual standard error: 0.2002 on 983 degrees of freedom
Multiple R-squared: 0.9229, Adjusted R-squared: 0.9228

F-statistic: 1.176e+04 on 1 and 983 DF, p-value: < 2.2e-16

The linear regression has a high R-squared value and a statistically significant p-value. However if you look at the residuals from this regression it does not seem like our fit is extracting all the relevant information from the distribution. Almost half of the corpus is made up of words that occurred only 1, 2 or 3 times. The small size of the corpus seems a likely issue in this regression.



There is a CRAN package for Zipfian analysis that takes a more advance approach and supports this conclusion:
https://cran.r-project.org/web/packages/zipfR/index.html

A frequency spectrum (the fundamental data structure in zipfR) summarizes a frequency distribution in terms of number of types (Vm) per frequency class (m), i.e., it reports how many distinct types occur once, how many types occur twice, and so on.

```
require(zipfR)
freq <- c(values from w.graph_data.keys)
freq_occurences <- c(values from w.graph_data.values)
wordFrequencySpectrum <- spc(freq, freq_occurences)
z <- lnre("fzm", d, exact=FALSE)
zFit <- lnre("fzm", wordFrequencySpectrum, exact=FALSE)
summary(zFit)
```

<u>Finite Zipf-Mandelbrot LNRE model:</u>

Goodness-of-fit (multivariate chi-squared test):

```
      X2 df           p
  32.2375  3 4.66371e-07
```

The low p-value indicates that the predicted probabilities from the model differ significantly from the observed probabilities in the data. Thus, Zipf's law does not fit the data well. Since the corpus is relatively small at 2570 lexical tokens with 985 distinct lexical types, we would expect the goodness-of-fit to become statistically significant as the size of the corpus grows.