

# CS 505 Homework 01: Exploratory Data Analysis

Due Wednesday 9/13 at midnight (1 minute after 11:59 pm) in Gradescope (with a grace period of 6 hours)

You may submit the homework up to 24 hours late (with the same grace period) for a penalty of 10%.

All homeworks will be scored with a maximum of 100 points; point values are given for individual problems, and if parts of problems do not have point values given, they will be counted equally toward the total for that problem.

The goals of this first homework are that you

1. Get up to speed on Python, Jupyter Notebooks, and Google Colab (by going through the various tutorials or other resources as needed);
2. Practice the submission process through Gradescope (and allow us to practice the grading process);
3. Get started thinking about programming with textual data; and
4. Practice exploratory data analysis, often an excellent first step in any project, to understand the basic characteristics of your data set.

Note: I strongly recommend you work in **Google Colab** (the free version) to complete homeworks in this class; in addition to (probably) being faster than your laptop, all the necessary libraries will already be available to you, and you don't have to hassle with `conda`, `pip`, etc. and resolving problems when the install doesn't work. But it is up to you! You should go through the necessary tutorials listed on the web site concerning Colab and storing files on a Google Drive. And of course, Dr. Google is always ready to help you resolve your problems.

I will post a "walk-through" video ASAP on my [Youtube Channel](#).

## Submission Instructions

You must complete the homework by editing **this notebook** and submitting the following two files in Gradescope by the due date and time:

- A file `HW01.ipynb` (be sure to select `Kernel -> Restart and Run All` before you submit, to make sure everything works); and

- A file `HW01.pdf` created from the previous.

For best results obtaining a clean PDF file on the Mac, select `File -> Print Review` from the Jupyter window, then choose `File-> Print` in your browser and then `Save as PDF`. Something similar should be possible on a Windows machine -- just make sure it is readable and no cell contents have been cut off. Make it easy to grade!

The date and time of your submission is the last file you submitted, so if your IPYNB file is submitted on time, but your PDF is late, then your submission is late.

## Collaborators (5 pts)

Describe briefly but precisely

1. Any persons you discussed this homework with and the nature of the discussion;
2. Any online resources you consulted and what information you got from those resources; and
3. Any AI agents (such as chatGPT or CoPilot) or other applications you used to complete the homework, and the nature of the help you received.

A few brief sentences is all that I am looking for here.

```
lists of Websites/tools consulted :  
1.geeksforgeeks.com: brushing up python, looking for help in  
counter and lists  
2.ChatGpt : for figuring out errors regarding  
lists,Percentages conversion of the data , help in word  
frequencies(2.D)  
3.tutorialsPoint.com : for brushing up on plots and figuring  
out for 'dotted line'
```

# Overview

In this homework, we will download some text from the well-known [Brown Corpus](#) in the Natural Language Toolkit (NLTK), and explore some of its statistical properties.

In addition to exploring the Wiki page just linked, you may also want to consult section 1.3 of the book chapter [Accessing Text Corpora and Lexical Resources](#) accompanying the NLTK system.

We are going to collect some basic statistical information about this corpus, and display it in various useful forms. Consult the [tutorials](#) as described above (especially `PythonRefresher.ipynb`) for recipes for dictionaries, sets, plots, and bar charts; for this first homework, we are providing sample outputs of at least the figures at the bottom of this notebook to guide your thinking. You should try to duplicate these closely, especially with titles, axis labels, and legends.

You may add additional code as needed, but anything other than simply filling in where it says `# your code here` should be accompanied by appropriate comments explaining what it does.

Read through the next few cells and understand what the code is doing, and then proceed to the problems.

```
In [1]: import numpy as np
import nltk

# The first time you will need to download the corpus:

nltk.download('brown')

# After the first time, Python will see that you already have it and not dow
# This is a typical paradigm for datasets that you download onto your local

[nltk_data] Downloading package brown to
[nltk_data]      /Users/mohanthota/nltk_data...
[nltk_data]   Package brown is already up-to-date!
Out[1]: True
```

```
In [2]: from nltk.corpus import brown

# We can access various components of this multi-text corpus: words, sentences,
# paragraphs, both raw and tagged with part-of-speech (POS) labels.
# (We won't be using the tagged ones right now.)

print("Words (a list of strings):\n")
print(brown.words())

print("\nWords with POS tags:\n")
print(brown.tagged_words())

print("\nSentences (a list of lists of strings):\n")
print(brown.sents())

print("\nSentences with POS-tagged words:\n")
print(brown.tagged_sents())

print("\nParagraphs (a list of lists of lists of strings):\n")
print(brown.paras())

print("\nParagraphs in various categories, here are reviews:\n")
print(brown.paras(categories='reviews'))
```

Words (a list of strings):

```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Words with POS tags:

```
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

Sentences (a list of lists of strings):

```
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', 'Atlanta's', 'recent', 'primary', 'election', 'produced', '``', 'no', 'evidence', '``', 'that', 'any', 'irregularities', 'took', 'place', ''], ['The', 'jury', 'further', 'said', 'in', 'term-end', 'presentments', 'that', 'the', 'City', 'Executive', 'Committee', 'which', 'had', 'over-all', 'charge', 'of', 'the', 'election', 'deserves', 'the', 'praise', 'and', 'thanks', 'of', 'the', 'City', 'of', 'Atlanta', 'for', 'the', 'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.'], ...]
```

Sentences with POS-tagged words:

```
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-TL'), ('Jury', 'NN-TL'), ('said', 'VBD'), ('Friday', 'NR'), ('an', 'AT'), ('investigation', 'NN'), ('of', 'IN'), ('Atlanta's', 'NP$'), ('recent', 'JJ'), ('primary', 'NN'), ('election', 'NN'), ('produced', 'VBD'), ('``', '``'), ('no', 'AT'), ('evidence', 'NN'), ('``', '``'), ('that', 'CS'), ('any', 'DTI'), ('irregularities', 'NNS'), ('took', 'VBD'), ('place', 'NN'), ('.', '.')] [ ('The', 'AT'), ('jury', 'NN'), ('further', 'RBR'), ('said', 'VBD'), ('in', 'I
```

```
N'), ('term-end', 'NN'), ('presentments', 'NNS'), ('that', 'CS'), ('the', 'A
T'), ('City', 'NN-TL'), ('Executive', 'JJ-TL'), ('Committee', 'NN-TL'), ('',
','), ('which', 'WDT'), ('had', 'HVD'), ('over-all', 'JJ'), ('charge', 'NN
'), ('of', 'IN'), ('the', 'AT'), ('election', 'NN'), ('', ','), (''', '''),
), ('deserves', 'VBZ'), ('the', 'AT'), ('praise', 'NN'), ('and', 'CC'), ('th
anks', 'NNS'), ('of', 'IN'), ('the', 'AT'), ('City', 'NN-TL'), ('of', 'IN-TL
'), ('Atlanta', 'NP-TL'), (''', '''), ('for', 'IN'), ('the', 'AT'), ('mann
er', 'NN'), ('in', 'IN'), ('which', 'WDT'), ('the', 'AT'), ('election', 'NN
'), ('was', 'BEDZ'), ('conducted', 'VBN'), ('.', '.')] , ...]
```

Paragraphs (a list of lists of lists of strings):

```
[[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'inve
stigation', 'of', 'Atlanta's', 'recent', 'primary', 'election', 'produced',
'', 'no', 'evidence', '', 'that', 'any', 'irregularities', 'took', 'plac
e', '.']], [['The', 'jury', 'further', 'said', 'in', 'term-end', 'presentmen
ts', 'that', 'the', 'City', 'Executive', 'Committee', '', 'which', 'had', '
over-all', 'charge', 'of', 'the', 'election', '', 'deserves', 'the',
'praise', 'and', 'thanks', 'of', 'the', 'City', 'of', 'Atlanta', '', 'for'
, 'the', 'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.
']], ...]
```

Paragraphs in various categories, here are reviews:

```
[[['It', 'is', 'not', 'news', 'that', 'Nathan', 'Milstein', 'is', 'a', 'wiza
rd', 'of', 'the', 'violin', '.'], ['Certainly', 'not', 'in', 'Orchestra', 'H
all', 'where', 'he', 'has', 'played', 'countless', 'recitals', '', 'and', '
where', 'Thursday', 'night', 'he', 'celebrated', 'his', '20th', 'season', 'w
ith', 'the', 'Chicago', 'Symphony', 'Orchestra', '', 'playing', 'the', 'Bra
hms', 'Concerto', 'with', 'his', 'own', 'slashing', '', 'demon-ridden', 'ca
denza', 'melting', 'into', 'the', 'high', '', 'pale', '', 'pure', 'and', '
lovely', 'song', 'with', 'which', 'a', 'violinist', 'unlocks', 'the', 'heart
', 'of', 'the', 'music', '', 'or', 'forever', 'finds', 'it', 'closed', '.']
], [['There', 'was', 'about', 'that', 'song', 'something', 'incandescent', '
', 'for', 'this', 'Brahms', 'was', 'Milstein', 'at', 'white', 'heat', '.'],
['Not', 'the', 'noblest', 'performance', 'we', 'have', 'heard', 'him', 'play
', '', 'or', 'the', 'most', 'spacious', '', 'or', 'even', 'the', 'most', '
eloquent', '.'], ['Those', 'would', 'be', 'reserved', 'for', 'the', 'orchest
ra's', 'great', 'nights', 'when', 'the', 'soloist', 'can', 'surpass', 'himse
lf', '.'], ['This', 'time', 'the', 'orchestra', 'gave', 'him', 'some', 'supe
rb', 'support', 'fired', 'by', 'response', 'to', 'his', 'own', 'high', 'mood
', '.'], ['But', 'he', 'had', 'in', 'Walter', 'Hendl', 'a', 'willing', 'cond
uctor', 'able', 'only', 'up', 'to', 'a', 'point', '.']], ...]
```

## Problem One (40 points): Characters

First we will explore this corpus at the level of characters. Each part of the problem is worth 10 points.

In [3]: # Part A

```
# Print out the number of occurrences of characters in the brown corpus. The
# upper or lower case, parentheses, white space, any character (printing or
# Make this readable by a human, e.g., "There are xxx occurrences of charact

# Hint: use the brown.words list and read about the Python join function. Yo
# of characters a lot in this problem, so calculate it once and assign it to

# I strongly recommend you read about f-strings (introduced in Python 3.6) a
# for Python print statements.

# Your code here
joined_strings= ''.join(brown.words())
#print(joined_strings[0:200]) #checking the concated string
occ_of_char=len(joined_strings)
print(f'There are {occ_of_char} occurrences of characters in the Brown corpus
```

There are 4965882 occurrences of characters in the Brown corpus.

In [4]: # Part B

```
# Print out the number of unique characters which occur in the Brown corpus
# occurrences do not count), and then print out a string consisting of all t
# Again, always print this information out in a readable form: "There are xx

# Hint: read about the Python functions set(...) and sorted(...)

# Your code here
c = set([])
for i in range (len(joined_strings)):
    c.add(joined_strings[i])
#used sets and sorted function
print(f'There are {len(c)} unique charectors and the given are : {sorted(c)}
```

There are 83 unique charectors and the given are : ['!', '\$', '%', '&', '"',  
'(', ')', '\*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',  
'8', '9', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',  
'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',  
'Z', '[', ']', '^', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',  
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '  
{', '}'']

NOTE: This distinction is sometimes confusing, here is an example:

text: "abdbab" There are 6 occurrences of characters, but only 3 unique characters: { 'a','b','d' }

We will NOT be using the list of unique characters in the rest of this problem; whenever 'characters' are mentioned, we mean occurrences of characters, as in Part A.

```
In [5]: # Part C

# Display a bar chart of the percentages of the characters that are in the f
# ASCII upper-case letters, digits, punctuation marks (the Brown corpus does
# Display this as a bar chart, labelling each of the bars as 'Lower', 'Upper
# You do not need to show the exact percentages in the figure (see the sampl

# Use a figsize of (8,4) so that the figures are not too small.

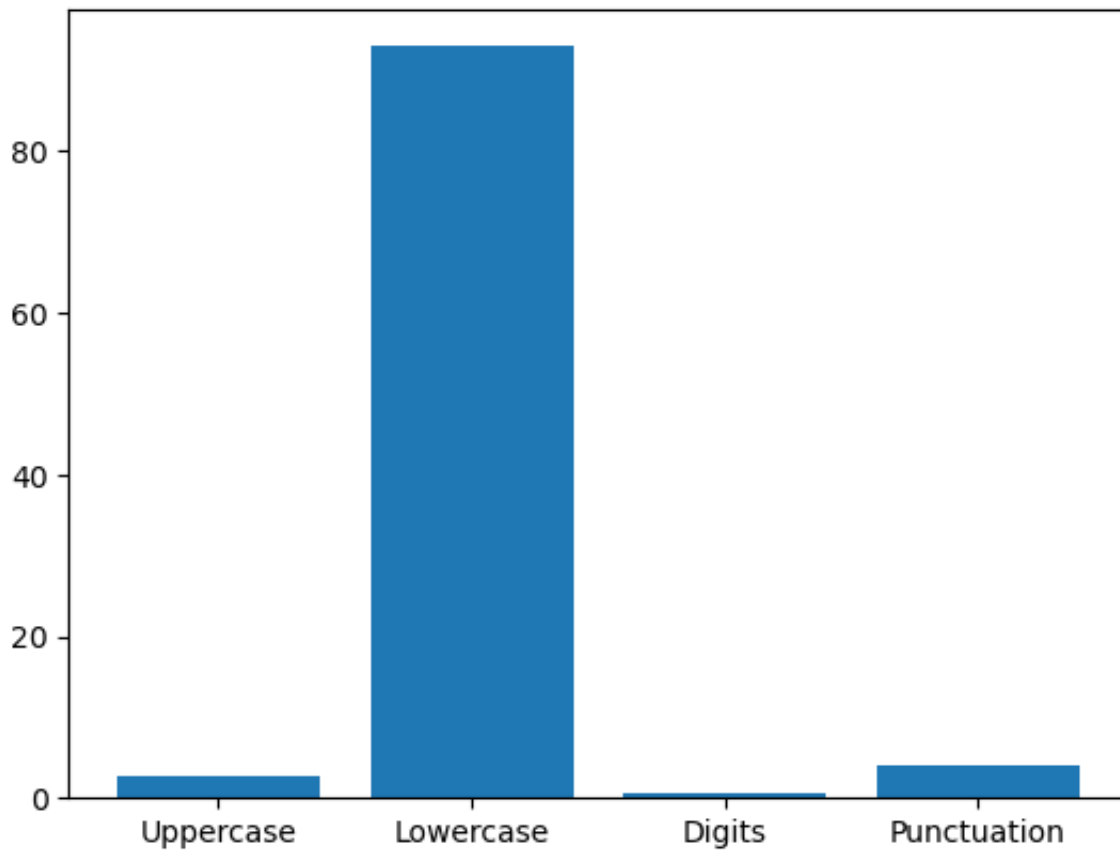
# Hint: import the Python string library (see https://docs.python.org/3/libr
# string constants specified there. Look at the "Probability Distribution fo
# PythonRefresher for how to create bar charts with labels on the bars.

# Be sure to give percentages on the Y axis, not probabilities.

# For a sample of what we expect, see the very bottom of this notebook.

# Your code here
import numpy as np
import string
import matplotlib.pyplot as plt
count_up = sum(1 for i in joined_strings if i in string.ascii_uppercase)
count_low = sum(1 for i in joined_strings if i in string.ascii_lowercase)
count_digits = sum(1 for i in joined_strings if i in string.digits)
count_punc = sum(1 for i in joined_strings if i in string.punctuation)

total=count_up+count_low+count_digits+count_punc #total count = 4965882 (ie
x_axis=np.array(["Uppercase", "Lowercase", "Digits", "Punctuation"])
y_axis =np.array([count_up/total*100,count_low/total*100,count_digits/total*
plt.bar(x_axis,y_axis)
plt.show()
```



```
In [6]: # Part D

# Print out a bar chart of the percentages of each character, in decreasing
# so upper and lower letters are different; for example 'H' and 'h' are the

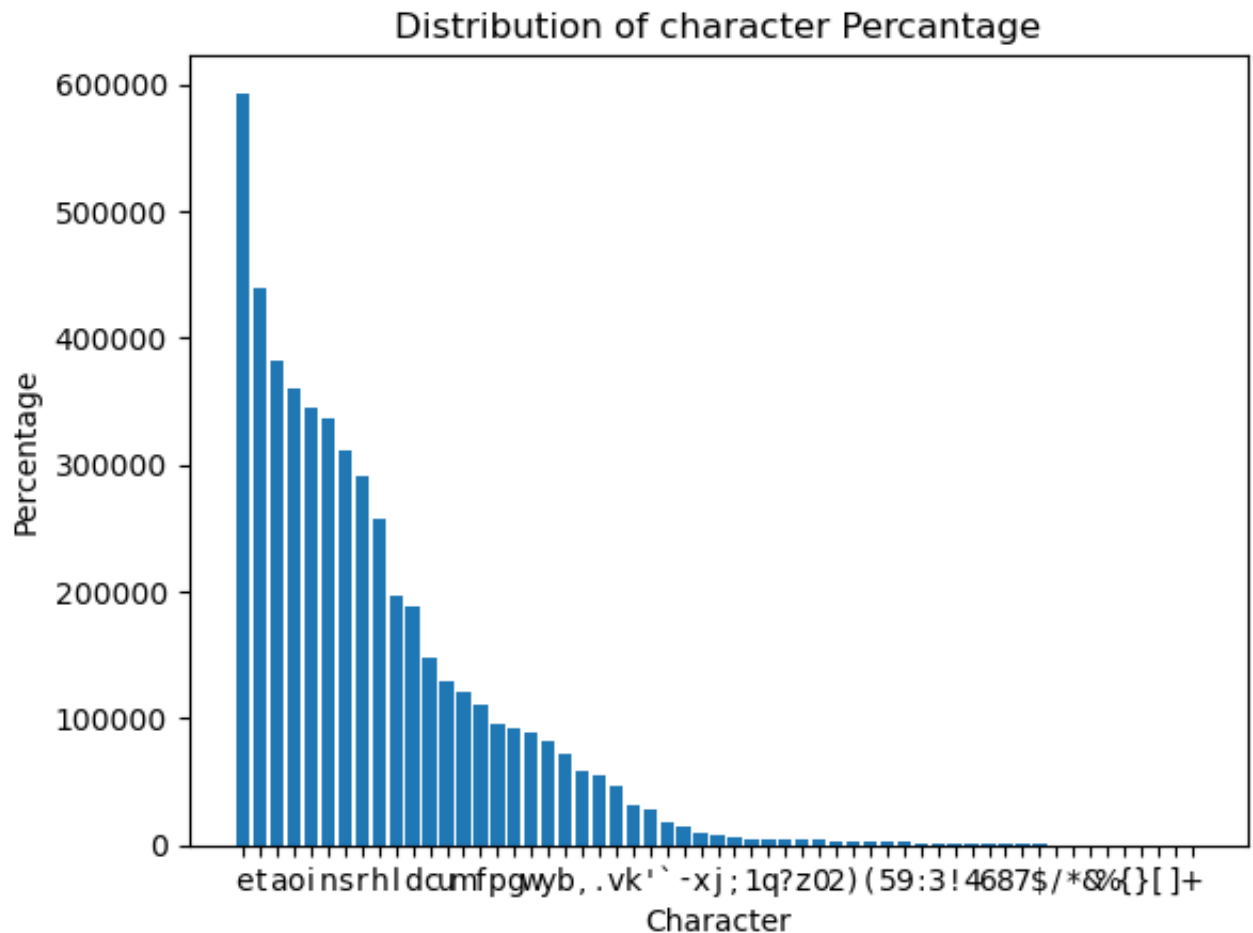
# Hint: Read about the Python function lower() and Counter from the Collect

# For a sample of what we expect, see the very bottom of this notebook.

# Your code here
from collections import Counter
lowered_strings = joined_strings.lower()
c=Counter(lowered_strings)
def sort(val):
    return val[1]
c_sorted=sorted(c.items(), key=sort, reverse=True);chars=[val[0] for val in
plt.bar(chars,occ)
plt.xlabel('Character')
plt.ylabel('Percentage')
plt.title('Distribution of character Percentage')
```

```
Out[6]: Text(0.5, 1.0, 'Distribution of character Percentage')
```





## Problem Two (50 points): Words

Next we will explore the Brown corpus at the level of words. A "word" in this problem will be case-insensitive, so you will create a list of the brown words in lower case and use it throughout the problem.

Each part is worth 10 points.

```
In [7]: # Part A

# Print out the number of occurrences of words, and the number of unique words
# case-insensitive (of course, this will only make a difference in the number of words)
# Print the answer out in human-readable form. Always make it easy for the reader

# Hint: First create a list of lower-case words, and use it throughout this problem

# Your code here

words=' '.join(brown.words()).lower().split()

unq_words= Counter(words) #creating a dict here

print(f"There are {len(words)} words in total")
print(f"There are {len(unq_words)} Unique words ")
```

There are 1161192 words in total  
There are 49815 Unique words

**NOTE: An example:**

```
text: [ "hi", "there", "hi" ]    There are 3 occurrences of
words, but only 2 unique words: { "hi", "there" }
```

Again, we will NOT be using the list of unique words in the rest of this problem; whenever 'words' are mentioned, we mean occurrences of words, as in the first part of Part A.

```
In [8]: # Part B

# Print out the length of the longest word(s), and all occurrences of words
# Print each of the words on a separate line, preceded by a tab '\t'.
# (There may be only one, and it may not look familiar -- just use the data)

# Your code here

long=""
long_words_count=0
for i in words:
    if len(i)>len(long):
        long=i
for i in words:
    if len(i)==len(long):
        long_words_count +=1
print(f"The longest word is: {long}")
print(f"There is only {long_words_count} word/s")
print(f"Length of Longest word :{len(long)}")
```

The longest word is: nnuolapertar-it-vuh-karti-birifw-  
 There is only 1 word/s  
 Length of Longest word :33

In [9]: *# Part C*

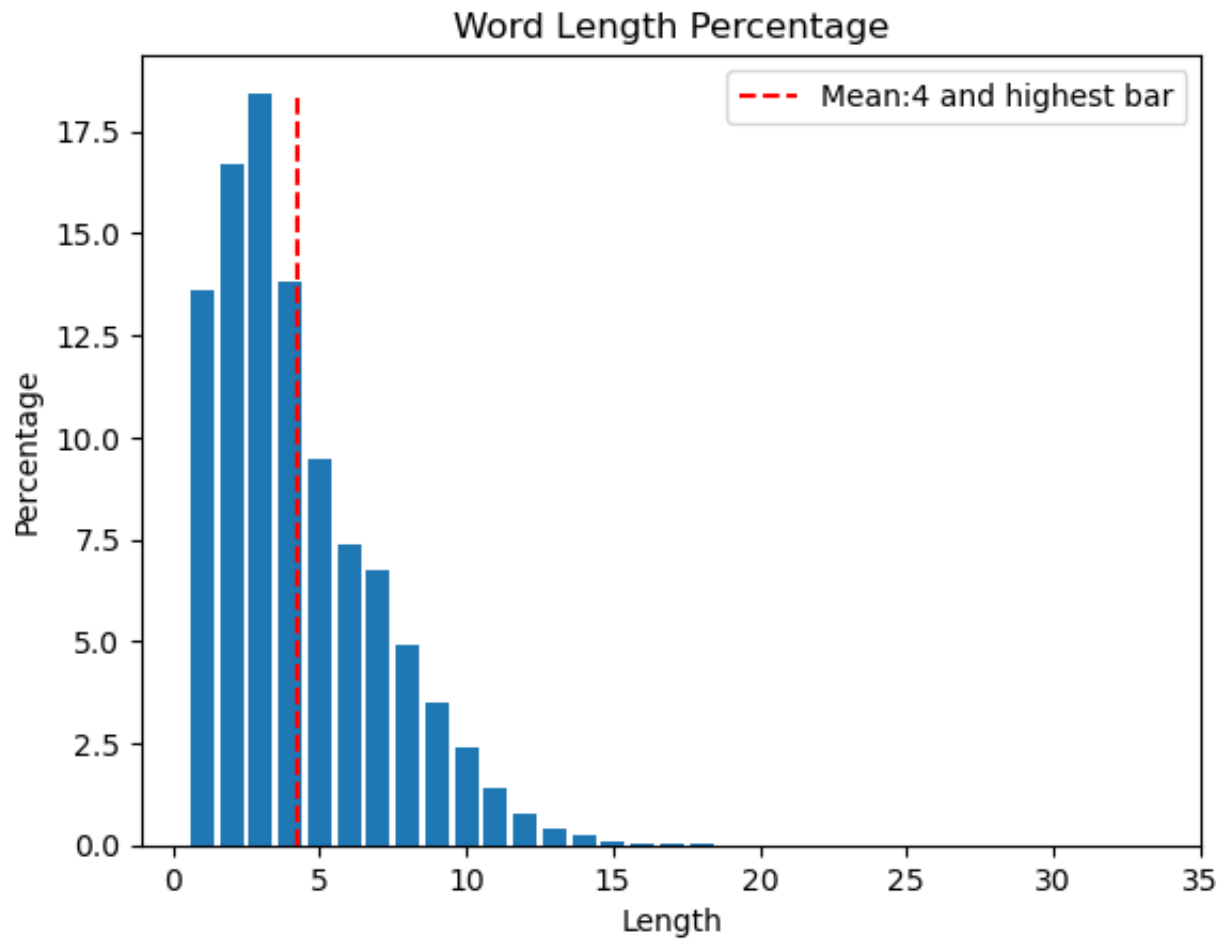
```
# Display a bar chart of the percentages of word lengths of all occurrences
# and give the average length of a word. Draw a dotted red vertical line who
# of the highest bar, and whose x position is the average word length; give
# what the bar means (see PythonRefresher, as usual, for examples of how to

# Print out a human-readable statement about the average word length (to 4 c

# For a sample of what we expect, see the very bottom of this notebook.

# Your code here
import math
words_len_dic = {}
for j in words:
    i = len(j)
    words_len_dic[i] = words_len_dic.get(i, 0) + 1
for length, count in words_len_dic.items():
    words_len_dic[length] = (count/len(words)*50*2)
mean=sum(len(word) for word in words)/len(words)
print(f"Average word Count is {(mean):.4f} ")
plt.bar(words_len_dic.keys(), words_len_dic.values())
plt.plot([sum(len(word) for word in words)/len(words), sum(len(word) for wor
plt.xlabel('Length')
plt.ylabel('Percentage')
plt.title('Word Length Percentage')
plt.legend()
plt.show()
```

Average word Count is 4.2765



```
In [10]: # Part D

# Now we will consider word frequencies (expressed as percentages). To simplify
# allow "normal" words, i.e., those consisting of only lower-case letters, words
# periods, and dashes. Since this involves regular expressions (we'll cover this
# this code is provided.

# Your task in this problem is to give the twenty most common normal words,
# all occurrences of normal words these represent. Be sure to give your answer.

# Hint: this is very similar to Part D in the previous problem.

import re
from operator import itemgetter

p = re.compile('[a-zA-Z\'.`-]+$')      # allow intra-word punctuation
q = re.compile('[\'''.`-]+$')

def is_normal_word(w):
    return (p.match(w) and not q.match(w))

# Your code here
normal_count=Counter(i for i in words if is_normal_word(i));per = [(word, count) for word, count in normal_count.items()]
per.sort(key=itemgetter(1), reverse=True) ;print("Twenty Most Common Normal Words are:")
for i, percent in per[:20]:
    print(f"{i}: {percent:.2f}%") #displaying 2 decimal points
```

Twenty Most Common Normal Words are:

```
the: 6.97%
of: 3.63%
and: 2.87%
to: 2.61%
a: 2.31%
in: 2.13%
that: 1.06%
is: 1.01%
was: 0.98%
he: 0.95%
for: 0.95%
it: 0.87%
with: 0.73%
as: 0.72%
his: 0.70%
on: 0.67%
be: 0.64%
at: 0.54%
by: 0.53%
i: 0.51%
```

```
In [11]: # Part E

# Now give the distribution of the percentages of normal word occurrences, i
# just as you did for Problem One, Part E, but now for words.

# You may give this as a bar chart, but it is more readable as a plot (i.e.,

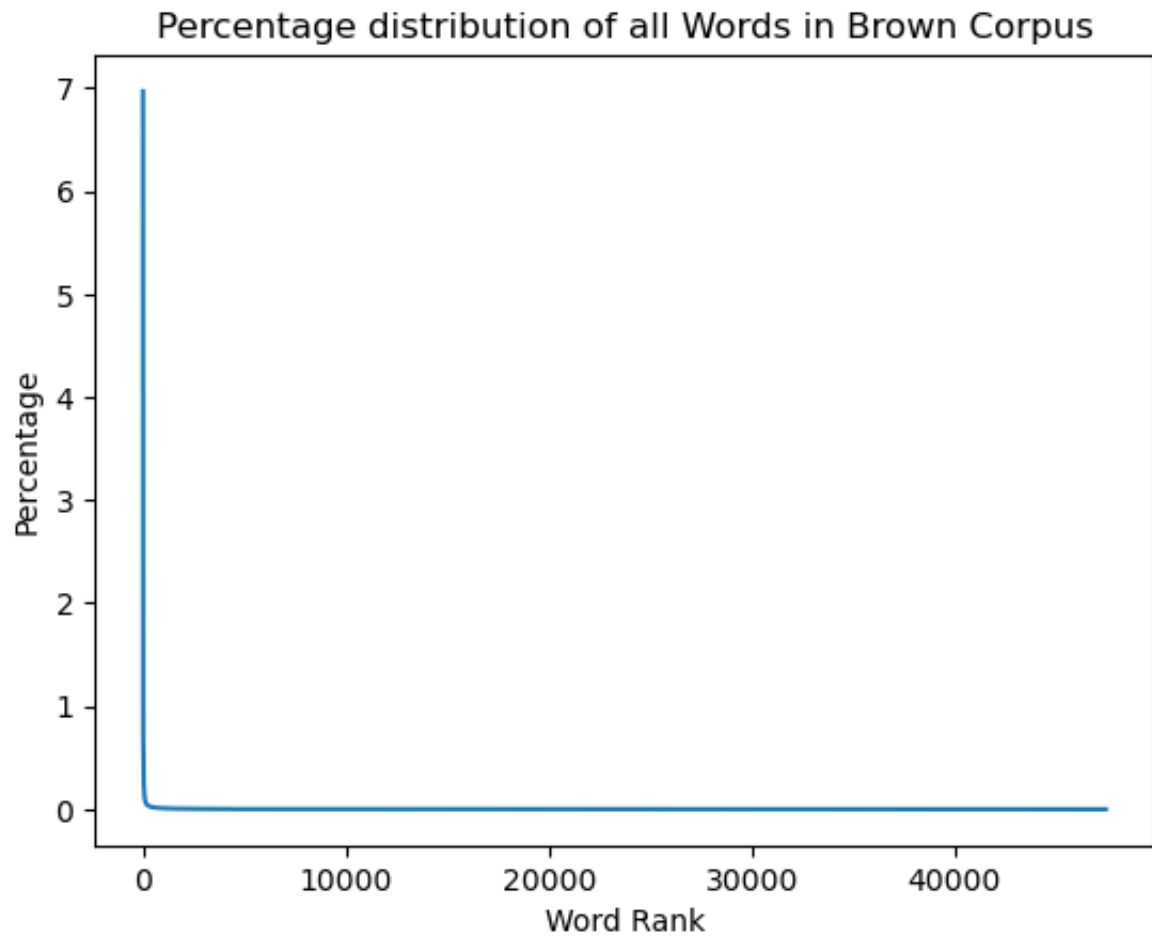
# Show this for all normal words, then for the 100 most common normal words,
# common normal words.

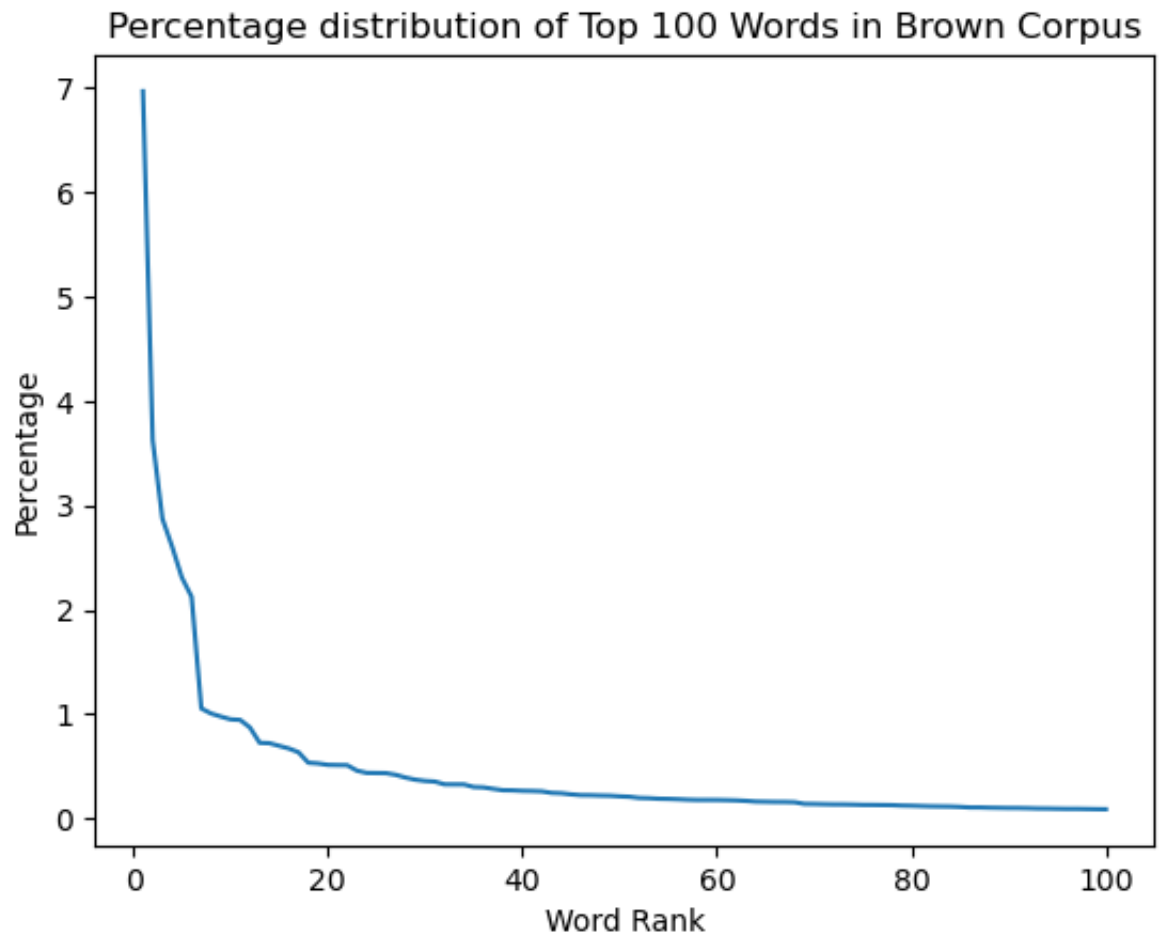
# For a sample of what we expect, see the very bottom of this notebook.

# Your code here
plt.plot(range(1, len(per) + 1), [i for _, i in per], linestyle='-')
plt.xlabel('Word Rank');plt.ylabel('Percentage')
plt.title('Percentage distribution of all Words in Brown Corpus')
plt.show()

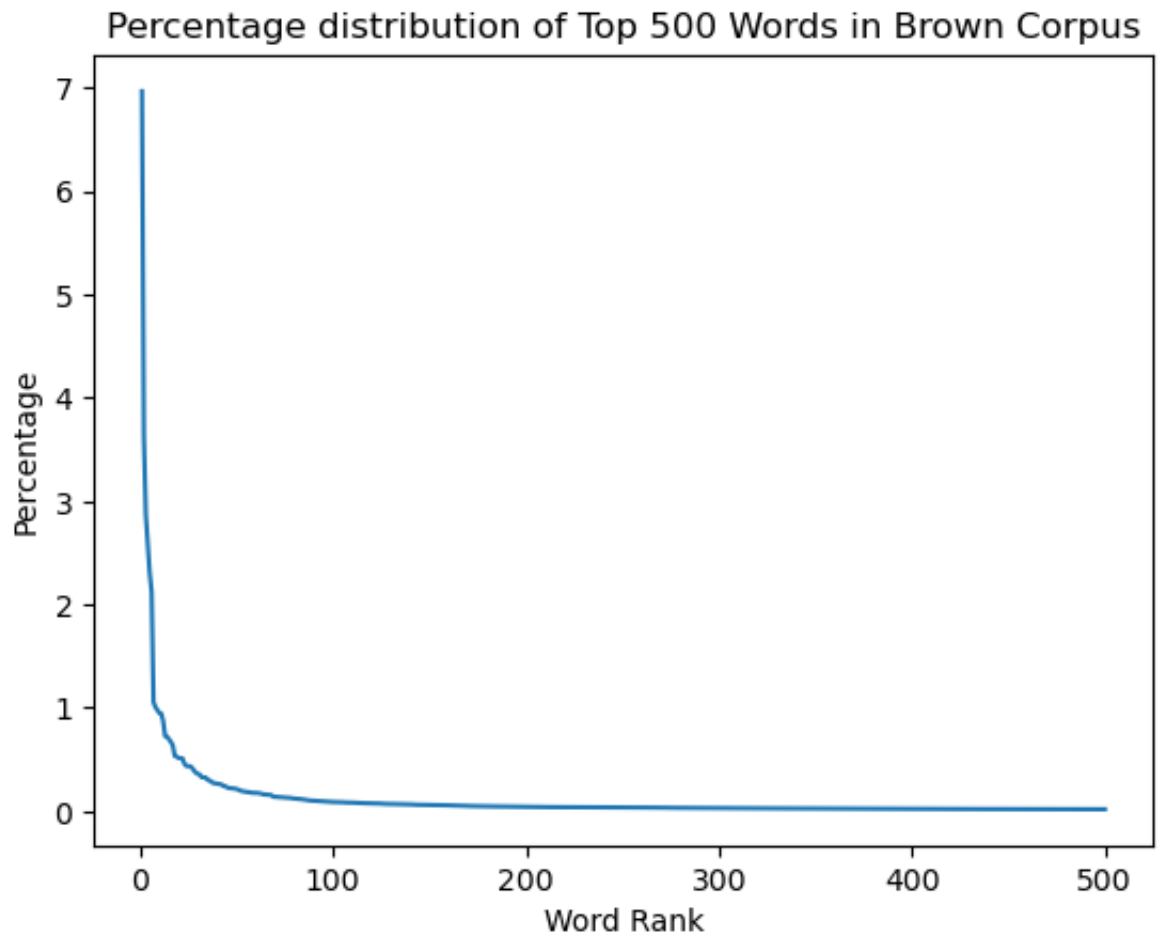
#top 100
plt.plot(range(1, 100+ 1), [i for _, i in per[:100]], linestyle='-')
plt.xlabel('Word Rank');plt.ylabel('Percentage')
plt.title('Percentage distribution of Top 100 Words in Brown Corpus')
plt.show()

#top 500
plt.plot(range(1, 500+ 1), [i for _, i in per[:500]], linestyle='-')
plt.xlabel('Word Rank');plt.ylabel('Percentage')
plt.title('Percentage distribution of Top 500 Words in Brown Corpus')
plt.show()
```









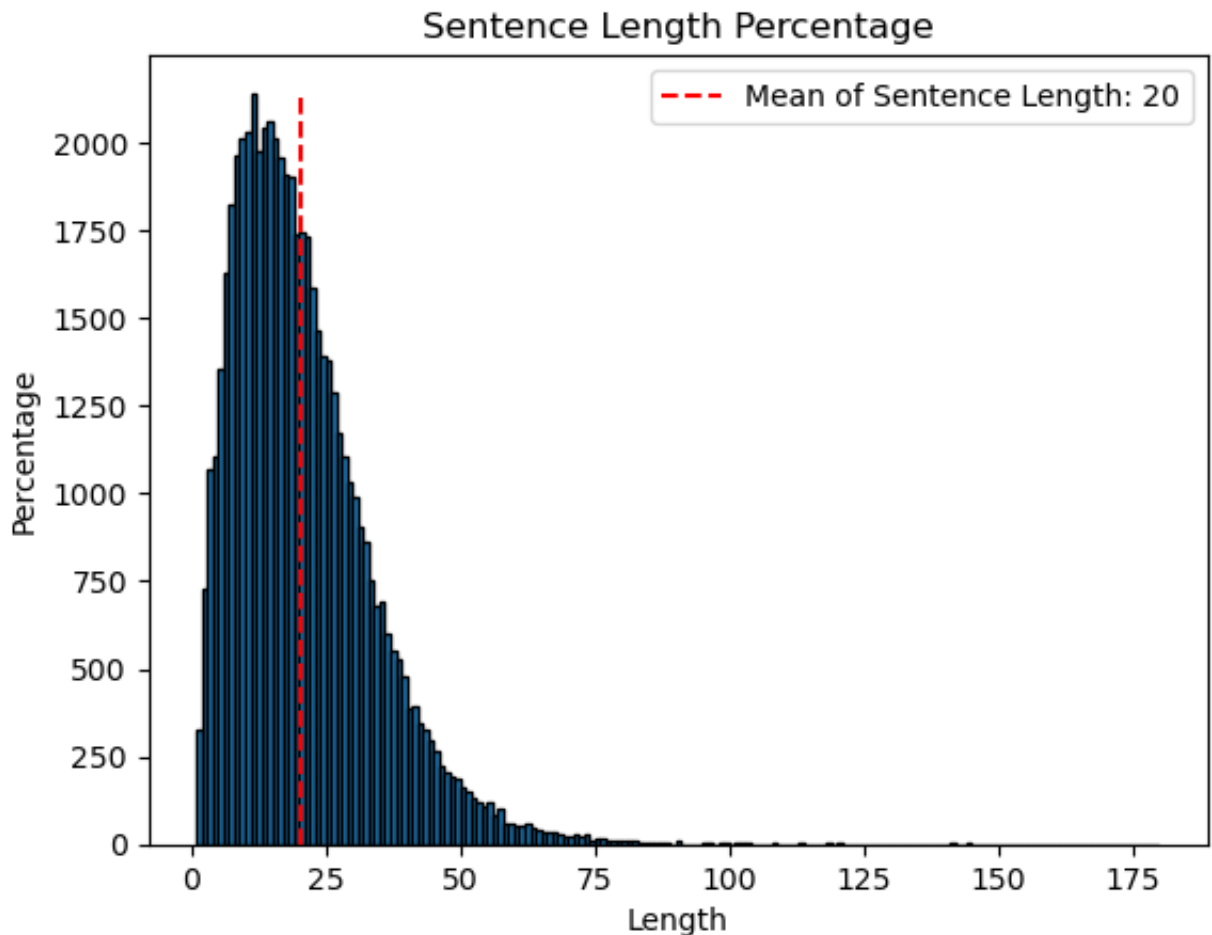
### Problem Three (5 points): Paragraphs

Ok, one more, just for fun! Produce a histogram of the length of all sentences, with the average length indicated, similar to what you did for Problem 2, Part C. Again, just consider a sentence to be anything in `brown.sents()`.

Hint: You should be able to cut and paste your solution from 2.C and just change a few things.

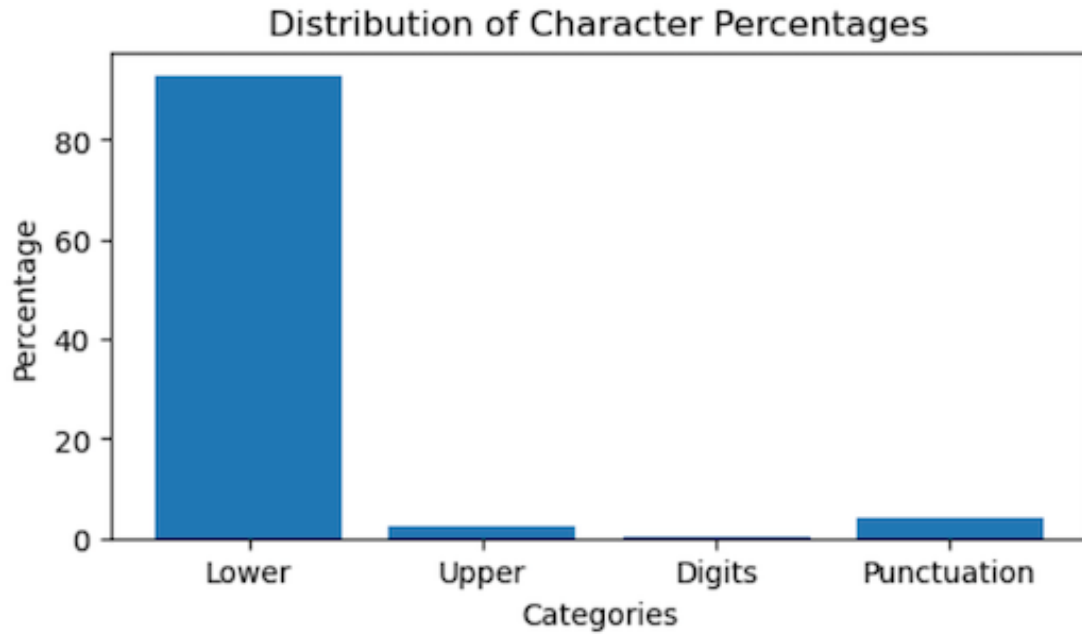
```
In [12]: # Your code here
import matplotlib.pyplot as plt
sent_len = [len(i) for i in brown.sents()]
mean_sent=sum(sent_len)/len(sent_len)
n, bins, patches = plt.hist(sent_len, bins=range(min(sent_len), max(sent_len)
print(f"Average sentence Length is {(mean_sent):.4f} ")
plt.axvline(x=mean_sent, color='r', linestyle='--',
            label=f'Mean of Sentence Length: {round(mean_sent)}', ymax=max(n
plt.xlabel('Length')
plt.ylabel('Percentage')
plt.title('Sentence Length Percentage')
plt.legend()
plt.show()
```

Average sentence Length is 20.2510

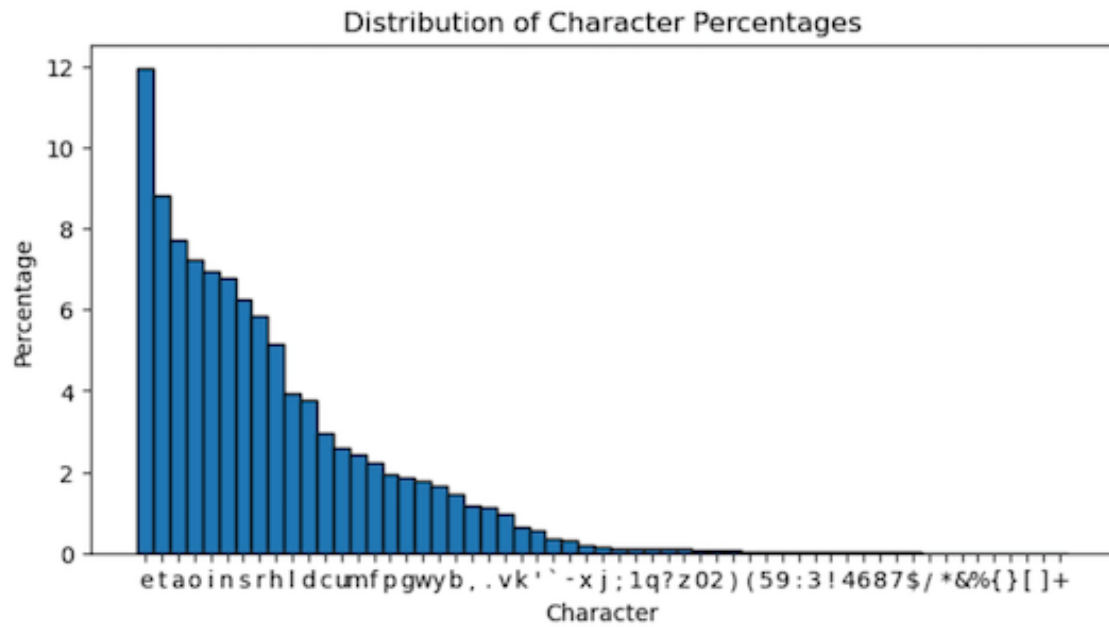


## Sample Outputs for the Bar Charts

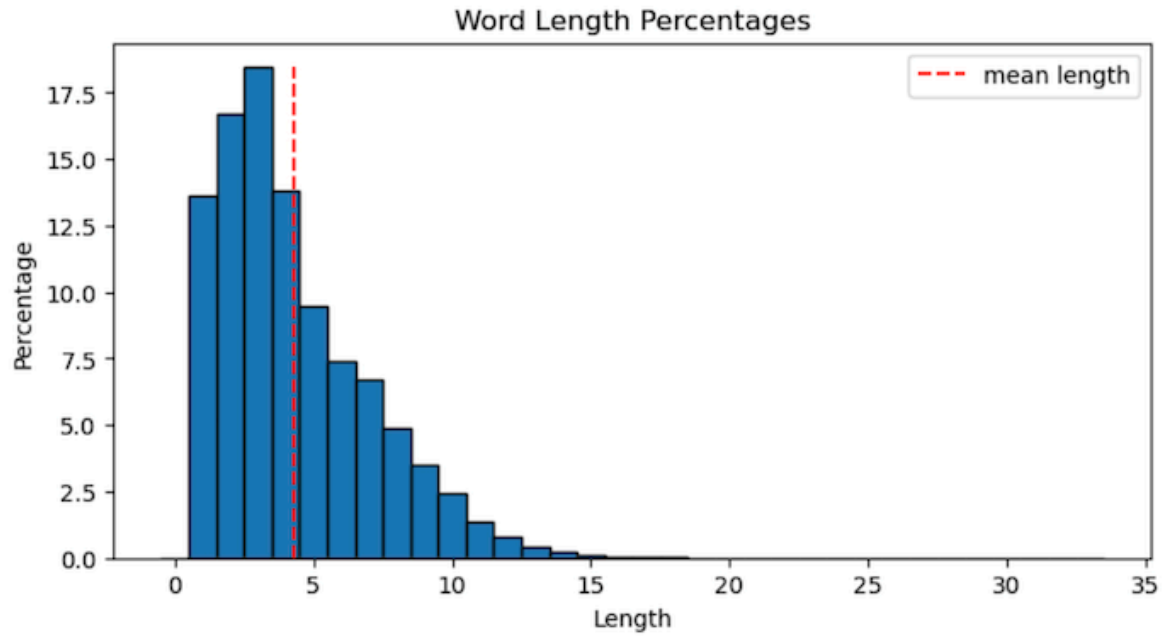
## Problem 1.C



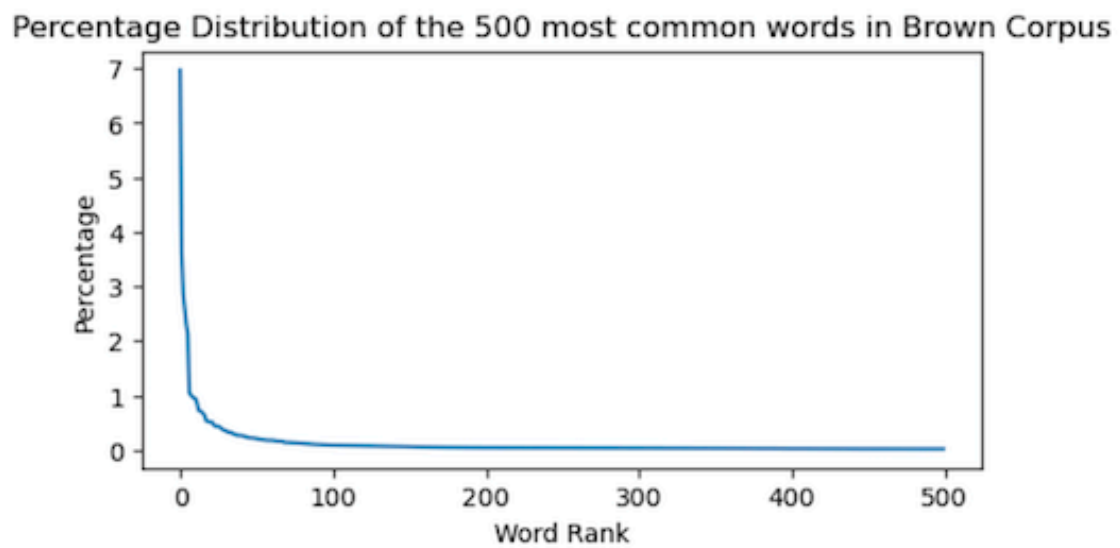
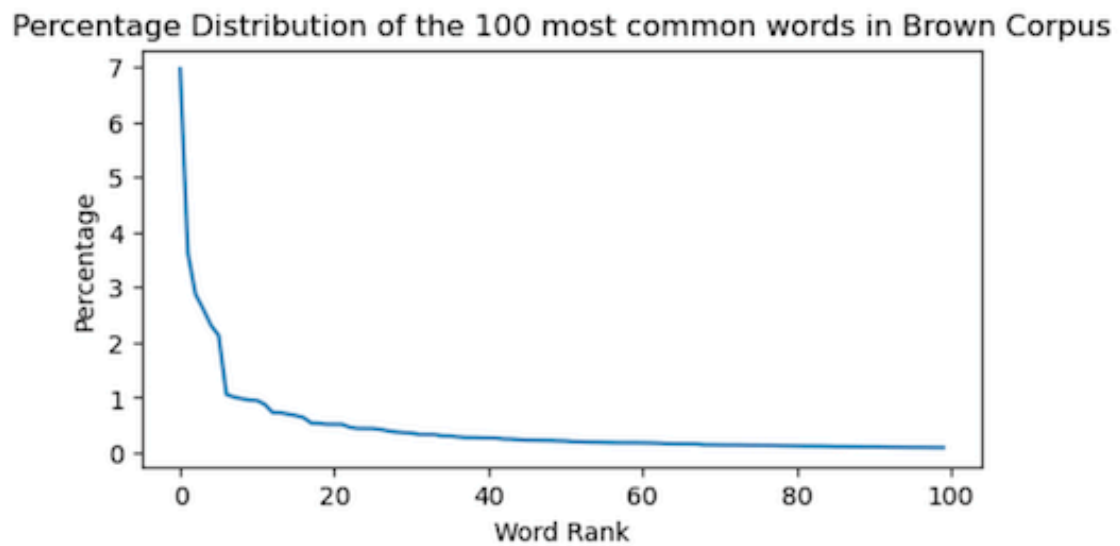
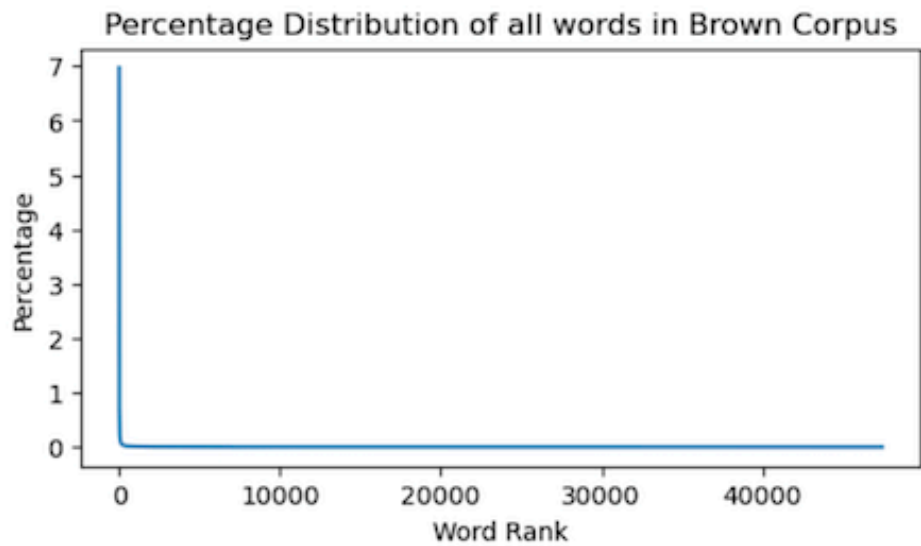
## Problem 1.D



## Problem 2.C



## Problem 2.D



## Problem 3

