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.

```
<Your answer here>

1. I discussed the homework with Junhui Cho. We discussed the
last question in particular where we believe we aer plotting
the bar graph instead of the histogram as the question stated
and the sections D and E on question 2 where we only had to
use normal words in this section instead of using all of the
words in Brown Corpus.

2. I used https://www.geeksforgeeks.org/python-string-join-
method/ to review the join function in python.

   I also used https://www.geeksforgeeks.org/formatted-
string-literals-f-strings-
python/#:~:text=To%20create%20an%20f%2Dstring,inside%20string%20lit
 to review how to use f-string in python while printing.

   I also used https://www.geeksforgeeks.org/python-counter-
objects-elements/ to learn about counter in python.
```

I also used https://docs.python.org/3/library/string.html to learn how to use ascii lower, upper, digit, and punctuations.

I used the file on the course website (the CS 505 tutorials directory's Python Refresher) to review how to draw dotted lines on the graph when plotting the average number of words and sentences.

I looked at the guide video in your Youtube channel to review organize the graphs (including the width, edgecolor, and etc.)

3. I used ChatGPT to learn how to sort the characters in descending order according to their percentage.

I also used ChatGPT to find the maximum length of word in Section B part 2.

I also used ChatGPT to make dictionaries in the format to do list comprehension.

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 [177…

```
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 /Users/chrisyang/nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

Out[177]:

```
True
```

In [178…

```
from nltk.corpus import brown

# We can access various components of this multi-text corpus: words, sentenc
# 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', 'inves
tigation', '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', 'ove
r-all', 'charge', 'of', 'the', 'election', ',', '``', 'deserves', 'the', 'pr
aise', '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'), ('inv
estigation', '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 [179…

```
# 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

brownWords = brown.words()
joined = "".join(brownWords)

totalCount = len(joined)

print(f"There are {totalCount} occurrences of characters in the Brown corpus
```

```
There are 4965882 occurrences of characters in the Brown corpus.
```

In [180…

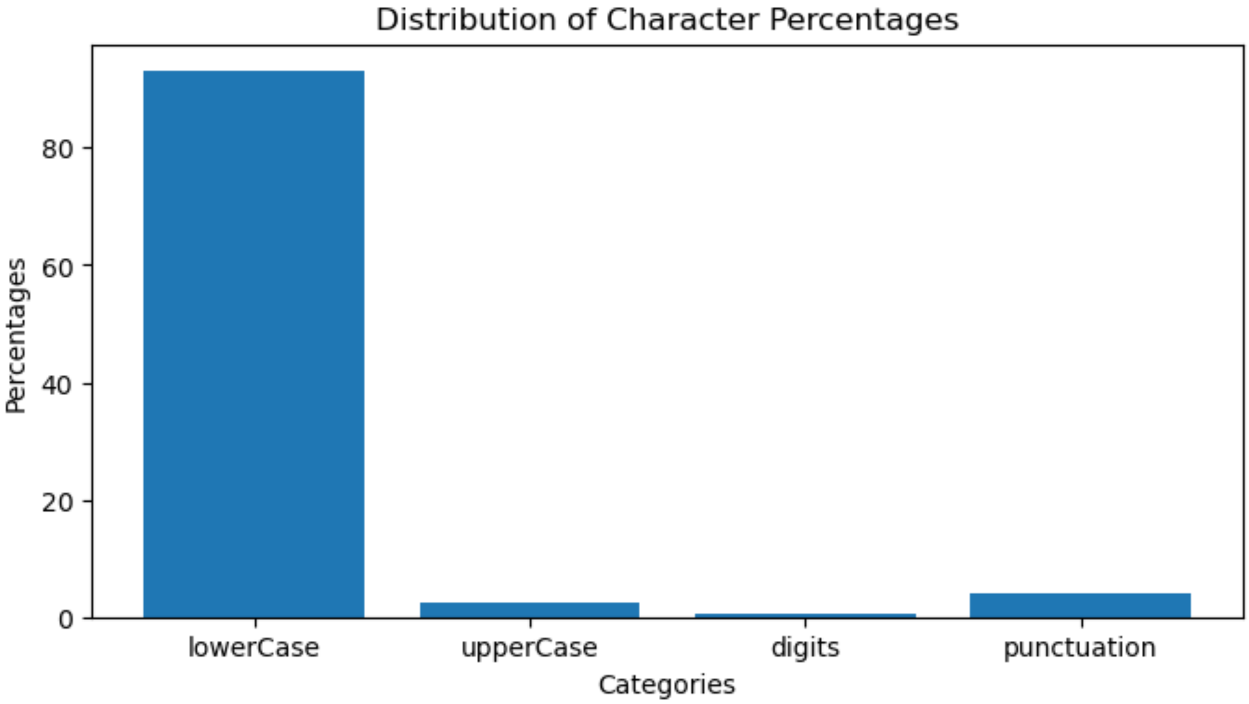
```
# Part B

# 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

uniqueChar = len(set(joined))
print(f"There are {uniqueChar} unique characters in the Brown corpus.")


sortedUniqueChar = ''.join(sorted(set(joined)))
print(f"The string consisting of all the characters in the Brown corpus, sor
```

```
There are 83 unique characters in the Brown corpus.
The string consisting of all the characters in the Brown corpus, sorted in o
rder is
!$%&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZ[]`abcdefghijklmnopqrstu
vwxyz{}
```

NOTE: 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.

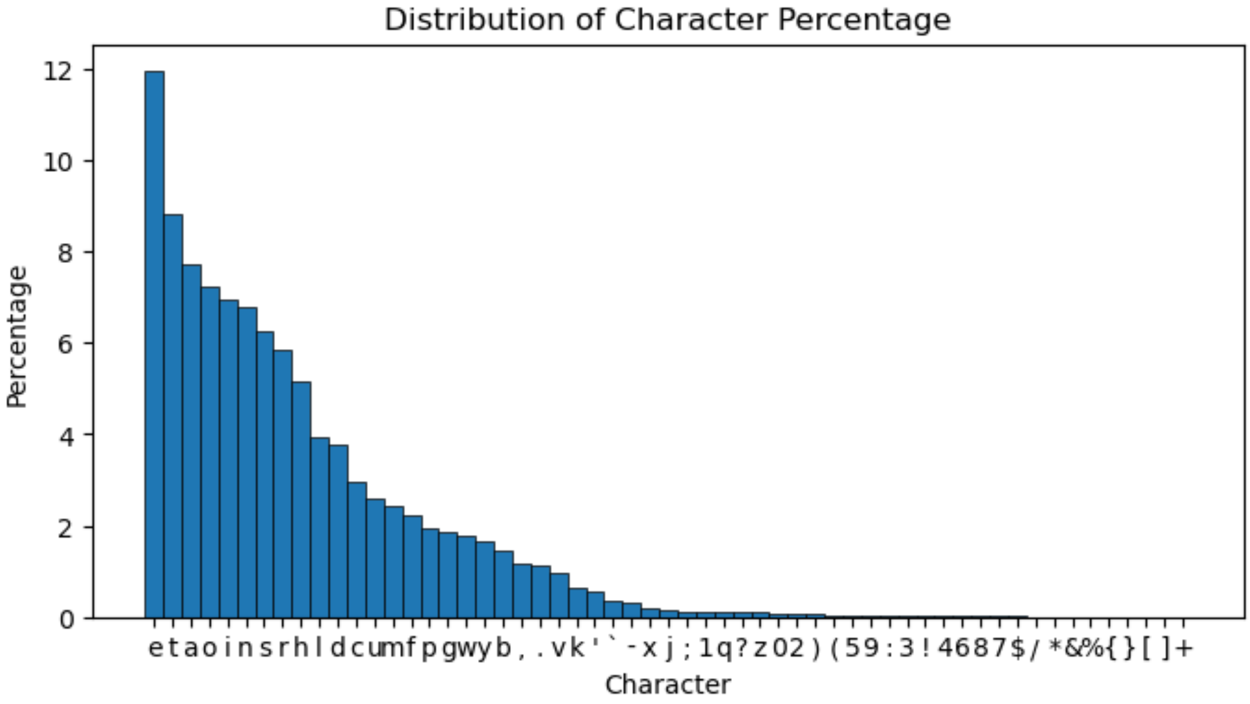
In [181…

```
# 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 string
import matplotlib.pyplot as plt




lowerCase = sum(1 for x in joined if x in string.ascii_lowercase)/totalCount
upperCase = sum(1 for x in joined if x in string.ascii_uppercase)/totalCount
digits = sum(1 for x in joined if x in string.digits)/totalCount * 100
punctuation = sum(1 for x in joined if x in string.punctuation)/totalCount *



categories = ["lowerCase", "upperCase", "digits", "punctuation"]
percentages = [lowerCase, upperCase, digits, punctuation]

plt.figure(figsize=(8, 4))
plt.bar(categories, percentages)


plt.xlabel('Categories')
plt.ylabel('Percentages')
plt.title('Distribution of Character Percentages')


plt.show()
```


Distribution of Character Percentages
Percentages
80
60
40
20
0
lowerCase
upperCase
digits
punctuation
Categories

In [182…

```
# 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

lowerCaseForm = joined.lower()
charCount = Counter(lowerCaseForm)

charPercentages = {char: (count / totalCount * 100) for char, count in charC

sortedChars = sorted(charPercentages.items(), key=lambda percentage: percent

char = [x[0] for x in sortedChars]
percent = [x[1] for x in sortedChars]


plt.figure(figsize=(8, 4))
plt.bar(char, percent,edgecolor='black', linewidth=0.5,width = 1)

plt.xlabel('Character')
plt.ylabel('Percentage')
plt.title('Distribution of Character Percentage')

plt.show()
```



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 [183…

```
# Part A

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

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

# Your code here
lowerCaseWordsLst = [x.lower() for x in brownWords]
print(f"There are {len(lowerCaseWordsLst)} occurrences of words in the Brown

wordCount = Counter(lowerCaseWordsLst)
print(f"There are {len(wordCount)} occurrences of unique words in the Brown
```

```
There are 1161192 occurrences of words in the Brown corpus.
There are 49815 occurrences of unique words in the Brown corpus.
```

NOTE: 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.

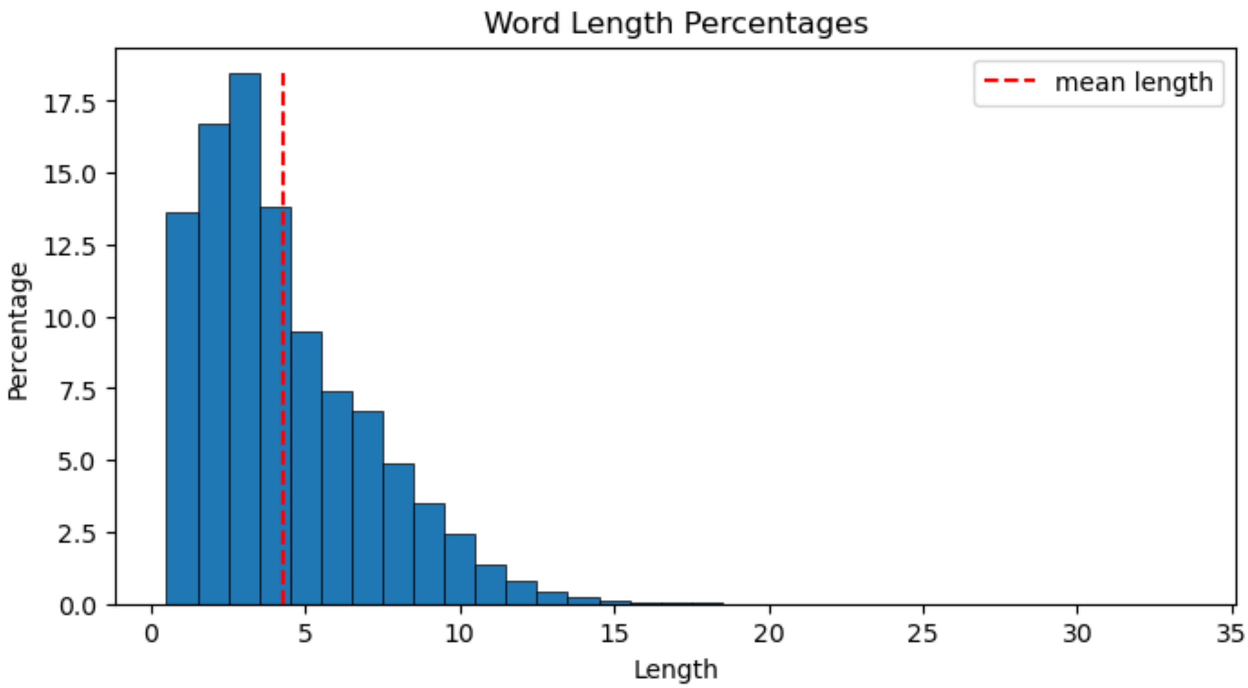
In [184...

```
# 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, preceeded by a tab '\t'.
# (There may be only one, and it may not look familiar -- just use the data

# Your code here

longestWordLength = len(max(lowerCaseWordsLst, key=len))

longLst = [x for x in lowerCaseWordsLst if len(x) == longestWordLength]
numberOfAppearances = len(longLst)

uniqueMaxWord = set(longLst)

print(f"The length of the longest word in Brown corpus is {longestWordLength
print("The word(s) that has maximum length: ")


for x in uniqueMaxWord:
    print(f"\t{x}")
```

```
The length of the longest word in Brown corpus is 33 and the number of occur
rence(s) of words of the maximum length
is 1.
The word(s) that has maximum length:
        nnuolapertar-it-vuh-karti-birifw-
```

In [185…

```
# 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 d

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

# Your code here

wordLength = [len(word) for word in lowerCaseWordsLst]

averageWordLength = sum(wordLength) / len(wordLength)

wordCount = Counter(wordLength)

wordLengthPercentages = {length: (count / len(wordLength) * 100) for length,

sortWordLength = sorted(wordLengthPercentages.items())


wordLen = [x[0] for x in sortWordLength]
percentWord = [x[1] for x in sortWordLength]


plt.figure(figsize=(8, 4))
plt.bar(wordLen, percentWord, edgecolor='black', linewidth=0.5,width = 1)


plt.xlabel('Length')
plt.ylabel('Percentage')
plt.title('Word Length Percentages')

plt.plot([averageWordLength,averageWordLength], [0, max(percentWord)], color
plt.legend()
plt.show()

print()
print(f"The average word length is {round(averageWordLength,4)}.")
```



```
The average word length is 4.2765.
```

In [186…

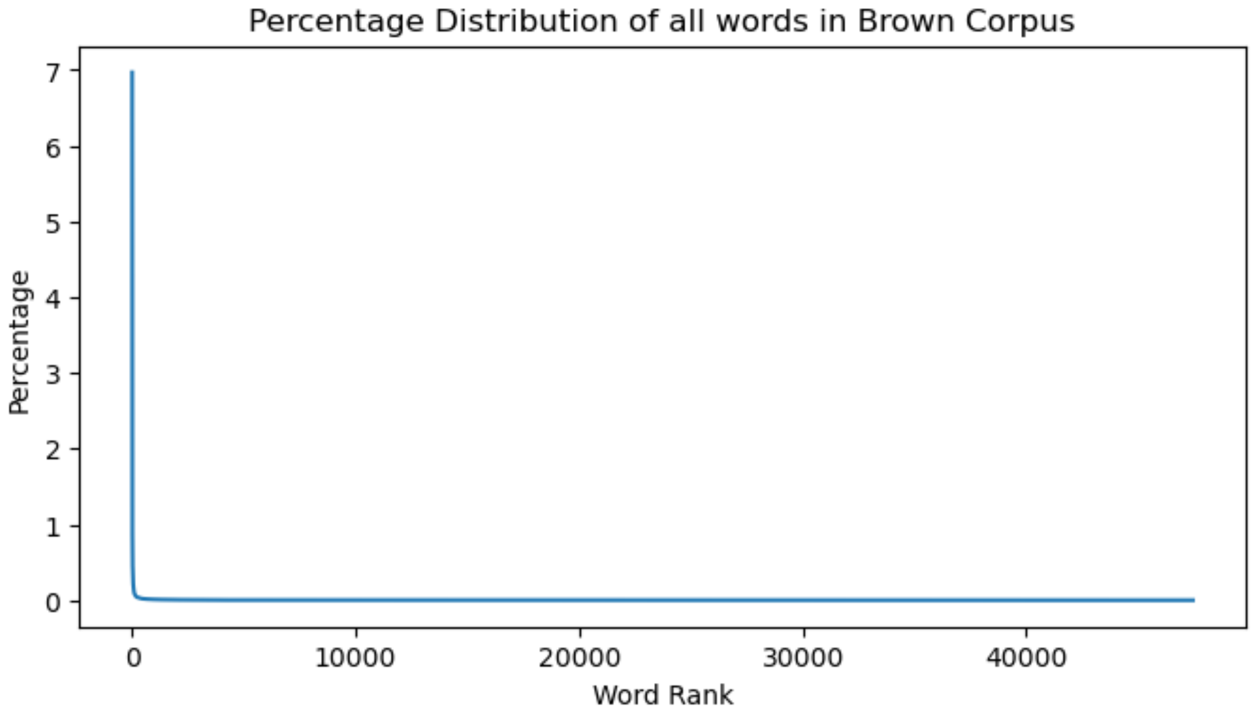
```
# Part D

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

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

import re

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))

commonWords = {}
countOfCommon = 0

for x in lowerCaseWordsLst:
    if is_normal_word(x):
        countOfCommon += 1
        if x in commonWords:
            commonWords[x] += 1
        else:
            commonWords[x] = 1
```

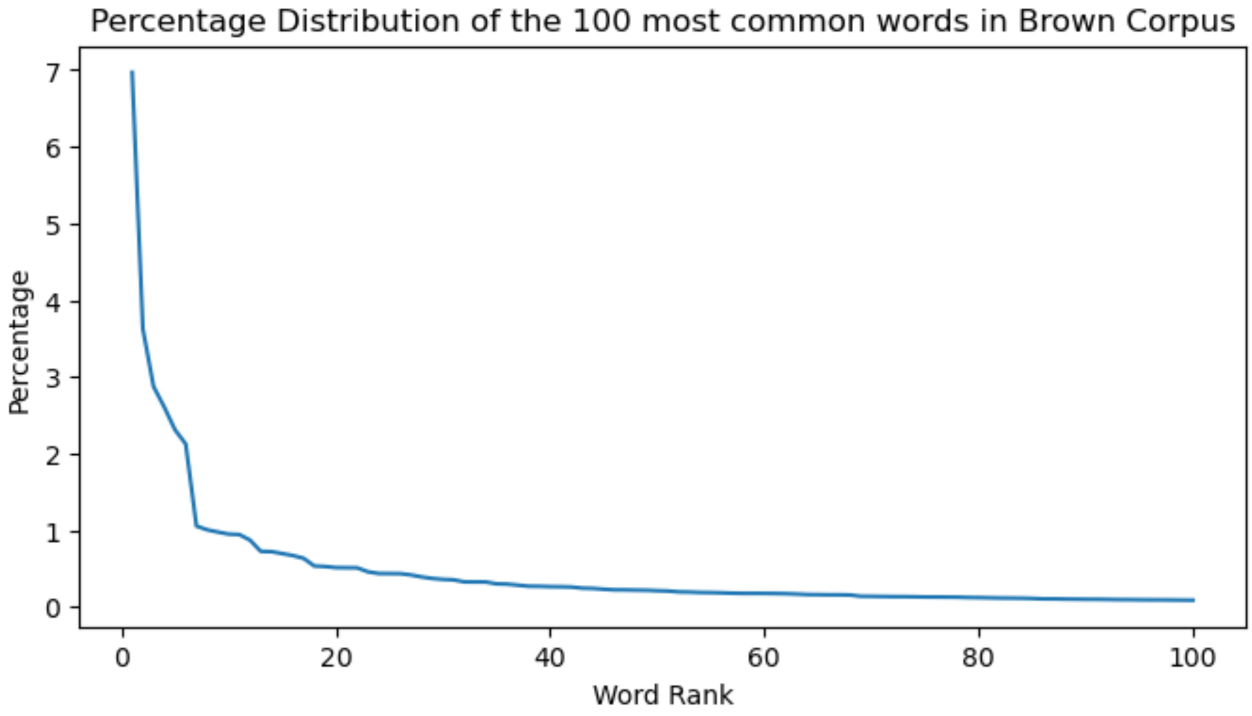
```
commonWords = {x: (commonWords[x] / countOfCommon * 100) for x in commonWord


sortedNormalWordsLst = sorted(commonWords.items(), key = lambda percentage:
sortedNormalWords = dict(sorted(commonWords.items(), key=lambda percentage:

numberToPrint = 20
count = 0
percentOfAll = 0

print(f"Below are the {numberToPrint} most frequent words and their correspo
for word, percentage in sortedNormalWords.items():
    print(f"Word: {word} \t Percentage: {round(percentage,2)}%")
    percentOfAll += percentage
    count += 1
    if count == 20:
        break

print()
print(f"The percentage of all occurrences of the twenty most common normal w
```

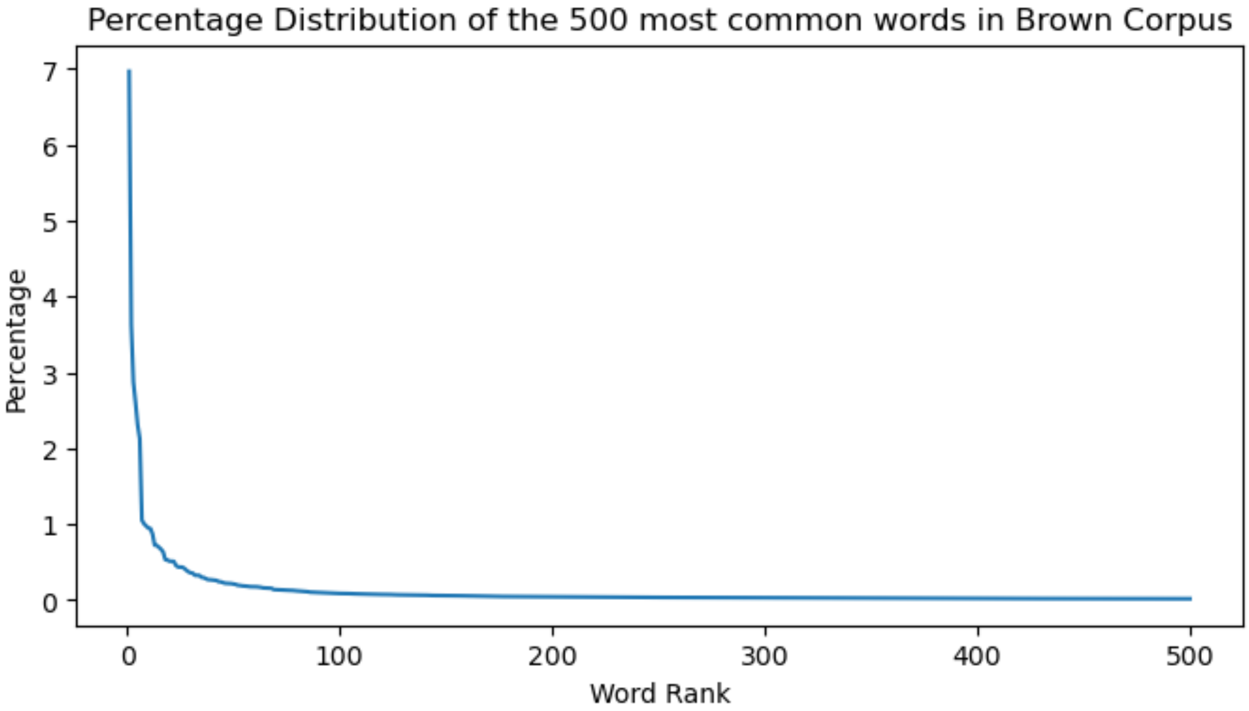
```
Below are the 20 most frequent words and their corresponding percentages in
Brown corpus in decreasing order.

Word: the        Percentage: 6.97%
Word: of         Percentage: 3.63%
Word: and        Percentage: 2.87%
Word: to         Percentage: 2.61%
Word: a          Percentage: 2.31%
Word: in         Percentage: 2.13%
Word: that       Percentage: 1.06%
Word: is         Percentage: 1.01%
Word: was        Percentage: 0.98%
Word: he         Percentage: 0.95%
Word: for        Percentage: 0.95%
Word: it         Percentage: 0.87%
Word: with       Percentage: 0.73%
Word: as         Percentage: 0.72%
Word: his        Percentage: 0.7%
Word: on         Percentage: 0.67%
Word: be         Percentage: 0.64%
Word: at         Percentage: 0.54%
Word: by         Percentage: 0.53%
Word: i          Percentage: 0.51%

The percentage of all occurrences of the twenty most common normal words the
se represent are 31.35%.
```

In [187…

```
# 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

normalWords = [x for x in range(1, len(sortedNormalWordsLst)+1)]
percentageNormalWords = [x[1] for x in sortedNormalWordsLst]



plt.figure(figsize=(8, 4))
plt.plot(normalWords, percentageNormalWords)


plt.xlabel('Word Rank')
plt.ylabel('Percentage')
plt.title('Percentage Distribution of all words in Brown Corpus')

plt.show()

stopPointOne = 100

normalWordsTwo = [x for x in range(1, stopPointOne+1)]
percentageNormalWordsTwo = [x[1] for x in sortedNormalWordsLst][:stopPointOn



plt.figure(figsize=(8, 4))
plt.plot(normalWordsTwo, percentageNormalWordsTwo)


plt.xlabel('Word Rank')
plt.ylabel('Percentage')
plt.title('Percentage Distribution of the 100 most common words in Brown Cor

plt.show()


stopPointTwo = 500

normalWordsThree = [x for x in range(1, stopPointTwo+1)]
percentageNormalWordsThree= [x[1] for x in sortedNormalWordsLst][:stopPointT

plt.figure(figsize=(8, 4))
plt.plot(normalWordsThree, percentageNormalWordsThree)
plt.xlabel('Word Rank')
plt.ylabel('Percentage')
plt.title('Percentage Distribution of the 500 most common words in Brown Cor

plt.show()
```







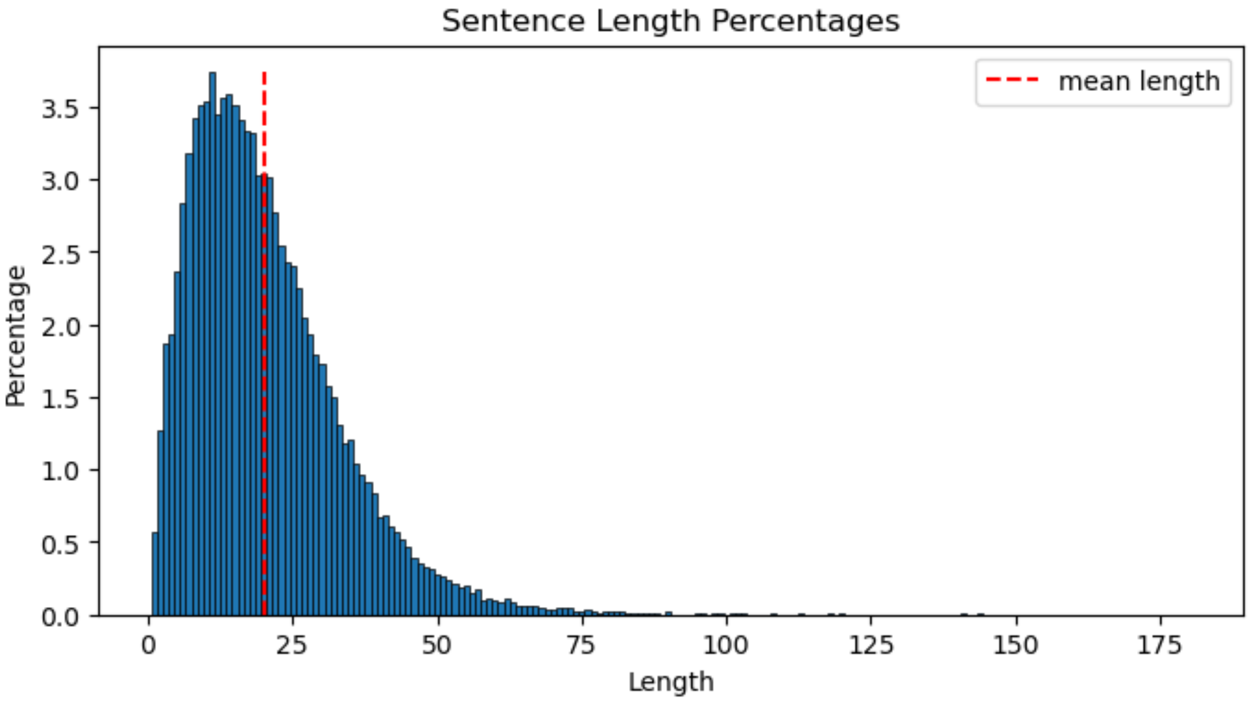
Problem Three (5 points): Paragraphs

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 [188…

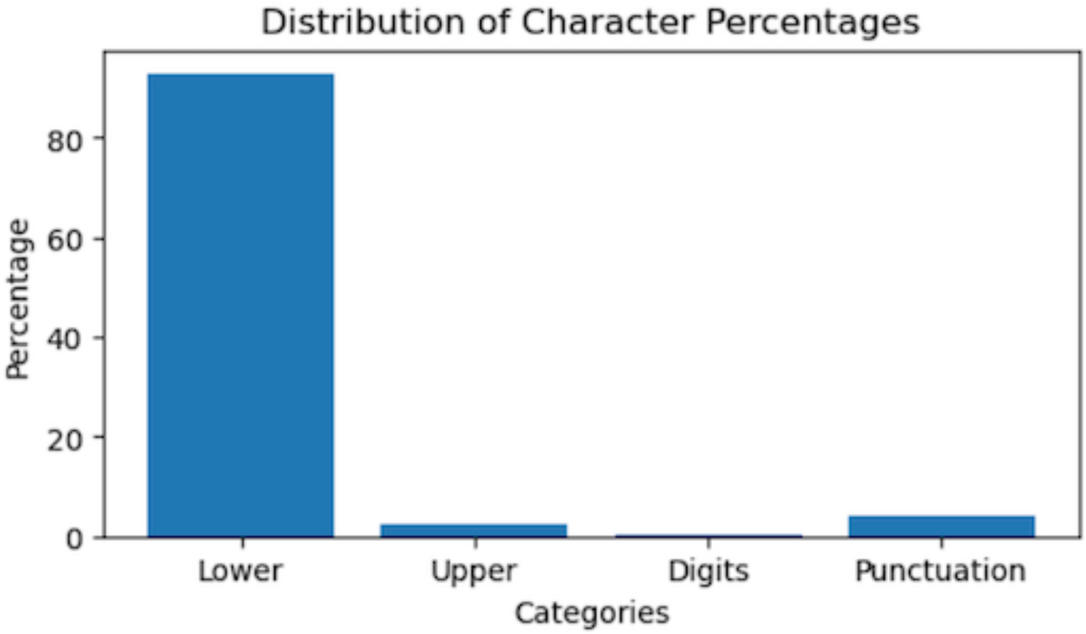
```
# Your code here

sentencesBrown = brown.sents()
sentenceLength = [len(sentence) for sentence in sentencesBrown]

averageSentenceLength = sum(sentenceLength) / len(sentenceLength)

sentenceCount = Counter(sentenceLength)
sentenceLengthPercentages = {length: (count / len(sentencesBrown) * 100) for

sortSentenceLength = sorted(sentenceLengthPercentages.items())

sentenceLen = [x[0] for x in sortSentenceLength]
percentSentence = [x[1] for x in sortSentenceLength]


plt.figure(figsize=(8, 4))
plt.bar(sentenceLen, percentSentence, edgecolor='black', linewidth=0.5,width


plt.xlabel('Length')
plt.ylabel('Percentage')
plt.title('Sentence Length Percentages')

plt.plot([averageSentenceLength,averageSentenceLength], [0, max(percentSente
plt.legend()

plt.show()

print()
print(f"The average sentence length is {round(averageSentenceLength,4)}.")
```



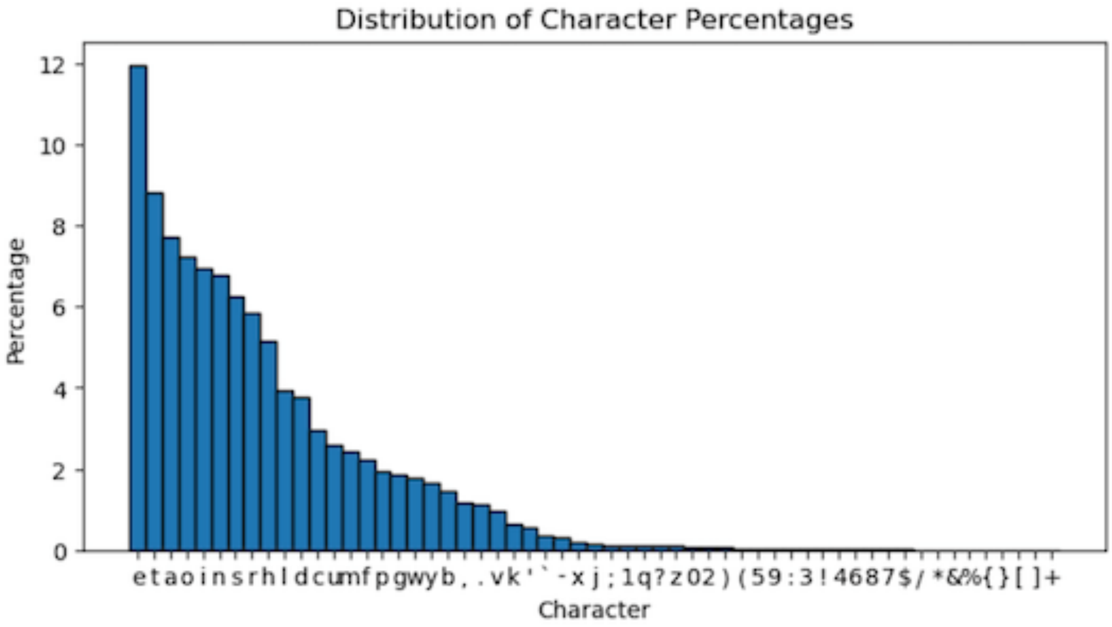
```
The average sentence length is 20.251.
```

Sample Outputs for the Bar Charts

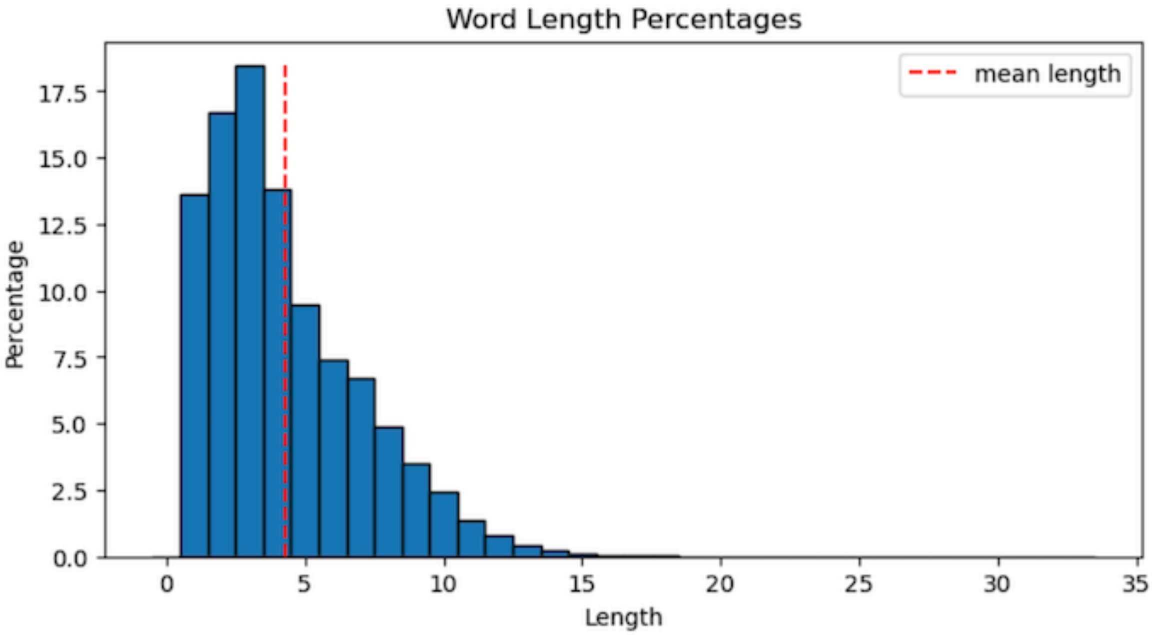
Problem 1.C



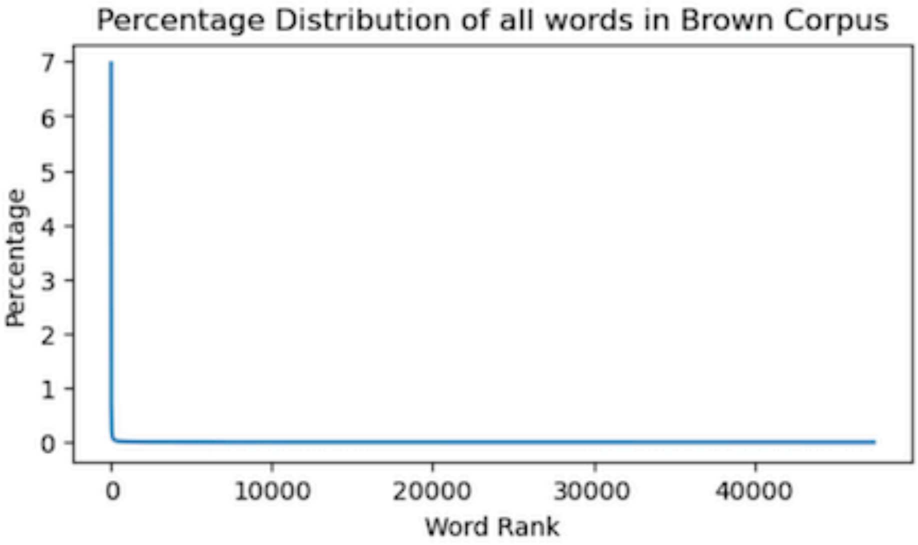
Problem 1.D



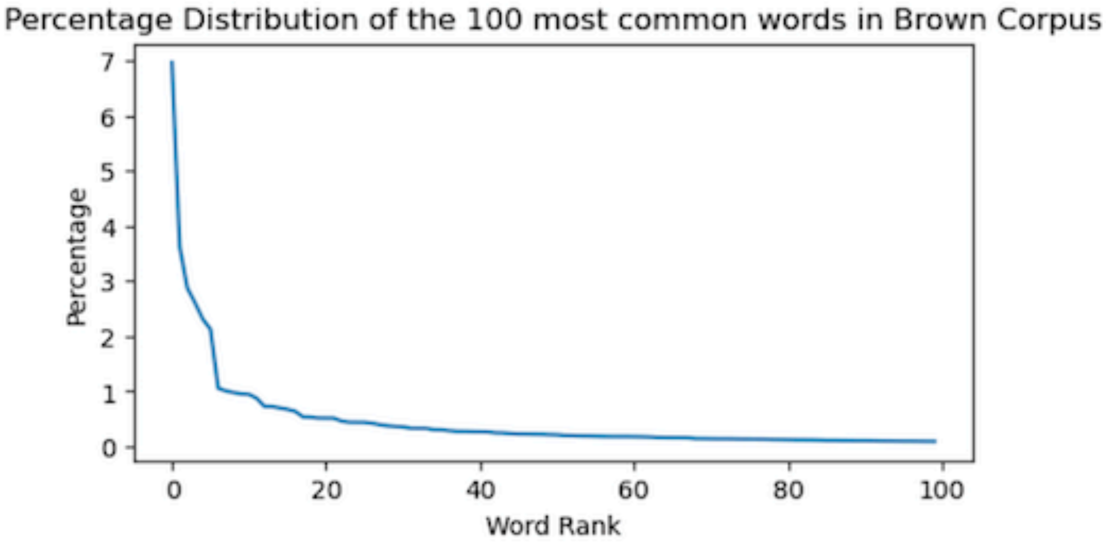
Problem 2.C



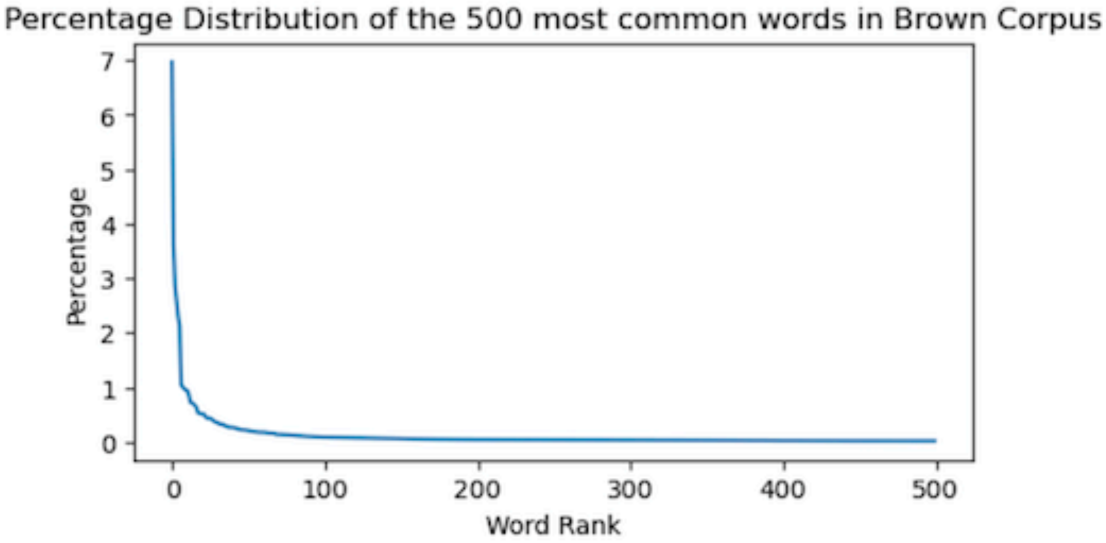
Problem 2.D



Percentage Distribution of all words in Brown Corpus

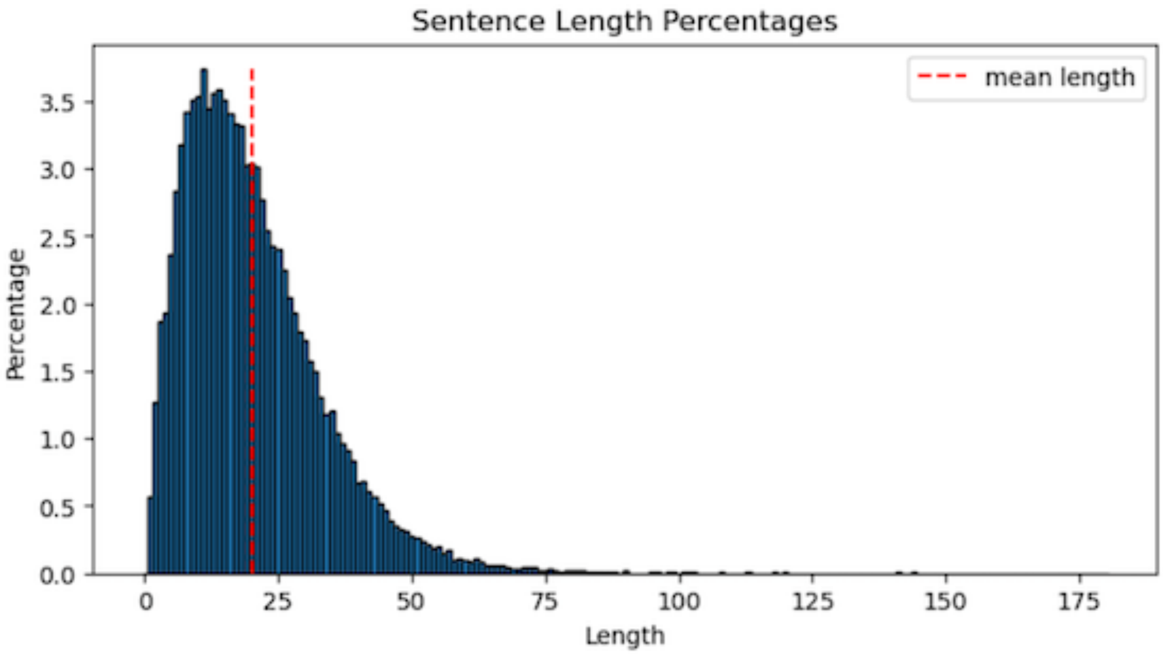


Percentage Distribution of the 100 most common words in Brown Corpus



Percentage Distribution of the 500 most common words in Brown Corpus

Problem 3


Sentence Length Percentages
mean length
Percentage
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Length
0
25
50
75
100
125
150
175