HW01

September 13, 2023

1 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.

1.1 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.

I discussed with Vineet Raju. I consulted the docs for common Python libraries, namely, matplotlib and used Stack Overflow to check the parameters of the plotting functions. I used ChatGPT to understand how to find the height of an axis in matplotlib.

1.2 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.

[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␣
 ↪download it again.
# This is a typical paradigm for datasets that you download onto your local␣
 ↪machine.
```

```
[nltk_data] Downloading package brown to /root/nltk_data…
[nltk_data]   Package brown is already up-to-date!
```

[1]: True

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

# We can access various components of this multi-text corpus: words, sentences,␣
 ↪and
# 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',
'IN'), ('term-end', 'NN'), ('presentments', 'NNS'), ('that', 'CS'), ('the',
'AT'), ('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'), ('thanks',
'NNS'), ('of', 'IN'), ('the', 'AT'), ('City', 'NN-TL'), ('of', 'IN-TL'),
('Atlanta', 'NP-TL'), ("''", "''"), ('for', 'IN'), ('the', 'AT'), ('manner',
'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',
'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', '.']], …]
```

Paragraphs in various categories, here are reviews:

```
[[['It', 'is', 'not', 'news', 'that', 'Nathan', 'Milstein', 'is', 'a', 'wizard',
'of', 'the', 'violin', '.'], ['Certainly', 'not', 'in', 'Orchestra', 'Hall',
'where', 'he', 'has', 'played', 'countless', 'recitals', ',', 'and', 'where',
'Thursday', 'night', 'he', 'celebrated', 'his', '20th', 'season', 'with', 'the',
'Chicago', 'Symphony', 'Orchestra', ',', 'playing', 'the', 'Brahms', 'Concerto',
'with', 'his', 'own', 'slashing', ',', 'demon-ridden', 'cadenza', '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', "orchestra's", 'great',
'nights', 'when', 'the', 'soloist', 'can', 'surpass', 'himself', '.'], ['This',
'time', 'the', 'orchestra', 'gave', 'him', 'some', 'superb', 'support', 'fired',
'by', 'response', 'to', 'his', 'own', 'high', 'mood', '.'], ['But', 'he', 'had',
'in', 'Walter', 'Hendl', 'a', 'willing', 'conductor', 'able', 'only', 'up',
```

'to', 'a', 'point', '.']], …]

1.2.1 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.

[3]:
```
# Part A

# Print out the number of occurrences of characters in the brown corpus. There␣
 ↪are no restrictions, so letters,
# upper or lower case, parentheses, white space, any character (printing or␣
 ↪otherwise) at all.
# Make this readable by a human, e.g., "There are xxx occurrences of characters␣
 ↪in the Brown corpus."

# Hint: use the brown.words list and read about the Python join function.␣
 ↪You'll be using this list
# of characters a lot in this problem, so calculate it once and assign it to an␣
 ↪appropriately-named variable.

# I strongly recommend you read about f-strings (introduced in Python 3.6) and␣
 ↪use them as your default format
# for Python print statements.


# Your code here
all_chars = "".join(brown.words())
num_char_occurences = len(all_chars)

print(f"There are {num_char_occurences} occurences of characters in the Brown␣
 ↪corpus.")
```

There are 4965882 occurences of characters in the Brown corpus.

[4]:
```
# Part B

# Print out the number of unique characters which occur in the Brown corpus (in␣
 ↪other words, duplicate
# occurrences do not count), and then print out a string consisting of all␣
 ↪these characters, sorted in order.
# Again, always print this information out in a readable form: "There are xxx␣
 ↪unique...."

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

# Your code here
unique_chars = "".join(sorted(set(all_chars)))
```

```
unique_chars_counts = len(unique_chars)

import textwrap

print(f"There are {unique_chars_counts} unique characters in the Brown corpus␣
 ↪and the sorted unique characters are:\n{unique_chars}")
```

```
There are 83 unique characters in the Brown corpus and the sorted unique
characters are:
!$%&'()*+,-
./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZ[]`abcdefghijklmnopqrstuvwxyz{}
```

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
```

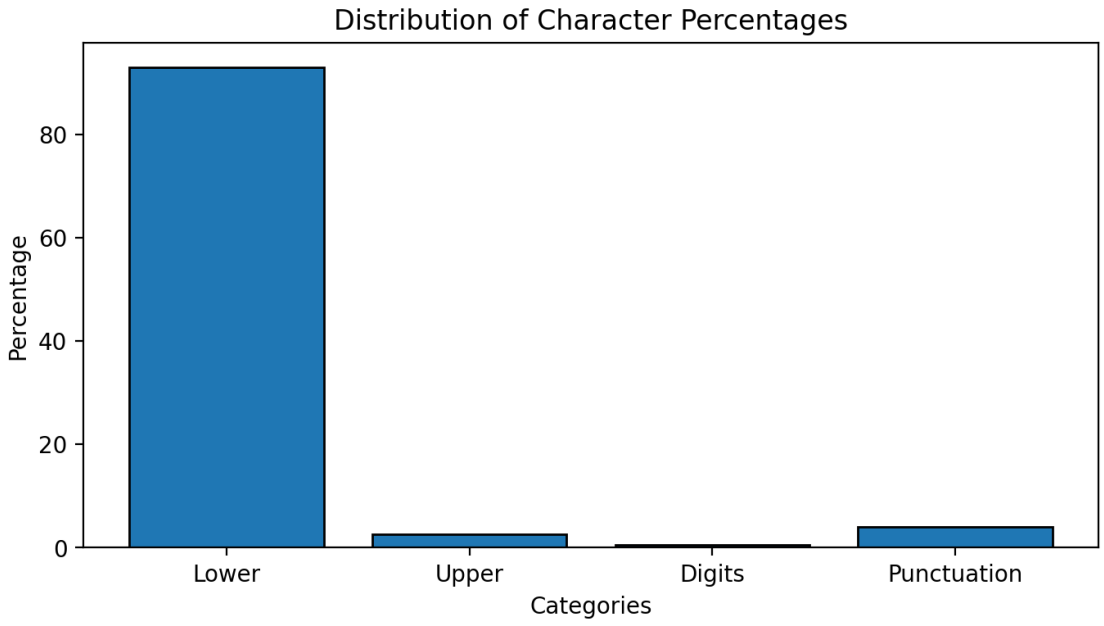
We will NOT be using the list of unique characters in the rest of this problem; whenever 'characters' are mentioned, we mean occurrences of characters, as in Part A.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib_inline
# get higher quality plots
matplotlib_inline.backend_inline.set_matplotlib_formats('retina')
import string
import collections
```

[6]:
```
# Part C

# Display a bar chart of the percentages of the characters that are in the␣
 ↪following categories:  ASCII lower-case letters,
# ASCII upper-case letters, digits, punctuation marks (the Brown corpus does␣
 ↪not contain anything but these).
# Display this as a bar chart, labelling each of the bars as 'Lower',␣
 ↪'Upper','Digits', 'Punctuation'.
# You do not need to show the exact percentages in the figure (see the sample␣
 ↪output).

# 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/library/
 ↪string.html) and use the
# string constants specified there. Look at the "Probability Distribution for␣
 ↪Coin Flips" in the
# 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.
```

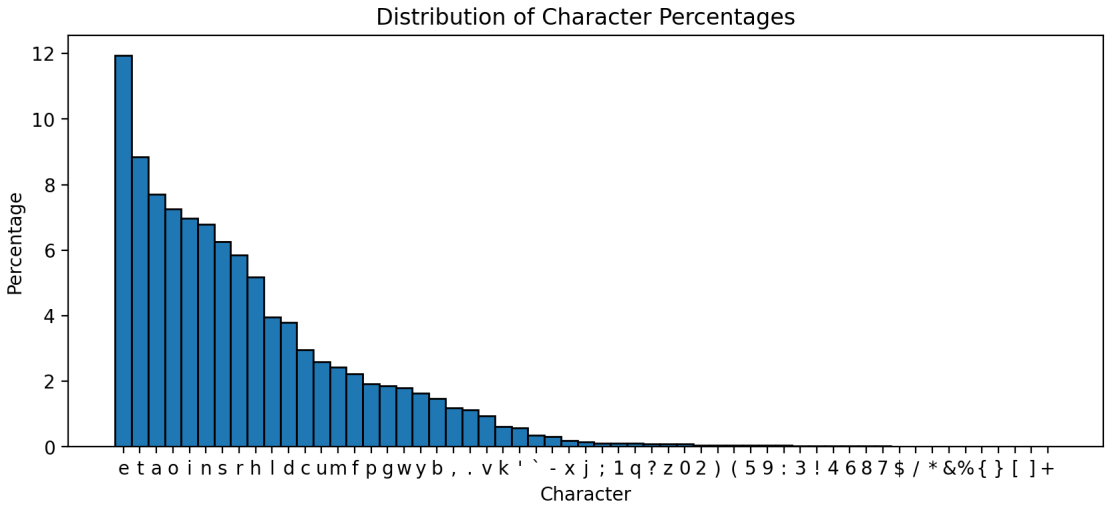
```
# Your code here
category_names = ["Lower", "Upper", "Digits", "Punctuation"]
category_constants = [string.ascii_lowercase, string.ascii_uppercase, string.
 ↪digits, string.punctuation]
category_counts = np.array([0 for _ in category_names])

for char in all_chars:
  for idx, constant in enumerate(category_constants):
    if char in constant:
      category_counts[idx] += 1

category_sum = sum(category_counts)
category_freqs = category_counts / category_sum
category_pcts = category_freqs * 100

fig = plt.figure(figsize=(8,4))

ax = fig.add_subplot(111)

ax.bar(category_names, category_pcts, edgecolor="k")
ax.set_title("Distribution of Character Percentages")
ax.set_xlabel("Categories")
ax.set_ylabel("Percentage")
```

[6]: Text(0, 0.5, 'Percentage')



[7]:
```
# Part D

# Print out a bar chart of the percentages of each character, in decreasing␣
 ↪order. Make this case-insensitive,
# so upper and lower letters are different; for example 'H' and 'h' are the␣
 ↪same letter.

# Hint:  Read about the Python function lower() and Counter from the␣
 ↪Collections library.

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

# Your code here
from collections import Counter

all_chars_lower = all_chars.lower()
all_chars_freq = Counter(all_chars_lower)

all_chars_freq = dict(sorted(all_chars_freq.items(), key=lambda item: item[1],␣
 ↪reverse=True))
x, height = zip(*all_chars_freq.items())
height = (np.asarray(height) / sum(height)) * 100

fig = plt.figure(figsize=(10,4))

ax = fig.add_subplot(111)

ax.bar(x, height, width=1.0, edgecolor="k")
ax.set_title("Distribution of Character Percentages")
ax.set_xlabel("Character")
ax.set_ylabel("Percentage")
```

[7]:
```
Text(0, 0.5, 'Percentage')
```



1.2.2 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.

[8]:
```
# Part A

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

# Hint: First create a list of lower-case words, and use it throughout this problem instead of brown.words()

# Your code here
all_words_lower = list(map(str.lower, brown.words()))
num_word_occurences = len(all_words_lower)
unique_words_lower = set(all_words_lower)
num_unique_words = len(unique_words_lower)

print(f"There are {num_word_occurences} occurences of words in the Brown corpus.")
print(f"There are {num_unique_words} unique words.")
```

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

NOTE: An example:

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

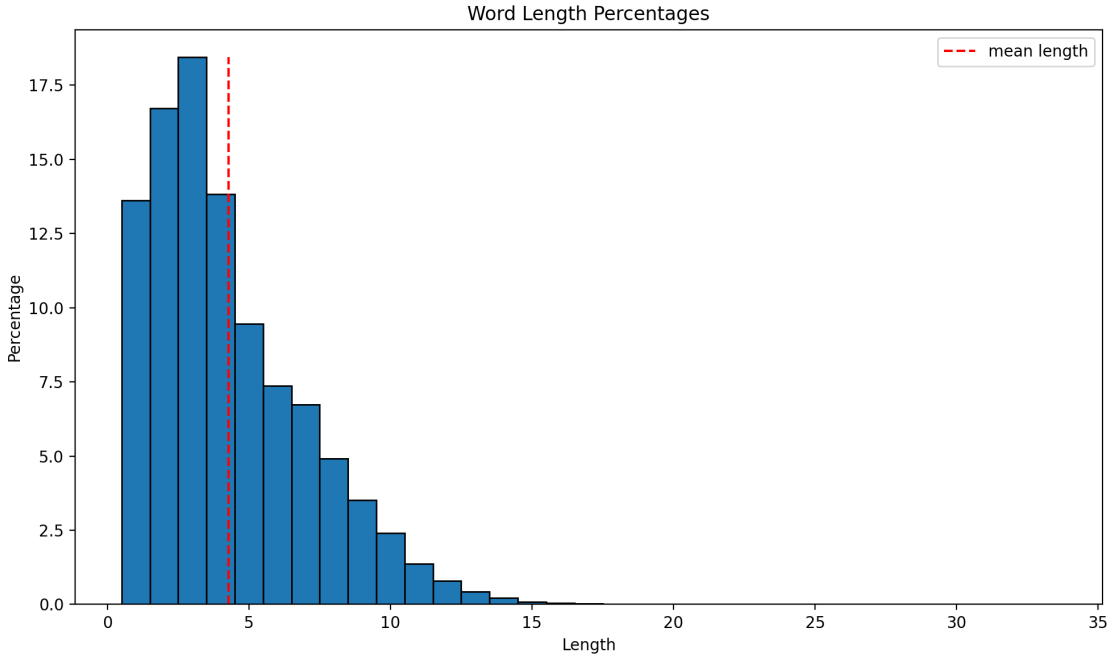
Again, we will NOT be using the list of unique words in the rest of this problem; whenever 'words' are mentioned, we mean occurrences of words, as in the first part of Part A.

[9]:
```
# Part B

# Print out the length of the longest word(s), and all occurrences of words of␣
 ↪that maximum length.
# 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 as␣
 ↪given!)

# Your code here
longest_word = max(unique_words_lower, key=len)
all_longest_words = "\n".join([word + "\t" for word in unique_words_lower if␣
 ↪word == longest_word])

print(f"The longest word is {longest_word}")
print("Here is an output of each of these words on a separate line, preceeded␣
 ↪by a tab:")
print(all_longest_words)
```

```
The longest word is nnuolapertar-it-vuh-karti-birifw-
Here is an output of each of these words on a separate line, preceeded by a tab:
nnuolapertar-it-vuh-karti-birifw-
```

[10]:
```
# Part C

# Display a bar chart of the percentages of word lengths of all occurrences of␣
 ↪words
# and give the average length of a word. Draw a dotted red vertical line whose␣
 ↪height is the height
# of the highest bar, and whose x position is the average word length; give a␣
 ↪legend explaining
# what the bar means (see PythonRefresher, as usual, for examples of how to do␣
 ↪this).

# Print out a human-readable statement about the average word length (to 4␣
 ↪decimal places) below the bar chart.

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

```python
# Your code here
word_lengths = np.array([len(word) for word in brown.words()])
avg_word_len = np.mean(word_lengths)
word_len, len_freqs = np.unique(word_lengths, return_counts=True)
pct_word_lengths = (len_freqs / len_freqs.sum()) * 100
max_pct_word = np.max(pct_word_lengths)

fig2 = plt.figure(figsize=(10, 6), tight_layout=True)

ax2 = fig2.add_subplot(111)

ax2.set_title("Word Length Percentages")
ax2.set_xlabel("Length")
ax2.set_ylabel("Percentage")

bar = ax2.bar(word_len, pct_word_lengths, width=1.0, edgecolor="k")

ax2.axvline(x=avg_word_len, ymax=max_pct_word / ax2.get_ylim()[1], color="r",
            linestyle="--",
            label="mean length")

ax2.legend()

plt.show(block=False)

print(f"\nThe average word length in the Brown corpus is \
{round(avg_word_len, 4)}")
```



The average word length in the Brown corpus is 4.2765

[11]:
```
# Part D

# Now we will consider word frequencies (expressed as percentages). To simplify␣
 ↪matters, we will only
# allow "normal" words, i.e., those consisting of only lower-case letters, with␣
 ↪possible single quotes,
# periods, and dashes. Since this involves regular expressions (we'll cover␣
 ↪next Tuesday, 9/11),
# this code is provided.

# Your task in this problem is to give the twenty most common normal words, in␣
 ↪decreasing order, and the percentage of
# all occurrences of normal words these represent. Be sure to give your answer␣
 ↪as percentages to 2 decimal places.

# 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))

# Your code here
normal_words = list(filter(is_normal_word, brown.words()))
normal_words, normal_word_counts = np.unique(normal_words, return_counts=True)
sorted_indices = np.argsort(normal_word_counts)
normal_words, normal_word_counts = normal_words[sorted_indices[::-1]],␣
 ↪normal_word_counts[sorted_indices[::-1]]
normal_word_pcts = (normal_word_counts / normal_word_counts.sum()) * 100

twenty_common_words = normal_words[:20]
twenty_common_words_pct = normal_word_pcts[:20]

print("The twenty most common normal words and their percentages are:\n")

for word, pct in zip(twenty_common_words, twenty_common_words_pct):
  print(f"word: {word} \tpercentage: {round(pct, 2)}")
```

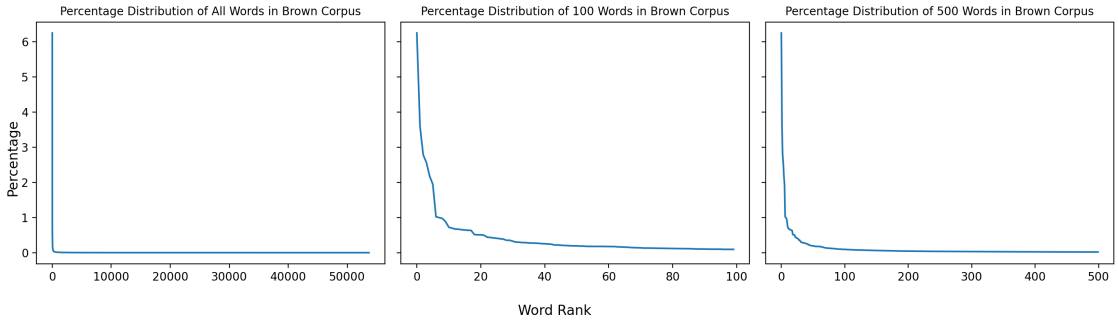
The twenty most common normal words and their percentages are:

```
word: the       percentage: 6.25
word: of        percentage: 3.59
word: and       percentage: 2.78
word: to        percentage: 2.56
word: a         percentage: 2.18
word: in        percentage: 1.95
word: that      percentage: 1.02
word: is        percentage: 1.0
word: was       percentage: 0.97
word: for       percentage: 0.88
word: The       percentage: 0.72
word: with      percentage: 0.7
word: it        percentage: 0.67
word: as        percentage: 0.67
word: he        percentage: 0.65
word: his       percentage: 0.64
word: on        percentage: 0.64
word: be        percentage: 0.63
word: I         percentage: 0.51
word: by        percentage: 0.51
```

[12]:
```
# Part E

# Now give the distribution of the percentages of normal word occurrences, in␣
 ↪decreasing order,
# 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.,␣
 ↪using plot(...) instead of bar(...))

# Show this for all normal words, then for the 100 most common normal words,␣
 ↪then for the 500 most
# common normal words.

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


# Your code here
fig3, axs = plt.subplots(nrows=1, ncols=3, figsize=(14, 4), tight_layout=True,
                         sharey=True)
dists = ["All", 100, 500]

for idx, ax in enumerate(axs.ravel()):
  ax.tick_params(axis="both", which="major", labelsize=10)
  dist = dists[idx]
  ax.set_title(f"Percentage Distribution of {dist} Words in Brown Corpus",
```

```
                fontsize=10)
  indices = slice(0, len(normal_words) if dist == "All" else dist)
  ax.plot(normal_word_pcts[indices])

fig3.supxlabel("Word Rank")
fig3.supylabel("Percentage")
```

[12]:
```
Text(0.02, 0.5, 'Percentage')
```



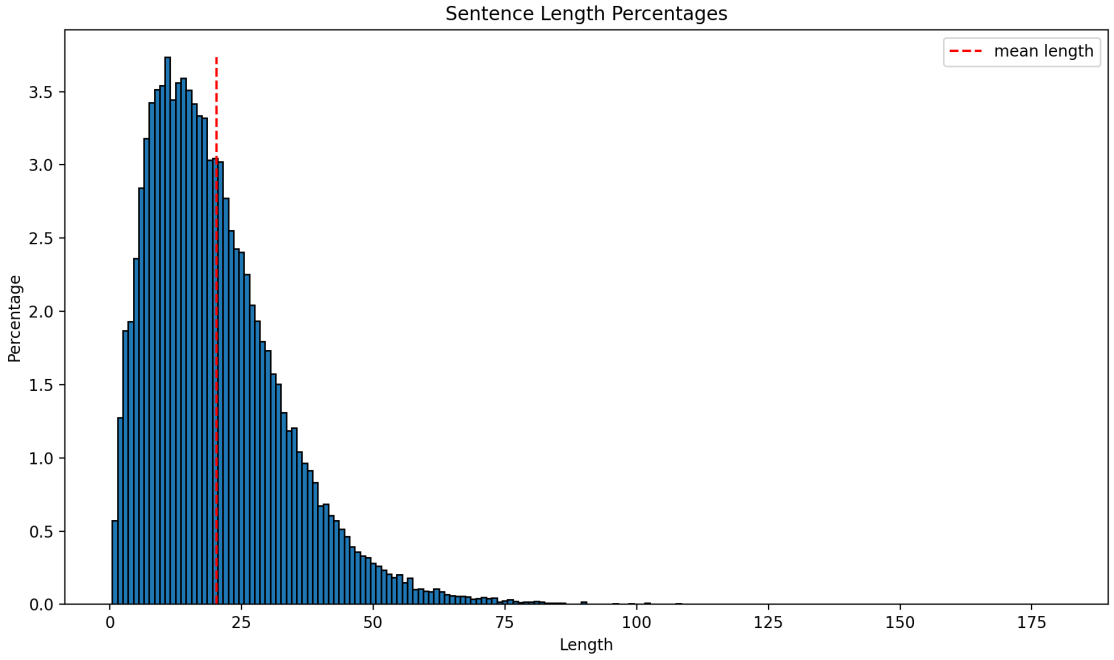
1.2.3 Problem Three (5 points): Paragraphs

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.

[13]:
```
# Your code here

sentence_lengths = np.array([len(sent) for sent in brown.sents()])
avg_sentence_len = np.mean(sentence_lengths)
sentence_len, len_sentence_freqs = np.unique(sentence_lengths,
                                             return_counts=True)
pct_sentence_lengths = (len_sentence_freqs / len_sentence_freqs.sum()) * 100
max_pct_sentence = np.max(pct_sentence_lengths)

fig3 = plt.figure(figsize=(10, 6), tight_layout=True)

ax3 = fig3.add_subplot(111)

ax3.set_title("Sentence Length Percentages")
ax3.set_xlabel("Length")
ax3.set_ylabel("Percentage")

ax3.bar(sentence_len, pct_sentence_lengths, width=1.0, edgecolor="k")
ax3.axvline(x=avg_sentence_len, ymax=max_pct_sentence / ax3.get_ylim()[1],
```

```
            color="r", linestyle="--", label="mean length")

ax3.legend()

plt.show(block=False)

print(f"\nThe average sentence length in the Brown corpus is \
{round(avg_sentence_len, 4)}")
```




```
The average sentence length in the Brown corpus is 20.251
```