<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Selecting [0] with <code>random.choice()</code> means that rows are used, and the final argument, 1, selects a single element.</td>
<td>Selecting [0] with <code>random.choice()</code> means that rows are used.</td>
<td>Print 2</td>
</tr>
<tr>
<td>32</td>
<td><code>import nltk&lt;br&gt;from nltk.corpus import stopwords&lt;br&gt;import matplotlib.pyplot as plt&lt;br&gt;from collections import Counter</code></td>
<td><code>from collections import Counter&lt;br&gt;import nltk&lt;br&gt;from nltk.corpus import stopwords&lt;br&gt;import matplotlib.pyplot as plt</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>33</td>
<td>Start by importing NLTK and the Stopwords Corpus. Then import matplotlib.</td>
<td>Start by importing the collections module's counter class (for counting list items), matplotlib, NLTK, and the Stopwords Corpus.</td>
<td>Print 3</td>
</tr>
</tbody>
</table>
| 39   | ```python<br>def stopwords_test(words_by_author, len_shortest_corpus):
    """Plot stopwords freq by author, truncated to shortest corpus length."""
    stopwords_by_author_freq_dist = dict()
    plt.figure(2)
    stop_words = set(stopwords.words('english'))  # Use set for speed.
    #print('Number of stopwords = {}
'.format(len(stop_words)))
    #print('Stopwords = {}
'.format(stop_words))
    for i, author in enumerate(words_by_author):
        stopwords_by_author = [word for word in words_by_author[author][:len_shortest_corpus] if word in stop_words]
        stopwords_by_author_freq_dist[author] = nltk.FreqDist(stopwords_by_author)
        stopwords_by_author_freq_dist[author].plot(50,
            label=author,
            linestyle=LINES[i],
            title= '50 Most Common Stopwords')
    fdist = dict()
    plt.figure(2)
    stop_words = stopwords.words('english')
    for i, author in enumerate(words_by_author):
        stopwords_by_author = [word for word in words_by_author[author][:len_shortest_corpus] if word in stop_words]
        fdist[author] = {word: stopwords_by_author.count(word) for word in stop_words[:50]}  # Use first 50 of 179 stopwords.
        k, v = list(fdist[author].keys()), list(fdist[author].values())
        plt.plot(k, v, label=author, linestyle=LINES[i], lw=1)
    ##    plt.xticks([])  # Turn off labels if plotting >50 stopwords.
    plt.title('First 50 Stopwords')
    plt.legend()
``` | ```python<br>def stopwords_test(words_by_author, len_shortest_corpus):
    """Plot stopwords freq by author, truncated to shortest corpus length."""
    stopwords_by_author_freq_dist = dict()
    plt.figure(2)
    stop_words = set(stopwords.words('english'))  # Use set for speed.
    #print('Number of stopwords = {}
'.format(len(stop_words)))
    #print('Stopwords = {}
'.format(stop_words))
    for i, author in enumerate(words_by_author):
        stopwords_by_author = [word for word in words_by_author[author][:len_shortest_corpus] if word in stop_words]
        stopwords_by_author_freq_dist[author] = nltk.FreqDist(stopwords_by_author)
        stopwords_by_author_freq_dist[author].plot(50,
            label=author,
            linestyle=LINES[i],
            title= '50 Most Common Stopwords')
        fdist[author] = {word: stopwords_by_author.count(word) for word in stop_words[:50]}  # Use first 50 of 179 stopwords.
        k, v = list(fdist[author].keys()), list(fdist[author].values())
        plt.plot(k, v, label=author, linestyle=LINES[i], lw=1)
    ##    plt.xticks([])  # Turn off labels if plotting >50 stopwords.
    plt.title('First 50 Stopwords')
    plt.legend()
``` | Print 3 |
### Listing 2-5: Defining the stopwords_test() function

Define a function that takes the words dictionary and the length of the shortest corpus variables as arguments. Then initialize a dictionary to hold the frequency distribution of stop words for each author.

```python
plt.legend()
##    plt.show() # Uncomment to see plot while coding function.
```

### Listing 2-5: Defining the stopwords_test() function

Define a function that takes the words dictionary (fdist) and the length of the shortest corpus variables as arguments. Then initialize a dictionary to hold the frequency distribution of stop words for each author.

```python
plt.xticks(rotation=90)
##    plt.show() # Uncomment to see plot while coding function.
```

Assign a local variable, `stop_words`, to the NLTK stop words corpus for English. Sets are quicker to search than lists, so make the corpus a set for faster lookups later. The next two lines, currently commented out, print the number of stop words (179) and the stop words themselves.

Now, start looping through the authors in the `words_by_author` dictionary. Use list comprehension to pull out all the stop words in each author's corpus and use these as the value in a new dictionary named `stopwords_by_author`. In the next line, you'll pass this dictionary to NLTK's `FreqDist()` method and use the output to populate the `stopwords_by_author_freq_dist` dictionary. This dictionary will contain the data needed to make the frequency distribution plots for each author.

Repeat the code you used to plot the word lengths in Listing 2-4, but set the number of samples to 50 and give it a different title. This will plot the top 50 stop words in use (Figure 2-4).

```
Assign a local variable, `stop_words`, to the NLTK stop words corpus for English. Now, start looping through the authors in the `words_by_author` dictionary. Use list comprehension to pull out all the stop words in each author's corpus to make a new list named `stopwords_by_author`. In the next line, fill the fdist dictionary with the stop words and their count, per author, truncated to the first 50 stop words. (As there are 179 stop words, it's best to plot them in chunks.)

The next step is to extract the fdist keys and values into lists and then pass these variables to the `plt.plot()` function. This produces Figure 2-4.

```
```

Figure 2-4: Frequency plot of top 50 stop words by author

```
```

Figure 2-4: Frequency plot of the first 50 stop words by author

```
```

Deletion

The taggers are typically trained on large datasets like the Penn Treebank or Brown Corpus, making them highly accurate though not perfect. You can also find training data and taggers for languages other than English. You don't need to worry about all
<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;&quot;&quot;Plot author use of parts-of-speech such as nouns, verbs, adverbs.&quot;&quot;&quot;</td>
<td>&quot;&quot;&quot;Plot author use of parts-of-speech such as nouns, verbs, adverbs.&quot;&quot;&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by_author_pos_freq_dist = dict()</td>
<td>by_author_pos_freq_dist = dict()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plt.figure(3)</td>
<td>plt.figure(3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for i, author in enumerate(words_by_author):</td>
<td>for i, author in enumerate(words_by_author):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by_author_pos_freq_dist[author].plot(35, label=author, linestyle=LINES[i], title='Part of Speech')</td>
<td>by_author_pos_freq_dist[author].plot(35, label=author, linestyle=LINES[i], title='Part of Speech')</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plt.legend() plt.show()</td>
<td>plt.legend() plt.show()</td>
<td></td>
</tr>
</tbody>
</table>

Listing 2-6: Defining the parts_of_speech_test() function

41

Next, make a frequency distribution of the POS list and with each loop plot the curve, using the top 35 samples. Note that there are only 36 POS tags and several, such as list item markers, rarely appear in novels.

42

Next, make a frequency distribution by calling the Counter class we imported previously. Then, extract the keys and values as lists and pass them to the plotting function. Use a triangle marker and turn off the connecting curves with linestyle=''.

42

Print 3
Once again, the match between the Doyle and unknown curves is clearly better than the match of unknown to Wells. This suggests that Doyle is the author of the unknown corpus.

Because there are only about a dozen sentences in the whole speech with 10 or fewer words...

After March 2021 install version 3.8.3 (https://pypi.org/project/gensim/3.8.3/).

Then, call the gensim summarize() function to summarize the speech in 225 words. This word count will produce about 15 sentences, assuming the average sentence has 15 words.

Then, summarize the speech and print the result in one step by calling the gensim summarize() function within the print() function. Set the word_count argument to 225. In theory, this will produce a summary of 15 sentences, assuming the average sentence contains 15 words.
<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>Deletion</td>
<td>Ideally, you could summarize the speech and print the summary in one step.</td>
<td>Print 2</td>
</tr>
</tbody>
</table>

| 63-64 | Unfortunately, gensim sometimes duplicates sentences in summaries, and that occurs here: | After running the program, you should get the following output: | Print 2 |

**Summary of Make Your Bed speech:**

Basic SEAL training is six months of long torturous runs in the soft sand, midnight swims in the cold water off San Diego, obstacle courses, unending calisthenics, days without sleep and always being cold, wet and miserable.

Basic SEAL training is six months of long torturous runs in the soft sand, midnight swims in the cold water off San Diego, obstacle courses, unending calisthenics, days without sleep and always being cold, wet and miserable.

To avoid duplicating text, you first need to break out the sentences in the summary variable using the NLTK sent_tokenize() function. Then make a set from these sentences, which will remove duplicates. Finish by printing the results.

Because sets are unordered, the arrangement of the sentences may change if you run the program multiple times.

**Summary of Make Your Bed speech:**

If you can’t do the little things right, you will never do the big things right. And, if by chance you have a miserable day, you will come home to a bed that is made – that you made – and a made bed gives you encouragement that tomorrow will be better. If you want to change the world, start off by making your bed. During SEAL training the students are broken down into boat crews. It’s just the way life is sometimes. If you want to change the world get over being a sugar cookie and keep moving forward. Every day during training you were challenged with multiple physical events – long runs, long swims, obstacle courses, hours of calisthenics – something designed to test your mettle. Basic SEAL training is six months of long torturous runs in the soft sand, midnight swims in the cold water off San Diego, obstacle courses, unending calisthenics, days without sleep and always being cold, wet and miserable.

---snip---

To avoid duplicating text, you first need to break out the sentences in the summary variable using the NLTK sent_tokenize() function. Then make a set from these sentences, which will remove duplicates. Finish by printing the results.

Because sets are unordered, the arrangement of the sentences may change if you run the program multiple times.

**Summary of Make Your Bed speech:**

If you can’t do the little things right, you will never do the big things right. And, if by chance you have a miserable day, you will come home to a bed that is made – that you made – and a made bed gives you encouragement that tomorrow will be better. If you want to change the world, start off by making your bed. During SEAL training the students are broken down into boat crews. It’s just the way life is sometimes. If you want to change the world get over being a sugar cookie and keep moving forward. Every day during training you were challenged with multiple physical events – long runs, long swims, obstacle courses, hours of calisthenics – something designed to test your mettle. Basic SEAL training is six months of long torturous runs in the soft sand, midnight swims in the cold water off San Diego, obstacle courses, unending calisthenics, days without sleep and always being cold, wet and miserable.

---snip---

If you increase the word count parameter to 450, the “make your bed” aspects of the speech are stressed even more (I’ve shortened the output for brevity).
If you take the time to read the full speech, you’ll probably conclude that gensim produced a fair summary. Although these two results are different, both extracted the key points of the speech, including the reference to making your bed.

If you take the time to read the full speech, you'll probably conclude that gensim produced a fair summary. Although these two results are different, both extracted the key points of the speech, including the reference to making your bed.

If you take the time to read the full speech, you’ll probably conclude that gensim produced a fair summary. It extracted many of the key points of the speech, including the reference to making your bed.