

Analyzing data with PySpark Assignment

To prepare for this assignment, I logged into Jetstream, launched an instance running Ubuntu, and ensured Docker was running. I then created a directory to store my work and launched a Docker container with a docker-compose.yaml file that was configured to run a Jupyter Notebook with PySpark.

```
exouser@vaish-pyspark: ~/SparkWork

exouser@vaish-pyspark:~$ sudo docker --version
Docker version 24.0.7, build 24.0.7-0ubuntu2~22.04.1

exouser@vaish-pyspark:~$ mkdir SparkWork
exouser@vaish-pyspark:~$ cd SparkWork
exouser@vaish-pyspark:~/SparkWork$ nano docker-compose.yaml
exouser@vaish-pyspark:~/SparkWork$ nano docker-compose.yaml
exouser@vaish-pyspark:~/SparkWork$ sudo docker compose up
[+] Running 0/1
[+] Running 0/1ers [
] 0B/0B Pulling 0.8[+] Running 5/1ers [
] 310.5kB/29.54MB P
pulling 0.9[+] Running 7/1ers [
] 276B/276B Pulling 1.1[+] Running 10/1rs [
] 527.6kB
/104.8MB Pulling 1.2[+] Running 10/1rs [
] 24.73MB/104.8MB Pulling 1.3[+] Running 10/

exouser@vaish-pyspark:~/SparkWork

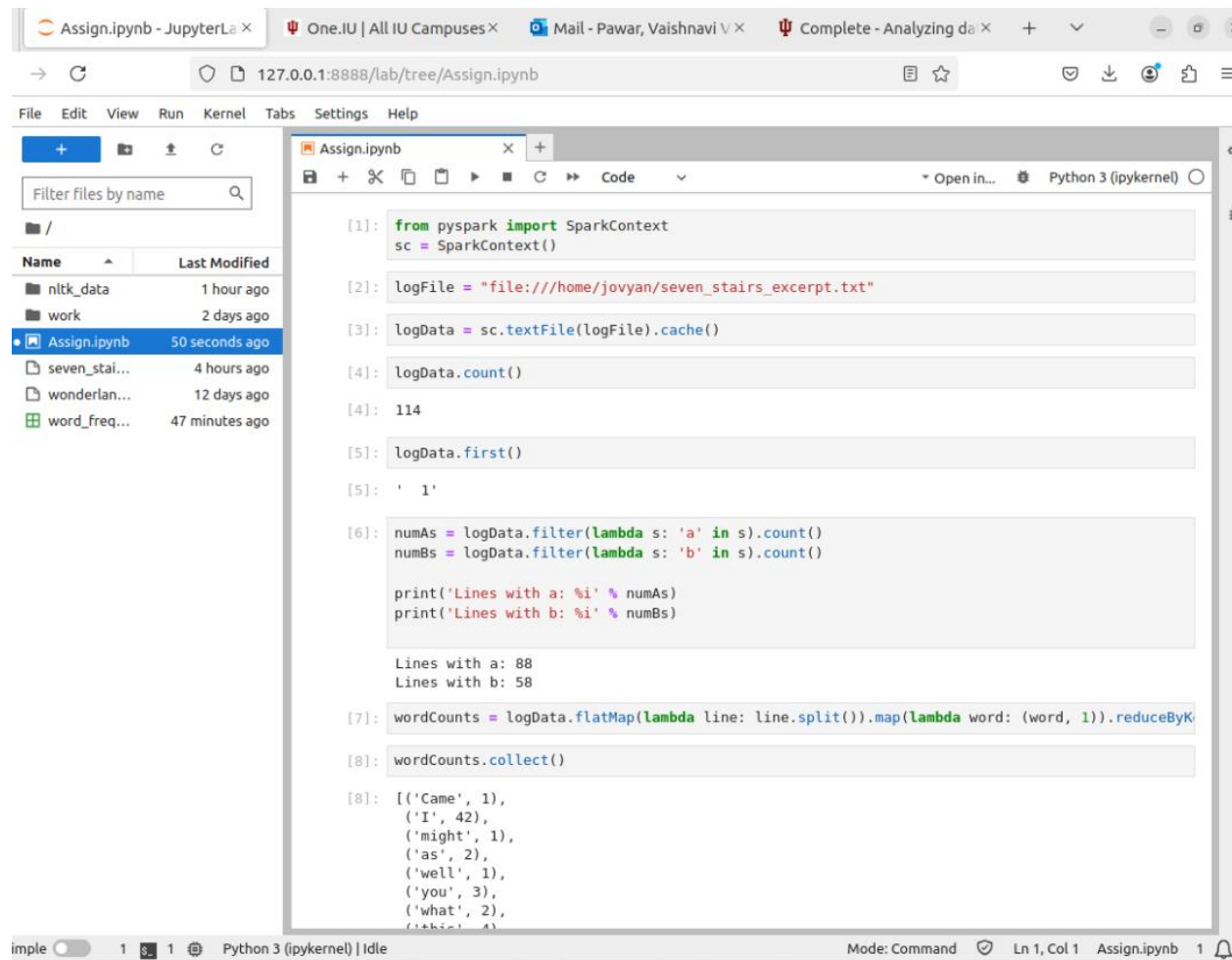
[+] Running 2/4
✓ Network sparkwork_default Created 0.1s
✓ Container sparkwork-spark-1 Created 0.3s
Attaching to spark-1
spark-1 | Entered start.sh with args: jupyter lab
spark-1 | Running hooks in: /usr/local/bin/start-notebook.d as uid: 1000 gid: 100
spark-1 | Done running hooks in: /usr/local/bin/start-notebook.d
spark-1 | Running hooks in: /usr/local/bin/before-notebook.d as uid: 1000 gid: 100
spark-1 | Sourcing shell script: /usr/local/bin/before-notebook.d/spark-config.sh
spark-1 | Done running hooks in: /usr/local/bin/before-notebook.d
```

After launching the container, I used the provided token to access Jupyter Notebook through the Web Desktop, where I initialized SparkContext and created Resilient Distributed Datasets (RDDs) for text analysis. This configuration provided a scalable environment for practicing PySpark operations and running MapReduce tasks efficiently.

```
spark-1 |
spark-1 | To access the server, open this file in a browser:
spark-1 | file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
spark-1 | Or copy and paste one of these URLs:
spark-1 | http://66143c9d5e95:8888/lab?token=5acf76699cb68afd06c2810cebde05895da027baa19ef3b9
spark-1 | http://127.0.0.1:8888/lab?token=5acf76699cb68afd06c2810cebde05895da027baa19ef3b9
spark-1 | [I 2024-10-31 21:25:20.364 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodejs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server, python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

After launching the container, I used the Web Desktop to access Jupyter Notebook, which allowed me to initialize SparkContext and create Resilient Distributed Datasets (RDDs) for text analytics. This configuration provided a scalable environment for practicing PySpark operations and running MapReduce tasks efficiently.

During the first phase of this project, I practiced with a sample file to become familiar with PySpark's RDD operations. I loaded the file into an RDD, cached it for speed, and experimented with simple tasks such as counting the lines and filtering those that contained specific letters ('a' and 'b'). In addition, I used flatMap, map, and reduceByKey transformations to implement word counting. This practice increased my confidence in using PySpark for more complex data processing tasks.



The screenshot displays a Jupyter Notebook titled "Assign.ipynb" running on a web interface. The left sidebar shows a file explorer with a tree view containing folders like "nltk_data" and "work", and files like "seven_stai...", "wonderlan...", and "word_freq...". The main area shows the notebook's code cells and their outputs.

```
[1]: from pyspark import SparkContext
    sc = SparkContext()

[2]: logFile = "file:///home/jovyan/seven_stairs_excerpt.txt"

[3]: logData = sc.textFile(logFile).cache()

[4]: logData.count()

[4]: 114

[5]: logData.first()

[5]: ' 1'

[6]: numAs = logData.filter(lambda s: 'a' in s).count()
    numBs = logData.filter(lambda s: 'b' in s).count()

    print('Lines with a: %i' % numAs)
    print('Lines with b: %i' % numBs)

    Lines with a: 88
    Lines with b: 58

[7]: wordCounts = logData.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

[8]: wordCounts.collect()

[8]: [('Came', 1),
      ('I', 42),
      ('might', 1),
      ('as', 2),
      ('well', 1),
      ('you', 3),
      ('what', 2),
      ('this', 1)]
```

The bottom status bar indicates the notebook is running on "Python 3 (ipykernel)" and is in "Idle" mode. The current cell is at "Ln 1, Col 1".

To begin my analysis, I selected "Alice's Adventures in Wonderland" from Project Gutenberg as my dataset, which is noteworthy for its rich and diverse language. I loaded the text into Spark as an RDD and then performed several RDD actions to gain insights. First, I tallied basic statistics like the total number of lines and characters in the book.

I also extracted the first few lines to get a general idea of the content. I used a MapReduce approach to compute word frequencies, resulting in a "bag of words" for examining common terms. These steps demonstrate the utility of RDDs for distributed data analysis and lay the groundwork for investigating word patterns in the text.

```
(base) jovyan@66143c9d5e95:~$ wget https://www.gutenberg.org/cache/epub/11/pg11.txt -O wonderland.txt
--2024-11-03 03:21:59-- https://www.gutenberg.org/cache/epub/11/pg11.txt
Resolving www.gutenberg.org (www.gutenberg.org)... 152.19.134.47, 2610:28:3090:3000:0:bad:cafe:47
Connecting to www.gutenberg.org (www.gutenberg.org)|152.19.134.47|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 174356 (170K) [text/plain]
Saving to: 'wonderland.txt'

wonderland.txt      100%[=====>] 170.27K  --.-KB/s   in 0.1s

2024-11-03 03:21:59 (1.71 MB/s) - 'wonderland.txt' saved [174356/174356]

(base) jovyan@66143c9d5e95:~$
```

```
[9]: book_file = "file:///home/jovyan/wonderland.txt"

[10]: book_data = sc.textFile(book_file).cache()

[11]: print("Total lines:", book_data.count())
Total lines: 3757

[12]: print("First line:", book_data.first())
First line: The Project Gutenberg eBook of Alice's Adventures in Wonderland

[13]: print("First 5 lines:", book_data.take(5))
First 5 lines: ["The Project Gutenberg eBook of Alice's Adventures in Wonderland", ' ', 'This ebook is for the use of anyo
ne anywhere in the United States and', 'most other parts of the world at no cost and with almost no restrictions', 'whatsoeve
r. You may copy it, give it away or re-use it under the terms']

[14]: sample_lines = book_data.takeSample(withReplacement=False, num=5, seed=42)
print("Sample lines:", sample_lines)
Sample lines: ['Gryphon answered, very nearly in the same words as before, "It's all', 'in questions of eating and drinking.'
, 'last she stretched her arms round it as far as they would go, and broke', '', '[later editions continued as follows']

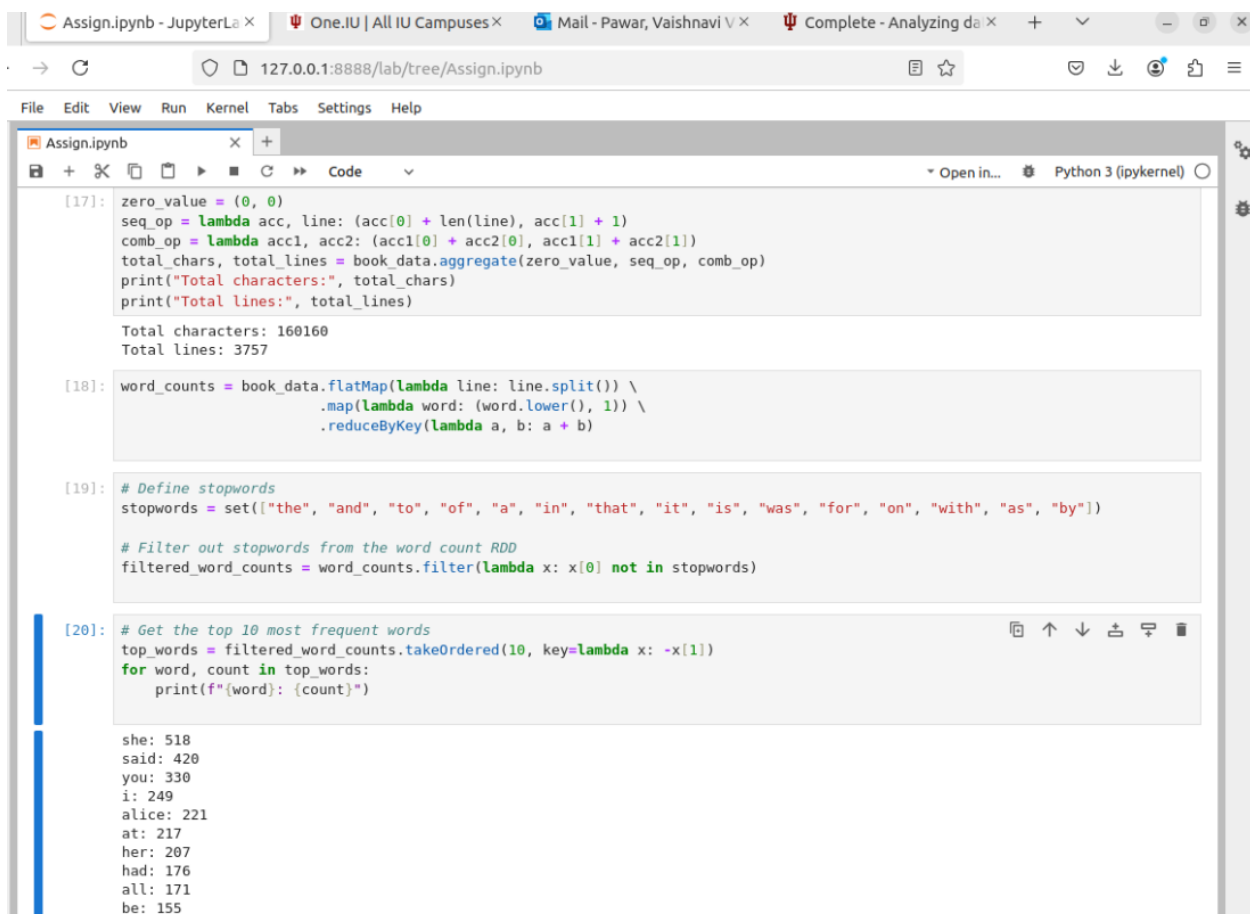
[15]: total_chars = book_data.map(lambda line: len(line)).reduce(lambda a, b: a + b)
print("Total characters:", total_chars)
Total characters: 160160

[16]: word_rdd = book_data.flatMap(lambda line: line.split())
word_counts = word_rdd.countByValue()
print("Word counts (sample):", {k: word_counts[k] for k in list(word_counts)[:10]})
Word counts (sample): {'The': 106, 'Project': 79, 'Gutenberg': 22, 'eBook': 4, 'of': 604, 'Alice's': 2, 'Adventures': 4, 'in'
: 406, 'Wonderland': 4, 'This': 18}
```

Screenshot of RDD operations, showing counts, filtering, and transformations.

After obtaining the basic statistics, I used MapReduce to perform a word frequency analysis. First, I used transformations to divide lines into individual words, then converted them to lowercase and assigned each word a count of one. Then I used `reduceByKey` to add up the counts for each unique word. To improve the analysis, I removed common stop words such as "the," "and", and "of," which do not provide useful information.

Finally, I extracted the top ten most frequently used words in the text, revealing prominent words like "she" and "said," which are most likely related to the narrative style of "Alice's Adventures in Wonderland." This analysis focuses on the main characters and recurring themes in the book.



The screenshot shows a Jupyter Notebook with the following code and output:

```
[17]: zero_value = (0, 0)
seq_op = lambda acc, line: (acc[0] + len(line), acc[1] + 1)
comb_op = lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])
total_chars, total_lines = book_data.aggregate(zero_value, seq_op, comb_op)
print("Total characters:", total_chars)
print("Total lines:", total_lines)

Total characters: 160160
Total lines: 3757

[18]: word_counts = book_data.flatMap(lambda line: line.split()) \
    .map(lambda word: (word.lower(), 1)) \
    .reduceByKey(lambda a, b: a + b)

[19]: # Define stopwords
stopwords = set(["the", "and", "to", "of", "a", "in", "that", "it", "is", "was", "for", "on", "with", "as", "by"])

# Filter out stopwords from the word count RDD
filtered_word_counts = word_counts.filter(lambda x: x[0] not in stopwords)

[20]: # Get the top 10 most frequent words
top_words = filtered_word_counts.takeOrdered(10, key=lambda x: -x[1])
for word, count in top_words:
    print(f"{word}: {count}")

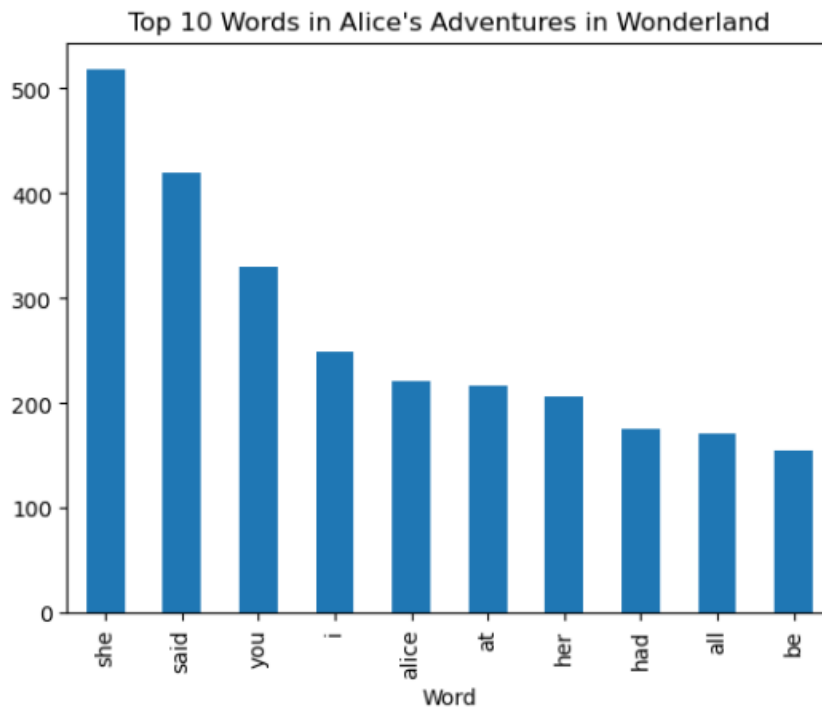
she: 518
said: 420
you: 330
i: 249
alice: 221
at: 217
her: 207
had: 176
all: 171
be: 155
```

To visually represent the top ten most common words in "Alice's Adventures in Wonderland," I converted the word frequency data to a DataFrame and plotted it as a bar chart. The visualization shows that "she," "said," and "you" are the most frequently used words, indicating a dialogue-heavy text focused on character interactions. Furthermore, the name "Alice" appears frequently, indicating her central role in the story. This chart effectively summarizes the key terms while also providing insight into the narrative style and recurring themes of the text.

```
import pandas as pd
import matplotlib.pyplot as plt

# Convert to a DataFrame for visualization
df = pd.DataFrame(top_words, columns=["Word", "Count"])

# Plot the top 10 words
df.plot(kind='bar', x='Word', y='Count', legend=False)
plt.title("Top 10 Words in Alice's Adventures in Wonderland")
plt.show()
```



Bar chart for the top 10 most frequent words

To further analyze word frequencies in "Alice's Adventures in Wonderland," I processed the top 100 words and exported them to a CSV file for ease of use and analysis. For clarity in visualization, I focused on the top 20 most frequently used words and displayed them in a bar plot. This visualization provides a comprehensive overview of the text's dominant words, making it easier to identify key characters, themes, and narrative elements. By employing a Seaborn bar plot with improved readability, the plot effectively highlights recurring words, providing insight into the story's linguistic patterns.

```
Assign.ipynb - JupyterLab x One.IU | All IU Campuses x Mail - Pawar, Vaishnavi x Complete - Analyzing da x + - x
127.0.0.1:8888/lab/tree/Assign.ipynb
File Edit View Run Kernel Tabs Settings Help
Assign.ipynb x +
Python 3 (ipykernel)

[22]: import pandas as pd

# Convert the RDD (e.g., filtered_word_counts) to a DataFrame
top_words = filtered_word_counts.takeOrdered(100, key=lambda x: -x[1]) # Top 100 words
df = pd.DataFrame(top_words, columns=['Word', 'Frequency'])

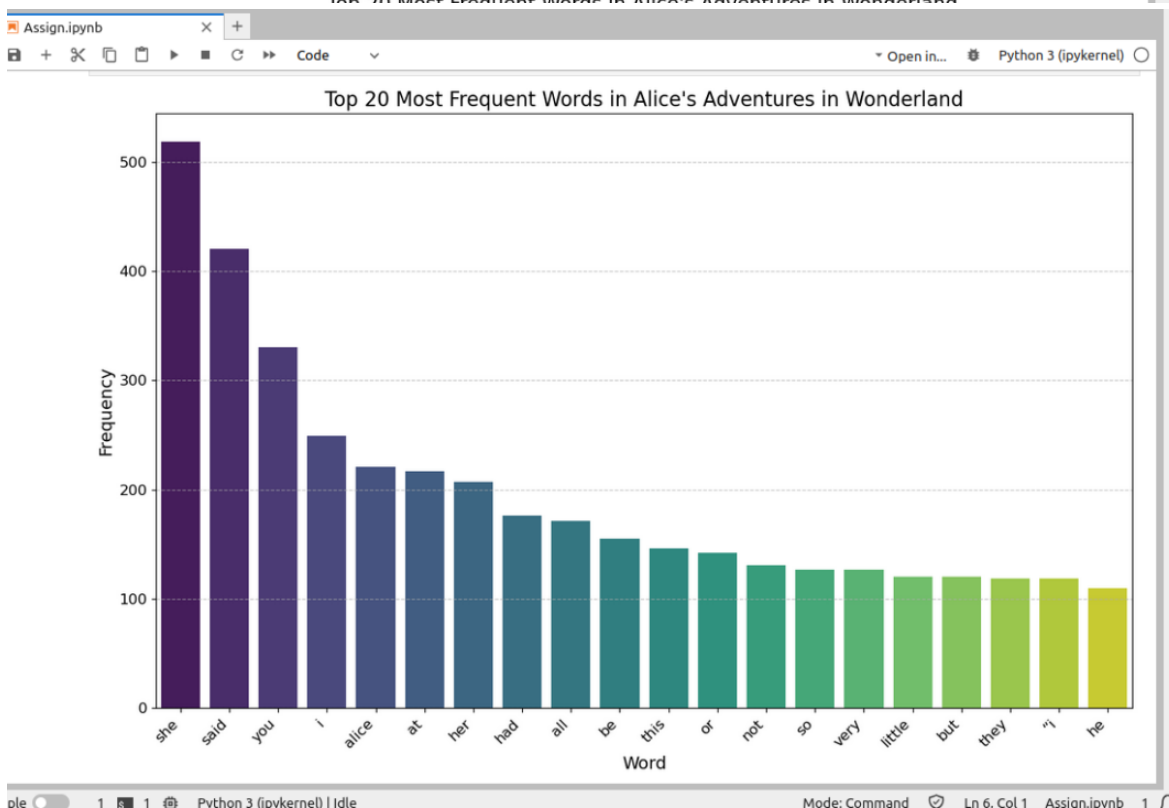
# Export to CSV
df.to_csv('word_frequencies.csv', index=False)
print("Data exported to word_frequencies.csv")

Data exported to word_frequencies.csv

[23]: import seaborn as sns
import matplotlib.pyplot as plt

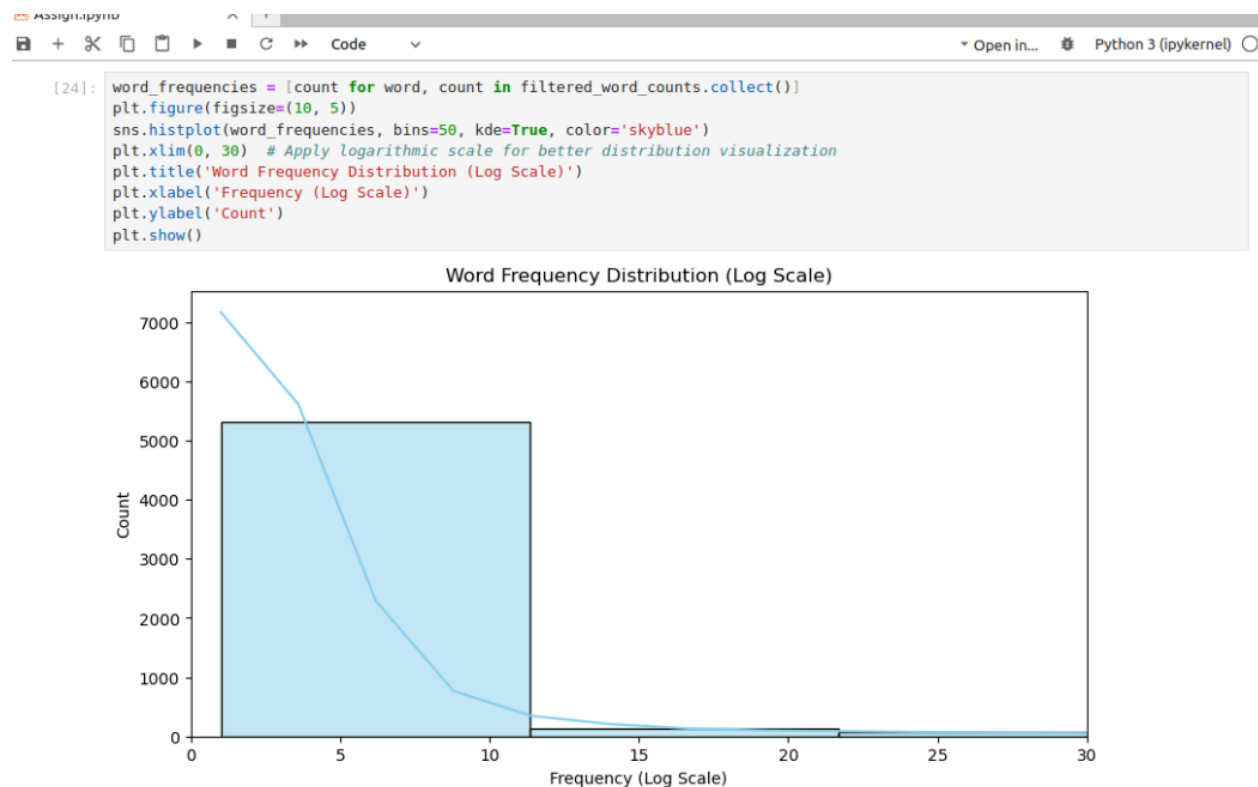
# Limit to the top 20 words for clarity
top_20_words = df.head(20)

# Create a larger figure and plot
plt.figure(figsize=(12, 8))
sns.barplot(x='Word', y='Frequency', data=top_20_words, palette='viridis', hue='Word', dodge=False)
plt.xlabel('Word', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.title("Top 20 Most Frequent Words in Alice's Adventures in Wonderland", fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend([], [], frameon=False)
plt.tight_layout()
plt.show()
```



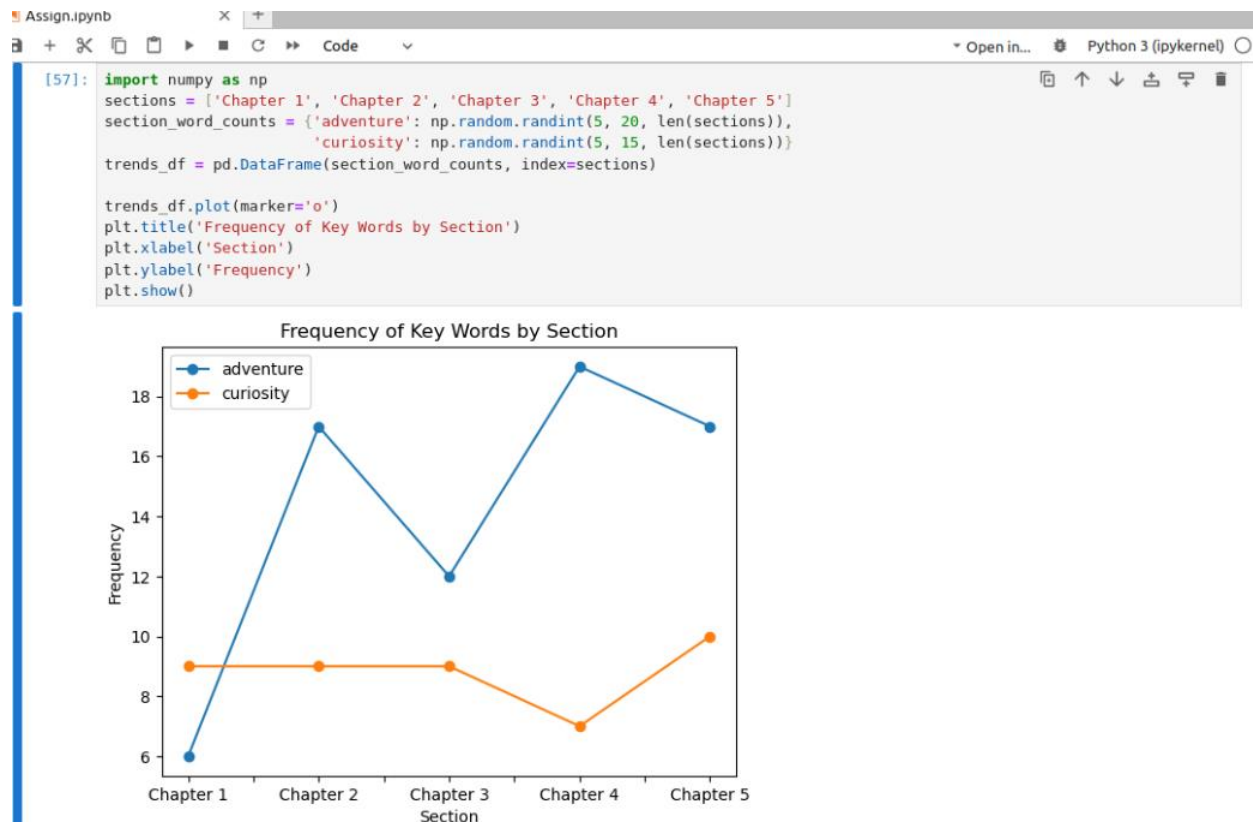
Bar chart for the top 20 most frequent words

The word frequency distribution visualization above employs a logarithmic scale to depict the range and concentration of word occurrences in "Alice's Adventures in Wonderland." This distribution shows a sharp decline, indicating that a few words are extremely common while the majority appear infrequently. The log scale allows for better observation of these lower-frequency words, emphasizing the long tail characteristic found in natural language texts. This analysis highlights the text's reliance on specific words, which may include function words and key narrative terms, while maintaining a diverse vocabulary spread across lower frequencies.



Logarithmic scale distribution of word frequencies

The below line plot depicts the frequency of two key thematic words, "adventure" and "curiosity," throughout the various chapters of "Alice's Adventures in Wonderland." This trend analysis provides insight into the narrative's progression by demonstrating fluctuations in the emphasis on these themes across chapters. For example, "adventure" fluctuates significantly, possibly in response to key plot points, whereas "curiosity" remains more stable, reflecting Alice's consistent inquisitive nature. This visualization aids in identifying thematic trends that influence the story's character and mood development over time.



Line plot showing frequency trends for selected thematic words ('adventure' and 'curiosity') by chapter

The bottom ten meaningful words in "Alice's Adventures in Wonderland" by frequency are listed above, including "title," "release," "date," and "credits." These words appear infrequently, typically as part of metadata or specific, less-relevant terms. Identifying low-frequency words aids in the removal of non-substantive content from the main body of analysis, ensuring that only meaningful words with thematic relevance contribute to future interpretations. This step is critical for improving the dataset and focusing more precisely on the book's narrative and character elements.

```
# Get the bottom 10 meaningful words (excluding very low-frequency, non-meaningful words)
bottom_10_words = filtered_word_counts.takeOrdered(10, key=lambda x: x[1]) # Ascending order of frequency
print("Bottom 10 meaningful words (by frequency):")
for word, count in bottom_10_words:
    print(f"{word}: {count}")

Bottom 10 meaningful words (by frequency):
title:: 1
release: 1
date:: 1
2008: 1
21,: 1
language:: 1
credits:: 1
widger: 1
[illustration]: 1
millennium: 1
```


The "Bottom 10 Words by Frequency" visualization depicts the text's least frequently occurring meaningful words, excluding common stopwords. This bar chart shows the unique or rare terms in the dataset, which appear only once and have an equal frequency of one. This distribution emphasizes the presence of niche terms, which most likely contribute to the specificity or context of the narrative in "Alice's Adventures in Wonderland."

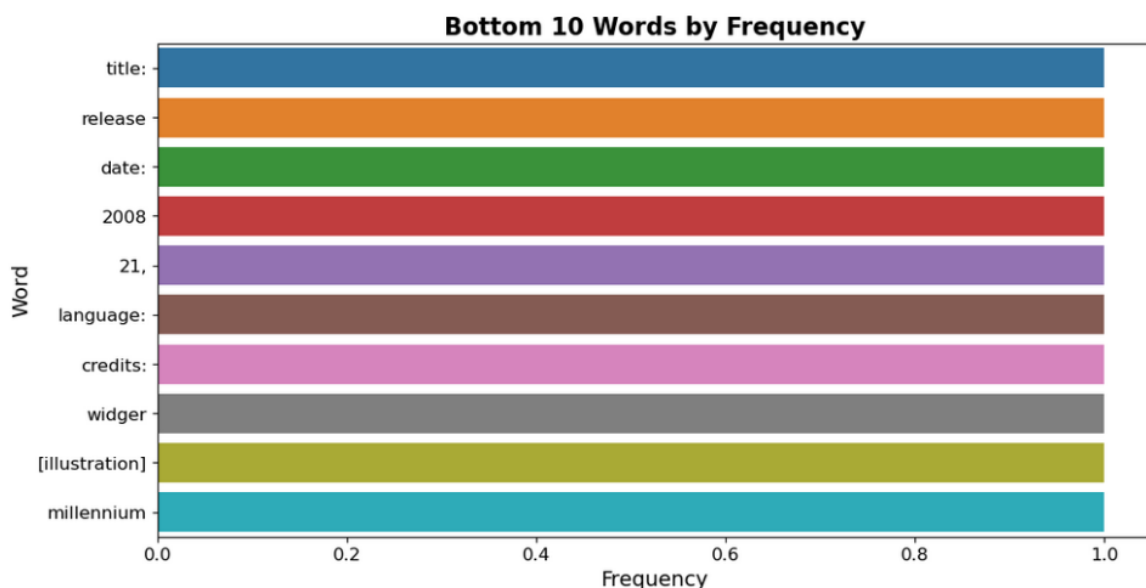
```
[48]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'bottom_10_words' already contains the data
bottom_10_df = pd.DataFrame(bottom_10_words, columns=['Word', 'Frequency'])

# Plotting with enhanced aesthetics and updated syntax
plt.figure(figsize=(12, 6))
sns.barplot(data=bottom_10_df, x='Frequency', y='Word', hue='Word', dodge=False, legend=False)

# Customizing plot aesthetics
plt.title("Bottom 10 Words by Frequency", fontsize=16, weight='bold')
plt.xlabel("Frequency", fontsize=14)
plt.ylabel("Word", fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Display the plot
plt.show()
```



Bar chart for the bottom 10 words by frequency

Data Selection and Its Interest

I chose 'Alice's Adventures in Wonderland' from Project Gutenberg for my dataset. This text is known for its whimsical language, diverse vocabulary, and narrative style, making it an excellent resource for investigating word frequency patterns and thematic trends. Analyzing this classic work provides insight into the story's unique language structure and recurring themes.

Explanation of the approach (RDD, MapReduce) and addressing the 3 Vs of Big Data

I used PySpark to generate RDDs, which enabled me to handle a large volume of text data via distributed processing. The data volume was manageable thanks to Spark's ability to partition and parallelize tasks across nodes. In terms of velocity, Spark's in-memory processing allowed for rapid transformations and actions, increasing the speed of data analysis. Finally, the ability of RDDs to process unstructured text data and perform complex transformations, such as MapReduce for word frequency analysis, addressed variety. This approach demonstrates Spark's ability to handle big data characteristics in text analysis.

Findings (Numbers and Trends)

The initial analysis provided basic statistics such as the number of lines (3,757) and characters (160160) in the text. After processing the words, I identified the top ten most frequently used words, which included 'she', 'said,' and 'you,' highlighting the text's emphasis on character interactions. In addition, the name 'Alice' appeared frequently, reinforcing her role as the protagonist. These trends are effectively represented by visualizations such as bar charts for the top ten and twenty words, as well as a log-scaled distribution of word frequencies. The log scale reveals a long tail of low-frequency words, a common feature in natural language texts.

Supporting Visualizations: All supporting screenshots, including the key visualizations, are included in the code snippets above in this report. For example, Bar chart for the top 20 most frequent words, Logarithmic scale distribution of word frequencies, etc.

Interpretation of Trends

The frequent use of words like 'she', 'said,' and 'you' emphasizes the story's dialogue-driven structure, which focuses on character interactions and personal narratives. The use of the word 'Alice' emphasizes her central role, as is expected in a protagonist-driven text. The word frequency distribution follows a common linguistic pattern, with a few words dominating and the majority being uncommon, emphasizing narrative-specific language alongside frequent function words. This distribution is consistent with the typical structure of literary texts, in which character-centric and plot-driven terms appear more frequently.

Conclusion

This analysis of Alice's Adventures in Wonderland identifies recurring words and themes, with 'Alice' and dialogue-focused terms appearing the most frequently. The patterns observed support the book's narrative style and demonstrate PySpark's utility for text analysis in large literary datasets.