Part I. Text Analysis

A. Overview of Tools for Text Analysis

Some popular tools and libraries for text analysis include:

- **NLTK** (Natural Language Toolkit) – A Python library for processing textual data.
- **spaCy** – A fast, production-ready NLP library in Python.
- **TextBlob** – Simplified text processing for Python.
- **Gensim** – For topic modeling and similarity detection.
- **Stanford CoreNLP** – A suite of NLP tools written in Java.
- **Voyant Tools** – A web-based text reading and analysis environment.
- **AntConc** – A freeware corpus analysis toolkit.
- **LIWC** – A tool for linguistic inquiry and word count analysis.
- **Mallet** – For topic modeling and clustering.
- **RapidMiner/KNIME/Orange** – Platforms that offer text mining modules.

In this assignment, we will use **NLTK** for static and dynamic text analysis.

B. Static & Dynamic Text Analysis Example

1. Research Question / Application

Example Research Question:
"How does the frequency of common words vary across different novel sections?"

Application:
Understanding the distribution of keywords (e.g., common words or keywords of interest) across a text can help reveal shifts in themes, narrative focus, or stylistic changes.

2. Choose Method

- **Static Analysis:**
  - Calculate overall word frequency in the entire text.
  - Visualize the top N words using a bar plot.
- **Dynamic Analysis:**
  - Divide the text into segments (e.g., chapters or fixed-size blocks of sentences).
  - Track the frequency of a target word (or a set of words) across these segments.
  - Plot the frequency trends over segments to observe changes over time.

Expected Outcome:

- A **static summary** (frequency distribution of words) that shows the overall most common words in the text.
- A **dynamic visualization** (line plot) showing how the frequency of a specific word (or words) changes as you progress through the text.

3. Choose the Tool: NLTK

Why NLTK?

- **NLTK** is a widely used Python library for natural language processing.
- It offers powerful tokenization, frequency distribution, and text processing functions.
- It can be easily integrated with visualization libraries like Matplotlib to produce static and dynamic plots.

4. Do Analysis by Uploading Text Data

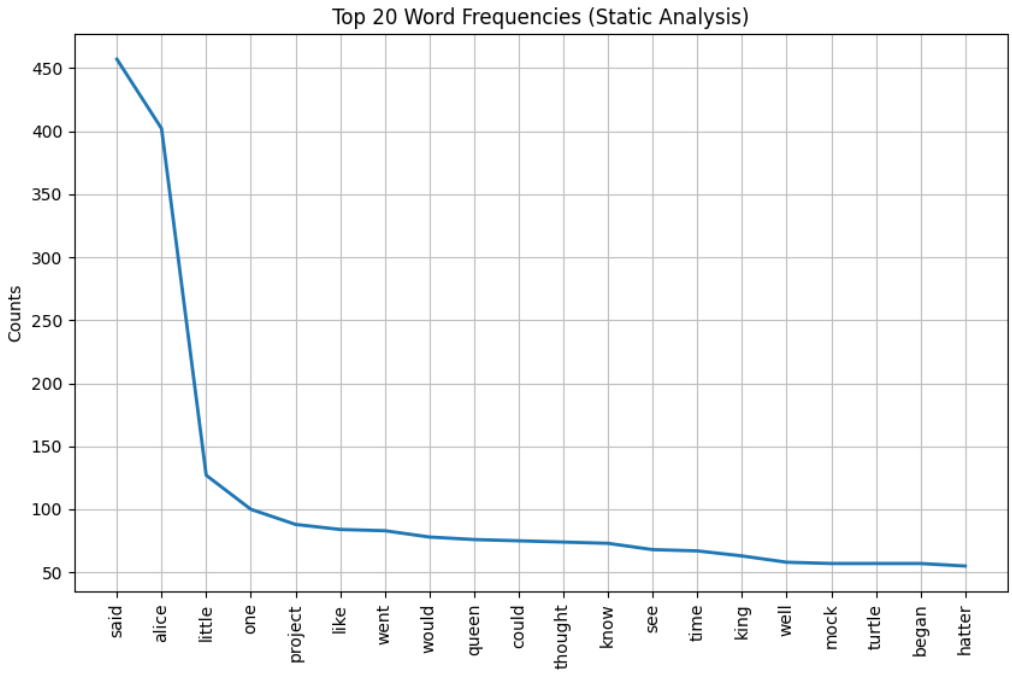
Text data link: https://www.gutenberg.org/cache/epub/11/pg11.txt

1) Static Text Analysis Code:

```python
import nltk
from nltk import FreqDist, word_tokenize, sent_tokenize
import matplotlib.pyplot as plt
import string

nltk.download('punkt')
nltk.download('stopwords')

file_path = r"C:/Users/91702/OneDrive\Desktop\SMA\ALice's_Adventure_In_The_Wonderland.txt"

with open(file_path, "r", encoding="utf-8") as file:
    text = file.read()


words = word_tokenize(text)
sentences = sent_tokenize(text)

words = [word.lower() for word in words if word.isalnum()]

stop_words = set(nltk.corpus.stopwords.words('english'))
filtered_words = [word for word in words if word not in stop_words]

# ---- Static Text Analysis ----

# Compute the overall frequency distribution of words
freq_dist = FreqDist(filtered_words)

# Print the 20 most common words
print("Top 20 most common words:")
for word, count in freq_dist.most_common(20):
    print(f"{word}: {count}")

# Visualize the top 20 words
plt.figure(figsize=(10, 6))
freq_dist.plot(20, title="Top 20 Word Frequencies (Static Analysis)")
plt.savefig("static_analysis.png")
plt.show()
```

Static Text Analysis Output (Most Common Words):

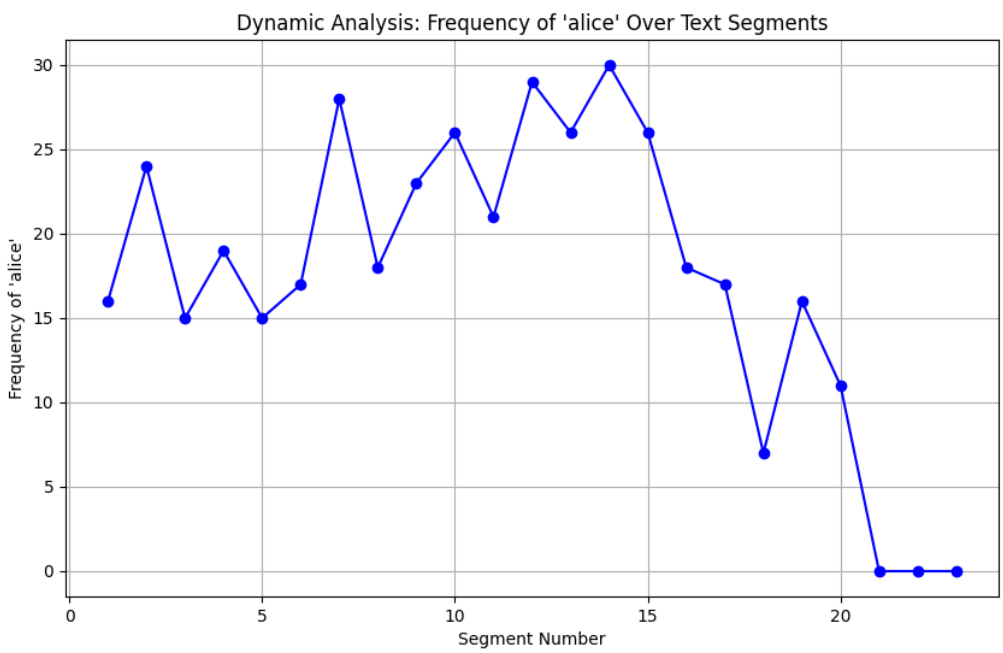
```
PS C:\Users\91702\OneDrive\Desktop\SMA> python textanalysis.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91702\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\91702\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Top 20 most common words:
said: 457
alice: 402
little: 127
one: 100
project: 88
like: 84
went: 83
would: 78
queen: 76
could: 75
thought: 74
know: 73
see: 68
time: 67
king: 63
well: 58
mock: 57
turtle: 57
began: 57
hatter: 55
```



2) **Dynamic Text Analysis Code:**

```python
# ---- Dynamic Text Analysis ----

# Analyze how a target word changes across the text
segment_size = 50
num_sentences = len(sentences)
segments = [sentences[i:i + segment_size] for i in range(0, num_sentences, segment_size)]

# Choose a target word to track
target_word = "alice"
dynamic_freq = []

# Calculate the frequency of the target word in each segment
for idx, segment in enumerate(segments, 1):
    segment_text = " ".join(segment)
    segment_words = word_tokenize(segment_text)
    segment_words = [word.lower() for word in segment_words if word.isalnum()]  # Clean words
    count = segment_words.count(target_word)
    dynamic_freq.append(count)
    print(f"Segment {idx}: Frequency of '{target_word}' = {count}")

# Plot the frequency of the target word over segments
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(dynamic_freq) + 1), dynamic_freq, marker='o', linestyle='-', color='b')
plt.xlabel("Segment Number")
plt.ylabel(f"Frequency of '{target_word}'")
plt.title(f"Dynamic Analysis: Frequency of '{target_word}' Over Text Segments")
plt.grid(True)
plt.savefig("dynamic_analysis.png")
plt.show()
```

Dynamic Text Analysis Output:



Part II. Hyperlink Analysis

A. Overview of Tools for Hyperlink Analysis

Some popular tools and software for hyperlink and network analysis include:

- **Gephi** – An open-source network visualization and analysis tool.
- **NodeXL** – A network analysis tool that works as an Excel add-in.
- **Cytoscape** – A software platform for visualizing complex networks.
- **Pajek** – A program for large network analysis.
- **UCINET** – Software for social network analysis.
- **SocNetV** – A tool for social network visualization.
- **Netlytic** – For analyzing social media networks.
- **Graph-tool** – A Python library for network analysis.
- **Linkurious** and **Maltego** – For investigative and forensic link analysis.

B. Static & Dynamic Hyperlink Analysis Example

1. Purpose of the Analysis

Hyperlink analysis is used to examine the structure of web links to gain insights into:

- Identifying **influential webpages** within Wikipedia
- Understanding the **interlinking structure** of Wikipedia articles
- Mapping **the most referenced Wikipedia pages**
- Analyzing **how topics are connected** through hyperlinks

2. Defining the Network

A hyperlink network consists of **nodes (Wikipedia pages)** and **edges (hyperlinks between them)**. It can be represented as:

- **Directed Graph:** Links have direction (e.g., Page A → Page B means Page A links to Page B)
- **Weighted Graph:** Some links have more importance than others based on frequency
- **Community Graph:** Groups of pages closely linked together

3. Tools for Hyperlink Analysis

For this assignment, we will use:

- **BeautifulSoup (Python):** To scrape hyperlinks from Wikipedia
- **NetworkX (Python):** To build and analyze the hyperlink network
- **Matplotlib:** To visualize the network structure

4. Scraping Wikipedia Hyperlinks

We will scrape hyperlinks from the Wikipedia page on **Web Scraping** (https://en.wikipedia.org/wiki/Web_scraping).

Web Scraping with BeautifulSoup:

hyperlinkanalysis.py

```python
from bs4 import BeautifulSoup
import requests
import pandas as pd

url = "https://en.wikipedia.org/wiki/Web_scraping"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Extract all hyperlinks
links = [a['href'] for a in soup.find_all('a', href=True) if a['href'].startswith("/wiki/")]

# Convert to full URLs
base_url = "https://en.wikipedia.org"
full_links = [base_url + link for link in links]

# Save to CSV
df = pd.DataFrame(full_links, columns=["target"])
df.insert(0, "source", url)
df.to_csv("wikipedia_hyperlinks.csv", index=False)

print("Scraped and saved", len(full_links), "links.")
```

Link for the csv file created:

https://github.com/Tavishaa/SMA-analysis/blob/main/wikipedia_hyperlinks.csv

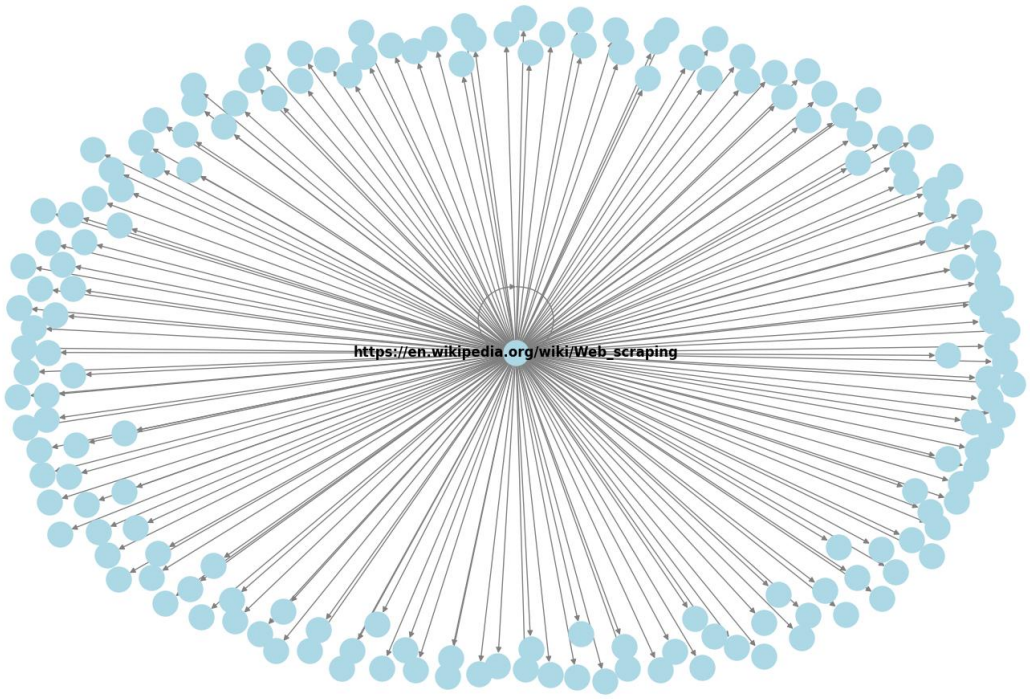
5. Performing Static and Dynamic Analysis

Static Analysis (Using NetworkX)

```python
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd

# Load hyperlink data
hyperlinks = pd.read_csv('wikipedia_hyperlinks.csv')

# Create a directed graph
G = nx.DiGraph()
for index, row in hyperlinks.iterrows():
    G.add_edge(row['source'], row['target'])

# Compute PageRank (measures influence of Wikipedia pages)
pagerank = nx.pagerank(G)
print("Top 10 Most Influential Wikipedia Pages:")
for page, rank in sorted(pagerank.items(), key=lambda x: x[1], reverse=True)[:10]:
    print(page)
    print(rank)
    print("------------------")

# Network visualization
plt.figure(figsize=(15, 10))
pos = nx.spring_layout(G, k=0.15)
nx.draw(G, pos, with_labels=False, node_size=500, node_color="lightblue", edge_color="gray", font_size=10, font_weight='bold')

central_node = max(pagerank, key=pagerank.get)
nx.draw_networkx_labels(G, pos, labels={central_node: central_node}, font_size=12, font_weight='bold', font_color='black')

plt.title("Wikipedia Hyperlink Network Structure", fontsize=14, fontweight='bold')
plt.savefig("static_hyperlink.png")
plt.show()
```

Output:

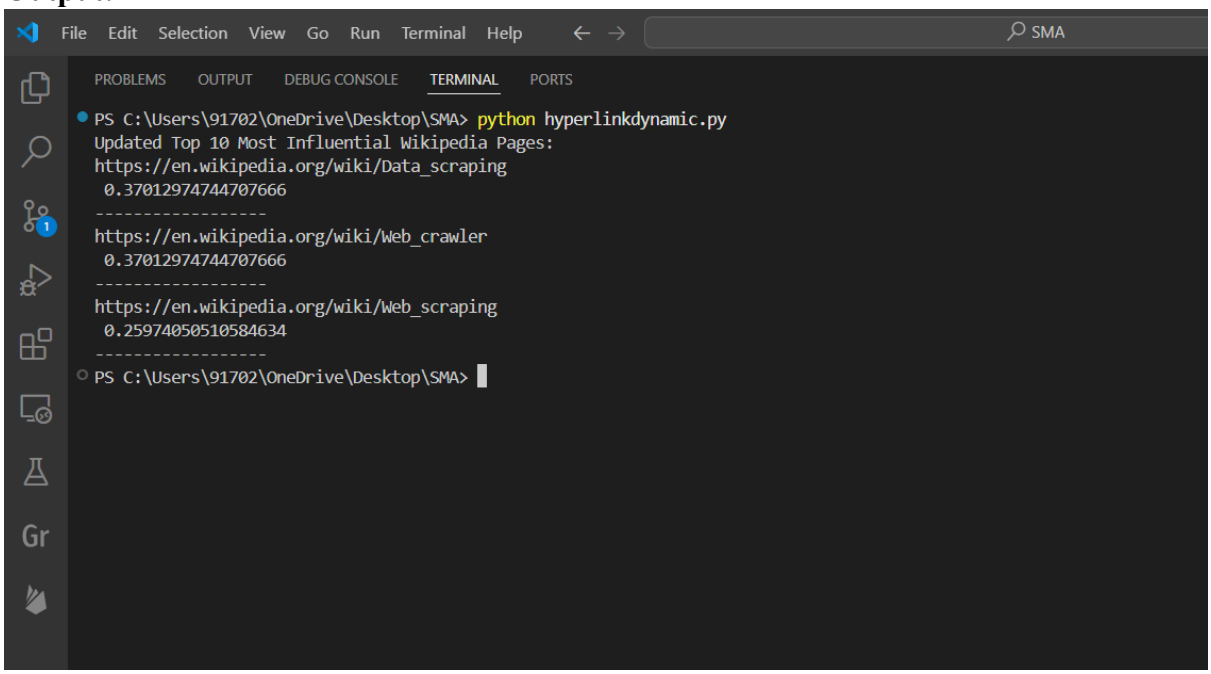
```
PS C:\Users\91702\OneDrive\Desktop\SMA> python hyperlinkstatic.py
Top 10 Most Influential Wikipedia Pages:
https://en.wikipedia.org/wiki/Web_scraping
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Main_Page
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Wikipedia:Contents
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Portal:Current_events
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Special:Random
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Wikipedia:About
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Help:Contents
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Help:Introduction
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Wikipedia:Community_portal
0.005780346820809269
------------------
https://en.wikipedia.org/wiki/Special:RecentChanges
0.005780346820809269
------------------
```



Dynamic Analysis (Tracking Changes Over Time)

```python
import networkx as nx
import matplotlib.pyplot as plt

# Function to update network over time
def update_network(graph, new_links):
    for src, tgt in new_links:
        graph.add_edge(src, tgt)
    return graph

# Ensure 'G' is defined before adding new links
if 'G' not in globals():
    G = nx.DiGraph()

# Simulating new links being added dynamically
new_links = [("https://en.wikipedia.org/wiki/Web_scraping", "https://en.wikipedia.org/wiki/Data_scraping"),
             ("https://en.wikipedia.org/wiki/Web_scraping", "https://en.wikipedia.org/wiki/Web_crawler")]
G = update_network(G, new_links)

# Recompute PageRank dynamically
pagerank_updated = nx.pagerank(G)
print("Updated Top 10 Most Influential Wikipedia Pages:")
for page, rank in sorted(pagerank_updated.items(), key=lambda x: x[1], reverse=True)[:10]:
    print(page, "\n", rank, "\n-----------------")

# Improved Graph Visualization
plt.figure(figsize=(15, 10))
pos = nx.spring_layout(G, k=0.3)  # Adjust k for better spacing
nx.draw(G, pos, with_labels=False, node_size=500, node_color="lightcoral", edge_color="gray", alpha=0.6)

# Label only the central node in the updated graph
central_node_updated = max(pagerank_updated, key=pagerank_updated.get)
nx.draw_networkx_labels(G, pos, labels={central_node_updated: central_node_updated},
                        font_size=10, font_weight='bold', font_color='red',
                        bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))

plt.title("Updated Wikipedia Hyperlink Network Structure", fontsize=14, fontweight='bold')
plt.savefig("dynamic_hyperlink.png", bbox_inches="tight")  # Save graph properly
```

Output:



Updated Wikipedia Hyperlink Network Structure