TASK 2

WEEK 2nd

TOPIC:

NLTK- Powered Text Analytics Web App (Flask/Streamlit + pandas)

Created by KIRTI BALA

Overview

Goal: Build a web application for text analytics where users can upload their own text corpus (e.g., reviews, tweets, documents) and analyze it interactively using NLTK and pandas. The application should:

- Preprocess text (tokenization, stop-word removal, lemmatization).
- Perform POS tagging, frequency distribution, collocation analysis, and sentiment analysis.
- Provide visual insights like N-gram plots and sentiment trends.
- Enable dynamic data exploration and an analysis dashboard.

Tech Stack:

- Backend / Processing: Python, NLTK, pandas
- Frontend / UI: Flask (HTML/CSS + Jinja2) or Streamlit (simpler, single file)

- Visualization: matplotlib or plotly (interactive)
- File handling: Pandas + Flask/Streamlit upload feature

2.Project Structure

nltk_text_analytics/

----app.py Or sttreamlit_app.py

# main web app file

----nlp_pipeline.py

# core text preprocessing & analysis logic

----static (Flask only)

----templates/

#HTML templates (Flask only)

```
----data / ---- example_dataset.txt
----requirements.txt
----README.md
```

3. Core Functionalities

A. Data Upload & Storage

Allow users to upload **.txt** files.

Read text and store as pandas Data Frame with columns like:

- sentence
- tokens
- clean_tokens
- pos_tags
- sentiment_score

B. Preprocessing (in nlp_pipeline.py)

1. Cleaning

Lowercasing, removing punctuation, numbers.

2. Tokenization

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
```

3. Stopword Removal

```
from nltk.corpus import stopwords
tokens = [t for t in tokens if t not in
stopwords.words('english')]
```

4. Lemmatization

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokens = [lemmatizer.lemmatize(t) for t in
tokens]
```

5. POS Tagging

```
from nltk import pos_tag
pos_tags = pos_tag(tokens)
```

C. Analysis Features

1. Frequency Distribution

- Use nltk.FreqDist to get top N words.
- Visualize with matplotlib/plotly bar chart.

2. Collocations

- Use BigramCollocationFinder or Trigram Collocation Finder.

3. Sentiment Analysis

- Use VADER SentimentIntensity Analyzer (from nltk.sentiment.vader).
- Compute polarity scores (positive, negative, compound).
- Plot sentiment trend per sentence.

4. N-Gram Visualization

- Generate top bigrams/trigrams and visualize.

4. Web App UI Pages

Page 1: Data Explorer

File upload section.

Display raw text & processed tokens in a table.

Show word counts and basic statistics.

Page 2: Analysis Dashboard

Visualizations:

Top N words frequency

Sentiment trend over time

Collocations

Option to filter by POS (e.g., only nouns, verbs).

5 Implementation

Code Structure

nlp_pipeline.py

It handles preprocessing and analytics functions.

```
import pandas as pd

import nltk

from nltk.corpus import stopwords

from nlt

k.stem import WordNetLemmatizer

from nltk import word_tokenize, pos_tag,
FreqDist, bigrams

from nltk.sentiment.vader import
SentimentIntensityAnalyzer

import string
```

```
# Ensure resources downloaded
Nltk.download('punkt')
Nltk.download('stopwords')
Nltk.download('wordnet')
Nltk.download('averaged_perceptron_tag
ger')
Nltk.download('vader_lexicon')

stop_words =
set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
sia = SentimentIntensityAnalyzer()

def preprocess_text(text: str) ->
pd.DataFrame:
    # Lowercase and remove punctuation
```

```
Text=text.lower().translate(str.maketrans(''
, '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in
stop_words and t.isalpha()]
    lemmas = [lemmatizer.lemmatize(t) for t
in tokens]
    pos_tags = pos_tag(lemmas)
    return pd.DataFrame(pos_tags,
columns=['token', 'POS'])


def get_freq_dist(df: pd.DataFrame,
top_n=20):
    freq = FreqDist(df['token'])
    return freq.most_common(top_n)
```

```
def get_collocations(df: pd.DataFrame,
n=2, top_n=20):
    if n == 2:
        bigram_list = list(bigrams(df['token']))
        freq = FreqDist(bigram_list)
    else:
        # implement trigram
        from nltk import trigrams
        trigram_list = list(trigrams(df['token']))
        freq = FreqDist(trigram_list)
    return freq.most_common(top_n)


def sentiment_analysis(text: str):
    sentences = nltk.sent_tokenize(text)
    sentiment_scores =
[sia.polarity_scores(sent) for sent in
sentences]
```

Return
 pd.DataFrame(sentiment_scores)
B. Streamlit Version (streamlit_app.py)

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
from nlp_pipeline import preprocess_text, get_frequency_distribution

st.title("NLTK-Powered Text Analytics")

# File upload
uploaded_file = st.file_uploader("Upload a text file", type=["txt"])
```

```
if uploaded_file is not None:
    text = uploaded_file.read().decode("utf-
8")

    # Preprocess
    df = preprocess_text(text)
    st.subheader("Data Explorer")
    st.dataframe(df[['sentence',
'clean_tokens', 'sentiment_score']])

    # Frequency distribution
    all_tokens = [token for tokens in
df['clean_tokens'] for token in tokens]
    freq =
get_frequency_distribution(all_tokens)

    # Plot frequency
```

```
  st.subheader("Top Words Frequency")
   words, counts = zip(*freq)
   fig, ax = plt.subplots()
   ax.bar(words, counts)
   plt.xticks(rotation=45)
   st.pyplot(fig)

   # Sentiment trend
   st.subheader("Sentiment Trend")
   fig2, ax2 = plt.subplots()
   df['sentiment_score'].plot(kind='line',
ax=ax2)
   st.pyplot
```

C. Flask Version (app.py)

Two routes:

/upload → Upload file, display Data Explorer

/analysis → Show frequency plots, sentiment trends

Uses render_template() for HTML pages and matplotlib/plotly for charts.

6. Example Dataset

example_dataset.txt:

I loved the movie! The acting was brilliant.
The storyline was dull and predictable.

Overall, I would recommend it to my friends.

7. Output Screenshots (What to Show)

Data Explorer Page: Table showing sentence, tokens, clean tokens, sentiment score.

Analysis Dashboard:

Bar chart of top 20 frequent words.

Line chart of sentiment scores over sentences.

Collocations list or chart.

8. User Guide (README)

Steps to Run Locally:

Delivaralbles:

1. Clone repo or download ZIP.

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Run Streamlit app:

```
streamlit run streamlit_app.py
```

OR Flask app:

```
python app.py
```

4. Open browser at http://localhost:8501 (Streamlit) or http://127.0.0.1:5000 (Flask).

5. Upload .txt file and explore results.

6.. Extensions (Optional)

Add word cloud visualization (using word cloud library).

Support CSV upload (multiple text rows).

Implement topic modeling (LDA with gensim).

Add downloadable report (PDF/CSV of analysis).

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize, pos_tag,
FreqDist, bigrams
from nltk.sentiment.vader import
SentimentIntensityAnalyzer
import string

# Ensure resources downloaded
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('vader_lexicon')
```

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
sia = SentimentIntensityAnalyzer()


def preprocess_text(text: str) ->
pd.DataFrame:
    # Lowercase and remove punctuation
    text = text.lower().translate(str.maketrans('',
'', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in
stop_words and t.isalpha()]
    lemmas = [lemmatizer.lemmatize(t) for t in
tokens]
    pos_tags = pos_tag(lemmas)
    return pd.DataFrame(pos_tags,
columns=['token', 'POS'])
```

```
def get_freq_dist(df: pd.DataFrame,
top_n=20):
    freq = FreqDist(df['token'])
    return freq.most_common(top_n)


def get_collocations(df: pd.DataFrame, n=2,
top_n=20):
    if n == 2:
        bigram_list = list(bigrams(df['token']))
        freq = FreqDist(bigram_list)
    else:
        # implement trigram
        from nltk import trigrams
        trigram_list = list(trigrams(df['token']))
        freq = FreqDist(trigram_list)
    return freq.most_common(top_n)
```

```
def sentiment_analysis(text: str):
    sentences = nltk.sent_tokenize(text)
    sentiment_scores =
[sia.polarity_scores(sent) for sent in
sentences]
    return pd.DataFrame(sentiment_scores)
```