Name: Atharva Admile

Batch no.06

**Task 1: Write a short Python script using any open dataset (e.g., movie reviews, tweets) to demonstrate one real-world NLP application:**

• Sentiment analysis (positive/negative)

```
import pandas as pd
import re

df = pd.read_csv('review_dataset.csv')

positive_words = ['good', 'great', 'excellent', 'love', 'amazing', 'fantastic']

negative_words = ['bad', 'terrible', 'hate', 'boring', 'awful', 'disappointing']

def sentiment_analysis(text):
    if pd.isna(text):
        return "Neutral"
    words = re.findall(r'\b\w+\b', text.lower())
    pos_count = sum(1 for word in words if word in positive_words)
    neg_count = sum(1 for word in words if word in negative_words)
    if pos_count > neg_count:
        return "Positive"
    elif neg_count > pos_count:
        return "Negative"
    else:
        return "Neutral"

# Apply to first 5 reviews
for i, row in df.head(5).iterrows():
    print(f"Review {i}: {row['reviews.text'][:100]}...")
    print(f"Sentiment: {sentiment_analysis(row['reviews.text'])}\n")
```

```
Review 0: I initially had trouble deciding between the paperwhite and the voyage because reviews more or less ...
Sentiment: Positive

Review 1: Allow me to preface this with a little history. I am (was) a casual reader who owned a Nook Simple T...
Sentiment: Positive

Review 2: I am enjoying it so far. Great for reading. Had the original Fire since 2012. The Fire used to make ...
Sentiment: Positive

Review 3: I bought one of the first Paperwhites and have been very pleased with it its been a constant compani...
Sentiment: Positive

Review 4: I have to say upfront - I don't like coroporate, hermetically closed stuff like anything by Apple or...
Sentiment: Neutral
```

**Text summarization**

```
import pandas as pd
import re

df = pd.read_csv('review_dataset.csv')

def text_summarization(text):
    if pd.isna(text):
        return ""
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text)
    sentences = [s.strip() for s in sentences if s.strip()]
    return ' '.join(sentences[:3])

# Apply to first review
text = df['reviews.text'].iloc[0]
print("Original (first 200 chars):", text[:200] + "...")
print("Summary:", text_summarization(text))
```

```
Original (first 200 chars): I initially had trouble deciding between the paperwhite and the voyage because review
Summary: I initially had trouble deciding between the paperwhite and the voyage because reviews more or less said
```

Text Data Basics

**Task 2: Write a Python function that takes a paragraph and outputs:**

• List of sentences

• List of tokens (words)

• Count of tokens, sentences, paragraphs

```
import pandas as pd
import re

def process_paragraph(paragraph):
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', paragraph)
    sentences = [s.strip() for s in sentences if s.strip()]
    tokens = re.findall(r'\b\w+\b', paragraph)
    num_sentences = len(sentences)
    num_tokens = len(tokens)
    num_paragraphs = 1
    return {
        'sentences': sentences,
        'tokens': tokens,
        'counts': {
            'tokens': num_tokens,
            'sentences': num_sentences,
            'paragraphs': num_paragraphs
        }
    }

df = pd.read_csv('review_dataset.csv')
paragraph = df['reviews.text'].iloc[0]
result = process_paragraph(paragraph)
print("Sentences:", result['sentences'][:3], "...")
print("Tokens:", result['tokens'][:10], "...")
print("Counts:", result['counts'])
```

```
Sentences: ["I initially had trouble deciding between the paperwhite and the voyage because reviews more or less
Tokens: ['I', 'initially', 'had', 'trouble', 'deciding', 'between', 'the', 'paperwhite', 'and', 'the'] ...
Counts: {'tokens': 202, 'sentences': 3, 'paragraphs': 1}
```

Text Preprocessing

**A. Tokenization Task: Tokenize a text document using simple split vs. regex tokenizer.**

Compare outputs.

```
import pandas as pd
import re

df = pd.read_csv('review_dataset.csv')
text = df['reviews.text'].iloc[0][:200]
print("Text:", text)
tokens_split = text.split()
tokens_re = re.findall(r'\b\w+\b', text)
print("Split tokens:", tokens_split)
print("Regex tokens:", tokens_re)
print("Difference: Split includes punctuation attached, regex removes non-word chars.")
```

```
Text: I initially had trouble deciding between the paperwhite and the voyage because reviews more or less said th
Split tokens: ['I', 'initially', 'had', 'trouble', 'deciding', 'between', 'the', 'paperwhite', 'and', 'the', 'voy
Regex tokens: ['I', 'initially', 'had', 'trouble', 'deciding', 'between', 'the', 'paperwhite', 'and', 'the', 'voy
Difference: Split includes punctuation attached, regex removes non-word chars.
```

B. Stopwords Removal

**Task: Remove stopwords from a text and print before vs after**

```
import pandas as pd
import re

df = pd.read_csv('review_dataset.csv')
stopwords = ['the', 'is', 'a', 'an', 'and', 'of', 'to', 'in', 'for', 'on', 'i', 'my', 'with']
def remove_stopwords(text):
    tokens = re.findall(r'\b\w+\b', text.lower())
    filtered = [word for word in tokens if word not in stopwords]
    return ' '.join(filtered)
text = df['reviews.text'].iloc[0][:200]
print("Before:", text)
print("After:", remove_stopwords(text))
```

```
Before: I initially had trouble deciding between the paperwhite and the voyage because reviews more or less said
After: initially had trouble deciding between paperwhite voyage because reviews more or less said same thing pape
```

C. Lemmatization & Stemming

**Task: Apply simple rule-based stemmer and lemmatizer on a text. Show differences**

(e.g., “studies” → “studi” vs. “study”).

```
import re

def simple_stem(word):
    word = word.lower()
    if word.endswith('ies'):
        return word[:-3] + 'y'
    elif word.endswith('ing'):
        return word[:-3]
    elif word.endswith('ed'):
        return word[:-2]
    elif word.endswith('es'):
        return word[:-2]
    elif word.endswith('s'):
        return word[:-1]
    return word

def simple_lemma(word):
    # Very basic, e.g. studies -> study
    word = word.lower()
    if word == 'studies':
        return 'study'
    if word == 'studi':
        return 'study'  # after stem
    return word

text = "He studies hard and is studying now. Studied yesterday."
tokens = re.findall(r'\b\w+\b', text)
stemmed = [simple_stem(word) for word in tokens]
lemmatized = [simple_lemma(word) for word in tokens]
print("Original:", tokens)
print("Stemmed:", stemmed)
print("Lemmatized:", lemmatized)
print('Difference example: “studies” → “studi” (stem) vs. “study” (lemma)')
```

```
Original: ['He', 'studies', 'hard', 'and', 'is', 'studying', 'now', 'Studied', 'yesterday']
Stemmed: ['he', 'study', 'hard', 'and', 'i', 'study', 'now', 'studi', 'yesterday']
Lemmatized: ['he', 'study', 'hard', 'and', 'is', 'studying', 'now', 'studied', 'yesterday']
Difference example: “studies” → “studi” (stem) vs. “study” (lemma)
```

D. Handling punctuation, special characters, emojis

**Task: Clean text containing #hashtags, @mentions, !!!, 😊 using regex.**

```
import re

def clean_text(text):
    text = re.sub(r'#\w+', '', text)  # Hashtags
    text = re.sub(r'@\w+', '', text)  # Mentions
    text = re.sub(r'[^\w\s]', '', text)  # Punctuation and special chars
    text = re.sub(r'[\U0001F600-\U0001F6FF\U0001F300-\U0001F5FF]', '', text)  # Emojis
    return text.strip()

text = "Hello @user! #cool 😊 This is great!!!"
print("Before:", text)
print("After:", clean_text(text))
```

```
Before: Hello @user! #cool 😊 This is great!!!
After: Hello    This is great
```

E. Lowercasing & Normalization

**Task: Convert mixed-case text into lowercase and normalize spacing.**

```
import re

def normalize_text(text):
    text = text.lower()
    text = re.sub(r'\s+', ' ', text)  # Normalize multiple spaces
    return text.strip()

text = "This   IS Mixed Case   Text With   Extra Spaces."
print("Before:", text)
print("After:", normalize_text(text))
```

```
Before: This   IS Mixed Case   Text With   Extra Spaces.
After: this is mixed case text with extra spaces.
```

F. Regex for text cleaning

**Task: Extract all emails from a paragraph. Remove numbers from text. Replace multiple spaces with one.**

```
import re

def extract_emails(text):
    return re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text)

def remove_numbers(text):
    return re.sub(r'\d+', '', text)

def replace_multiple_spaces(text):
    return re.sub(r'\s+', ' ', text).strip()

paragraph = "Contact me at example@email.com or test123@example.com. The year is 2023.   Extra   spaces."
print("Emails:", extract_emails(paragraph))
cleaned = remove_numbers(paragraph)
print("Without numbers:", cleaned)
print("Normalized spaces:", replace_multiple_spaces(cleaned))
```

```
Emails: ['example@email.com', 'test123@example.com']
Without numbers: Contact me at example@email.com or test@example.com. The year is .   Extra   spaces.
Normalized spaces: Contact me at example@email.com or test@example.com. The year is . Extra spaces.
```

Bag-of-Words (BoW), TF-IDF

**Task: Convert a list of sentences (first 4 reviews) into a BoW matrix. Convert the same into a TF-IDF matrix. Compare word importance scores.**

```
import pandas as pd
import numpy as np
import re
from collections import Counter

df = pd.read_csv('review_dataset.csv')
sentences = df['reviews.text'].head(4).tolist()  # take 4 reviews
# Preprocess: lower, tokens
texts = []
for sent in sentences:
    tokens = re.findall(r'\b\w+\b', str(sent).lower())
    texts.append(tokens)
# Vocabulary
vocab = sorted(set(word for text in texts for word in text))
vocab_size = len(vocab)
# BoW matrix
bow_matrix = np.zeros((len(texts), vocab_size), dtype=int)
for i, text in enumerate(texts):
    counts = Counter(text)
    for word, count in counts.items():
        idx = vocab.index(word)
        bow_matrix[i, idx] = count
print("Vocabulary:", vocab[:10], "...")
print("BoW Matrix shape:", bow_matrix.shape)
print("Example BoW for first doc:", bow_matrix[0][:10])
# TF-IDF
# TF = bow / row sum
row_sums = np.sum(bow_matrix, axis=1, keepdims=True)
row_sums[row_sums == 0] = 1  # avoid div0
tf = bow_matrix / row_sums
# IDF = log(N / df) where df is num docs with word
N = len(texts)
dfreq = np.sum(bow_matrix > 0, axis=0)
idf = np.log(N / (dfreq + 1e-8))  # avoid div0
tfidf = tf * idf
print("TF-IDF Matrix shape:", tfidf.shape)
print("Example TF-IDF for first doc:", tfidf[0][:10])
```

```
Vocabulary: ['00', '139', '1984', '2011', '2012', '3', '300', '4', '400', '80'] ...
BoW Matrix shape: (4, 362)
Example BoW for first doc: [0 0 0 0 0 0 1 0 0 1]
TF-IDF Matrix shape: (4, 362)
Example TF-IDF for first doc: [0.         0.         0.         0.         0.         0.
 0.00686284 0.         0.         0.00686284]
```