Name:P.Srikar
Htno:2403A52002

```
import pandas as pd

df = pd.read_csv('/content/Twitter_Data.csv')
display(df.head())
```

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0

```
import re

def preprocess_text(text):
    # Ensure the input is a string
    if not isinstance(text, str):
        return '' # Or handle as appropriate, e.g., str(text)
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove mentions
    text = re.sub(r'@\w+', '', text)
    return text

# Convert the column to string type and fill NaN values before applying the function
df['clean_text'] = df['clean_text'].astype(str).apply(preprocess_text)
display(df.head())
```

	clean_text	category
0	when modi promised “minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0

Toggle Gemini

```
import nltk

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
import nltk

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
```

```
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
True
```

Toggle Gemini

```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

def pos_tag_text(text):
    if not isinstance(text, str):
        return [] # Return empty list for non-string inputs
    tokens = word_tokenize(text)
    return pos_tag(tokens)

df['pos_tags'] = df['clean_text'].apply(pos_tag_text)
display(df.head())
```

	clean_text	category	pos_tags
0	when modi promised “minimum government maximum...	-1.0	[(when, WRB), (modi, NN), (promised, VBD), (“,...
1	talk all the nonsense and continue all the dra...	0.0	[(talk, NN), (all, PDT), (the, DT), (nonsense,...
2	what did just say vote for modi welcome bjp t...	1.0	[(what, WP), (did, VBD), (just, RB), (say, VB)...
3	asking his supporters prefix chowkidar their n...	1.0	[(asking, VBG), (his, PRP$), (supporters, NNS)...
4	answer who among these the most powerful world...	1.0	[(answer, NN), (who, WP), (among, IN), (these,...

```
Start coding or generate with AI.
```

```
from collections import defaultdict

# Initialize dictionaries for counts
```

Toggle Gemini

```
tag_counts = defaultdict(lambda: defaultdict(int))
word_tag_counts = defaultdict(lambda: defaultdict(int))
trans_counts = defaultdict(lambda: defaultdict(int))

# Iterate through each tweet's POS tags to build counts
for tags_list in df['pos_tags']:
    if not tags_list:
        continue # Skip empty tag lists

    # Count initial tags (for sentence start probability, if needed, though not explicitly asked here)
    # We'll focus on emission and transition for now.

    previous_tag = None
    for word, current_tag in tags_list:
        # Emission counts: P(word|tag)
        word_tag_counts[current_tag][word] += 1
        tag_counts[current_tag]['total'] += 1 # Total count for the tag

        # Transition counts: P(tag_i | tag_i-1)
        if previous_tag:
            trans_counts[previous_tag][current_tag] += 1
        previous_tag = current_tag

# Calculate probabilities (without smoothing for now)
emission_probabilities = defaultdict(lambda: defaultdict(float))
transition_probabilities = defaultdict(lambda: defaultdict(float))

# Calculate Emission Probabilities P(word|tag)
for tag, word_counts in word_tag_counts.items():
    total_words_for_tag = tag_counts[tag]['total']
    if total_words_for_tag > 0:
        for word, count in word_counts.items():
            emission_probabilities[tag][word] = count / total_words_for_tag

# Calculate Transition Probabilities P(tag_i | tag_i-1)
for prev_tag, next_tag_counts in trans_counts.items():
    total_next_tags_from_prev = sum(next_tag_counts.values())
    if total_next_tags_from_prev > 0:
        for next_tag, count in next_tag_counts.items():
            transition_probabilities[prev_tag][next_tag] = count / total_next_tags_from_prev
                                                                      Toggle Gemini

print("--- Sample Emission Probabilities (P(word|tag)) ---")
# Display a few examples for common tags
example_tags = ['NN', 'VB', 'JJ']
for tag in example_tags:
    if tag in emission_probabilities:
        print(f"Top words for tag '{tag}': {sorted(emission_probabilities[tag].items(), key=lambda x: x[1], re
    else:
        print(f"No emission probabilities found for tag '{tag}'")

print("\n--- Sample Transition Probabilities (P(tag_i | tag_i-1)) ---")
# Display a few examples for common tags
example_prev_tags = ['NN', 'DT', 'VB']
for prev_tag in example_prev_tags:
    if prev_tag in transition_probabilities:
        print(f"Top next tags for previous tag '{prev_tag}': {sorted(transition_probabilities[prev_tag].items(
    else:
        print(f"No transition probabilities found for tag '{prev_tag}'")
```

```
--- Sample Emission Probabilities (P(word|tag)) ---
Top words for tag 'NN': [('modi', 0.10320215229946092), ('india', 0.016277776092079577), ('congress', 0.0107574
Top words for tag 'VB': [('have', 0.04313074462485322), ('modi', 0.04215896667611451), ('get', 0.03151799813742
Top words for tag 'JJ': [('modi', 0.04614891259984841), ('indian', 0.017589750928914825), ('india', 0.014936844

--- Sample Transition Probabilities (P(tag_i | tag_i-1)) ---
Top next tags for previous tag 'NN': [('NN', 0.3595183710397132), ('IN', 0.09323072283437303), ('VBD', 0.060803
Top next tags for previous tag 'DT': [('NN', 0.5083480729583272), ('JJ', 0.19831461307076145), ('NNS', 0.117847
Top next tags for previous tag 'VB': [('NN', 0.19055963294986075), ('DT', 0.14863160389736818), ('JJ', 0.118715
```

```
index = 5
selected_tweet_text = df.loc[index, 'clean_text']
original_pos_tags = df.loc[index, 'pos_tags']

print(f"Selected Tweet Text: {selected_tweet_text}")
print(f"Original POS Tags: {original_pos_tags}")
```

```
Selected Tweet Text: kiya tho refresh maarkefir comment karo
Original POS Tags: [('kiya', 'NN'), ('tho', 'NN'), ('refresh', 'JJ'), ('maarkefir', 'JJ'), ('comment', 'NN'), (
```

```
from nltk.tokenize import word_tokenize

# 1. Tokenize the selected_tweet_text
tweet_words = word_tokenize(selected_tweet_text)

# 2. Create a list of all unique possible POS tags
all_pos_tags = list(emission_probabilities.keys())

# 3. Define a small constant value for smoothing
default_prob = 1e-6

print(f"Tokenized Tweet Words: {tweet_words}")
print(f"Total Unique POS Tags: {len(all_pos_tags)}")
print(f"Sample Unique POS Tags: {all_pos_tags[:10]}")
print(f"Default Smoothing Probability: {default_prob}")
```

```
Tokenized Tweet Words: ['kiya', 'tho', 'refresh', 'maarkefir', 'comment', 'karo']
Total Unique POS Tags: 38
Sample Unique POS Tags: ['WRB', 'NN', 'VBD', 'NNP', 'JJ', 'PRP', 'VB', 'DT', 'VBG', 'VBZ']
Default Smoothing Probability: 1e-06
```

```
def viterbi(words, all_tags, emission_prob, transition_prob, default_prob):
    # Initialize Viterbi path and score matrices
    # Viterbi matrix: stores the maximum probability of a tag sequence ending at a specific tag for a specific
    viterbi_matrix = [{}] # Stores probability
    # Path matrix: stores the backpointer (previous tag) for reconstructing the best path
    path_matrix = [{}] # Stores backpointer

    # Initialize for the first word
    first_word = words[0]
    for tag in all_tags:
        # P(tag | start) - assuming uniform probability for starting tags or a start tag distribution if avail
        # For simplicity, we'll assume a uniform start probability for all tags, or 1 if no 'start' tag is mod
        # and focus on emission probability for the first word.
```

```
            # If a start tag distribution is available, it would be used here.
            # For now, let's treat the first word's tag probability as just its emis
```

Toggle Gemini

```
            # Emission probability for the first word and current tag
            emit_prob = emission_prob.get(tag, {}).get(first_word, default_prob)
            viterbi_matrix[0][tag] = emit_prob
            path_matrix[0][tag] = None # No previous tag for the first word

        # Iterate through the rest of the words
        for t in range(1, len(words)):
            viterbi_matrix.append({}) # New dict for current word's probabilities
            path_matrix.append({})     # New dict for current word's paths
            current_word = words[t]

            for current_tag in all_tags:
                max_prob = 0
                best_prev_tag = None

                # Emission probability for the current word and current tag
                emit_prob = emission_prob.get(current_tag, {}).get(current_word, default_prob)

                for prev_tag in all_tags:
                    # Transition probability from previous tag to current tag
                    trans_prob = transition_prob.get(prev_tag, {}).get(current_tag, default_prob)

                    # Calculate the probability of the path ending with prev_tag -> current_tag
                    # multiplied by emission of current_word by current_tag
                    current_path_prob = viterbi_matrix[t-1].get(prev_tag, 0) * trans_prob * emit_prob

                    if current_path_prob > max_prob:
                        max_prob = current_path_prob
                        best_prev_tag = prev_tag

                viterbi_matrix[t][current_tag] = max_prob
                path_matrix[t][current_tag] = best_prev_tag

        # Backtrack to find the best path
        best_path = []
        # Find the tag with the highest probability for the last word
        last_word_idx = len(words) - 1
        if not viterbi_matrix[last_word_idx]: # Handle case where no probabilities we Toggle Gemini (e.g., all words
            return []

        max_final_prob = 0
        last_best_tag = None
        for tag, prob in viterbi_matrix[last_word_idx].items():
            if prob > max_final_prob:
                max_final_prob = prob
                last_best_tag = tag

        if last_best_tag is None: # If no tag was found, return empty
            return []

        best_path.append(last_best_tag)
        current_best_tag = last_best_tag

        # Iterate backwards through the path_matrix
        for t in range(len(words) - 1, 0, -1):
            current_best_tag = path_matrix[t][current_best_tag]
            best_path.insert(0, current_best_tag) # Insert at the beginning to maintain order

        return list(zip(words, best_path))

    # Apply the Viterbi algorithm
    predicted_pos_tags = viterbi(tweet_words, all_pos_tags, emission_probabilities, transition_probabilities, defa

    print(f"Selected Tweet: {selected_tweet_text}")
    print(f"Tokenized Words: {tweet_words}")
    print(f"Predicted POS Tags: {predicted_pos_tags}")
    print(f"Original POS Tags: {original_pos_tags}")
```

```
Selected Tweet: kiya tho refresh maarkefir comment karo
Tokenized Words: ['kiya', 'tho', 'refresh', 'maarkefir', 'comment', 'karo']
Predicted POS Tags: [('kiya', 'POS'), ('tho', 'NN'), ('refresh', 'VB'), ('maarkefir', 'JJ'), ('comment', 'NN'),
Original POS Tags: [('kiya', 'NN'), ('tho', 'NN'), ('refresh', 'JJ'), ('maarkefir', 'JJ'), ('comment', 'NN'), (
```

Toggle Gemini