```
!pip install nltk pandas
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.3)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.
```

```
import pandas as pd
import re
import nltk
from collections import defaultdict, Counter
```

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')
```

```
[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.
[nltk_data] Downloading package tagsets to /root/nltk_data...
[nltk_data]   Unzipping help/tagsets.zip.
True
```

```
df = pd.read_csv('Twitter_Data.csv')
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

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

```
tweets = df['clean_text'].dropna().tolist()
print(tweets[:5])
```

```
['when modi promised “minimum government maximum governance” expected him begin the difficult job reforming the state why dc
```

```
def preprocess_tweet(tweet):
    tweet = tweet.lower()
    tweet = re.sub(r'http\S+', '', tweet)     # remove URLs
    tweet = re.sub(r'@\w+', '', tweet)         # remove mentions
    tweet = re.sub(r'#\w+', '', tweet)         # remove hashtags
    tweet = re.sub(r'[^a-z\s]', '', tweet)     # remove special chars
    return tweet.strip()
```

```
cleaned_tweets = [preprocess_tweet(t) for t in tweets]
print(cleaned_tweets[:5])
```

```
['when modi promised minimum government maximum governance expected him begin the difficult job reforming the state why does
```

```
nltk.download('punkt_tab')
tokenized_tweets = [nltk.word_tokenize(tweet) for tweet in cleaned_tweets]
print(tokenized_tweets[:2])
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[['when', 'modi', 'promised', 'minimum', 'government', 'maximum', 'governance', 'expected', 'him', 'begin', 'the', 'difficul
```

```
nltk.download('averaged_perceptron_tagger_eng')
tagged_tweets = [nltk.pos_tag(tokens) for tokens in tokenized_tweets]
print(tagged_tweets[:2])
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[[('when', 'WRB'), ('modi', 'NN'), ('promised', 'VBD'), ('minimum', 'JJ'), ('government', 'NN'), ('maximum', 'JJ'), ('goverr
```

```
tag_sequences = [[tag for (_, tag) in sent] for sent in tagged_tweets]
word_tag_pairs = [(word, tag) for sent in tagged_tweets for (word, tag) in sent]
```

```
transition_counts = defaultdict(Counter)

for tags in tag_sequences:
    for i in range(len(tags) - 1):
        transition_counts[tags[i]][tags[i+1]] += 1
```

```
transition_probs = {}

for prev_tag, next_tags in transition_counts.items():
    total = sum(next_tags.values())
    transition_probs[prev_tag] = {
        tag: count / total for tag, count in next_tags.items()
    }
```

```
emission_counts = defaultdict(Counter)

for word, tag in word_tag_pairs:
    emission_counts[tag][word] += 1
```

```
emission_probs = {}

for tag, words in emission_counts.items():
    total = sum(words.values())
    emission_probs[tag] = {
        word: count / total for word, count in words.items()
    }
```

```
transition_probs['UH']  # interjections
```

```
{'NN': 0.39748953974895396,
 'UH': 0.03347280334728033,
 'JJ': 0.100418410041841,
 'DT': 0.02510460251046025,
 'PRP$': 0.100418410041841,
 'VBZ': 0.0041841004184100415,
 'VB': 0.029288702928870293,
 'VBG': 0.02510460251046025,
 'RB': 0.058577405857740586,
 'WRB': 0.008368200836820083,
 'PDT': 0.016736401673640166,
 'NNS': 0.100418410041841,
 'VBD': 0.016736401673640166,
 'IN': 0.03765690376569038,
 'PRP': 0.03347280334728033,
 'JJS': 0.0041841004184100415,
 'WDT': 0.0041841004184100415,
 'WP': 0.0041841004184100415}
```

```
word_frequencies = Counter([word for word, _ in word_tag_pairs])
rare_words = [w for w, c in word_frequencies.items() if c == 1]
len(rare_words)
```

```
62190
```

```
test_sentence = tokenized_tweets[0]
test_sentence
```

```
['when',
 'modi',
 'promised',
 'minimum',
 'government',
 'maximum',
 'governance',
 'expected',
 'him',
 'begin',
 'the',
 'difficult',
 'job',
 'reforming',
 'the',
 'state',
 'why',
 'does',
 'take',
 'years',
 'get',
 'justice',
 'state',
 'should',
 'and',
 'not',
 'business',
 'and',
 'should',
 'exit',
 'psus',
 'and',
 'temples']
```

```
# The '×' character is not valid in Python for multiplication; use '*' instead.
# 'P(tag)' and 'P(word₀ | tag)' are mathematical notations, not valid Python.
# We need to replace them with actual probability values from our calculated dictionaries.

# Calculate initial tag probabilities (P(tag))
# This is the probability of a tag being the first tag in a sentence.
# We'll use the 'tag_sequences' to count how often each tag starts a sentence.
start_tag_counts = defaultdict(int)
for seq in tag_sequences:
    if seq: # Ensure the sequence is not empty
        start_tag_counts[seq[0]] += 1

total_sentences = len(tag_sequences)
# Avoid division by zero if no sentences or all are empty
if total_sentences == 0:
    start_tag_probs = {}
else:
    start_tag_probs = {tag: count / total_sentences for tag, count in start_tag_counts.items()}

# Initialize V (Viterbi path probabilities) and BP (Backpointer) dictionaries
# V[k][tag] will store the probability of the most likely path ending in 'tag' at step 'k'
V = defaultdict(dict)
BP = defaultdict(dict) # Not explicitly used in this single line, but typically part of Viterbi

# Get all unique tags from our emission probabilities for iteration
all_tags = list(emission_probs.keys())

# Get the first word of the test sentence
first_word = test_sentence[0]

# Viterbi initialization for the first word (word_0)
for tag in all_tags:
    # Get the initial probability for the current tag.
    # Use a small constant for tags that never start a sentence to avoid zero probability issues (smoothing).
    initial_prob_for_tag = start_tag_probs.get(tag, 1e-10)

    # Get the emission probability of the first_word given the current tag.
    # Use a small constant for words that haven't been seen with this tag (smoothing).
    emission_prob_for_word = emission_probs.get(tag, {}).get(first_word, 1e-10)

    # Calculate V[0][tag] using corrected syntax and actual probability values
    V[0][tag] = initial_prob_for_tag * emission_prob_for_word
```

```
# The '×' character is not valid in Python for multiplication; use '*' instead.
# 'P(tag | previous_tag)' and 'P(word_t | tag)' are mathematical notations, not valid Python.
```

```
# We need to replace them with actual probability values from our calculated dictionaries.

# Iterate for each word in the test sentence, starting from the second word (index 1)
# The first word's probabilities (V[0]) have already been initialized.
for t in range(1, len(test_sentence)):
    current_word = test_sentence[t]

    # Initialize V[t] and BP[t] for the current time step
    V[t] = {}
    BP[t] = {}

    # Iterate over all possible current tags
    for current_tag in all_tags:
        max_prob_for_current_tag = 0
        best_prev_tag = None

        # Iterate over all possible previous tags that could have led to the current state
        # We only consider tags that actually had a non-zero probability at the previous time step
        for prev_tag in V[t-1]:
            if V[t-1][prev_tag] == 0: # Skip paths with zero probability
                continue

            # Get the transition probability from prev_tag to current_tag
            # Use smoothing (1e-10) for unseen transitions
            transition_prob = transition_probs.get(prev_tag, {}).get(current_tag, 1e-10)

            # Get the emission probability of the current_word given the current_tag
            # Use smoothing (1e-10) for unseen emissions
            emission_prob = emission_probs.get(current_tag, {}).get(current_word, 1e-10)

            # Calculate the probability of this path
            current_path_prob = V[t-1][prev_tag] * transition_prob * emission_prob

            # Check if this path is more likely than the current maximum
            if current_path_prob > max_prob_for_current_tag:
                max_prob_for_current_tag = current_path_prob
                best_prev_tag = prev_tag

        # Store the maximum probability and the backpointer for the current_tag at time t
        V[t][current_tag] = max_prob_for_current_tag
        BP[t][current_tag] = best_prev_tag

# Note: After this loop, you would typically perform a backward pass
# to reconstruct the most likely sequence of tags.
```