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

```
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
```

```
# Update column name if needed
df = pd.read_csv("/content/twitter_dataset.csv")

# Display all columns to find the correct tweet text column
print("Available columns in the DataFrame:")
print(df.columns.tolist())

# Replace 'YOUR_ACTUAL_TWEET_COLUMN_NAME' with the correct column name found above
# For example, if the column is 'TweetContent', change it to df['TweetContent']
tweets = df['Text'].dropna().tolist()
```

```
Available columns in the DataFrame:
['Tweet_ID', 'Username', 'Text', 'Retweets', 'Likes', 'Timestamp']
```

```
# Function to preprocess tweet text
def preprocess_tweet(tweet):
    """
    This function cleans a tweet by:
    1. Removing URLs
    2. Removing user mentions (@username)
    3. Converting text to lowercase
    4. Removing extra spaces
    """

    # Remove URLs (http, https, www links)
    tweet = re.sub(r"http\S+|www\S+", "", tweet)

    # Remove Twitter mentions (@username)
    tweet = re.sub(r"@\w+", "", tweet)

    # Convert text to lowercase for uniformity
    tweet = tweet.lower()

    # Remove leading and trailing whitespaces
    tweet = tweet.strip()

    return tweet


# Apply preprocessing function to all tweets
# Each tweet in the list is cleaned one by one
tweets = [preprocess_tweet(t) for t in tweets]
```

```
# List to store POS-tagged tweets
tagged_sentences = []
```

```
# Tokenize and POS-tag each preprocessed tweet
for tweet in tweets:

    # Split tweet into individual words/tokens
    tokens = nltk.word_tokenize(tweet)

    # Avoid empty tweets
    if tokens:
        # Assign POS tags using NLTK (acts as weak supervision)
        pos_tags = nltk.pos_tag(tokens)

        # Store the tagged sentence
        tagged_sentences.append(pos_tags)
```

```
# Dictionary to store transition counts: P(tag_i | tag_{i-1})
transition_counts = defaultdict(Counter)

# Dictionary to store emission counts: P(word | tag)
emission_counts = defaultdict(Counter)

# Counter to store total occurrences of each POS tag
tag_counts = Counter()
```

```
# Iterate through each POS-tagged sentence
for sentence in tagged_sentences:

    # Start symbol for each sentence
    previous_tag = "<START>"
    tag_counts[previous_tag] += 1

    # Process each word-tag pair in the sentence
    for word, tag in sentence:

        # Count transition from previous tag to current tag
        transition_counts[previous_tag][tag] += 1

        # Count emission of word given the tag
        emission_counts[tag][word] += 1

        # Count total occurrences of the tag
        tag_counts[tag] += 1

        # Update previous tag
        previous_tag = tag
```

```
# Dictionary to store transition probabilities
transition_probs = defaultdict(dict)

# Convert transition counts into probabilities
for prev_tag in transition_counts:

    # Total transitions from previous tag
    total_transitions = sum(transition_counts[prev_tag].values())

    for curr_tag in transition_counts[prev_tag]:

        # P(curr_tag | prev_tag)
        transition_probs[prev_tag][curr_tag] = (
            transition_counts[prev_tag][curr_tag] / total_transitions
        )
```

```
# Dictionary to store emission probabilities
emission_probs = defaultdict(dict)

# Convert emission counts into probabilities
for tag in emission_counts:

    # Total words emitted by the tag
    total_emissions = sum(emission_counts[tag].values())
```

```
    for word in emission_counts[tag]:

        # P(word | tag)
        emission_probs[tag][word] = (
            emission_counts[tag][word] / total_emissions
        )
```

```
# Display sample transition probabilities
print("Sample Transition Probabilities:\n")

for prev_tag in list(transition_probs.keys())[:5]:
    print(f"{prev_tag} → {dict(list(transition_probs[prev_tag].items())[:5])}")
```

```
Sample Transition Probabilities:

<START> → {'NN': 0.507, 'RB': 0.0837, 'JJ': 0.1894, 'CD': 0.0106, 'PRP': 0.0165}
NN → {'RBS': 0.0007694584196507843, 'NN': 0.40899674459899377, '.': 0.2308848771825984, 'VBG': 0.0050902633915359576,
RBS → {'JJ': 0.2922374429223744, 'IN': 0.045662100456621, '.': 0.1917808219178082, 'RB': 0.0547945205479452, 'VBP': 0
JJ → {'VBP': 0.012950210250472301, '.': 0.10992443171430312, 'JJ': 0.15540252300566762, 'NN': 0.570738619050521, 'RB'
VBP → {'CC': 0.0023619722468261, 'NNS': 0.006495423678771774, 'WDT': 0.0023619722468261, 'DT': 0.020076764098021848,
```

```
# Counter to store word frequencies
word_frequency = Counter()

# Count word occurrences across all tagged sentences
for sentence in tagged_sentences:
    for word, tag in sentence:
        word_frequency[word] += 1

# Identify words that appear only once
rare_words = [word for word, freq in word_frequency.items() if freq == 1]

print("Total number of rare words:", len(rare_words))
print("Sample rare words:", rare_words[:10])
```

```
Total number of rare words: 0
Sample rare words: []
```

```
def viterbi(tokens, transition_probs, emission_probs, tag_counts):
    """
    Applies the Viterbi algorithm to find the most probable
    POS tag sequence for a given list of tokens.
    """

    # List to store Viterbi probability tables
    V = [{}]

    # Dictionary to store best tag paths
    path = {}

    # List of all possible POS tags
    tags = list(tag_counts.keys())

    # ----- Initialization Step -----
    for tag in tags:
        if tag == "<START>":
            continue

        # Transition probability from START to current tag
        transition_p = transition_probs["<START>"].get(tag, 1e-6)

        # Emission probability of first word given the tag
        emission_p = emission_probs[tag].get(tokens[0], 1e-6)

        # Store log probability to avoid underflow
        V[0][tag] = math.log(transition_p) + math.log(emission_p)

        # Initialize path
        path[tag] = [tag]

    # ----- Recursion Step -----
    for t in range(1, len(tokens)):
```

```
        V.append({})
        new_path = {}

        for current_tag in tags:
            if current_tag == "<START>":
                continue

            # Emission probability for current word
            emission_p = emission_probs[current_tag].get(tokens[t], 1e-6)

            # Find best previous tag
            (probability, best_prev_tag) = max(
                (
                    V[t-1][prev_tag]
                    + math.log(transition_probs[prev_tag].get(current_tag, 1e-6))
                    + math.log(emission_p),
                    prev_tag
                )
                for prev_tag in V[t-1]
            )

            # Store best probability and path
            V[t][current_tag] = probability
            new_path[current_tag] = path[best_prev_tag] + [current_tag]

        path = new_path

    # ----- Termination Step -----
    best_final_tag = max(V[-1], key=V[-1].get)

    # Return word-tag pairs
    return list(zip(tokens, path[best_final_tag]))
```

```
# Sample noisy tweet
test_tweet = "love this movie 😍"

# Preprocess and tokenize the tweet
test_tokens = nltk.word_tokenize(preprocess_tweet(test_tweet))

# Apply Viterbi decoding
viterbi_result = viterbi(
    test_tokens,
    transition_probs,
    emission_probs,
    tag_counts
)

# Display final POS tags
print("Viterbi POS Tagging Result:\n")
for word, tag in viterbi_result:
    print(f"{word} / {tag}")
```

```
Viterbi POS Tagging Result:

love / NN
this / DT
movie / NN
😍 / NN
```