```
import pandas as pd
import os
```

```
os.listdir('datasets')
```

```
df = pd.read_csv('/content/clean_train.csv')
df.head(5)
```

	Unnamed: 0	id	movie_name	synopsis	genre
0	0	44978	Super Me	A young scriptwriter starts bringing valuable ...	fantasy
1	1	50185	Entity Project	A director and her friends renting a haunted h...	horror
2	2	34131	Behavioral Family Therapy for Serious Psychiat...	This is an educational video for families and ...	family
3	3	78522	Blood Glacier	Scientists working in the Austrian Alps discov...	scifi
4	4	2206	Apat na anino	Buy Dav - Four Men Widely - Apart in Life - By...	action

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
len(df)//2
```

```
21053
```

```
import nltk
from nltk import word_tokenize, pos_tag
```

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

```
[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 punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
print(pos_tag(word_tokenize(df['synopsis'][0])))
```

```
[('A', 'DT'), ('young', 'JJ'), ('scriptwriter', 'NN'), ('starts', 'VBZ'), ('bringing', 'VBG'), ('valuable', 'JJ'), ('objects', 'NNS
```

```
training_data = []
for index, sentence in enumerate(df['synopsis'][:10]):
    training_data.append(pos_tag(word_tokenize(df['synopsis'][index])))

print(training_data[0])
print(training_data[1])
print(training_data[2])
```

```
[('A', 'DT'), ('young', 'JJ'), ('scriptwriter', 'NN'), ('starts', 'VBZ'), ('bringing', 'VBG'), ('valuable', 'JJ'), ('objects', 'NNS
[('A', 'DT'), ('director', 'NN'), ('and', 'CC'), ('her', 'PRP$'), ('friends', 'NNS'), ('renting', 'VBG'), ('a', 'DT'), ('haunted',
[('This', 'DT'), ('is', 'VBZ'), ('an', 'DT'), ('educational', 'JJ'), ('video', 'NN'), ('for', 'IN'), ('families', 'NNS'), ('and', '(
```

```
states = set()
state_list = []
```

```
for x in training_data:
    for y in x:
        state_list.append(y[1])

states = set(state_list)

print(states)
```

```
{'CD', 'DT', 'TO', 'VBZ', 'NNPS', ',', 'VBG', 'WRB', 'VBD', 'VBP', 'JJ', 'POS', 'PRP', 'NNS', 'WDT', 'VBN', '.', 'RB', 'NNP', 'VB',
```

```
 # Dictionary mapping POS tags to shorter descriptions (values) and comments (provided as Python comm
pos_tag_mapping = {
    'PRP$': 'Possessive',
    # CD: Cardinal Number (e.g., "one", "3")
    'CD': 'Cardinal',
```

```
    # VBN: Past Participle Verb (e.g., "gone", "written")
    'VBN': 'Past Participle',
    'TO': 'to', # TO: "to" (e.g., "to go")
    'PRP': 'Personal',
    # WDT: Wh-determiner (e.g., "which", "whose")
    'WDT': 'Wh-determiner',
    # DT: Determiner (e.g., "the", "this")
    'DT': 'Determiner',
    # VBG: Present Participle Verb (Gerund) (e.g., "running", "swimming")
    'VBG': 'Gerund',
    # RP: Particle (e.g., "up", "out")
     'RP': 'Particle',
    # VB: Base Form Verb (e.g., "run", "eat")
    'VB': 'Base Verb',
    # JJ: Adjective (e.g., "happy", "red")
    'JJ': 'Adjective',
    # CC: Coordinating Conjunction (e.g., "and", "but")
    'CC': 'Conjunction',
    # VBZ: Third Person Singular Present Verb (e.g., "he runs")
    'VBZ': '3rd Person Singular Verb',
    # WP: Wh-pronoun (e.g., "who", "what")
    'WP': 'Wh-pronoun',
    # NN: Noun, Singular or Mass (e.g., "cat", "money")
    'NN': 'Noun',
    # RB: Adverb (e.g., "quickly", "very")
    'RB': 'Adverb',
    # IN: Preposition (e.g., "in", "on")
    'IN': 'Preposition',
    # VBP: Non-3rd Person Singular Present Verb (e.g., "I run")
    'VBP': 'Non-3rd Person Singular Verb',
    # WRB: Wh-adverb (e.g., "why", "where")
    'WRB': 'Wh-adverb',
    # VBD: Past Tense Verb (e.g., "he ran")
    'VBD': 'Past Tense Verb',
    # NNS: Noun, Plural (e.g., "cats", "dogs")
    'NNS': 'Plural Noun'
    }
```

```
for x, y in enumerate(training_data):
    for i, j in enumerate(y):
        training_data[x][i] = (j[0], pos_tag_mapping.get(j[1], 'UNKNOWN'))

print(training_data[0])
```

```
[('A', 'UNKNOWN'), ('young', 'UNKNOWN'), ('scriptwriter', 'UNKNOWN'), ('starts', 'UNKNOWN'), ('bringing', 'UNKNOWN'), ('valuable',
```

```
import collections
```

```
initial_counts = collections.defaultdict(int)
transition_counts = collections.defaultdict(lambda: collections.defaultdict(int))
emission_counts = collections.defaultdict(lambda: collections.defaultdict(int))
```

```
for sentence in training_data:
    initial_counts[sentence[0][1]] += 1
    for i in range(len(sentence) - 1):
        current_tag, next_tag = sentence[i][1], sentence[i + 1][1]
        transition_counts[current_tag][next_tag] += 1
        word = sentence[i][0]
        emission_counts[current_tag][word] += 1
```

```
total_sentences = len(training_data)

initial_probabilities = {tag: count / total_sentences for tag, count in initial_counts.items()}
transition_probabilities = {current_tag: {next_tag: count / sum(transition_counts[current_tag].values())
                                          for next_tag, count in transition_counts[current_tag].items()}
                            for current_tag in transition_counts}
emission_probabilities = {tag: {word: count / sum(emission_counts[tag].values())
                                for word, count in emission_counts[tag].items()}
                          for tag in emission_counts}
```

```
print("Initial Probabilities:")
print(initial_probabilities)
print("\nTransition Probabilities:")
print(transition_probabilities)
print("\nEmission Probabilities:")
print(emission_probabilities)
```

```
Initial Probabilities:
{'UNKNOWN': 0.3, 'Determiner': 0.5, 'Plural Noun': 0.2}

Transition Probabilities:
{'UNKNOWN': {'UNKNOWN': 0.6458333333333334, 'Gerund': 0.041666666666666664, 'Noun': 0.041666666666666664, 'Base Verb': 0.020833333333

Emission Probabilities:
{'UNKNOWN': {'A': 0.020833333333333332, 'young': 0.020833333333333332, 'scriptwriter': 0.020833333333333332, 'starts': 0.020833333333
```

```
# Viterbi decoding function
def viterbi_decode(initial_probabilities, transition_probabilities, emission_probabilities, new_data):
    best_path = [None] * len(new_data)
    best_prob = [0.0] * len(new_data)

    # Initialize for the first word
    for tag, prob in initial_probabilities.items():
        emission_prob = emission_probabilities.get(tag, {}).get(new_data[0], 1e-10)
        best_prob[0] = prob * emission_prob
        best_path[0] = tag

    # Process remaining words
    for t in range(1, len(new_data)):
        max_probs = {}
        for current_tag in emission_probabilities.keys():
            max_prob = 0.0
            max_tag = None
            for previous_tag in initial_probabilities.keys():
                transition_prob = transition_probabilities.get(previous_tag, {}).get(current_tag, 1e-10)
                prob = best_prob[t - 1] * transition_prob
                if prob > max_prob:
                    max_prob = prob
                    max_tag = previous_tag
            emission_prob = emission_probabilities.get(current_tag, {}).get(new_data[t], 1e-10)
            max_probs[current_tag] = max_prob * emission_prob
        best_tag = max(max_probs, key=max_probs.get)
        best_prob[t] = max_probs[best_tag]
        best_path[t] = best_tag

    # Backtrack to find the best path
    pos_tags = [None] * len(new_data)
    pos_tags[-1] = best_path[-1]
    for t in range(len(new_data) - 2, -1, -1):
        pos_tags[t] = best_path[t]

    return pos_tags
```

```
new_data = word_tokenize(df['synopsis'][2])
predicted_tags = viterbi_decode(initial_probabilities, transition_probabilities, emission_probabilities, new_data)

print(f'Predicted:\n{predicted_tags}')
print(f'\nOriginal:\n{[x[1] for x in training_data[2]]}')
```

```
Predicted:
['Plural Noun', '3rd Person Singular Verb', 'Determiner', 'Adjective', 'Noun', 'Preposition', 'Plural Noun', 'Conjunction', 'Noun',

Original:
['Determiner', '3rd Person Singular Verb', 'Determiner', 'Adjective', 'Noun', 'Preposition', 'Plural Noun', 'Conjunction', 'Noun',
```