```
import pandas as pd

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

| | clean text | category |
|---|---|---|
| O | 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 remove_urls(text) :
    # Remove URLs (http/https) from the text
    return re. sub(r'http\S+ www\S+' , '' , text)
def remove_mentions (text) :
    # Remove mentions (@username) from the text return
    re. sub(r'@\w+' , text)


from nltk. probability import ConditionalFreqDist,     FreqDist
# Initialize frequency distributions
initial_tag_fd = FreqDist() # Frequency of the first tag in a tweet transition fd =
ConditionalFreqDist() # Frequency of a tag given the previous tag emission fd =
ConditionalFreqDist() # Frequency of a word given its tag

# Process each tweet' s POS tags for
tags_list in             'pos_tags ' ] :
    if not tags_list : # Skip empty tag lists continue

    # Initial tag frequency


    for i, (word, tag) in                              :
        #           Emission            frequency
        emission_fd[tag] [word]          1

        # Transition frequency
        if i > 0:
            prev_tag              =              tags_list[i-l]              [l]
            transition_fd[prev_tag][tag] += 1

# Convert frequencies to probabilities (using Maximum Likelihood Estimation)
# This is a simplification; for a robust HMM, smoothing (e.g. , Laplace) is often used.
# Initial probabilities: P (Tag_i at start) initial_prob = {tag: initial_tag_fd[tag] /
initial_tag_fd.N() for tag in initial_tag_fd}

# Transition probabilities: P (Tag_j I Tag_i
) transition_prob for prev_tag in
transition_fd:
    total transitions = transition_fd[prev_tag] .N()
    if total transitions > e:
        transition_prob[prev_tag] = {tag: [tag] / total_transitions for tag in transition_fd[prev_tag]} else:
        transition_prob[prev_tag] =

# Emission probabilities: P(Word_k I Tag-j)
emission_prob for tag in emission_fd:
    total emissions = emission_fd[tag] ) if
    total emissions > e: emission_prob[tag] =
    {word : emission_fd[tag] [word] /
```

```
total_emissions for word in
emission_fd[tag]} else:
     emission_prob[tag] =

print("Ca1cu1ated HMM Parameters (first 5 for each type) :
print("\nlnitial Probabilities: " {k: v for k, v in list(initia1_prob. items( ) ) [ : 5] })
print("\nTransition Probabilities (first 5 previous tags) .

for i, (prev_tag, next_tags) in enumerate(transition_prob. items()) :
     if i 5: break print (f" {prev_tag} : " , {k: v for k, v in
list(next_tags.items()) [ :5]}) print("\nEmission Probabilities (first 5
tags) :
for i, (tag, words) in enumerate(emission_prob.items()) :
     if i >= 5: break print (f" {tag} : " , {k: v for k, v in list
     (words. items( ) ) [ : 5] })
```

```
Calculated HMM Parameters (first 5 for each type) :
Initial Probabilities: { 1NN' : o. 3353867676203683,       o. 13077144908239713, 'RBI : o. 09839918026248781, 'DT 1• o. 0734327735475122
Transition Probabilities (first 5 previous tags) :
   WRB: { 'NN' : e. 2631164484693375,        e. 18554975883453095,        'DTI : e. 06969189880893789, 'VBD' : e. 066
  NN: { 'NN' : e. 3595183710397132, ' IN' : 0.09323072283437303, 'VBD' : e. 06080342233565473, 'NNS' : e. 05916485627642285, e. 0545 VBD: : 0.1855463656346634, : e. 15572258857717672, e. 11564397749711262, 'RBI : e. 09730449685183115, 'VBN' : e. 0839 NNP: { 'NN' : 0.4392312711311466, 'NNP' : e. 20932439646479625, e. 08434663977697372, 'RB' : e. 03873302093837119, 'VBZ' : 0 .038 e. 5665254923118425, INNS' : e. 15297814944699217 , 0.0995063393579714, 'IN' : e. 0328999190720259, 'RBI : e. 0207607
Emission Probabilities (first 5 tags) :
  WRB: { 'why' : e. 3527957583840224, 'when' : e. 262271086100901, 'how' : 0.25147187977751034, 'where' : e. 1224669030348372, 'whenever { 'modi'
       : e. 10320215229946092, ' india ' : e. 016277776092079577 ,        congress   o. 01075745162889017 , 'bjp'    o .01020622831770691,
  VBD: { 'was' : e. 14797588285960378, 'did ' : e. 06361122444353778, 'had' : 0.047944149780135094, ' said ' : 0.04508817262795231, 'were'
  NNP: 0.4527071102413568, e. 06582458637253158, o. 055387534839589636, o. 050702721935598646, I' : 0.0192136 { 'modi' : e.
       04614891259984841, ' indian ' : e. 017589750928914825, india' : o .01493684439285279, narendra ' : o. 0144226946745750
```

```
import n 1 tk from nltk. tokenize import
word_tokenize from nltk. tag import pos_tag

# Download necessary NLT K data ( if not already downloaded) try :
     nltk. data. find( ' tokenizers/punkt ' )
except LookupError:
     nltk. download( ' punkt' ) try :
     nltk. data . find( ' taggers/ averaged_perceptron_tagger_eng ' )
except LookupError:
     nltk. download( averaged_perceptron_tagger_eng ' ) try :
     nltk. data . find( ' tokenizers/punkt_tab ) except
LookupError:
     nltk. download( ' punkt_tab' )
# Function to perform POS tagging
def get_pos_tags(text) :
     tokens = word_tokenize (text)
     return pos_tag(tokens)

# Apply POS tagging to the ' clean_text_cleaned' column df[ '
pos_tags ' ] = ' clean_text_cleaned ' ] . apply(get_pos_tags)
# Display the DataFrame with the new POS tags column display(df[
[ 'clean_text_cleaned' , 'pos_tags ' ] ] . head())
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to [nltk_data]
          / root/nltk_data. . .
[nltk_data]      Unzipping taggers/ averaged_perceptron_tagger_eng. zip.
```

| | clean text cleaned | pos_tags |
|---|---|---|
| O | when modi promised "minimum government maximum. | [(when, WRB), (modi, NN), (promised, VBD), (",... |
| | talk all the nonsense and continue all the dra. | (talk, NN), (all, PDT), (the, DT), (nonsense,... |
| 2 | what did just say vote for modi welcome bjp t... | [(what, WP), (did, VBD), (just, RB), (say, VB)... |
| 3 | asking his supporters prefix chowkidar their n... | |
| 4 | answer who among these the most powerful world.. | |

[(asking, VBG), (his, PRP$), (supporters, NNS)... [(answer, NN), (who, WP), (among, IN), (these,...

```
# Apply the cleaning functions to the 'clean_text ' column df[ '
clean_text_cleaned' ] = df[ ' clean_text ' ] . astype(str) . apply( remove_urls) df[
' clean_text_cleaned ' ] = ' clean_text_cleaned ' ] . apply(remove_mentions)

# Display the DataFrame with the new cleaned column
display(df[ [ 'clean_text ' , 'clean_text_cleaned' ] ] . head())
```

| | clean text | clean text cleaned |
|---|---|---|
| O | when modi promised "minimum government maximum... | when modi promised "minimum government maximum... |
| 1 | talk all the nonsense and continue all the dra... | talk all the nonsense and continue all the dra... |
| 2 | what did just say vote for modi welcome bjp t... | what did just say vote for modi welcome bjp t... |
| 3 | asking his supporters prefix chowkidar their n.. | asking his supporters prefix chowkidar their n... |
| 4 | answer who among these the most powerful world... | answer who among these the most powerful world... |

```
print(f"Tota1 unique previous tags in transition_prob:
{len(transition_prob)}") print("\nExamp1e transition probabilities for the
first 3 previous tags: for i, (prev_tag, next_tags) in items()) : if i >= 3: break
print (f" {prev_tag}: {next_tags}" )
```

Total unique previous tags in transition_prob: 38

Example transition probabilities for the first 3 previous tags:
WRB: { 'NN' : e. 2631164484693375, e. 18554975883453095, 'DTI : e. 06969189880893789, 'VBD' : e. 066
NN: { 'NN' : o. 3595183710397132, ' IN' : e. 09323072283437303, 'VBD' : e. 06080342233565473, 'NNS' : e. 05916485627642285, e. 0545
VBD: { ' JJ ' : e. 1855463656346634, e. 15572258857717672, e. 11564397749711262, 'RB' : e. 09730449685183115, 'VBN' •. e. 0839

```
print("\n--- Analysis of Transition Probabilities ---\n )

high_prob_transitions =
narrow_distribution_tags =
broad_distribution_tags =

for prev_tag, next_tags in transition_prob. items() :
    if not next_tags: # Skip if no transitions from this tag (shouldn't happen with FreqDist but good practice) continue

    # Identify highest probability transition for the current prev_tag
    max_prob_tag = max(next_tags, . get) max_prob_value = next_tags

    if max_prob_value > 0.9: # Threshold for unusually high probability high_prob_transitions . append
        (f" ' {prev_tag} ' with P={max_prob_value:
    # Check for narrow/ broad distributions num_next_tags = len(next_tags) if num_next_tags <= 2: # Very narrow
    distribution narrow_distribution_tags . append ( f" '{prev_tag}' has only {num_next_tags} unique next tags. el if
    num_next_tags >= 20: # Very broad distribution (arbitrary threshold, adjust as needed) broad_distribution_tags .
    append ( f" '{prev_tag}' has {num_next_tags} unique next tags. ")

print("l. Tags with unusually high transition probabilities (e.g. , > 0. 9) :
" ) if high_prob_transitions: for transition in high_prob_transitions:
print(transition) else:
    print( " No transitions with probability > 0.9 found. " )

print("\n2. Tags exhibiting very narrow distributions (2 or fewer unique next tags) :
if narrow_distribution_tags:
    for tag_info in
narrow_distribution_tags: print(tag_info)
else:
    print( " No tags with very narrow distributions found.
print("\n3. Tags exhibiting very broad distributions (20 or more unique next tags)
: if broad_distribution_tags:
```

for tag_info in broad_distribution_tags: print (tag_info)
else:
print( " No tags with very broad distributions found. " )

# Also explicitly look for any tags that have certainty to be followed by another perfect_transitions = for prev_tag, next_tags in transition_prob. items() : if len(next_tags) = next_tag, prob = list (next_tags . items()) [0] perfect_transitions . append (f" ' {prev_tag} ' *always* followed by '{next_tag} ' (P=l. eeee) " )

if perfect_transitions:
print("\n4. Tags with 100% certainty transitions (only one possible next tag)
. for transition in perfect_transitions: print(transition) else:
print("\n4. No tags found with 100% certainty transitions. ")

Analysis of Transition Probabilities

1 . Tags with unusually high transition probabilities (e.g. , > 0. 9) :
'PDT' 'DT' with p=e.9427
'$ ' 'CD' with P=e.9956
'SYM' 'NN' with P=I.OOOO
'RB I with P=l.eooo

2. Tags exhibiting very narrow distributions (2 or fewer unique next tags) :
'$ ' has only 2 unique next tags.
'SYM' has only 1 unique next tags. has only
1 unique next tags.

3. Tags exhibiting very broad distributions (20 or more unique next tags) :
'WRB' has 31 unique next tags.
'NN' has 37 unique next tags.
'VBD' has 35 unique next tags. 'NNP' has 31 unique next tags. has 36 unique next tags. 'PRP' has 32 unique next tags.
'VB' has 35 unique next tags.
'DT' has 32 unique next tags.
'VBG' has 33 unique next tags.
'VBZ' has 34 unique next tags. ' NNS' has 35 unique next tags. has 27 unique next tags. 'CC' has 32 unique next tags.
'RB' has 33 unique next tags.
' IN ' has 31 unique next tags.
'WP' has 30 unique next tags.
'VBP' has 34 unique next tags.
'PRP$' has 30 unique next tags. 'EX' has 26 unique next tags. has 27 unique next tags. has 25 unique next tags. 'WDT' has 29 unique next tags. 'VBN' has 35 unique next tags. has 28 unique next tags. 'RBR' has 29 unique next tags.
'CD' has 35 unique next tags.
'FW' has 29 unique next tags. 'RP' has 31 unique next tags.

4. Tags with 100% certainty transitions (only one possible next tag) :
'SYM' *always* followed by 'NN' (P=l.eeee) *always* followed by ' RB I (P=I.OOOO)

print(f"Tota1 unique tags with emission probabilities: {len(emission_prob)}") print("\nTop 5 words with emission probabilities for the first 5 unique tags: ")

for i, (tag, words) in enumerate(emission_prob.items()) :
if i 5: # Limit to first 5 tags break
print(f" Tag: ' {tag} '

```
# Sort words by probability in descending order and take the top 5 sorted words =
sorted (words . items(), key=lambda item: item[l] , reverse=True) for j , (word,
prob) in            :
    if j >= 5: # Limit to top 5 words per tag break print(f"
    Word : ' {word} , Probability: {prob: .4f}")
```

Total unique tags with emission probabilities: 38

Top 5 words with emission probabilities for the first 5 unique tags: Tag:
'WRB'

Word : 'why' , Probability: 0.3528
Word: 'when' , Probability: 0.2623
Word: 'how' , Probability: 0.2515
Word: 'where' , Probability: 0.1225
Word: 'whenever' , Probability: 0.0053 Tag:

Word :' modi' , Probability: 0.1032
Word :' india'  Probability: 0.0163
Word : congress '  Probability: 0.0108
Word : ' bjp' , Probability: 0.0102
Word : country' , Probability: 0.0079
Tag: 'VBD'

Word: 'was' , Probability: o. 1480
Word : 'did ' , Probability: o. 0636
Word : 'had' , Probability: o .0479
Word : ' said ' , Probability: o. 0451
Word: 'were' , Probability: o .0403
Tag: ' NNP'
Word :      Probability : 0.4527
Word :      Probability : o. 0658
Word :      Probability : o. 0554
Word :      Probability : o. 0507
Word :      Probability: 0.0192
Tag•

Word: 'modi' , Probability: 0.0461
Word :' indian'  Probability: 0.0176
Word : ' india ' , Probability: 0.0149
Word :       narendra ' , Probability: 0.0144
Word :' good ' , Probability: 0.0134

```
print("\n--- Words with very low emission probabilities (P < 0.0001)
low_prob_emissions = threshold = e.eeel

for tag, words_probs in emission_prob. items() :
    for word, prob in words_probs . items() :
        if prob < threshold:
            low_prob_emissions . append( (word, tag, prob))
# Sort by probability for better readability and take a few examples


if low_prob_emissions:
    print(f"Found {len(low_prob_emissions)} word-tag pairs with emission probability below {threshold}. Here are the first 10 e for i,
    (word, tag, prob) in enumerate(low_prob_emissions) :
        if i 10: break print (f" Word : '{word} ' , Tag ' {tag} ' , Probability:
{prob: .6f}") else:
    print( " No word-tag pairs found with emission probability below the threshold. '
```

Words with very low emission probabilities (P < €.€001) -
Found 152332 word-tag pairs with emission probability below 0.0001. Here are the first 10 examples:
Word :    constituency2' , Tag: INN' , Probability: 0.000001
Word : 'tuthukudi' , Tag          Probability: e. eeee€l

Word : 'thuthukudi' , Tag: Probability: e. eeee€l
Word : 'leadershipwho' , Tag: Probability: e.eeeeel
Word : 'modiganga' , Tag: 'NN' , Probability: e. eeee€l
Word :repressive' , Tag: INN' , Probability: e. eeee€l
Word : 'ministerdisgrace' , Tag. Probability: e .000001
Word: 'archery' , Tag: INN' , Probability: e. eeee€l
Word : ' idu ' , Tag: I NN' , Probability: 0.000001
Word: 'bekagittu' , Tag . 'NN' , Probability: e. eeee€l

print("\n--- Infrequently occurring words (total count < 5) word
counts = for tag in emission_fd:
for word in emission_fd[tag] :
word_counts [word ] = word_counts . get (word, 0) + [word]

infrequent_words =
count threshold = 5 for
word, count in
word_counts.items() :
if count < count threshold:
infrequent_words . append( (word, count))
# Sort by count for better readability and take a few examples infrequent_words.sort(key=lambda x:
x[l])

if infrequent_words:
print(f"Found {len(infrequent_words)} words with total occurrences below {count_threshold}. Here are the first 10 examples: for i, (word, count) in enumerate(infrequent_words) : if i >= 10: break print (f" Word : ' {word} ' , Total Count: {count}" ) else:
print( " No words found with total occurrences below the threshold. 'I )

- Infrequently occurring words (total count < 5)
Found 88234 words with total occurrences below 5. Here are the first 10 examples:
Word : 'wiselythe' , Total Count: 1
Word : 'withot ' , Total Count: 1
Word : jaiwere ' Total Count: 1
Word: 'kongujratwhere' , Total Count: 1
Word: 'workersthere' , Total Count: 1
Word : 'whatbieber' , Total Count: 1
Word : 'wrongthen ' , Total Count: 1
Word: 'withoue' , Total Count: 1
Word : ' wrz promised jobs ' , Total Count: Word: 1
'wadhai' , Total Count: 1
selected_phrase = "modi power" tokens =
selected_phrase. split( ) print(f"Se1ected
phrase: ' {selected_phrase} print(f"Tokenized
phrase: {tokens}" )

Selected phrase: 'modi power'
Tokenized phrase: [ 'modi' , ' power' ]

all_tags = keys()) # All possible tags
viterbi table = [{ } ] # Stores scores for each step backpointer_table [{}] #
Stores backpointers for path reconstruction

first word = tokens [0] print(f"\n--- Viterbi
Initialization for ' {first_word}' ---\n")

for tag in all_tags:
initial_p = initial_prob.get(tag, 0) # P(Tag_i at start) emission_p =
emission_prob.get(tag, { } ) .get(first_word, e) # P (Word_k I Tag_j)
if initial_p > 0 and emission_p > e:
score = initial_p emission_p
[tag] score
# No backpointer for the first word [tag] = None print(f" Tag: {tag}, Initial Prob: {initial_p: .6f}, Emission Prob( ' {first_word}' I ' {tag} ' ) : {emission_p: .6f}' Score:

print(f"\nScores for '{first_word}' :

- Viterbi Initialization for 'modi'

Tag: NN, Initial Prob: 0.335387, Emission Prob( 'modi' I 'NN' ) : 0.103202, Score: 0.034613

Tag: Initial Prob: 0.130771, Emission Prob( 'modi' I 0.046149, Score: 0.006035

Tag: RB, Initial Prob: o. 098399, Emission Prob( 'modi' I 'RBI ) : 0.013175, Score: 0.001296

Tag: NNS, Initial Prob: o. 069665, Emission Prob( 'modi I 'NNS' ) : 0.047367, Score: o. 003300

Tag: VB, Initial Prob: o. 042711, Emission Prob( 'modi' I 'VB' ) : 0.042159, Score: 0.001801

Tag: IN, Initial Prob: 0.033661, Emission Prob( 'modi' I ' IN ' ) : 0.002181, Score: 0.000073

Tag: WRB, Initial Prob: 0 .030844, Emission Prob( 'modi' I 'WRB' ) : 0.000033, Score: o .000001

Tag: PRP, Initial Prob: o. 028642, Emission Prob( 'modi' I 'PRP 0.007099, Score: o. 000203

Tag: VBG, Initial Prob: 0.019739, Emission Prob( 'modi' I 'VBG' ) : e.eeee44, Score: o .000001

Tag: MD, Initial Prob: 0.018591, Emission Prob( 'modi ' I 'MD' ) : 0.006102, Score: 0.000113

Tag: WP, Initial Prob: 0.017518, Emission Prob( 'modi' I 0.001031, Score: 0.000018

Tag: CC, Initial Prob: o. 016450, Emission Prob( 'modi' I 'CC ' ) : o. 001110, Score: 0.000018

Tag: CD, Initial Prob: 0.014137, Emission Prob( 'modi' I 'CD' ) : 0.007211, Score: 0.000102

Tag: VBD, Initial Prob: 0.012652, Emission Prob( 'modi' I 'VBD' ) : 0.017553, Score: o. 000222

Tag: VBN, Initial Prob: o. 012204, Emission Prob( 'modi' I 'VBN' ) : 0.009745, Score: o. 000119

Tag: VBZ, Initial Prob: o. 008388, Emission Prob( 'modi' I 'VBZ' ) : 0.033028, Score: o. 000277

Tag: VBP, Initial Prob: o. 006651, Emission Prob( 'modi ' 0.046309, Score: o. 000308

Tag: JJS, Initial Prob: o. 003442, Emission Prob( 'modi' 'JJS' ) : 0.004163, Score: o . 000014

Tag: WDT , Initial Prob: o. 002878, Emission Prob( 'modi' I 'WDT' ) : 0.001665, score: o .000005

Tag: EX, Initial Prob: o. 002534, Emission Prob( 'modi' I ' EX' ) : 0.000491, Score: 0.000001

Tag: PDT, Initial Prob: o. 001902, Emission Prob( 'modi I 'PDT' ) : 0.066915, Score: o. 000127

Tag: RBR, Initial Prob: o. 001608, Emission Prob( 'modi' I 'RBR' ) : 0.059779, Score: o .000096

Tag: JJR, Initial Prob: o. 000742, Emission Prob( 'modi [1] I ' DR' ) : 0.006814, Score: o .000005

Tag: RBS, Initial Prob: o. 000675, Emission Prob( 'modi' I 'RBS' ) : e. 207652, Score: o. 000140

Tag: UH, Initial Prob: o. 000577 , Emission Prob( 'modi [1] I 'UH' ) : o. ee4237, Score: 0.000002

Tag: FW, Initial Prob: o. 000092, Emission Prob( 'modi' I 'FW' ) : 0.276727, Score: 0.000025

Tag: NNP, Initial Prob: o. 000031, Emission Prob( 'modi I 'NNF)' ) : 0.007946, Score: o .000000

Scores for 'modi' : { INN' : .034612636271181156, : o. 006034960174259071, 'RBI : o. 0012963867801829595, 'NNS' : o. 00329986759025

```
viterbi_table. append ( { } ) # Scores for the current word backpointer_table.append({})
# Backpointers for the current word

current word idx = 1 current word =
tokens [current_word_idx]

print(f"\n--- Viterbi Recursion for ' {current_word}

for current_tag in all_tags:
    max_score_for_current_tag =
    best_prev_tag_for_current_tag =
    None

    # Emission probability for the current word given the current tag emission_p_current =
    emission_prob.get(current_tag, { } ) .get(current_word, 0)

    if emission_p_current == e:
        continue # If word cannot be emitted by this tag, no need to calculate

    print (f" Considering current tag: {current tag}" )
                                        _                   _

    for prev_tag, prev_score in viterbi_table[current_word_idx 1] . items() : #
        Transition probability from previous tag to current tag transition_p =
        transition_prob.get(prev_tag, { } ) .get(current_tag, 0)

        # Calculate path score path score = prev_score * transition_p
         emission_p_current
            _ if path score >
        max_score_for_current_tag:
             _ max score for current tag =
            path_score best_prev_tag_for_current_tag
            = prev_tag

        if path_score > e: # Only print non-zero paths print(f" Path: {prev_tag} - > {current_tag}, Prev Score: {prev_score: .6f},
            Transition P: {transition_p: .6f}, Emis

    if max_score_for_current_tag > 0:
        viterbi_table                  [current tag] = max_score_for_current_tag
                                        _
        backpointer_table[current_word_idx] [current_tag] = best_prev_tag_for_current_tag print(f" Max Score for ' {current_tag} '
        : {max_score_for_current_tag: .9f} (from {best_prev_tag_for_current_tag})\n")
```

print(f"Scores for '{current_word}' : {viterbi_table } " ) print(f"Backpointers for '{current_word}' : {backpointer_table } " )

- Viterbi Recursion for ' power'
Considering current tag: NN

_

Path: Prev Score: o. 034613, Transition P: 0.359518, Emission P ( 'power' I INN' ) : e. 005742, Path Score: e. 000071451

Path: NN, Prev Score: o. 006035, Transition P: o. 566525, Emission P( ' power' I 'NN' ) : e. 005742, Path Score: e. 000019631

Path: - > NN, Prev Score: o. 001296, Transition P: 0.067306, Emission P( 'power' I INN' ) : e. 005742, Path Score: o .000000501

Path: NNS NN, Prev Score: o. 003300, Transition P: o. 076584, Emission P ( 'power' I 'NN' ) : e. 005742, Path Score: e. 000001451

Path: VB - > NN, Prev Score: o. 001801, Transition p: o. 190560, Emission P( 'power' I INN' ) : e. 005742, Path Score: o .000001970

Path: IN - > NN, Prev Score: o. 000073, Transition P: 0.321770, Emission P( 'power' I 'NN' ) : e. 005742, Path Score: o .000000136

Path: WRB prev Score: o. eeeeel, Transition P: o. 263116, Emission P ( 'power' I e. 005742, Path Score: e. eeeeeeee2

Path: PRP - > NN, Prev Score: o. 000203, Transition P: o .024407 , Emission P ( 'power' I 'NN' ) : e. 005742, Path Score: e. eeeeeee28

Path: VBG Prev Score: o. eeeeel, Transition P: o. 257260, Emission P( 'power' I e. 005742, Path Score: e. eeeeeeeel

Path: MD - > NN, Prev Score: o. 000113, Transition P: 0.018673, Emission P ( 'power' I e. 005742, Path Score: e. 000000012

Path: - > NN, Prev Score: o. 000018, Transition P: 0.109971, Emission P( 'power' I INN' ) : e. 005742, Path Score: e. eeeeeeell

Path: cc Prev Score: o. 000018, Transition P: 0.237532, Emission P ( 'power' I e. 005742, Path Score: e. 000000025

Path: CD - > NN, P rev Score: o. 000102, Transition P: o. 366125, Emission P( 'power' I INN' ) : e. 005742, Path Score: o .000000214

Path: VBD Prev Score: o. 000222, Transition P: 0.155723, Emission P ( 'power' I 'NN' ) : e. 005742, Path Score:

Path: VBN NN, Prev Score: o. 000119, Transition P: o. 209336, Emission P( 'power' I INN' ) : e. 005742, Path Score: e. eeeeee143

Path: VBZ - > NN, Prev Score: o. 000277, Transition P: o. 140420, Emission P ( 'power' I 'NN' ) : e. 005742, Path Score: e. eeeeee223

Path: VBP - > NN, Prev Score: o. 000308, Transition p: o. 145603, Emission P( 'power' I INN' ) : e. ee5742, Path Score: e. eeeeee258

Path: JJS - > NN, Prev Score: o .000014, Transition P: o. 502695, Emission P ( 'power' I 'NN' ) : e. 005742, Path Score: e. eeeeeee41

Path: WDT NN, Prev Score: o. 000005, Transition P: Emission P( 'power' I 'NN' ) : e. ee5742, Path Score: e. eeeeeeee3

Path: EX Prev Score: o. 000001, Transition P: 0.178723, Emission P ( 'power' I INN' ) : e. 005742, Path Score: o. oeeeeeeel

Path: PDT Prev Score: o. 000127, Transition P: o. 000981, Emission P ( power' INN' ) : e. 005742, Path Score: o. eeeeeeeel

Path: RBR - > NN, prev Score: o. 000096, Transition P: o. 136520, Emission P ( ' power' I 'NN' ) : e. 005742, Path Score: e. eeeeeee75

Path: - > NN, Prev Score: o. 000005, Transition P: 0.330635, Emission P ( ' power' I 'NN' ) : e. 005742, Path Score: o .000000010

Path: RBS - > NN, Prev Score: o. 000140, Transition P: o. 054429, Emission P ( ' power' I 'NN' ) : e. 005742, Path Score: e. eeeeeee44

Path: UH Prev Score: o. 000002, Transition P: 0.411017, Emission P ( 'power' I e. 005742, Path Score: e. eeeeeeee6

Path: FW NN, Prev Score: o. 000025, Transition P: 0.495081, Emission P( 'power' I INN' ) : e. 005742, Path Score: o. eeeeeee72

Path: NNP Prev Score: o. eeeeee, Transition P: 0.439231, Emission P ( 'power' I 'NN' ) : e. 005742, Path Score: e. eeeeeeeel

Max Score for 'NN' : 0.000071451 (from NN)

Scores for 'power' : 7 .145057586828129e-e5} Backpointers for 'power' : INN' }

print(
"\n--- Viterbi Termination and Backtracking ---\n" )

# Find the overall best score for the last word last word idx = len(tokens) 1 final scores = viterbi_table[last_word_idx]

if not final scores:
print("No possible tag sequences found for the given phrase. ")
predicted_tags else:
best_last_tag = max(final_scores, . get)
max_final_score = final_scores[best_last_tag]

'

print(f"Highest probability for the last word {max_final_score: .9f} with tag ' {best_last_tag}
# Backtrack to find the full sequence of tags
predicted_tags = [best_last_tag] for i in range(last_word_idx, 0, -1) :
prev_tag = backpointer_table[i] [predicted_tags
predicted_tags . insert(€, prev_tag)

print(f"Most likely POS tag sequence for '{ ' ' . join(tokens)} ' : {predicted_tags} " )

- Viterbi Termination and Backtracking

Highest probability for the last word ( 'power' ) : 0.000071451 with tag

Most likely POS tag sequence for 'modi power' : 'NN' ]