Write down python code to find out sentimental polarity +ve -ve neutral of a sentence .

> Indented block

```
pip install textblob
```

```
Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-packages (0.17.1)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-packages (from textblob) (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (4.66.1)
```

```
from textblob import TextBlob

def get_sentiment_polarity(sentence):
    analysis = TextBlob(sentence)
    sentiment_polarity = analysis.sentiment.polarity
    if sentiment_polarity > 0:
        return "Positive"
    elif sentiment_polarity < 0:
        return "Negative"
    else:
        return "Neutral"

# List of sentences
sentences = [
    "I love this product! It's amazing.",
    "The weather is terrible today.",
    "Neutral sentences are neither positive nor negative.",
    "my name is vedant"
]

# Analyze sentiment for each sentence
for sentence in sentences:
    sentiment = get_sentiment_polarity(sentence)
    print(f"Sentence: '{sentence}' has a sentiment of {sentiment}")
```

```
Sentence: 'I love this product! It's amazing.' has a sentiment of Positive
Sentence: 'The weather is terrible today.' has a sentiment of Negative
Sentence: 'Neutral sentences are neither positive nor negative.' has a sentiment of Negative
Sentence: 'my name is vedant' has a sentiment of Neutral
```

- To use different approches similar to following :

- input a sentence
- preprocess
- tokenize the sentence
- find out the sentiment colarity of an individual word token
- sum of all of them to calculate the total colarity

```
!pip install textblob
!pip install nltk
```

```
Requirement already satisfied: textblob in /usr/local/lib/python3.10/dist-packages (0.17.1)
Requirement already satisfied: nltk>=3.1 in /usr/local/lib/python3.10/dist-packages (from textblob) (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob) (4.66.1)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
```

```
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
```

```
import nltk
from textblob import TextBlob

# Download NLTK data
nltk.download('punkt')

# Function to calculate sentiment polarity for a word
def get_word_sentiment_polarity(word):
    analysis = TextBlob(word)
    return analysis.sentiment.polarity

# Function to preprocess, tokenize, and calculate total sentence polarity
def analyze_sentence_sentiment(sentence):
    # Tokenize the sentence into words
    words = nltk.word_tokenize(sentence)

    # Calculate the sentiment polarity for each word
    word_polarities = [get_word_sentiment_polarity(word) for word in words]

    # Calculate the total sentence polarity by summing word polarities
    total_polarity = sum(word_polarities)

    return total_polarity

# List of sentences
sentences = [
    "I love this product! It's amazing.",
    "The weather is terrible today.",
    "Neutral sentences are neither positive nor negative.",
]

# Analyze sentiment for each sentence
for sentence in sentences:
    total_polarity = analyze_sentence_sentiment(sentence)
    print(f"Sentence: '{sentence}'")
    print(f"Total Sentiment Polarity: {total_polarity}\n")
```

```
Sentence: 'I love this product! It's amazing.'
Total Sentiment Polarity: 1.1

Sentence: 'The weather is terrible today.'
Total Sentiment Polarity: -1.0

Sentence: 'Neutral sentences are neither positive nor negative.'
Total Sentiment Polarity: -0.07272727272727272

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

1) Write a code in python using ready function to input some tect from user and identify each token in it

2) How HMM can be used for tagging illustrate python code for probability , transition probability and emission

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
```

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

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
```

```
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

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

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

# Input text from the user
user_input = input("Enter a sentence: ")

# Tokenize the input text
tokens = word_tokenize(user_input)

# Perform POS tagging
pos_tags = pos_tag(tokens)

# Print the token and its corresponding POS tag
for token, pos_tag in pos_tags:
    print(f"Token: {token}, POS Tag: {pos_tag}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Enter a sentence: my roll number is 4163
Token: my, POS Tag: PRP$
Token: roll, POS Tag: NN
Token: number, POS Tag: NN
Token: is, POS Tag: VBZ
Token: 4163, POS Tag: CD
```

```
import numpy as np

# Define the set of states (POS tags)
states = ['Noun', 'Verb', 'Adjective', 'Adverb']

# Define the transition matrix (example probabilities)
# Each row represents the current state, and each column represents the next state
transition_matrix = np.array([
    [0.4, 0.3, 0.1, 0.2],  # Noun
    [0.2, 0.4, 0.2, 0.2],  # Verb
    [0.1, 0.2, 0.5, 0.2],  # Adjective
    [0.3, 0.1, 0.2, 0.4]   # Adverb
])

# Define the emission matrix (example probabilities)
# Rows represent states (POS tags), and columns represent words
emission_matrix = np.array([
    [0.1, 0.2, 0.3, 0.4],  # Noun
    [0.3, 0.1, 0.2, 0.4],  # Verb
    [0.2, 0.4, 0.2, 0.2],  # Adjective
    [0.4, 0.2, 0.1, 0.3]   # Adverb
])

# Example input sentence (a sequence of words)
sentence = ["The", "quick", "brown", "fox"]

# Initialize a matrix to store the probabilities for each state at each position in the sentence
# Each row represents a state, and each column represents a word position in the sentence
probabilities = np.zeros((len(states), len(sentence)))
```

```
# Initialize the probabilities for the first word in the sentence (emission probabilities)
for i, state in enumerate(states):
    probabilities[i, 0] = emission_matrix[i, sentence.index(sentence[0])]

# Forward algorithm to calculate the probabilities for the remaining words
for t in range(1, len(sentence)):
    for j, current_state in enumerate(states):
        probability_sum = 0
        for i, previous_state in enumerate(states):
            transition_prob = transition_matrix[i, j]
            emission_prob = emission_matrix[j, sentence.index(sentence[t])]
            probability_sum += probabilities[i, t - 1] * transition_prob * emission_prob
        probabilities[j, t] = probability_sum
```

```
# Print the final probabilities for each state at each position
for i, state in enumerate(states):
    print(f"Probabilities for {state}: {probabilities[i]}")
```

```
Probabilities for Noun: [0.1     0.048   0.01518 0.00446]
Probabilities for Verb: [0.3      0.023    0.00984  0.004752]
Probabilities for Adjective: [0.2       0.1       0.01412   0.0023356]
Probabilities for Adverb: [0.4       0.056     0.00566   0.0030276]
```