NLP HOLADAYS ASSIGNMENT

2211CS020506
AIML SIGMA

Q1] Correct the Search Query

In [1]:

```
#Correct the Search Query
import re
import json
from collections import Counter
import zlib

# Pre-built dictionary of words
def build_corpus():
    corpus = """
    going to china who was the first president of india winner of the match food in america
    india china usa america president first winner match food going
    """
    words = re.findall(r'\w+', corpus.lower())
    return Counter(words)

# Load compressed dictionary
def get_corpus():
    compressed_corpus = zlib.compress(json.dumps(build_corpus()).encode())
    return json.loads(zlib.decompress(compressed_corpus).decode())

# Calculate edit distance
def edit_distance(word1, word2):
    dp = [[0] * (len(word2) + 1) for _ in range(len(word1) + 1)]
    for i in range(len(word1) + 1):
        for j in range(len(word2) + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])
    return dp[-1][-1]

# Get candidate corrections
def correct(word, corpus):
    if word in corpus:
        return word
    candidates = [(w, edit_distance(word, w)) for w in corpus if edit_distance(word, w) <= 2]
    candidates.sort(key=lambda x: (x[1], -corpus[x[0]]))  # Sort by distance, then frequency
    return candidates[0][0] if candidates else word

# Correct a query
def correct_query(query, corpus):
    words = query.split()
    corrected = [correct(word, corpus) for word in words]
    return ' '.join(corrected)

# Main program
def main():
    # Input
    n = int(input())
    queries = [input().strip() for _ in range(n)]

    # Load dictionary
    corpus = get_corpus()

    # Correct queries
    corrected_queries = [correct_query(query, corpus) for query in queries]

    # Output
    print("Output:")
    for corrected in corrected_queries:
        print(corrected)

# Run program
if __name__ == "__main__":
    main()
```

```
4
gong to china
who ws the first president of india
 winr of the match
fod in america
Output:
going to china
who was the first president of india
winner of the match
food in america
```

Q2] Deterministic Url and HashTag Segmentation

In [2]:

```python
#Deterministic Url and HashTag Segmentation
import re

def load_words(file_path):
    """Load the lexicon from the words.txt file."""
    with open(file_path, 'r') as file:
        return set(line.strip().lower() for line in file)

def segment_string(input_string, lexicon):
    """Segment the input string into valid tokens."""
    n = len(input_string)
    dp = [None] * (n + 1)
    dp[n] = []

    for i in range(n - 1, -1, -1):
        for j in range(i + 1, n + 1):
            substring = input_string[i:j]
            if substring in lexicon or re.fullmatch(r'\d+(\.\d+)?', substring):
                if dp[j] is not None:
                    dp[i] = [substring] + dp[j]
                    break

    return dp[0] if dp[0] else [input_string]

def preprocess_input(input_string):
    """Clean the input string by removing prefixes and extensions."""
    input_string = input_string.lower()
    if input_string.startswith("www."):
        input_string = input_string[4:]
    input_string = re.sub(r'\.(com|edu|org|in|net|gov|io|us|co|uk)$', '', input_string)
    if input_string.startswith("#"):
        input_string = input_string[1:]
    return input_string

def main():
    lexicon = load_words("words.txt")
    n = int(input())
    results = []

    for _ in range(n):
        raw_input = input().strip()
        cleaned_input = preprocess_input(raw_input)
        segmented = segment_string(cleaned_input, lexicon)
        results.append(" ".join(segmented))
    print("Output:\n")
    print("\n".join(results))

if __name__ == "__main__":
    main()
```

```
5
#whatimissmost
#entrepreneurship
youtube.com
wordpress.org
adobe.com
Output:

whatimissmost
entrepreneurship
youtube
wordpress
adobe
```

Q3] Disambiguation: Mouse vs Mouse

In [3]:

```python
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
import numpy as np

# Sample labeled dataset
train_data = [
    ("The complete mouse reference genome was sequenced in 2002.", "animal"),
    ("Tail length varies according to the environmental temperature of the mouse during postnatal development."
    ("A mouse is an input device.", "computer-mouse"),
    ("I need to buy a new mouse for my computer.", "computer-mouse"),
    ("The house mouse is a small rodent.", "animal"),
    ("You can control the cursor with a computer mouse.", "computer-mouse"),
    ("This mouse has a tail.", "animal"),
    ("I use a wireless mouse for my laptop.", "computer-mouse")
]

# Separate the data into texts and labels
texts, labels = zip(*train_data)

# Convert to numpy arrays
texts = np.array(texts)
labels = np.array(labels)

# Create a classifier pipeline with TfidfVectorizer and NaiveBayes Classifier
model = make_pipeline(TfidfVectorizer(), MultinomialNB())

# Train the model
model.fit(texts, labels)

def classify_sentence(sentence):
    """Classify a given sentence into 'animal' or 'computer-mouse'."""
    return model.predict([sentence])[0]

def main():
    # Read the number of sentences
    n = int(input().strip())

    # Process each sentence
    for _ in range(n):
        sentence = input().strip().lower()  # Read sentence and convert to lowercase
        result = classify_sentence(sentence)
        print(result)

if __name__ == "__main__":
    main()
```

```
3
The complete mouse reference genome was sequenced in 2002.
animal
 A mouse is an input device.
computer-mouse
 Tail length varies according to the environmental temperature of the mouse during postnatal development.
animal
```

Q4] Language Detection

In [4]:

```python
def detect_language(text):
    """
    Detects the language of the given text snippet using basic heuristics.

    This function uses simple character-level analysis to make a basic
    language guess. It's not as accurate as a trained model but might
    be sufficient for this specific challenge.

    Args:
      text: The text snippet to detect the language of.

    Returns:
      The guessed language of the text snippet in Title Case.
    """

    # Basic heuristics (can be improved with more sophisticated rules)
    if "the " in text.lower() or "a " in text.lower() or "an " in text.lower():
        return "English"
    elif "le " in text.lower() or "la " in text.lower() or "les " in text.lower():
        return "French"
    elif "der " in text.lower() or "die " in text.lower() or "das " in text.lower():
        return "German"
    elif "el " in text.lower() or "la " in text.lower() or "los " in text.lower():
        return "Spanish"
    else:
        return "Unknown"
```

```
if __name__ == "__main__":
    text = input("")
    language = detect_language(text)
    print(language)
```

```
The story of Rip Van Winkle is set in the years before and after the American Revolutionary War. In a pleasant v
illage, at the foot of New York's Catskill Mountains, lives kindly Rip Van Winkle, a Dutch villager. Van Winkle
enjoys solitary activities in the wilderness, but he is also loved by all in town—especially the children to who
m he tells stories and gives toys. However, he tends to shirk hard work, to his nagging wife's dismay, which has
caused his home and farm to fall into disarray. One autumn day, to escape his wife's nagging, Van Winkle wanders
up the mountains with his dog, Wolf. Hearing his name called out, Rip sees a man wearing antiquated Dutch clothi
ng; he is carrying a keg up the mountain and requires help.
English
```

Q5] The Missing Apostrophes

In [13]:

```
import re
# List of common words that need apostrophes (can be expanded)
contractions = {
    "dont": "don't",
    "cant": "can't",
    "wont": "won't",
    "isnt": "isn't",
    "arent": "aren't",
    "hasnt": "hasn't",
    "havent": "haven't",
    "doesnt": "doesn't",
    "didnt": "didn't",
    "shouldnt": "shouldn't",
    "wouldnt": "wouldn't",
    "couldnt": "couldn't",
    "partys": "party's",
    "wheres": "where's",
    "heres": "here's",
    "whos": "who's",
    "whats": "what's",
    "lets": "let's",
}

def fix_apostrophes(text):
    # Iterate through contractions and replace
    for word, corrected in contractions.items():
        text = re.sub(r'\b' + word + r'\b', corrected, text, flags=re.IGNORECASE)
    return text

# Function to handle both single and multi-line input
def process_input():
    # Read the input
    lines = []
    count=0
    while True:
        try:
            line = input()
            count+=1
            if not line:
                break
            lines.append(line)
            #if count==n:
                #break
        except EOFError:
            break

    text = "\n".join(lines)  # Combine multi-line input into a single string
    fixed_text = fix_apostrophes(text)  # Fix the apostrophes
    print(fixed_text)

# Call the function to start processing the input
#n=int(input())
process_input()
```

```
At a news conference Thursday at the Russian manned-space facility in Baikonur, Kazakhstan, Kornienko said "we w
ill be missing nature, we will be missing landscapes, woods." He admitted that on his previous trip into space i
n 2010 "I even asked our psychological support folks to send me a calendar with photographs of nature, of rivers
, of woods, of lakes."
Kelly was asked if hed miss his twin brother Mark, who also was an astronaut.
"Were used to this kind of thing," he said. "Ive gone longer without seeing him and it was great." The mission w
ont b
The mission wont be the longest time that a human has spent in space - four Russians spent a year or more aboard
the Soviet-built Mir space station in the 1990s.
SCI Astronaut Twins
Scott Kelly (left) was asked Thursday if hed miss his twin brother, Mark, who also was an astronaut. Were used t
o this kind of thing, he said. Ive gone longer without seeing him and it was great. (NASA/Associated Press)
"The last time we had such a long duration flight was almost 20 years and of course al{-truncated-}

At a news conference Thursday at the Russian manned-space facility in Baikonur, Kazakhstan, Kornienko said "we w
ill be missing nature, we will be missing landscapes, woods." He admitted that on his previous trip into space i
n 2010 "I even asked our psychological support folks to send me a calendar with photographs of nature, of rivers
, of woods, of lakes."
Kelly was asked if hed miss his twin brother Mark, who also was an astronaut.
"Were used to this kind of thing," he said. "Ive gone longer without seeing him and it was great." The mission w
on't b
The mission won't be the longest time that a human has spent in space - four Russians spent a year or more aboar
d the Soviet-built Mir space station in the 1990s.
SCI Astronaut Twins
Scott Kelly (left) was asked Thursday if hed miss his twin brother, Mark, who also was an astronaut. Were used t
o this kind of thing, he said. Ive gone longer without seeing him and it was great. (NASA/Associated Press)
"The last time we had such a long duration flight was almost 20 years and of course al{-truncated-}
```

Q6] Segment the Twitter Hashtags

In [6]:

```python
# Assuming we have a predefined dictionary of common words.
# This dictionary can be expanded based on the problem's requirements.
valid_words = set([
    "we", "are", "the", "people", "mention", "your", "faves", "now", "playing",
    "the", "walking", "dead", "follow", "me", "and", "us", "love", "best", "music"
    # More words can be added here based on the context or corpus provided.
])

def segment_hashtag(hashtag):
    # This function splits a single hashtag into constituent words using the valid words dictionary.

    # dp[i] will be a list containing the valid segmentation of the first i characters
    dp = [None] * (len(hashtag) + 1)
    dp[0] = []  # Start with an empty segmentation

    # Iterate over the string
    for i in range(1, len(hashtag) + 1):
        for j in range(i):
            word = hashtag[j:i]
            if word in valid_words and dp[j] is not None:
                dp[i] = dp[j] + [word]
                break

    # If we have a valid segmentation for the entire hashtag, return it
    if dp[len(hashtag)] is not None:
        return " ".join(dp[len(hashtag)])
    else:
        return hashtag  # Return the hashtag as-is if no valid segmentation is found

def process_input():
    N = int(input())  # Number of hashtags
    hashtags = [input().strip() for _ in range(N)]  # Collect the hashtags

    for hashtag in hashtags:
        print(segment_hashtag(hashtag))  # Output the segmented version of each hashtag

# Start processing the input
process_input()
```

```
5
wearethepeople
   mentionyourfaves
nowplaying
thewalkingdead
followme
we are the people
mention your faves
now playing
the walking dead
follow me
```

Q7] Expand the Acronyms

In [7]:

```python
import re

def expand_acronyms(snippets, test_acronyms):
    acronym_dict = {}

    # Regex pattern to match acronyms and their corresponding expansions in parentheses
    acronym_pattern = re.compile(r'([A-Z]{2,})\s*\(([^)]+)\)')

    # Process each snippet to extract acronyms and their expansions
    for snippet in snippets:
        # Extract all occurrences of acronyms with their expansions (in parentheses)
        matches = re.findall(acronym_pattern, snippet)
        for acronym, expansion in matches:
            acronym_dict[acronym] = expansion

        # Process acronyms that are used without parentheses
        # Match all uppercase acronyms that are not yet in the dictionary
        for acronym in acronym_dict.keys():
            if acronym not in snippet:  # Check for acronyms that are used without parentheses
                continue
            # Now, find the first occurrence and associate it with an expansion
            if acronym not in acronym_dict:
                acronym_dict[acronym] = snippet.split(acronym)[0]

    # Answer each test query
    result = []
    for acronym in test_acronyms:
        # We return the expansion from the dictionary or "Expansion not found"
        result.append(acronym_dict.get(acronym, "Expansion not found"))

    return result

def main():
    # Input reading
    N = int(input())  # Number of snippets
    snippets = [input().strip() for _ in range(N)]  # N snippets
    test_acronyms = [input().strip() for _ in range(N)]  # N test acronyms

    # Get the expansions for the test acronyms
    results = expand_acronyms(snippets, test_acronyms)

    # Print the results (expansions for the test acronyms)
    for result in results:
        print(result)

if __name__ == "__main__":
    main()
```

```
3
The United Nations Children's Fund (UNICEF) is a United Nations Programme headquartered in New York City,
The National University of Singapore is a leading global university located in Singapore, Southeast Asia.
Massachusetts Institute of Technology (MIT) is a private research university located in Cambridge, Massachusetts
, United States.
UNICEF
NUS
MIT
Expansion not found
Expansion not found
Expansion not found
```

Q8] Correct the Search Query

In [10]:

```python
#Correct the Search Query
import re
import json
from collections import Counter
import zlib

# Pre-built dictionary of words
def build_corpus():
    corpus = """
    going to china who was the first president of india winner of the match food in america
    india china usa america president first winner match food going
    """
    words = re.findall(r'\w+', corpus.lower())
    return Counter(words)

# Load compressed dictionary
def get_corpus():
    compressed_corpus = zlib.compress(json.dumps(build_corpus()).encode())
    return json.loads(zlib.decompress(compressed_corpus).decode())
```

```python
# Calculate edit distance
def edit_distance(word1, word2):
    dp = [[0] * (len(word2) + 1) for _ in range(len(word1) + 1)]
    for i in range(len(word1) + 1):
        for j in range(len(word2) + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])
    return dp[-1][-1]

# Get candidate corrections
def correct(word, corpus):
    if word in corpus:
        return word
    candidates = [(w, edit_distance(word, w)) for w in corpus if edit_distance(word, w) <= 2]
    candidates.sort(key=lambda x: (x[1], -corpus[x[0]]))  # Sort by distance, then frequency
    return candidates[0][0] if candidates else word

# Correct a query
def correct_query(query, corpus):
    words = query.split()
    corrected = [correct(word, corpus) for word in words]
    return ' '.join(corrected)

# Main program
def main():
    # Input
    n = int(input())
    queries = [input().strip() for _ in range(n)]

    # Load dictionary
    corpus = get_corpus()

    # Correct queries
    corrected_queries = [correct_query(query, corpus) for query in queries]

    # Output
    print("Output:")
    for corrected in corrected_queries:
        print(corrected)

# Run program
if __name__ == "__main__":
    main()
```

```
4
gong to china
who ws the first president of india
winr of the match fod in america
fod in america
Output:
going to china
who was the first president of india
winner of the match food in america
food in america
```

Q9] A Text-Processing Warmup

In [16]:

```python
import re

# Function to count articles and dates
def process_text(text):
    # Counting articles "a", "an", "the"
    a_count = len(re.findall(r'\ba\b', text, re.IGNORECASE))
    an_count = len(re.findall(r'\ban\b', text, re.IGNORECASE))
    the_count = len(re.findall(r'\bthe\b', text, re.IGNORECASE))

    # Regular expression for date matching
    date_pattern = r'\b(\d{1,2}(?:st|nd|rd|th)?(?:\s*(?:of\s*)?\s*(?:January|February|March|April|May|June|July

    # Find all matches for dates
    dates = re.findall(date_pattern, text)
    date_count = len(dates)

    return a_count, an_count, the_count, date_count

# Input handling
T = int(input())  # Number of test cases
```

```
for _ in range(T):
    # Read each text fragment
    text = input().strip()

    # We expect a blank line after the text fragment
    input()  # Read the blank line

    # Process the text
    a_count, an_count, the_count, date_count = process_text(text)

    # Output the results for this test case
    print(a_count)
    print(an_count)
    print(the_count)
    print(date_count)
```

```
5
Delhi, is a metropolitan and the capital region of India which includes the national capital city, New Delhi. It
is the second most populous metropolis in India after Mumbai and the largest city in terms of area.

1
0
4
0
Mumbai, also known as Bombay, is the capital city of the Indian state of Maharashtra. It is the most populous ci
ty in India, and the fourth most populous city in the world, with a total metropolitan area population of approx
imately 20.5 million.

1
0
5
0
New York is a state in the Northeastern region of the United States. New York is the 27th-most extensive, the 3r
d-most populous, and the 7th-most densely populated of the 50 United States.

1
0
6
0
The Indian Rebellion of 1857 began as a mutiny of sepoys of the East India Company's army on 10 May 1857, in the
town of Meerut, and soon escalated into other mutinies and civilian rebellions largely in the upper Gangetic pla
in and central India, with the major hostilities confined to present-day Uttar Pradesh, Bihar, northern Madhya P
radesh, and the Delhi region.

1
0
6
2
The{-truncated-}

0
0
1
0
```

Q10] Who is it?

In [18]:

```
import re

def resolve_anaphora(text, entities):
    # Create a list of entities (names or noun-phrases)
    entity_list = entities.split(";")

    # Initialize a list to hold the resolved entities for each pronoun
    results = []

    # Split the text into sentences or clauses to process the pronouns
    sentences = re.split(r'(?<=[.!?])\s+', text)

    # Initialize the last entity that we encounter before the pronoun
    last_entity = None

    # Iterate through each sentence to resolve pronouns
    for sentence in sentences:
        # For each entity in the sentence, update the last entity if found
        for entity in entity_list:
            if entity.lower() in sentence.lower():
                last_entity = entity  # Update the last encountered entity

        # Look for pronouns in the sentence (e.g., **he**, **she**, **they**)
        pronouns = re.findall(r'\*\*([a-zA-Z]+)\*\*', sentence)
        # For each pronoun found, append the last encountered entity
        for pronoun in pronouns:
            results.append(last_entity)

    return results

def main():
    # Read input
    N = int(input())  # First line: number of text lines
    text_lines = [input() for _ in range(N)]  # Next N lines: the text
    entities = input()  # Last line: the list of entities

    # Combine the text lines into a single string
    text = " ".join(text_lines)

    # Resolve the anaphora
    result = resolve_anaphora(text, entities)

    # Print the results (output the entity corresponding to each pronoun in order)
    for res in result:
        print(res)

if __name__ == "__main__":
    main()
```

```
3
Alice was not a bit hurt, and **she** jumped up on to her feet in a moment: she looked up, but it was all dark o
verhead; before **her** was another long passage, and the White Rabbit was still in sight, hurrying down it. The
re was not a moment to be lost: away went Alice like the wind, and was just in time to hear it say, as **it** tu
rned a corner, 'Oh my ears and whiskers, how late it's getting!' She was close behind **it** when she turned the
corner, but the Rabbit was no longer to be seen: she found herself in a long, low hall, which was lit up by a ro
w of lamps hanging from the roof. There were doors all round the hall, but they were all locked; and when Alice
had been all the way down one side and up the other, trying every door, she walked sadly down the middle, wonder
ing how she was ever to get out again. Suddenly she came upon a little three-legged table, all made of solid gla
ss; there was nothing on **it** except a tiny golden key, and Alice's first thought was that **it** might belong
to one of the doors of the hall; but, alas! either the locks were too large, or the key was too small, but at an
y rate it would not open any of them.  However, on the second time round, she came upon a low curtain she had no
t noticed before, and behind it was a little door about fifteen inches high: she tried the little golden key in
the lock, and to her great delight it fitted! Alice opened the door and found that **it** led into a small passa
ge, not much larger than a rat-hole: she knelt down and looked along the passage into the loveliest garden you e
ver saw. How she longed to get out of that dark hall, and wander about among those beds of bright flowers and th
ose cool fountains, but she could not even get her head through the doorway; 'and even if my head would go throu
gh,' thought poor Alice, 'it would be of very little use without my shoulders. Oh, how I wish I could shut up li
ke a telescope! I think I could, if I only knew how to begin.'
For, you see, so many out-of-the-way things had happened lately, that Alice had begun to think that very few thi
ngs indeed were really impossible.
White Rabbit;Alice;three-legged table;door;tiny golden key
```

NLP Case Study

Sentiment Analysis on Customer Reviews

Importing libraries

In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import RandomForestClassifier

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\edbid\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\edbid\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\edbid\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]: True

Load the dataset

In [2]:

```
# Load the dataset
df = pd.read_csv("amazon_reviews.csv")

# Display the first few rows of the dataset
print("Dataset Loaded:")
print(df.head())

# Check for missing values in the `reviewText` column
missing_count = df['reviewText'].isnull().sum()
print(f"\nNumber of missing values in 'reviewText': {missing_count}")
```

```
Dataset Loaded:
   Unnamed: 0  reviewerName  overall  \
0           0           NaN      4.0   
1           1          0mie      5.0   
2           2           1K3      4.0   
3           3           1m2      5.0   
4           4  2&amp;1/2Men      5.0   

                                          reviewText  reviewTime  day_diff  \
0                                         No issues.  2014-07-23       138   
1  Purchased this for my device, it worked as adv...  2013-10-25       409   
2  it works as expected. I should have sprung for...  2012-12-23       715   
3  This think has worked out great.Had a diff. br...  2013-11-21       382   
4  Bought it with Retail Packaging, arrived legit...  2013-07-13       513   

   helpful_yes  helpful_no  total_vote  score_pos_neg_diff  \
0            0           0           0                   0   
1            0           0           0                   0   
2            0           0           0                   0   
3            0           0           0                   0   
4            0           0           0                   0   

   score_average_rating  wilson_lower_bound  
0                   0.0                 0.0  
1                   0.0                 0.0  
2                   0.0                 0.0  
3                   0.0                 0.0  
4                   0.0                 0.0  

Number of missing values in 'reviewText': 1
```

In [3]:

```
print(df.head())
```

```
   Unnamed: 0  reviewerName  overall  \
0           0           NaN      4.0
1           1          0mie      5.0
2           2           1K3      4.0
3           3           1m2      5.0
4           4  2&amp;1/2Men      5.0

                                          reviewText  reviewTime  day_diff  \
0                                         No issues.  2014-07-23       138
1  Purchased this for my device, it worked as adv...  2013-10-25       409
2  it works as expected. I should have sprung for...  2012-12-23       715
3  This think has worked out great.Had a diff. br...  2013-11-21       382
4  Bought it with Retail Packaging, arrived legit...  2013-07-13       513

   helpful_yes  helpful_no  total_vote  score_pos_neg_diff  \
0            0           0           0                   0
1            0           0           0                   0
2            0           0           0                   0
3            0           0           0                   0
4            0           0           0                   0

   score_average_rating  wilson_lower_bound
0                   0.0                 0.0
1                   0.0                 0.0
2                   0.0                 0.0
3                   0.0                 0.0
4                   0.0                 0.0
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4915 entries, 0 to 4914
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4915 non-null   int64
 1   reviewerName          4914 non-null   object
 2   overall               4915 non-null   float64
 3   reviewText            4914 non-null   object
 4   reviewTime            4915 non-null   object
 5   day_diff              4915 non-null   int64
 6   helpful_yes           4915 non-null   int64
 7   helpful_no            4915 non-null   int64
 8   total_vote            4915 non-null   int64
 9   score_pos_neg_diff    4915 non-null   int64
 10  score_average_rating  4915 non-null   float64
 11  wilson_lower_bound    4915 non-null   float64
dtypes: float64(3), int64(6), object(3)
memory usage: 460.9+ KB
```

In [5]:

```
df.describe()
```

Out[5]:

	Unnamed: 0	overall	day_diff	helpful_yes	helpful_no	total_vote	score_pos_neg_diff	score_average_rating
count	4915.000000	4915.000000	4915.000000	4915.000000	4915.000000	4915.000000	4915.000000	4915.000000
mean	2457.000000	4.587589	437.367040	1.311089	0.210376	1.521465	1.100712	0.075468
std	1418.982617	0.996845	209.439871	41.619161	4.023296	44.123095	39.367949	0.256062
min	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	-130.000000	0.000000
25%	1228.500000	5.000000	281.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	2457.000000	5.000000	431.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	3685.500000	5.000000	601.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	4914.000000	5.000000	1064.000000	1952.000000	183.000000	2020.000000	1884.000000	1.000000

In [6]:

```
df.columns
```

Out[6]:

```
Index(['Unnamed: 0', 'reviewerName', 'overall', 'reviewText', 'reviewTime',
       'day_diff', 'helpful_yes', 'helpful_no', 'total_vote',
       'score_pos_neg_diff', 'score_average_rating', 'wilson_lower_bound'],
      dtype='object')
```

Handle Missing Values

In [7]:

```
# Drop rows with missing 'reviewText'
df = df.dropna(subset=['reviewText'])

# Verify that missing values are removed
print("\nDataset after removing rows with missing 'reviewText':")
print(df.info())
```

```
Dataset after removing rows with missing 'reviewText':
<class 'pandas.core.frame.DataFrame'>
Index: 4914 entries, 0 to 4914
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Unnamed: 0            4914 non-null   int64  
 1   reviewerName          4913 non-null   object 
 2   overall               4914 non-null   float64
 3   reviewText            4914 non-null   object 
 4   reviewTime            4914 non-null   object 
 5   day_diff              4914 non-null   int64  
 6   helpful_yes           4914 non-null   int64  
 7   helpful_no            4914 non-null   int64  
 8   total_vote            4914 non-null   int64  
 9   score_pos_neg_diff    4914 non-null   int64  
 10  score_average_rating  4914 non-null   float64
 11  wilson_lower_bound    4914 non-null   float64
dtypes: float64(3), int64(6), object(3)
memory usage: 499.1+ KB
None
```

Define Preprocessing Functions

In [8]:

```
# Initialize the lemmatizer and stop words
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Function to clean text
def preprocess_text(text):
    # Remove special characters, numbers, and punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Tokenization
    tokens = word_tokenize(text)
    # Remove stopwords and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)
```

Apply Preprocessing to the Text Data

In [9]:

```
# Apply the preprocessing function to the 'reviewText' column
df['cleaned_review'] = df['reviewText'].apply(preprocess_text)

# Display the original and cleaned text for verification
print("\nOriginal and Cleaned Reviews:")
print(df[['reviewText', 'cleaned_review']].head())
```

```
Original and Cleaned Reviews:
                                          reviewText  \
0                                         No issues.   
1  Purchased this for my device, it worked as adv...   
2  it works as expected. I should have sprung for...   
3  This think has worked out great.Had a diff. br...   
4  Bought it with Retail Packaging, arrived legit...   

                                      cleaned_review  
0                                              issue  
1  purchased device worked advertised never much ...  
2  work expected sprung higher capacity think mad...  
3  think worked greathad diff bran gb card went s...  
4  bought retail packaging arrived legit orange e...  
```

Save the Cleaned Dataset (Optional)

In [10]:

```
# Save the cleaned dataset to a new CSV file
df.to_csv("cleaned_amazon_reviews.csv", index=False)

print("\nCleaned dataset saved as 'cleaned_amazon_reviews.csv'.")
```

```
Cleaned dataset saved as 'cleaned_amazon_reviews.csv'.
```

Create Sentiment Labels

In [11]:

```
# Step 1: Load the preprocessed dataset
df = pd.read_csv("cleaned_amazon_reviews.csv")

# Step 2: Create Sentiment Labels
```

```
def assign_sentiment(overall):
    if overall >= 4:
        return "Positive"
    elif overall == 3:
        return "Neutral"
    else:
        return "Negative"

df['sentiment'] = df['overall'].apply(assign_sentiment)

# Step 3: Text Vectorization (TF-IDF)
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
X = tfidf.fit_transform(df['cleaned_review'])  # Use cleaned text column
y = df['sentiment']

# Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Train the Model

In [12]:

```
# Step 5: Train the Model
model = MultinomialNB()
model.fit(X_train, y_train)
```

Out[12]:

```
▾ MultinomialNB
MultinomialNB()
```

Evaluate the Model

In [13]:

```
# Step 6: Evaluate the Model
y_pred = model.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

    Negative       0.00      0.00      0.00        56
     Neutral       0.00      0.00      0.00        30
    Positive       0.91      1.00      0.95       897

    accuracy                           0.91       983
   macro avg       0.30      0.33      0.32       983
weighted avg       0.83      0.91      0.87       983

Accuracy Score: 0.9125127161749745
```

```
C:\Users\edbid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Prec
ision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\edbid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Prec
ision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\edbid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Prec
ision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Predict Sentiments for New Reviews

In [14]:

```
# Step 7: Predict Sentiments for New Reviews
new_reviews = ["The product is excellent and exceeded my expectations.",
               "Worst purchase ever. Bad product.",
               "It's okay, but could be better."]
new_reviews_cleaned = [" ".join(word for word in review.lower().split() if word.isalnum()) for review in new_re
new_reviews_tfidf = tfidf.transform(new_reviews_cleaned)
predictions = model.predict(new_reviews_tfidf)

for review, sentiment in zip(new_reviews, predictions):
    print(f"Review: {review}\nSentiment: {sentiment}\n")
```

```
Review: The product is excellent and exceeded my expectations.
Sentiment: Positive

Review: Worst purchase ever. Bad product.
Sentiment: Negative

Review: It's okay, but could be better.
Sentiment: Neutral
```

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js