Correct the Search Query

```
import re
from difflib import get_close_matches
import sys
valid_words = set([
    "gong", "going", "to", "china", "who", "was", "the", "first",
"president",
    "of", "india", "winner", "match", "food", "in", "america"
])
def correct_word(word):
    candidates = get_close_matches(word, valid_words, n=1)
    return candidates[0] if candidates else word
def split_words(query):
    words = query.split()
    corrected_words = [correct_word(word) for word in words]
    return " ".join(corrected_words)


def process_queries():

    num_queries = int(input().strip())

    for _ in range(num_queries):
        query = input().strip()

        corrected_query = split_words(query)
        print(corrected_query)


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

```
who was the first president of india
```

Deterministic Url and HashTag Segmentation

```
def load_lexicon():

    return set([
        "a", "about", "above", "after", "again", "against", "all",
"an", "and", "any",
        "are", "as", "at", "back", "be", "because", "been", "before",
"being", "below",
        "between", "both", "but", "by", "can", "cannot", "could",
"down", "during", "for",
        "from", "go", "goes", "gone", "have", "having", "he", "he's",
"here", "here's",
        "his", "how", "how's", "i", "i'm", "i've", "in", "into", "is",
```

```
"it", "it's", "its",
        "let", "like", "make", "man", "me", "my", "myself", "no",
"nor", "not", "of", "off",
        "on", "once", "only", "or", "other", "ought", "out", "over",
"per", "re", "right",
        "sought", "the", "theirs", "there", "there's", "they",
"they're", "to", "under",
        "until", "upon", "was", "we", "we're", "we've", "what",
"what's", "when", "where",
        "where's", "which", "who", "who's", "why", "why's", "with",
"without", "you", "you're",
        "you've", "your", "yourself", "yours", "down", "seconds",
"earth", "hour", "human",
        "time", "name", "my", "name", "namecheap", "big", "rock",
"apple", "check", "domain",
        "being", "follow", "back", "social", "media", "today", "let",
"go", "this", "is", "insane",
        "now", "to", "down"
    ])

def segment_string(string, lexicon):
    n = len(string)
    dp = [None] * (n + 1)
    dp[0] = []

    for i in range(1, n + 1):
        for j in range(i):
            word = string[j:i]
            if word in lexicon:
                if dp[j] is not None:
                    dp[i] = dp[j] + [word]
                    break

    return dp[n] if dp[n] is not None else [string]

def preprocess_input(input_string):

    if input_string.startswith('www.'):
        input_string = input_string[4:]


    extensions = ['.com', '.org', '.net', '.edu', '.in', '.co', '.io']
    for ext in extensions:
        if input_string.endswith(ext):
            input_string = input_string[:-len(ext)]
            break


    if input_string.startswith('#'):
        input_string = input_string[1:]
```

```
    return input_string.lower()

def segment_input(input_string, lexicon):
    processed_input = preprocess_input(input_string)
    segmented = segment_string(processed_input, lexicon)
    return ' '.join(segmented)

def main():
    lexicon = load_lexicon()


    n = int(input().strip())

    for _ in range(n):
        input_string = input().strip()
        result = segment_input(input_string, lexicon)
        print(result)

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

letusgo

Disambiguation: Mouse vs Mouse

```
def classify_mouse(sentence):

    sentence = sentence.lower()

    computer_mouse_keywords = ['input device', 'usb', 'click',
'pointer', 'cursor', 'computer']


    animal_mouse_keywords = ['genome', 'tail', 'postnatal',
'development', 'rodent']

    if any(keyword in sentence for keyword in
computer_mouse_keywords):
        return "computer-mouse"

    elif any(keyword in sentence for keyword in
animal_mouse_keywords):
        return "animal"

    else:
        return "animal"


def main():
```

```
    n = int(input().strip())

    for _ in range(n):
        sentence = input().strip()

        print(classify_mouse(sentence))


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

`animal`

Language Detection

```
import operator
import string


eng_words = 'the|of|and|to|a|in|for|is|on|that|by|this|with|i|you|it|
not|or|be|are|from|at|as|your|all|have|new|more|an|was|we|will|home|
can|us|about|if|page|my|has|search|free|but|our|one|other|do|no|
information|time|they|site|he|up|may|what|which|their|news|out|use|
any|there|see|only|so|his|when|contact|here|business|who|web|also|now|
help|get|pm|view|online|c|e|first|am|been|would|how|were|me|s|
services|some|these|click|its|like|service|x|than|find'
spa_words = 'va|mientras|menos|momento|hacia|hace|estos|mayor|otro|
antes|le|ver|dice|han|la|lo|vida|tu|vez|bien|otra|hay|decir|creo|te|
porque|estaba|esa|yo|ya|cuando|nada|de|algunos|tanto|mucho|tambin|nos|
ao|cosas|espa|desde|gran|sido|hoy|el|en|bueno|ser|otras|como|ejemplo|
que|toda|as|sea|casi|todo|es|ademas|pues|nunca|muy|aqu|poco|ese|un|
sus|estas|sobre|eso|vamos|solo|aos|tienen|forma|puede|segun|sino|les|
que|como|aunque|veces|luego|tena|ahora|o|una|nosotros|habla|mismo|
gente|uno|despues|por|durante|son|cada|donde|otros|tiene|siempre|m|
contra|estan|pero|los|todas|ellos|poder|trabajo|a|ms|da|parte|
personas|gobierno|ha|he|me|casa|caso|mi|fue|del|era|das|tres|usted|
este|unos|esta|esto|al|mundo|general|pas|mejor|tal|mujer|tan|ni|para|
no|parece|politica|hecho|pueden|s|sin|todos|algo|lugar|tiempo|est|
ella|entonces|hombre|estado|las|hacer|e|entre|su|hasta|primera|si|y|
dos|con|se'
ger_words = 'august|siehe|kommt|etwa|begriff|immer|liste|selbst|meist|
aber|weitere|als|denen|alle|auf|genannt|ihr|aus|einige|hatte|hat|ca|
geschichte|waren|unter|beim|landkreis|de|da|band|isbn|das|leben|dr|
bis|wenn|diesen|name|zeit|die|deutschland|teil|haben|erste|jedoch|ihn|
kirche|bereits|kann|art|deutschen|jahrhundert|nur|welt|jahren|artikel|
zu|es|er|wird|zwei|diesem|bekannt|werden|dieser|dieses|fr|gemeinde|
dort|soll|menschen|welche|diese|ort|seine|auch|drei|nicht|ende|
bezeichnung|je|sind|zur|wurden|of|jahre|literatur|ist|und|durch|zum|
and|wie|einer|eines|namen|nach|keine|damit|eine|basisdaten|ihm|einem|
```

```
usa|oder|liegt|befindet|was|war|sondern|konnte|viele|gegen|wurde|
mnchen|adresse|gibt|beiden|heute|muss|schon|bei|karte|seit|januar|der|
des|beispiel|um|dann|stadt|dem|den|politik|sein|ein|ihre|seinem|
seinen|ab|wieder|ohne|noch|vom|von|dass|am|an|im|zwischen|vor|in|
allem|ersten|mehr|seiner|verwendet|sowie|steht|form|bedeutung|
bezeichnet|jahr|also|einwohner|sich|sie|neue|hier|verlag|anderen|mit|
besteht|sehr|dabei|wappen|gilt|deutsche|man|bzw|deren|ihren|m|berlin|
km|st|so|oft|the|ihrer'
frn_words = 'vraiment|monde|l|comme|trop|femme|le|mais|la|donner|tu|
ici|aux|te|regarder|ta|ami|me|de|personne|moi|ces|mon|ma|du|voir|sans|
d|faire|toujours|vouloir|tres|l|partir|t|peutetre|attendre|oui|en|ses|
tuer|laisser|chez|autre|et|jamais|homme|rien|quelque|peu|bien|sur|lui|
avoir|accord|chose|fois|savoir|les|que|comprendre|dire|ser|qui|je|
vrai|on|juste|oh|pouvoir|cette|s|tout|une|bon|estce|demander|ou|
comment|aimer|mes|vie|croire|ce|son|besoin|passer|avec|parler|toi|
penser|temps|venir|suivre|vous|arreter|sortir|meme|prendre|o|des|dans|
pour|merci|un|falloir|mettre|connaitre|encore|aller|pere|petit|aussi|
non|an|pourquoi|il|par|pas|quand|alors|seul|ne|mourir|deux|plus|quoi|
ils|arriver|rester|devoir|notre|ca|elle|dieu|maintenant|jour|apres|
mal|trouver|fille|si|y|nous|sa|se'

en_count = 0
spa_count = 0
ger_count = 0
frn_count = 0


def cleanString(s):

    table = s.maketrans("", "", string.punctuation)
    cleaned = s.translate(table)

    return ''.join([c for c in cleaned if ord(c) < 128])


sentinel = ''
line = input()
for word in cleanString(line.strip()).split(' '):
    if len(word) > 2:
        if word in eng_words:
            en_count += 1
        elif word in spa_words:
            spa_count += 1
        elif word in ger_words:
            ger_count += 1
        elif word in frn_words:
            frn_count += 1

count_list = {'English': en_count, 'Spanish': spa_count, 'French':
frn_count, 'German': ger_count}
```

```
sorted_count_list = sorted(count_list.items(),
key=operator.itemgetter(1), reverse=True)


English
```

The Missing Apostrophes

```
import re

contractions = {
    "dont": "don't", "didnt": "didn't", "cant": "can't", "isnt":
"isn't",
    "arent": "aren't", "wasnt": "wasn't", "werent": "weren't",
"hasnt": "hasn't",
    "havent": "haven't", "im": "I'm", "its": "it's", "youre":
"you're",
    "theyre": "they're", "whos": "who's", "whats": "what's", "id":
"I'd",
    "ive": "I've", "youve": "you've", "shes": "she's", "hes": "he's",
    "were": "we're", "ill": "I'll", "youll": "you'll", "hell":
"he'll",
    "theyll": "they'll"
}

def restore_apostrophes(text):

    for wrong, correct in contractions.items():
        text = text.replace(wrong, correct)


    text = re.sub(r"(\b\w+'s\b)(?=\s)", r"\1", text)
    text = re.sub(r"(\bhe|she|they|we|I)(?=d\b)", r"\1'd", text)

    return text

def main():

    import sys
    input_text = sys.stdin.read()

    fixed_text = restore_apostrophes(input_text)

    print(fixed_text)

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

Segment the Twitter Hashtags

```
def getDict():

    words_arr = ['we', 'are', 'the', 'people', 'mention', 'your',
'faves', 'now', 'playing', 'the', 'walking', 'dead', 'follow', 'me']
    word_dict = {}

    for word in words_arr:
        word_dict[word] = 0
    return word_dict

def isValidWord(word, word_dict):
    return word in word_dict

def getSegmentedWord(rword, word_dict):
    start = 0
    valid_words = []
    while start < len(rword):
        found = False

        for length in range(len(rword), start, -1):
            if isValidWord(rword[start:length], word_dict):
                valid_words.append(rword[start:length])
                start = length
                found = True
                break

        if not found:
            start += 1
    return valid_words



test_cases = int(input())
word_dict = getDict()
for _ in range(test_cases):
    print(' '.join(getSegmentedWord(input().strip(), word_dict)))
```

```
we are
```

Expand the Acronyms

```
import re

def extract_acronyms_and_expansions(snippets):
    acronym_expansion = {}

    for snippet in snippets:
        matches = re.findall(r'([A-Z]{2,})\s?\(([^)]+)\)', snippet)
```

```
        for acronym, expansion in matches:
            if acronym not in acronym_expansion:
        matches = re.findall(r'([A-Z]{2,})\s?:\s?([^\.]+)', snippet)
        for acronym, expansion in matches:
            if acronym not in acronym_expansion:
                acronym_expansion[acronym] = expansion

    return acronym_expansion

def main():
    N = int(input())
    snippets = [input().strip() for _ in range(N)]
    acronym_expansion = extract_acronyms_and_expansions(snippets)
    for _ in range(N):
        query = input().strip()
        if query in acronym_expansion:
            print(acronym_expansion[query])
if __name__ == "__main__":
    main()
```

A Text-Processing Warmup

```
import re

def process_text_fragments(T, fragments):
    article_patterns = {
        'a': r'\b a \b',
        'an': r'\b an \b',
        'the': r'\b the \b'
    }
    date_pattern = r'(\d{1,2})(st|nd|rd|th)? (\d{1,2}) (January|Feb|
Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|\d{1,2}) (\d{4})|'\
                   r'(\d{1,2})\/(\d{1,2})\/(\d{4})'

    for fragment in fragments:
        count_a = len(re.findall(article_patterns['a'], fragment))
        count_an = len(re.findall(article_patterns['an'], fragment))
        count_the = len(re.findall(article_patterns['the'], fragment))
        date_count = len(re.findall(date_pattern, fragment))
        print(count_a)
        print(count_an)
        print(count_the)
        print(date_count)
T = int(input())
fragments = []
for _ in range(T):
    fragment = ""
    while True:
        try:
```

```
            line = input()
            if line == "":
                break
            fragment += line + "\n"
        except EOFError:
            break
    fragments.append(fragment.strip())
process_text_fragments(T, fragments)
```

```
6
1
29
0
```

Who is it?

```
import re

def resolve_anaphora(text_lines, names_list):
    text = " ".join(text_lines)
    candidates = names_list.split(';')
    pronouns = []
    pronoun_pattern = re.compile(r'\*\*([a-zA-Z]+)\*\*')

    for match in pronoun_pattern.finditer(text):
        pronouns.append((match.group(1), match.start(), match.end()))
    results = []

    for pronoun, start, end in pronouns:
        candidate_match = None
        for candidate in candidates:
            candidate_pos = text.lower().find(candidate.lower(), 0,
start)
            if candidate_pos != -1:
                candidate_match = candidate


        if candidate_match:
            results.append(candidate_match)

    return results

N = int(input())
text_lines = [input().strip() for _ in range(N)]
names_list = input().strip()


result = resolve_anaphora(text_lines, names_list)
```

```
for r in result:
    print(r)
```

SENTIMENT ANALYSIS ON CUSTOMER REVIEW

ABSTRACT:

Sentiment analysis is a Natural Language Processing (NLP) task aimed at identifying and classifying the sentiment expressed in textual data. This project focuses on analyzing customer reviews to determine whether they convey positive, negative, or neutral sentiments. The process involves text preprocessing techniques such as cleaning, tokenization, lemmatization, and vectorization, followed by the development and evaluation of machine learning models for classification. The outcome of this project provides insights into customer opinions, aiding businesses in decision-making and improving customer satisfaction.

METHODOLOGY :

1. *Data Collection*

Source: Use publicly available datasets such as Kaggle's customer review datasets or scrape data from e-commerce websites.

2. *Data Preprocessing*

a. Cleaning the Text:

- Remove special characters, numbers, and punctuation.
- Convert text to lowercase.
- Remove stopwords (e.g., "is," "the," "and").

b. Tokenization:

- Split the text into individual words or tokens.

c. Lemmatization:

- Reduce words to their base or root form (e.g., "running" -> "run").

d. Vectorization:

- Convert textual data into numerical form using methods like: o Bag of Words (BoW) o Term Frequency-Inverse Document Frequency (TF-IDF) o Word Embeddings (e.g., Word2Vec, GloVe, or BERT).

3. *Model Development*

a. Train-Test Split:

- Split the dataset into training and testing sets (e.g., 80% training, 20% testing). b. Model Selection:
- Use classification algorithms like:

o Logistic Regression o Support Vector Machines (SVM) o Naïve Bayes o Random Forest o Neural Networks (if using deep learning).

-

4. *Model Training*

- Train the model using the preprocessed data.

5. *Model Evaluation*

- Evaluate the model's performance using metrics such as:

  o Accuracy o Precision o Recall o F1-score

- Visualize performance using a confusion matrix.

```
!pip install nltk scikit-learn seaborn
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-
packages (3.9.1)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.10/dist-packages (1.6.0)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.10/dist-packages (0.13.2)
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) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from nltk) (4.67.1)
Requirement already satisfied: numpy>=1.19.5 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: pandas>=1.2 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (24.2)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn)
(2024.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
```

```
# 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.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression from
sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix import seaborn as
sns import matplotlib.pyplot as plt

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

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

```
True
```

```
from google.colab import files
uploaded = files.upload()
```

```
# Load the dataset
df = pd.read_csv('amazon.csv')  # Ensure 'amazon.csv' is the correct
file name <IPython.core.display.HTML object>
```

```
Saving amazon.csv to amazon.csv
```

```
# Check the first few rows of the dataset
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 19996,\n  \"fields\":
[\n    {\n      \"column\": \"Text\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 19996,\n
\"samples\": [\n          \"WORST APP EVER!! Don't get if you have a
Kindle Fire, you have to have a microphone for it to work but its
stupid!!\",\n          \"This app is a pretty good app, I have a great
time when I play this.  Don't listen to the bad reviews because they
are off of previous versions of the game.  I have a kindle fire, so
this game is meant for it.  For all those people complaining about how
\",\n      \"Hopefully this will get fixed. Force close on droid x
gingerbread. I hope they send an update son to fix the issue.\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"label\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


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

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

```
import nltk
nltk.download('punkt')  # Ensure the correct 'punkt' resource is
downloaded
```

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

```
import nltk
nltk.data.path.append('/usr/share/nltk_data')  # Add the resource path
nltk.download('punkt', download_dir='/usr/share/nltk_data')  # Force
download to this directory
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

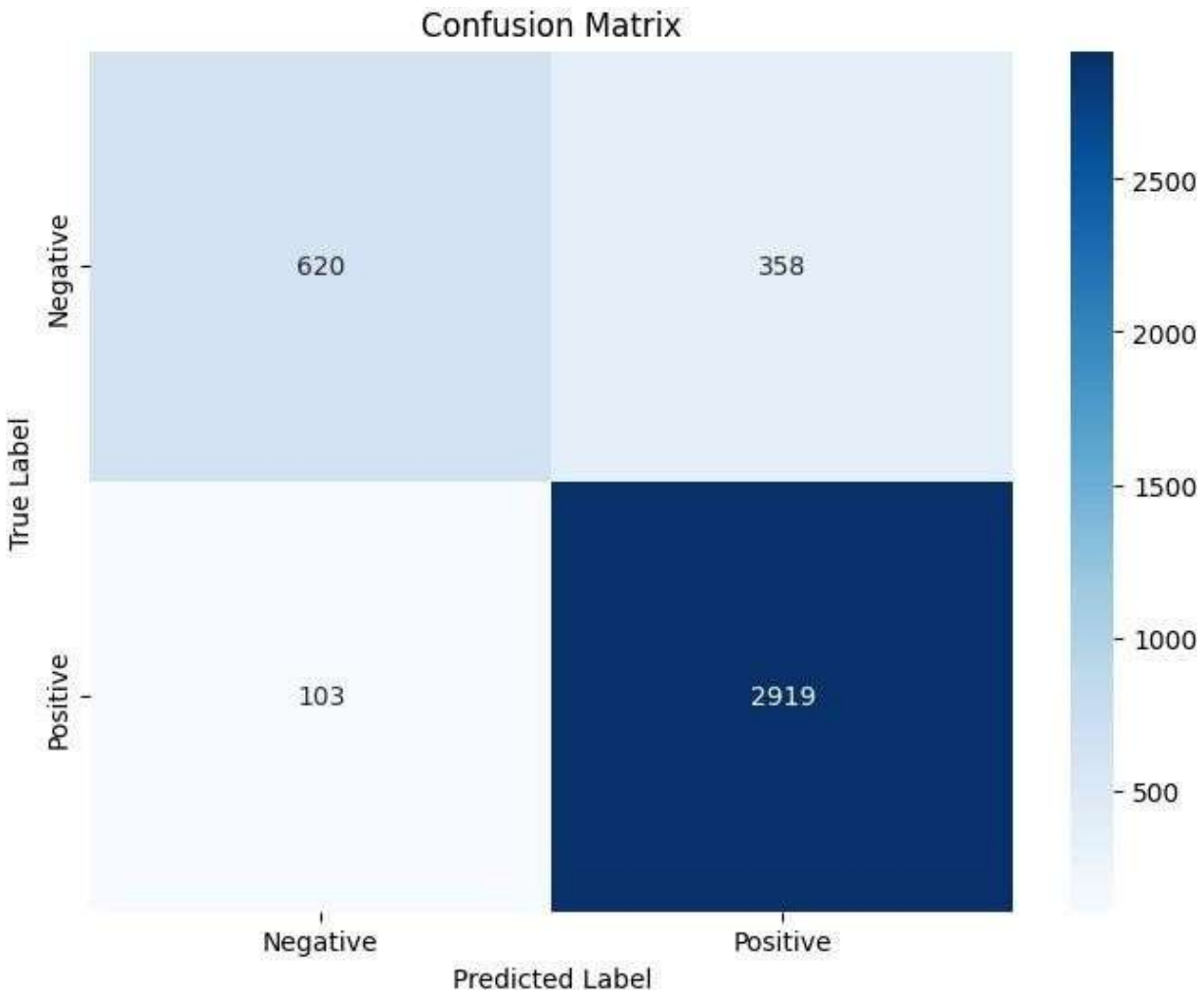
```
True
```

```
import nltk

# Add the resource path to NLTK
nltk.data.path.append('/usr/share/nltk_data')  # Ensure nltk looks in
the correct directory

# Download the punkt resource and punkt_tab resource
nltk.download('punkt', download_dir='/usr/share/nltk_data')  # Force
download to the specified path
nltk.download('punkt_tab', download_dir='/usr/share/nltk_data')  #
Ensure punkt_tab is also downloaded
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /usr/share/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
# Data Preprocessing Functions

# Clean the text data def clean_text(text):     text =
re.sub(r'\W', ' ', text)  # Remove non-alphanumeric characters
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
text = text.lower()  # Convert to lowercase      text =
re.sub(r'\d+', '', text)  # Remove numbers      stop_words =
set(stopwords.words('english'))  # Stopwords      text = '
'.join([word for word in text.split() if word not in stop_words])
return text

# Lemmatize words
lemmatizer = WordNetLemmatizer()

def lemmatize_words(text):
tokens = word_tokenize(text)
    return ' '.join([lemmatizer.lemmatize(word) for word in tokens])

# Apply the cleaning and lemmatization functions
# Apply the cleaning and lemmatization functions
df['cleaned_reviews'] = df['Text'].apply(clean_text)  # Use 'Text' as
```

```
the column name
df['lemmatized_reviews'] =
df['cleaned_reviews'].apply(lemmatize_words)

# Vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['lemmatized_reviews']).toarray()

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, df['label'],
test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)
LogisticRegression()
# Predict the sentiment of the test set
y_pred = model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='binary')  #
Adjust for multiclass if needed
recall = recall_score(y_test, y_pred, average='binary')
f1 = f1_score(y_test, y_pred, average='binary')

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

Accuracy: 0.88475
Precision: 0.8907537381751602
Recall: 0.9659166115155526
F1 Score: 0.9268137799650739

# Generate and visualize the confusion matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Negative', 'Positive'], yticklabels=['Negative',
'Positive'])
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
plt.show()
```




```
import joblib

# Save the model
joblib.dump(model, 'sentiment_model.pkl')

# Save the vectorizer
joblib.dump(vectorizer, 'vectorizer.pkl')
['vectorizer.pkl']
def predict_sentiment(text):
cleaned_text = clean_text(text)
    lemmatized_text = lemmatize_words(cleaned_text)
vectorized_text =
vectorizer.transform([lemmatized_text]).toarray()
return model.predict(vectorized_text)

# Test with a new review
new_review = "This product is amazing! Highly recommend it."
```

```
sentiment = predict_sentiment(new_review)
print(f"The sentiment of the review is: {sentiment[0]}")
```
```
The sentiment of the review is: 1
```
```
import joblib

# Save the trained model
joblib.dump(model, 'sentiment_model.pkl')

# Save the TF-IDF vectorizer
joblib.dump(vectorizer, 'vectorizer.pkl')
```
```
['vectorizer.pkl']
```
```
!zip sentiment_analysis_files384.zip sentiment_model.pkl
vectorizer.pkl
```
```
  adding: sentiment_model.pkl (deflated 5%)
  adding: vectorizer.pkl (deflated 72%)
```
```
!ls
```
```
amazon.csv   sentiment_analysis_files384.zip sentiment_model.pkl
sample_data sentiment_analysis_files.zip      vectorizer.pkl
```
```
from google.colab import files
files.download('sentiment_analysis_files384.zip')
```
```
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
```

DONE BY-
DEVAKATHE VIKAS
2211CS020125
AIML-BETA