**Stock sentiment analysis**



COIMBATORE INSTITUTE OF TECHNOLOGY

COIMBATORE – 641 014 TAMILNADU, INDIA

Done by:

Vikash V.D

Rishvanthan R.R

**Abstract**

The objective of this project was to analyze stock-related news articles for companies like Apple, Tesla, Amazon, Microsoft, and Google. Using Natural Language Processing (NLP) techniques, we scraped the latest news headlines, applied Part-of-Speech (POS) tagging, Named Entity Recognition (NER), and sentiment analysis to understand the sentiment and structure of the headlines. The processed data was stored in a knowledge base, which allowed for semantic analysis to retrieve relevant headlines based on user queries. Finally, the project employed a ML model to classify stock movements as "up" or "down" based on the sentiment and analysis of the headlines

**Code:**

**Web scraping:**

import requests

from bs4 import BeautifulSoup

import pandas as pd

import nltk

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

import spacy

# Download required NLTK data

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

# Load the small English model from spacy

nlp = spacy.load("en\_core\_web\_sm")

def fetch\_stock\_news(symbol, company\_name):

url = f'https://finviz.com/quote.ashx?t={symbol}&p=d'

headers = {'User-Agent': 'Mozilla/5.0'}

for attempt in range(3):

try:

response = requests.get(url, headers=headers)

response.raise\_for\_status()

soup = BeautifulSoup(response.content, 'html.parser')

news\_table = soup.find('table', class\_='fullview-news-outer')

if not news\_table:

print("News table not found.")

return pd.DataFrame()

news\_rows = news\_table.findAll('tr')

data = []

for row in news\_rows:

date\_data = row.td.text.strip().split(' ')

if len(date\_data) == 2:

date, time = date\_data

else:

date = date\_data[0]

time = None

headline = row.a.text.strip()

# Perform POS tagging

tokens = nltk.word\_tokenize(headline)

pos\_tags = nltk.pos\_tag(tokens)

# Perform Named Entity Recognition (NER) using spacy

doc = nlp(headline)

named\_entities = [(ent.text, ent.label\_) for ent in doc.ents]

# Analyze sentiment using VADER

analyzer = SentimentIntensityAnalyzer()

sentiment\_score = analyzer.polarity\_scores(headline)

compound\_score = sentiment\_score['compound']

# Categorize sentiment based on the compound score

if compound\_score >= 0.05:

sentiment = "positive"

elif compound\_score <= -0.05:

sentiment = "negative"

else:

sentiment = "neutral"

# Append data

data.append([date, headline, company\_name, pos\_tags, named\_entities, sentiment])

# Create a DataFrame from the data

df = pd.DataFrame(data, columns=['Date', 'Headline', 'Company', 'POS\_Tags', 'Named\_Entities', 'Sentiment'])

# Convert 'Date' to a proper format if necessary

df['Date'] = pd.to\_datetime(df['Date'], errors='coerce').dt.date # Store only the date

# Display the 5 most recent headlines with POS tags and named entities

print("\nRecent 5 Headlines for", company\_name)

recent\_headlines = df.head(5) # Display the first 5 rows

for index, row in recent\_headlines.iterrows():

print(f"\nHeadline {index + 1}: {row['Headline']}")

print(f"POS Tags: {row['POS\_Tags']}")

print(f"Named Entities: {row['Named\_Entities']}")

return df

except requests.HTTPError as e:

print(f"Attempt {attempt + 1} failed: {e}. Retrying...")

print(f"Failed after 3 attempts: {e}")

return pd.DataFrame()

def append\_to\_excel(df, file\_name='knowledgebase2.xlsx'):

try:

# Read existing data if the file exists

try:

existing\_data = pd.read\_excel(file\_name)

df = pd.concat([existing\_data, df], ignore\_index=True)

except FileNotFoundError:

print(f"{file\_name} not found. Creating a new file.")

# Write the updated DataFrame to the Excel file

with pd.ExcelWriter(file\_name, engine='openpyxl', mode='w') as writer:

df.to\_excel(writer, index=False)

print(f"Data successfully appended to {file\_name}.")

except Exception as e:

print(f"Error while saving to Excel: {e}")

def main():

company\_dict = {

1: ('TSLA', 'Tesla'),

2: ('AAPL', 'Apple'),

3: ('AMZN', 'Amazon'),

4: ('MSFT', 'Microsoft'),

5: ('GOOGL', 'Google')

} print("Choose a company to analyze:")

for key, (symbol, name) in company\_dict.items():

print(f"{key}. {name}")

choice = int(input("Enter the number of the company: "))

if choice in company\_dict:

symbol, company\_name = company\_dict[choice]

print(f"Fetching data for {company\_name} from {symbol}...")

df = fetch\_stock\_news(symbol, company\_name)

if not df.empty:

append\_to\_excel(df)

else:

print("No data to save. Please try again later.")

else:

print("Invalid choice. Please select a valid company number.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Semantics:

import pandas as pd

import re

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import spacy

from textblob import TextBlob

nlp = spacy.load("en\_core\_web\_sm")

knowledge\_base = pd.read\_excel('knowledgebase2.xlsx')

print("Loaded knowledge base from knowledgebase2.xlsx.")

predefined\_companies = ['Apple', 'Tesla', 'Amazon', 'Microsoft', 'Google', 'Elon Musk']

common\_spelling\_corrections = {

"teh": "the",

"negotive": "negative",

"positivee": "positive",

"splling": "spelling"

}

def correct\_spelling(query):

query\_lower = query.lower()

company\_name\_in\_query = None

for company in predefined\_companies:

if company.lower() in query\_lower:

company\_name\_in\_query = company

query\_lower = re.sub(r'\b' + re.escape(company.lower()) + r'\b', "COMPANY\_PLACEHOLDER", query\_lower)

corrected\_words = []

for word in query\_lower.split():

if word in common\_spelling\_corrections:

corrected\_words.append(common\_spelling\_corrections[word])

elif word != "company\_placeholder":

corrected\_words.append(str(TextBlob(word).correct()))

else:

corrected\_words.append(word)

corrected\_query = ' '.join(corrected\_words)

if company\_name\_in\_query:

corrected\_query = corrected\_query.replace("COMPANY\_PLACEHOLDER", company\_name\_in\_query)

original\_words = set(query.lower().split())

corrected\_words\_set = set(corrected\_query.lower().split())

uncorrected\_words = original\_words - corrected\_words\_set

if uncorrected\_words:

print(f"Warning: The following words were not corrected: {', '.join(uncorrected\_words)}")

return corrected\_query

def understand\_query(query):

doc = nlp(query)

entities = [(ent.text, ent.label\_) for ent in doc.ents]

keywords = [token.text.lower() for token in doc if token.is\_alpha and not token.is\_stop]

chunks = [chunk.text for chunk in doc.noun\_chunks]

print(f"Entities identified: {entities}")

print(f"Keywords identified: {keywords}")

print(f"Noun chunks identified: {chunks}")

return entities, keywords, chunks

def retrieve\_relevant\_headlines(query, knowledge\_base, top\_n=5):

company, sentiment = extract\_company\_and\_sentiment(query)

if company is None:

print("No company found in the query. Please specify a company.")

return

company\_headlines = knowledge\_base[knowledge\_base['Company'].str.lower() == company.lower()]

if sentiment:

company\_headlines = company\_headlines[company\_headlines['Sentiment'].str.lower() == sentiment.lower()]

company\_headlines = company\_headlines.drop\_duplicates(subset=['Headline'])

cleaned\_query = clean\_text(query)

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(

[cleaned\_query] + [clean\_text(str(headline)) for headline in company\_headlines['Headline']]

)

query\_vector = tfidf\_matrix[0]

headline\_vectors = tfidf\_matrix[1:]

similarity\_scores = cosine\_similarity(query\_vector, headline\_vectors).flatten()

company\_headlines['Similarity Score'] = similarity\_scores

company\_headlines\_sorted = company\_headlines.sort\_values(

by='Similarity Score', ascending=False

).head(top\_n)

print(f"\nTop {top\_n} relevant headlines based on your query for '{company}':\n")

for index, row in company\_headlines\_sorted.iterrows():

print(f"Headline: {row['Headline']}")

print(f"Company: {row['Company']}")

print(f"Sentiment: {row['Sentiment']}")

print(f"Similarity Score: {row['Similarity Score']:.4f}\n")

def extract\_company\_and\_sentiment(query):

query = query.lower()

company = None

sentiment = None

if 'positive' in query:

sentiment = 'positive'

elif 'negative' in query:

sentiment = 'negative'

for known\_company in predefined\_companies:

if known\_company.lower() in query:

company = known\_company

break

return company, sentiment

def clean\_text(text):

text = text.lower()

text = re.sub(r'\W+', ' ', text)

return text

def main():

query = input("Enter your query: ")

corrected\_query = correct\_spelling(query)

print(f"Corrected query: {corrected\_query}")

entities, keywords, chunks = understand\_query(corrected\_query)

print(f"Noun chunks used in analysis: {chunks}")

retrieve\_relevant\_headlines(corrected\_query, knowledge\_base)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Prediction:**

import pandas as pd

import re

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

df = pd.read\_csv('Dataset.csv', encoding="ISO-8859-1")

train, test = train\_test\_split(df, test\_size=0.2, random\_state=42)

data = train.iloc[:, 2:27]

data.replace("[^a-zA-Z]", " ", regex=True, inplace=True)

list1 = [i for i in range(25)]

new\_Index = [str(i) for i in list1]

data.columns = new\_Index

stop\_words = set(stopwords.words('english'))

stemmer = PorterStemmer()

def preprocess\_text(text):

if isinstance(text, float):

text = ''

text = re.sub('[^a-zA-Z]', ' ', text)

text = text.lower()

words = text.split()

words = [stemmer.stem(word) for word in words if word not in stop\_words] # Remove stopwords and stem words

return ' '.join(words)

for index in new\_Index:

data[index] = data[index].astype(str).apply(preprocess\_text)

headlines = []

for row in range(0, len(data.index)):

headlines.append(' '.join(str(x) for x in data.iloc[row, 0:25]))

tfidfvector = TfidfVectorizer(ngram\_range=(1, 2), max\_features=15000, min\_df=3)

traindataset = tfidfvector.fit\_transform(headlines)

scaler = StandardScaler(with\_mean=False)

traindataset\_scaled = scaler.fit\_transform(traindataset)

test\_data = test.iloc[:, 2:27]

test\_data.replace("[^a-zA-Z]", " ", regex=True, inplace=True)

test\_data.columns = new\_Index

for index in new\_Index:

test\_data[index] = test\_data[index].astype(str).apply(preprocess\_text)

test\_transform = []

for row in range(0, len(test\_data.index)):

test\_transform.append(' '.join(str(x) for x in test\_data.iloc[row, 0:25]))

est\_dataset = tfidfvector.transform(test\_transform)

test\_dataset\_scaled = scaler.transform(test\_dataset)

random\_forest = RandomForestClassifier(

n\_estimators=500,

max\_depth=30,

min\_samples\_split=4,

min\_samples\_leaf=1,

criterion='entropy',

class\_weight='balanced\_subsample',

random\_state=42

)

logistic\_regression = LogisticRegression(

max\_iter=300,

class\_weight='balanced',

random\_state=42

)

# Train the models

random\_forest.fit(traindataset\_scaled, train['Label'])

logistic\_regression.fit(traindataset\_scaled, train['Label'])

print("Select a company from the list:")

print("1. Amazon")

print("2. Tesla")

print("3. Apple")

print("4. Microsoft")

print("5. Google")

choice = int(input("Enter the number corresponding to your choice: "))

company\_dict = {

1: "Amazon",

2: "Tesla",

3: "Apple",

4: "Microsoft",

5: "Google"

}

selected\_company = company\_dict.get(choice, None)

if not selected\_company:

print("Invalid choice. Please restart and choose a valid number.")

exit()

knowledgebase = pd.read\_excel('knowledgebase2.xlsx')

# Filter headlines for the selected company and ensure only up to 25 are used

company\_headlines = knowledgebase[knowledgebase['Company'] == selected\_company]['Headline'].tolist()[:25]

# Preprocess the headlines

preprocessed\_headlines = [preprocess\_text(headline) for headline in company\_headlines]

# Transform the headlines into TF-IDF vectors

company\_tfidf = tfidfvector.transform(preprocessed\_headlines)

company\_tfidf\_scaled = scaler.transform(company\_tfidf)

# Make predictions using both models and average the probabilities

rf\_predictions = random\_forest.predict\_proba(company\_tfidf\_scaled)[:, 1]

lr\_predictions = logistic\_regression.predict\_proba(company\_tfidf\_scaled)[:, 1]

final\_predictions = (rf\_predictions + lr\_predictions) / 2 > 0.5

final\_predictions = final\_predictions.astype(int)

if final\_predictions.mean() > 0.5:

print(f"The prediction indicates that {selected\_company}'s stock is likely to go up.")

else:

print(f"The prediction indicates that {selected\_company}'s stock is likely to go down.")

**Main:**

import streamlit as st

import pandas as pd

from app import fetch\_stock\_news, append\_to\_excel

from semantics import correct\_spelling, understand\_query, retrieve\_relevant\_headlines

from prediction import preprocess\_text, tfidfvector, scaler, random\_forest, logistic\_regression

knowledge\_base\_file = 'knowledgebase2.xlsx'

knowledge\_base = pd.read\_excel(knowledge\_base\_file)

company\_dict = {

1: ('TSLA', 'Tesla'),

2: ('AAPL', 'Apple'),

3: ('AMZN', 'Amazon'),

4: ('MSFT', 'Microsoft'),

5: ('GOOGL', 'Google')

}

st.title("Stock Analysis & Prediction")

st.sidebar.header("Select a Company")

company\_choice = st.sidebar.selectbox(

"Choose a company to analyze:",

list(company\_dict.keys()),

format\_func=lambda x: company\_dict[x][1]

)

symbol, company\_name = company\_dict[company\_choice]

st.write(f"\*\*Selected Company:\*\* {company\_name}")

if st.button("Fetch Latest Headlines"):

st.write(f"Fetching latest news for {company\_name}...")

df = fetch\_stock\_news(symbol, company\_name)

if df is not None and not df.empty:

append\_to\_excel(df, file\_name=knowledge\_base\_file)

st.write("Headlines successfully fetched and saved to the knowledge base.")

st.dataframe(df.head())

else:

st.write("No new headlines retrieved. Please try again later.")

st.header("Query Analysis")

query = st.text\_input("Enter your query about the stock or company:")

if st.button("Analyze Query"):

if query:

st.write(f"Analyzing the query: {query}")

corrected\_query = correct\_spelling(query)

st.write(f"\*\*Corrected Query:\*\* {corrected\_query}")

entities, keywords, chunks = understand\_query(corrected\_query)

st.write(f"\*\*Identified Entities:\*\* {entities}")

st.write(f"\*\*Extracted Keywords:\*\* {keywords}")

st.write(f"\*\*Identified Noun Chunks:\*\* {chunks}")

st.write("Retrieving relevant headlines...")

try:

top\_n = 5

relevant\_headlines = retrieve\_relevant\_headlines(corrected\_query, knowledge\_base, top\_n)

if relevant\_headlines is not None and not relevant\_headlines.empty:

st.table(relevant\_headlines[['Date', 'Time', 'Headline', 'Sentiment', 'Similarity Score']])

else:

st.write(f"No relevant headlines found based on the query for '{company\_name}'.")

except Exception as e:

st.write(f"Error retrieving headlines: {e}")

st.header("Stock Movement Prediction")

if st.button("Predict Stock Movement"):

st.write(f"Predicting stock movement for {company\_name}...")

company\_headlines = knowledge\_base[knowledge\_base['Company'] == company\_name]['Headline'].tolist()

preprocessed\_headlines = [preprocess\_text(headline) for headline in company\_headlines]

company\_tfidf = tfidfvector.transform(preprocessed\_headlines)

company\_tfidf\_scaled = scaler.transform(company\_tfidf)

rf\_predictions = random\_forest.predict\_proba(company\_tfidf\_scaled)[:, 1]

lr\_predictions = logistic\_regression.predict\_proba(company\_tfidf\_scaled)[:, 1]

final\_predictions = (rf\_predictions + lr\_predictions) / 2 > 0.5

final\_predictions = final\_predictions.astype(int)

if final\_predictions.mean() > 0.5:

st.write(f"The prediction indicates that \*\*{company\_name}'s stock is likely to go up.\*\*")

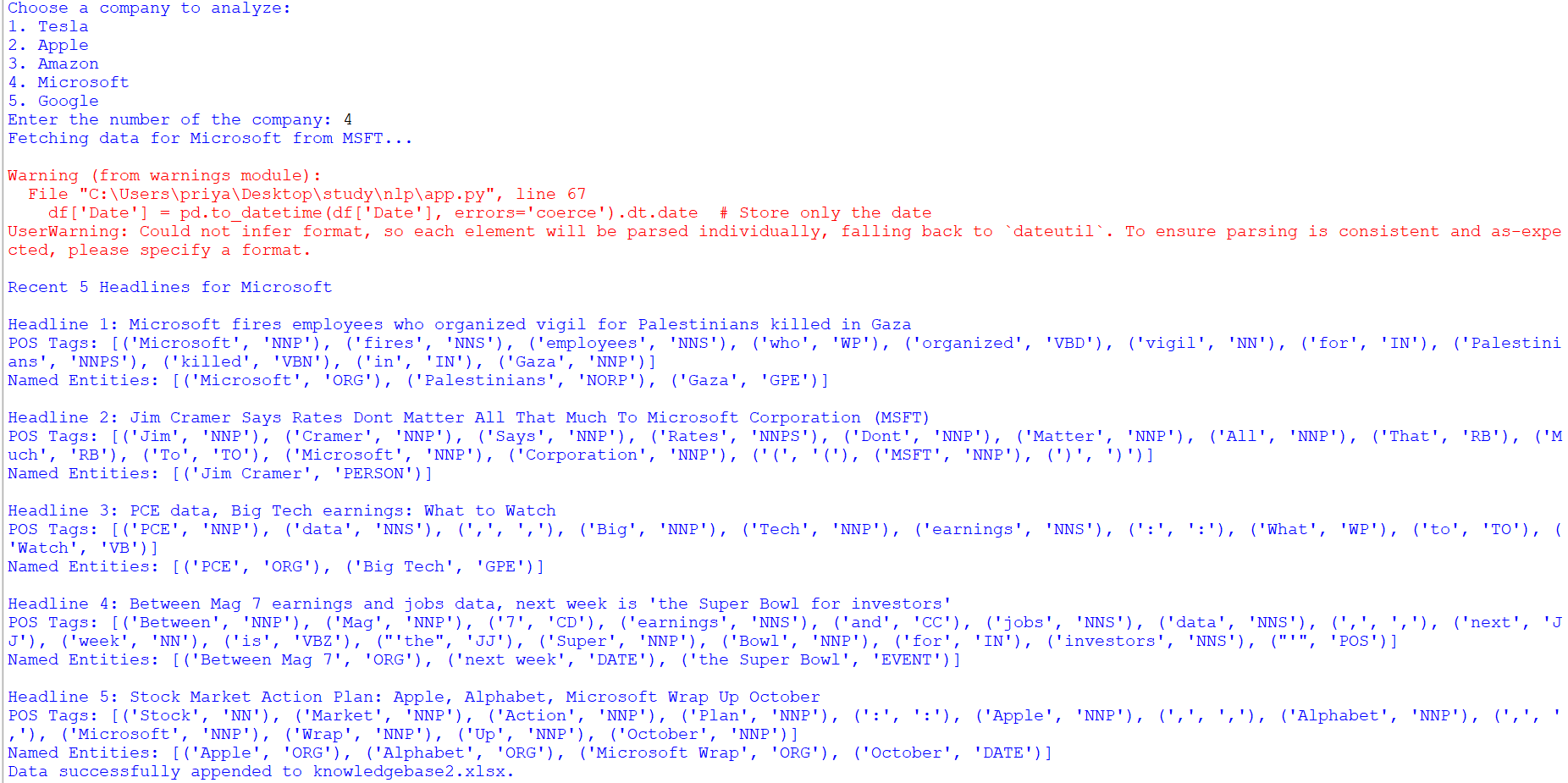
else:

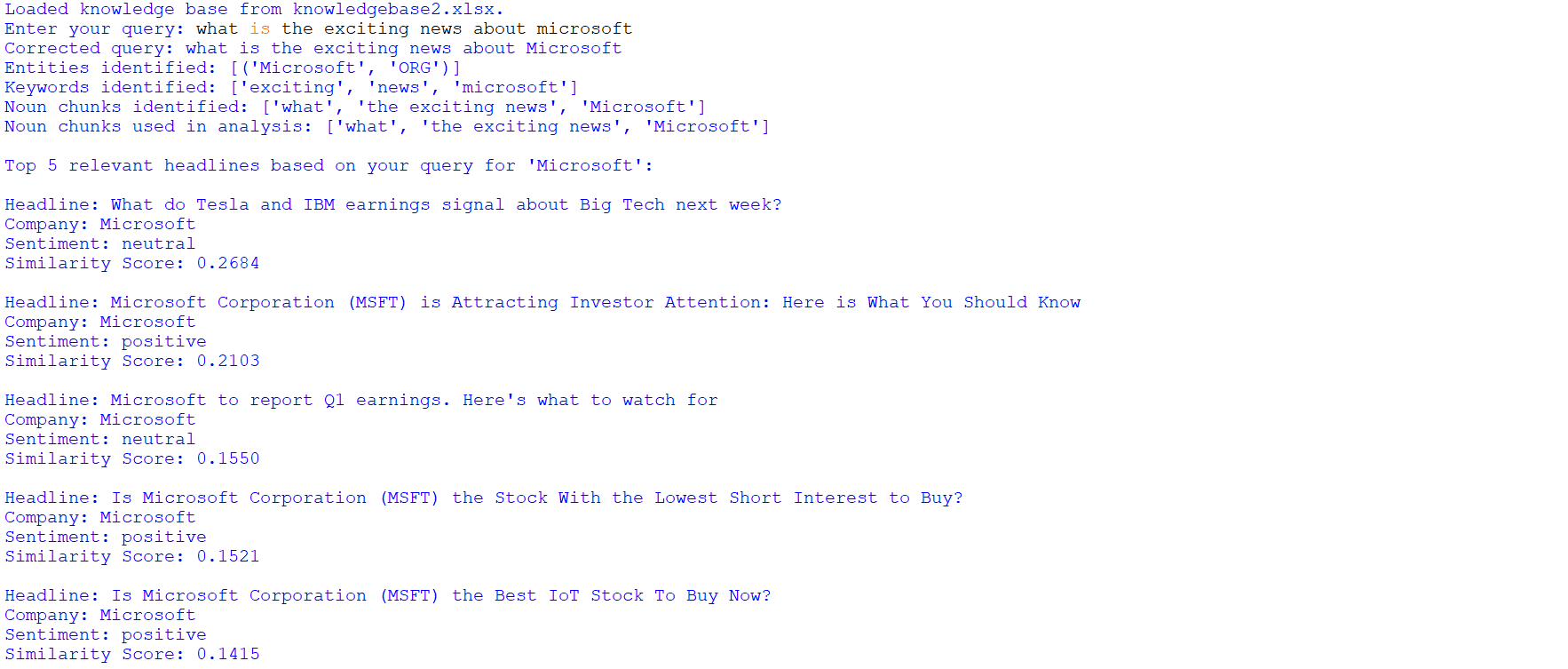
st.write(f"The prediction indicates that \*\*{company\_name}'s stock is likely to go down.\*\*")

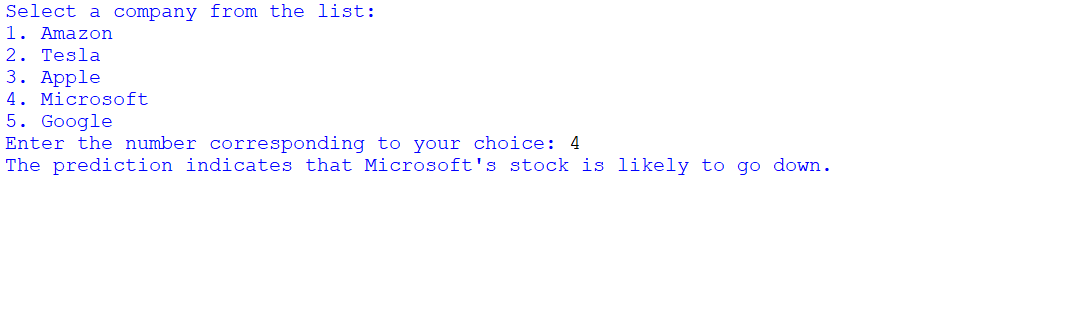
**Output:**

The output included:

* Creating Knowledge Base: An Excel file containing headlines, sentiment scores, POS tags, and identified entities.
* Semantic Analysis: Retrieved the most relevant headlines based on user query, displaying those with the highest similarity scores.
* Stock Movement Predictions: Forecasts whether a company’s stock is likely to move up or down based on the analysis of sentiment of the headlines with the help of ML models.







**Conclusion**

The project successfully demonstrated how NLP techniques could be applied to analyze stock-related news articles. We web scraped the latest article headlines of five companies, applied POS tagging and NER, and analyzed the sentiment, storing this data in a knowledge base. A semantic analysis module matched user queries with relevant headlines, providing insights into the news most pertinent to the query. The stock movement prediction model helped forecast potential stock trends based on sentiment, offering a valuable tool for investors. The implementation of retry mechanisms and data storage safety checks ensured robustness and reliability throughout the analysis, making this project adaptable for a broader range of companies and financial needs.