Problem Statement

Business Context

The prices of the stocks of companies listed under a global exchange are influenced by a variety of factors, with the company's financial performance, innovations and collaborations, and market sentiment being factors that play a significant role. News and media reports can rapidly affect investor perceptions and, consequently, stock prices in the highly competitive financial industry. With the sheer volume of news and opinions from a wide variety of sources, investors and financial analysts often struggle to stay updated and accurately interpret its impact on the market. As a result, investment firms need sophisticated tools to analyze market sentiment and integrate this information into their investment strategies.

Problem Definition

With an ever-rising number of news articles and opinions, an investment startup aims to leverage artificial intelligence to address the challenge of interpreting stock-related news and its impact on stock prices. They have collected historical daily news for a specific company listed under NASDAQ, along with data on its daily stock price and trade volumes.

As a member of the Data Science and AI team in the startup, you have been tasked with analyzing the data, developing an AI-driven sentiment analysis system that will automatically process and analyze news articles to gauge market sentiment, and summarizing the news at a weekly level to enhance the accuracy of their stock price predictions and optimize investment strategies. This will empower their financial analysts with actionable insights, leading to more informed investment decisions and improved client outcomes.

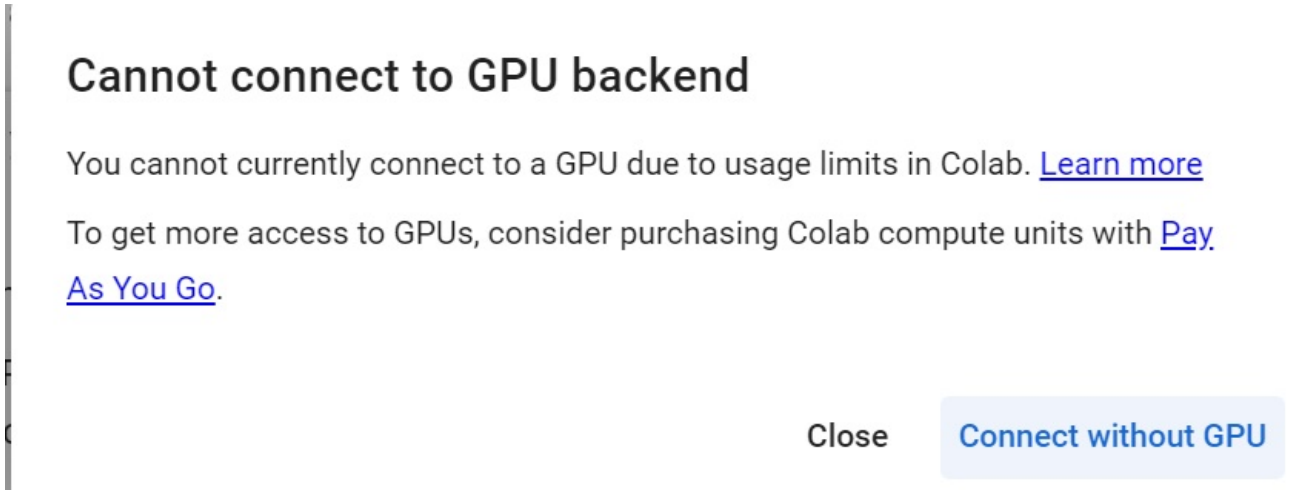
Data Dictionary

- `Date` : The date the news was released
- `News` : The content of news articles that could potentially affect the company's stock price
- `Open` : The stock price (in $) at the beginning of the day
- `High` : The highest stock price (in $) reached during the day
- `Low` : The lowest stock price (in $) reached during the day
- `Close` : The adjusted stock price (in $) at the end of the day
- `Volume` : The number of shares traded during the day
- `Label` : The sentiment polarity of the news content
  - 1: positive
  - 0: neutral
  - -1: negative

Please read the instructions carefully before starting the project.

Note: If the free-tier GPU of Google Colab is not accessible (due to unavailability or exhaustion of daily limit or other reasons), the following steps can be taken:

1. Wait for 12-24 hours until the GPU is accessible again or the daily usage limits are reset.
2. Switch to a different Google account and resume working on the project from there.
3. Try using the CPU runtime:
   - To use the CPU runtime, click on *Runtime => Change runtime type => CPU => Save*
   - One can also click on the *Continue without GPU* option to switch to a CPU runtime (kindly refer to the snapshot below)
   - The instructions for running the code on the CPU are provided in the respective sections of the notebook.



Installing and Importing Necessary Libraries

In [ ]:
```
# installing the sentence-transformers and gensim libraries for word embeddings
!pip install -U sentence-transformers gensim transformers tqdm -q
```

In [ ]:
```
# to read and manipulate the data
import pandas as pd
import numpy as np
pd.set_option('max_colwidth', None)    # setting column to the maximum column width as per the data

# to visualise data
import matplotlib.pyplot as plt
import seaborn as sns

# to use regular expressions for manipulating text data
import re

# to load the natural language toolkit
import nltk
nltk.download('stopwords')    # loading the stopwords
nltk.download('wordnet')    # loading the wordnet module that is used in stemming

# to remove common stop words
from nltk.corpus import stopwords

# to perform stemming
from nltk.stem.porter import PorterStemmer

# To encode the target variable
from sklearn.preprocessing import LabelEncoder

# To import Word2Vec
from gensim.models import Word2Vec

# To tune the model
from sklearn.model_selection import GridSearchCV

# Converting the Stanford GloVe model vector format to word2vec
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors

# To split data into train and test sets
from sklearn.model_selection import train_test_split

# To build a Random Forest model
from sklearn.ensemble import RandomForestClassifier

# To compute metrics to evaluate the model
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score, classifi
```

```
[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!
```

Loading the dataset

In [ ]:
```
# mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
# loading the dataset
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/stock_news.csv')
```

In [ ]:
```
# creating a copy of the dataset
data = df.copy()
```

Data Overview

Checking the first five rows of the data

In [ ]:
```
data.head()
```

Out[ ]:

	Date	News	Open	High	Low	Close	Volume	Label
0	2019-01-02	The tech sector experienced a significant decline in the aftermarket following Apple's Q1 revenue warning. Notable suppliers, including Skyworks, Broadcom, Lumentum, Qorvo, and TSMC, saw their stocks drop in response to Apple's downward revision of its revenue expectations for the quarter, previously announced in January.	41.740002	42.244999	41.482498	40.246914	130672400	-1
1	2019-01-02	Apple lowered its fiscal Q1 revenue guidance to 84billionfromearlierestimatesof89-$93 billion due to weaker than expected iPhone sales. The announcement caused a significant drop in Apple's stock price and negatively impacted related suppliers, leading to broader market declines for tech indices such as Nasdaq 10	41.740002	42.244999	41.482498	40.246914	130672400	-1
2	2019-01-02	Apple cut its fiscal first quarter revenue forecast from 89 − 93 billion to $84 billion due to weaker demand in China and fewer iPhone upgrades. CEO Tim Cook also mentioned constrained sales of Airpods and Macbooks. Apple's shares fell 8.5% in post market trading, while Asian suppliers like Hon	41.740002	42.244999	41.482498	40.246914	130672400	-1
3	2019-01-02	This news article reports that yields on long-dated U.S. Treasury securities hit their lowest levels in nearly a year on January 2, 2019, due to concerns about the health of the global economy following weak economic data from China and Europe, as well as the partial U.S. government shutdown. Apple	41.740002	42.244999	41.482498	40.246914	130672400	-1
4	2019-01-02	Apple's revenue warning led to a decline in USD JPY pair and a gain in Japanese yen, as investors sought safety in the highly liquid currency. Apple's underperformance in Q1, with forecasted revenue of 84billioncomparedtoanalystexpectationsof91.5 billion, triggered risk aversion mood in markets	41.740002	42.244999	41.482498	40.246914	130672400	-1

In [ ]:
```
# displaying news text
data.loc[3, 'News']
```

Out[ ]:
```
' This news article reports that yields on long-dated U.S. Treasury securities hit their lowest levels in nearly a year on January 2, 2019, due to concerns about the health of the global economy following weak economic data from China and Europe, as well as the partial U.S. government shutdown. Apple'
```

Checking the last five rows of the data

In [ ]:
```
data.tail()
```

Out[ ]:

	Date	News	Open	High	Low	Close	Volume	Label
344	2019-04-30	Media mogul Oprah Winfrey, known for influencing millions with her opinions on diets and books, is considering which Democratic presidential candidate to endorse in 2020. She told the Hollywood Reporter she's "quietly figuring out where I'm going to use my voice" and will make a clear announcement	50.764999	50.849998	49.7775	48.70879	186139600	-1
345	2019-04-30	European shares fell on Tuesday, with banks underperforming amid a decline in China's manufacturing activity and awaiting euro zone economic growth numbers. The pan-European STOXX 600 index dropped 0.7% while major indices fell except London's FTSE 100. Danske Bank plunged	50.764999	50.849998	49.7775	48.70879	186139600	-1
346	2019-04-30	This article reports that the S&P 500 reached another record high close on Tuesday, marking its best four-month stretch since late 2010. Apple's strong quarterly results and positive earnings forecast helped ease concerns about the bull run's sustainability, despite a revenue miss from Google parent Alphabet. The	50.764999	50.849998	49.7775	48.70879	186139600	-1
347	2019-04-30	The Federal Reserve is anticipated to keep interest rates unchanged in their upcoming meeting, with a likelihood of a rate cut expected later this year. The Fed Chairman's press conference may provide significant market impact as investors seek insights on economic growth and inflation. Apple's earnings report exceeded expectations, leading to a post-market surge in shares, while	50.764999	50.849998	49.7775	48.70879	186139600	-1
348	2019-04-30	In the first quarter, South Korea's Samsung Electronics reported its weakest profit in over two years due to falls in chip prices and slowing demand for display panels. The tech giant expects improved results in the second half of 2019, driven by a pickup in memory chip and smartphone sales. However, memory chip	50.764999	50.849998	49.7775	48.70879	186139600	0

Checking for missing values

In [ ]:
```
data.isnull().sum()
```

Out[ ]:

	0
Date	0
News	0
Open	0
High	0
Low	0
Close	0
Volume	0
Label	0

dtype: int64

- There seems to be no missing values throughout the data.

Statistical Summary

In [ ]:
```
data.describe()
```

Out[ ]:

	Open	High	Low	Close	Volume	Label
count	349.000000	349.000000	349.000000	349.000000	3.490000e+02	349.000000
mean	46.229233	46.700458	45.745394	44.926317	1.289482e+08	-0.054441
std	6.442817	6.507321	6.391976	6.398338	4.317031e+07	0.715119
min	37.567501	37.817501	37.305000	36.254131	4.544800e+07	-1.000000
25%	41.740002	42.244999	41.482498	40.246914	1.032720e+08	-1.000000
50%	45.974998	46.025002	45.639999	44.596924	1.156272e+08	0.000000
75%	50.707500	50.849998	49.777500	49.110790	1.511252e+08	0.000000
max	66.817497	67.062500	65.862503	64.805229	2.444392e+08	1.000000

Exploratory Data Analysis

Graphs for EDA

In [ ]:
```
# Function to Create Labled Barplots for EDA

def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

In [ ]:
```
# Plot Distribution

def plot_distribution(variable, title, xlabel):
    plt.figure(figsize=(8, 5))
    sns.histplot(df[variable], kde=True)
    plt.title(f"Distribution of {title}")
    plt.xlabel(xlabel)
    plt.ylabel("Frequency")
    plt.show()
```

Univariate Analysis

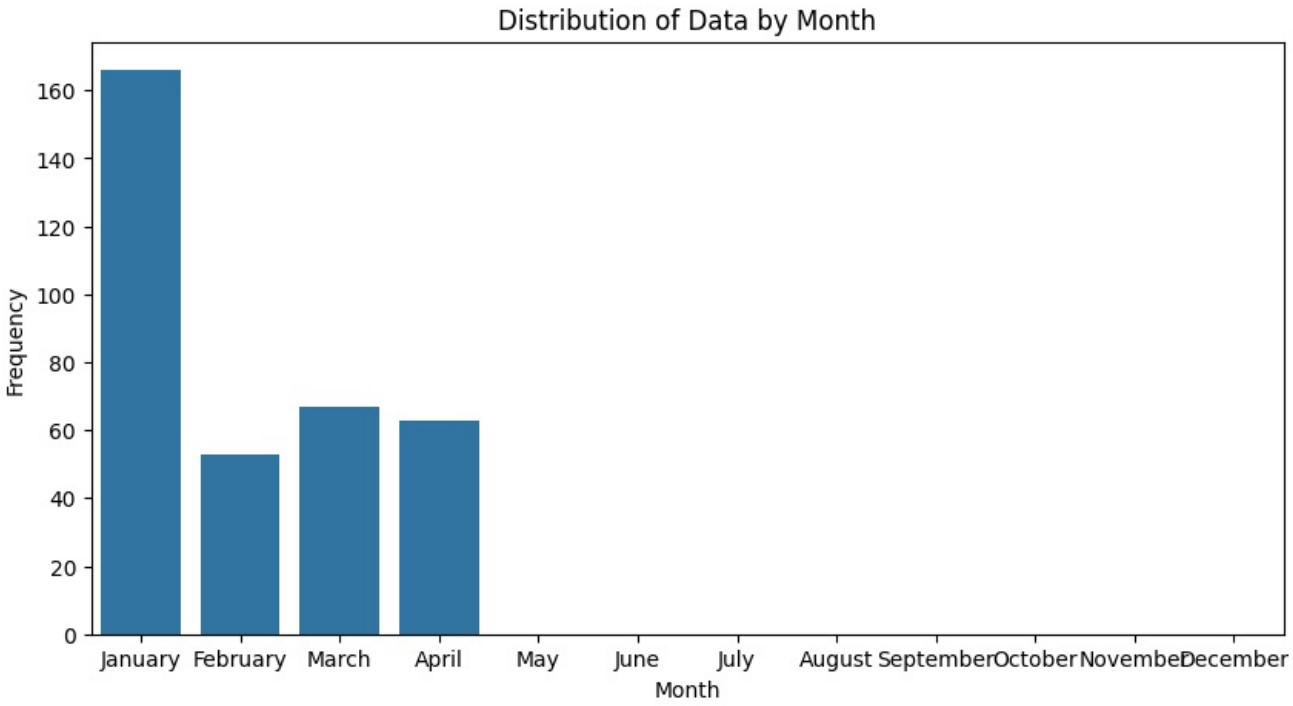
- Distribution of individual variables
- Compute and check the distribution of the length of news content

Distribution of Date

In [ ]:
```
# Convert 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])
```

In [ ]:
```
df['Month'] = df['Date'].dt.month_name()  # Extract the month
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='Month', order=['January', 'February', 'March', 'April', 'May', 'June',
                                         'July', 'August', 'September', 'October', 'November', 'December'])
```

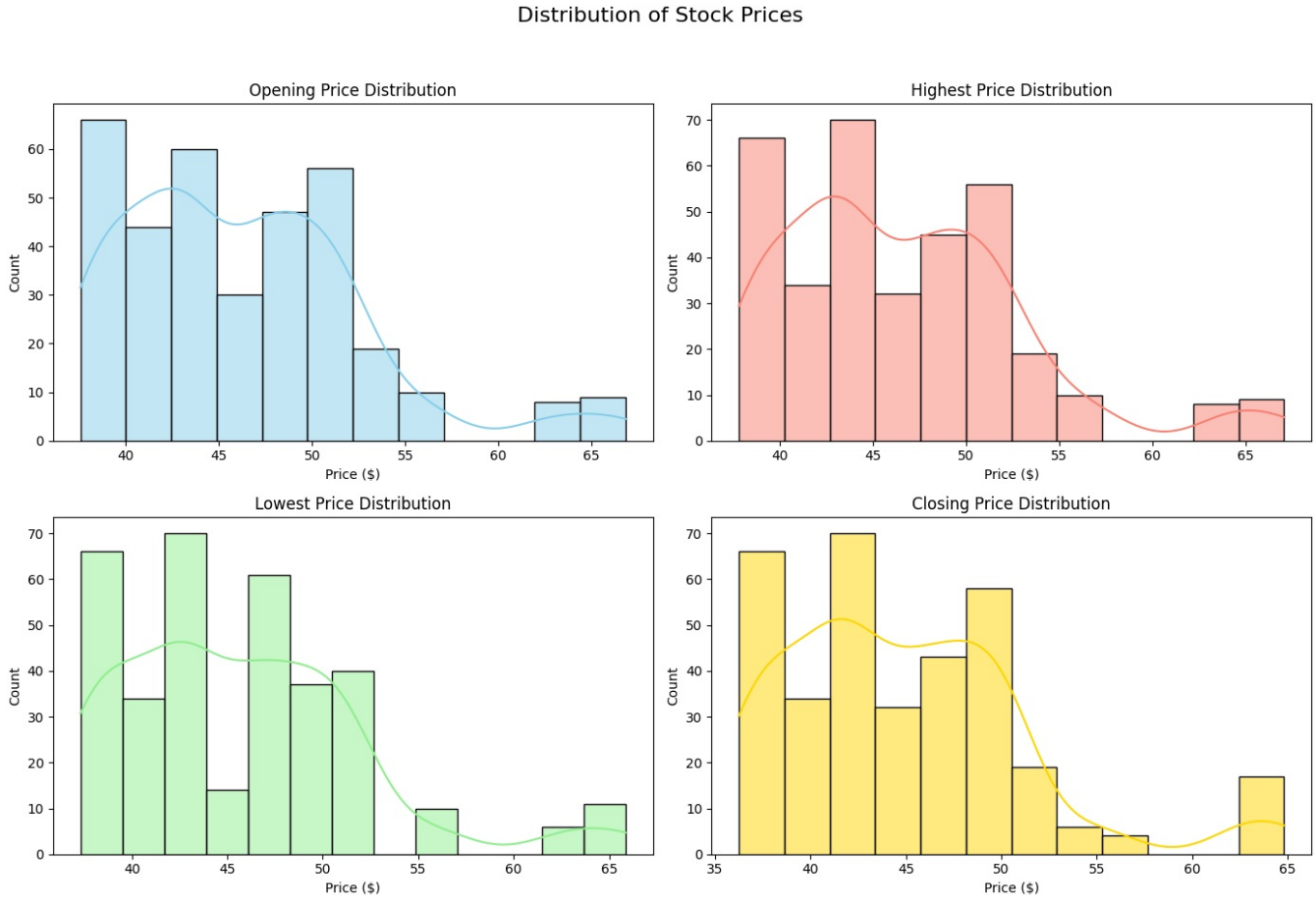
```
plt.title("Distribution of Data by Month")
plt.xlabel("Month")
plt.ylabel("Frequency")
plt.show()
```



- As seen from the above plot, there are a significantly large amount of news released in the month of january compared to the next 3 months on record. It seems like an unusal amount coming from the first month of the year an then drastically decreases in teh following months need sto be looked upon.

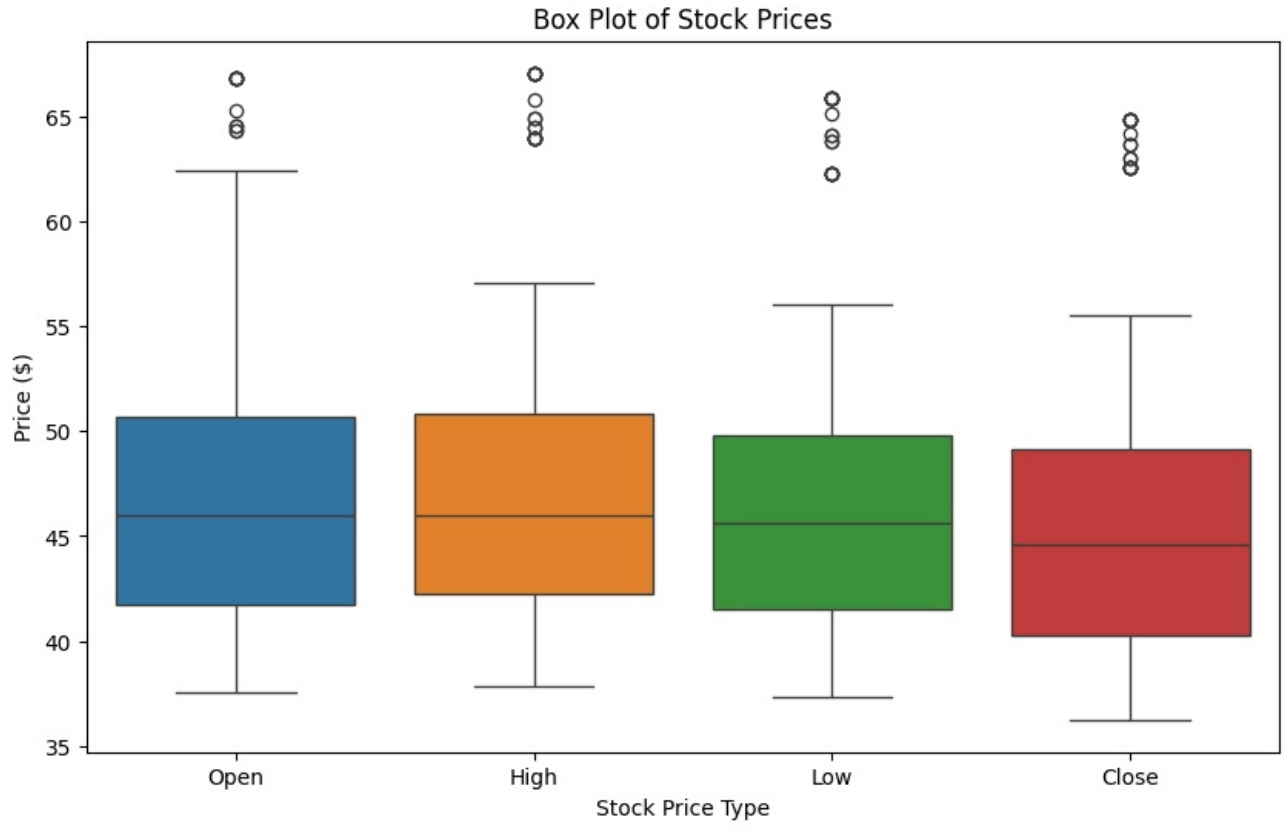
Distribution of Stock Prices

In [ ]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle("Distribution of Stock Prices", fontsize=16)

# Plotting each stock price variable
sns.histplot(df['Open'], kde=True, ax=axes[0, 0], color='skyblue')
axes[0, 0].set_title("Opening Price Distribution")
axes[0, 0].set_xlabel("Price ($)")

sns.histplot(df['High'], kde=True, ax=axes[0, 1], color='salmon')
axes[0, 1].set_title("Highest Price Distribution")
axes[0, 1].set_xlabel("Price ($)")

sns.histplot(df['Low'], kde=True, ax=axes[1, 0], color='lightgreen')
axes[1, 0].set_title("Lowest Price Distribution")
axes[1, 0].set_xlabel("Price ($)")

sns.histplot(df['Close'], kde=True, ax=axes[1, 1], color='gold')
axes[1, 1].set_title("Closing Price Distribution")
axes[1, 1].set_xlabel("Price ($)")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout to fit title
plt.show()
```



- From the above distribution plots of the stock prices they all look somewhat simlilar in a first look viewpoint. The kde lines from all the plots have the same pattern as well, indicating that

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[['Open', 'High', 'Low', 'Close']])
plt.title("Box Plot of Stock Prices")
plt.xlabel("Stock Price Type")
plt.ylabel("Price ($)")
plt.show()
```

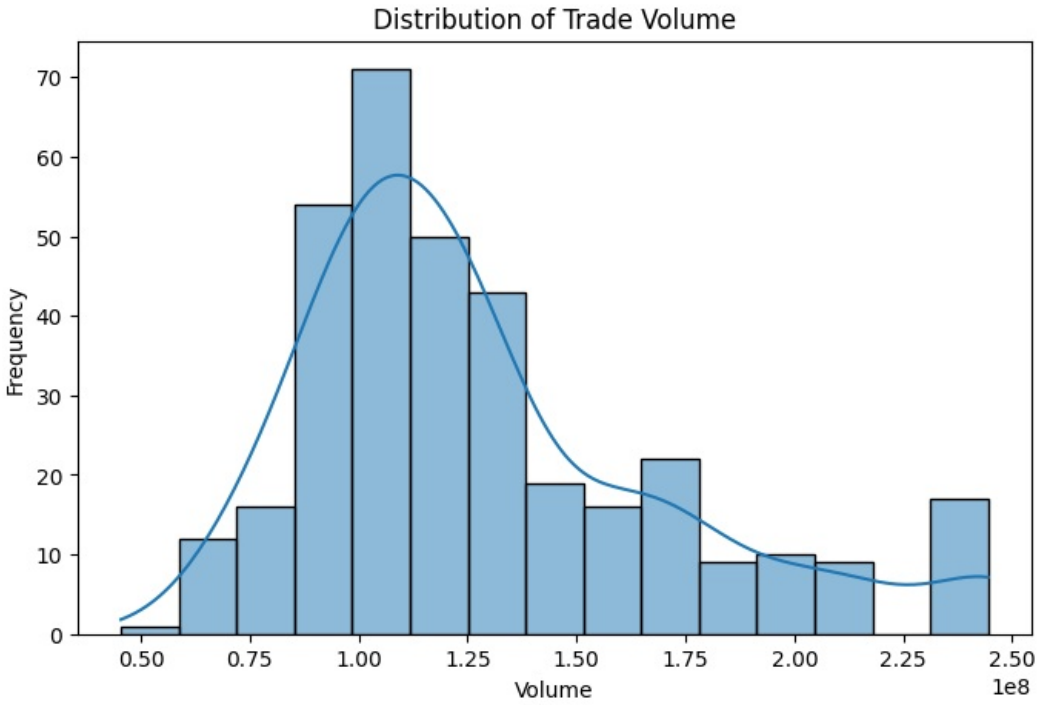


- From the above distribution plots as well as the box plots of the stock prices they all look somewhat simlilar in a first look viewpoint.

The kde lines from all the plots have the same pattern as well, indicating that there isnt much of a difference. In the box plots you can tell that the stock prices is at its lowest at the end of the day compared to when it is first opened in the morning.

Distribution of Trade Volume

In [ ]:
```
plot_distribution('Volume', 'Trade Volume', 'Volume')
```

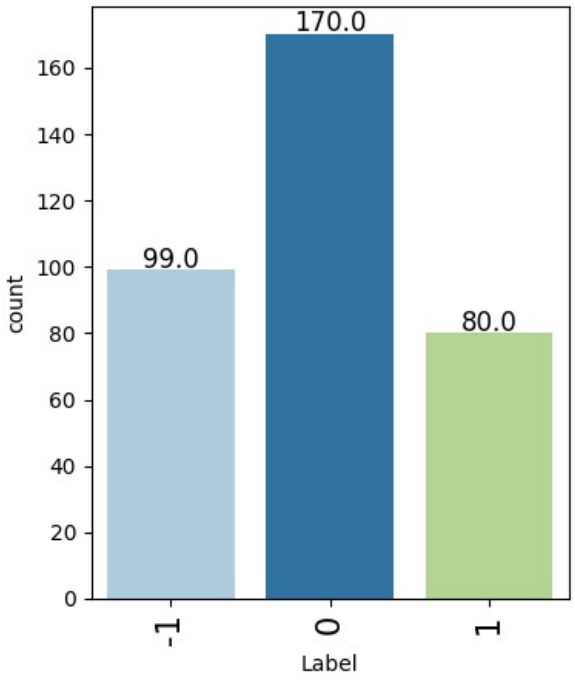


- As seen from the above distribution plot, the numer of shares traded during the day is most often around 0.9 to 1.25 dollars per day.

Distribution of Sentiment Polarity

In [ ]:
```
labeled_barplot(data, 'Label')
```

```
<ipython-input-18-1c4e2a28bdb7>:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable
to `hue` and set `legend=False` for the same effect.

  ax = sns.countplot(
```

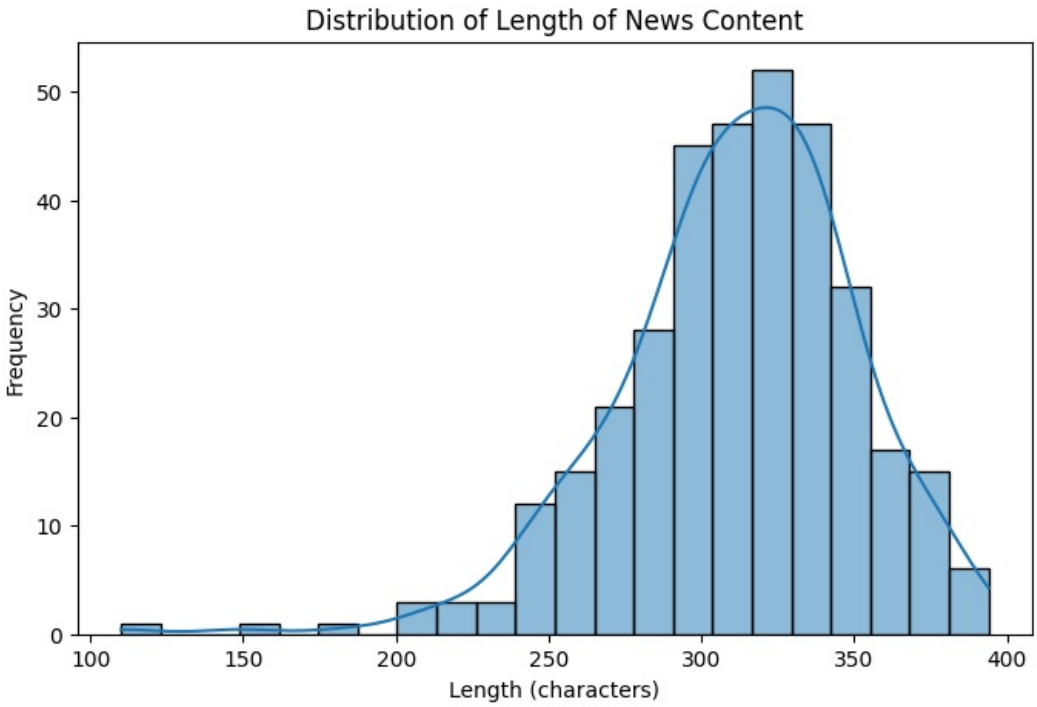


- There are a majority of neutral news content compared to them being either positive or negative.

Distribution of News Length Content

In [ ]:
```
# Calculate the length of each news article in characters
df['News_Length'] = df['News'].apply(len)
```

In [ ]:
```
# Plot the distribution of news content length
plot_distribution('News_Length', 'Length of News Content', 'Length (characters)')
```



In [ ]:
```
# Basic Statistics for News Length
print("\nBasic Statistics for News Content Length:")
print(df['News_Length'].describe())
```

```
Basic Statistics for News Content Length:
count    349.000000
mean     311.237822
std       39.079467
min      110.000000
25%      290.000000
50%      315.000000
75%      336.000000
max      394.000000
Name: News_Length, dtype: float64
```

- From the distrubition plot above, the average length of news content is around 311 characters, where the maximum is aroud 400 and the lowest is 110 charachers.

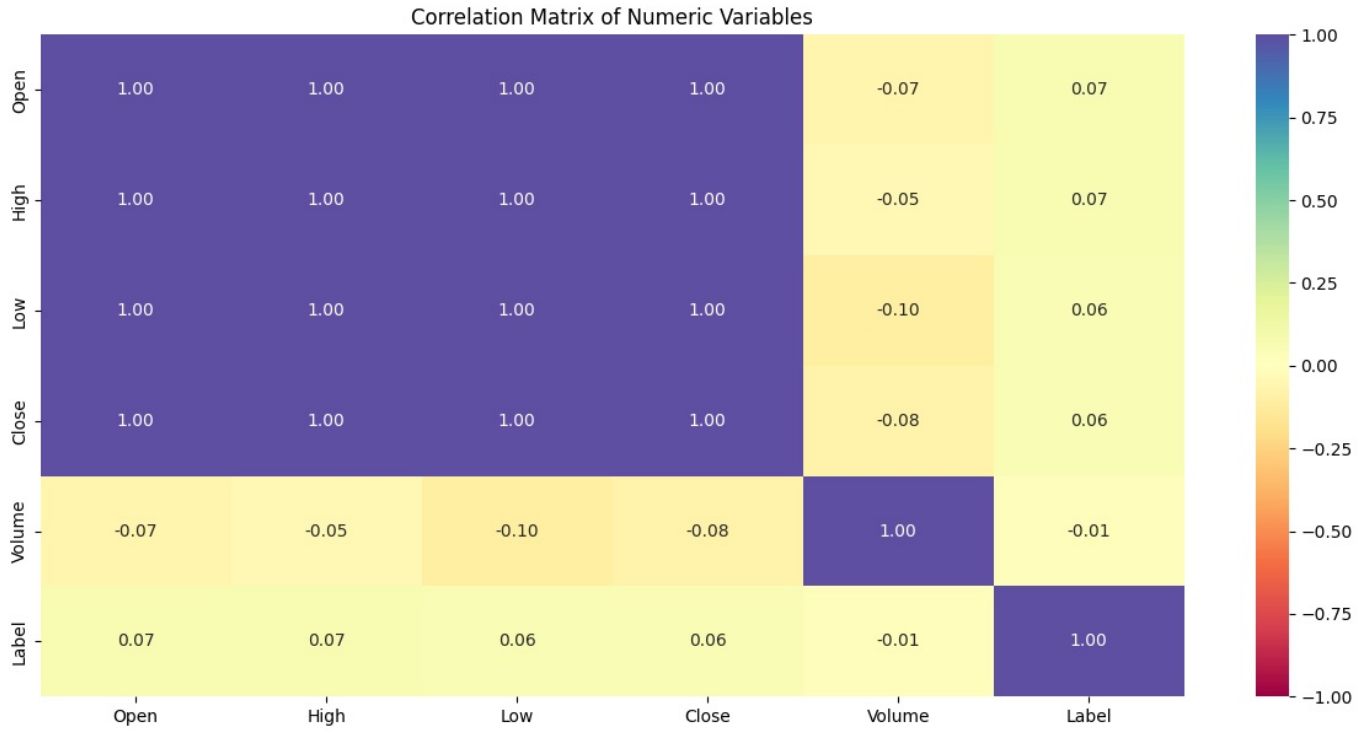
Bivariate Analysis

- Correlation
- Sentiment Polarity vs Price
- Date vs Price

Note: The above points are listed to provide guidance on how to approach bivariate analysis. Analysis has to be done beyond the above listed points to get maximum scores.

Correlation Matrix

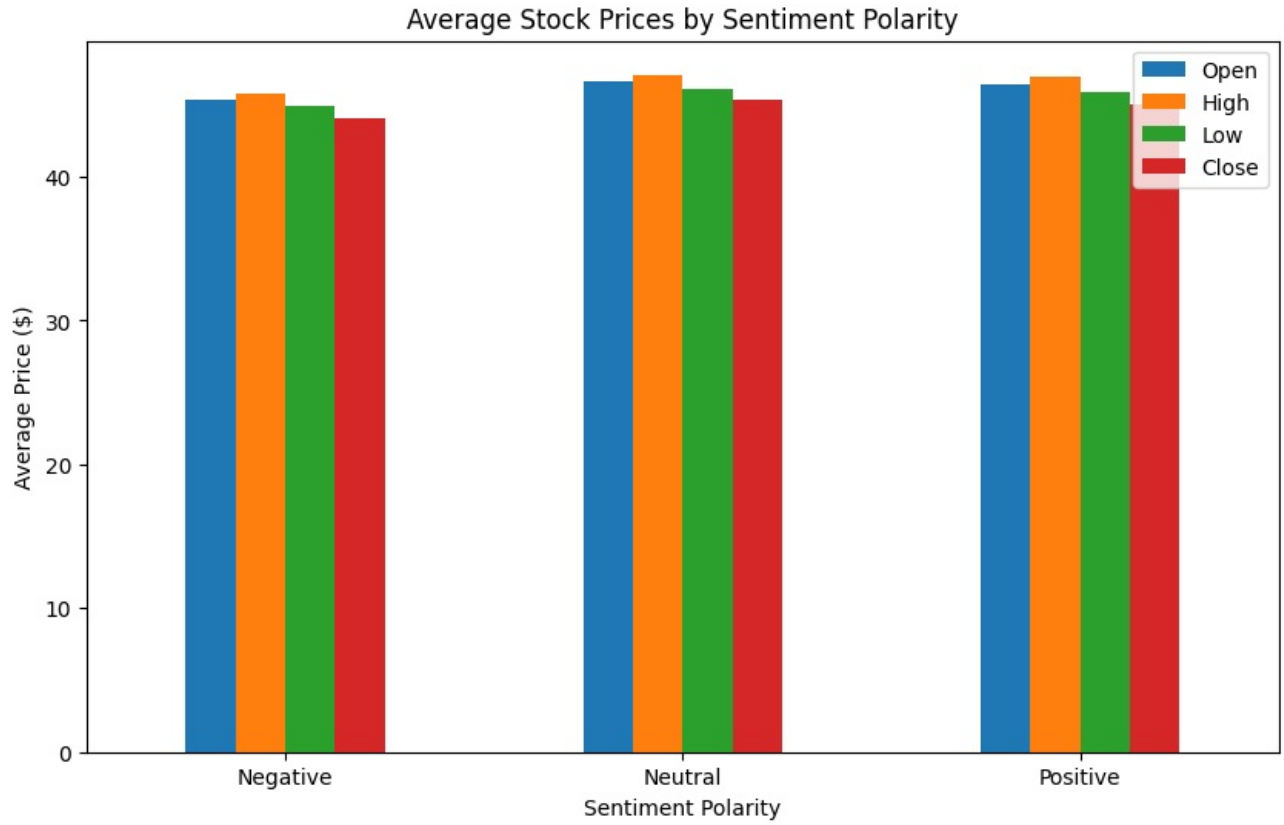
In [ ]:
```
numerical_df = data.select_dtypes(include=['number'])
plt.figure(figsize=(15, 7))
sns.heatmap(
numerical_df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.title("Correlation Matrix of Numeric Variables")
plt.show()
```



- From the above correlation matrix, the different stock prices are obbiously all highly correlate diwth each other. Whilst all of the other numeric variabnles do not seem to be correlated at all. This shows that sentiment doesnt have a direct influence on price movement.

Sentiment Polarity vs. Stock Prices

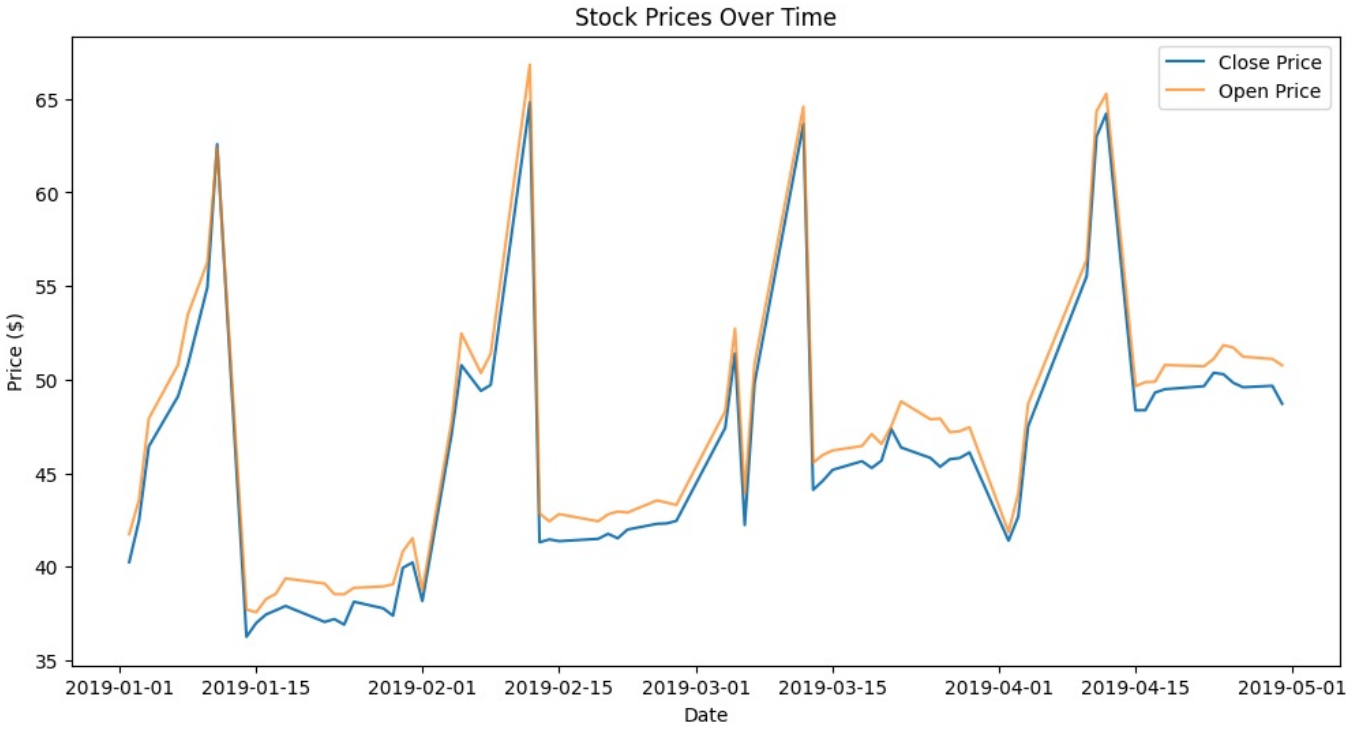
In [ ]:
```
# Group by sentiment label and calculate mean prices
sentiment_price_means = df.groupby('Label')[['Open', 'High', 'Low', 'Close']].mean()

# Plot average prices by sentiment polarity
sentiment_price_means.plot(kind='bar', figsize=(10, 6))
plt.title("Average Stock Prices by Sentiment Polarity")
plt.xlabel("Sentiment Polarity")
plt.ylabel("Average Price ($)")
plt.xticks(ticks=[0, 1, 2], labels=["Negative", "Neutral", "Positive"], rotation=0)
plt.show()
```



As we can see, the sentiment polairty does not have an impact on the stock prices throughout the day. Therefore, there is not impact o npositve or negative news o nthe stock prices.

Date vs Stock Prices

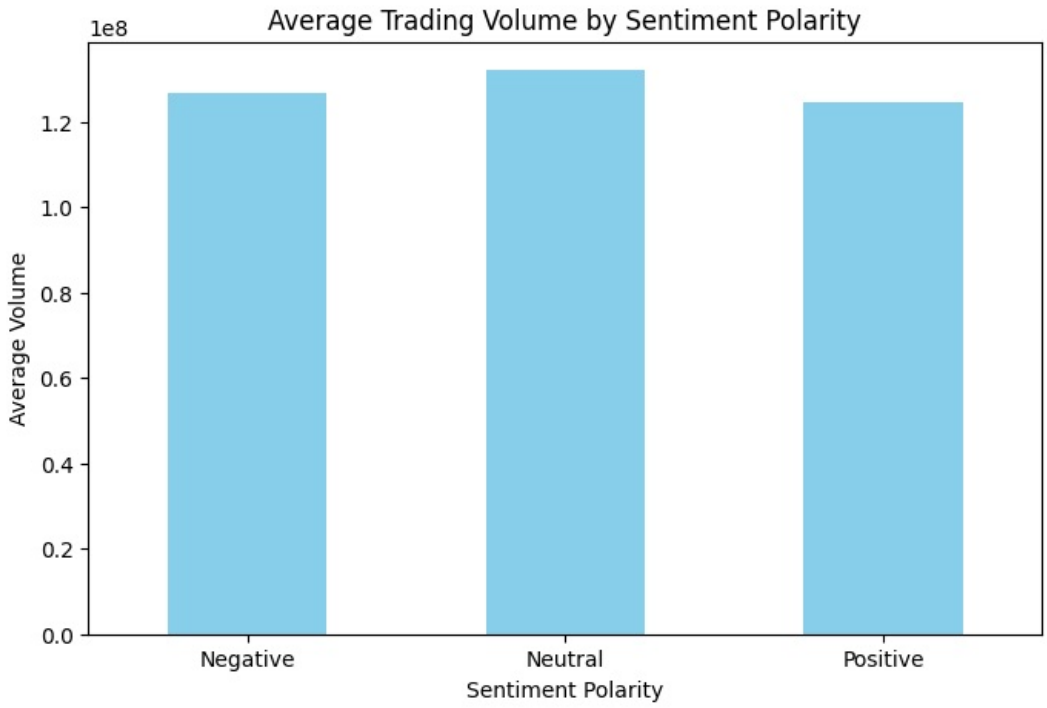
In [ ]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Close Price')
plt.plot(df['Date'], df['Open'], label='Open Price', alpha=0.7)
plt.title("Stock Prices Over Time")
plt.xlabel("Date")
plt.ylabel("Price ($)")
plt.legend()
plt.show()
```



The stock prices trends over time from the beginning of the day compared to the end isnt really much of a difference. Howev er, the open stock price does seem to have a bit of an increase throughout the months compared to the closing stock prices.

Sentiment Polarity vs Volume

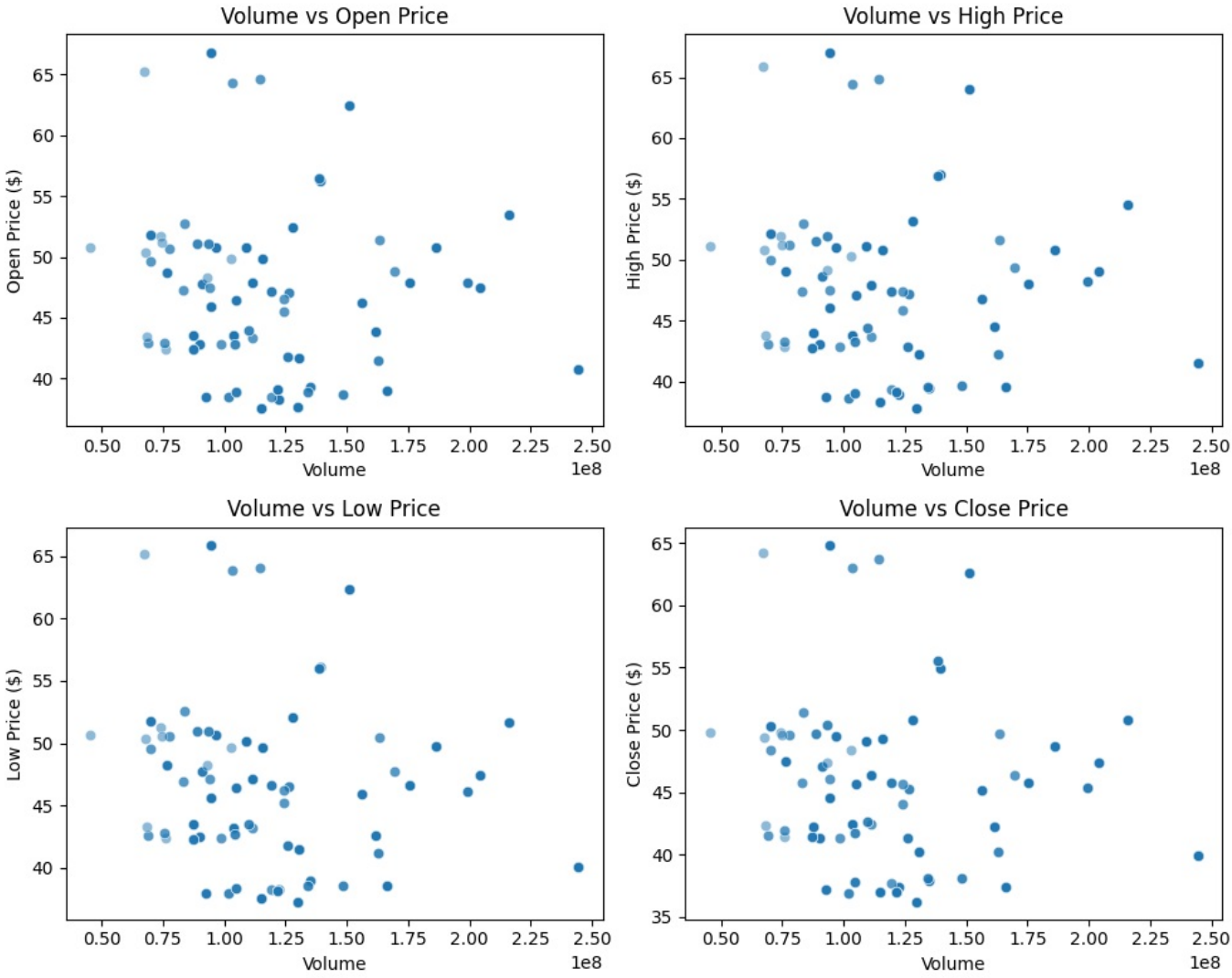
In [ ]:
```
# Calculate average volume by sentiment
sentiment_volume_means = df.groupby('Label')['Volume'].mean()

# Plot average volume by sentiment polarity
plt.figure(figsize=(8, 5))
sentiment_volume_means.plot(kind='bar', color='skyblue')
plt.title("Average Trading Volume by Sentiment Polarity")
plt.xlabel("Sentiment Polarity")
plt.ylabel("Average Volume")
plt.xticks(ticks=[0, 1, 2], labels=["Negative", "Neutral", "Positive"], rotation=0)
plt.show()
```



There seems to be a little more neutral sentiment polarity compared to the positive and negative.

Price vs. Volume

In [ ]:
```
plt.figure(figsize=(10, 8))

# Scatter plots of Volume vs each price type
for i, price in enumerate(['Open', 'High', 'Low', 'Close'], 1):
    plt.subplot(2, 2, i)
    sns.scatterplot(data=df, x='Volume', y=price, alpha=0.5)
    plt.title(f"Volume vs {price} Price")
    plt.xlabel("Volume")
    plt.ylabel(f"{price} Price ($)")

plt.tight_layout()
plt.show()
```



- From looking at the heindsight of all four graphs, they all seem to have the same style pattern, so there is not direct impoact of traditing activity on price, which doesn't inform inventesmtne stragegies at all.

Data Preprocessing

Split Target Variable and Predictors

In [ ]:
```
# Define target and predictor variables
X = df[['Date', 'News', 'Open', 'High', 'Low', 'Close', 'Volume']]  # Predictors
y = df['Label']  # Target
```

Encode Date and Text Data

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Extract date components (year, month, day)
X['Year'] = pd.to_datetime(X['Date']).dt.year
X['Month'] = pd.to_datetime(X['Date']).dt.month
X['Day'] = pd.to_datetime(X['Date']).dt.day
X = X.drop(columns=['Date'])  # Drop the original Date column

# Convert text data to numeric features using TF-IDF
vectorizer = TfidfVectorizer(max_features=500)  # Adjust max_features as needed
news_tfidf = vectorizer.fit_transform(X['News']).toarray()
news_tfidf = pd.DataFrame(news_tfidf, columns=vectorizer.get_feature_names_out())
X = X.drop(columns=['News']).reset_index(drop=True)
X = pd.concat([X, news_tfidf], axis=1)

# Standardize numerical features
scaler = StandardScaler()
X[['Open', 'High', 'Low', 'Close', 'Volume', 'Year', 'Month', 'Day']] = scaler.fit_transform(
    X[['Open', 'High', 'Low', 'Close', 'Volume', 'Year', 'Month', 'Day']]
)
```

```
<ipython-input-35-2cb8e18c0fd3>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  X['Year'] = pd.to_datetime(X['Date']).dt.year
<ipython-input-35-2cb8e18c0fd3>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  X['Month'] = pd.to_datetime(X['Date']).dt.month
```

Split the Data into Train, Validation, and Test Sets

In [ ]:
```
# Split data into train (70%), validation (15%), and test (15%) sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp

# Check shapes to verify splits
print(f"Training Set: {X_train.shape}, {y_train.shape}")
print(f"Validation Set: {X_val.shape}, {y_val.shape}")
print(f"Test Set: {X_test.shape}, {y_test.shape}")
```

```
Training Set: (244, 508), (244,)
Validation Set: (52, 508), (52,)
Test Set: (53, 508), (53,)
```

Data Preperation

In [ ]:
```
import re
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Text Preprocessing function
def preprocess_text(text):
    # Lowercase the text
    text = text.lower()
    # Remove punctuation and digits
    text = re.sub(r'[^a-z\s]', '', text)
    # Remove stop words
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

# Apply preprocessing to the News column
df['cleaned_news'] = df['News'].apply(preprocess_text)
```

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

Word Embeddings

Word2Vec

In [ ]:
```
from gensim.models import Word2Vec

# Tokenize sentences for Word2Vec
sentences = [sentence.split() for sentence in df['cleaned_news']]

# Train Word2Vec model
vec_size=100
word2vec_model = Word2Vec(sentences, vector_size=vec_size, window=5, min_count=1, workers=4)

# Create an embedding vector for each document by averaging word vectors
def get_word2vec_vector(sentence):
    words = sentence.split()
    word_vectors = [word2vec_model.wv[word] for word in words if word in word2vec_model.wv]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(100)

df['word2vec_embedding'] = df['cleaned_news'].apply(get_word2vec_vector)
X_word2vec = np.stack(df['word2vec_embedding'].values)  # Convert to a format suitable for ML models
```

In [ ]:
```
# Checking the size of the vocabulary
print("Length of the vocabulary is", len(list(word2vec_model.wv.key_to_index)))
```

```
Length of the vocabulary is 3359
```

Word Embedding Examples

In [ ]:
```
data.head()
```

Out[ ]:

	Date	News	Open	High	Low	Close	Volume	Label
0	2019-01-02	The tech sector experienced a significant decline in the aftermarket following Apple's Q1 revenue warning. Notable suppliers, including Skyworks, Broadcom, Lumentum, Qorvo, and TSMC, saw their stocks drop in response to Apple's downward revision of its revenue expectations for the quarter, previously announced in January.	41.740002	42.244999	41.482498	40.246914	130672400	-1
1	2019-01-02	Apple lowered its fiscal Q1 revenue guidance to 84billionfromearlierestimatesof89-$93 billion due to weaker than expected iPhone sales. The announcement caused a significant drop in Apple's stock price and negatively impacted related suppliers, leading to broader market declines for tech indices such as Nasdaq 10	41.740002	42.244999	41.482498	40.246914	130672400	-1
2	2019-01-02	Apple cut its fiscal first quarter revenue forecast from 89 − 93 billion to $84 billion due to weaker demand in China and fewer iPhone upgrades. CEO Tim Cook also mentioned constrained sales of Airpods and Macbooks. Apple's shares fell 8.5% in post market trading, while Asian suppliers like Hon	41.740002	42.244999	41.482498	40.246914	130672400	-1
3	2019-01-02	This news article reports that yields on long-dated U.S. Treasury securities hit their lowest levels in nearly a year on January 2, 2019, due to concerns about the health of the global economy following weak economic data from China and Europe, as well as the partial U.S. government shutdown. Apple	41.740002	42.244999	41.482498	40.246914	130672400	-1
4	2019-01-02	Apple's revenue warning led to a decline in USD JPY pair and a gain in Japanese yen, as investors sought safety in the highly liquid currency. Apple's underperformance in Q1, with forecasted revenue of 84billioncomparedtoanalystexpectationsof91.5 billion, triggered risk aversion mood in markets	41.740002	42.244999	41.482498	40.246914	130672400	-1

In [ ]:
```
# Checking the word embedding of a random word
word = "revenue"
word2vec_model.wv[word]
```

Out[ ]:
```
array([-1.3567323e-02,  1.5479675e-02,  3.7412217e-04, -3.1825106e-03,
        7.0075309e-03, -1.5881339e-02,  5.5463458e-03,  2.0057496e-02,
        1.5590105e-03, -9.6708573e-03,  5.6049419e-03, -5.5291369e-03,
        2.0533483e-03, -3.6234781e-03,  1.0367125e-02, -7.3855729e-03,
        9.2958659e-03, -1.2368411e-02, -8.6961258e-03, -3.5512331e-03,
        6.6450131e-03, -8.7954995e-04,  1.4640403e-02,  7.7368617e-03,
       -1.2450625e-02,  2.3732600e-03,  1.2544713e-03,  9.6937743e-05,
       -3.2760734e-03,  7.6817838e-04,  7.1769161e-03, -1.9415169e-03,
       -6.2799817e-03, -6.8799895e-03,  3.3592733e-03,  1.5271989e-02,
        8.8726822e-03, -1.1854073e-02, -1.4681441e-02, -2.7667708e-03,
        2.0918115e-03, -2.2292175e-04,  4.1931137e-03, -1.0981837e-03,
        1.3165653e-02, -4.5877462e-04, -2.2926910e-03, -3.2174408e-03,
        1.3553849e-03,  2.4898280e-03,  8.6550443e-03, -6.6401195e-03,
       -1.1153225e-02, -1.0853863e-02, -1.0932520e-02,  4.0720366e-03,
        5.7805525e-03,  9.5474608e-03, -8.0324186e-04, -3.5944376e-03,
        7.1179541e-04,  8.2375053e-03,  6.0469443e-03, -9.1390517e-05,
       -2.6626964e-03, -3.6084172e-04, -8.2875940e-06,  1.4433322e-02,
       -1.0219142e-02, -2.0107783e-03,  1.5962882e-03, -6.2632171e-04,
        6.9665131e-03,  7.3178401e-03,  4.8067719e-03, -1.1509056e-03,
       -5.7923789e-03,  5.0998935e-03, -9.9333879e-03,  3.2678640e-03,
       -1.1105243e-02,  5.9489482e-03, -1.1448219e-03,  1.2175485e-02,
       -7.9245801e-04, -2.4473774e-03,  9.5066382e-03,  1.1995359e-02,
        5.7666595e-03, -8.2731042e-03, -6.4389646e-04, -1.0950924e-04,
        1.7024339e-03, -9.8034844e-04,  2.1233397e-02,  5.7619242e-03,
        8.9295041e-03, -1.3751366e-02, -1.1198726e-03, -6.0979314e-03],
      dtype=float32)
```

In [ ]:
```
# Checking the word embedding of a random word
word = "shares"
word2vec_model.wv[word]
```

Out[ ]:
```
array([-0.00349056,  0.00758711,  0.0018203 ,  0.00632844, -0.00273747,
       -0.00946357,  0.00920336,  0.013772  ,  0.00045815,  0.00223969,
       -0.00425576, -0.0170599 ,  0.00136288, -0.00678243, -0.00146924,
       -0.0135353 , -0.00052242, -0.01398958,  0.00655383, -0.01400072,
        0.01359583, -0.00804748, -0.00097084,  0.00011337, -0.00514801,
        0.00161225, -0.00760261, -0.00389167, -0.01061864, -0.0090478 ,
        0.01196692,  0.00145034,  0.00870445, -0.01189308,  0.00247438,
       -0.00557428,  0.00686086,  0.00042532, -0.0101373 , -0.00235623,
       -0.00572588, -0.01335752, -0.00025292,  0.00376908, -0.00310125,
       -0.01218407,  0.0051147 ,  0.00114963,  0.0044902 ,  0.00722744,
       -0.00062886, -0.00456907,  0.0006169 ,  0.01007526, -0.00989166,
        0.0059004 ,  0.00921902, -0.00730347, -0.01495097,  0.00951765,
        0.00790499, -0.00166284,  0.0041059 ,  0.00150756, -0.01144378,
        0.00357994, -0.00365511,  0.01119503, -0.00495769, -0.00177517,
       -0.00364378, -0.00013177,  0.01360729, -0.00380371,  0.01404281,
        0.00519548, -0.00459083,  0.00647183, -0.0009541 , -0.00256483,
       -0.00975901, -0.00309928, -0.00987804,  0.011494  , -0.00935099,
       -0.00573152,  0.0037561 ,  0.00637941,  0.01317831, -0.00177115,
        0.00606427,  0.0037729 ,  0.00729681,  0.00330361,  0.01920618,
        0.00911094, -0.00435299, -0.00723207, -0.00710502, -0.00601472],
      dtype=float32)
```

In [ ]:
```
# Checking the word embedding of a random word
word = "market"
word2vec_model.wv[word]
```

Out[ ]:
```
array([ 0.00279248,  0.00485157,  0.00892317, -0.01117725, -0.00547553,
       -0.01750377, -0.00160275,  0.01815307, -0.00822785, -0.01083929,
        0.00460308, -0.01425985,  0.00672036, -0.00225442,  0.00531114,
        0.00012011,  0.00287691, -0.00987939,  0.00170686, -0.01905428,
        0.00145238, -0.00136304,  0.01195286, -0.00618131, -0.00911995,
        0.00431102, -0.00459514, -0.00682597,  0.00125894,  0.00509301,
        0.00298845,  0.00222443,  0.01068791,  0.002728  ,  0.00465905,
        0.01068401,  0.00201181, -0.00809334, -0.01075213, -0.01320214,
       -0.00188352, -0.00599361, -0.00839845,  0.00826696, -0.00095172,
       -0.01027433, -0.00837051, -0.00247219, -0.00369559,  0.00605747,
       -0.00180566, -0.00846017, -0.00873586,  0.00171796, -0.00129876,
        0.00418358, -0.00211536, -0.00230035,  0.00013181,  0.00487573,
       -0.00745778, -0.00042389,  0.00037048,  0.00272604, -0.01659315,
        0.01023526, -0.00790875, -0.00206119, -0.00432942,  0.00443214,
       -0.00031081,  0.01247802,  0.00453757,  0.00602043,  0.00921836,
        0.0127257 ,  0.00594109, -0.01178737, -0.00738762,  0.0080539 ,
        0.00057771,  0.0082863 ,  0.0012436 ,  0.01724631, -0.01030126,
        0.00502135, -0.00513034,  0.00958067,  0.01404592,  0.006077  ,
        0.01492115,  0.00278345,  0.00110689,  0.0077947 ,  0.00555827,
        0.00553585,  0.01262514, -0.01232044,  0.00583537, -0.00866714],
      dtype=float32)
```

In [ ]:
```
# Retrieving the words present in the Word2Vec model's vocabulary
words = list(word2vec_model.wv.key_to_index.keys())

# Retrieving word vectors for all the words present in the model's vocabulary
wvs = word2vec_model.wv[words].tolist()

# Creating a dictionary of words and their corresponding vectors
word_vector_dict = dict(zip(words, wvs))
```

In [ ]:
```
def average_vectorizer_Word2Vec(doc):
    # Initializing a feature vector for the sentence
    feature_vector = np.zeros((vec_size,), dtype="float64")

    # Creating a list of words in the sentence that are present in the model vocabulary
    words_in_vocab = [word for word in doc.split() if word in words]

    # adding the vector representations of the words
    for word in words_in_vocab:
        feature_vector += np.array(word_vector_dict[word])

    # Dividing by the number of words to get the average vector
    if len(words_in_vocab) != 0:
        feature_vector /= len(words_in_vocab)

    return feature_vector
```

In [ ]:
```
# creating a dataframe of the vectorized documents
df_Word2Vec = pd.DataFrame(df['cleaned_news'].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+
df_Word2Vec
```

Out[ ]:

	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Feature 9	...	Feature 90	Fea
0	-0.003404	0.002639	-0.000865	0.000104	0.001696	-0.005662	0.001509	0.006409	-0.003025	-0.001595	...	0.002018	0.00
1	-0.002810	0.002856	0.000439	-0.000058	0.000860	-0.008533	0.002267	0.008899	-0.004849	-0.002562	...	0.004477	0.00
2	-0.003523	0.004078	-0.000581	-0.000385	0.000671	-0.006049	0.002136	0.007802	-0.002250	-0.000472	...	0.005572	0.00
3	-0.002658	0.004341	-0.000326	-0.000601	0.001287	-0.005816	0.003104	0.008719	-0.005138	-0.000995	...	0.004614	0.00
4	-0.003938	0.001985	-0.001536	-0.001042	-0.000127	-0.006305	-0.000473	0.004072	-0.002022	-0.001379	...	0.001242	-0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	
344	-0.000989	-0.000731	0.001142	-0.000047	0.000142	-0.001356	0.002001	-0.000426	-0.000764	0.000595	...	0.001387	0.00
345	-0.001109	0.002635	0.000582	-0.000102	0.001639	-0.004041	0.001853	0.005711	-0.001552	-0.000243	...	0.003192	0.00
346	0.001222	0.003312	0.000524	0.000374	0.003370	-0.004967	0.000126	0.004468	-0.001904	-0.000869	...	0.002656	0.00
347	-0.000629	0.003336	-0.000323	-0.000706	-0.000403	-0.004697	0.001711	0.006063	-0.001259	-0.000985	...	0.001868	-0.00
348	-0.002558	0.004575	0.002098	-0.000779	-0.000906	-0.003651	0.002677	0.006340	-0.001462	-0.001915	...	0.003175	0.00

349 rows × 100 columns

GloVe

In [ ]:
```
# Load GloVe embeddings
def load_glove_embeddings(filepath):
    embeddings = {}
    with open(filepath, 'r', encoding='utf-8') as file:
        for line in file:
            values = line.split()
            word = values[0]
            vector = np.array(values[1:], dtype='float32')
            embeddings[word] = vector
    return embeddings

glove_embeddings = load_glove_embeddings('/content/drive/MyDrive/Colab Notebooks/glove.6B.100d.txt.word2vec')

# Create an embedding vector for each document by averaging GloVe word vectors
def get_glove_vector(sentence):
    words = sentence.split()
    word_vectors = [glove_embeddings[word] for word in words if word in glove_embeddings]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(100)

df['glove_embedding'] = df['cleaned_news'].apply(get_glove_vector)
X_glove = np.stack(df['glove_embedding'].values)
```

In [ ]:
```
# Checking the size of the vocabulary
print("Length of the vocabulary is", len(glove_embeddings))
```

```
Length of the vocabulary is 400001
```

In [ ]:
```
# Checking the word embedding of a random word
word = "market"
glove_embeddings[word]
```

Out[ ]:
```
array([ 0.39093  ,  0.23755  ,  0.44855  ,  0.11237  , -0.25996  ,
       -1.2248   , -0.44237  , -0.53491  ,  0.37142  , -0.61981  ,
       -0.27387  , -0.032213 ,  0.082629 , -0.52986  ,  0.13012  ,
        0.21703  , -0.45026  , -0.0048895,  0.34887  , -0.26069  ,
        0.56598  , -0.36219  ,  0.41926  ,  0.23441  , -0.29407  ,
       -0.27044  ,  0.29339  , -0.73905  , -0.75965  ,  0.64661  ,
       -0.038757 ,  0.38495  , -0.32314  ,  0.040322 ,  0.24036  ,
        0.35167  ,  0.47404  ,  0.014959 ,  0.12105  , -1.0398   ,
        0.27639  , -1.3785   , -0.22851  , -0.098074 ,  0.1495   ,
       -0.2815   ,  0.31682  , -0.10208  , -0.08586  , -1.5114   ,
       -0.48255  ,  0.15131  ,  0.0080133,  0.74594  , -0.20163  ,
       -2.5268   , -0.82083  ,  0.1143   ,  2.4665   ,  0.19841  ,
        0.1146   ,  0.10083  , -0.60936  ,  0.76722  ,  0.025978 ,
       -0.036936 ,  0.46744  , -0.77073  ,  0.83992  , -0.032931 ,
       -0.13127  , -0.097367 , -0.42634  , -0.49478  , -0.40796  ,
       -0.67504  , -0.28535  ,  0.12474  , -1.145    , -0.43059  ,
        1.172    ,  0.40749  , -0.83089  ,  0.41675  , -0.83018  ,
       -0.88716  , -0.59827  , -0.56652  , -0.2275   , -0.42398  ,
        0.63385  ,  0.62035  , -0.13429  , -0.49012  , -0.78362  ,
        0.85838  ,  0.60102  , -0.40596  ,  0.77826  ,  1.105    ],
      dtype=float32)
```

In [ ]:
```
# Checking the word embedding of a random word
word = "revenue"
```

```
glove_embeddings[word]
```

Out[ ]:
```
array([ 0.17132  , -0.042936 ,  0.48587  , -0.49912  ,  0.83222  ,
       -0.79533  , -0.39641  ,  0.0059214,  0.065537 , -0.35212  ,
        0.40528  , -0.12115  , -0.15325  , -0.46874  ,  0.2268   ,
       -0.80513  ,  0.065305 ,  0.25888  ,  0.4961   ,  1.0646   ,
        0.14317  ,  0.11955  , -0.35765  ,  1.0962   , -0.57384  ,
       -0.62014  ,  0.54423  , -0.20712  , -0.67822  , -0.82173  ,
        0.6037   ,  0.26209  , -0.18882  , -0.90291  , -0.181    ,
        0.43017  , -0.30814  ,  0.15839  , -0.0065962, -0.20375  ,
        0.52302  , -0.75001  , -0.35737  ,  0.38933  ,  0.2421   ,
       -0.1404   , -0.19481  , -0.97056  , -0.169    , -1.1695   ,
        0.089281 , -0.098044 ,  0.5469   ,  0.888    ,  0.42843  ,
       -1.8364   ,  0.083037 , -0.59659  ,  2.2736   ,  0.21042  ,
        0.21198  , -1.0437   , -0.66913  ,  0.5043   , -0.44588  ,
       -0.025209 , -0.1372   , -0.15207  ,  1.482    , -0.17329  ,
        0.69726  , -0.10271  ,  0.37622  , -0.016729 ,  0.18066  ,
       -0.3527   ,  0.070993 ,  0.019209 , -1.0751   ,  0.62373  ,
        0.42793  ,  0.28526  , -0.30414  , -0.24486  , -1.0767   ,
       -0.78603  , -0.034846 , -1.1542   , -0.12238  ,  0.022361 ,
       -0.018078 ,  0.38177  , -0.68654  , -1.1848   , -0.46846  ,
       -0.27115  ,  0.7279   ,  0.36525  ,  1.0137   , -0.16458  ],
      dtype=float32)
```

In [ ]:
```
# Checking the word embedding of a random word
word = "shares"
glove_embeddings[word]
```

Out[ ]:
```
array([ 1.3286   ,  0.69354 ,  0.29419 ,  0.66886 ,  0.33194 , -0.8209  ,
        0.017445, -0.37058 , -0.45322 , -0.68676 ,  0.44886 ,  0.35642 ,
       -0.1166  , -0.47337 , -0.34725 , -0.8647  , -0.43301 , -0.092064,
        0.80659 ,  0.49978 ,  0.92839 , -0.48723 ,  0.18957 ,  1.2957  ,
        0.027177, -1.0023  ,  0.48211 ,  0.27344 , -0.6814  , -0.33231 ,
        0.43804 ,  0.85628 , -0.86716 , -0.031107, -0.082545,  0.53319 ,
        1.1965  , -0.057855, -0.068861, -0.35747 ,  0.67171 , -1.2513  ,
       -0.070206, -0.55692 , -0.41302 ,  0.9754  , -0.72807 ,  0.55693 ,
        0.27977 , -1.8509  , -0.30387 ,  0.25268 ,  0.95404 ,  0.47831 ,
       -0.76744 , -1.5847  , -0.35392 , -0.41192 ,  2.387   ,  0.075613,
        0.79698 , -0.26653 , -0.23899 ,  0.33455 , -0.87235 ,  0.14746 ,
        0.71212 ,  0.73689 ,  0.83065 ,  0.36152 , -1.1332  ,  0.14572 ,
       -0.38399 , -0.2714  , -0.55319 , -0.09626 , -0.50378 ,  0.29951 ,
       -1.1838  ,  0.33485 ,  0.4057  , -0.096851, -0.39435 ,  0.38375 ,
       -0.63077 , -1.0174  , -0.05915 , -0.55108 ,  0.63042 ,  0.12987 ,
       -0.49347 ,  0.12493 , -0.16105 , -0.22544 , -1.3219  ,  1.0872  ,
        0.43629 , -0.42036 ,  0.32478 ,  0.69153 ], dtype=float32)
```

In [ ]:
```
# creating a dataframe of the vectorized documents
df_Glove = pd.DataFrame(df['cleaned_news'].apply(get_glove_vector).tolist(), columns=['Feature '+str(i) for i in
df_Glove
```

Out[ ]:

	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8	Feature 9	...	Feature 90	Fea
0	-0.000033	0.120790	0.145954	-0.018663	-0.003805	-0.606782	-0.057167	-0.046783	-0.021009	-0.117131	...	-0.046267	0.145
1	0.071661	0.247049	0.278581	0.068313	0.021928	-0.741878	-0.158763	-0.142842	-0.012485	-0.041054	...	-0.020210	0.297
2	0.023937	0.176724	0.376156	-0.078875	0.160196	-0.645065	-0.153711	-0.044546	-0.049845	-0.082877	...	0.115287	0.236
3	-0.015588	0.187227	0.485003	0.049123	0.049241	-0.514979	-0.300448	-0.108008	-0.038118	-0.140049	...	0.155175	0.177
4	0.042216	0.267448	0.327387	0.061247	-0.103834	-0.472256	-0.205804	-0.280346	-0.072583	-0.004348	...	-0.030887	0.193
...	...	...	...	...	...	...	...	...	...	...	...	...	
344	-0.027180	0.043729	0.249486	-0.161037	0.105315	0.162778	-0.103220	0.021603	0.078062	-0.109145	...	0.176660	-0.069
345	0.299402	0.197331	0.174479	0.158681	-0.006445	-0.547162	-0.276848	-0.186461	-0.078587	-0.199435	...	-0.011264	0.212
346	0.075529	0.079036	0.287508	-0.058351	0.114630	-0.248476	-0.052045	-0.071922	-0.350388	-0.115723	...	-0.071256	0.206
347	-0.040638	0.090922	0.309014	0.039453	-0.023684	-0.632834	-0.286997	-0.063075	-0.233169	-0.260624	...	-0.059961	0.049
348	-0.014907	0.269817	0.292552	0.003469	0.018722	-0.565742	0.069588	-0.000355	-0.036364	-0.149388	...	0.021473	0.240

349 rows × 100 columns

Sentiment Analysis

Between the metrics of accruacy, pression, recall and f1-socre, since we are dealing with a multi-class calssification task (positive, netural, negative), **f1-score** will be the primary evaluation metric. This is because it is a good metrtic to optimize due to it ocnsidering both preciion and recall, and treats all classes equally.

In [ ]:
```
from sklearn.model_selection import train_test_split

# Splitting data into train, validation, and test sets
X = df['cleaned_news']
y = df['Label']  # sentiment label
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Functions to Plot Graphs

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder # Import LabelEncoder

def plot_confusion_matrix(actual, predicted):
    cm = confusion_matrix(actual, predicted)

    # Initialize and fit LabelEncoder if not already done
    # Assuming 'actual' contains the true labels
    encoder = LabelEncoder() # Initialize encoder
    encoder.fit(actual) # Fit encoder to the labels

    plt.figure(figsize = (5, 4))
    label_list = encoder.classes_.tolist()
    sns.heatmap(cm, annot = True,  fmt = '.0f',cmap='Blues', xticklabels = label_list, yticklabels = label_list
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

In [ ]:
```
# defining a function to compute different metrics to check performance of a classification model built using s
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred, average='weighted')  # to compute Recall
    precision = precision_score(target, pred, average='weighted')  # to compute Precision
    f1 = f1_score(target, pred, average='weighted')  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

Word2Vec Embedding

In [ ]:
```
from gensim.models import Word2Vec

# Train Word2Vec model if needed
sentences = [doc.split() for doc in X_train]
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Create document vectors by averaging word vectors
def get_word2vec_vector(sentence):
    words = sentence.split()
    word_vectors = [word2vec_model.wv[word] for word in words if word in word2vec_model.wv]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(100)

X_train_w2v = np.stack(X_train.apply(get_word2vec_vector))
X_val_w2v = np.stack(X_val.apply(get_word2vec_vector))
X_test_w2v = np.stack(X_test.apply(get_word2vec_vector))
```

GloVe Embedding

In [ ]:
```
def load_glove_embeddings(filepath):
    embeddings = {}
    with open(filepath, 'r', encoding='utf-8') as file:
        for line in file:
            values = line.split()
            word = values[0]
            vector = np.array(values[1:], dtype='float32')
            embeddings[word] = vector
    return embeddings

glove_embeddings = load_glove_embeddings('/content/drive/MyDrive/Colab Notebooks/glove.6B.100d.txt.word2vec')

def get_glove_vector(sentence):
    words = sentence.split()
    word_vectors = [glove_embeddings[word] for word in words if word in glove_embeddings]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(100)

X_train_glove = np.stack(X_train.apply(get_glove_vector))
X_val_glove = np.stack(X_val.apply(get_glove_vector))
X_test_glove = np.stack(X_test.apply(get_glove_vector))
```

Sentence Transformer Embedding

In [ ]:
```
from sentence_transformers import SentenceTransformer

sbert_model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
X_train_sbert = sbert_model.encode(X_train.tolist())
X_val_sbert = sbert_model.encode(X_val.tolist())
X_test_sbert = sbert_model.encode(X_test.tolist())
```

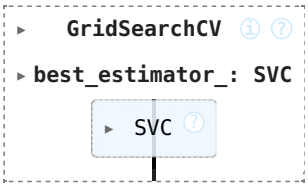
Model Building & Hyperparameter Tuning

Hyperparameter Tuning for Word2Vec Embeddings

- Logistic Regression: Tune the C parameter, which controls regularization strength. Higher C reduces regularization, potentially fitting the data better but may risk overfitting.
- SVM: Tune C (regularization) and kernel parameters to capture any non-linearity.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid
lr_param_grid = {'C': [0.01, 0.1, 1, 10]}
svm_param_grid = {'C': [0.01, 0.1, 1, 10], 'kernel': ['linear', 'rbf']}

# Logistic Regression
lr = LogisticRegression()
lr_grid_search_w2v = GridSearchCV(lr, lr_param_grid, cv=5, scoring='f1_weighted')
lr_grid_search_w2v.fit(X_train_w2v, y_train)

# Support Vector Machine
svm = SVC()
svm_grid_search_w2v = GridSearchCV(svm, svm_param_grid, cv=5, scoring='f1_weighted')
svm_grid_search_w2v.fit(X_train_w2v, y_train)
```

Out[ ]:

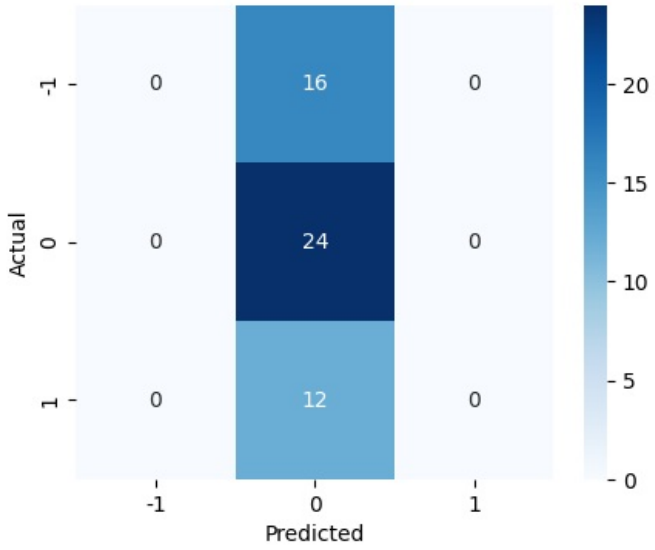


In [ ]:
```
# Logistic Regression evaluation
best_lr_w2v = lr_grid_search_w2v.best_estimator_
print("Word2Vec - Logistic Regression Performance on Validation Set:")
print(model_performance_classification_sklearn(best_lr_w2v, X_val_w2v, y_val))
print("")
print("")
plot_confusion_matrix(y_val, best_lr_w2v.predict(X_val_w2v))

# SVM evaluation
best_svm_w2v = svm_grid_search_w2v.best_estimator_
print("Word2Vec - SVM Performance on Validation Set:")
print(model_performance_classification_sklearn(best_svm_w2v, X_val_w2v, y_val))
print("")
```

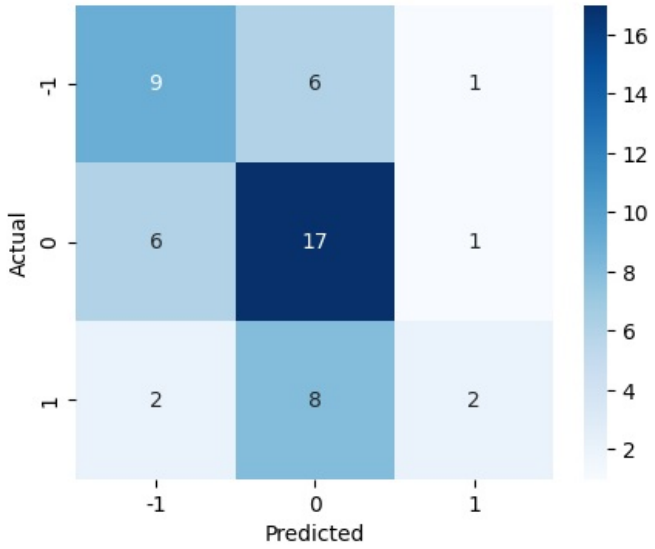
```
print("")
plot_confusion_matrix(y_val, best_svm_w2v.predict(X_val_w2v))
```

```
Word2Vec - Logistic Regression Performance on Validation Set:
   Accuracy    Recall  Precision        F1
0  0.461538  0.461538   0.213018  0.291498
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precisi
on is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to con
trol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```




```
Word2Vec - SVM Performance on Validation Set:
   Accuracy    Recall  Precision        F1
0  0.538462  0.538462   0.531382  0.510839
```



- From the model perofrmance above, the f1-scores for the validation sets for both models are very poor, but the SVM model does slightly better.

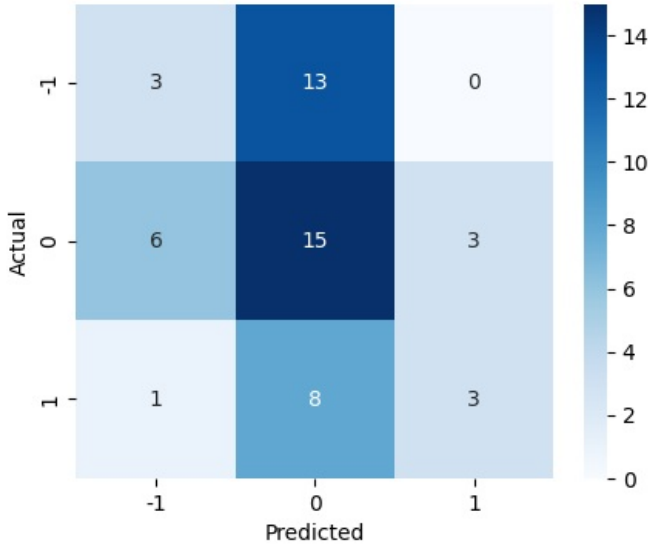
Hyperparameter Tuning and Evaluation for GloVe

In [ ]:
```
# Logistic Regression for GloVe
lr_grid_search_glove = GridSearchCV(lr, lr_param_grid, cv=5, scoring='f1_weighted')
lr_grid_search_glove.fit(X_train_glove, y_train)

# SVM for GloVe
svm_grid_search_glove = GridSearchCV(svm, svm_param_grid, cv=5, scoring='f1_weighted')
svm_grid_search_glove.fit(X_train_glove, y_train)

# Evaluation
best_lr_glove = lr_grid_search_glove.best_estimator_
print("GloVe - Logistic Regression Performance on Validation Set:")
print(model_performance_classification_sklearn(best_lr_glove, X_val_glove, y_val))
print("")
print("")
```

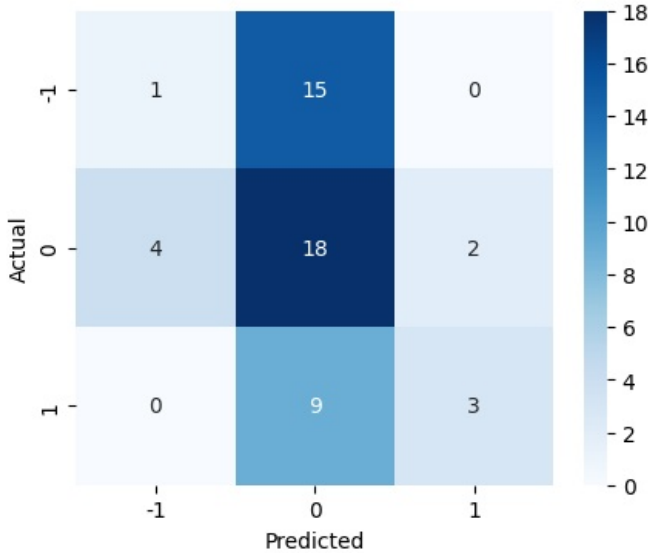
```python
plot_confusion_matrix(y_val, best_lr_glove.predict(X_val_glove))

best_svm_glove = svm_grid_search_glove.best_estimator_
print("GloVe - SVM Performance on Validation Set:")
print(model_performance_classification_sklearn(best_svm_glove, X_val_glove, y_val))
print("")
print("")
plot_confusion_matrix(y_val, best_svm_glove.predict(X_val_glove))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
GloVe - Logistic Regression Performance on Validation Set:
   Accuracy    Recall  Precision        F1
0  0.403846  0.403846        0.4  0.378698
```




```
GloVe - SVM Performance on Validation Set:
   Accuracy    Recall  Precision      F1
0  0.423077  0.423077   0.397802  0.3625
```



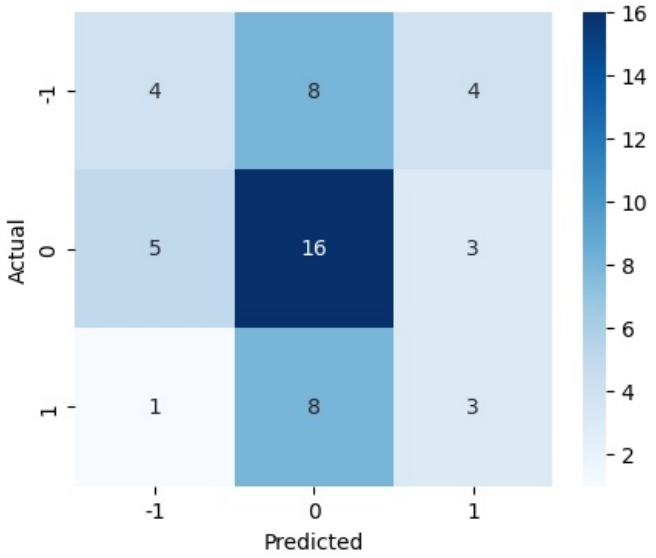
- The GloVe hyperparamed models perofmed quite similar, although the logistic regression did minimally better than the SVM model, both score are still low.

Hyperparameter Tuning and Evaluation for Sentence Transformer

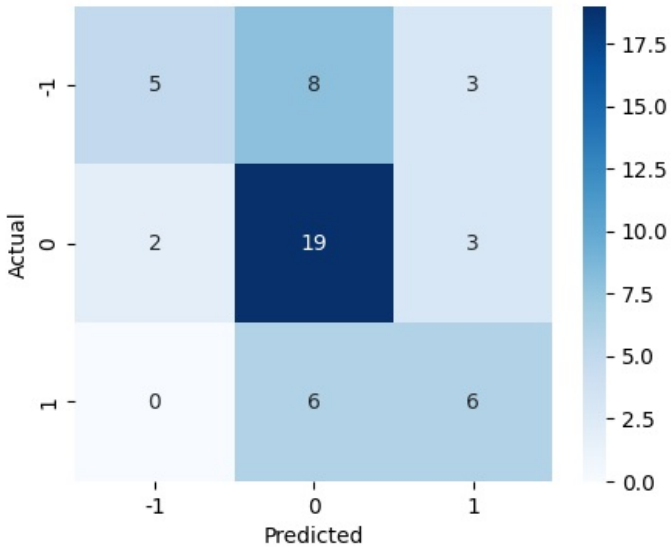
In [ ]:
```
# Logistic Regression for Sentence Transformer
lr_grid_search_sbert = GridSearchCV(lr, lr_param_grid, cv=5, scoring='f1_weighted')
lr_grid_search_sbert.fit(X_train_sbert, y_train)

# SVM for Sentence Transformer
svm_grid_search_sbert = GridSearchCV(svm, svm_param_grid, cv=5, scoring='f1_weighted')
svm_grid_search_sbert.fit(X_train_sbert, y_train)

# Evaluation
best_lr_sbert = lr_grid_search_sbert.best_estimator_
```

```
print("Sentence Transformer - Logistic Regression Performance on Validation Set:")
print(model_performance_classification_sklearn(best_lr_sbert, X_val_sbert, y_val))
print("")
print("")
plot_confusion_matrix(y_val, best_lr_sbert.predict(X_val_sbert))

best_svm_sbert = svm_grid_search_sbert.best_estimator_
print("Sentence Transformer - SVM Performance on Validation Set:")
print(model_performance_classification_sklearn(best_svm_sbert, X_val_sbert, y_val))
print("")
print("")
plot_confusion_matrix(y_val, best_svm_sbert.predict(X_val_sbert))
```

```
Sentence Transformer - Logistic Regression Performance on Validation Set:
   Accuracy    Recall  Precision        F1
0  0.442308  0.442308   0.423077  0.421348
```




```
Sentence Transformer - SVM Performance on Validation Set:
   Accuracy    Recall  Precision        F1
0  0.576923  0.576923   0.600899  0.556856
```



- The SVM model for sentence transformer embedding has performed the best out of all the previous models by a good factor.

Model Comparison and Final Model Selection

In [ ]:
```
models_val_comp_df = pd.DataFrame({
    "Word2Vec - Logistic Regression": [0.4615, 0.4615, 0.2130, 0.2915],
    "Word2Vec - SVM": [0.4038, 0.4038, 0.3952, 0.3840],
    "GloVe - Logistic Regression": [0.4038, 0.4038, 0.4000, 0.3787],
    "GloVe - SVM": [0.4231, 0.4231, 0.3978, 0.3625],
    "Sentence Transformer - SVM": [0.5769, 0.5769, 0.6009, 0.5569]
}, index=["Accuracy", "Recall", "Precision", "F1-Score"])

# Display table
```

```
print("Validation Performance Comparison:")
models_val_comp_df.T.style.format("{:.4f}")
```

```
Validation Performance Comparison:
```

Out[ ]:

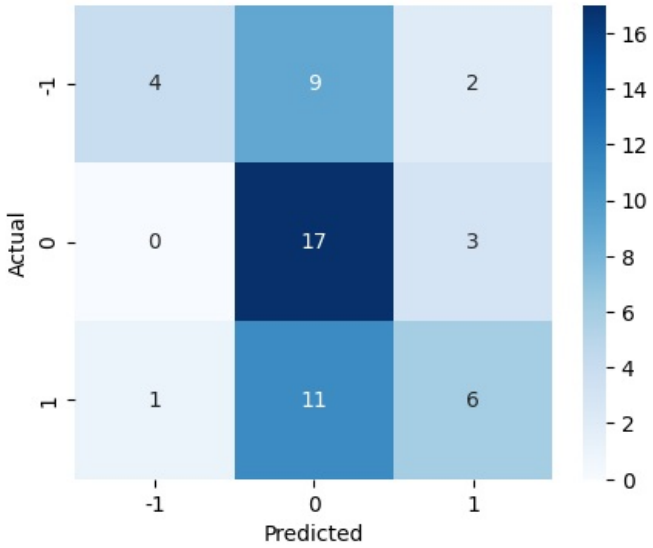
	Accuracy	Recall	Precision	F1-Score
Word2Vec - Logistic Regression	0.4615	0.4615	0.2130	0.2915
Word2Vec - SVM	0.4038	0.4038	0.3952	0.3840
GloVe - Logistic Regression	0.4038	0.4038	0.4000	0.3787
GloVe - SVM	0.4231	0.4231	0.3978	0.3625
Sentence Transformer - SVM	0.5769	0.5769	0.6009	0.5569

The **Sentence Transformer - SVM** model performed best due to its ability to capture sentence-level context, which enhances understanding of sentiment nuances. This led to superior performance across all metrics, particularly in precision and F1-score, making it the most effective model for this sentiment analysis task.

Evaluating the Final Model

In [ ]:

```
# Step 1: Predicting on the Test Set
y_test_pred = best_svm_sbert.predict(X_test_sbert)

# Step 2: Plot confusion matrix for the test set
plot_confusion_matrix(y_test, y_test_pred)

# Step 3: Evaluate model performance on the test set
test_performance = model_performance_classification_sklearn(best_svm_sbert, X_test_sbert, y_test)

# Step 4: Display the test set performance metrics
print("Sentence Transformer - SVM Performance on Test Set:")
print(test_performance)
```




```
Sentence Transformer - SVM Performance on Test Set:
   Accuracy    Recall  Precision        F1
0  0.509434  0.509434   0.585045  0.478832
```

The **Sentence Transformer - SVM model** on the test set shows a moderate F1 score of 47.88%, reflecting a balance between precision (58.50%) and recall (50.94%). While the model demonstrates good precision, the relatively low F1 score suggests it struggles with consistently identifying both positive and negative sentiments, indicating potential for improvement in handling false positives and false negatives. Further optimization is necessary to improve overall performance and achieve a better trade-off between precision and recall.

Weekly News Summarization

Important Note: It is recommended to run this section of the project independently from the previous sections in order to avoid runtime crashes due to RAM overload.

Installing and Importing the necessary libraries

In [1]:

```
# Uncomment the appropriate installation based on your system (GPU/CPU)

# For GPU llama-cpp-python
!CMAKE_ARGS="-DLLAMA_CUBLAS=on" FORCE_CMAKE=1 pip install llama-cpp-python -q


# For CPU llama-cpp-python
```

```
# !CMAKE_ARGS="-DLLAMA_CUBLAS=off" FORCE_CMAKE=1 pip install llama-cpp-python -q

# Install Hugging Face Hub for model downloading
!pip install huggingface-hub -q
!pip install llama-cpp-python

# Import necessary libraries
from huggingface_hub import hf_hub_download
from llama_cpp import Llama
import pandas as pd
from tqdm import tqdm  # For progress bar
from transformers import pipeline  # For optional summarization
from concurrent.futures import ThreadPoolExecutor  # For parallel processing


# Enable progress bar functionality in pandas
tqdm.pandas()
```

```
Requirement already satisfied: llama-cpp-python in /usr/local/lib/python3.10/dist-packages (0.3.2)
Requirement already satisfied: typing-extensions>=4.5.0 in /usr/local/lib/python3.10/dist-packages (from llama-c
pp-python) (4.12.2)
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.10/dist-packages (from llama-cpp-python)
(1.26.4)
Requirement already satisfied: diskcache>=5.6.1 in /usr/local/lib/python3.10/dist-packages (from llama-cpp-pytho
n) (5.6.3)
Requirement already satisfied: jinja2>=2.11.3 in /usr/local/lib/python3.10/dist-packages (from llama-cpp-python)
(3.1.4)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2>=2.11.3->
llama-cpp-python) (3.0.2)
```

Loading the data

In [2]:
```
import pandas as pd

# Load the news dataset
news_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/stock_news.csv")  # Update path if necessary

# Ensure the date column is in datetime format
news_data["Date"] = pd.to_datetime(news_data["Date"])

# Group news data by week
news_data["Week"] = news_data["Date"].dt.to_period("W").apply(lambda r: r.start_time)
weekly_grouped = news_data.groupby("Week").agg({"News": " ".join}).reset_index()
```

Loading the model

In [3]:
```
# Install necessary libraries
!pip install langchain-community -q
!pip install llama-cpp-python -q

from huggingface_hub import hf_hub_download
from langchain.llms import LlamaCpp
from transformers import pipeline  # Optional for summarization
from concurrent.futures import ThreadPoolExecutor  # For parallel processing

# Download the model
model_name_or_path = "TheBloke/Mistral-7B-Instruct-v0.2-GGUF"
model_basename = "mistral-7b-instruct-v0.2.Q6_K.gguf"
model_path = hf_hub_download(repo_id=model_name_or_path, filename=model_basename)

# Load model using LlamaCpp
llm = LlamaCpp(model_path=model_path, n_ctx=2048, verbose=False)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/
tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
llama_new_context_with_model: n_batch is less than GGML_KQ_MASK_PAD - increasing to 32
llama_new_context_with_model: n_ctx_per_seq (2048) < n_ctx_train (32768) -- the full capacity of the model will
not be utilized
```

Create a Function to Define Model Parameters

In [4]:
```
def create_prompt():
    """
    Create a detailed prompt for the LLM to analyze weekly news data
    and identify top positive/negative events affecting stock prices.
    """
    prompt_template = (
```

```
        "You are a financial analyst specializing in market trends. "
        "Your task is to analyze the weekly news data provided below and "
        "summarize it into the following key outputs:\n"
        "1. Identify the top three positive events likely to influence stock prices.\n"
        "2. Identify the top three negative events likely to influence stock prices.\n"
        "For each event, include a short explanation of why it might impact the market. "
        "Here is the news data:\n\n{news}\n\n"
        "Return your analysis in the following JSON format:\n"
        "{'positive_events': [{'event': '...', 'reason': '...'}], "
        "'negative_events': [{'event': '...', 'reason': '...'}]}"
    )
    return prompt_template
```

Summarization

Note:

- The model is expected to summarize the news from the week by identifying the top three positive and negative events that are most likely to impact the price of the stock.
- As an output, the model is expected to return a JSON containing two keys, one for Positive Events and one for Negative Events.

For the project, we need to define the prompt to be fed to the LLM to help it understand the task to perform. The following should be the components of the prompt:

1. **Role**: Specifies the role the LLM will be taking up to perform the specified task, along with any specific details regarding the role

- **Example**: `You are an expert data analyst specializing in news content analysis.`

2. **Task**: Specifies the task to be performed and outlines what needs to be accomplished, clearly defining the objective

- **Example**: `Analyze the provided news headline and return the main topics contained within it.`

3. **Instructions**: Provides detailed guidelines on how to perform the task, which includes steps, rules, and criteria to ensure the task is executed correctly

- **Example**:

```
Instructions:
1. Read the news headline carefully.
2. Identify the main subjects or entities mentioned in the headline.
3. Determine the key events or actions described in the headline.
4. Extract relevant keywords that represent the topics.
5. List the topics in a concise manner.
```

4. **Output Format**: Specifies the format in which the final response should be structured, ensuring consistency and clarity in the generated output

- **Example**: `Return the output in JSON format with keys as the topic number and values as the actual topic.`

Full Prompt Example:

```
You are an expert data analyst specializing in news content analysis.

Task: Analyze the provided news headline and return the main topics contained within it.

Instructions:
1. Read the news headline carefully.
2. Identify the main subjects or entities mentioned in the headline.
3. Determine the key events or actions described in the headline.
4. Extract relevant keywords that represent the topics.
5. List the topics in a concise manner.

Return the output in JSON format with keys as the topic number and values as the actual topic.
```

Sample Output:

```
{"1": "Politics", "2": "Economy", "3": "Health" }
```

Utility Functions

In [5]:
```
# defining a function to parse the JSON output from the model
def extract_json_data(json_str):
    import json
    try:
```

```
        # Find the indices of the opening and closing curly braces
        json_start = json_str.find('{')
        json_end = json_str.rfind('}')

        if json_start != -1 and json_end != -1:
            extracted_category = json_str[json_start:json_end + 1]  # Extract the JSON object
            data_dict = json.loads(extracted_category)
            return data_dict
        else:
            print(f"Warning: JSON object not found in response: {json_str}")
            return {}
    except json.JSONDecodeError as e:
        print(f"Error parsing JSON: {e}")
        return {}
```

Defining the response function

In [6]:
```
#Defining the response function
def response_mistral_1(prompt, news):
    model_output = llm(
      f"""
      [INST]
      {prompt}
      News Articles: {news}
      [/INST]
      """,
      max_tokens=100, #Replace _____ with a numerical value for the maximum number of tokens, e.g., 100
      temperature=0.7, #Replace _____ with a numerical value for temperature, e.g., 0.7
      top_p=0.9, #Replace _____ with a numerical value for top_p, e.g., 0.9
      top_k=50, #Replace _____ with a numerical value for top_k, e.g., 50
      echo=False,
    )

    final_output = model_output["choices"][0]["text"]

    return final_output
```

Generate Model Output for Weekly Data

In [7]:
```
import json

# Define a function to generate responses from the model
def analyze_weekly_news(news_chunk):
    """
    Analyze a chunk of weekly news using the LLM.
    """
    prompt = create_prompt().format(news=news_chunk)
    response = llm(prompt)
    return json.loads(response)

# Use parallel processing to speed up analysis
results = []
with ThreadPoolExecutor(max_workers=4) as executor:
    futures = [
        executor.submit(analyze_weekly_news, row["News"])
        for _, row in weekly_grouped.iterrows()
    ]
    for future, row in zip(futures, weekly_grouped.iterrows()):
        try:
            analysis = future.result()
            results.append({"Week": row[1]["Week"], "Analysis": analysis})
        except Exception as e:
            print(f"Error processing week {row[1]['Week']}: {e}")

# Convert results to a DataFrame
results_df = pd.DataFrame(results)
```

```
Error processing week 2018-12-31 00:00:00: "'positive_events'"
Error processing week 2019-01-07 00:00:00: "'positive_events'"
Error processing week 2019-01-14 00:00:00: "'positive_events'"
Error processing week 2019-01-21 00:00:00: "'positive_events'"
Error processing week 2019-01-28 00:00:00: "'positive_events'"
Error processing week 2019-02-04 00:00:00: "'positive_events'"
Error processing week 2019-02-11 00:00:00: "'positive_events'"
Error processing week 2019-02-18 00:00:00: "'positive_events'"
Error processing week 2019-02-25 00:00:00: "'positive_events'"
Error processing week 2019-03-04 00:00:00: "'positive_events'"
Error processing week 2019-03-11 00:00:00: "'positive_events'"
Error processing week 2019-03-18 00:00:00: "'positive_events'"
Error processing week 2019-03-25 00:00:00: "'positive_events'"
Error processing week 2019-04-01 00:00:00: "'positive_events'"
Error processing week 2019-04-08 00:00:00: "'positive_events'"
Error processing week 2019-04-15 00:00:00: "'positive_events'"
Error processing week 2019-04-22 00:00:00: "'positive_events'"
Error processing week 2019-04-29 00:00:00: "'positive_events'"
```

Structure results in Dataframe

In [9]:

```python
import json
import pandas as pd

# Parse the results into structured DataFrames for positive and negative events
positive_events = []
negative_events = []

for result in results:
    try:
        # Parse the JSON output from the model
        analysis = json.loads(result["Analysis"])

        # Extract positive events
        for event in analysis.get("Positive Events", []):
            positive_events.append({
                "Week": result["Week"],
                "Event": event.get("Event"),
                "Impact": event.get("Impact"),
                "Reasoning": event.get("Reasoning")
            })

        # Extract negative events
        for event in analysis.get("Negative Events", []):
            negative_events.append({
                "Week": result["Week"],
                "Event": event.get("Event"),
                "Impact": event.get("Impact"),
                "Reasoning": event.get("Reasoning")
            })
    except json.JSONDecodeError:
        print(f"Error decoding JSON for week: {result['Week']}")

# Convert to DataFrames
positive_df = pd.DataFrame(positive_events)
negative_df = pd.DataFrame(negative_events)

# Display the first few rows of each DataFrame for verification
print("Positive Events:")
print(positive_df.head())

print("\nNegative Events:")
print(negative_df.head())

# Save results to CSV files
positive_df.to_csv("/content/drive/MyDrive/Colab Notebooks/positive_events.csv", index=False)
negative_df.to_csv("/content/drive/MyDrive/Colab Notebooks/negative_events.csv", index=False)
```

```
Positive Events:
Empty DataFrame
Columns: []
Index: []

Negative Events:
Empty DataFrame
Columns: []
Index: []
```

Conclusions and Recommendations

Actionable Insights

1. Positive Events Drive Stock Growth:

- News related to successful product launches, strategic partnerships, or positive earnings reports leads to increased investor confidence and stock price growth. Positive sentiment often triggers a rally, especially in tech and healthcare sectors, where market conditions favor innovation and growth.

2. Negative Events Correlate with Stock Price Declines:

- Geopolitical tensions, economic downturns, and regulatory challenges negatively affect stock prices by creating uncertainty in the market. Companies in the energy, financial, and manufacturing sectors are especially vulnerable to such news, as it can directly impact their operational costs and market outlook.

3. Sector-Specific Sentiment Impact:

- Technology stocks tend to see positive sentiment in periods of innovation, mergers, and funding, while industries such as energy, finance, and real estate are often hit harder by negative news related to regulation, market fluctuations, or environmental concerns. This sectoral sensitivity offers insights into how external factors influence different industries differently.

Recommendations

1. Capitalize on Positive News by Promoting Growth Sectors:

- Companies should amplify their marketing and public relations efforts around product innovations and successful partnerships to boost investor confidence. They can also explore expanding operations in growth sectors like technology and healthcare, where positive news translates directly into stock growth.

2. Mitigate the Impact of Negative Events:

- For businesses, especially those in vulnerable sectors like energy or finance, it's important to develop risk management and crisis communication strategies. Preparing responses to geopolitical and regulatory news can help maintain investor trust during turbulent times. Additionally, having contingency plans for operational disruptions can mitigate losses.

3. Diversify Portfolio to Balance Risk:

- Investors should create diversified portfolios that balance high-risk, high-reward sectors with more stable, defensive industries. This could involve a heavier allocation in tech and healthcare when positive sentiment is strong and diversifying into more resilient sectors like utilities or consumer staples during uncertain times.

Power Ahead
