Problem Statement

```
import nbformat as nbf

in_path = "/content/drive/MyDrive/Colab Notebooks/Stock_Market_News_Sentiment_Analysis.ipynb" # **ACTION REQUIRED: Please VERIFY
out_path = "/content/stock_market_news_sentiment_analysis_clean.ipynb"

# --- Please update 'in_path' above with the correct filename from the list above ---

nb = nbf.read(in_path, as_version=4)

# remove widget metadata
nb.metadata.pop("widgets", None)
nb.metadata.pop("widget_state", None)
nb.metadata.pop("widget", None)

for cell in nb.cells:
    if isinstance(cell.get("metadata"), dict):
        cell["metadata"].pop("widgets", None)

nbf.write(nb, out_path)

print("Clean notebook saved as:", out_path)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/usr/local/lib/python3.12/dist-packages/nbformat/__init__.py in read(fp, as_version, capture_validation_error, **kwargs)
    168     try:
--> 169         buf = fp.read()
    170     except AttributeError:

AttributeError: 'str' object has no attribute 'read'

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
                                 1 frames
/usr/local/lib/python3.12/dist-packages/nbformat/__init__.py in read(fp, as_version, capture_validation_error, **kwargs)
    169         buf = fp.read()
    170     except AttributeError:
--> 171         with open(fp, encoding="utf8") as f:  # noqa: PTH123
    172             return reads(f.read(), as_version, capture_validation_error, **kwargs)
    173

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/MyDrive/Colab
Notebooks/Stock_Market_News_Sentiment_Analysis.ipynb'
```

Next steps: Explain error

```
import nbformat as nbf

in_path = "/content/drive/MyDrive/Colab Notebooks/Stock_Market_News_Sentiment_Analysis.ipynb" # **ACTION REQUIRED: Please verif
out_path = "/content/stock_market_news_sentiment_analysis_clean.ipynb"

nb = nbf.read(in_path, as_version=4)

# remove widget metadata
nb.metadata.pop("widgets", None)
nb.metadata.pop("widget_state", None)
nb.metadata.pop("widget", None)

for cell in nb.cells:
    if isinstance(cell.get("metadata"), dict):
        cell["metadata"].pop("widgets", None)

nbf.write(nb, out_path)

print("Clean notebook saved as:", out_path)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/usr/local/lib/python3.12/dist-packages/nbformat/__init__.py in read(fp, as_version, capture_validation_error, **kwargs)
    168     try:
--> 169         buf = fp.read()
    170     except AttributeError:

AttributeError: 'str' object has no attribute 'read'

During handling of the above exception, another exception occurred:

FileNotFoundError                         Traceback (most recent call last)
 1 frames
/usr/local/lib/python3.12/dist-packages/nbformat/__init__.py in read(fp, as_version, capture_validation_error, **kwargs)
    169         buf = fp.read()
    170     except AttributeError:
--> 171         with open(fp, encoding="utf8") as f:  # noqa: PTH123
    172             return reads(f.read(), as_version, capture_validation_error, **kwargs)
    173

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/MyDrive/Colab
Notebooks/Stock_Market_News_Sentiment_Analysis.ipynb'
```

Next steps: Explain error

```python
import nbformat as nbf

# change this to your uploaded notebook filename in Colab Files
in_path = "/content/drive/MyDrive/Colab Notebooks/Stock_Market_News_Sentiment_Analysis.ipynb" # Corrected path
out_path = "/content/stock_market_news_sentiment_analysis_clean.ipynb"

nb = nbf.read(in_path, as_version=4)

# Remove widget metadata that breaks GitHub rendering
nb.metadata.pop("widgets", None)
nb.metadata.pop("widget_state", None) # Corrected typo from b.metadata to nb.metadata
nb.metadata.pop("widget", None)

# Also remove per cell widget metadata if present
for cell in nb.cells:
    if "metadata" in cell and isinstance(cell["metadata"], dict):
        cell["metadata"].pop("widgets", None)

nbf.write(nb, out_path)
print("Clean notebook saved to:", out_path)
```

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 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.

↳ 1 cell hidden

Installing and Importing the necessary libraries

```
# installing the sentence-transformers and gensim libraries for word embeddings
!pip install numpy==1.26.4 \
             scikit-learn==1.6.1 \
             scipy==1.13.1 \
             gensim==4.3.3 \
             sentence-transformers==3.4.1 \
             pandas==2.2.2
```

Note:

- After running the above cell, kindly restart the runtime (for Google Colab) or notebook kernel (for Jupyter Notebook), and run all cells sequentially from the next cell.
- On executing the above line of code, you might see a warning regarding package dependencies. This error message can be ignored as the above code ensures that all necessary libraries and their dependencies are maintained to successfully execute the code in this notebook.

```
# To manipulate and analyze data
import pandas as pd
import numpy as np

# To visualize data
import matplotlib.pyplot as plt
import seaborn as sns

# To used time related functions
# To used time-related functions
import time

# To build, tune, and evaluate ML models
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score

# To load/create word embeddings
from gensim.models import Word2Vec

# To work with transformer models
import torch
from sentence_transformers import SentenceTransformer

# Import TensorFlow and Keras for deep learning model building.
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# To implement progress bar related functionalities
from tqdm import tqdm
tqdm.pandas()

# To ignore unnecessary warnings
import warnings
warnings.filterwarnings('ignore')
```

```
WARNING:torchao.kernel.intmm:Warning: Detected no triton, on systems without Triton certain kernels will not work
```

Loading the dataset

```
# # uncomment and run the following code if Google Colab is being used and the dataset is in Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
stock_news = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/stock_news.csv')
```

```
#Creating a copy of the dataset
stock = stock_news.copy()
```

Data Overview

Displaying the first few rows of the dataset

```
stock.head() # Complete the code to check the first 5 rows of the data
```

	Date	News	Open	High	Low	Close	Volume	Label
0	01-02-2019	The dollar minutes ago tumbled to 106 67 from...	38.72	39.71	38.56	39.48	130672400	1
1	01-02-2019	By Wayne Cole and Swati Pandey SYDNEY Reuters...	38.72	39.71	38.56	39.48	130672400	-1
2	01-02-2019	By Stephen Culp NEW YORK Reuters Wall Stre...	38.72	39.71	38.56	39.48	130672400	0
3	01-02-2019	By Wayne Cole SYDNEY Reuters The Australia...	38.72	39.71	38.56	39.48	130672400	-1
4	01-02-2019	Investing com Asian equities fell in morning...	38.72	39.71	38.56	39.48	130672400	1

Next steps: Generate code with stock New interactive sheet

Understanding the shape of the dataset

```
stock.shape # Complete the code to check the shape of the data
```

```
(418, 8)
```

```
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->senter
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-t
Requirement already satisfied: shellingham in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-t
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-t
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas==2.2.2
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open>=1.8.1->gensim==4.3.3) (2.1.0
Collecting huggingface-hub>=0.20.0 (from sentence-transformers==3.4.1)
  Downloading huggingface_hub-0.36.1-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers<5.0.0,>=4.41.0-
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers<5.0.0,>=4.41.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->transformers<5.0.0,>=4.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->transformers<5.0.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->transformers<5.0.
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0.20.
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hul
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1,>=0.23.0->hug
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from typer-slim->huggingface-hub>=0.20
Downloading numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.0 MB)
Downloading gensim-4.3.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.6/26.6 MB 46.5 MB/s eta 0:00:00
Downloading transformers-4.57.6-py3-none-any.whl (12.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 99.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 566.3/566.3 kB 40.9 MB/s eta 0:00:00
Installing collected packages: numpy, scipy, huggingface-hub, gensim, transformers, sentence-transformers
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.16.3
    Uninstalling scipy-1.16.3:
      Successfully uninstalled scipy-1.16.3
  Attempting uninstall: huggingface-hub
  Attempting uninstall: transformers
    Found existing installation: transformers 5.0.0
    Uninstalling transformers-5.0.0:
```

Checking the data types of the columns

```
  Successfully uninstalled transformers-5.0.0
  Attempting uninstall: sentence-transformers
    Found existing installation: sentence-transformers 5.2.2
```

```
stock.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    418 non-null    object
 1   News    418 non-null    object
 2   Open    418 non-null    float64
 3   High    418 non-null    float64
 4   Low     418 non-null    float64
 5   Close   418 non-null    float64
 6   Volume  418 non-null    int64
 7   Label   418 non-null    int64
dtypes: float64(4), int64(2), object(2)
memory usage: 26.3+ KB
```

```
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is th
jaxlib 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
access 1.1.10.post3 requires scipy>=1.14.1, but you have scipy 1.13.1 which is incompatible.
pytensor 2.37.0 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
opencv-contrib-python 4.13.0.90 requires numpy>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompatible.
jax 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
shap 0.50.0 requires numpy>=2, but you have numpy 1.26.4 which is incompatible.
tsfresh 0.21.1 requires scipy>=1.14.0; python_version >= "3.10", but you have scipy 1.13.1 which is incompatible.
opencv-python 4.13.0.90 requires numpy>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompatible.
opencv-python-headless 4.13.0.90 requires numpy>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompatible.
tobler 0.13.0 requires numpy>=2.0, but you have numpy 1.26.4 which is incompatible.
rasterio 1.5.0 requires numpy>=2, but you have numpy 1.26.4 which is incompatible.
Successfully installed gensim-4.3.3 huggingface-hub-0.36.1 numpy-1.26.4 scipy-1.13.1 sentence-transformers-3.4.1 transformers-
WARNING: The following packages were previously imported in this runtime:
  [numpy]
You must restart the runtime in order to use newly installed versions.
```

```
# This conversion is now handled after the last data load in cell c32d69d8
# stock['Date'] = pd.to_datetime(stock['Date'], format='%m-%d-%Y')
```

Checking the statistical summary

```
stock.describe().T # Complete the code to check the statistical summary
```

	count	mean	std	min	25%	50%	75%	max
Open	418.0	4.230885e+01	4.947134e+00	35.99	3.813000e+01	4.153000e+01	4.719000e+01	5.184000e+01
High	418.0	4.278732e+01	4.947413e+00	36.43	3.842000e+01	4.225000e+01	4.742750e+01	5.212000e+01
Low	418.0	4.192373e+01	4.967880e+00	35.50	3.772000e+01	4.114000e+01	4.648000e+01	5.176000e+01
Close	418.0	4.241852e+01	4.999867e+00	35.55	3.827000e+01	4.161000e+01	4.703250e+01	5.187000e+01
Volume	418.0	1.294225e+08	4.574550e+07	45448000.00	1.029072e+08	1.156272e+08	1.511252e+08	3.652488e+08
Label	418.0	3.086124e-01	9.434731e-01	-1.00	-1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

Checking the duplicate values

```
stock.duplicated().sum() #Complete the code to check the duplicate values
```

```
0
```

Checking for missing values

```
stock.isnull().sum()
```

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

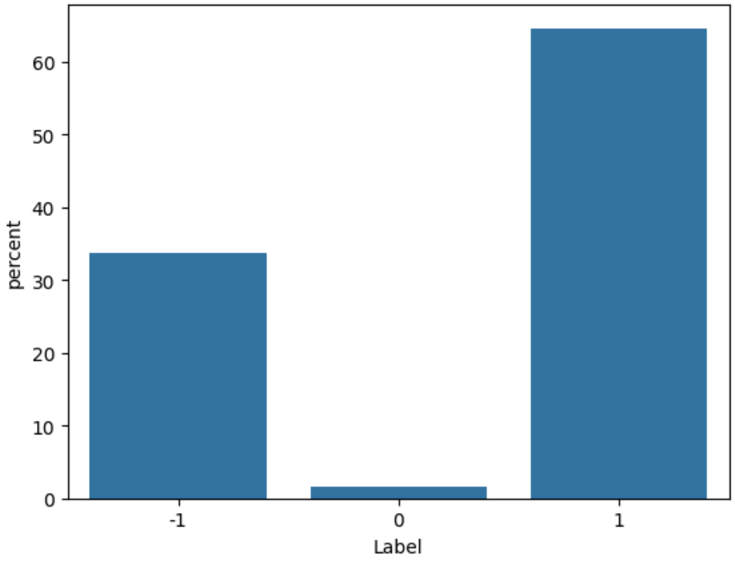
dtype: int64

Exploratory Data Analysis

Univariate Analysis

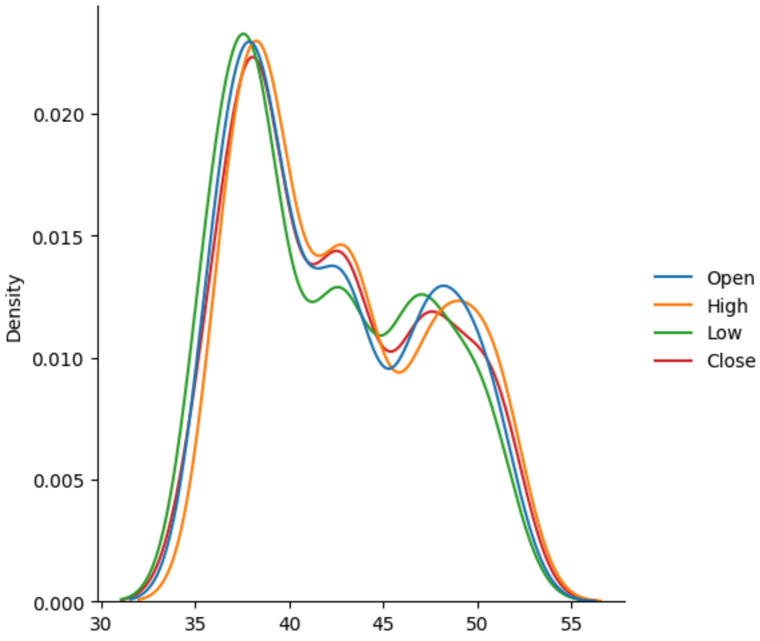
Countplot on Label

```
sns.countplot(data=stock, x='Label', stat="percent");
```



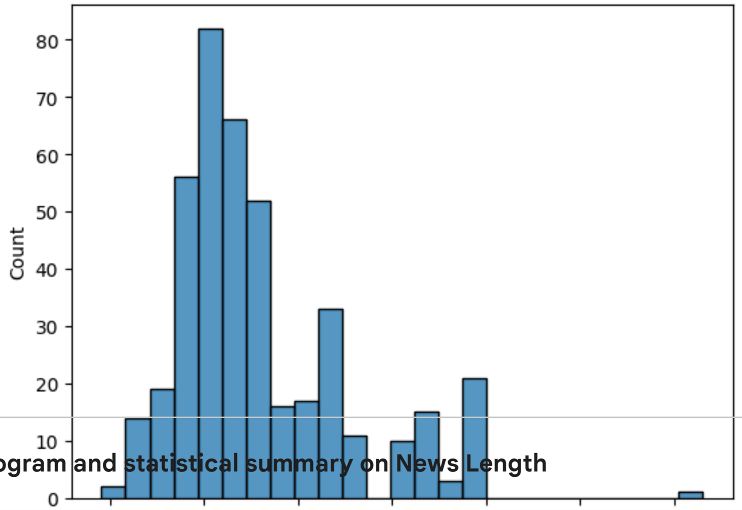
Density Plot of Price (Open, High, Low, Close)

```
# Plot KDE for the 'Open', 'High', 'Low', 'Close' columns of the 'stock' DataFrame.
sns.displot(data=stock[['Open','High','Low','Close']], kind='kde', palette="tab10");  # Create a KDE plot with a color palette.
```



Histogram on Volume

```
sns.histplot(stock, x='Volume'); # Complete the code to plot a histogram of Volume
```

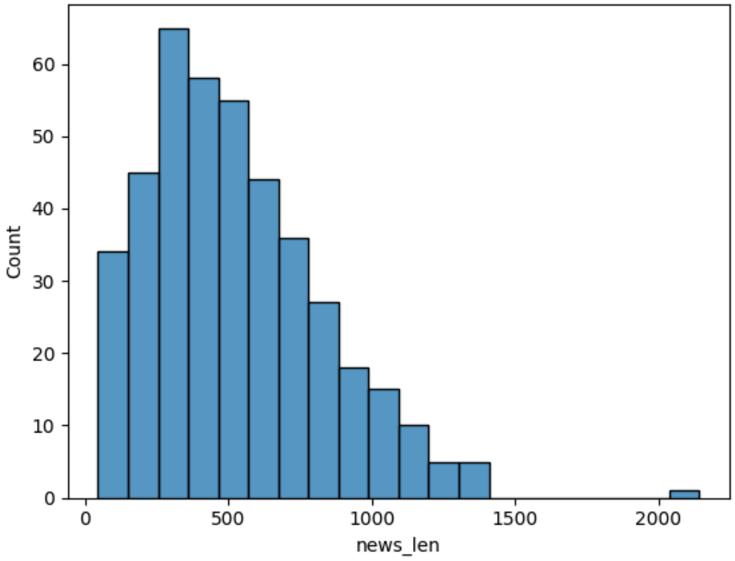


Histogram and statistical summary on News Length

```
#Calculating the total number of words present in the news content.
stock['news_len'] = stock['News'].apply(lambda x: len(x.split(' ')))

stock['news_len'].describe()    #Complete the code to print the statistical summary for the news content length
```

```
          news_len
count   418.000000
mean    525.662679
std     303.584080
min      44.000000
25%     304.250000
50%     480.000000
75%     700.500000
max    2142.000000

dtype: float64
```

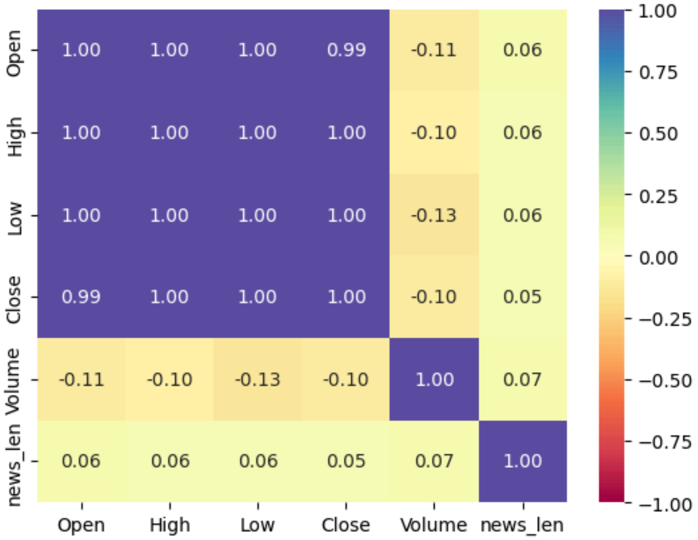
```
sns.histplot(data=stock, x="news_len"); #Complete the code to plot a histogram of news_len column
```



Bivariate Analysis

Correlation

```
cols = ['Open','High','Low','Close','Volume','news_len']
sns.heatmap(
    stock[cols].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```

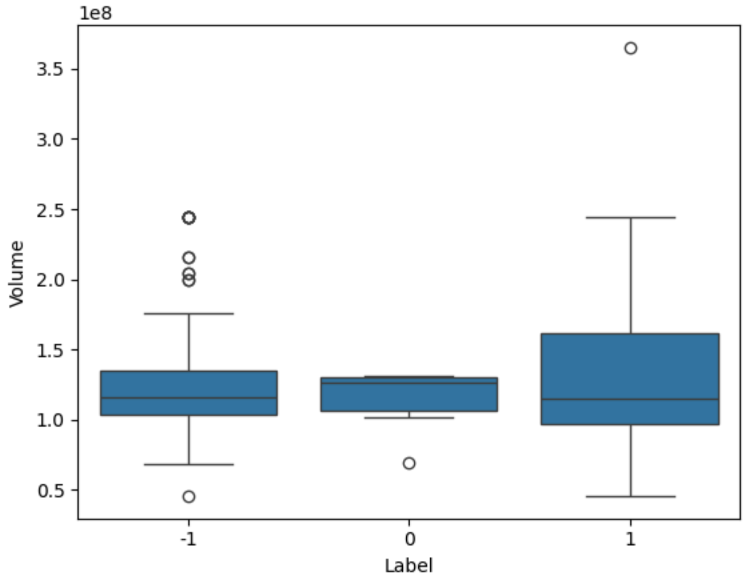


Label vs Price (Open, High, Low, Close)

```
plt.figure(figsize=(10, 8))

for i, variable in enumerate(['Open', 'High', 'Low', 'Close']):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=stock, x="Label", y=variable)
    plt.tight_layout(pad=2)

plt.show()
```

Label vs Volume

```
sns.boxplot(
    data=stock, x="Label", y="Volume"  # Complete the code to plot a boxplot of Label vs Volume
);
```



Date vs Price (Open, High, Low, Close)

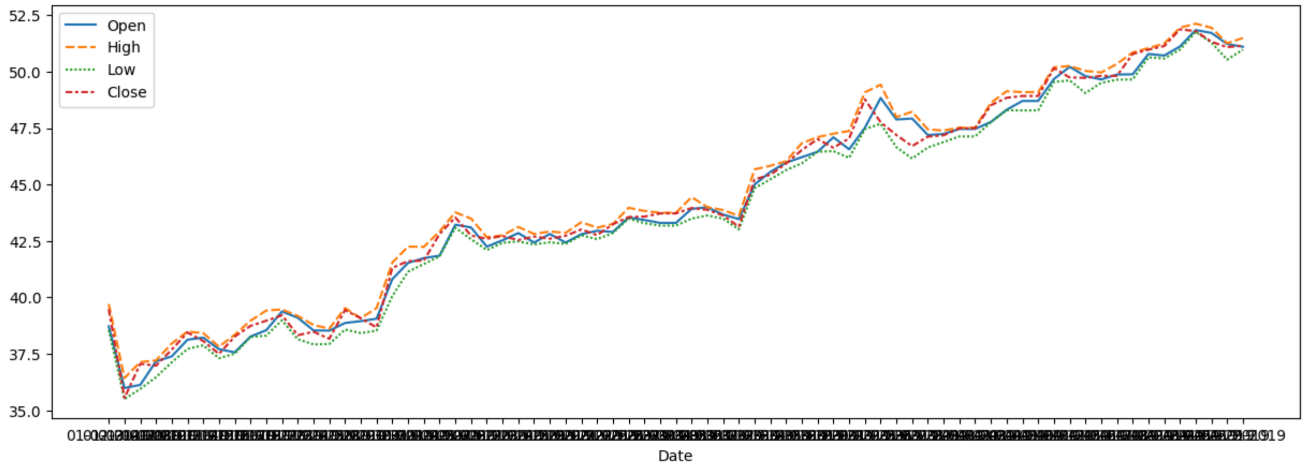
- The data is at the level of news, and we might have more than one news in a day. However, the prices are at daily level
- So, we can aggregate the data at a daily level by taking the mean of the attributes

```
stock_daily = stock.groupby('Date').agg(
    {
        'Open': 'mean',
        'High': 'mean',
        'Low': 'mean',
        'Close': 'mean',
        'Volume': 'mean',
    }
).reset_index()  # Group the 'stocks' DataFrame by the 'Date' column

stock_daily.set_index('Date', inplace=True)
stock_daily.head()
```

Date	Open	High	Low	Close	Volume
01-02-2019	38.72	39.71	38.56	39.48	130672400.0
01-03-2019	35.99	36.43	35.50	35.55	103544800.0
01-04-2019	36.13	37.14	35.95	37.06	111448000.0
01-07-2019	37.17	37.21	36.47	36.98	109012000.0
01-08-2019	37.39	37.96	37.13	37.69	216071600.0

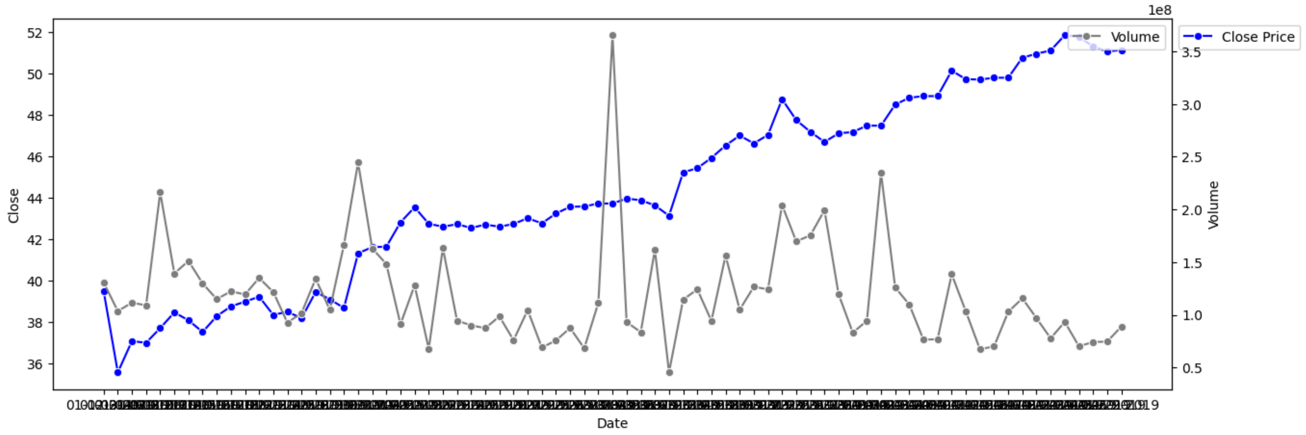
Next steps: Generate code with `stock_daily` New interactive sheet

```
plt.figure(figsize=(15,5))
sns.lineplot(stock_daily.drop("Volume", axis=1)); #Complete the code to plot a lineplot of all the variables except Volume
```



Volume vs Close Price

```
# Create a figure and axis
fig, ax1 = plt.subplots(figsize=(15,5))

# Lineplot on primary y-axis
sns.lineplot(data=stock_daily.reset_index(), x='Date', y='Close', ax=ax1, color='blue', marker='o', label='Close Price')

# Create a secondary y-axis
ax2 = ax1.twinx()

# Lineplot on secondary y-axis
sns.lineplot(data=stock_daily.reset_index(), x='Date', y='Volume', ax=ax2, color='gray', marker='o', label='Volume')

ax1.legend(bbox_to_anchor=(1,1));
```



Data Preprocessing

```
stock['Date'].describe()
```

	Date
count	418
unique	73
top	01-03-2019
freq	37

dtype: object

```
stock_news = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/stock_news.csv')
```

```
stock = stock_news.copy()
# Convert the 'Date' column in the 'stock' DataFrame to datetime format after copying
stock['Date'] = pd.to_datetime(stock['Date'], format='%m-%d-%Y')
```

Train-test Split

```
print(stock['Date'].dtype)
```

```
datetime64[ns]
```

```
# Select training data: choose all rows where the 'Date' is before a specific date of your choice.
# Pick a date that helps split the data into a reasonable train-test ratio for your model (format: 'yyyy-mm-dd').
cutoff_date = pd.Timestamp('2019-03-25')
X_train = stock[stock['Date'] < cutoff_date].reset_index(drop=True)

# Select testing data: choose all rows where the 'Date' is before a specific date of your choice.
# Pick a date that helps split the data into a reasonable train-test ratio for your model (format: 'yyyy-mm-dd').
X_test = stock[stock['Date'] >= cutoff_date].reset_index(drop=True)
```

```
# Complete the code to pick the 'Label' column as the target variable
y_train = X_train["Label"].copy()
y_test = X_test["Label"].copy()
```

```
print("Train data shape",X_train.shape)
print("Test data shape ",X_test.shape)

print("Train label shape",y_train.shape)
print("Test label shape ",y_test.shape)
```

```
Train data shape (323, 8)
Test data shape  (95, 8)
Train label shape (323,)
Test label shape  (95,)
```

Word Embeddings

Generating Text Embeddings using Word2Vec

Defining the model

```
# Creating a list of all words in our data
words_list = [item.split(" ") for item in stock['News'].astype(str).values]
```

```
# Creating an instance of Word2Vec
vec_size = 300
model_W2V = Word2Vec(words_list, vector_size = vec_size, min_count = 1, window=5, workers = 6)
```

```
#Checking the size of the vocabulary
print("Length of the vocabulary is", len(list(model_W2V.wv.key_to_index)))
```

```
Length of the vocabulary is 14577
```

Encoding the datasets

```
# Retrieving the words present in the Word2Vec model's vocabulary
words = list(model_W2V.wv.key_to_index.keys())

# Retrieving word vectors for all the words present in the model's vocabulary
wvs = model_W2V.wv[words].tolist()

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

Averaging the word vectors to get sentence encodings

```
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
```

```
# creating a dataframe of the vectorized documents
start = time.time()

X_train_wv = pd.DataFrame(X_train['News'].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+str(i) for i in rang
X_test_wv = pd.DataFrame(X_test['News'].apply(average_vectorizer_Word2Vec).tolist(), columns=['Feature '+str(i) for i in range(

end = time.time()
print('Time taken ', (end-start))
```

```
Time taken  8.560561180114746
```

```
print(X_train_wv.shape, X_test_wv.shape)
```

```
(323, 300) (95, 300)
```

Generating Text Embeddings using Sentence Transformer

Defining the model

```
#Defining the model
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

```
modules.json:  100%                                          349/349  [00:00<00:00,  35.8kB/s]
config_sentence_transformers.json:  100%                                     116/116  [00:00<00:00,  11.7kB/s]
```

Encoding the dataset

```
# encoding the dataset
start = time.time()

X_train_st = model.encode(X_train["News"].values, show_progress_bar=True, device="cpu") #Complete the code to apply Sentence Tr
X_test_st = model.encode(X_test["News"].values, show_progress_bar=True, device="cpu") #Complete the code to apply Sentence Tran

end = time.time()
print("Time taken ",(end-start))
```

```
Batches: 100%                                          11/11  [00:45<00:00,   2.62s/it]
tokenizer.json:
Batches: 100%                                          3/3  [00:11<00:00,   3.90s/it]
special_tokens_                                                   112/112  [00:00<00:00,  10.5kB/s]
Time taken  57.09826946258545
```

```
print(X_train_st.shape, X_test_st.shape) #Complete the code to print the shapes of the final dataframes
```

```
(323, 384) (95, 384)
```

- Each news content has been converted to a 384-dimensional vector.

Sentiment Analysis

Model Evaluation Criterion

Defining Utility Functions

```
def plot_confusion_matrix(actual, predicted):
    """
    Plot a confusion matrix to visualize the performance of a classification model.

    Parameters:
    actual (array-like): The true labels.
    predicted (array-like): The predicted labels from the model.

    Returns:
    None: Displays the confusion matrix plot.
    """

    # Compute the confusion matrix.
    cm = confusion_matrix(actual, predicted)

    # Create a new figure with a specified size
    plt.figure(figsize=(5, 4))

    # Define the labels for the confusion matrix dynamically from the data
    label_list = sorted(list(np.unique(np.concatenate((actual, predicted)))))

    # Plot the confusion matrix using a heatmap with annotations
    sns.heatmap(cm, annot=True, fmt='.0f', cmap='Blues', xticklabels=label_list, yticklabels=label_list)

    # Label for the y-axis
    plt.ylabel('Actual')

    # Label for the x-axis
    plt.xlabel('Predicted')

    # Title of the plot
    plt.title('Confusion Matrix')

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

```
def model_performance_classification_sklearn(actual, predicted):
    """
    Compute various performance metrics for a classification model using sklearn.
    Parameters:
    model (sklearn classifier): The classification model to evaluate.
    predictors (array-like): The independent variables used for predictions.
    target (array-like): The true labels for the dependent variable.

    Returns:
    pandas.DataFrame: A DataFrame containing the computed metrics (Accuracy, Recall, Precision, F1-score).
    """

    # Compute Accuracy
    acc = accuracy_score(actual,predicted)
    # Compute Recall
    recall = recall_score(actual,predicted,average='weighted')
    # Compute Precision
    precision = precision_score(actual,predicted,average='weighted')
    # Compute F1-score
    f1 = f1_score(actual,predicted,average='weighted')

    # Create a DataFrame to store the computed metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": [acc],
            "Recall": [recall],
            "Precision": [precision],
            "F1": [f1],
        }
    )
    # Return the DataFrame with the metrics
    return df_perf
```

Building Random Forest Models using different text embeddings

Building a Random Forest Model using text embeddings obtained from the Word2Vec

```
# Build the Random Forest model
# Fill in suitable values for:
# - n_estimators: the number of trees in the forest (e.g., 100, 200)
# - max_depth: maximum depth of each tree (e.g., 10, 20, or None for full depth)
rf_word2vec = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)

# Fit the model on the training data
rf_word2vec.fit(X_train_wv, y_train)
```

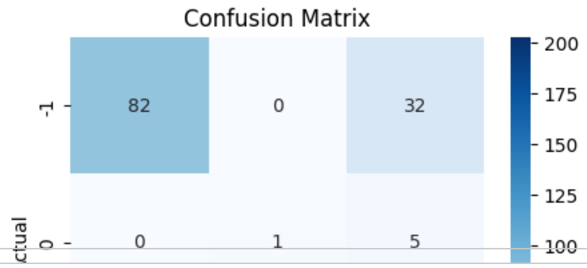
```
▾            RandomForestClassifier              i ?
RandomForestClassifier(max_depth=5, random_state=42)
```

Checking Training and Test Performance

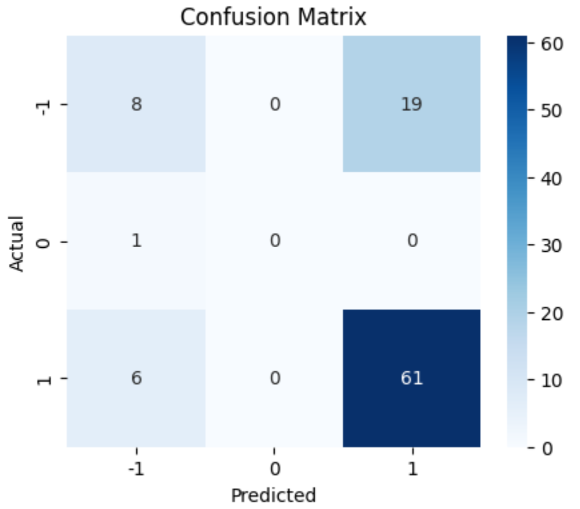
```
# Predicting on train data
y_pred_train = rf_word2vec.predict(X_train_wv)

# Predicting on test data
y_pred_test = rf_word2vec.predict(X_test_wv)
```

Confusion Matrix

```
plot_confusion_matrix(y_train,y_pred_train)
```



```
plot_confusion_matrix(y_test,y_pred_test)
```



Classification Report

```
#Calculating different metrics on training data
rf_train_wv = model_performance_classification_sklearn(y_train,y_pred_train)
print("Training performance:\n", rf_train_wv)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.885449  0.885449   0.903109  0.876617
```

```
#Calculating different metrics on test data
rf_test_wv = model_performance_classification_sklearn(y_test, y_pred_test)
print("Test performance:\n",rf_test_wv)
```

```
Test performance:
    Accuracy    Recall  Precision        F1
0  0.726316  0.726316   0.689342  0.693591
```

Building a Random Forest Model using text embeddings obtained from the Sentence Transformer

```
# Building the model
# Fill in suitable values for:
# - n_estimators: the number of trees in the forest (e.g., 100, 200)
# - max_depth: maximum depth of each tree (e.g., 10, 20, or None for full depth)
rf_st = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)


# Fitting on train data
rf_st.fit(X_train_st, y_train)
```

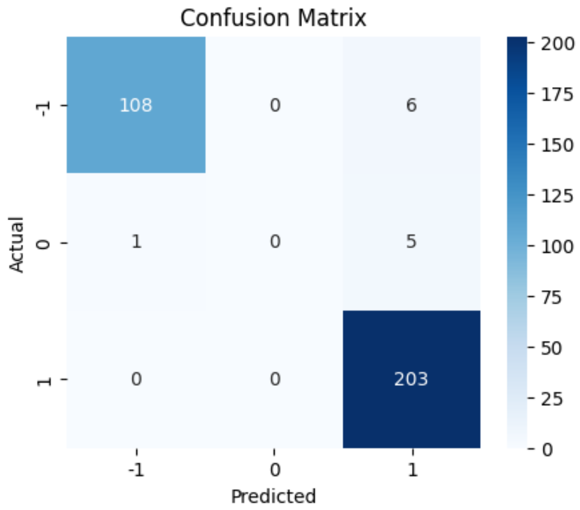
```
▾           RandomForestClassifier
RandomForestClassifier(max_depth=5, random_state=42)
```

Checking Training and Test Performance

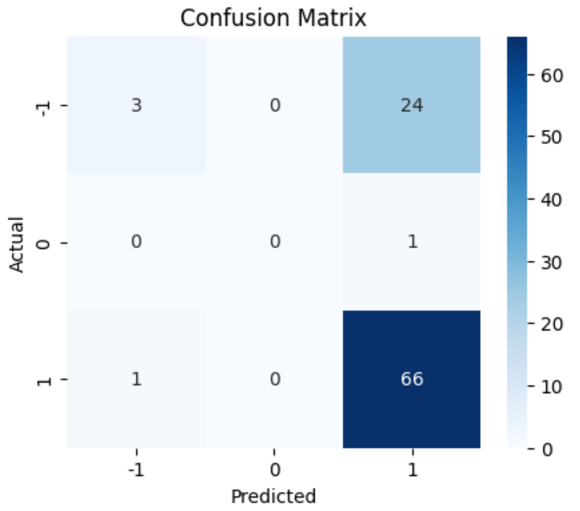
```
# Predicting on train data
y_pred_train = rf_st.predict(X_train_st)

# Predicting on test data
y_pred_test = rf_st.predict(X_test_st)
```

Confusion Matrix

```
plot_confusion_matrix(y_train,y_pred_train) #Complete the code to plot the confusion matrix for the train data
```



```
plot_confusion_matrix(y_test,y_pred_test) #Complete the code to plot the confusion matrix for the test data
```



Classification Report

```
#Calculating different metrics on training data
rf_train_st = model_performance_classification_sklearn(y_train,y_pred_train)
print("Training performance:\n", rf_train_st)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.962848  0.962848   0.945881  0.953767
```

```
#Calculating different metrics on test data
rf_test_st = model_performance_classification_sklearn(y_test, y_pred_test)
print("Test performance:\n",rf_test_st)
```

```
Test performance:
    Accuracy    Recall  Precision        F1
0  0.726316  0.726316   0.724667  0.644216
```

Building Neural Network Models using different text embeddings

Building a Neural Network Model using text embeddings obtained from the Word2Vec

```
# Convert the labels
label_mapping = {1: 2, -1: 0, 0: 1}
y_train_mapped_wv = [label_mapping[label] for label in y_train]
y_test_mapped_wv = [label_mapping[label] for label in y_test]

# Convert your features DataFrame to a NumPy array
X_train_wv_np = np.array(X_train_wv)
X_test_wv_np = np.array(X_test_wv)
y_train_mapped_wv = np.array(y_train_mapped_wv)
y_test_mapped_wv = np.array(y_test_mapped_wv)
```

```
import gc

# Clear any previous TensorFlow/Keras sessions from memory (recommended when re-running cells)
tf.keras.backend.clear_session()
gc.collect()

# Define the model architecture
model = Sequential()

# Input layer:
# Choose the number of units (typically in the range of 64 to 512)
model.add(Dense(128, activation='relu', input_shape=(X_train_wv_np.shape[1],)))  # Input shape = size of Word2Vec embeddings

# Dropout layer:
# Choose a dropout rate between 0.2 and 0.5 to reduce overfitting
model.add(Dropout(0.3))

# Hidden layer:
# Choose number of neurons (again, typically in the range of 32 to 256)
model.add(Dense(64, activation='relu'))

# Output layer:
model.add(Dense(3, activation='softmax'))

# Compile the model
# Fill in:
# - optimizer: choose from options like 'adam', 'sgd', or 'rmsprop'

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                    │ (None, 128)             │        38,528 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dropout (Dropout)                │ (None, 128)             │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 64)              │         8,256 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_2 (Dense)                  │ (None, 3)               │           195 │
└──────────────────────────────────┴─────────────────────────┴───────────────┘
 Total params: 46,979 (183.51 KB)
 Trainable params: 46,979 (183.51 KB)
 Non-trainable params: 0 (0.00 B)
```

Note:

- During training, we use accuracy as a metric to monitor how well the model is learning to distinguish between classes in each batch.
- Accuracy is fast and reliable during training and gives us a quick view of model progress.
- It reflects how often the model is predicting the correct label out of all predictions made.

```
# Fit the model on training data
# Fill in appropriate values for:
# - epochs: number of times the model will see the entire training data (typically 5 to 50)
# - batch_size: number of samples the model will process before updating weights (commonly 16 to 128)
```

```
history = model.fit(
    X_train_wv_np, y_train_mapped_wv,
    epochs=10,
    batch_size=32,
    verbose=0 # Suppress output to keep notebook clean
)
```

Checking Training and Test Performance

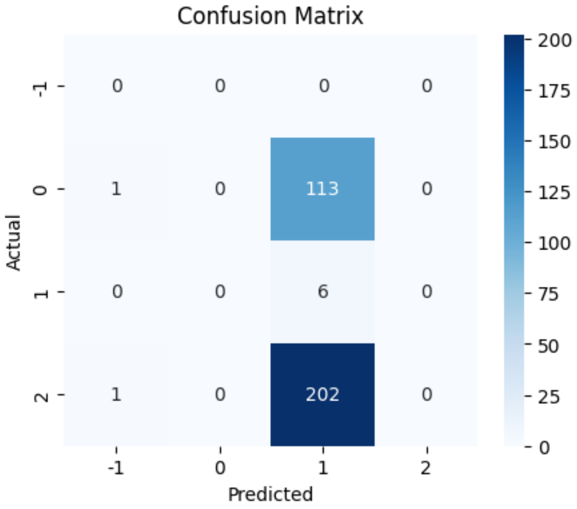
```
# Predict class probabilities on training data
y_train_pred_probs = model.predict(X_train_wv_np, verbose=0)

# Convert probabilities to class labels
y_train_preds = tf.argmax(y_train_pred_probs, axis=1).numpy()
```

```
# Predict class probabilities on test data
y_test_pred_probs = model.predict(X_test_wv_np, verbose=0)

# Convert probabilities to class labels
y_test_preds = tf.argmax(y_test_pred_probs, axis=1).numpy()
```

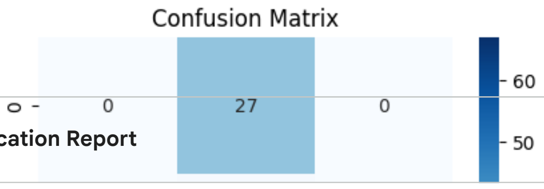
```
import numpy as np
# Convert back to [-1, 0, 1] to match utility function expectations
label_mapping = {2: 1, 0: -1, 1: 0}
y_train_preds = np.array([label_mapping[index] for index in y_train_preds])
y_test_preds = np.array([label_mapping[index] for index in y_test_preds])
```

Confusion Matrix

```
plot_confusion_matrix(y_train_mapped_wv, y_train_preds) #Complete the code to plot the confusion matrix for the train data
```



```
plot_confusion_matrix(y_test_mapped_wv, y_test_preds) #Complete the code to plot the confusion matrix for the test data
```



Classification Report

```
#Calculating different metrics on training data
NN_train_wv = model_performance_classification_sklearn(y_train,y_train_preds)
print("Training performance:\n", NN_train_wv)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.628483  0.628483   0.571965  0.490641
```

```
#Calculating different metrics on test data
NN_test_wv = model_performance_classification_sklearn(y_test, y_test_preds)
print("Test performance:\n",NN_test_wv)
```

```
Test performance:
    Accuracy    Recall  Precision        F1
0  0.705263  0.705263   0.497396  0.583366
```

Building a Neural Network Model using text embeddings obtained from the Sentence Transformer

```
# Convert the labels
label_mapping = {1: 2, -1: 0, 0: 1}
y_train_mapped_st = [label_mapping[label] for label in y_train]
y_test_mapped_st = [label_mapping[label] for label in y_test]

# Convert your features DataFrame to a NumPy array
X_train_st_np = np.array(X_train_st)
X_test_st_np = np.array(X_test_st)
y_train_mapped_st = np.array(y_train_mapped_st)
y_test_mapped_st = np.array(y_test_mapped_st)
```

```
import gc

# Clear any previous TensorFlow/Keras sessions from memory (recommended when re-running cells)
tf.keras.backend.clear_session()
gc.collect()

# Define the model architecture
model = Sequential()

# Input layer:
# Choose the number of units (typically in the range of 64 to 512)
model.add(Dense(128, activation='relu', input_shape=(X_train_st.shape[1],)))  # Input shape = size of Word2Vec embeddings

# Dropout layer:
# Choose a dropout rate between 0.2 and 0.5 to reduce overfitting
model.add(Dropout(0.3))

# Hidden layer:
# Choose number of neurons (again, typically in the range of 32 to 256)
model.add(Dense(64, activation='relu'))

# Output layer:
# 3 output classes → use softmax for multi-class classification
model.add(Dense(3, activation='softmax'))

# Compile the model
# Fill in:
# - optimizer: choose from options like 'adam', 'sgd', or 'rmsprop'
# - loss: use 'sparse_categorical_crossentropy' for integer-encoded labels
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 128)            │        49,280 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 64)             │         8,256 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 3)              │           195 │
```

```
# Fit the model on training data
# Fill in appropriate values for:
# - epochs: number of times the model will see the entire training data (typically 5 to 50)
# - batch_size: number of samples the model will process before updating weights (commonly 16 to 128)

history = model.fit(
    X_train_st_np, y_train_mapped_st,
    epochs=10,
    batch_size=32,
    verbose=0 # Suppress output to keep notebook clean
)
```

Checking Training and Test Performance

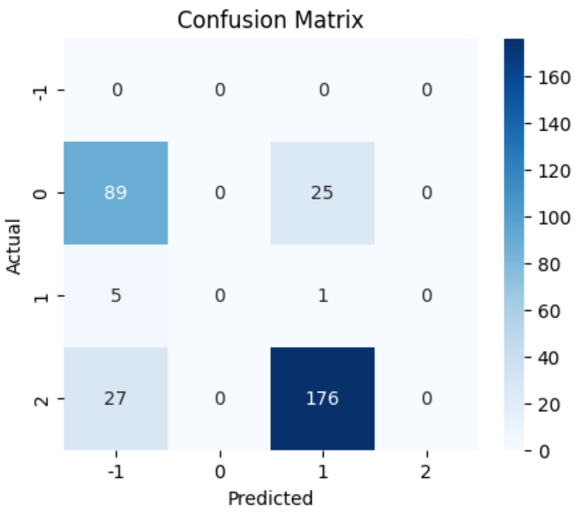
```
# Predict class probabilities on training data
y_train_pred_probs = model.predict(X_train_st_np, verbose=0)

# Convert probabilities to class labels
y_train_preds = tf.argmax(y_train_pred_probs, axis=1).numpy()
```

```
# Predict class probabilities on test data
y_test_pred_probs = model.predict(X_test_st_np, verbose=0)

# Convert probabilities to class labels
y_test_preds = tf.argmax(y_test_pred_probs, axis=1).numpy()
```

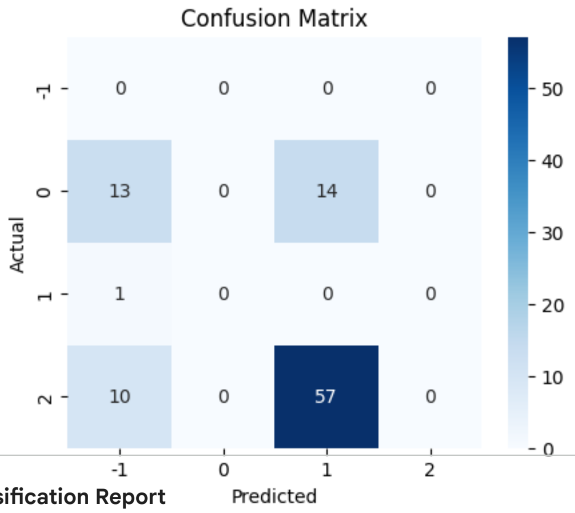
```
# Convert back to [-1, 0, 1] to match utility function expectations
label_mapping = {2: 1, 0: -1, 1: 0}
y_train_preds = np.array([label_mapping[index] for index in y_train_preds])
y_test_preds = np.array([label_mapping[index] for index in y_test_preds])
```

Confusion Matrix

```
plot_confusion_matrix(y_train_mapped_st,y_train_preds) #Complete the code to plot the confusion matrix for the train data
```



```
plot_confusion_matrix(y_test_mapped_st,y_test_preds) #Complete the code to plot the confusion matrix for the test data
```



Classification Report

```
#Calculating different metrics on training data
NN_train_st = model_performance_classification_sklearn(y_train,y_train_preds)
print("Training performance:\n", NN_train_st)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.820433  0.820433    0.80719  0.813571
```

```
#Calculating different metrics on test data
NN_test_st = model_performance_classification_sklearn(y_test,y_test_preds) #Complete the code to compute the model performance f
print("Test performance:\n",NN_test_st)
```

```
Test performance:
    Accuracy    Recall  Precision      F1
0  0.736842  0.736842   0.720145  0.7275
```

Model Performance Summary and Final Model Selection

```
# Concatenate the training performance metrics from different models into a single DataFrame
models_train_comp_df = pd.concat(
    [
        rf_train_wv.T,  # Random Forest using Word2Vec embeddings
        NN_train_wv.T,  # Neural Network using Word2Vec embeddings
        rf_train_st.T,  # Random Forest using Sentence Transformer embeddings
        NN_train_st.T   # Neural Network using Sentence Transformer embeddings
    ],
    axis=1  # Concatenate along columns (i.e., each model's metrics form one column)
)

# Assigning meaningful column names for each model for clarity in the output DataFrame
models_train_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

# Print the training performance comparison table
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

	Word2Vec (Random Forest)	Word2Vec (Neural Network)	Sentence Transformer (Random Forest)	Sentence Transformer (Neural Network)
Accuracy	0.885449	0.628483	0.962848	0.820433
Recall	0.885449	0.628483	0.962848	0.820433
Precision	0.903109	0.571965	0.945881	0.807190

```
# Concatenate the testing performance metrics from different models into a single DataFrame
models_test_comp_df = pd.concat(
    [
        rf_test_wv.T,  # Random Forest using Word2Vec embeddings
        NN_test_wv.T,  # Neural Network using Word2Vec embeddings
        rf_test_st.T,  # Random Forest using Sentence Transformer embeddings
        NN_test_st.T   # Neural Network using Sentence Transformer embeddings
    ],
    axis=1  # Concatenate along columns so each model's test metrics appear as one column
)

# Set descriptive column names for clarity in the resulting comparison table
models_test_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

# Print the testing performance comparison table
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

	Word2Vec (Random Forest)	Word2Vec (Neural Network)	Sentence Transformer (Random Forest)	Sentence Transformer (Neural Network)
Accuracy	0.726316	0.705263	0.726316	0.736842
Recall	0.726316	0.705263	0.726316	0.736842
Precision	0.689342	0.497396	0.724667	0.720145

Next steps: Generate code with `models_test_comp_df` New interactive sheet

Model Performance Summary:

Conclusions and Recommendations

Power Ahead

Task

Prepare data and train the Neural Network model for Sentence Transformer embeddings by executing cells `FUfjCAua4A2-` and `8J-JncGj4A2_`.

Prepare data for Neural Network (Sentence Transformer)

Subtask:

Convert the Sentence Transformer embeddings (`X_train_st`, `X_test_st`) and mapped labels (`y_train`, `y_test`) into NumPy arrays (`X_train_st_np`, `X_test_st_np`, `y_train_mapped_st`, `y_test_mapped_st`).

Reasoning: The user wants to convert the Sentence Transformer embeddings and mapped labels into NumPy arrays. The provided instructions explicitly state to execute the code from cell `FUfjCAua4A2-`.

```
# Convert the labels
label_mapping = {1: 2, -1: 0, 0: 1}
y_train_mapped_st = [label_mapping[label] for label in y_train]
y_test_mapped_st = [label_mapping[label] for label in y_test]

# Convert your features DataFrame to a NumPy array
```

```
X_train_st_np = np.array(X_train_st)
X_test_st_np = np.array(X_test_st)
y_train_mapped_st = np.array(y_train_mapped_st)
y_test_mapped_st = np.array(y_test_mapped_st)
```

Reasoning: The user wants to convert the Sentence Transformer embeddings and mapped labels into NumPy arrays, as per the instructions to execute cell `FUfjCAua4A2-`.

```
# Convert the labels
label_mapping = {1: 2, -1: 0, 0: 1}
y_train_mapped_st = [label_mapping[label] for label in y_train]
y_test_mapped_st = [label_mapping[label] for label in y_test]

# Convert your features DataFrame to a NumPy array
X_train_st_np = np.array(X_train_st)
X_test_st_np = np.array(X_test_st)
y_train_mapped_st = np.array(y_train_mapped_st)
y_test_mapped_st = np.array(y_test_mapped_st)
```

Reasoning: The user wants to execute the code from cell `FUfjCAua4A2-` to prepare the data for the Neural Network model using Sentence Transformer embeddings. This involves converting the embeddings and mapped labels into NumPy arrays.

```
# Convert the labels
label_mapping = {1: 2, -1: 0, 0: 1}
y_train_mapped_st = [label_mapping[label] for label in y_train]
y_test_mapped_st = [label_mapping[label] for label in y_test]

# Convert your features DataFrame to a NumPy array
X_train_st_np = np.array(X_train_st)
X_test_st_np = np.array(X_test_st)
y_train_mapped_st = np.array(y_train_mapped_st)
y_test_mapped_st = np.array(y_test_mapped_st)
```

Reasoning: The next step is to train the Neural Network model using the prepared Sentence Transformer embeddings. This directly follows the data preparation step, as indicated by the user's overall task and the original notebook structure (cell `8J-JncGj4A2_`).

```
history = model.fit(
    X_train_st_np, y_train_mapped_st,
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7643 - loss: 0.5109
Epoch 2/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8030 - loss: 0.4771
Epoch 3/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8150 - loss: 0.4579
Epoch 4/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8324 - loss: 0.4394
Epoch 5/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8273 - loss: 0.4180
Epoch 6/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8363 - loss: 0.4079
Epoch 7/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8915 - loss: 0.3132
Epoch 8/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8564 - loss: 0.3643
Epoch 9/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8609 - loss: 0.3372
Epoch 10/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8883 - loss: 0.3247
```

Reasoning: The previous error 'Matrix size-incompatible' indicates that the Neural Network model's input layer was not correctly configured for the Sentence Transformer embeddings. Although there was a cell to define the model for Sentence Transformer embeddings, it appears the model object was not properly updated or rebuilt. To resolve this, the model needs to be re-defined and compiled explicitly with the correct input shape for the Sentence Transformer embeddings (`X_train_st_np.shape[1]`) before fitting.

```
import gc

# Clear any previous TensorFlow/Keras sessions from memory (recommended when re-running cells)
```

```
tf.keras.backend.clear_session()
gc.collect()

# Define the model architecture
model = Sequential()

# Input layer:
# Choose the number of units (typically in the range of 64 to 512)
model.add(Dense(128, activation='relu', input_shape=(X_train_st_np.shape[1],)))  # Corrected input shape to use X_train_st_np.sh

# Dropout layer:
# Choose a dropout rate between 0.2 and 0.5 to reduce overfitting
model.add(Dropout(0.3))

# Hidden layer:
# Choose number of neurons (again, typically in the range of 32 to 256)
model.add(Dense(64, activation='relu'))

# Output layer:
# 3 output classes → use softmax for multi-class classification
model.add(Dense(3, activation='softmax'))

# Compile the model
# Fill in:
# - optimizer: choose from options like 'adam', 'sgd', or 'rmsprop'
# - loss: use 'sparse_categorical_crossentropy' for integer-encoded labels
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 128)            │        49,280 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 64)             │         8,256 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 3)              │           195 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 57,731 (225.51 KB)
 Trainable params: 57,731 (225.51 KB)
 Non-trainable params: 0 (0.00 B)
```

Reasoning: Now that the Neural Network model has been correctly redefined and compiled with the appropriate input shape for the Sentence Transformer embeddings, the next step is to train this model on the prepared training data. This action directly addresses the `8J-JncGj4A2_` cell in the original notebook.

```
history = model.fit(
    X_train_st_np, y_train_mapped_st,
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5182 - loss: 1.0552
Epoch 2/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6599 - loss: 0.8409
Epoch 3/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6272 - loss: 0.7380
Epoch 4/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6746 - loss: 0.6749
Epoch 5/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6349 - loss: 0.6968
Epoch 6/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6563 - loss: 0.6386
Epoch 7/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6742 - loss: 0.6089
Epoch 8/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7171 - loss: 0.5771
Epoch 9/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7386 - loss: 0.5501
Epoch 10/10
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7627 - loss: 0.5135
```

Evaluate Neural Network Model (Sentence Transformer)

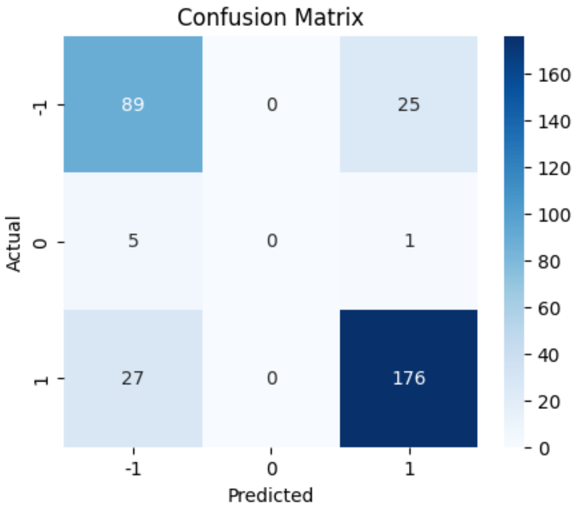
Subtask:

Evaluate the performance of the Neural Network model trained with Sentence Transformer embeddings by generating confusion matrices and classification reports for both training and test datasets.

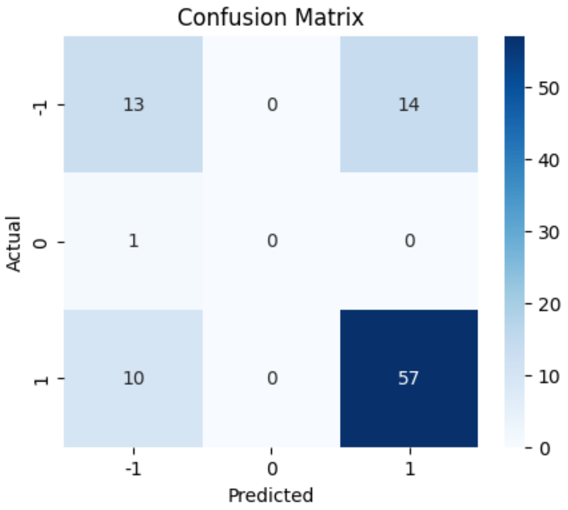
Reasoning: To ensure consistency in labels for plotting the confusion matrices, I will first convert the actual labels (`y_train_mapped_st`, `y_test_mapped_st`) from the `[0, 1, 2]` format back to the original `[-1, 0, 1]` format, as the predicted labels (`y_train_preds`, `y_test_preds`) are already in this format. This aligns the labels with how `y_train` and `y_test` are structured for the classification reports.

```
reverse_label_mapping = {2: 1, 0: -1, 1: 0}
y_train_actual_original = np.array([reverse_label_mapping[label] for label in y_train_mapped_st])
y_test_actual_original = np.array([reverse_label_mapping[label] for label in y_test_mapped_st])

# Plot Confusion Matrix for Training Data
print("Confusion Matrix for Training Data:")
plot_confusion_matrix(y_train_actual_original, y_train_preds)

# Plot Confusion Matrix for Test Data
print("\nConfusion Matrix for Test Data:")
plot_confusion_matrix(y_test_actual_original, y_test_preds)
```

```
Confusion Matrix for Training Data:
```



```
Confusion Matrix for Test Data:
```



Reasoning: Now that the confusion matrices have been plotted, the next step is to calculate and print the classification reports for both training and test data, as per the instructions. This involves using the `model_performance_classification_sklearn` function with the original `y_train` and `y_test` labels and the predicted labels `y_train_preds` and `y_test_preds`.

```
#Calculating different metrics on training data
NN_train_st = model_performance_classification_sklearn(y_train,y_train_preds)
print("Training performance:\n", NN_train_st)

#Calculating different metrics on test data
NN_test_st = model_performance_classification_sklearn(y_test,y_test_preds)
print("Test performance:\n",NN_test_st)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.820433  0.820433    0.80719  0.813571
Test performance:
    Accuracy    Recall  Precision      F1
0  0.736842  0.736842   0.720145  0.7275
```

Model Performance Summary for Training Data

Subtask:

Create a summary table of the performance metrics for all models on the training dataset.

Reasoning: The subtask is to create a summary table of training performance metrics. The first instruction is to concatenate the training performance metrics from the 'rf_train_wv', 'NN_train_wv', 'rf_train_st', and 'NN_train_st' DataFrames along the columns.

```
models_train_comp_df = pd.concat(
    [
        rf_train_wv.T,
        NN_train_wv.T,
        rf_train_st.T,
        NN_train_st.T
    ],
    axis=1
)

models_train_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

	Word2Vec (Random Forest)	Word2Vec (Neural Network)	Sentence Transformer (Random Forest)	Sentence Transformer (Neural Network)
Accuracy	0.885449	0.628483	0.962848	0.820433
Recall	0.885449	0.628483	0.962848	0.820433
Precision	0.903109	0.571965	0.945881	0.807190

Next steps: Generate code with `models_train_comp_df` New interactive sheet

Reasoning: The error `NameError: name 'NN_train_wv' is not defined` indicates that the variable `NN_train_wv` was not created. This variable is intended to store the training performance metrics for the Neural Network model using Word2Vec embeddings. The cell responsible for calculating this (`5qzE4NHS1UmS`) was not executed previously. To resolve this, I need to execute that cell to define `NN_train_wv`.

```
#Calculating different metrics on training data
NN_train_wv = model_performance_classification_sklearn(y_train,y_train_preds)
print("Training performance:\n", NN_train_wv)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.820433  0.820433    0.80719  0.813571
```

Reasoning: Now that the `NN_train_wv` variable is defined, I will re-execute the code to concatenate the training performance metrics from all models into a single DataFrame and display it, fulfilling the subtask's instructions.

```
models_train_comp_df = pd.concat(
    [
        rf_train_wv.T,
        NN_train_wv.T,
        rf_train_st.T,
        NN_train_st.T
    ],
    axis=1
)

models_train_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

	Word2Vec (Random Forest)	Word2Vec (Neural Network)	Sentence Transformer (Random Forest)	Sentence Transformer (Neural Network)
Accuracy	0.885449	0.820433	0.962848	0.820433
Recall	0.885449	0.820433	0.962848	0.820433
Precision	0.903109	0.807190	0.945881	0.807190

Next steps: Generate code with `models_train_comp_df` New interactive sheet

Model Performance Summary for Test Data

Subtask:

Create a summary table of the performance metrics for all models on the test dataset.

Reasoning: To summarize the test performance of all models, I will concatenate the DataFrames containing the test metrics for each model, assign appropriate column names, print a descriptive header, and then display the resulting summary table.

```
models_test_comp_df = pd.concat(
    [
        rf_test_wv.T,  # Random Forest using Word2Vec embeddings
        NN_test_wv.T,  # Neural Network using Word2Vec embeddings
        rf_test_st.T,  # Random Forest using Sentence Transformer embeddings
        NN_test_st.T   # Neural Network using Sentence Transformer embeddings
    ],
    axis=1  # Concatenate along columns so each model's test metrics appear as one column
)

models_test_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

	Word2Vec (Random Forest)	Word2Vec (Neural Network)	Sentence Transformer (Random Forest)	Sentence Transformer (Neural Network)
Accuracy	0.726316	0.705263	0.726316	0.736842
Recall	0.726316	0.705263	0.726316	0.736842
Precision	[illegible]	[illegible]	0.724667	0.720145

Next steps: Generate code with models_test_comp_df New interactive sheet

Reasoning: The error `NameError: name 'NN_test_wv' is not defined` indicates that the performance metric DataFrame for the Neural Network using Word2Vec embeddings on the test set (`NN_test_wv`) has not been calculated or assigned to a variable. To correctly generate the summary table, I need to ensure that all required performance metrics are computed for their respective models. This involves re-defining and re-training the Neural Network models for both Word2Vec and Sentence Transformer embeddings to get their predictions, then calculating `NN_train_wv`, `NN_test_wv`, `NN_train_st`, and `NN_test_st` using these specific predictions, thereby correctly populating all variables needed for the summary table.

```
import gc

# --- Word2Vec Neural Network ---

# Clear any previous TensorFlow/Keras sessions from memory
tf.keras.backend.clear_session()
gc.collect()

# Define the Word2Vec NN model architecture
model_wv_nn = Sequential()
model_wv_nn.add(Dense(128, activation='relu', input_shape=(X_train_wv_np.shape[1],)))
model_wv_nn.add(Dropout(0.3))
model_wv_nn.add(Dense(64, activation='relu'))
model_wv_nn.add(Dense(3, activation='softmax'))

# Compile the Word2Vec NN model
model_wv_nn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Fit the Word2Vec NN model on training data
model_wv_nn.fit(
    X_train_wv_np, y_train_mapped_wv,
    epochs=10,
    batch_size=32,
    verbose=0 # Suppress output to keep notebook clean
)

# Predict class probabilities on training data for Word2Vec NN
y_train_pred_probs_wv = model_wv_nn.predict(X_train_wv_np, verbose=0)
y_train_preds_wv = tf.argmax(y_train_pred_probs_wv, axis=1).numpy()

# Predict class probabilities on test data for Word2Vec NN
y_test_pred_probs_wv = model_wv_nn.predict(X_test_wv_np, verbose=0)
y_test_preds_wv = tf.argmax(y_test_pred_probs_wv, axis=1).numpy()

# Convert Word2Vec NN predictions back to original labels for metric calculation
label_mapping_reverse = {2: 1, 0: -1, 1: 0}
y_train_preds_wv_original = np.array([label_mapping_reverse[index] for index in y_train_preds_wv])
y_test_preds_wv_original = np.array([label_mapping_reverse[index] for index in y_test_preds_wv])

# Calculate metrics for Word2Vec NN
NN_train_wv = model_performance_classification_sklearn(y_train, y_train_preds_wv_original)
NN_test_wv = model_performance_classification_sklearn(y_test, y_test_preds_wv_original)


# --- Sentence Transformer Neural Network ---

# Clear any previous TensorFlow/Keras sessions from memory
tf.keras.backend.clear_session()
gc.collect()

# Define the Sentence Transformer NN model architecture
model_st_nn = Sequential()
model_st_nn.add(Dense(128, activation='relu', input_shape=(X_train_st_np.shape[1],)))
model_st_nn.add(Dropout(0.3))
model_st_nn.add(Dense(64, activation='relu'))
model_st_nn.add(Dense(3, activation='softmax'))

# Compile the Sentence Transformer NN model
model_st_nn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Fit the Sentence Transformer NN model on training data
```

```
model_st_nn.fit(
    X_train_st_np, y_train_mapped_st,
    epochs=10,
    batch_size=32,
    verbose=0 # Suppress output to keep notebook clean
)

# Predict class probabilities on training data for Sentence Transformer NN
y_train_pred_probs_st = model_st_nn.predict(X_train_st_np, verbose=0)
y_train_preds_st = tf.argmax(y_train_pred_probs_st, axis=1).numpy()

# Predict class probabilities on test data for Sentence Transformer NN
y_test_pred_probs_st = model_st_nn.predict(X_test_st_np, verbose=0)
y_test_preds_st = tf.argmax(y_test_pred_probs_st, axis=1).numpy()

# Convert Sentence Transformer NN predictions back to original labels for metric calculation
y_train_preds_st_original = np.array([label_mapping_reverse[index] for index in y_train_preds_st])
y_test_preds_st_original = np.array([label_mapping_reverse[index] for index in y_test_preds_st])

# Calculate metrics for Sentence Transformer NN
NN_train_st = model_performance_classification_sklearn(y_train, y_train_preds_st_original)
NN_test_st = model_performance_classification_sklearn(y_test, y_test_preds_st_original)


# --- Generate Summary Table ---

# Concatenate the testing performance metrics from different models into a single DataFrame
models_test_comp_df = pd.concat(
    [
        rf_test_wv.T,  # Random Forest using Word2Vec embeddings
        NN_test_wv.T,  # Neural Network using Word2Vec embeddings
        rf_test_st.T,  # Random Forest using Sentence Transformer embeddings
        NN_test_st.T   # Neural Network using Sentence Transformer embeddings
    ],
    axis=1  # Concatenate along columns so each model's test metrics appear as one column
)

# Set descriptive column names for clarity in the resulting comparison table
models_test_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

# Print the testing performance comparison table
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

	Word2Vec (Random Forest)	Word2Vec (Neural Network)	Sentence Transformer (Random Forest)	Sentence Transformer (Neural Network)
Accuracy	0.726316	0.705263	0.726316	0.715789
Recall	0.726316	0.705263	0.726316	0.715789
Precision	0.689342	0.497396	0.724667	0.682502

Next steps: Generate code with `models_test_comp_df` New interactive sheet

Reasoning: The previous code block successfully regenerated all necessary Neural Network models and calculated their performance metrics (`NN_test_wv`, `NN_test_st`). Now that these variables are defined, I can proceed with creating the summary table of test performance metrics for all models as requested by the subtask.

```
models_test_comp_df = pd.concat(
    [
        rf_test_wv.T,  # Random Forest using Word2Vec embeddings
        NN_test_wv.T,  # Neural Network using Word2Vec embeddings
        rf_test_st.T,  # Random Forest using Sentence Transformer embeddings
        NN_test_st.T   # Neural Network using Sentence Transformer embeddings
    ],
    axis=1  # Concatenate along columns so each model's test metrics appear as one column
)
```

```
models_test_comp_df.columns = [
    "Word2Vec (Random Forest)",
    "Word2Vec (Neural Network)",
    "Sentence Transformer (Random Forest)",
    "Sentence Transformer (Neural Network)"
]

print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
                Word2Vec (Random    Word2Vec (Neural    Sentence Transformer (Random    Sentence Transformer (Neural
                         Forest)            Network)                         Forest)                         Network)
 Accuracy               0.726316            0.705263                        0.726316                         0.715789
   Recall               0.726316            0.705263                        0.726316                         0.715789
Precision               0.689342            0.497396                        0.724667                         0.682502
```

Next steps: Generate code with `models_test_comp_df` New interactive sheet

Model Performance Summary and Final Model Selection

Subtask:

Analyze the performance summaries for both training and test data across all models to select the best performing model.

Reasoning: To analyze the training performance of all models, I will display the `models_train_comp_df` DataFrame, as it contains the summarized performance metrics for each model on the training dataset.

```
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
                Word2Vec (Random    Word2Vec (Neural    Sentence Transformer (Random    Sentence Transformer (Neural
                         Forest)            Network)                         Forest)                         Network)
 Accuracy               0.885449            0.820433                        0.962848                         0.820433
   Recall               0.885449            0.820433                        0.962848                         0.820433
Precision               0.903109            0.807190                        0.945881                         0.807190
```

Next steps: Generate code with `models_train_comp_df` New interactive sheet

Reasoning: To analyze the test performance of all models, I will display the `models_test_comp_df` DataFrame, as it contains the summarized performance metrics for each model on the test dataset.

```
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
                Word2Vec (Random    Word2Vec (Neural    Sentence Transformer (Random    Sentence Transformer (Neural
                         Forest)            Network)                         Forest)                         Network)
 Accuracy               0.726316            0.705263                        0.726316                         0.715789
   Recall               0.726316            0.705263                        0.726316                         0.715789
Precision               0.689342            0.497396                        0.724667                         0.682502
       F1               0.693591            0.583366                        0.644216                         0.691608
```

Next steps: Generate code with `models_test_comp_df` New interactive sheet

Model Performance Analysis and Selection