REPORT

STOCK SENTIMENT ANALYSIS PROJECT

USING MACHINE LEARNING (#FC24OPS3)

RAJAN VERMA

ENROLLMENT NUMBER: 23124028

OVERVIEW ABOUT THE PROJECT:

This project focuses on the application of sentiment analysis techniques to assess the sentiment of financial news and other relevant text data to predict stock price movements. By analysing the sentiments expressed in these sources, we aim to provide investors and financial analysts with valuable insights that can inform their trading strategies and decision-making processes.

SIGNIFICANCE

Understanding market sentiment is crucial for investors, traders, and financial analysts. Positive or negative sentiment can significantly influence stock prices, often causing rapid changes in market behaviour. By leveraging sentiment analysis, this project aims to:

- Improve Trading Strategies: Aid in the development of trading strategies that take into account sentiment trends, potentially leading to better investment outcomes.
- Mitigate Risks: Help investors identify potential risks by detecting negative sentiment early, allowing for timely adjustments to their portfolios.

METHODOLOGY:

1. DATA COLLECTION:

I have collected the data i.e. news headlines from the website seekingalpha.com and correspondingly made the csv file named as news.csv which contains the news headlines along with dates.

The historical data of stock prices was fetched from yfinance (a Python library that provides a way to download historical market data from Yahoo Finance. It allows users to fetch data such as historical stock prices, dividend information, and other financial data.)

2. DATA CLEANING AND PREPROCESSING:

I have merged the news headlines which have same date together before data cleaning and preprocessing.

```
# Function to preprocess text
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    return text

# Apply preprocessing
df2['cleaned_headline'] = df2['News Headlines'].apply(preprocess_text)
print(df2[['News Headlines', 'cleaned_headline']])
```

```
# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()
# Function to apply lemmatization
def lemmatize_text(text):
    tokens = text.split()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(lemmatized_tokens)

# Apply lemmatization
df2['lemmatized_headline'] = df2['cleaned_headline'].apply(lemmatize_text)
print(df2[['cleaned_headline', 'lemmatized_headline']])
```

I have used lemmatization for preprocessing. It involves reducing a word to its base or root form, known as a "lemma." This process helps in

normalizing the text, making it easier to analyse and ensuring that different forms of a word are treated as a single entity.

3. SENTIMENT ANALYSIS:

```
from textblob import TextBlob
def getSubjectivity(text):
  return TextBlob(text).sentiment.subjectivity
def getPolarity(text):
  return TextBlob(text).sentiment.polarity

df2_reset['Subjectivity']=df2_reset['lemmatized_headline'].apply(getSubjectivity)
df2_reset['Polarity']=df2_reset['lemmatized_headline'].apply(getPolarity)
df2_reset
```

I have used TextBlob which is a Python library for text analysis. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, sentiment analysis, classification, translation, and more.

TextBlob has a built-in sentiment analysis tool that can analyze the sentiment of a given text. It provides two main metrics: subjectivity and polarity.

- Polarity is a measure of the sentiment expressed in a text. It ranges from -1 to 1, where:
  - -1 indicates a very negative sentiment.
  - 0 indicates a neutral sentiment.
  - 1 indicates a very positive sentiment.
- Subjectivity quantifies the amount of personal opinion and factual information contained in the text. It ranges from 0 to 1, where:
  - 0 is very objective (fact-based).
  - 1 is very subjective (opinion-based).

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

def getSIA(text):
  sia=SentimentIntensityAnalyzer()
  sentiment=sia.polarity_scores(text)
  return sentiment
```

I have used SentimentIntensityAnalyzer, which is part of the VADER sentiment analysis tool available in the NLTK (Natural Language Toolkit) library. VADER is designed for analyzing sentiment in text. It is sensitive to both the polarity (positive/negative) and intensity (strength) of sentiment expressions.

- **Initialization**: The SentimentIntensityAnalyzer is initialized by creating an instance of the SentimentIntensityAnalyzer class.
- **Polarity Scores**: The polarity_scores method is used to obtain the sentiment scores for each text. This method returns a dictionary with four keys:

- **pos**: Positive sentiment score.(denotes the proportion of text that is positive)
- **neu**: Neutral sentiment score.
- **neg**: Negative sentiment score.
- **compound**: The overall sentiment score, which is a normalized score ranging from -1 (most extreme negative) to +1 (most extreme positive).

4. FEATURE EXTRACTION:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer



#Word Frequencies (TF-IDF)
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(merge['lemmatized_headline'])
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())

print("TF-IDF Features:")
print(tfidf_df)
```

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It is commonly used in information retrieval, text mining, and natural language processing to transform textual data into a numerical representation that can be used for various machine learning algorithms.

```
#Topic Modeling (LDA)
lda = LatentDirichletAllocation(n_components=2, random_state=42)
lda_matrix = lda.fit_transform(tfidf_df)
lda_df = pd.DataFrame(lda_matrix, columns=[f'Topic {i}' for i in range(1, lda.n_components + 1)])

print("LDA Topics:")
print(lda_df)
```

I have applied Topic modelling which is a technique in natural language processing (NLP) and text mining used to discover the abstract "topics" that occur in a collection of documents. One of the most popular methods for topic modeling is Latent Dirichlet Allocation (LDA).

LDA is a generative probabilistic model for collections of discrete data such as text corpora. It assumes that documents are mixtures of topics and that topics are mixtures of words. The goal of LDA is to identify the hidden topic structure in a large collection of documents.

5. DATA LABELING:

I have labelled the collected data with respect to the stock movement next day to the day on which news is published (1 if it goes up and 0 if it goes down).

6. MODEL DEVELOPMENT:

- For my model I have used one of the most efficient machine learning algorithm LinearDiscriminantAnalysis(LDA).

- About the algorithm-Linear Discriminant Analysis (LDA) is a technique used in statistics, pattern recognition, and machine learning to find a linear combination of features that best separates two or more classes of objects or events.

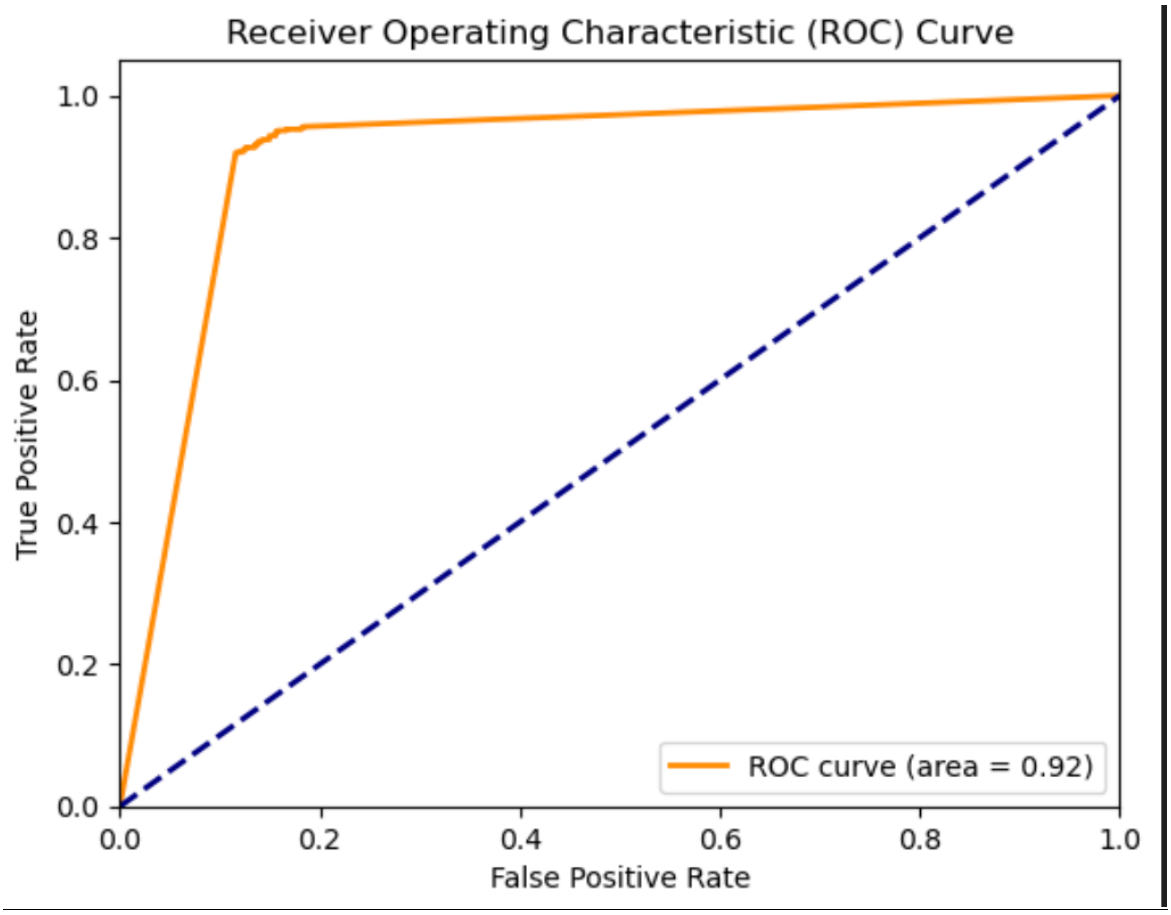
7. TRAINING AND TESTING:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

model=LinearDiscriminantAnalysis().fit(x_train,y_train)
```

8. MODEL EVALUATION:

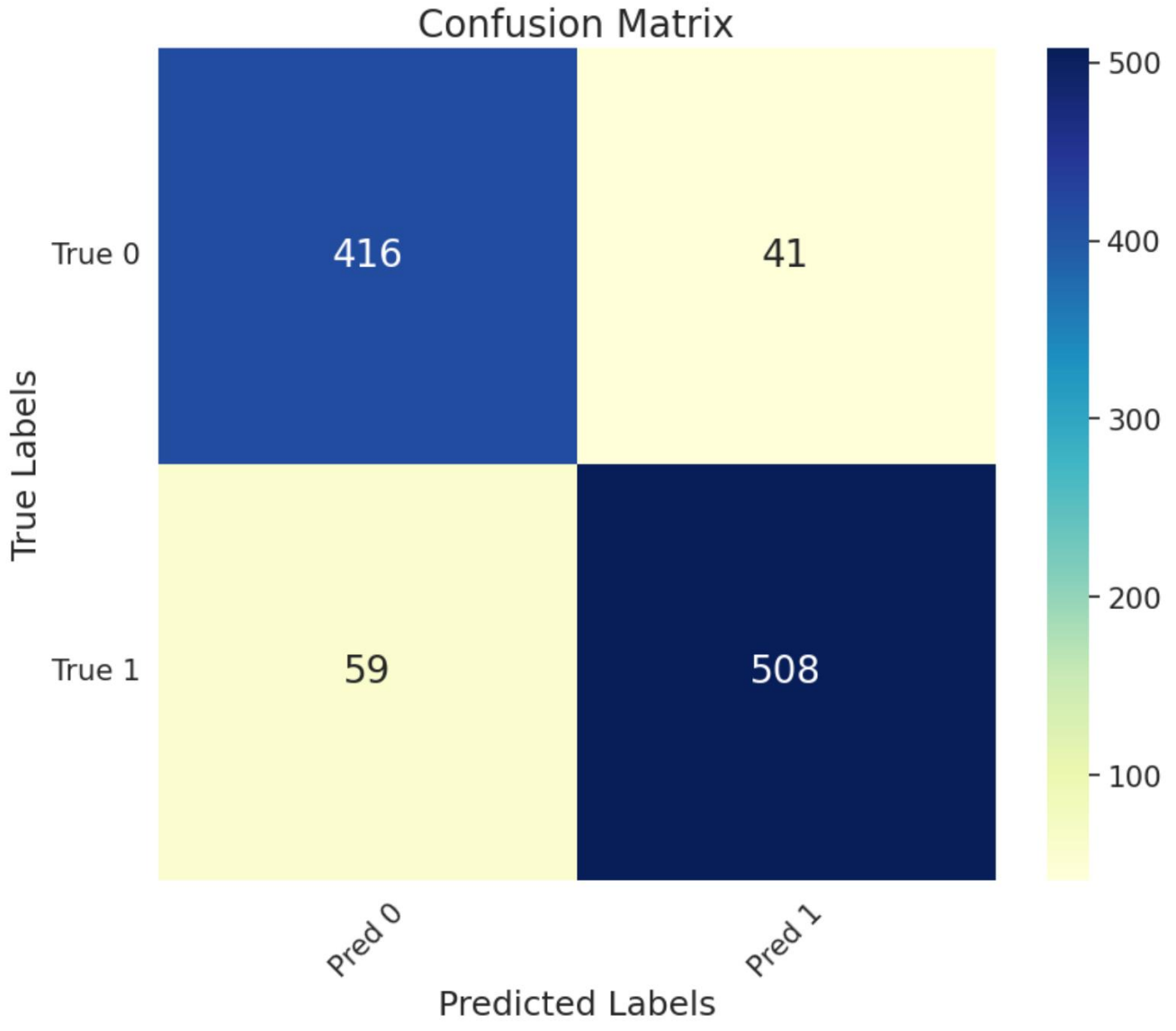
● Metrics like accuracy score, precision, recall, confusion matrix, F1 score and ROC curve analysis are used to assess the performance of the model.



```
[[416  41]
 [ 59 508]]
              precision    recall  f1-score   support

           0       0.88      0.91      0.89       457
           1       0.93      0.90      0.91       567

    accuracy                           0.90      1024
   macro avg       0.90      0.90      0.90      1024
weighted avg       0.90      0.90      0.90      1024
```



9. TRAINING STRATEGY AND EXPLANATION:

Overview

● The trading strategy implemented in the code is a simple buy-and-hold strategy with specific conditions for buying and selling stocks based on predictions and certain risk management rules. The goal is to buy stocks when a prediction signal indicates a potential price increase and sell them when a prediction signal indicates a potential price decrease or when predefined stop-loss or take-profit conditions are met. The performance of this strategy is evaluated using several key metrics, including the Sharpe ratio, maximum drawdowns, number of trades executed, and win ratio.

***Initial Setup**:*

- **Initial Capital:** The strategy starts with an initial capital of $10,000.
- **stop_loss** = 0.05: A stop-loss is set at 5%, meaning if the stock price drops 5% below the buy price, the position will be sold to limit losses.
- **take_profit** = 0.10: A take-profit is set at 10%, meaning if the stock price rises 10% above the buy price, the position will be sold to lock in profits.
- **initial_capital** = 1000: The initial amount of money available for trading is $1000.
- **cash = initial_capital**: The available cash for trading starts with the initial capital.

- **position** = 0: Initially, there are no stock holdings.
- **buy_price** = 0: The price at which the stock was bought (initialized to 0).
- **portfolio_values** = []: A list to keep track of the portfolio value over time

- **Buy Condition:**
- If the label is 1 (indicating a buy signal) and there is enough cash to buy the stock at the opening price, a buy order is executed:
    - position : The number of shares bought.
    - cash = 0: Cash is now fully invested in the stock.

- buy_price: Record the price at which the stock was bought.

- **Sell Condition**:

  If the label is 0 (indicating a sell signal) and there is an existing position, a sell order is executed:

  - Cash: Convert the stock position back to cash at the opening price.
  - position = 0: Reset the stock holdings to zero.

- **Stop-Loss and Take-Profit**:

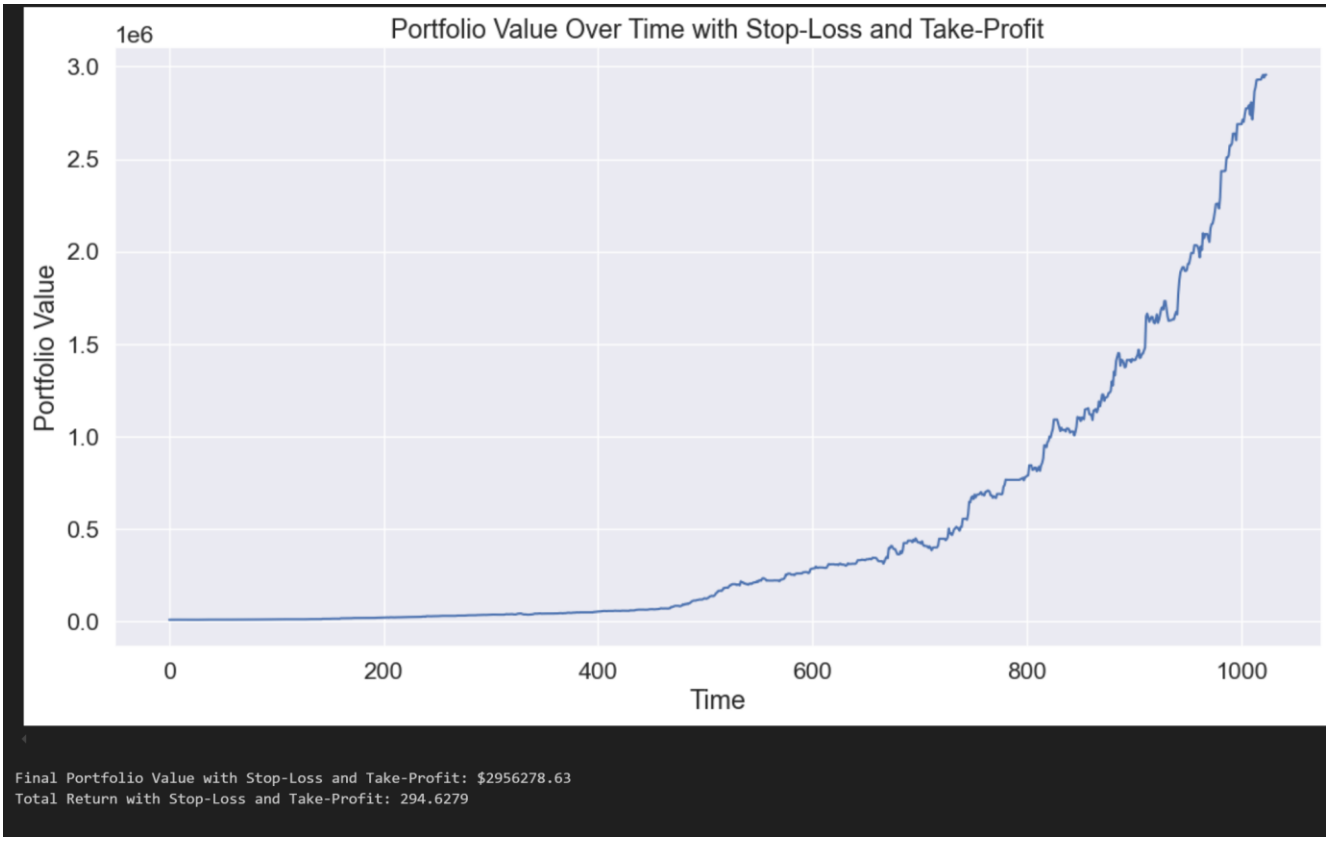
  If there is an existing position:

  - The stop-loss condition checks if the lowest price of the day falls below the stop-loss threshold. If so, the position is sold at the lowest price of the day.
  - The take-profit condition checks if the highest price of the day exceeds the take-profit threshold. If so, the position is sold at the highest price of the day.

- **Portfolio Value Calculation**:

  The portfolio value is the sum of the available cash and the value of the stock holdings at the closing price of the day. This value is appended to the portfolio_values list for tracking over time.

- **Plotting**:
- The plot section visualizes the stock prices along with the buy/sell signals:
  - The stock price over time is plotted.
  - Buy signals are marked with green upward-pointing triangles.
  - Sell signals are marked with red downward-pointing triangles.
  - Labels, legend, and title are added for better interpretation.



PERFORMANCE METRICS:

- Number of Trades Executed: The total number of buy/sell transactions.

● Win Ratio: The proportion of profitable trades out of the total trades executed.

● Sharpe Ratio: A measure of risk-adjusted return.

● Maximum Drawdown: The largest peak-to-trough decline in the value of the portfolio.

INITIAL CAPITAL: $10,000

FINAL CAPITAL: $2956278.63

```
# Calculate daily returns
df6['Return'] = df6['Close'].pct_change()

# Calculate strategy returns based on signals
df6['StrategyReturn'] = df6['Return'] * df6['Label'].shift(1)
# Annualized return and standard deviation
annualized_return = df6['StrategyReturn'].mean() * 252
annualized_volatility = df6['StrategyReturn'].std() * np.sqrt(252)

# Assuming a risk-free rate of 0 for simplicity
sharpe_ratio = annualized_return / annualized_volatility
print(f"Sharpe Ratio: {sharpe_ratio:.2f}")
```

✓ 0.0s

```
Sharpe Ratio: 7.84
```

```
# Calculate cumulative returns
df6['CumulativeReturn'] = (1 + df6['StrategyReturn']).cumprod()

# Calculate drawdowns
df6['CumulativeMax'] = df6['CumulativeReturn'].cummax()
df6['Drawdown'] = df6['CumulativeReturn'] / df6['CumulativeMax'] - 1

# Maximum drawdown
max_drawdown = df6['Drawdown'].min()
print(f"Maximum Drawdown: {max_drawdown:.2%}")
```

✓ 0.0s

```
Maximum Drawdown: -6.21%
```

```python
# Count the number of trades executed
num_trades = df6['Label'].diff().abs().sum()  # assuming each change in signal is a trade
print(f"Number of Trades Executed: {num_trades}")
```

✓ 0.0s

```
Number of Trades Executed: 495.0
```

```python
# Calculate individual trade returns
df6['TradeReturn'] = df6['StrategyReturn'][df6['Label'] != 0]

# Count winning trades
num_winning_trades = df6['TradeReturn'][df6['TradeReturn'] > 0].count()
num_losing_trades = df6['TradeReturn'][df6['TradeReturn'] <= 0].count()

# Win ratio
win_ratio = num_winning_trades / (num_winning_trades + num_losing_trades)
print(f"Win Ratio: {win_ratio:.2%}")
```

✓ 0.0s

```
Win Ratio: 49.47%
```

- **DRAWBACKS:**

  The accuracy and reliability of sentiment analysis may be affected by factors such as ambiguity in language, sarcasm, and context-dependent interpretations, requiring robust NLP techniques and validation procedures. The availability and quality of textual data may vary across different stocks and time periods, necessitating careful data selection and preprocessing to ensure consistency and relevance. The performance of the sentiment analysis model may be influenced by changes in market conditions, investor behavior, and external events, requiring regular updates and recalibration of the model.