Task 7 - Extension - Sentiment-Based Stock Price Movement Prediction

Overview

In this task, the existing stock price prediction model would be extended with sentiment analysis from external textual sources (i.e., financial news or social media) incorporated into itself. The goal is to develop a classification model that predicts whether the stock price will rise or fall on the following day.

Data Collection and Preprocessing

This component of the task required relevant textual data using appropriate APIs or publicly available sources. An important notice is that the collected data must be time-aligned with the historical stock price dataset used in previous tasks, which was collected from the ticker `CBA.AX` representing Commonwealth Bank of Australia from 01/01/2020 to 01/08/2025.

Similar to the previous tasks, the historical stock price data is downloaded using the Yahoo Finance API, which monitors real-time price data from the US market. As `CBA.AX` is listed on the Australian Securities Exchange (ASX), its price on Yahoo Finance would be delayed. Moreover, the news and headlines mentioning `CBA.AX` probably would not be time-aligned with the collected price data due to 15 hours of time zone difference. Therefore, `CBA.AX` is replace with `AAPL`, which is the ticker representing Apple Inc., a major American technology company, because its prices would be monitored in real time, and it would be easier to collect news and media data mentioning a company of that amount of attention and speculation.

In terms of data source, even when a company with huge media attention like Apple was chosen as a replacement, there were challenges regarding the process of data collection. Having considered finance and media APIs offering company news, I noticed that they offered too little data comparing to the five-

and-a-half years of historical price data I used, and very limited numbers of requests and calls for free or free trial users (e.g., NewsAPI only allows searching articles up to a month old for 100 times daily at most; Yahoo Finance, although allowing unlimited calls as an unofficial API, only displays a maximum of 8 items per request; renowned sources such as Bloomberg and Reuters do not allow general use). As a result, the data which would be used in this task was collected from two non-overlapping sources:

- The dataset "Apple Stock (AAPL): Historical Financial News Data" contains comprehensive financial news and headlines related to Apple Inc. spanning nearly nine years, from 19/02/2016 to 27/11/2024. The included articles are mostly fetched from sites such as Yahoo Finance, Yahoo News, and The Motley Fool, and was retrieved from Kaggle, an open data science community platform.
- A total of 226 articles published by and summarized from reputable agencies including Bloomberg, Yahoo News, Reuters, Business Insider were collected using Finnhub API. This API offers 1 year of historical news data, but perhaps the limit had been reached, with the collected articles and headlines only spanning for a week's time leading to the ending date of 31/07/2025. The amount of data is somewhat small, but would ensure the data used to predict for the trading days beyond the ending date to be accurate and updated.

With that in mind, a new module named `news.py` had been set up to accommodate functions fetching, loading, and preprocessing the news data. The code below shows the implementation of the `load_kaggle_aapl_data()` function. It would load the Kaggle AAPL dataset, which had previously been downloaded to the `data` folder; normalize the time index to only date for consistency with the stock price data; eliminate out-of-scope dates; combine the entries having the same publication date; fill in missing dates (i.e., not having any relevant articles); and finally impute invalid entries with sensible values.

```python
def load_kaggle_aapl_data(start_date, end_date):
    """Load and preprocess Kaggle AAPL news dataset.
    Args:
        start_date (str): Start date in 'YYYY-MM-DD' format.
        end_date (str): End date in 'YYYY-MM-DD' format.
    Returns:
        pd.DataFrame: Preprocessed news data with date as index.
    """
    # Load Kaggle AAPL news dataset.
    df = pd.read_csv(KAGGLE_AAPL_NEWS_PATH, index_col='date',
                     parse_dates=True).sort_index()
    # Make sure the index contains only date, similar to the stock data.
    df.index = df.index.date
    df.index = pd.to_datetime(df.index)
    # Select data only after the specified start date.
    df = df.loc[start_date:]
    # Select only relevant columns (title, content).
    df = df[['title', 'content']]
    # Append a space to title and content to separate concatenated texts.
    df['title'] += ' '
    df['content'] += ' '
    # Group by date and aggregate (sum).
    df = df.groupby(pd.Grouper(freq='D')).sum()
    # Convert input date strings to datetime objects.
    start_date_dt = datetime.strptime(start_date, '%Y-%m-%d')
    end_date_dt = datetime.strptime(end_date, '%Y-%m-%d')
    # Create a complete date range from start to end date.
    all_dates = pd.date_range(start=start_date_dt, end=end_date_dt, freq='D')
    # Reindex the DataFrame to include all dates in the range.
    df = df.reindex(all_dates)
    df = df.rename_axis('date')
    # Fill missing and zero values: empty strings for text.
    df.fillna('', inplace=True)
    df.replace(0, '', inplace=True)
    # Drop the last row, as it exceeds the stock data.
    df.drop(df.index[-1], inplace=True)
    # Return the preprocessed Kaggle DataFrame.
    return df
```

The `load_kaggle_aapl_data()` function.

```
                                                 title                                                content
2020-03-23                          3 ETFs To Short The Dow  With last Friday's 4.55% slide, the Dow Jones ...
2020-03-24       Are Sporting Goods Essential in a Pandemic?  (Bloomberg Opinion) -- You really can’t blame ...
2020-03-25                                                0                                                  0
2020-03-26                                                0                                                  0
2020-03-27  Olympics delay deals setback to Samsung's plan...  By Hyunjoo Jin\n\nSEOUL (Reuters) - For Samsun...
...                                                       ...                                                ...
2024-11-23  Could Buying Apple Stock Today Set You Up for ...  Apple Intelligence is just the company's lates...
2024-11-24  How Tim Cook Cracked the Code on Working With ...  The election is leading to a return of persona...
2024-11-25  A Complete Guide to College Savings Accounts i...  A Complete Guide to College Savings Accounts i...
2024-11-26  What Is the S&P 500 Index? Dow Jones Industria...  What Is the S&P 500 Index? Dow Jones Industria...
2024-11-27  Berkshire Stock Hits Record Even as Company Re...  Warren Buffett’s caution, his advancing age, a...

[1711 rows x 2 columns]
           title content
date
2020-01-01
2020-01-02
2020-01-03
2020-01-04
2020-01-05
...          ...     ...
2025-07-27
2025-07-28
2025-07-29
2025-07-30
2025-07-31

[2039 rows x 2 columns]
```

The output of the `load_kaggle_aapl_data()` function, before and after date range imputations.

As the recent data would be retrieved from Finnhub API, the `finnhub-python` package had been added to the `requirements.txt` file, and thus installed in the virtual environment. The code below shows the `fetch_finnhub_data()` function, which would download the data required using a free API key and save it to the `data` folder for later use (and also for saving API calls due to Finnhub’s free user limit).

```python
def fetch_finnhub_data(start_date, end_date):
    """Fetch Finnhub AAPL news data and save to local CSV.
    Args:
        start_date (str): Start date in 'YYYY-MM-DD' format.
        end_date (str): End date in 'YYYY-MM-DD' format.
    """
    # Initialize Finnhub client.
    client = finnhub.Client(api_key=FINNHUB_API_KEY)
    # Adjust end date not to be inclusive, similar to Yahoo Finance.
    end_date_dt = datetime.strptime(end_date, '%Y-%m-%d')
    end_date_dt -= timedelta(days=1)
    end_date = end_date_dt.strftime('%Y-%m-%d')
    # Fetch company news for AAPL from Finnhub.
    data = client.company_news('AAPL', _from=start_date, to=end_date)
    # Convert to DataFrame and save the fetched data to local CSV.
    data = pd.DataFrame(data)
    data.to_csv(FINNHUB_AAPL_NEWS_PATH, index=False)
    # CLI output confirmation.
    print(f"Fetched Finnhub data from {start_date} to {end_date}"
          f"and saved to {FINNHUB_AAPL_NEWS_PATH}.")
    return data
```

The `fetch_finnhub_data()` function.

Having saved the AAPL news data from Finnhub, the function `load_finnhub_aapl_data()` shown in the code below would follow a similar procedure to the `load_kaggle_aapl_data()` function, and ensure that the format of two output DataFrames to be identical.

```python
def load_finnhub_aapl_data():
    """Load and preprocess Finnhub AAPL news dataset.
    Args:
        start_date (str): Start date in 'YYYY-MM-DD' format.
        end_date (str): End date in 'YYYY-MM-DD' format.
    Returns:
        pd.DataFrame: Preprocessed news data with date as index.
    """
    # Load Finnhub AAPL news dataset from local storage.
    df = pd.read_csv('data/finnhub_aapl_news.csv', index_col='datetime',
                     parse_dates=True).sort_index()
    # Drop irrelevant columns, retain only headline and summary.
    df.drop(columns=['category', 'id', 'image', 'related', 'source',
                     'url'], inplace=True)
    # Drop missing entries.
    df.dropna(inplace=True)
    # Convert index from UNIX timestamp to datetime, and retain only date.
    df.index = pd.to_datetime(df.index, unit='s').date
    df.index = pd.to_datetime(df.index)
    # Append a space to headline and summary to separate concatenated texts.
    df['headline'] += ' '
    df['summary'] += ' '
    # Group by date and aggregate by summing the text fields.
    df = df.groupby(pd.Grouper(freq='D')).sum()
    # Rename index and columns for integrity.
    df.rename_axis('date', inplace=True)
    df.rename(columns={'headline': 'title', 'summary': 'content'},
              inplace=True)
    # Return the preprocessed Finnhub DataFrame.
    return df
```

The `load_finnhub_aapl_data()` function.

date	title	content
2025-07-26	Alphabet Had a ‘Standout Quarter.’ Should You ...	Alphabet stock rose only slightly yesterday de...
2025-07-27	EPU: Peru Isn't One Of My Top LatAm Picks Righ...	Peru's political instability and upcoming elec...
2025-07-28	Dow Jones Futures Rise On Trump-EU Trade Deal;...	Dow Jones Futures Rise On Trump-EU Trade Deal;...
2025-07-29	Franklin Growth Fund Q2 2025 Commentary Apple ...	Franklin Growth Fund outperformed its S&P 500 ...
2025-07-30	ClearBridge Large Cap Growth ESG Portfolios Q2...	U.S. equities persevered through tariff and ge...
2025-07-31	UBS Lifts PT on QUALCOMM Incorporated (QCOM) t...	QUALCOMM Incorporated (NASDAQ:QCOM) is one of ...

The output DataFrame of the `load_finnhub_aapl_data()` function.

By running the `news.py` script, the two output DataFrames would be combined together using the `add()` function, merging text data from each same date, in order to create a ready-to-use news DataFrame and save it inside the `data` folder for later use.

	title	content
date		
2020-01-01		
2020-01-02		
2020-01-03		
2020-01-04		
2020-01-05		
...	...	...
2025-07-27	EPU: Peru Isn't One Of My Top LatAm Picks Righ...	Peru's political instability and upcoming elec...
2025-07-28	Dow Jones Futures Rise On Trump-EU Trade Deal;...	Dow Jones Futures Rise On Trump-EU Trade Deal;...
2025-07-29	Franklin Growth Fund Q2 2025 Commentary Apple ...	Franklin Growth Fund outperformed its S&P 500 ...
2025-07-30	ClearBridge Large Cap Growth ESG Portfolios Q2...	U.S. equities persevered through tariff and ge...
2025-07-31	UBS Lifts PT on QUALCOMM Incorporated (QCOM) t...	QUALCOMM Incorporated (NASDAQ:QCOM) is one of ...

[2039 rows x 2 columns]

The fully-preprocessed AAPL news DataFrame.

Sentiment Analysis

With text data collected and ready to use, I then proceeded to perform sentiment analysis. Sentiment analysis is the process of using natural language processing (NLP) to determine the underlying emotional opinion within the text (i.e., positive, negative, or neutral). By performing this technique on news data regarding Apple, the system would be able to better understand the aggregated events that approach each trading day, and hence able to quantitatively evaluate the amount of effect toward the daily closing price.

For this component, I had chosen to work with Natural Language Toolkit (NLTK), a popular platform focused on development using human language data. It has a parsimonious lexicon and rule-based model specifically for sentiment analysis called VADER (Valence Aware Dictionary and sEntiment Reasoner). First, I installed NLTK into the virtual development environment, and used the provided downloader tool to fetch the package containing VADER.

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

The setup process of NLTK and VADER.

Having aggregated the news data to a daily basis while preprocessing, I went straight to implementing sentiment analysis using VADER in a then-newly created module `sentiments.py`. This code below shows the function used to load the combined AAPL data, initialize a VADER sentiment intensity analyzer model, using it to perform sentiment analysis, and finally append the compound sentiment score to the news DataFrame.

```python
def vader_sentiment_analysis(path: str) -> pd.DataFrame:
    """Perform sentiment analysis on AAPL news data using Vader.
    Args:
        path (str): Path to the combined AAPL news CSV file.
    Returns:
        pd.DataFrame: DataFrame with Vader sentiment scores added.
    """
    # Input combined CSV AAPL data, and rename index to 'date'.
    df = pd.read_csv(path, index_col=0, parse_dates=True)
    df.rename_axis('date', inplace=True)
    # Loaded DataFrame contains NaN values. Impute them as empty strings.
    df = df.fillna('')
    # Initialize Vader sentiment analyzer.
    vader = SentimentIntensityAnalyzer()
    # Perform sentiment analysis using Vader.
    df['vader_sentiment'] = df['title'].apply(vader.polarity_scores)
    # Calculate compound sentiment score from Vader results.
    df['vader_compound'] = df['vader_sentiment'].apply(lambda x: x['compound'])
    return df
```

The `vader_sentiment_analysis()` function.

An error I had encountered showing that when loading the AAPL news data from CSV format, any values that I have modified from `NaN` to empty strings would become `NaN` again. Therefore, they would be re-imputed to empty strings, as shown in the code snippet above.

The output below shows the successfully implemented sentiment analysis metrics of the news data appended to the DataFrame itself.

	title	...	vader_compound
date		...	
2020-01-01		...	0.0000
2020-01-02		...	0.0000
2020-01-03		...	0.0000
2020-01-04		...	0.0000
2020-01-05		...	0.0000
...	...	...	...
2025-07-27	EPU: Peru Isn't One Of My Top LatAm Picks Righ...	...	0.9398
2025-07-28	Dow Jones Futures Rise On Trump-EU Trade Deal;...	...	0.9963
2025-07-29	Franklin Growth Fund Q2 2025 Commentary Apple ...	...	0.9599
2025-07-30	ClearBridge Large Cap Growth ESG Portfolios Q2...	...	0.9826
2025-07-31	UBS Lifts PT on QUALCOMM Incorporated (QCOM) t...	...	0.9714

The output DataFrame with sentiment analysis results successfully appended.

Feature Engineering and Modelling

This phase of the task required creating input features that combine both historical stock data (e.g., technical indicators, lagged values) and sentiment scores, and training a classification model using these features to predict whether the next day's closing price will be higher or lower than the current days. As the feature engineering functionality had already been implemented within the `load_data()` function responsible for downloading and preprocessing historical stock price data in the previous tasks, I thought it would be a good idea to include an option argument inside that function to whether sentiment scores would be included as a feature.

Here is when a problem arose. As shown earlier, the output AAPL news data with sentiment scores had been aggregated on a daily basis. The stock price data from Yahoo Finance is also collected daily; however, the weekend days (aka. non-trading days) are not included. As these non-trading days would be present in the news data, further actions should be made to make the two different sources of data effectively time-aligned. Simply dropping these entries would be a possible solution, but on the other hand, it would be unable to make use of the headlines posted on the weekends. That is why I had chosen to calculate the average sentiment scores of every Monday along with each one's previous weekends, and assign the result to the Monday entries. By doing this, no news data would be wasted, and the sentiment analysis data would be usable for feature engineering, as the monitored dates are well-aligned.

With that decided, I extended the `vader_sentiment_analysis()` function as shown by the code below. Besides the mentioned functionalities, the function now return only the sentiment score column, calculating rolling averages of every weekend and Monday, and drop all entries recorded on non-trading days.

```
    # Keep only compound sentiment score column.
    series = df['vader_compound'].copy()
    # -----
    # Filter out mondays to perform weekend averaging.
    mondays = series.index[series.index.weekday == 0]
    # Perform weekend rolling average to mondays.
    for monday in mondays:
        # Include Saturday, Sunday, and Monday for averaging.
        weekend_dates = [monday - pd.Timedelta(days=2),
                         monday - pd.Timedelta(days=1), monday]
        # Reindex in case of missing dates.
        vals = series.reindex(weekend_dates)
        # Calculate mean sentiment for the weekend, and assign to Monday.
        mean = vals.mean(skipna=True)
        if not pd.isna(mean):
            series.loc[monday] = mean
    # Remove weekend entries from the series.
    weekend_mask = series.index.weekday >= 5
    series = series[~weekend_mask]
    return series
```

The extended `vader_sentiment_analysis()` function, now with rolling weekend and Monday averages.

With that done, I proceeded as planned, which is extending the existing `load_data()` function inside the `stock_prediction.py` module, which is the function responsible for feature engineering and described in detail in previous tasks' reports. The first things to do were including a new boolean argument `sentiment` along with `news_data_path` for the path to the combined dataset containing Apple news, appending `'Sentiment'` to the list of feature columns, and calculating sentiment scores to be used. These were done right at the beginning of the function body.

```
if sentiment:
    assert news_data_path is not None, 'News data path must be provided if sentiment analysis is enabled.'
    feature_columns.append('Sentiment')
    sentiment_series = vader_sentiment_analysis(news_data_path)
```

Appending `'Sentiment'` to the list of feature columns.

Then, if the option is enabled, the `'Sentiment'` column would be appended to the downloaded stock DataFrame.

```
    if sentiment:
        # Merge the sentiment scores into the main DataFrame.
        df = df.merge(sentiment_series.rename("Sentiment"), how="left",
                      left_index=True, right_index=True)
        # Fill any NaN sentiment values with 0 (neutral sentiment).
        df["Sentiment"].fillna(0, inplace=True)
```

Appending the calculated `'Sentiment'` column to the downloaded DataFrame.

As a fact, the sentiment scores are already normalized to `[-1, 1]`, so I chose not to use `MinMaxScaler`, but manual scaling instead. As shown in the code snippet, the scores would be manually scaled to `[0, 1]` using a simple formula, after having been excluded from the list of columns to scale.

```
    # Scale the data if requested.
    if scale_data:
        column_scaler = {}
        # Scale the data from 0 to 1.
        for col in feature_columns:
            if col != 'Sentiment':  # Do not scale sentiment column
                scaler = MinMaxScaler()
                # Reshape the data to be a 2D array for the scaler.
                # The parameter -1 allows the function to determine the size of that dimension automatically.
                df[col] = scaler.fit_transform(df[col].values.reshape(-1, 1))
                # Save the scaler for this column.
                column_scaler[col] = scaler
            else:
                # Because sentiment had already been normalized, use manual scaling.
                df['Sentiment'] = (df['Sentiment'] + 1) / 2  # Scale from [-1, 1] to [0, 1]
```

Manual scaling for sentiment scores.

Before creating moving window sequences as specified in previous tasks, the `'Target'` column would be split into two cases: shifted `'Close'` price for the previous tasks of regression, and increase/decrease labels for this task's classification problem. The labels would be assigned by comparing the shifted `'Close'` price to the `'Close'` price of each day, and then changed to 1 if increasing, and 0 if decreasing. This process was helped by introducing a new boolean `classification` argument to the function for the choice of problems.

```
    # Create the target column by shifting "Close" upwards.
    df["Target"] = df["Close"].shift(-predict_days)
    if classification:
        df['Target'] = (df['Target'] >= df['Close']).astype(int)
```

Distinguishing targets for different cases of problems, and assigning labels.

Additionally, I chose to use a stacked LSTM neural network for this classification problem, similar to the previous regression problem, by extending an option for the `create_model()` function. The boolean `classification` option argument

had also been introduced for the same purpose of distinguishing different cases of problems. The only remaining modification to be done was splitting two different output layers for two cases of problems. For regression, the configuration would be kept the same, but for classification, the activation function for the output layer would be changed to sigmoid, while the loss function for the model would be set to binary cross-entropy, along with metrics including accuracy and F1-score.

```
219     if classification:
220         model.add(Dense(1, activation="sigmoid"))
221         # Compile the model.
222         model.compile(loss="binary_crossentropy", metrics=['accuracy', 'f1_score'], optimizer=optimizer)
223     else:
224         model.add(Dense(1, activation="linear"))
225         # Compile the model.
226         model.compile(loss=loss, metrics=['mae'], optimizer=optimizer)
```

Differentiating two different cases of output layers.

For training the model, a `ModelCheckpoint` callback would be configured similarly to the regression problem. This would help disregarding any overfitting, retrieving the best model during training and saved to the `models` folder.

```
35  # Callbacks.
36  checkpointer = ModelCheckpoint(filepath=f'models/{CLASSIFICATION_MODEL_NAME}.keras', save_best_only=True, verbose=1)
```

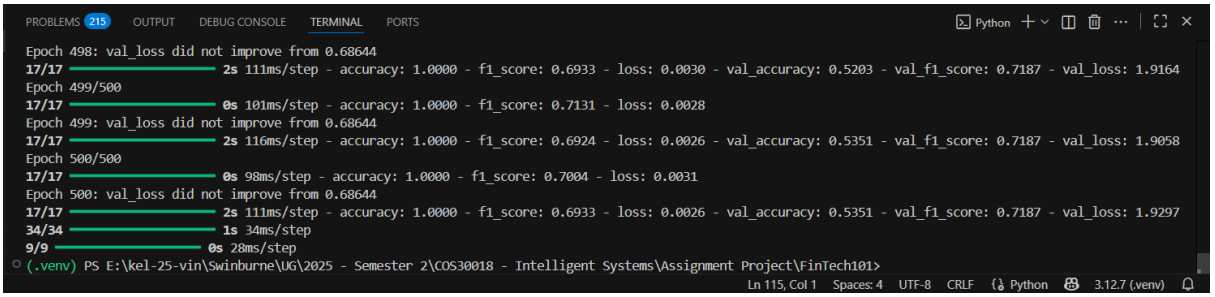
`ModelCheckpoint` callback configured.

When testing, I encountered a `ValueError` indicating the incompatible target shape which had prevented the training process to be executed. The reason is when using `MinMaxScaler`, all price columns had also been reshaped to the compatible 2D shape required by the model. This was not configured to the `'Target'` column, which had not been engineered at that moment. By reshaping the training and testing targets right before returning the result dictionary of the `load_data()` function, the problem was effectively resolved.

```
166     # Reshape y to 2D array for Keras compatibility.
167     result["y_train"] = result["y_train"].astype(np.float32).reshape(-1, 1)
168     result["y_test"] = result["y_test"].astype(np.float32).reshape(-1, 1)
```

Reshaping the targets for compatibility with the model and its metrics.

The figure below shows the successful training of the model.



Successful model training.

Evaluation

Having completed training the model, I proceeded to evaluate its predictions on testing data. The code below shows how the predictions were made using the retrieved best model, and how evaluation metrics are calculated and shown as outputs.

```python
# Retrieve the best model.
best_model = load_model(f'models/{CLASSIFICATION_MODEL_NAME}.keras', compile=False)
# Make predictions.
y_pred_train = (best_model.predict(data["X_train"]) >= 0.5).astype(int)
y_pred_test = (best_model.predict(data["X_test"]) >= 0.5).astype(int)

# Evaluations.
metrics = {'accuracy': accuracy_score(data['y_test'], y_pred_test),
           'precision': precision_score(data['y_test'], y_pred_test),
           'recall': recall_score(data['y_test'], y_pred_test),
           'f1_score': f1_score(data['y_test'], y_pred_test),
           'confusion_matrix': confusion_matrix(data['y_test'], y_pred_test),
           'classification_report': classification_report(data['y_test'],
                                                          y_pred_test,
                                                          output_dict=True)}
print('Evaluation Metrics of the Sentiment Classification Model:')
print(f"Accuracy: {metrics['accuracy']:.4f}")
print(f"Precision: {metrics['precision']:.4f}")
print(f"Recall: {metrics['recall']:.4f}")
print(f"F1 Score: {metrics['f1_score']:.4f}")
print('Confusion Matrix:')
print(metrics['confusion_matrix'])
print('Classification Report:')
print(pd.DataFrame(metrics['classification_report']).transpose())
```

Evaluation implementations.

And this was how the evaluation metrics of the sentiment analyzed classification model turned out.

```
Evaluation Metrics of the Sentiment Classification Model:
Accuracy: 0.5609
Precision: 0.5609
Recall: 1.0000
F1 Score: 0.7187
Confusion Matrix:
[[  0 119]
 [  0 152]]
Classification Report:
              precision    recall  f1-score     support
0.0            0.000000  0.000000  0.000000  119.000000
1.0            0.560886  1.000000  0.718676  152.000000
accuracy       0.560886  0.560886  0.560886    0.560886
macro avg      0.280443  0.500000  0.359338  271.000000
weighted avg   0.314593  0.560886  0.403095  271.000000
```

Evaluation metrics of the sentiment analyzed classification model.

As shown, the model predicted the price to go up every day. Having analyzed the sentiment scores, I found that the media had expressed positive or neutral sentiments on 259 out of 271 (i.e., 95.57%) trading days included in the testing data. Therefore, perhaps it can be considered that sentiment scores had a relatively strong impact on the model's predictions.

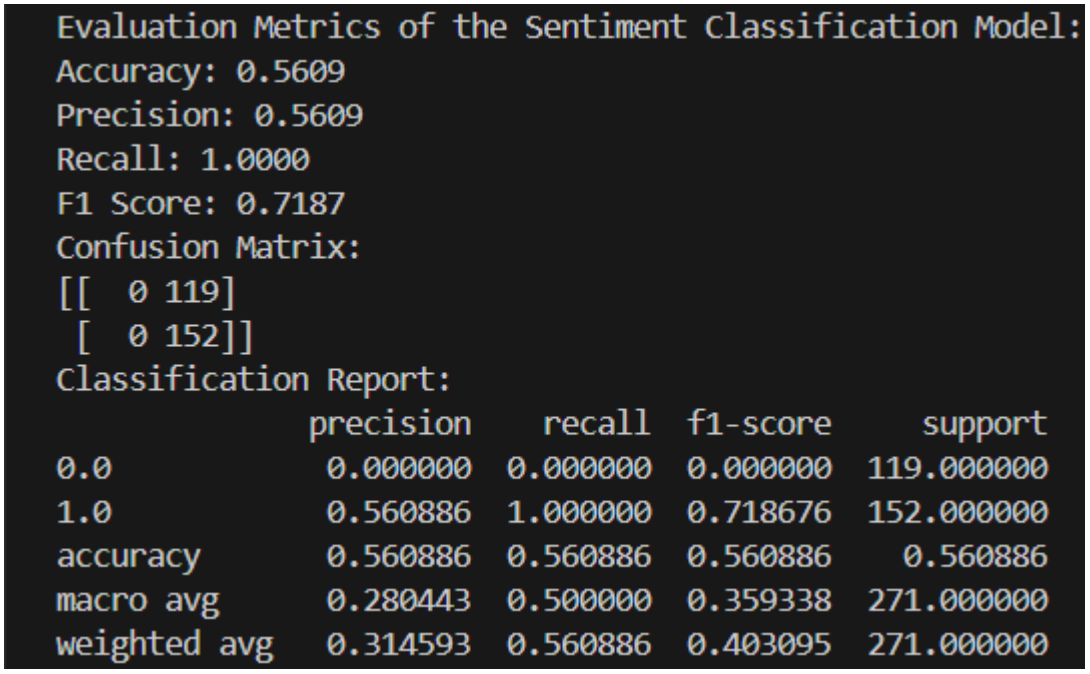
```
75  print(data['test_df'].shape)  # (271, 6)
76  print(data['test_df'][data['test_df']['Sentiment'] >= 0].shape)  # (259, 6)
```

A dominant ratio of positive sentiments from media resulted in a highly-diverted prediction summary.

By setting all sentiment-related parameters to False, I was able to train and observe the predictions and metrics of a baseline model. Here is how they turned out.

```
Epoch 499/500
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.5402 - f1_score: 0.6985 - loss: 0.6833
Epoch 499: val_loss did not improve from 0.68551
17/17 ━━━━━━━━━━━━━━━━━━━━ 2s 105ms/step - accuracy: 0.5388 - f1_score: 0.6924 - loss: 0.6854 - val_accuracy: 0.4502 - val_f1_score: 0.7187 - val_loss: 0.6952
Epoch 500/500
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.5544 - f1_score: 0.7045 - loss: 0.6857
Epoch 500: val_loss did not improve from 0.68551
17/17 ━━━━━━━━━━━━━━━━━━━━ 2s 106ms/step - accuracy: 0.5490 - f1_score: 0.6924 - loss: 0.6843 - val_accuracy: 0.4354 - val_f1_score: 0.7187 - val_loss: 0.6997
34/34 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

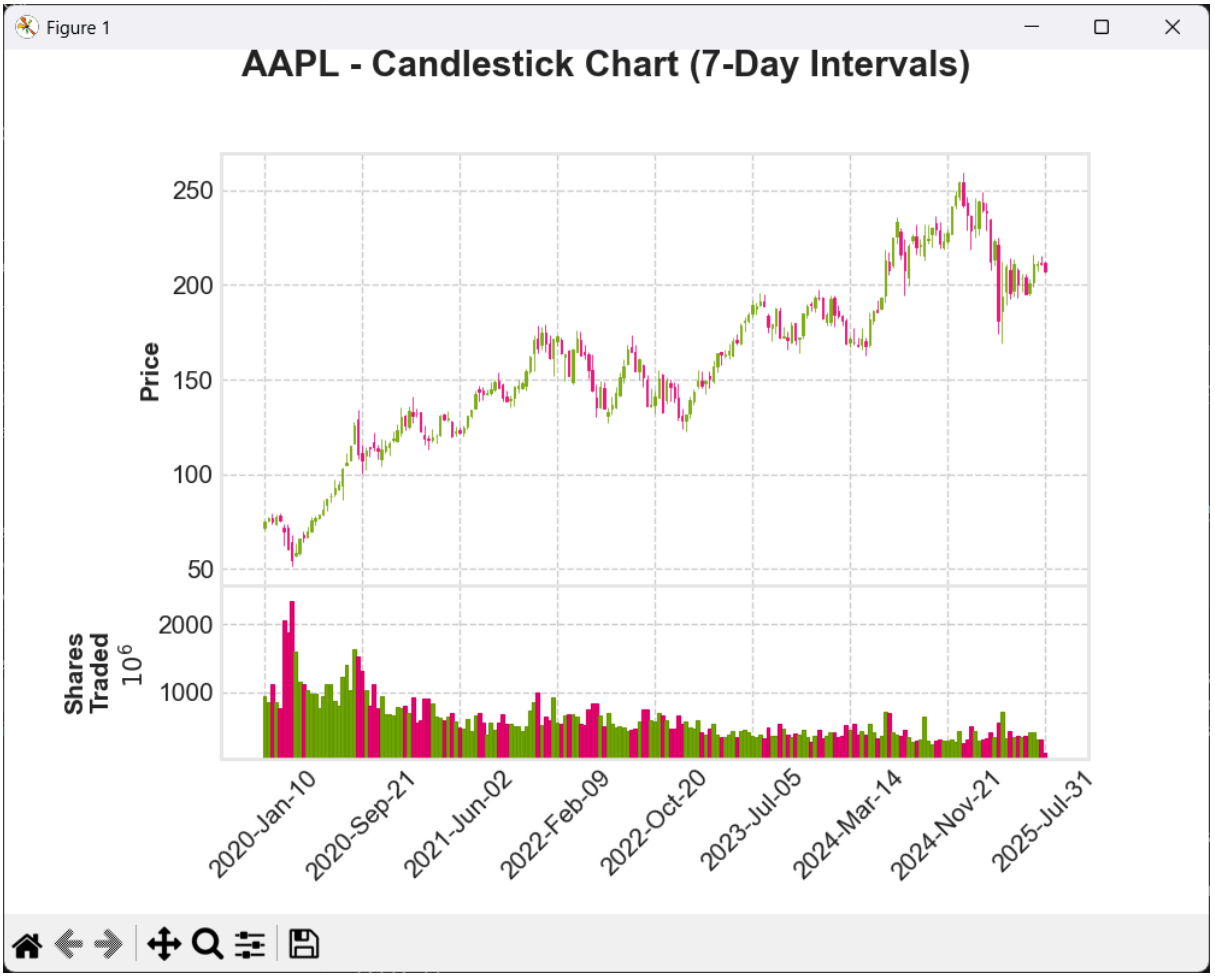
Successful training of sentiment-free baseline classification model.



```
Evaluation Metrics of the Sentiment Classification Model:
Accuracy: 0.5609
Precision: 0.5609
Recall: 1.0000
F1 Score: 0.7187
Confusion Matrix:
[[  0 119]
 [  0 152]]
Classification Report:
              precision    recall  f1-score     support
0.0            0.000000  0.000000  0.000000  119.000000
1.0            0.560886  1.000000  0.718676  152.000000
accuracy       0.560886  0.560886  0.560886    0.560886
macro avg      0.280443  0.500000  0.359338  271.000000
weighted avg   0.314593  0.560886  0.403095  271.000000
```

Evaluation metrics of sentiment-free baseline classification model.

Here, the predictions of the baseline model on the testing data were also fully rising. That led me to a conclusion that positive sentiments from the media were not the major influence on the highly-diverted predictions of both models, but perhaps that is due to the continuous pattern of rising price which may not be present in a short time, but can be easily identified over the five-and-a-half years of monitored data.



The ever-rising prices of Apple, which might make the predictions biased.

Although it could be inferred that mostly positive sentiments of the media was not the major influence on both model's predictions, it is easy to see that when training both models, the model with sentiment data was able to capture much more underlying patterns of the training data than the baseline model (see training metrics).

These above observations on the impact of sentiment analysis and the patterns of the training data can be highly beneficial on the course of improving the overall performance of the system.

References

- The dataset "Apple Stock (AAPL): Historical Financial News Data" on Kaggle: https://www.kaggle.com/datasets/frankossai/apple-stock-aapl-historical-financial-news-data.
- Finnhub API Documentation: https://finnhub.io/docs/api/company-news.

- NLTK Documentation: https://www.nltk.org/index.html.