PROJECT:

Stock Price Analysis Using Machine Learning G Deep Learning

By: Harshit Vishnoi

Third Year B.Tech (2022-2026)

DRONACHARYA GROUP OF INSTITUTIONS

TABLE OF CONTENTS

Project Overview:

This project explores stock price prediction using machine learning (ML) and deep learning (DL) models. By analyzing historical stock data of Apple Inc. (AAPL), the goal is to predict future stock prices and understand the impact of various technical indicators on stock price movements. The project utilizes a variety of models, including LSTM (Long Short-Term Memory), and techniques such as moving averages, RSI, MACD, and Bollinger Bands for feature engineering.

Objectives:

- Stock Price Prediction: Predict future stock prices of Apple Inc. (AAPL) using ML/DL models.
- Feature Engineering: Calculate technical indicators (SMA, EMA, RSI, MACD, Bollinger Bands) to enhance prediction accuracy.
- Model Implementation: Train a deep learning model (LSTM) and evaluate its performance using various metrics.
- Data Visualization: Visualize stock price trends and technical indicators for better insights.

Key Technologies and Tools Used:

- Languages C Libraries: Python, Pandas, Numpy, Matplotlib, Seaborn, TensorFlow, Keras, Scikit-Learn, Yahoo Finance
- Algorithms C Models: Linear Regression, Random Forest, LSTM (Deep Learning)
- Technical Indicators: Simple Moving Average (SMA), Exponential Moving Average (EMA), Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands
- Data Sources: Yahoo Finance (AAPL stock data from 2018-2024)

Methodology:

1. Data Collection: Stock price data for Apple Inc. (AAPL) was downloaded from Yahoo Finance for the period from January 1, 2018, to January 1, 2024.
2. Feature Engineering: Key technical indicators (SMA, EMA, RSI, MACD, Bollinger Bands) were computed to use as features for the prediction models.
3. Model Training: The LSTM model was used to train on the data and predict future stock prices.
4. Evaluation Metrics: The model's performance was evaluated using Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-Squared (R^2).
5. Visualization: Various visualizations, including the actual vs predicted stock prices, were created to assess the model's performance.

Expected Outcomes:

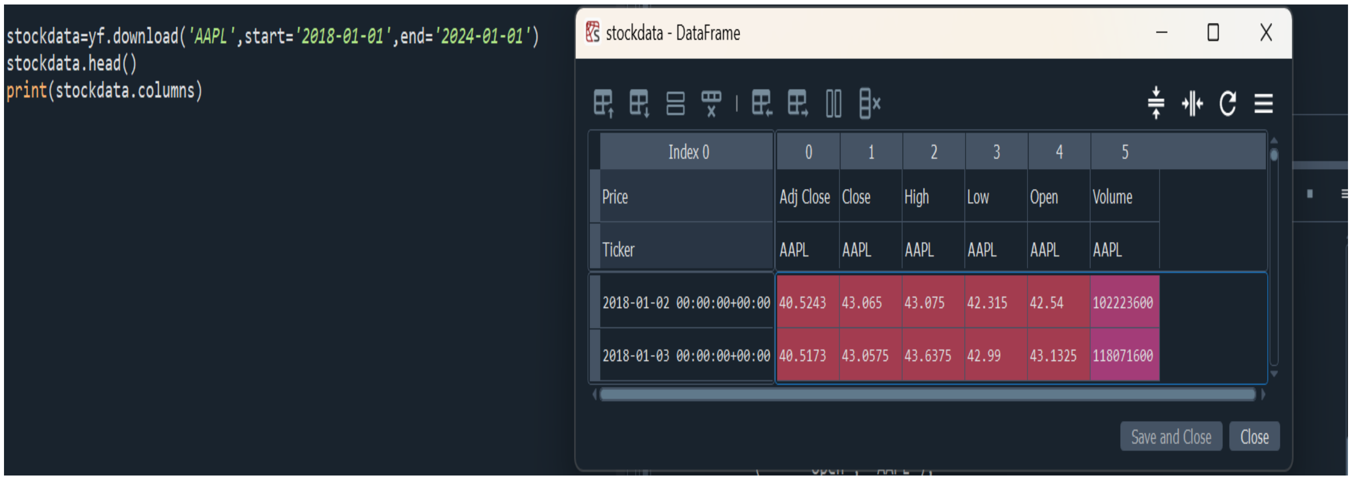
- Stock Price Prediction: The deep learning model (LSTM) is expected to predict stock prices with a reasonable level of accuracy.
- Insight Generation: The project will provide valuable insights into the behavior of Apple's stock and how technical indicators influence price movements.
- Model Evaluation: The results from the LSTM model will be compared with traditional ML models (e.g., Linear Regression, Random Forest) to assess performance.

IMPORTING LIBRARIES:

```
import yfinance as yf
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error, r2_score
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.optimizers import Adam
```

DATASET LOADING:

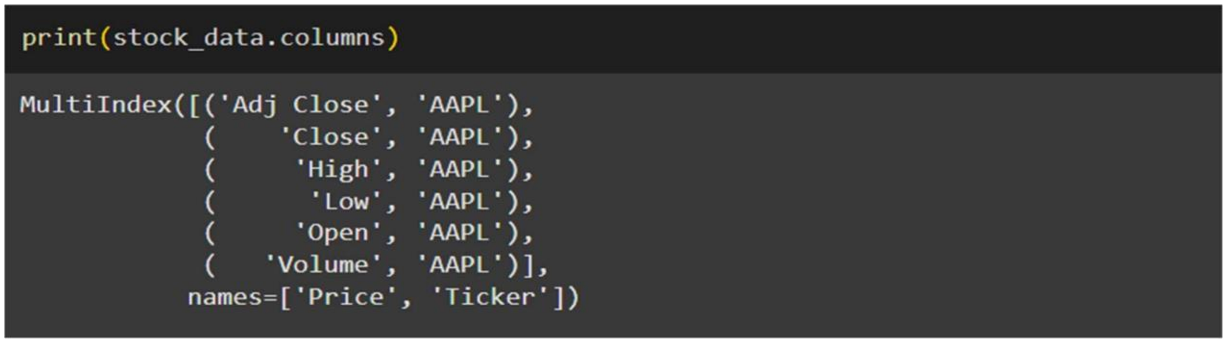
I have used the the yfinance that acts as API to extract the historical data about the stocks. I have retrieved the data from January 1, 2018, to January 1, 2024.



```
stockdata=yf.download('AAPL',start='2018-01-01',end='2024-01-01')
stockdata.head()
print(stockdata.columns)
```

stockdata - DataFrame

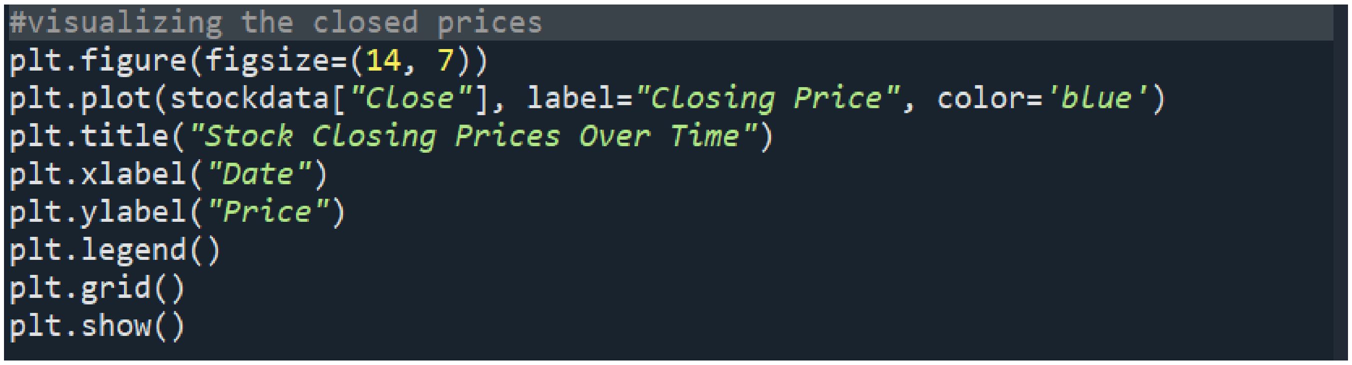
Index 0	0	1	2	3	4	5
Price	Adj Close	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL	AAPL
2018-01-02 00:00:00+00:00	40.5243	43.065	43.075	42.315	42.54	102223600
2018-01-03 00:00:00+00:00	40.5173	43.0575	43.6375	42.99	43.1325	118071600



```
print(stock_data.columns)
```

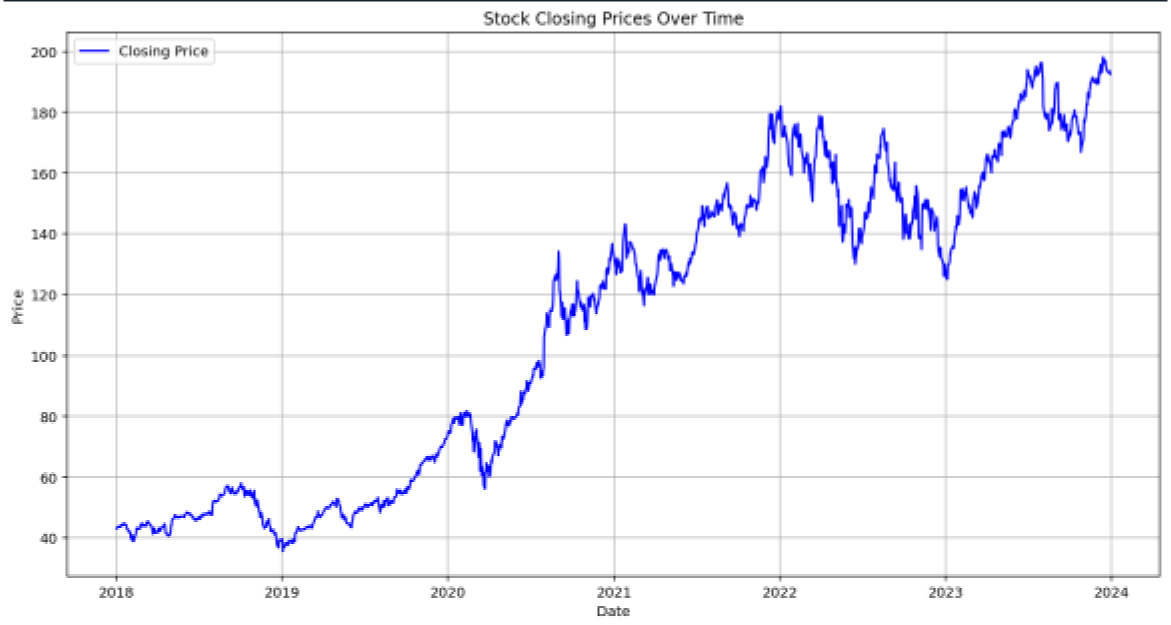
```
MultiIndex([('Adj Close', 'AAPL'),
            (    'Close', 'AAPL'),
            (     'High', 'AAPL'),
            (      'Low', 'AAPL'),
            (     'Open', 'AAPL'),
            (   'Volume', 'AAPL')],
           names=['Price', 'Ticker'])
```

VISUALIZING THE CLOSE PRIZES:



```
#visualizing the closed prices
plt.figure(figsize=(14, 7))
plt.plot(stockdata["Close"], label="Closing Price", color='blue')
plt.title("Stock Closing Prices Over Time")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid()
plt.show()
```

This would help me to see the trend of the close prices along with the years.

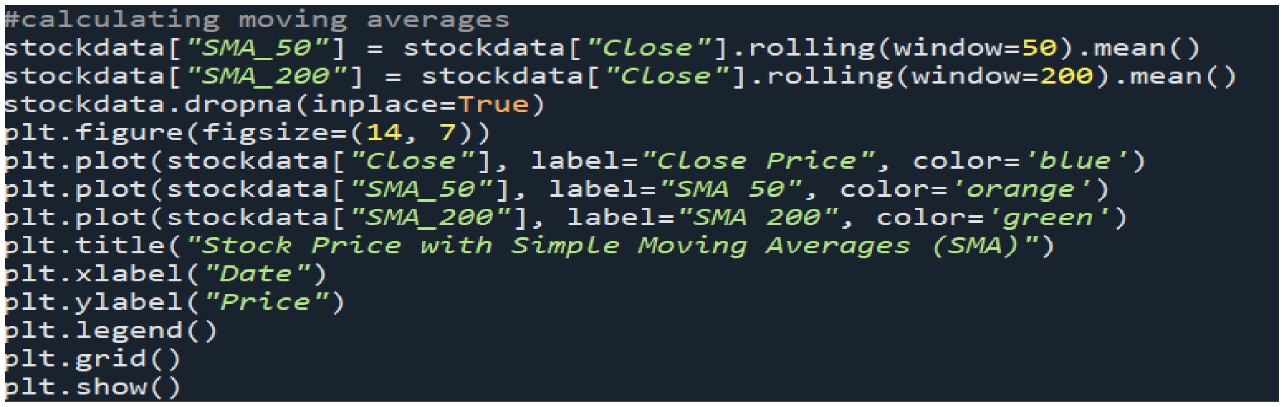


SMA AND EMA in Stock Prediction:

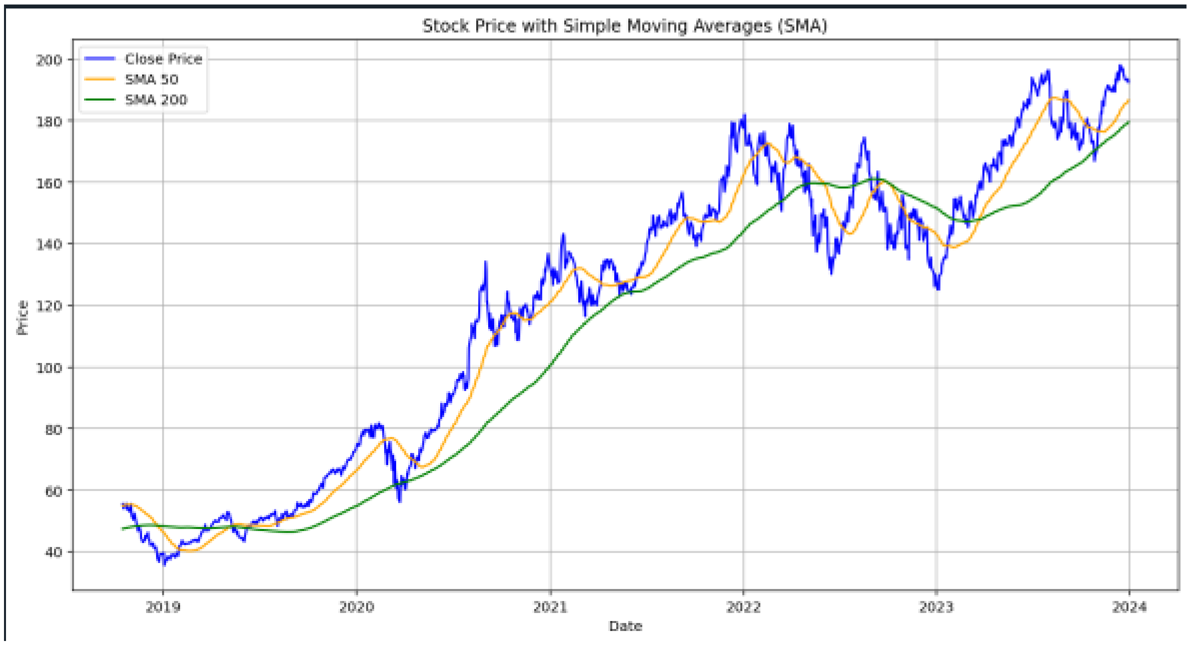
Moving Averages (SMA, EMA): These are used to smooth out stock price fluctuations and identify trends.

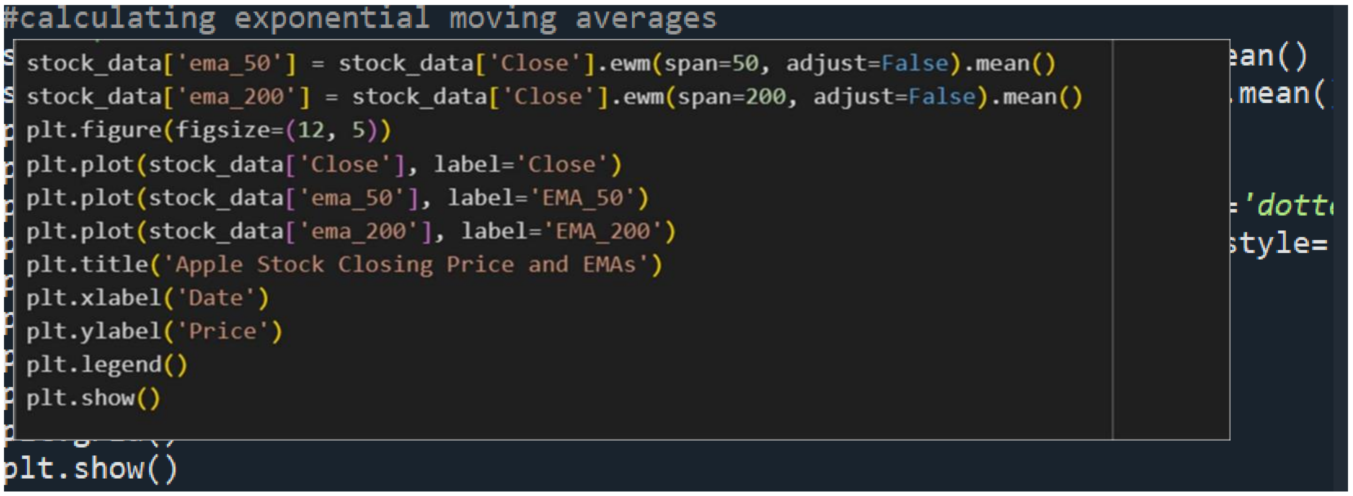
SMA(Simple Moving Average): Average of the stock prices over a specified window (e.g., 50 days).

EMA(Exponential Moving Average): Gives more weight to recent prices to respond faster to price changes.

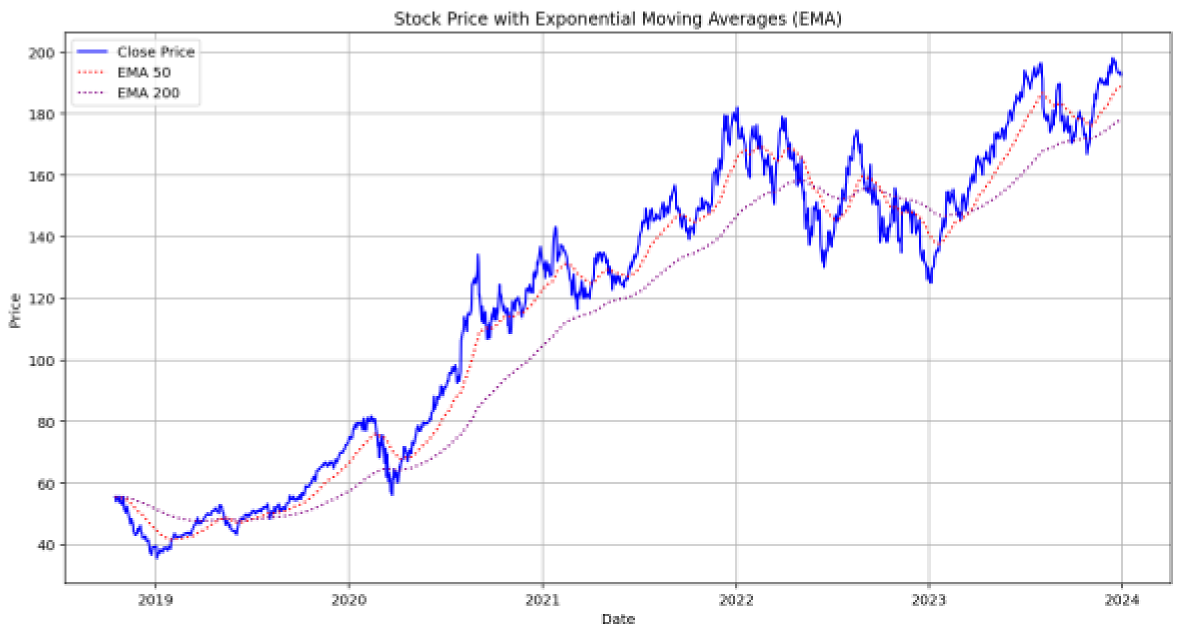


```python
#calculating moving averages
stockdata["SMA_50"] = stockdata["Close"].rolling(window=50).mean()
stockdata["SMA_200"] = stockdata["Close"].rolling(window=200).mean()
stockdata.dropna(inplace=True)
plt.figure(figsize=(14, 7))
plt.plot(stockdata["Close"], label="Close Price", color='blue')
plt.plot(stockdata["SMA_50"], label="SMA 50", color='orange')
plt.plot(stockdata["SMA_200"], label="SMA 200", color='green')
plt.title("Stock Price with Simple Moving Averages (SMA)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid()
plt.show()
```





```python
#calculating exponential moving averages
stock_data['ema_50'] = stock_data['Close'].ewm(span=50, adjust=False).mean()
stock_data['ema_200'] = stock_data['Close'].ewm(span=200, adjust=False).mean()
plt.figure(figsize=(12, 5))
plt.plot(stock_data['Close'], label='Close')
plt.plot(stock_data['ema_50'], label='EMA_50')
plt.plot(stock_data['ema_200'], label='EMA_200')
plt.title('Apple Stock Closing Price and EMAs')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
plt.show()
```



All about RSI/MACD:

These indicators can help predict potential buy/sell signals:

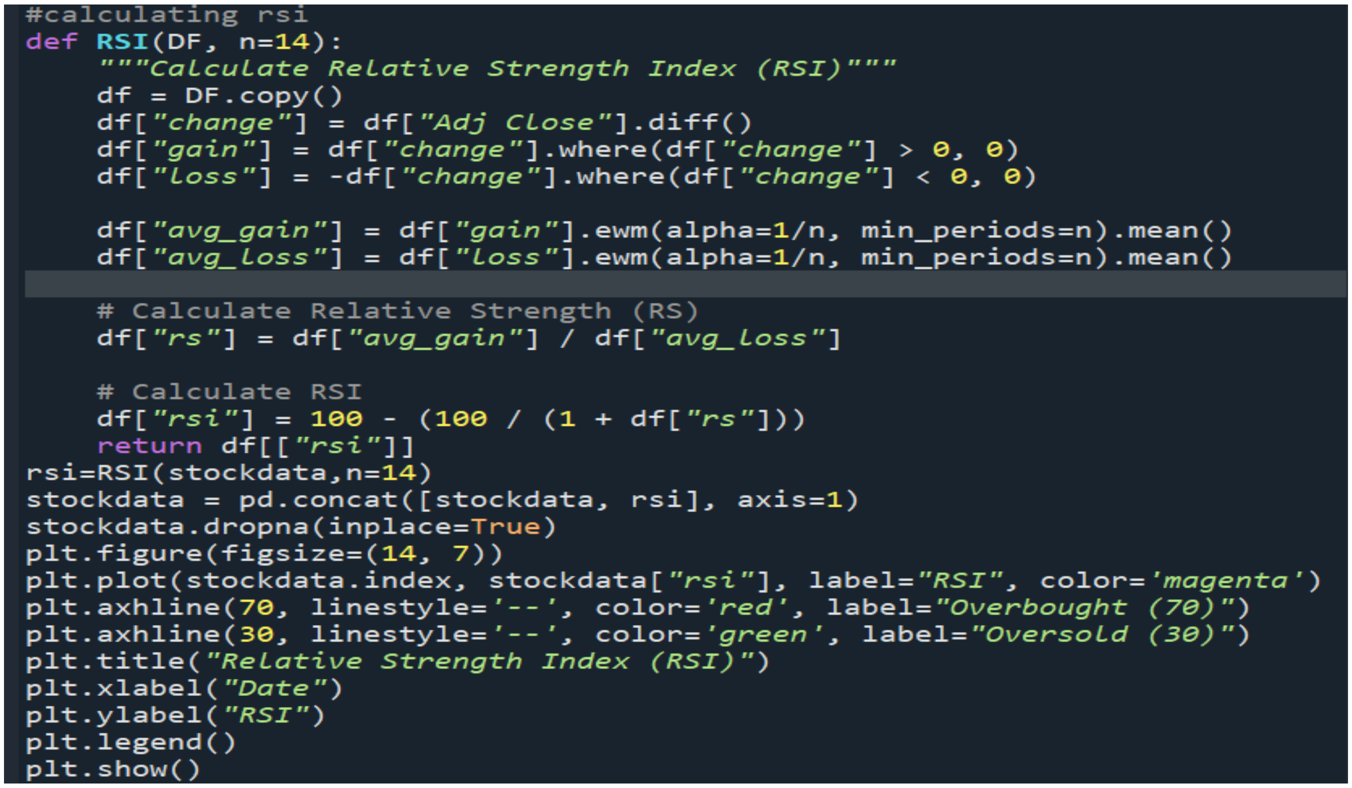
Relative Strength Index (RSI): Measures the strength of a stock's price movement to identify overbought or oversold conditions. Average gain and average loss over a look-back period (commonly 14 days).

$$RSI=100- 100/(1+RS)$$

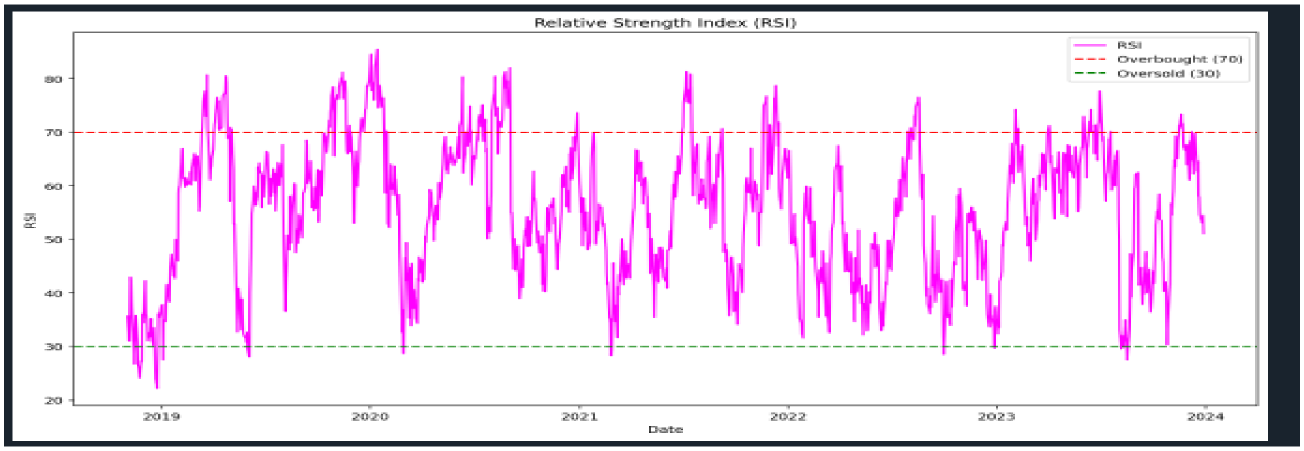
Moving Average Convergence Divergence (MACD): A trend-following momentum indicator that shows the relationship between two moving averages. The difference between the 12-day EMA and 26-day EMA

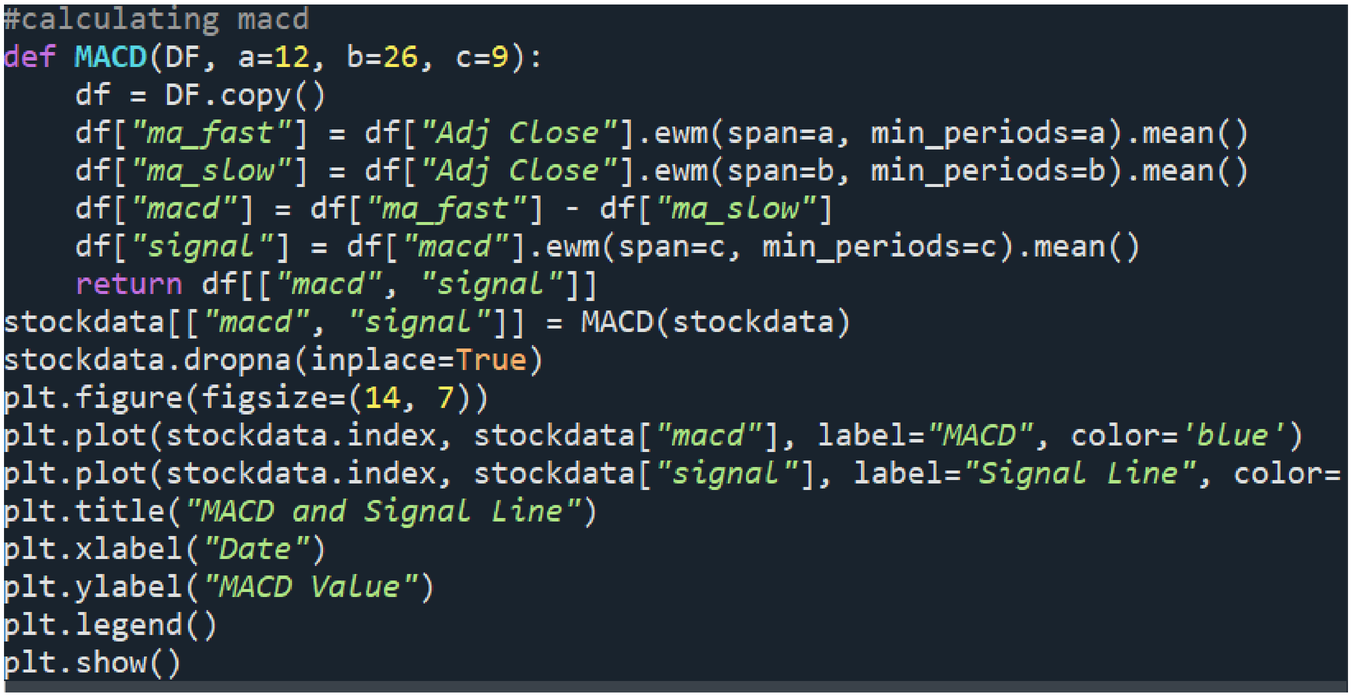
Signal Line: 9-day EMA of the MACD line.

Histogram: The difference between the MACD line and the Signal Line.

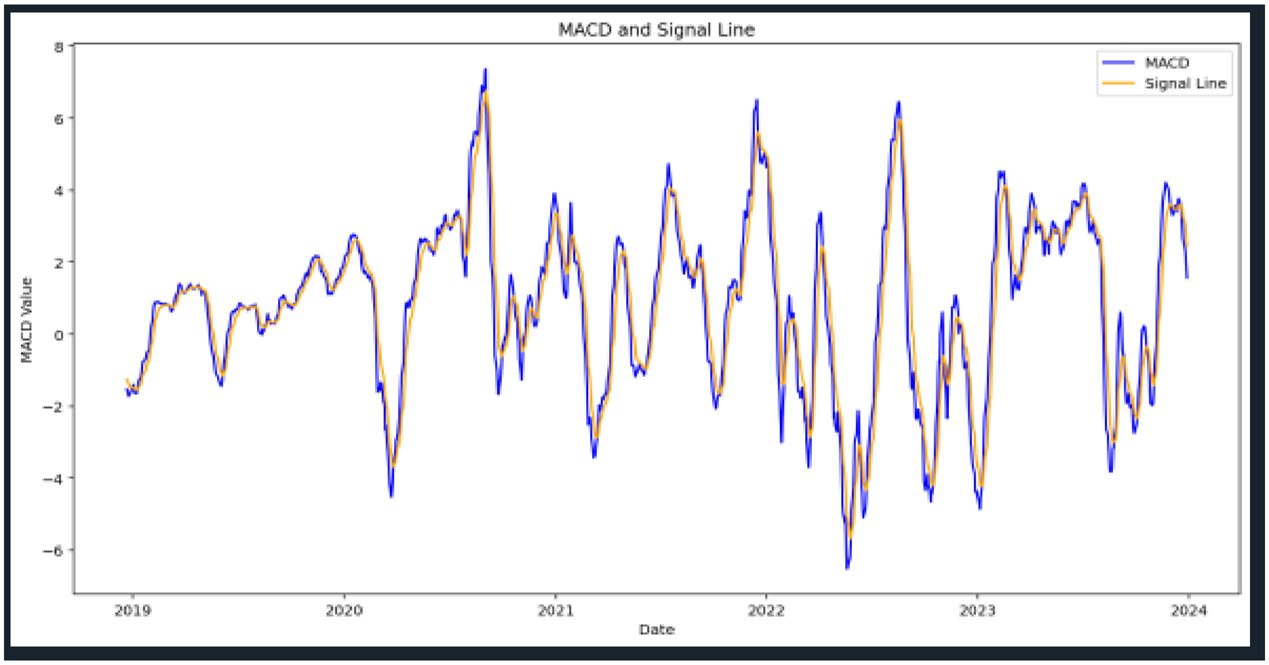


```python
#calculating rsi
def RSI(DF, n=14):
    """Calculate Relative Strength Index (RSI)"""
    df = DF.copy()
    df["change"] = df["Adj Close"].diff()
    df["gain"] = df["change"].where(df["change"] > 0, 0)
    df["loss"] = -df["change"].where(df["change"] < 0, 0)

    df["avg_gain"] = df["gain"].ewm(alpha=1/n, min_periods=n).mean()
    df["avg_loss"] = df["loss"].ewm(alpha=1/n, min_periods=n).mean()

    # Calculate Relative Strength (RS)
    df["rs"] = df["avg_gain"] / df["avg_loss"]

    # Calculate RSI
    df["rsi"] = 100 - (100 / (1 + df["rs"]))
    return df[["rsi"]]
rsi=RSI(stockdata,n=14)
stockdata = pd.concat([stockdata, rsi], axis=1)
stockdata.dropna(inplace=True)
plt.figure(figsize=(14, 7))
plt.plot(stockdata.index, stockdata["rsi"], label="RSI", color='magenta')
plt.axhline(70, linestyle='--', color='red', label="Overbought (70)")
plt.axhline(30, linestyle='--', color='green', label="Oversold (30)")
plt.title("Relative Strength Index (RSI)")
plt.xlabel("Date")
plt.ylabel("RSI")
plt.legend()
plt.show()
```





```
#calculating macd
def MACD(DF, a=12, b=26, c=9):
    df = DF.copy()
    df["ma_fast"] = df["Adj Close"].ewm(span=a, min_periods=a).mean()
    df["ma_slow"] = df["Adj Close"].ewm(span=b, min_periods=b).mean()
    df["macd"] = df["ma_fast"] - df["ma_slow"]
    df["signal"] = df["macd"].ewm(span=c, min_periods=c).mean()
    return df[["macd", "signal"]]
stockdata[["macd", "signal"]] = MACD(stockdata)
stockdata.dropna(inplace=True)
plt.figure(figsize=(14, 7))
plt.plot(stockdata.index, stockdata["macd"], label="MACD", color='blue')
plt.plot(stockdata.index, stockdata["signal"], label="Signal Line", color=
plt.title("MACD and Signal Line")
plt.xlabel("Date")
plt.ylabel("MACD Value")
plt.legend()
plt.show()
```



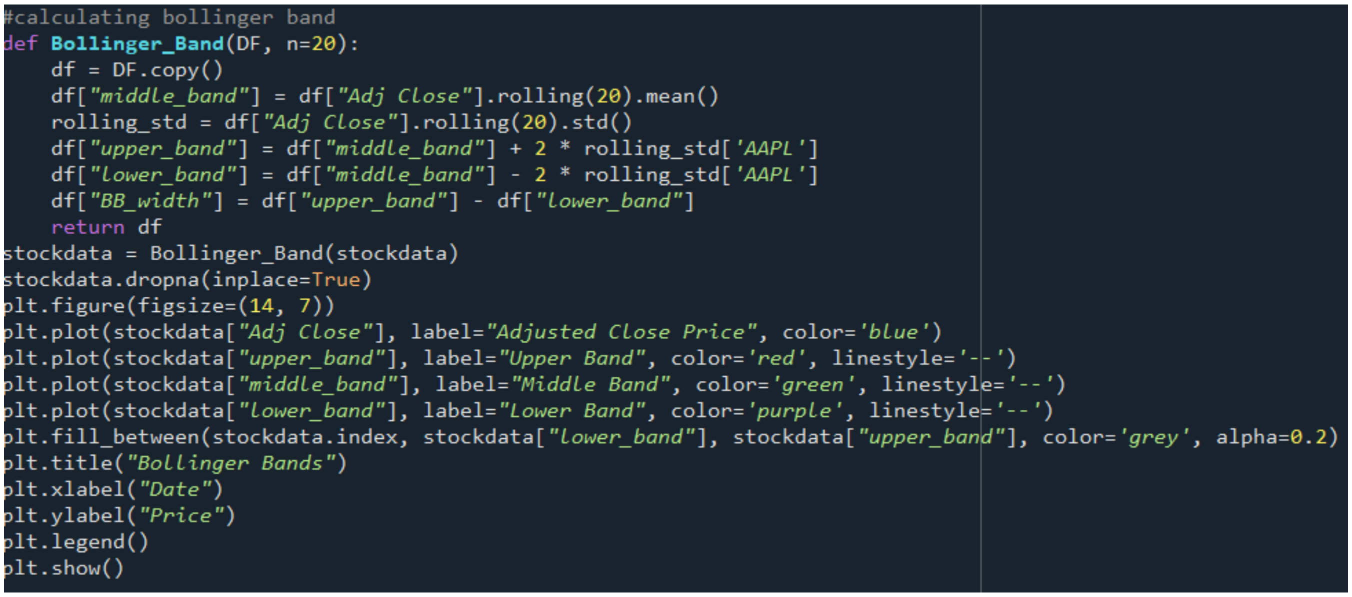
Bollinger Bands:

Visualize market volatility. Spot potential breakout zones when prices cross the bands.

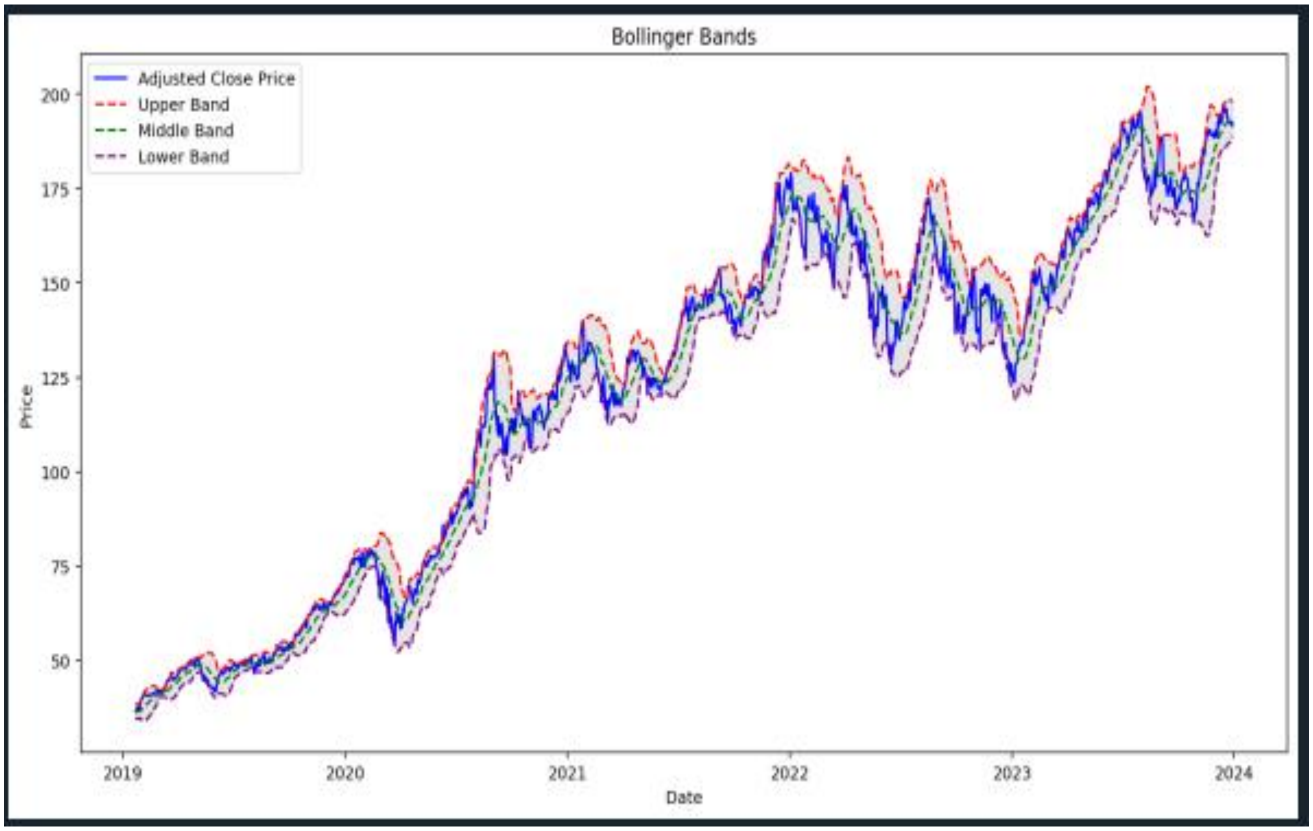
Middle Band: Simple Moving Average (SMA) over a window period (commonly 20 days).

Upper Band: Middle Band + (2 × Standard Deviation of prices in the window).

Lower Band: Middle Band – (2 × Standard Deviation of prices in the window).



```python
#calculating bollinger band
def Bollinger_Band(DF, n=20):
    df = DF.copy()
    df["middle_band"] = df["Adj Close"].rolling(20).mean()
    rolling_std = df["Adj Close"].rolling(20).std()
    df["upper_band"] = df["middle_band"] + 2 * rolling_std['AAPL']
    df["lower_band"] = df["middle_band"] - 2 * rolling_std['AAPL']
    df["BB_width"] = df["upper_band"] - df["lower_band"]
    return df
stockdata = Bollinger_Band(stockdata)
stockdata.dropna(inplace=True)
plt.figure(figsize=(14, 7))
plt.plot(stockdata["Adj Close"], label="Adjusted Close Price", color='blue')
plt.plot(stockdata["upper_band"], label="Upper Band", color='red', linestyle='--')
plt.plot(stockdata["middle_band"], label="Middle Band", color='green', linestyle='--')
plt.plot(stockdata["lower_band"], label="Lower Band", color='purple', linestyle='--')
plt.fill_between(stockdata.index, stockdata["lower_band"], stockdata["upper_band"], color='grey', alpha=0.2)
plt.title("Bollinger Bands")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.show()
```



Stochastic Oscillator

The Stochastic Oscillator measures the relative position of the closing price to its price range over a specified period, indicating momentum and potential reversals.

VWAP

The VWAP is an indicator used to measure the average price a stock has traded at throughout the day, based on both volume and price. It's often used to identify the general direction of the market.

ATR(Average True Range)

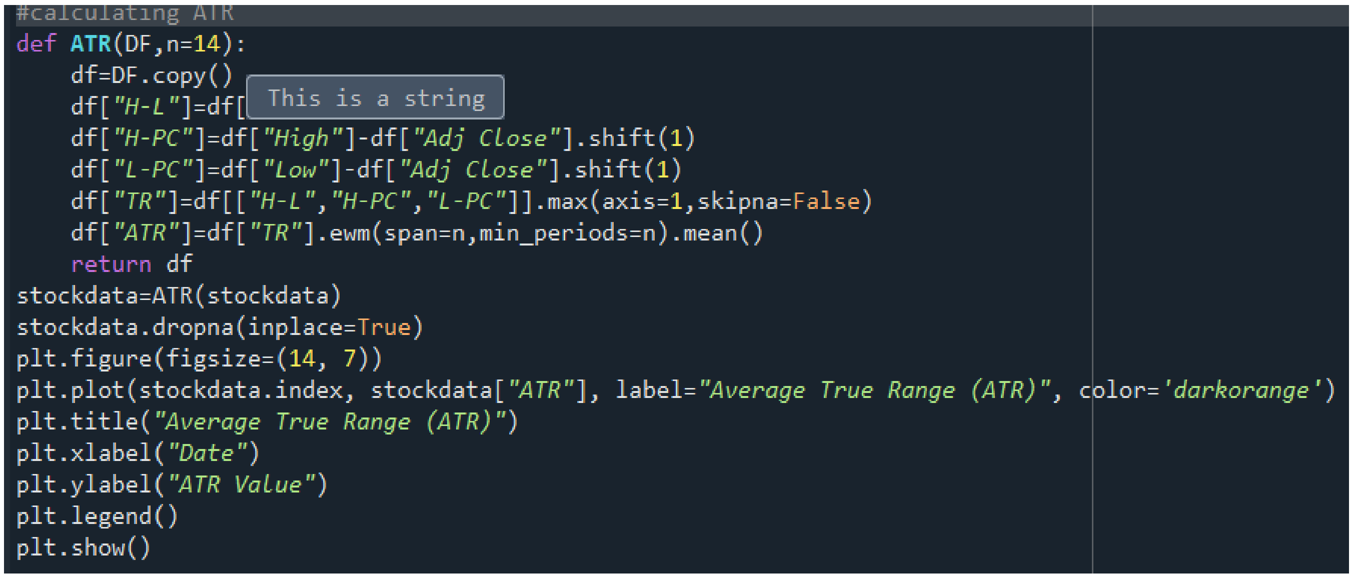
The ATR measures market volatility. It calculates the average of the true range over a specified period (usually 14 days). The true range is the maximum of the following:

The current high minus the current low.

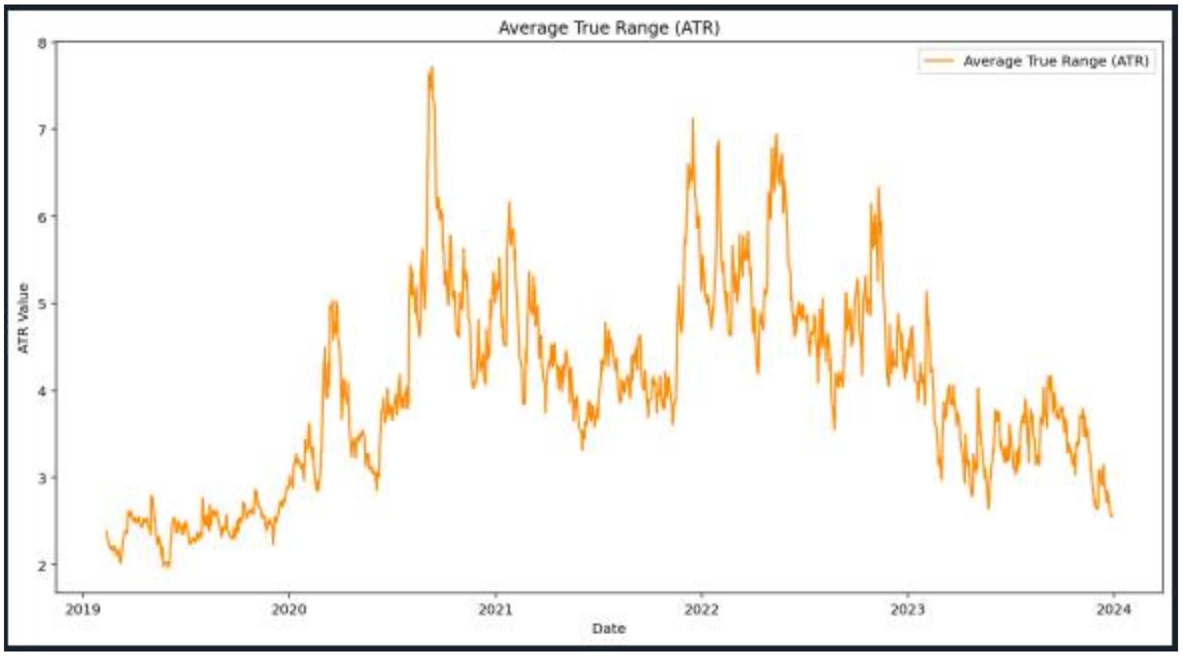
The absolute value of the current high minus the previous close.

The absolute value of the current low minus the previous close.

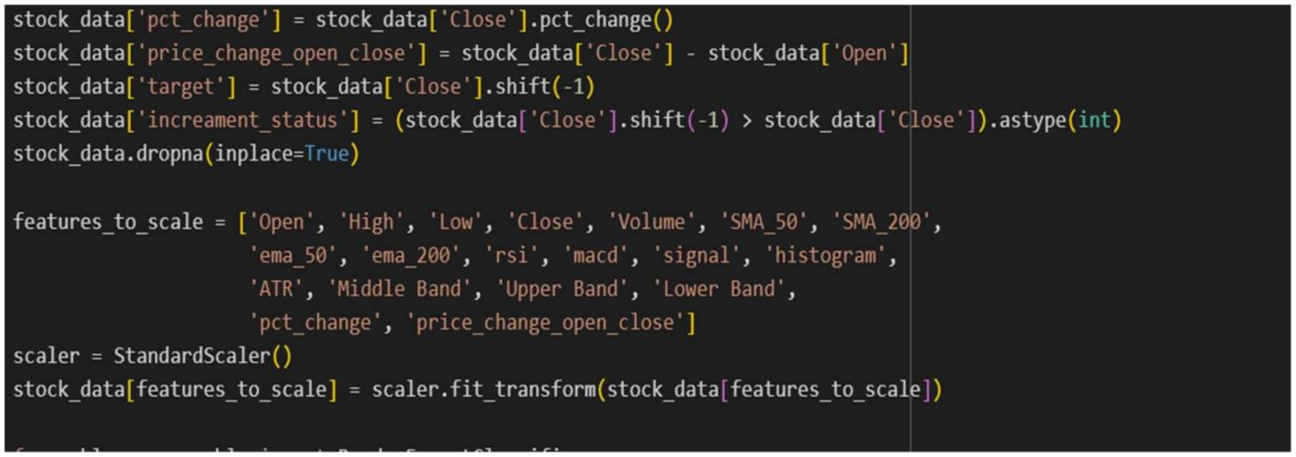
CAN TELL HOW MUCH A STOCK CAN MOVE IN ONE DAY.



```
#calculating ATR
def ATR(DF,n=14):
    df=DF.copy()
    df["H-L"]=df[ This is a string
    df["H-PC"]=df["High"]-df["Adj Close"].shift(1)
    df["L-PC"]=df["Low"]-df["Adj Close"].shift(1)
    df["TR"]=df[["H-L","H-PC","L-PC"]].max(axis=1,skipna=False)
    df["ATR"]=df["TR"].ewm(span=n,min_periods=n).mean()
    return df
stockdata=ATR(stockdata)
stockdata.dropna(inplace=True)
plt.figure(figsize=(14, 7))
plt.plot(stockdata.index, stockdata["ATR"], label="Average True Range (ATR)", color='darkorange')
plt.title("Average True Range (ATR)")
plt.xlabel("Date")
plt.ylabel("ATR Value")
plt.legend()
plt.show()
```

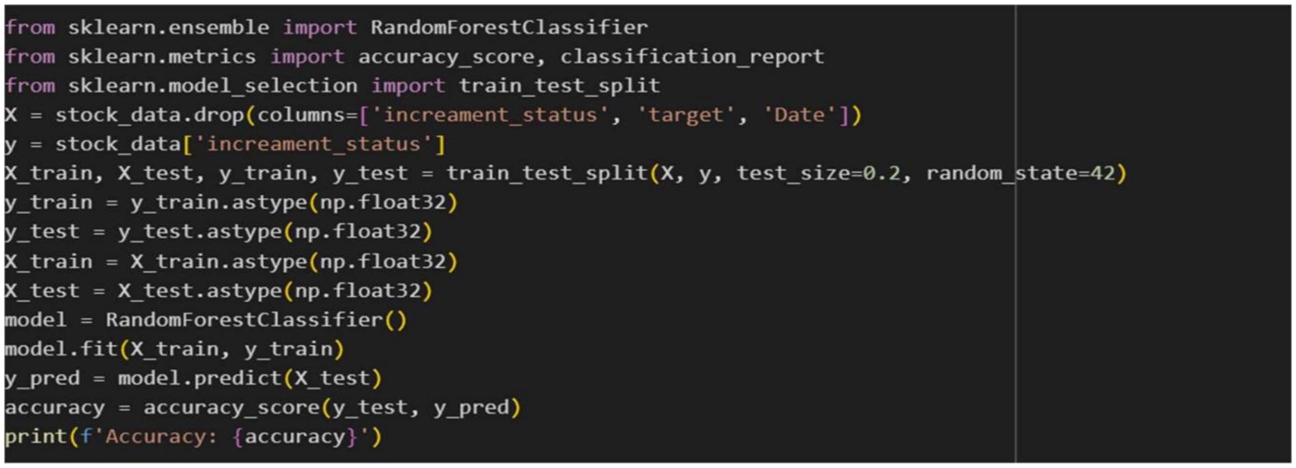


FEATURE ENGINEERING AND SCALING:

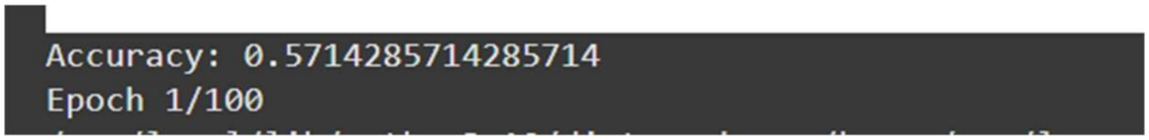


```
stock_data['pct_change'] = stock_data['Close'].pct_change()
stock_data['price_change_open_close'] = stock_data['Close'] - stock_data['Open']
stock_data['target'] = stock_data['Close'].shift(-1)
stock_data['increament_status'] = (stock_data['Close'].shift(-1) > stock_data['Close']).astype(int)
stock_data.dropna(inplace=True)

features_to_scale = ['Open', 'High', 'Low', 'Close', 'Volume', 'SMA_50', 'SMA_200',
                     'ema_50', 'ema_200', 'rsi', 'macd', 'signal', 'histogram',
                     'ATR', 'Middle Band', 'Upper Band', 'Lower Band',
                     'pct_change', 'price_change_open_close']
scaler = StandardScaler()
stock_data[features_to_scale] = scaler.fit_transform(stock_data[features_to_scale])
```

MODEL FOR CHECKING THE INCREAMENT STATUS:



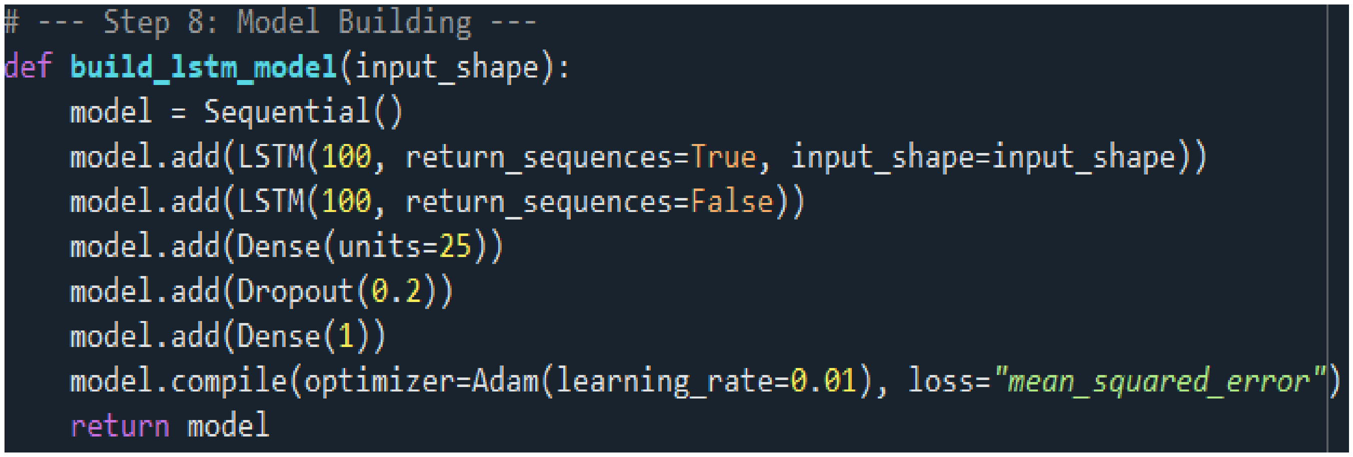
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
X = stock_data.drop(columns=['increament_status', 'target', 'Date'])
y = stock_data['increament_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y_train = y_train.astype(np.float32)
y_test = y_test.astype(np.float32)
X_train = X_train.astype(np.float32)
X_test = X_test.astype(np.float32)
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```



```
Accuracy: 0.5714285714285714
Epoch 1/100
```

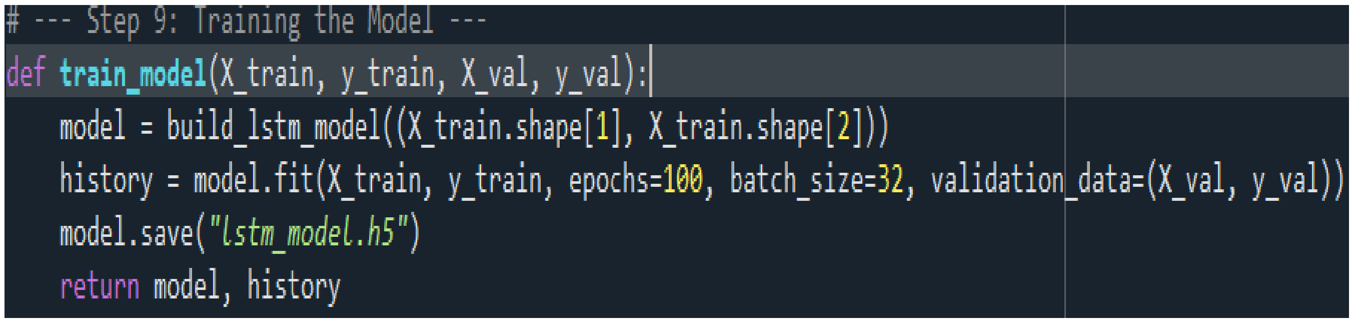
THE PERFORMANCE OF THIS CLASSIFIER WAS REALLY BAD!!

LSTM MODEL FOR THE PREDICTION OF THE STOCKS:



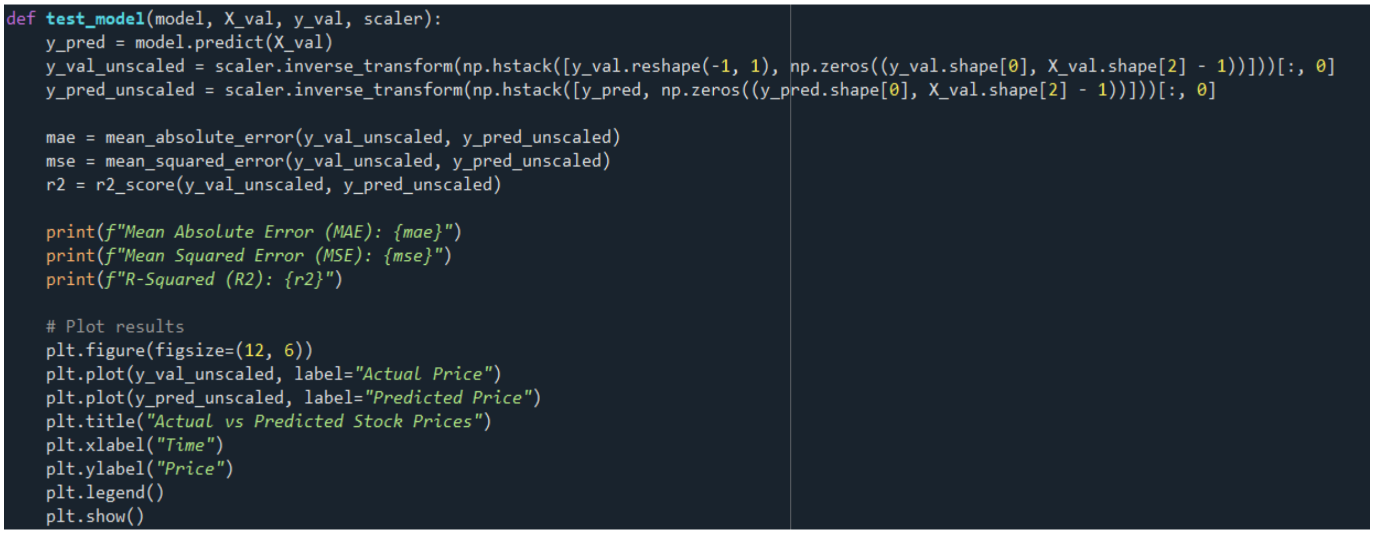
```python
# --- Step 8: Model Building ---
def build_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(100, return_sequences=True, input_shape=input_shape))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dense(units=25))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(learning_rate=0.01), loss="mean_squared_error")
    return model
```

TRAINING THE MODEL:



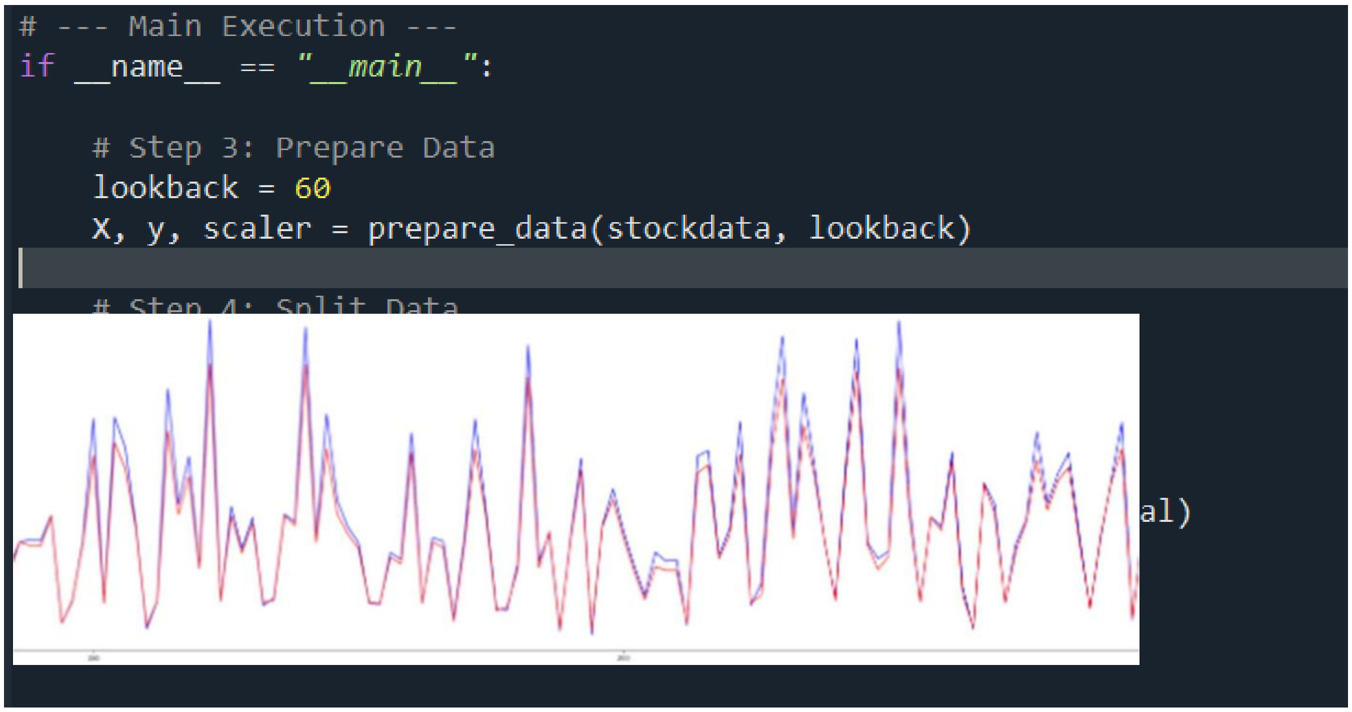
```python
# --- Step 9: Training the Model ---
def train_model(X_train, y_train, X_val, y_val):
    model = build_lstm_model((X_train.shape[1], X_train.shape[2]))
    history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val))
    model.save("lstm_model.h5")
    return model, history
```

TESTING THE MODEL:



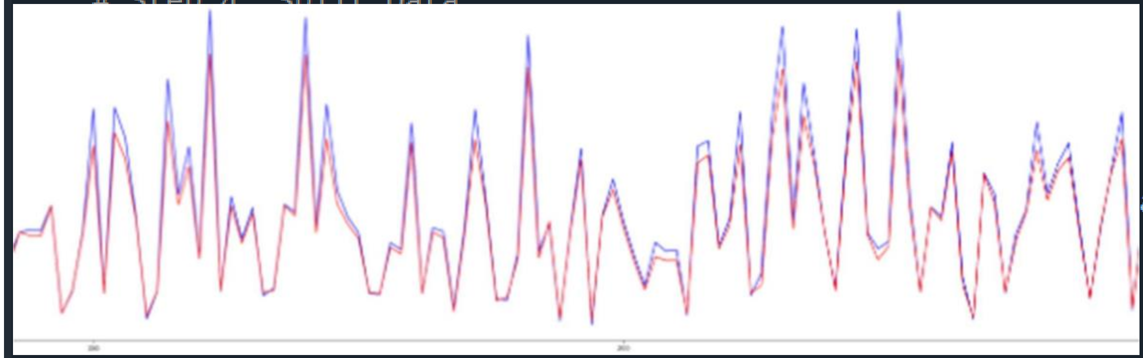
```python
def test_model(model, X_val, y_val, scaler):
    y_pred = model.predict(X_val)
    y_val_unscaled = scaler.inverse_transform(np.hstack([y_val.reshape(-1, 1), np.zeros((y_val.shape[0], X_val.shape[2] - 1))]))[:, 0]
    y_pred_unscaled = scaler.inverse_transform(np.hstack([y_pred, np.zeros((y_pred.shape[0], X_val.shape[2] - 1))]))[:, 0]

    mae = mean_absolute_error(y_val_unscaled, y_pred_unscaled)
    mse = mean_squared_error(y_val_unscaled, y_pred_unscaled)
    r2 = r2_score(y_val_unscaled, y_pred_unscaled)

    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R-Squared (R2): {r2}")

    # Plot results
    plt.figure(figsize=(12, 6))
    plt.plot(y_val_unscaled, label="Actual Price")
    plt.plot(y_pred_unscaled, label="Predicted Price")
    plt.title("Actual vs Predicted Stock Prices")
    plt.xlabel("Time")
    plt.ylabel("Price")
    plt.legend()
    plt.show()
```

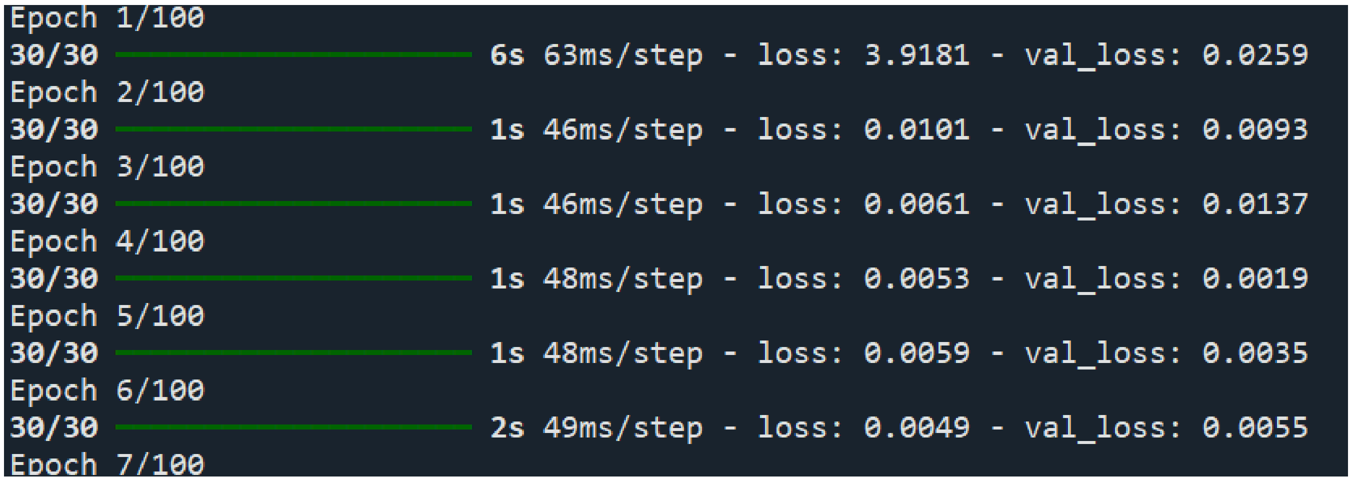
MAIN EXECUTION:



```python
# --- Main Execution ---
if __name__ == "__main__":

    # Step 3: Prepare Data
    lookback = 60
    X, y, scaler = prepare_data(stockdata, lookback)

    # Step 4: Split Data
```

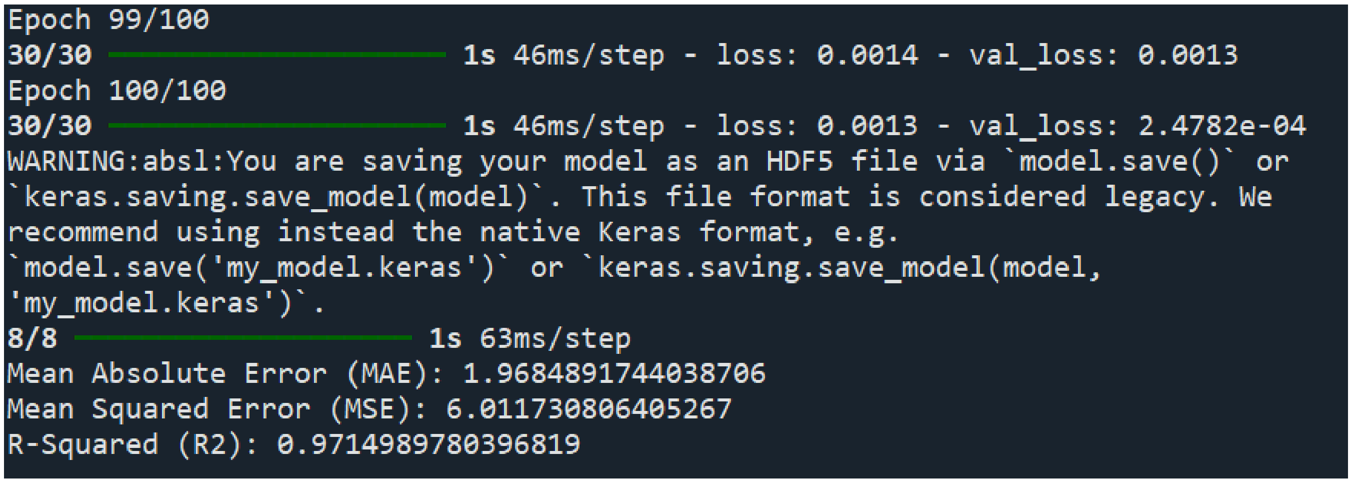
```
al)
```





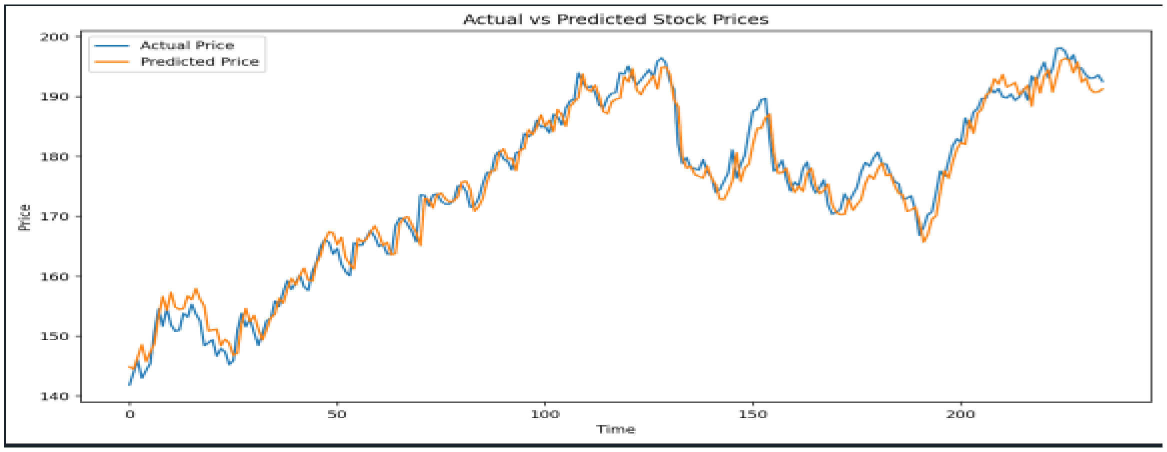
```
Epoch 1/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 63ms/step - loss: 3.9181 - val_loss: 0.0259
Epoch 2/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 0.0101 - val_loss: 0.0093
Epoch 3/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 0.0061 - val_loss: 0.0137
Epoch 4/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 0.0053 - val_loss: 0.0019
Epoch 5/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 0.0059 - val_loss: 0.0035
Epoch 6/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - loss: 0.0049 - val_loss: 0.0055
Epoch 7/100
```

ERRORS IN THE PREDICTIONS:

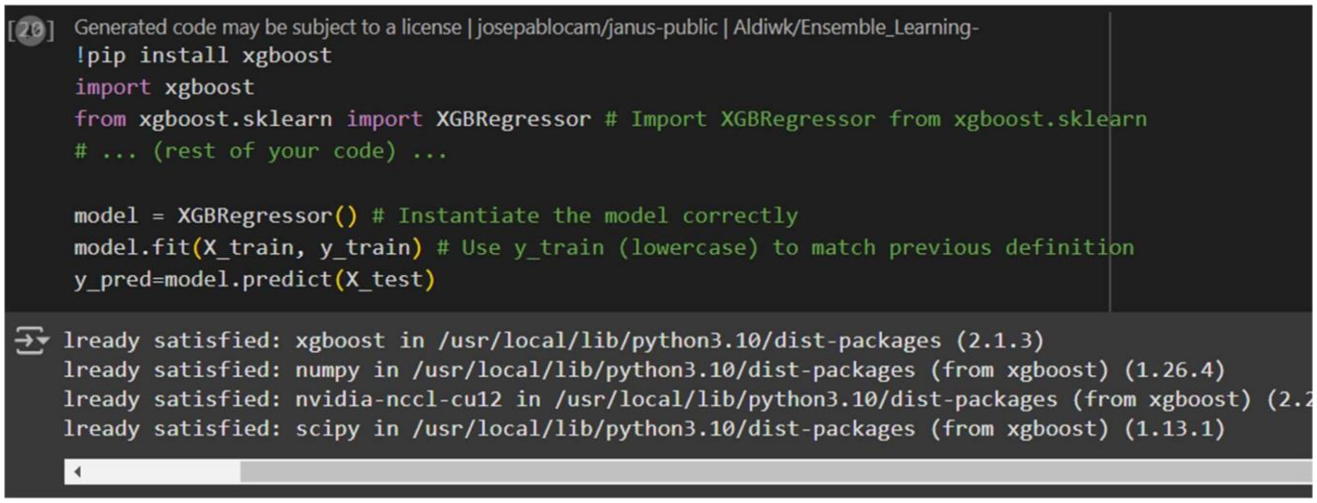


```
Epoch 99/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 0.0014 - val_loss: 0.0013
Epoch 100/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 0.0013 - val_loss: 2.4782e-04
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step
Mean Absolute Error (MAE): 1.9684891744038706
Mean Squared Error (MSE): 6.011730806405267
R-Squared (R2): 0.9714989780396819
```

PLOTTING THE GRAPH FOR THE ACTUAL VS MODEL'S PREDICTION:



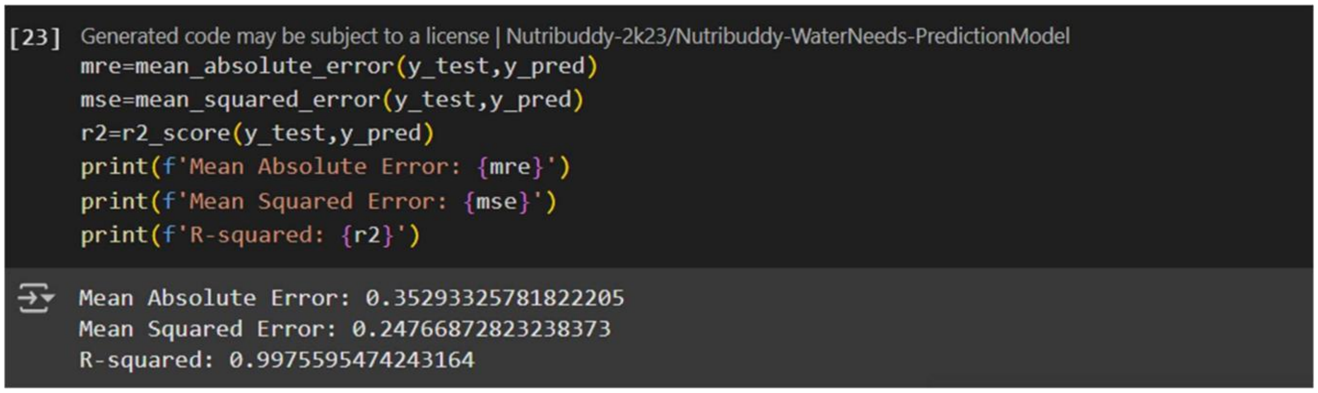
XGBOOST MODEL:



```
[20]  Generated code may be subject to a license | josepablocam/janus-public | Aldiwk/Ensemble_Learning-
      !pip install xgboost
      import xgboost
      from xgboost.sklearn import XGBRegressor # Import XGBRegressor from xgboost.sklearn
      # ... (rest of your code) ...

      model = XGBRegressor() # Instantiate the model correctly
      model.fit(X_train, y_train) # Use y_train (lowercase) to match previous definition
      y_pred=model.predict(X_test)
```

```
lready satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.3)
lready satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
lready satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.2
lready satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

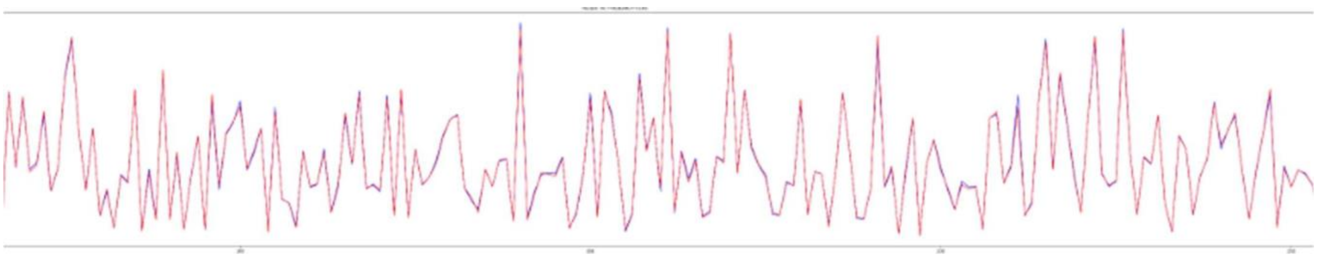
PREDICTION OF XGBOOST:



```
[23]  Generated code may be subject to a license | Nutribuddy-2k23/Nutribuddy-WaterNeeds-PredictionModel
      mre=mean_absolute_error(y_test,y_pred)
      mse=mean_squared_error(y_test,y_pred)
      r2=r2_score(y_test,y_pred)
      print(f'Mean Absolute Error: {mre}')
      print(f'Mean Squared Error: {mse}')
      print(f'R-squared: {r2}')
```

```
Mean Absolute Error: 0.35293325781822205
Mean Squared Error: 0.24766872823238373
R-squared: 0.9975595474243164
```

PLOTTING THE GRAPH FOR THE ACTUAL VS MODEL'S PREDICTION:



CONCLUSION

In this project, we aimed to predict stock price movements using machine learning models, specifically **Random Forest Classifier**, **Long Short-Term Memory (LSTM)** networks and **XGBOOST model**. We used historical stock data, including various technical indicators like **Simple Moving Average (SMA)**, **Exponential Moving Average (EMA)**, **Relative Strength Index (RSI)**, and **Moving Average Convergence Divergence (MACD)**.

- **Random Forest Classifier** was initially used to predict stock price increments (whether the stock price would increase). However, the performance of the Random Forest Classifier was not satisfactory, possibly due to the high complexity and noise present in stock market data. This led us to explore alternative models.
- **LSTM (Long Short-Term Memory)**, a type of recurrent neural network (RNN) designed for sequential data like time-series, was then employed for predicting stock prices. LSTM models, due to their ability to capture long-term dependencies and patterns in time-series data, showed better results for stock price prediction. The LSTM model outperformed the Random Forest Classifier in terms of prediction accuracy and generalization, as shown by metrics like **Mean Absolute Error (MAE)** and **Mean Squared Error (MSE)**.
- **XGBOOST model:** By leveraging feature engineering and gradient boosting, XGBoost outperformed LSTM in terms of accuracy and reliability. The model's ability to handle tabular data and static patterns effectively made it more suitable for stock price prediction. Its robustness to noise and better generalization ensured a lower error rate and improved prediction accuracy.
- Overall, **XGBoost** emerged as a better-performing model in this scenario due to the nature of the stock data and the importance of engineered features. However, **LSTM** remains a valuable tool for capturing temporal dependencies and can potentially excel in scenarios with more robust sequential patterns or larger datasets.
- **ALSO BASED ON THE ERRORS, R-SQUARED ERROR WE CAN CONCLUDE THAT LSTM WAS ABOUT 97% ACCURATE AND 99% FROM XGBOOST MODEL.**