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 & Libraries: Python, Pandas, Numpy, Matplotlib, Seaborn, TensorFlow, Keras, Scikit-Learn, Yahoo Finance
- Algorithms & 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 2012-2018)

Methodology:

1. Data Collection: Stock price data for Apple Inc. (AAPL) was downloaded from Yahoo Finance for the period from January 1, 2012, to December 31, 2018.
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 required libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
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, 2012, to December 31, 2018. I have not taken the data after it as due to pandemic the stocks were affected and model was not that good over that dataset.

```
import yfinance as yf
stock_data = yf.download('AAPL', start='2012-01-01', end='2018-12-31')
stock_data.head()
```

```
[*********************100%***********************]  1 of 1 completed
```

Price	Adj Close	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL	AAPL
Date						
2012-01-03	12.388997	14.686786	14.732143	14.607143	14.621429	302220800
2012-01-04	12.455575	14.765714	14.810000	14.617143	14.642857	260022000
2012-01-05	12.593859	14.929643	14.948214	14.738214	14.819643	271269600
2012-01-06	12.725510	15.085714	15.098214	14.972143	14.991786	318292800
2012-01-09	12.705327	15.061786	15.276786	15.048214	15.196429	394024400

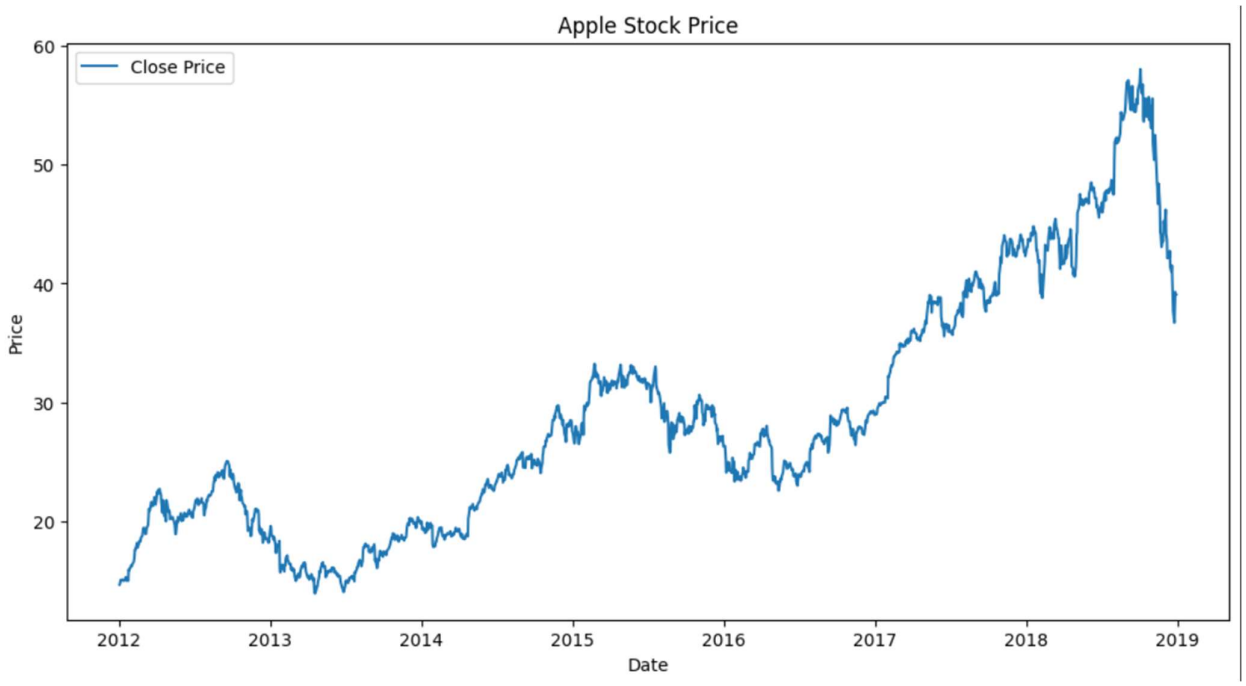
```
print(stock_data.columns)
```

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

VISUALIZING THE CLOSE PRIZES:

```
# Step 2: Visualize Closing Prices
plt.figure(figsize=(12, 6))
plt.plot(stock_data['Close'], label='Close Price')
plt.title('Apple Stock Price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

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



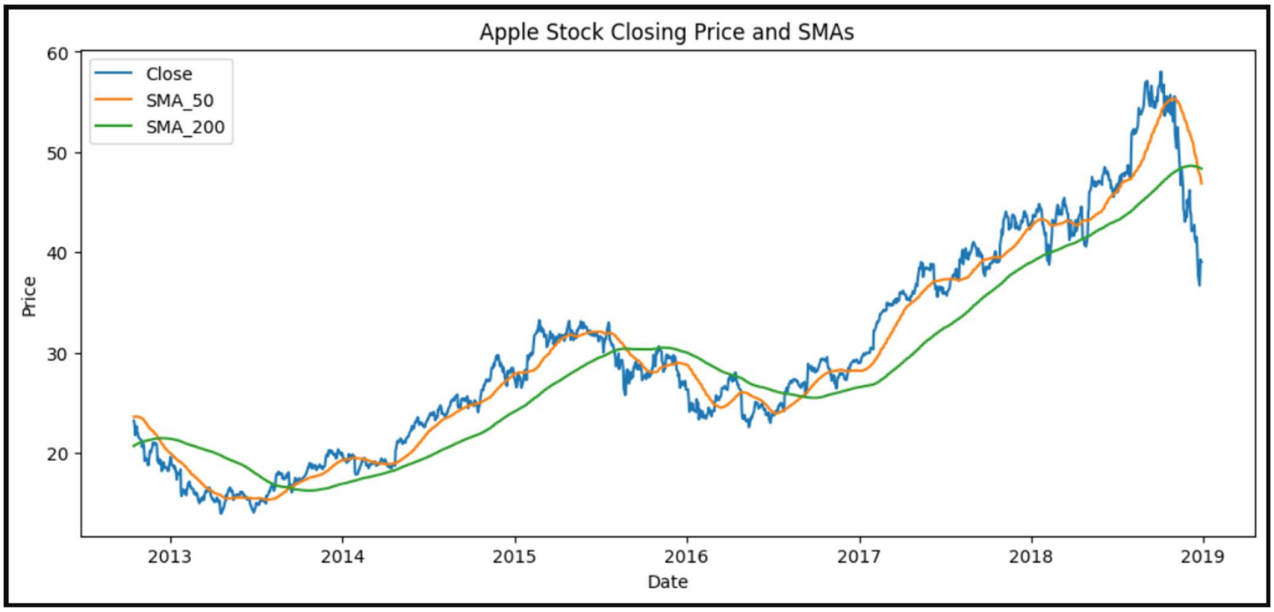
SMA AND EMA in Stock Prediction:

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

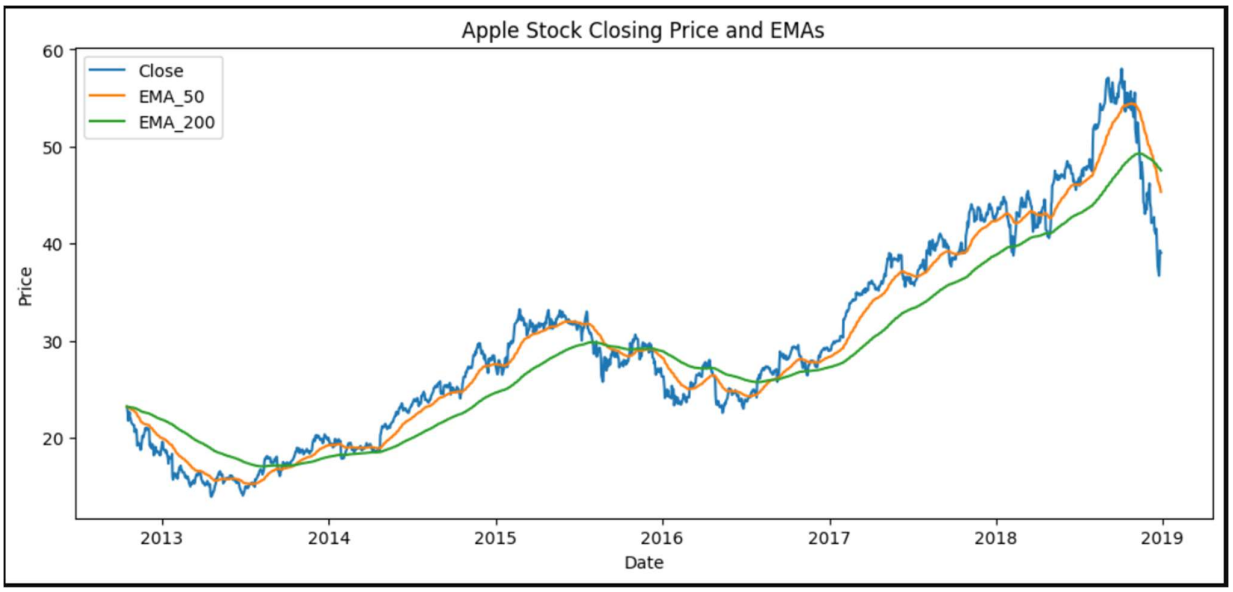
SMA(Simple Moving Average): Average of the stock prices over a specified window (e.g., 50 days).

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

```
stock_data['SMA_50'] = stock_data['Close'].rolling(window=50).mean()
stock_data['SMA_200'] = stock_data['Close'].rolling(window=200).mean()
stock_data.dropna(inplace=True)
plt.figure(figsize=(12, 5))
plt.plot(stock_data['Close'], label='Close')
plt.plot(stock_data['SMA_50'], label='SMA_50')
plt.plot(stock_data['SMA_200'], label='SMA_200')
plt.title('Apple Stock Closing Price and SMAs')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```




```
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()
```



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).

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

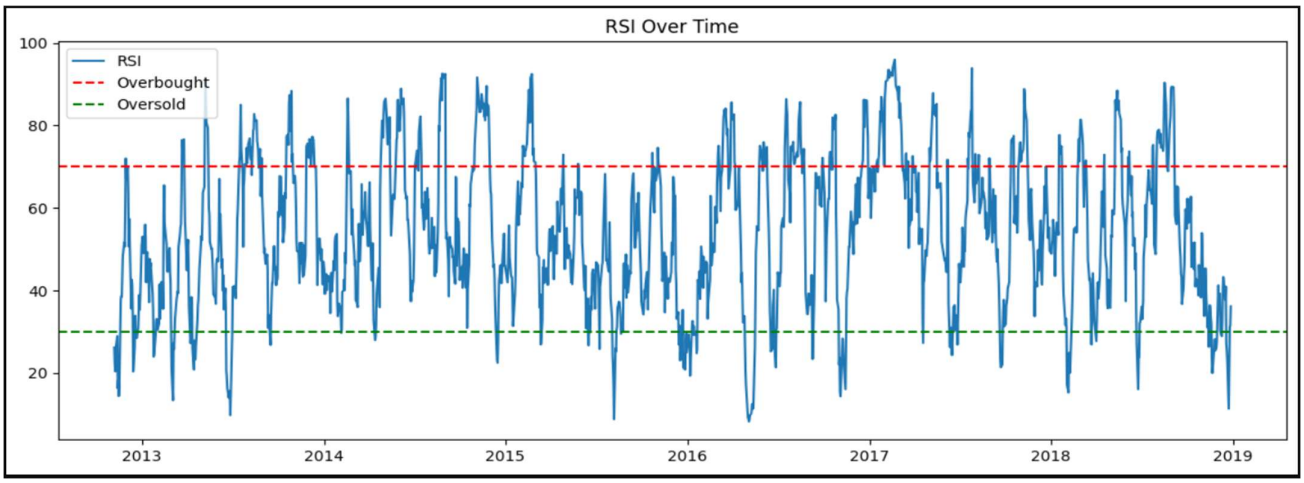
Moving Average Convergence Divergence (MACD): A trend-following momentum indicator that shows the relationship between two moving averages. The difference between the 12-day EMA and 26-day EMA

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

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

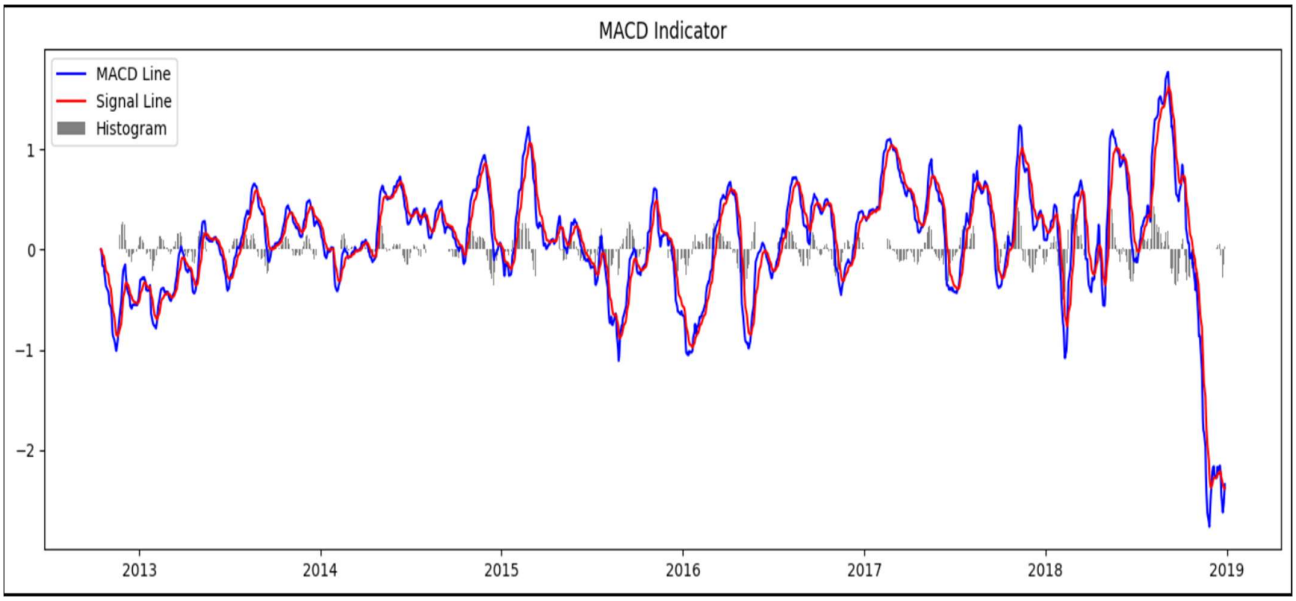
```
def rsi(stock_data,window=14):
  delta=stock_data['Close'].diff()
  gain=delta.where(delta>0,0)
  loss=-delta.where(delta<0,0)
  avg_gain=gain.rolling(window=window).mean()
  avg_loss=loss.rolling(window=window).mean()
  rs=avg_gain/avg_loss
  rsi=100-(100/(1+rs))
  stock_data['rsi']=rsi
  return stock_data
stock_data=rsi(stock_data)
stock_data.dropna(inplace=True)
stock_data[['Close','rsi']].plot(figsize=(12,5))
plt.title('Apple Stock Closing Price and RSI')
plt.axhline(70, color='red', linestyle='--', label='Overbought')
plt.axhline(30, color='green', linestyle='--', label='Oversold')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.show()
```

Calculation above of the gain, loss was done by GPT as I was very confused to calculate it .




```python
def calculate_macd(data, short_window=12, long_window=26, signal_window=9):
    data['ema_short'] = data['Close'].ewm(span=short_window, adjust=False).mean()
    data['ema_long'] = data['Close'].ewm(span=long_window, adjust=False).mean()
    data['macd'] = data['ema_short'] - data['ema_long']
    data['signal'] = data['macd'].ewm(span=signal_window, adjust=False).mean()
    data['histogram'] = data['macd'] - data['signal']
    return data

stock_data = calculate_macd(stock_data)
plt.figure(figsize=(16, 5))
plt.plot(stock_data['macd'], label='MACD Line', color='blue')
plt.plot(stock_data['signal'], label='Signal Line', color='red')
plt.bar(stock_data.index, stock_data['histogram'], label='Histogram', color='gray')
plt.title('MACD Indicator')
plt.legend()
plt.show()
```



Bollinger Bands:

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

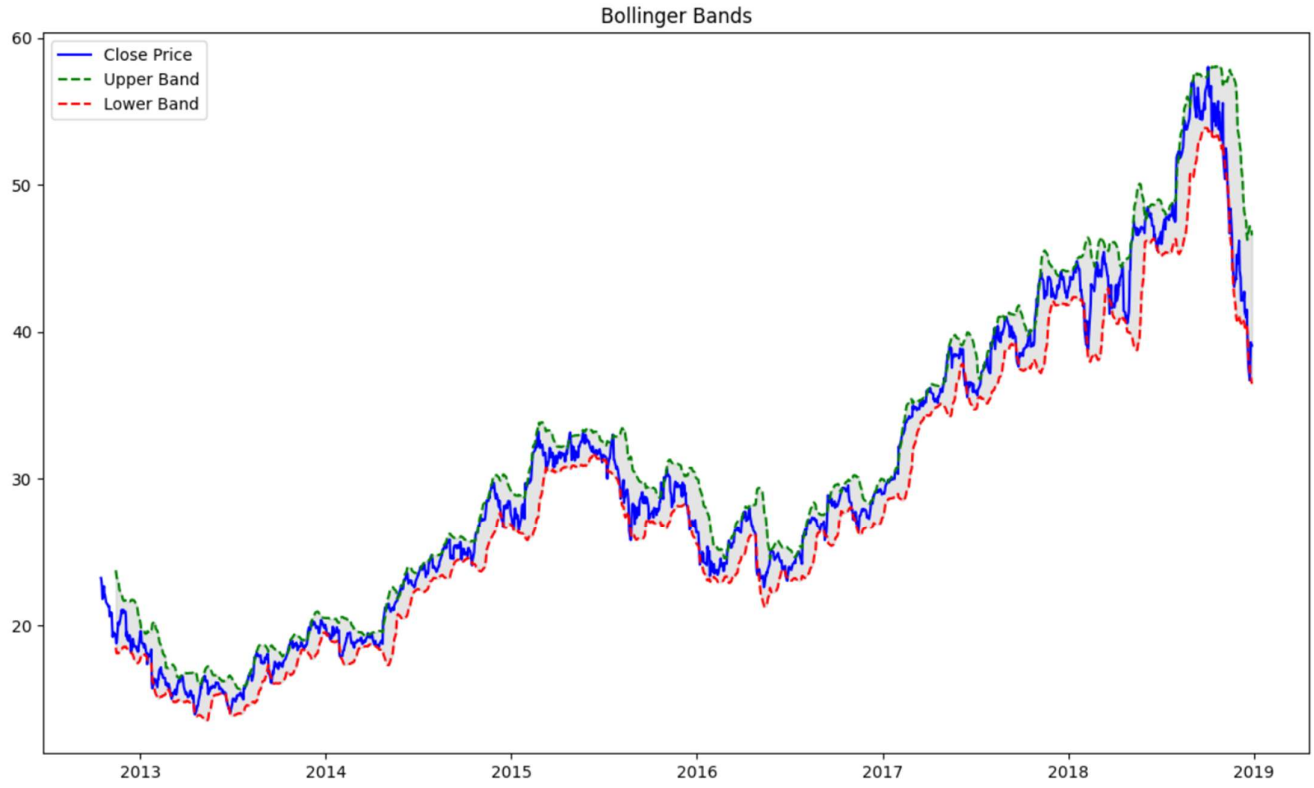
Middle Band: Simple Moving Average (SMA) over a window period (commonly 20 days).

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
def calculate_bollinger_bands(data, window=20):
    data['Middle Band'] = data['Close'].rolling(window=window).mean()
    data['Rolling Std Dev'] = data['Close'].rolling(window=window).std()
    data['Upper Band'] = data['Middle Band'] + (2 * data['Rolling Std Dev'])
    data['Lower Band'] = data['Middle Band'] - (2 * data['Rolling Std Dev'])
    data.drop(columns=['Rolling Std Dev'], inplace=True)
    return data

stock_data = calculate_bollinger_bands(stock_data)
plt.figure(figsize=(14, 8))
plt.plot(stock_data['Close'], label='Close Price', color='blue')
plt.plot(stock_data['Upper Band'], label='Upper Band', color='green', linestyle='--')
plt.plot(stock_data['Lower Band'], label='Lower Band', color='red', linestyle='--')
plt.fill_between(stock_data.index, stock_data['Upper Band'], stock_data['Lower Band'], color='gray', alpha=0.2)
plt.title('Bollinger Bands')
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.

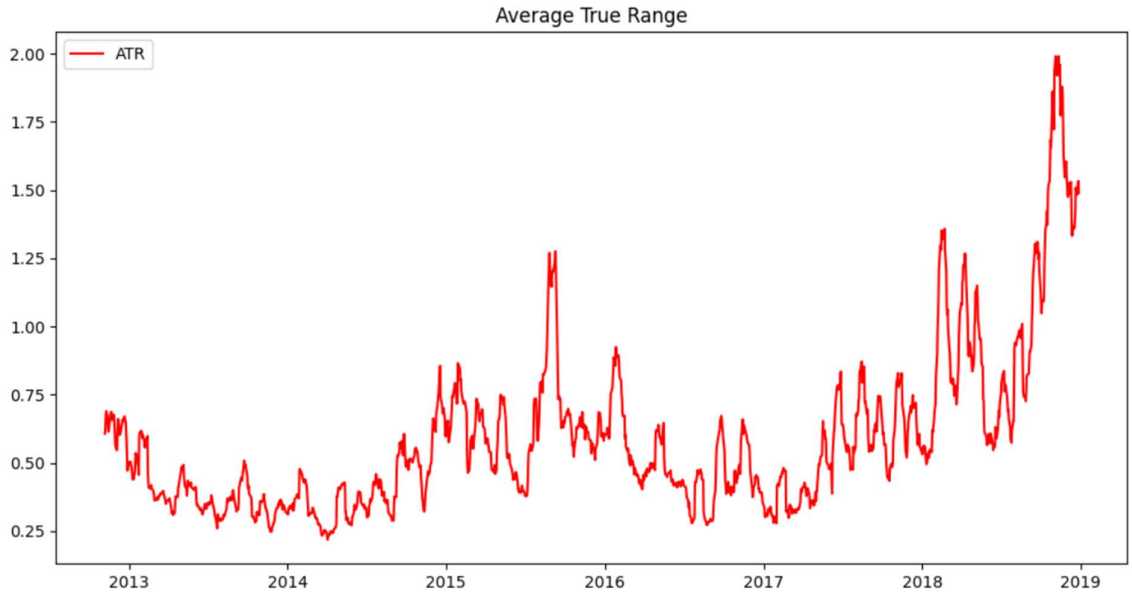
The absolute value of the current low minus the previous close.

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

```
def calculate_atr(data, window=14):
    data['High-Low'] = data['High'] - data['Low']
    data['High-Close'] = abs(data['High'] - data['Close'].shift(1))
    data['Low-Close'] = abs(data['Low'] - data['Close'].shift(1))
    data['True Range'] = data[['High-Low', 'High-Close', 'Low-Close']].max(axis=1)
    data['ATR'] = data['True Range'].rolling(window=window).mean()
    return data

stock_data = calculate_atr(stock_data)
plt.figure(figsize=(12, 6))
plt.plot(stock_data['ATR'], label='ATR', color='red')
plt.title('Average True Range')
plt.legend()
plt.show()
```

THE ABOVE CALCULATION OF ATR IS DONE BY GPT , it was confusing.



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:

```
X = stock_data.drop(columns=['increament_status', 'target', 'Date'])
y = stock_data['target']
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)

# Step 12: LSTM Model
X_train_reshaped = X_train.values.reshape(X_train.shape[0], 1, X_train.shape[1]).astype(np.float32)
X_test_reshaped = X_test.values.reshape(X_test.shape[0], 1, X_test.shape[1]).astype(np.float32)

model = Sequential()
model.add(LSTM(units=100, return_sequences=True, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
model.add(LSTM(units=100, return_sequences=False))
model.add(Dense(units=25))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')
model.fit(X_train_reshaped, y_train, epochs=100, batch_size=32)
```

PREDICTING THE STOCKS:

```
y_pred = model.predict(X_test_reshaped)
mae = 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: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

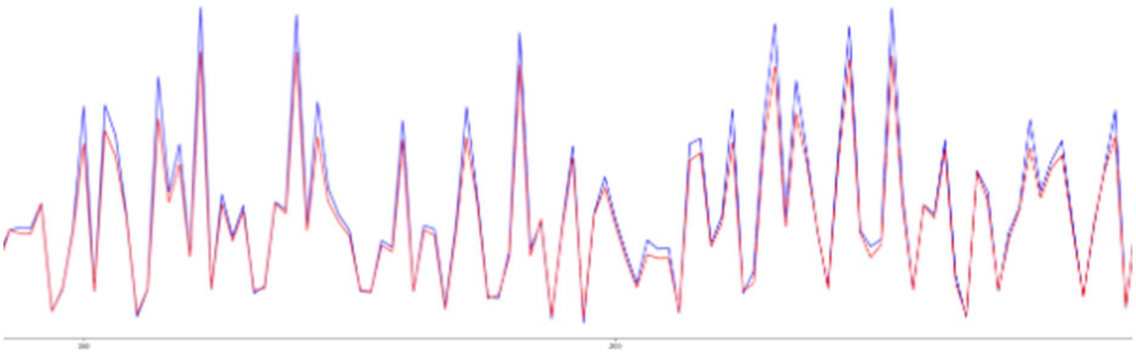
```
Epoch 1/100
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
  super().__init__(**kwargs)
39/39 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 402.3211
Epoch 2/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 31.1809
Epoch 3/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 15.1356
Epoch 4/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 15.9477
Epoch 5/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 12.6216
Epoch 6/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 13.6847
Epoch 7/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 12.4150
Epoch 8/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 12.4058
Epoch 9/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 11.8167
Epoch 10/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 11.6894
```

ERRORS IN THE PREDICTIONS:

```
Epoch 100/100
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.9395
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step
Mean Absolute Error: 1.400261640548706
Mean Squared Error: 4.421954154968262
R-squared: 0.9564269781112671
```

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

```
plt.figure(figsize=(100, 10))
plt.plot(y_test.values, label='Actual', color='blue')
plt.plot(y_pred, label='Predicted', color='red')
plt.title('Actual vs Predicted Prices')
plt.legend()
plt.show()
```



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)
```

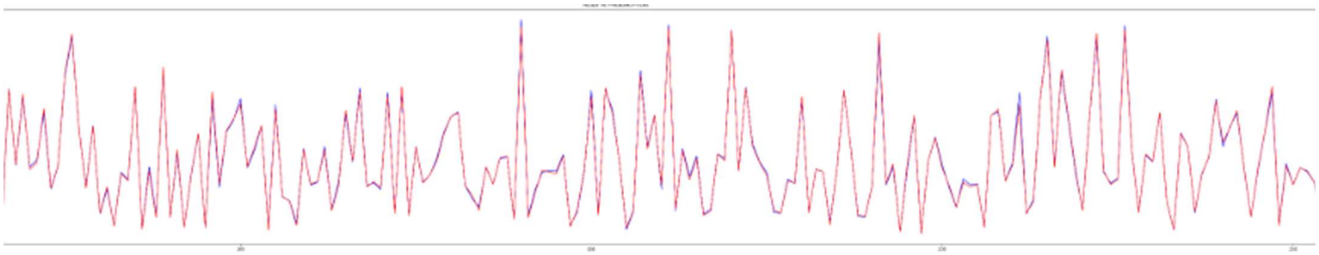
```
lready satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.3)
lready satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
lready satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.2
lready satisfied: scipy in /usr/local/lib/python3.10/dist-packages (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 95.64% ACCURATE AND 99% FROM XGBOOST MODEL.**