STOCK PRICE PREDICTION

Feature engineering is a critical step in building a stock price prediction model. The goal is to create informative input features that help our model capture meaningful patterns in the data.

Lagged Prices and Returns:

Create lag features by shifting historical prices and returns. For example, we can include features like the closing price of the previous day, week, or month.

Calculate daily or intraday returns, which can provide information about short-term price movements.

Moving Averages:

Moving averages, such as the simple moving average (SMA) and exponential moving average (EMA), can help smooth out noise and identify trends.

Volatility Indicators:

Features like historical price volatility, Bollinger Bands, or Average True Range (ATR) can capture the stock's price volatility.

Volume-Related Features:

Incorporate trading volume data, which can provide insights into market interest and liquidity.

Create moving averages or other statistics related to trading volume.

Feature Scaling and Transformation:

Standardize or normalize features to ensure they are on a similar scale, which can improve the performance of some machine learning algorithms.

+ Code + Text

```
[2] import pandas as pd
    import numpy as np
```

```
!curl -L http://prdownloads.sourceforge.net/ta-lib/ta-lib-0.4.0-src.tar.gz -O && tar xzvf ta-lib-0.4.0-src.tar.gz
!cd ta-lib && ./configure --prefix=/usr && make && make install && cd - && pip install ta-lib
```

+ Code + Text

RAM
Disk

```
[5] import talib  # Technical Analysis Library for technical indicators
```

```
[7] # Load your stock price data into a DataFrame
    # Replace 'your_data.csv' with the actual file or data source
    df = pd.read_csv('/content/MSFT.csv')
    df['Date'] = pd.to_datetime(df['Date'])
    df.set_index('Date', inplace=True)
```

```
[8] # Create lagged returns
    df['Lagged_Return'] = df['Close'].pct_change()
```

```
[9] # Create moving averages
    df['SMA_50'] = df['Close'].rolling(window=50).mean()
    df['SMA_200'] = df['Close'].rolling(window=200).mean()
```

```
[10] # Calculate RSI (Relative Strength Index)
     df['RSI'] = talib.RSI(df['Close'])
```

```
[11] # Calculate MACD (Moving Average Convergence Divergence)
     macd, signal, _ = talib.MACD(df['Close'])
     df['MACD'] = macd
     df['MACD_Signal'] = signal
```

```
[12] # Calculate Bollinger Bands
     upper, middle, lower = talib.BBANDS(df['Close'])
     df['Bollinger_Upper'] = upper
     df['Bollinger_Middle'] = middle
     df['Bollinger_Lower'] = lower
```

```
[13] # Display the first few rows of the DataFrame
     print(df.head())
```

```
                Open      High       Low     Close  Adj Close      Volume  \
Date
1986-03-13  0.088542  0.101563  0.088542  0.097222   0.062549  1031788800
1986-03-14  0.097222  0.102431  0.097222  0.100694   0.064783   308160000
1986-03-17  0.100694  0.103299  0.100694  0.102431   0.065899   133171200
1986-03-18  0.102431  0.103299  0.098958  0.099826   0.064224    67766400
1986-03-19  0.099826  0.100694  0.097222  0.098090   0.063107    47894400

            Lagged_Return  SMA_50  SMA_200  RSI  MACD  MACD_Signal  \
Date
1986-03-13            NaN     NaN      NaN  NaN   NaN          NaN
1986-03-14       0.035712     NaN      NaN  NaN   NaN          NaN
1986-03-17       0.017250     NaN      NaN  NaN   NaN          NaN
1986-03-18      -0.025432     NaN      NaN  NaN   NaN          NaN
1986-03-19      -0.017390     NaN      NaN  NaN   NaN          NaN

            Bollinger_Upper  Bollinger_Middle  Bollinger_Lower
Date
1986-03-13              NaN               NaN              NaN
1986-03-14              NaN               NaN              NaN
1986-03-17              NaN               NaN              NaN
1986-03-18              NaN               NaN              NaN
1986-03-19          0.10336          0.099653         0.095945
```

```
[14] import matplotlib.pyplot as plt

     # Plot the stock's closing price and lagged returns
     plt.figure(figsize=(12, 6))
     plt.subplot(2, 1, 1)
     plt.plot(df.index, df['Close'], label='Closing Price', color='blue')
     plt.title('Stock Price and Lagged Returns')
     plt.ylabel('Price')
     plt.legend(loc='best')
```

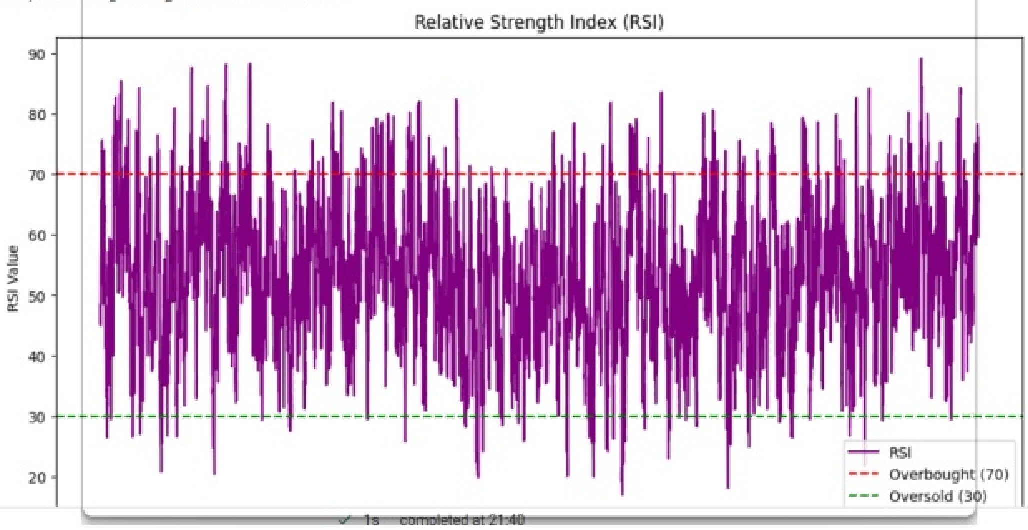
```
[14]
# Plot the stock's closing price and lagged returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(df.index, df['Close'], label='Closing Price', color='blue')
plt.title('Stock Price and Lagged Returns')
plt.ylabel('Price')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7b0d58182e30>
```


```
# Plot RSI
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['RSI'], label='RSI', color='purple')
plt.axhline(y=70, color='r', linestyle='--', label='Overbought (70)')
plt.axhline(y=30, color='g', linestyle='--', label='Oversold (30)')
plt.title('Relative Strength Index (RSI)')
plt.xlabel('Date')
plt.ylabel('RSI Value')
plt.legend(loc='best')
```

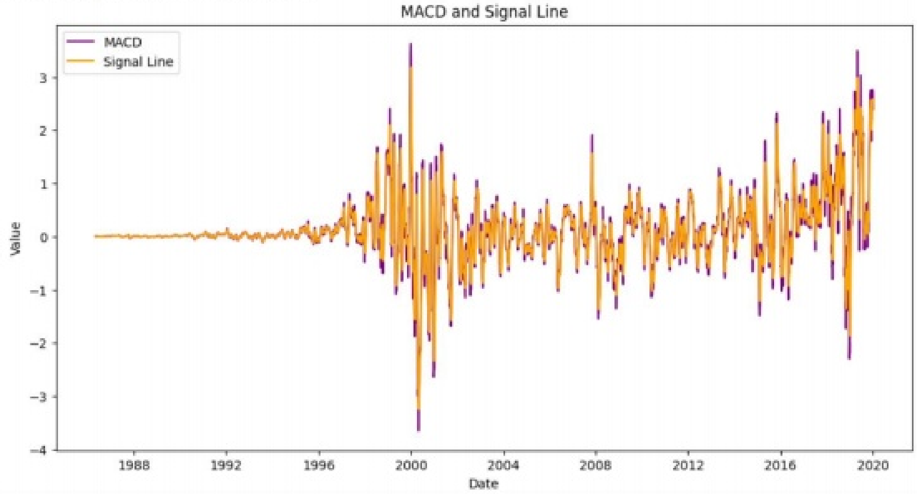

```
# Plot RSI
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['RSI'], label='RSI', color='purple')
plt.axhline(y=70, color='r', linestyle='--', label='Overbought (70)')
plt.axhline(y=30, color='g', linestyle='--', label='Oversold (30)')
plt.title('Relative Strength Index (RSI)')
plt.xlabel('Date')
plt.ylabel('RSI Value')
plt.legend(loc='best')
```


```
<matplotlib.legend.Legend at 0x7b0d4e5e3d30>
```



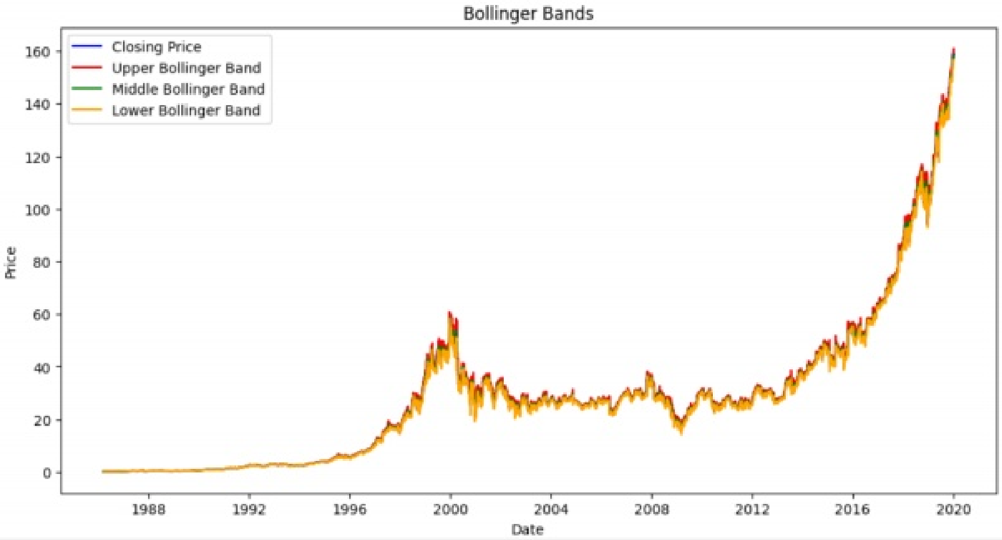
```python
plt.plot(df.index, df['MACD'], label='MACD', color='purple')
plt.plot(df.index, df['MACD_Signal'], label='Signal Line', color='orange')
plt.title('MACD and Signal Line')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7b0d50845ba0>
```




```python
plt.plot(df.index, df['Bollinger_Lower'], label='Lower Bollinger Band', color='orange')
plt.title('Bollinger Bands')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend(loc='best')

plt.show()
```



Stock Price and Lagged Returns:

The first subplot shows the stock's closing price (in blue).

The second subplot displays the lagged returns (in red), indicating the daily percentage change in the closing price.

Relative Strength Index (RSI):

The RSI plot (in purple) is a momentum oscillator that ranges from 0 to 100.

Red dashed line indicates overbought conditions (RSI > 70), and green dashed line indicates oversold conditions (RSI < 30).

MACD and Signal Line:

The plot shows the MACD (in purple) and the MACD signal line (in orange).

MACD can help identify trend changes and momentum.

Bollinger Bands:

This plot displays the closing price (in blue) along with the upper Bollinger Band (in red), middle Bollinger Band (in green), and lower Bollinger Band (in orange).

Bollinger Bands are used to measure volatility and potential price reversals.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
```

```python
# Load the dataset
data = pd.read_csv # Load the dataset
data = pd.read_csv('/content/MSFT.csv')
```

```python
# Perform exploratory data analysis
print(data.head())  # Display the first few rows of the dataset
print(data.info())  # Get information about the dataset
```

```
         Date      Open      High       Low     Close  Adj Close      Volume
0  1986-03-13  0.088542  0.101563  0.088542  0.097222   0.062549  1031788800
1  1986-03-14  0.097222  0.102431  0.097222  0.100694   0.064783   308160000
2  1986-03-17  0.100694  0.103299  0.100694  0.102431   0.065899   133171200
3  1986-03-18  0.102431  0.103299  0.098958  0.099826   0.064224    67766400
4  1986-03-19  0.099826  0.100694  0.097222  0.098090   0.063107    47894400
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8525 entries, 0 to 8524
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       8525 non-null   object
 1   Open       8525 non-null   float64
 2   High       8525 non-null   float64
 3   Low        8525 non-null   float64
 4   Close      8525 non-null   float64
 5   Adj Close  8525 non-null   float64
 6   Volume     8525 non-null   int64
dtypes: float64(5), int64(1), object(1)
```

Tools Help All changes saved Comment

+ Code + Text

```python
# Data preprocessing
# Handle missing values, feature selection, scaling, etc.

# Split the data into training and testing sets
X = data[['High', 'Low', 'Close']]  # Select relevant features
y = data['Close']  # The target variable (stock price)
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Standardize the data (optional)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
# Make predictions on the test set
predictions = model.predict(X_test)
```

```python
# Evaluate the model's performance (e.g., calculate Mean Squared Error)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 5.238262710259928e-29
```

Load your stock price data, rename the columns and split the data into training and test sets.

Initialize the model and train it using the training data.

Create a dataframe for future dates using model.make_future_dataframe(), where the number of future periods is set to the length of the test data.

Generate forecasts with model.predict(future).

Extract the predicted values for the test set.

Evaluate the model using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2) score.