another-copy-of-all-models-nasdaq

November 10, 2024

[ ]:
```
import pandas as pd
```

[ ]:
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import sklearn.metrics
```

[ ]:
```
NASDAQ=pd.read_csv('/content/nasdaq.csv')
```

[ ]:
```
NASDAQ=NASDAQ.iloc[:,0:11]
```

[ ]:
```
NASDAQ.shape
```

[ ]: (2684, 11)

[ ]:
```
NASDAQ = NASDAQ.dropna(subset=['MA_50'])
```

[ ]:
```
NASDAQ.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2635 entries, 49 to 2683
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          2635 non-null   object
 1   Open          2635 non-null   float64
 2   High          2635 non-null   float64
 3   Low           2635 non-null   float64
 4   Close         2635 non-null   float64
 5   Adj Close     2635 non-null   float64
 6   Volume        2635 non-null   int64
 7   MA_50         2635 non-null   float64
 8   Daily_Return  2635 non-null   float64
 9   Volatility    2635 non-null   float64
 10  Change        2635 non-null   float64
dtypes: float64(9), int64(1), object(1)
memory usage: 247.0+ KB
```

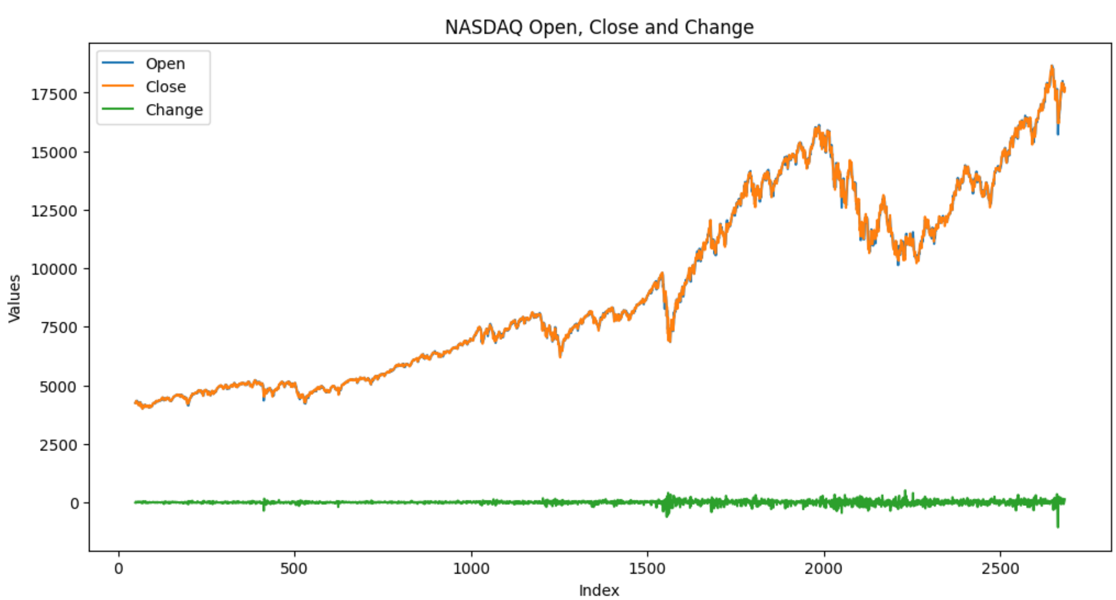
[ ]:
```
NASDAQ['Date'] = pd.to_datetime(NASDAQ['Date']).dt.date
```

[ ]:
```
NASDAQ.head()
```

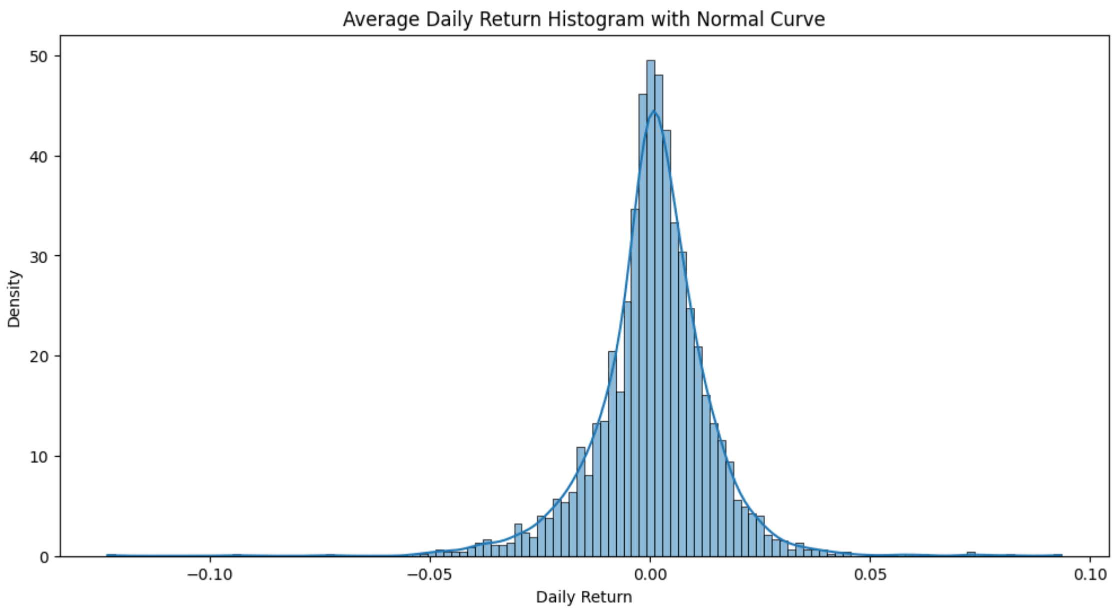
[ ]:
```
          Date         Open         High          Low        Close  \
49  2014-03-14  4250.450195  4272.339844  4241.939941  4245.399902
50  2014-03-17  4274.220215  4301.279785  4273.009766  4279.950195
51  2014-03-18  4286.220215  4334.660156  4284.109863  4333.310059
52  2014-03-19  4331.459961  4334.299805  4283.540039  4307.600098
53  2014-03-20  4297.990234  4329.609863  4287.410156  4319.290039

      Adj Close      Volume        MA_50  Daily_Return  Volatility     Change
49  4245.399902  2196890000  4203.550991     -0.003525    0.007043  -9.969727
50  4279.950195  1810410000  4206.288599      0.008138    0.006970  28.820313
51  4333.310059  1962890000  4210.316597      0.012467    0.007447   6.270020
52  4307.600098  1992750000  4214.194995     -0.005933    0.007472  -1.850098
53  4319.290039  1847270000  4217.517192      0.002714    0.007221  -9.609864
```

[ ]:
```
gdata = NASDAQ[['Open', 'Close', 'Change']]
```

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

gdata.plot(figsize=(12, 6))
plt.title('NASDAQ Open, Close and Change')
plt.xlabel('Index')
plt.ylabel('Values')
plt.legend()
plt.show()
```

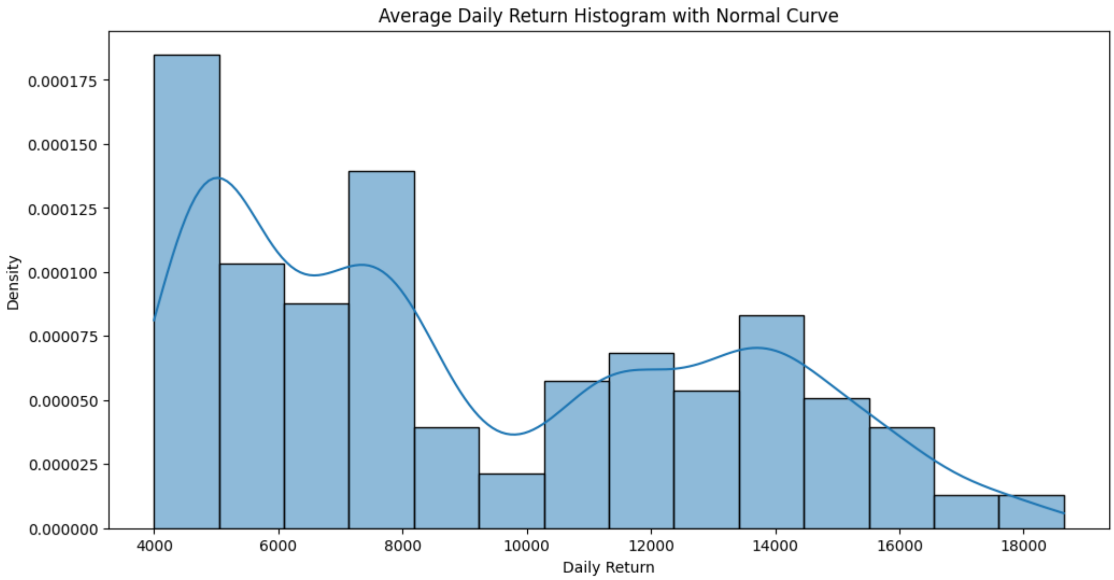


[ ]:
```
import numpy as np
import seaborn as sns
from scipy.stats import norm

plt.figure(figsize=(12, 6))
sns.histplot(NASDAQ['Daily_Return'].dropna(), kde=True, stat='density')

mu, std = norm.fit(NASDAQ['Daily_Return'].dropna())

xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
plt.title('Average Daily Return Histogram with Normal Curve')
plt.xlabel('Daily Return')
plt.ylabel('Density')
plt.show()
```



[ ]:
```
import numpy as np
import seaborn as sns
from scipy.stats import norm

plt.figure(figsize=(12, 6))
sns.histplot(NASDAQ['Close'].dropna(), kde=True, stat='density')

mu, std = norm.fit(NASDAQ['Close'].dropna())

xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
```

```
plt.title('Average Daily Return Histogram with Normal Curve')
plt.xlabel('Daily Return')
plt.ylabel('Density')
plt.show()
```



[ ]:
```
# prompt: Check outliers in NASDAQ. give count

def count_outliers_iqr(data):
  """Counts the number of outliers in a DataFrame using the IQR method.

  Args:
    data: A pandas DataFrame.

  Returns:
    A dictionary where keys are column names and values are the number of␣
↪outliers in each column.
  """
  outlier_counts = {}
  for column in data.columns:
    if pd.api.types.is_numeric_dtype(data[column]):
      Q1 = data[column].quantile(0.25)
      Q3 = data[column].quantile(0.75)
      IQR = Q3 - Q1
      lower_bound = Q1 - 1.5 * IQR
      upper_bound = Q3 + 1.5 * IQR
      outliers = data[(data[column] < lower_bound) | (data[column] >␣
↪upper_bound)]
```

```
      outlier_counts[column] = len(outliers)
  return outlier_counts


outlier_counts = count_outliers_iqr(NASDAQ)
print("Number of Outliers for Each Attribute:")
for column, count in outlier_counts.items():
  print(f"{column}: {count}")
```

```
Number of Outliers for Each Attribute:
Open: 0
High: 0
Low: 0
Close: 0
Adj Close: 0
Volume: 9
MA_50: 0
Daily_Return: 163
Volatility: 56
Change: 303
```

[ ]:
```
def calculate_outlier_percentage(data):
  outlier_percentages = {}
  for column in data.columns:
    if pd.api.types.is_numeric_dtype(data[column]):
      Q1 = data[column].quantile(0.25)
      Q3 = data[column].quantile(0.75)
      IQR = Q3 - Q1
      lower_bound = Q1 - 1.5 * IQR
      upper_bound = Q3 + 1.5 * IQR
      outliers = data[(data[column] < lower_bound) | (data[column] >␣
↪upper_bound)]
      outlier_percentage = (len(outliers) / len(data)) * 100 if len(data) > 0␣
↪else 0
      outlier_percentages[column] = outlier_percentage

  return outlier_percentages
outlier_percentages = calculate_outlier_percentage(NASDAQ)
print("Outlier Percentages for Each Attribute:")
for column, percentage in outlier_percentages.items():
  print(f"{column}: {percentage:.2f}%")
```

```
Outlier Percentages for Each Attribute:
Open: 0.00%
High: 0.00%
Low: 0.00%
Close: 0.00%
```

```
Adj Close: 0.00%
Volume: 0.34%
MA_50: 0.00%
Daily_Return: 6.19%
Volatility: 2.13%
Change: 11.50%
```

[ ]:
```
def remove_outliers_iqr(df):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile) for each column
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)

    # Calculate the Interquartile Range (IQR)
    IQR = Q3 - Q1

    # Define the lower and upper bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove outliers
    df_no_outliers_iqr = df[~((df < lower_bound) | (df > upper_bound)).
  ↪any(axis=1)]
    return df_no_outliers_iqr
NSEI = remove_outliers_iqr(NASDAQ)
```

ARIMA

[ ]:
```
#arima
NASDAQ['Close'] = np.log(NASDAQ['Close'])
```

[ ]:
```
NASDAQ.head()
```

[ ]:
```
          Date         Open         High          Low     Close    Adj Close  \
49  2014-03-14  4250.450195  4272.339844  4241.939941  8.353591  4245.399902
50  2014-03-17  4274.220215  4301.279785  4273.009766  8.361697  4279.950195
51  2014-03-18  4286.220215  4334.660156  4284.109863  8.374087  4333.310059
52  2014-03-19  4331.459961  4334.299805  4283.540039  8.368136  4307.600098
53  2014-03-20  4297.990234  4329.609863  4287.410156  8.370846  4319.290039

        Volume        MA_50  Daily_Return  Volatility     Change
49  2196890000  4203.550991     -0.003525    0.007043  -9.969727
50  1810410000  4206.288599      0.008138    0.006970  28.820313
51  1962890000  4210.316597      0.012467    0.007447   6.270020
52  1992750000  4214.194995     -0.005933    0.007472  -1.850098
53  1847270000  4217.517192      0.002714    0.007221  -9.609864
```

[ ]:
```
!pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86
_64.manylinux_2_28_x86_64.whl.metadata (7.8 kB)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.11)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (2.2.2)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.5.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.13.1)
Requirement already satisfied: statsmodels>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (2.2.3)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (75.1.0)
Requirement already satisfied: packaging>=17.1 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.19->pmdarima) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.19->pmdarima) (2024.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima)
(3.5.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels>=0.13.2->pmdarima) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.6->statsmodels>=0.13.2->pmdarima) (1.16.0)
Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
4.manylinux_2_28_x86_64.whl (2.1 MB)
                         2.1/2.1 MB
59.2 MB/s eta 0:00:00
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

[ ]:

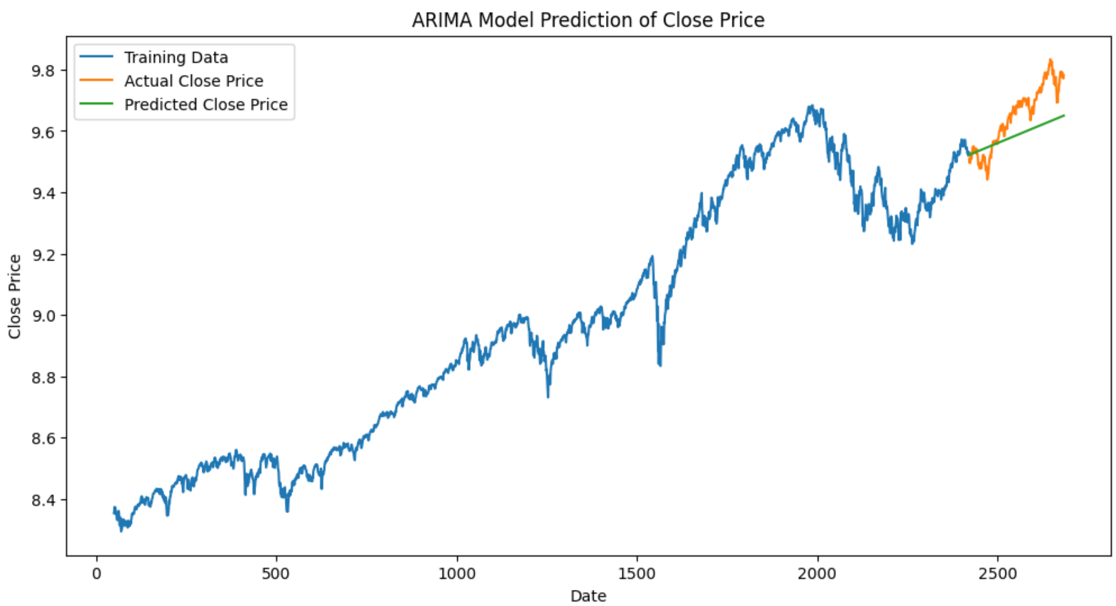
```
# prompt: write code for arima using autoarima to predict closing price in␣
↪given dataset. use training as 90% data

from pmdarima import auto_arima
```

```python
from sklearn.metrics import mean_squared_error

# Assuming 'NSEI' DataFrame is already loaded and prepared

# Split data into training and testing sets (90% train, 10% test)
train_data = NASDAQ['Close'][:-int(len(NASDAQ) * 0.1)]
test_data = NASDAQ['Close'][-int(len(NASDAQ) * 0.1):]

# Fit auto_arima model to the training data
model = auto_arima(train_data, start_p = 1, start_q = 1,
                          max_p = 100, max_q = 100,
                          start_P = 0,alpha=0.05,
                          trace = True,information_criterion='aic',
                          error_action ='ignore',   # we don't want to know if␣
 ↪an order does not work
                          suppress_warnings = True,  # we don't want␣
 ↪convergence warnings
                          stepwise = True)

# Make predictions on the test data
predictions = model.predict(n_periods=len(test_data))

# Evaluate the model
rmse = np.sqrt(mean_squared_error(test_data, predictions))
print(f'RMSE: {rmse}')

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(train_data, label='Training Data')
plt.plot(test_data.index, test_data, label='Actual Close Price')
plt.plot(test_data.index, predictions, label='Predicted Close Price')
plt.title('ARIMA Model Prediction of Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-13722.228, Time=0.60 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-13690.049, Time=0.29 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-13721.400, Time=0.31 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-13718.545, Time=0.62 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-13688.891, Time=0.14 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-13721.042, Time=1.29 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-13720.308, Time=2.43 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-13722.730, Time=2.93 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-13723.042, Time=3.52 sec
```

```
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-13720.867, Time=2.40 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=-13722.229, Time=2.90 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=-13720.308, Time=5.92 sec
 ARIMA(0,1,3)(0,0,0)[0]             : AIC=-13721.173, Time=0.60 sec

Best model:  ARIMA(0,1,3)(0,0,0)[0] intercept
Total fit time: 23.995 seconds
RMSE: 0.09160708428980627
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:837:
ValueWarning: No supported index is available. Prediction results will be given
with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:837:
FutureWarning: No supported index is available. In the next version, calling
this method in a model without a supported index will result in an exception.
  return get_prediction_index(
```



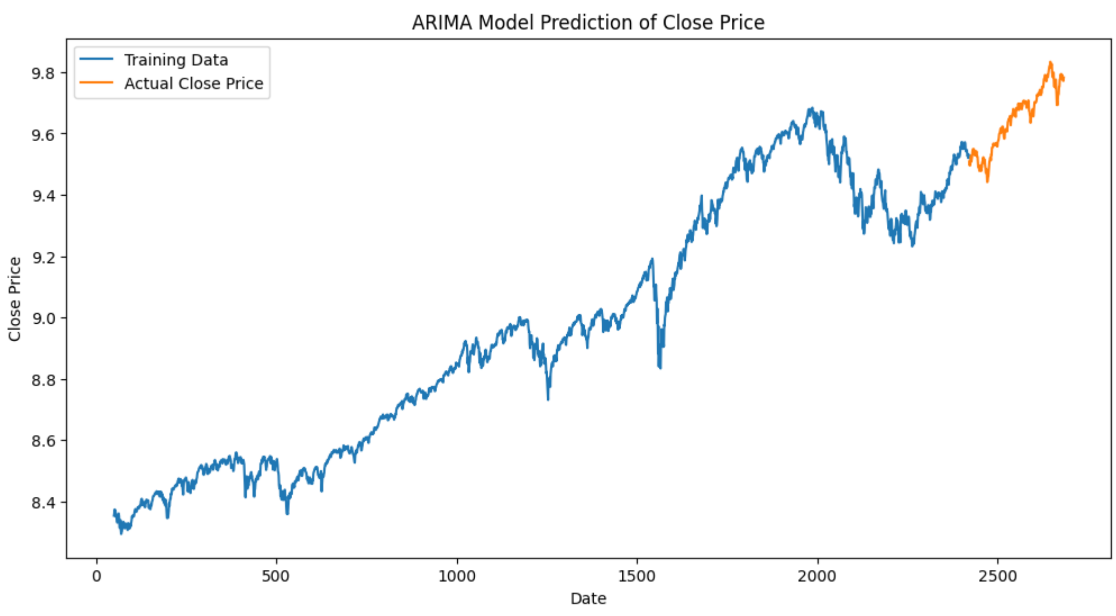
[ ]:
```
# prompt: find out confusion matrix,rmse,mse and other evaluation matrics for
↪the above fir

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
↪recall_score, f1_score

# Assuming 'test_data' and 'predictions' are already defined from the previous
↪code
```

```
# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(test_data, predictions))

# Calculate MSE (Mean Squared Error)
mse = mean_squared_error(test_data, predictions)

# Calculate MAE (Mean Absolute Error)
mae = mean_absolute_error(test_data, predictions)

# Calculate R-squared
r2 = r2_score(test_data, predictions)

print(f'RMSE: {rmse}')
print(f'MSE: {mse}')
print(f'MAE: {mae}')
print(f'R-squared: {r2}')

# You can also calculate other metrics like MAPE (Mean Absolute Percentage␣
↪Error)
# if needed, but it might require some custom implementation.

# For Classification metrics, you'd need to convert your predictions into␣
↪discrete classes
# (e.g., based on a threshold) and then calculate things like confusion matrix,
# accuracy, precision, recall, F1-score.


# Example of converting predictions to binary classes (assuming a threshold of␣
↪0.5):
# predicted_classes = (predictions > 0.5).astype(int)
# actual_classes = (test_data > 0.5).astype(int)

# Calculate confusion matrix
# cm = confusion_matrix(actual_classes, predicted_classes)
# print("Confusion Matrix:\n", cm)

# Calculate accuracy
# accuracy = accuracy_score(actual_classes, predicted_classes)
# print("Accuracy:", accuracy)

# Calculate precision
# precision = precision_score(actual_classes, predicted_classes)
# print("Precision:", precision)

# Calculate recall
# recall = recall_score(actual_classes, predicted_classes)
```

```
# print("Recall:", recall)

# Calculate F1-score
# f1 = f1_score(actual_classes, predicted_classes)
# print("F1-score:", f1)
```

```
RMSE: 0.09160708428980627
MSE: 0.008391857892079671
MAE: 0.07688855764152723
R-squared: 0.19323692909430001
```

[ ]:
```
# prompt: # prompt: generate the  training testing graph

# Assuming 'train_data', 'test_data', and 'predictions' are already defined

plt.figure(figsize=(12, 6))
plt.plot(train_data, label='Training Data')
plt.plot(test_data.index, test_data, label='Actual Close Price')
plt.title('ARIMA Model Prediction of Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[ ]:
```
# prompt: check for overfitting in above model

# Calculate the training and testing RMSE
```

```python
train_predictions = model.predict_in_sample()
train_rmse = np.sqrt(mean_squared_error(train_data, train_predictions))
test_rmse = np.sqrt(mean_squared_error(test_data, predictions))

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")

# Check for overfitting by comparing training and testing RMSE
if test_rmse > train_rmse and (test_rmse - train_rmse) > some_threshold:  # You can define a threshold for significance
  print("Warning: The model might be overfitting.")
  print("The testing RMSE is considerably higher than the training RMSE, indicating the model is performing poorly on unseen data.")
else:
  print("The model doesn't appear to be overfitting significantly.")
```

```
Training RMSE: 0.17203029535266565
Testing RMSE: 0.09160708428980627
The model doesn't appear to be overfitting significantly.
```

[ ]:
```python
# Calculate the training and testing RMSE
train_predictions = model.predict_in_sample()
train_rmse = np.sqrt(mean_squared_error(train_data, train_predictions))
test_rmse = np.sqrt(mean_squared_error(test_data, predictions))

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")

# Check for underfitting by comparing training and testing RMSE and the baseline RMSE
# Create an array of baseline predictions with the same length as test_data
baseline_predictions = np.repeat(np.mean(train_data), len(test_data))  # Repeat the mean for each test data point
baseline_rmse = np.sqrt(mean_squared_error(test_data, baseline_predictions))  # Calculate RMSE using baseline predictions
print(f"Baseline RMSE: {baseline_rmse}")

if train_rmse > baseline_rmse and test_rmse > baseline_rmse:
    print("Warning: The model might be underfitting.")
    print("Both training and testing RMSE are higher than the baseline RMSE, indicating the model is not learning effectively.")
elif train_rmse < baseline_rmse and test_rmse > baseline_rmse:
    print("The model is performing better than the baseline on the training data but not on the testing data.")
    print("This might indicate that it's not generalizing well or that the training data is not representative enough.")
```

```python
else:
    print("The model doesn't appear to be underfitting significantly.")

# You can also consider the R-squared value as another indicator for
 ↪underfitting.
# A low R-squared value (e.g., close to 0) suggests that the model is not
 ↪explaining much of the variance in the data.
```

```
Training RMSE: 0.17203029535266565
Testing RMSE: 0.09160708428980627
Baseline RMSE: 0.6955269187852495
The model doesn't appear to be underfitting significantly.
```

GRU

[ ]:
```python
!pip install tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-
packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (24.1)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (75.1.0)
```

```
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.44.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (13.9.3)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (0.13.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
(3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.18,>=2.17->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.18,>=2.17->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.18,>=2.17->tensorflow) (3.0.6)
Requirement already satisfied: MarkupSafe>=2.1.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow)
(2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-
packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)
```

[ ]:

```
# prompt: # prompt: write a code to apply GRU on NSEI dataset to predict close␣
↪using all attributes. use min max scalar for pre-processing on all numeric␣
↪attributes. use 90% training data and 10% testing data.

# Assuming 'NSEI' DataFrame is already loaded and prepared
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
# Extract relevant features for prediction (all attributes except 'Date')
data = NASDAQ.drop('Date', axis=1)

# Normalize the data using MinMaxScaler for numeric attributes
scaler = MinMaxScaler()
numeric_cols = data.select_dtypes(include=np.number).columns
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])

# Split the data into training and testing sets (90% train, 10% test)
train_size = int(len(data) * 0.90)
test_size = len(data) - train_size
train_data, test_data = data[0:train_size], data[train_size:len(data)]

# Separate the 'Close' column as the target variable for both train and test␣
↪sets
trainY = train_data['Close'].values
trainX = train_data.drop('Close', axis=1).values
testY = test_data['Close'].values
testX = test_data.drop('Close', axis=1).values


# Reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Create and fit the GRU network
model = Sequential()
model.add(GRU(units=50, return_sequences=True, input_shape=(trainX.shape[1],␣
↪trainX.shape[2])))
```

```
model.add(GRU(units=50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=32, verbose=2)

# Make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)


# Invert predictions back to original scale for 'Close' column
trainPredict = scaler.inverse_transform(np.concatenate((np.zeros((trainPredict.
 ↪shape[0], data.shape[1] - 1)), trainPredict), axis=1))[:, -1]
trainY = scaler.inverse_transform(np.concatenate((np.zeros((trainY.shape[0],␣
 ↪data.shape[1] - 1)), trainY.reshape(-1, 1)), axis=1))[:, -1]

testPredict = scaler.inverse_transform(np.concatenate((np.zeros((testPredict.
 ↪shape[0], data.shape[1] - 1)), testPredict), axis=1))[:, -1]
testY = scaler.inverse_transform(np.concatenate((np.zeros((testY.shape[0], data.
 ↪shape[1] - 1)), testY.reshape(-1, 1)), axis=1))[:, -1]


# Calculate root mean squared error
trainScore = np.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = np.sqrt(mean_squared_error(testY, testPredict))
print('Test Score: %.2f RMSE' % (testScore))

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(NASDAQ.index[:train_size], trainY, label='Training Actual Close')
plt.plot(NASDAQ.index[train_size:], testY, label='Testing Actual Close')
plt.plot(NASDAQ.index[train_size:], testPredict, label='Testing Predicted␣
 ↪Close')
plt.title('GRU Model Prediction of Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
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
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

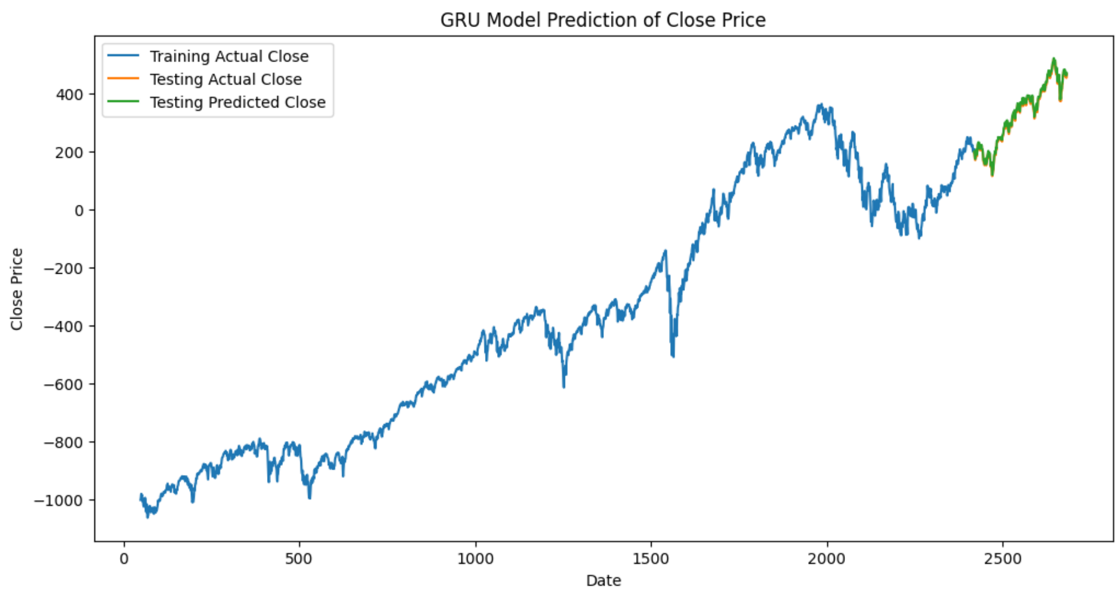
```
75/75 - 4s - 50ms/step - loss: 0.0256
Epoch 2/100
75/75 - 0s - 4ms/step - loss: 0.0024
Epoch 3/100
75/75 - 0s - 3ms/step - loss: 0.0018
Epoch 4/100
75/75 - 0s - 4ms/step - loss: 0.0012
Epoch 5/100
75/75 - 0s - 3ms/step - loss: 6.5292e-04
Epoch 6/100
75/75 - 0s - 3ms/step - loss: 2.8587e-04
Epoch 7/100
75/75 - 0s - 3ms/step - loss: 1.0633e-04
Epoch 8/100
75/75 - 0s - 3ms/step - loss: 5.5724e-05
Epoch 9/100
75/75 - 0s - 4ms/step - loss: 4.8550e-05
Epoch 10/100
75/75 - 0s - 4ms/step - loss: 4.3420e-05
Epoch 11/100
75/75 - 0s - 4ms/step - loss: 4.1905e-05
Epoch 12/100
75/75 - 0s - 4ms/step - loss: 4.0392e-05
Epoch 13/100
75/75 - 0s - 3ms/step - loss: 3.8016e-05
Epoch 14/100
75/75 - 0s - 4ms/step - loss: 3.6732e-05
Epoch 15/100
75/75 - 0s - 4ms/step - loss: 3.4862e-05
Epoch 16/100
75/75 - 0s - 3ms/step - loss: 3.5182e-05
Epoch 17/100
75/75 - 0s - 4ms/step - loss: 3.0787e-05
Epoch 18/100
75/75 - 0s - 3ms/step - loss: 2.8944e-05
Epoch 19/100
75/75 - 0s - 4ms/step - loss: 2.7095e-05
Epoch 20/100
75/75 - 0s - 3ms/step - loss: 2.5978e-05
Epoch 21/100
75/75 - 0s - 3ms/step - loss: 2.4059e-05
Epoch 22/100
75/75 - 0s - 5ms/step - loss: 2.3399e-05
Epoch 23/100
75/75 - 1s - 8ms/step - loss: 2.1424e-05
Epoch 24/100
75/75 - 1s - 8ms/step - loss: 1.9857e-05
Epoch 25/100
```

```
75/75 - 1s - 9ms/step - loss: 1.7684e-05
Epoch 26/100
75/75 - 1s - 7ms/step - loss: 1.7387e-05
Epoch 27/100
75/75 - 0s - 3ms/step - loss: 1.5521e-05
Epoch 28/100
75/75 - 0s - 3ms/step - loss: 1.4706e-05
Epoch 29/100
75/75 - 0s - 4ms/step - loss: 1.3999e-05
Epoch 30/100
75/75 - 0s - 4ms/step - loss: 1.3120e-05
Epoch 31/100
75/75 - 0s - 4ms/step - loss: 1.0617e-05
Epoch 32/100
75/75 - 0s - 4ms/step - loss: 9.9165e-06
Epoch 33/100
75/75 - 0s - 4ms/step - loss: 9.1915e-06
Epoch 34/100
75/75 - 0s - 4ms/step - loss: 8.4845e-06
Epoch 35/100
75/75 - 0s - 4ms/step - loss: 7.1258e-06
Epoch 36/100
75/75 - 0s - 3ms/step - loss: 7.1526e-06
Epoch 37/100
75/75 - 0s - 3ms/step - loss: 5.4532e-06
Epoch 38/100
75/75 - 0s - 3ms/step - loss: 5.1894e-06
Epoch 39/100
75/75 - 0s - 3ms/step - loss: 4.5841e-06
Epoch 40/100
75/75 - 0s - 4ms/step - loss: 4.1645e-06
Epoch 41/100
75/75 - 0s - 3ms/step - loss: 4.2088e-06
Epoch 42/100
75/75 - 0s - 3ms/step - loss: 3.3457e-06
Epoch 43/100
75/75 - 0s - 4ms/step - loss: 3.1352e-06
Epoch 44/100
75/75 - 0s - 3ms/step - loss: 2.9114e-06
Epoch 45/100
75/75 - 0s - 3ms/step - loss: 2.6578e-06
Epoch 46/100
75/75 - 0s - 4ms/step - loss: 2.9683e-06
Epoch 47/100
75/75 - 0s - 3ms/step - loss: 2.6130e-06
Epoch 48/100
75/75 - 0s - 4ms/step - loss: 2.8488e-06
Epoch 49/100
```

```
75/75 - 0s - 3ms/step - loss: 2.4250e-06
Epoch 50/100
75/75 - 0s - 4ms/step - loss: 2.6934e-06
Epoch 51/100
75/75 - 0s - 3ms/step - loss: 2.2217e-06
Epoch 52/100
75/75 - 0s - 3ms/step - loss: 2.4243e-06
Epoch 53/100
75/75 - 0s - 4ms/step - loss: 2.2279e-06
Epoch 54/100
75/75 - 0s - 4ms/step - loss: 2.3043e-06
Epoch 55/100
75/75 - 0s - 4ms/step - loss: 2.7563e-06
Epoch 56/100
75/75 - 0s - 4ms/step - loss: 2.3556e-06
Epoch 57/100
75/75 - 0s - 3ms/step - loss: 2.0991e-06
Epoch 58/100
75/75 - 0s - 4ms/step - loss: 2.7307e-06
Epoch 59/100
75/75 - 0s - 3ms/step - loss: 2.7855e-06
Epoch 60/100
75/75 - 0s - 4ms/step - loss: 2.3381e-06
Epoch 61/100
75/75 - 0s - 4ms/step - loss: 2.0558e-06
Epoch 62/100
75/75 - 0s - 3ms/step - loss: 2.1374e-06
Epoch 63/100
75/75 - 0s - 3ms/step - loss: 2.8520e-06
Epoch 64/100
75/75 - 0s - 5ms/step - loss: 2.1359e-06
Epoch 65/100
75/75 - 1s - 8ms/step - loss: 2.1678e-06
Epoch 66/100
75/75 - 1s - 8ms/step - loss: 2.1968e-06
Epoch 67/100
75/75 - 0s - 4ms/step - loss: 2.0685e-06
Epoch 68/100
75/75 - 0s - 5ms/step - loss: 2.4201e-06
Epoch 69/100
75/75 - 1s - 8ms/step - loss: 4.2931e-06
Epoch 70/100
75/75 - 0s - 7ms/step - loss: 1.8721e-06
Epoch 71/100
75/75 - 0s - 3ms/step - loss: 2.7118e-06
Epoch 72/100
75/75 - 0s - 4ms/step - loss: 2.8438e-06
Epoch 73/100
```

```
75/75 - 0s - 3ms/step - loss: 1.9313e-06
Epoch 74/100
75/75 - 0s - 3ms/step - loss: 1.6936e-06
Epoch 75/100
75/75 - 0s - 4ms/step - loss: 2.8782e-06
Epoch 76/100
75/75 - 0s - 3ms/step - loss: 2.1119e-06
Epoch 77/100
75/75 - 0s - 3ms/step - loss: 2.1577e-06
Epoch 78/100
75/75 - 0s - 4ms/step - loss: 2.5749e-06
Epoch 79/100
75/75 - 0s - 4ms/step - loss: 3.2137e-06
Epoch 80/100
75/75 - 0s - 4ms/step - loss: 2.9047e-06
Epoch 81/100
75/75 - 0s - 4ms/step - loss: 3.0393e-06
Epoch 82/100
75/75 - 0s - 4ms/step - loss: 1.9966e-06
Epoch 83/100
75/75 - 0s - 4ms/step - loss: 2.7231e-06
Epoch 84/100
75/75 - 0s - 4ms/step - loss: 3.7489e-06
Epoch 85/100
75/75 - 0s - 3ms/step - loss: 3.7785e-06
Epoch 86/100
75/75 - 0s - 4ms/step - loss: 2.9460e-06
Epoch 87/100
75/75 - 0s - 3ms/step - loss: 3.2710e-06
Epoch 88/100
75/75 - 0s - 3ms/step - loss: 2.3612e-06
Epoch 89/100
75/75 - 0s - 4ms/step - loss: 1.7207e-06
Epoch 90/100
75/75 - 0s - 4ms/step - loss: 1.8078e-06
Epoch 91/100
75/75 - 0s - 4ms/step - loss: 1.6877e-06
Epoch 92/100
75/75 - 0s - 4ms/step - loss: 3.3667e-06
Epoch 93/100
75/75 - 0s - 3ms/step - loss: 2.7449e-06
Epoch 94/100
75/75 - 0s - 3ms/step - loss: 5.0644e-06
Epoch 95/100
75/75 - 0s - 3ms/step - loss: 3.6456e-06
Epoch 96/100
75/75 - 0s - 3ms/step - loss: 1.7819e-06
Epoch 97/100
```

```
75/75 - 0s - 3ms/step - loss: 3.4746e-06
Epoch 98/100
75/75 - 0s - 4ms/step - loss: 2.0165e-06
Epoch 99/100
75/75 - 0s - 3ms/step - loss: 4.1315e-06
Epoch 100/100
75/75 - 0s - 4ms/step - loss: 2.0275e-06
75/75             1s 6ms/step
9/9               0s 2ms/step
Train Score: 3.22 RMSE
Test Score: 5.77 RMSE
```

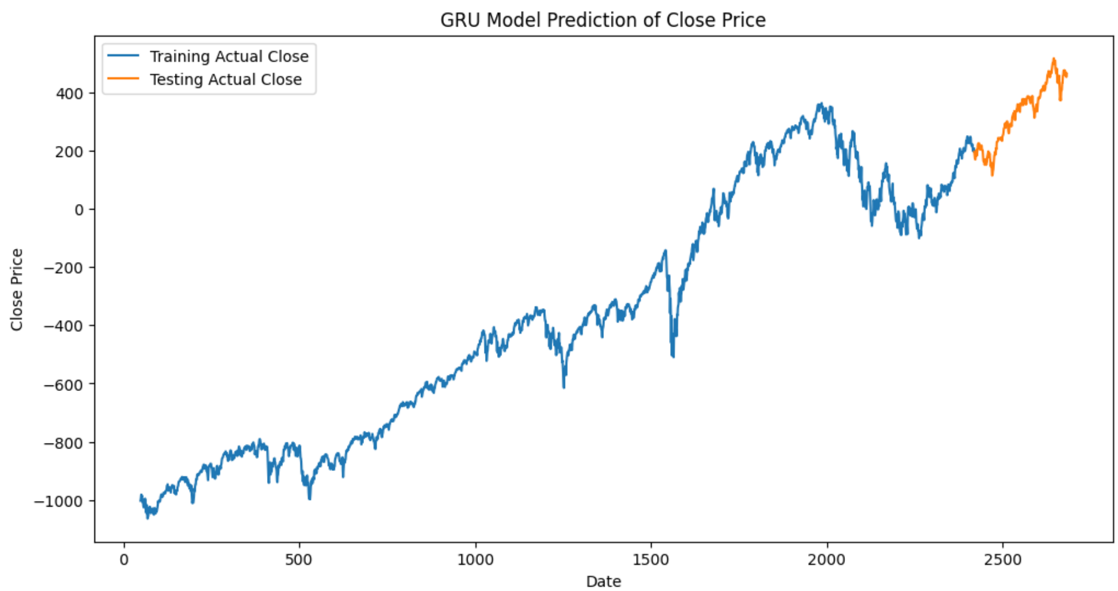


[ ]:
```
# prompt: # prompt: find out confusion matrix,rmse,mse and other evaluation␣
↪matrics for the above model
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score #␣
↪Import mean_absolute_error,r2_score

# Assuming 'test_data' and 'predictions' are already defined from the previous␣
↪code

# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(testY,testPredict))

# Calculate MSE (Mean Squared Error)
mse = mean_squared_error(testY,testPredict)

# Calculate MAE (Mean Absolute Error)
```

```
mae = mean_absolute_error(testY,testPredict)

# Calculate R-squared
r2 = r2_score(testY,testPredict)

print(f'RMSE: {rmse}')
print(f'MSE: {mse}')
print(f'MAE: {mae}')
print(f'R-squared: {r2}')
```

```
RMSE: 5.771684372909906
MSE: 33.31234050049242
MAE: 5.648565132574167
R-squared: 0.9969644258183652
```

[ ]:
```
plt.figure(figsize=(12, 6))
plt.plot(NASDAQ.index[:train_size], trainY, label='Training Actual Close')
plt.plot(NASDAQ.index[train_size:], testY, label='Testing Actual Close')
plt.title('GRU Model Prediction of Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[ ]:
```
# Calculate the training and testing RMSE
trainScore = np.sqrt(mean_squared_error(trainY, trainPredict))
testScore = np.sqrt(mean_squared_error(testY, testPredict))
```

```python
print(f"Training RMSE: {trainScore}")
print(f"Testing RMSE: {testScore}")

# Define a threshold for significance
some_threshold = 5  # You can adjust this value based on your data and model

# Check for overfitting by comparing training and testing RMSE
if testScore > trainScore and (testScore - trainScore) > some_threshold:  # You␣
↪can define a threshold for significance
  print("Warning: The model might be overfitting.")
  print("The testing RMSE is considerably higher than the training RMSE,␣
↪indicating the model is performing poorly on unseen data.")
else:
  print("The model doesn't appear to be overfitting significantly.")


# Check for underfitting by comparing training and testing RMSE and the␣
↪baseline RMSE
# Create an array of baseline predictions with the same length as test_data
baseline_predictions = np.repeat(np.mean(trainY), len(testY))  # Repeat the␣
↪mean for each test data point
baseline_rmse = np.sqrt(mean_squared_error(testY, baseline_predictions))  #␣
↪Calculate RMSE using baseline predictions
print(f"Baseline RMSE: {baseline_rmse}")

if trainScore > baseline_rmse and testScore > baseline_rmse:
    print("Warning: The model might be underfitting.")
    print("Both training and testing RMSE are higher than the baseline RMSE,␣
↪indicating the model is not learning effectively.")
elif trainScore < baseline_rmse and testScore > baseline_rmse:
    print("The model is performing better than the baseline on the training␣
↪data but not on the testing data.")
    print("This might indicate that it's not generalizing well or that the␣
↪training data is not representative enough.")
else:
    print("The model doesn't appear to be underfitting significantly.")
```

```
Training RMSE: 3.2207928948868303
Testing RMSE: 5.771684372909906
The model doesn't appear to be overfitting significantly.
Baseline RMSE: 713.4333018458588
The model doesn't appear to be underfitting significantly.
```

LSTM

[ ]:
```
# prompt: write a code to apply LSTM on NSEI dataset to predict close using all␣
↪attributes. use min max scalar for pre-processing on all numeric attributes.␣
↪use 90% training data and 10% testing data.

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error

# Assuming 'NASDAQ' DataFrame is already loaded and prepared
# Extract relevant features for prediction (all attributes except 'Date')
data = NASDAQ.drop('Date', axis=1)

# Normalize the data using MinMaxScaler for numeric attributes
scaler = MinMaxScaler()
numeric_cols = data.select_dtypes(include=np.number).columns
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])

# Split the data into training and testing sets (90% train, 10% test)
train_size = int(len(data) * 0.90)
test_size = len(data) - train_size
train_data, test_data = data[0:train_size], data[train_size:len(data)]

# Separate the 'Close' column as the target variable for both train and test␣
↪sets
trainY = train_data['Close'].values
trainX = train_data.drop('Close', axis=1).values
testY = test_data['Close'].values
testX = test_data.drop('Close', axis=1).values

# Reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(trainX.shape[1],␣
↪trainX.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=32, verbose=2)

# Make predictions
trainPredict = model.predict(trainX)
```

```
testPredict = model.predict(testX)

# Invert predictions back to original scale for 'Close' column
trainPredict = scaler.inverse_transform(np.concatenate((np.zeros((trainPredict.
↪shape[0], data.shape[1] - 1)), trainPredict), axis=1))[:, -1]
trainY = scaler.inverse_transform(np.concatenate((np.zeros((trainY.shape[0],␣
↪data.shape[1] - 1)), trainY.reshape(-1, 1)), axis=1))[:, -1]

testPredict = scaler.inverse_transform(np.concatenate((np.zeros((testPredict.
↪shape[0], data.shape[1] - 1)), testPredict), axis=1))[:, -1]
testY = scaler.inverse_transform(np.concatenate((np.zeros((testY.shape[0], data.
↪shape[1] - 1)), testY.reshape(-1, 1)), axis=1))[:, -1]

# Calculate root mean squared error
trainScore = np.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = np.sqrt(mean_squared_error(testY, testPredict))
print('Test Score: %.2f RMSE' % (testScore))

# Plot the results
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(NASDAQ.index[:train_size], trainY, label='Training Actual Close')
plt.plot(NASDAQ.index[train_size:], testY, label='Testing Actual Close')
plt.plot(NASDAQ.index[train_size:], testPredict, label='Testing Predicted␣
↪Close')
plt.title('LSTM Model Prediction of Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
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
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

```
75/75 - 4s - 54ms/step - loss: 0.0666
Epoch 2/100
75/75 - 1s - 7ms/step - loss: 0.0042
Epoch 3/100
75/75 - 0s - 3ms/step - loss: 0.0029
Epoch 4/100
75/75 - 0s - 4ms/step - loss: 0.0018
Epoch 5/100
```

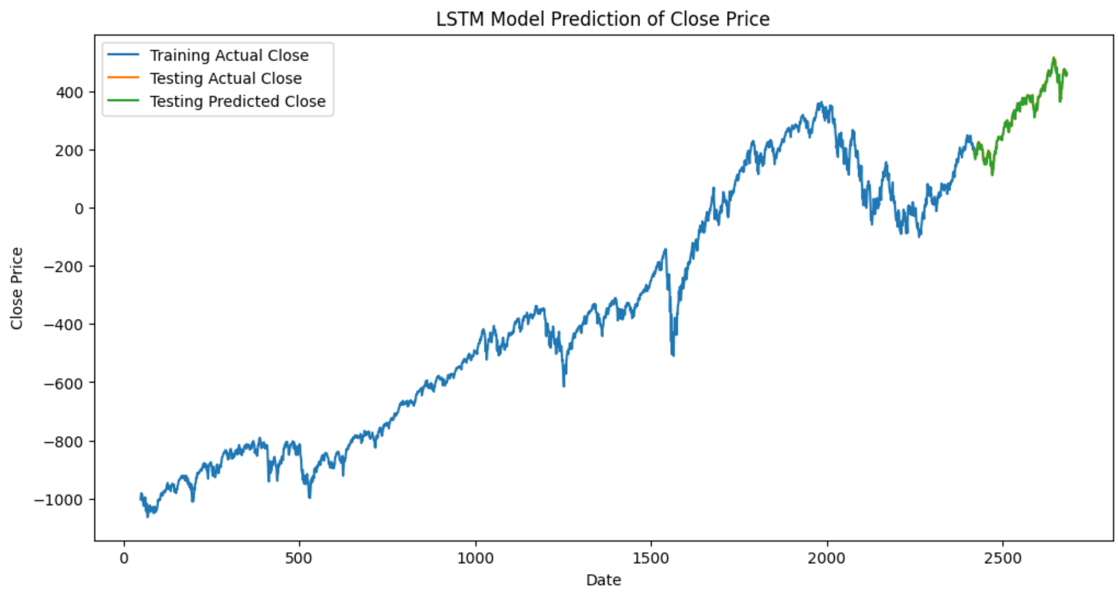
```
75/75 - 0s - 3ms/step - loss: 7.6651e-04
Epoch 6/100
75/75 - 0s - 4ms/step - loss: 2.0082e-04
Epoch 7/100
75/75 - 0s - 4ms/step - loss: 9.8946e-05
Epoch 8/100
75/75 - 0s - 4ms/step - loss: 8.8653e-05
Epoch 9/100
75/75 - 0s - 3ms/step - loss: 7.6906e-05
Epoch 10/100
75/75 - 0s - 4ms/step - loss: 7.0135e-05
Epoch 11/100
75/75 - 0s - 4ms/step - loss: 6.5231e-05
Epoch 12/100
75/75 - 0s - 4ms/step - loss: 5.9920e-05
Epoch 13/100
75/75 - 0s - 3ms/step - loss: 5.5918e-05
Epoch 14/100
75/75 - 0s - 4ms/step - loss: 5.1222e-05
Epoch 15/100
75/75 - 0s - 4ms/step - loss: 4.8290e-05
Epoch 16/100
75/75 - 0s - 3ms/step - loss: 4.5802e-05
Epoch 17/100
75/75 - 0s - 4ms/step - loss: 4.0521e-05
Epoch 18/100
75/75 - 0s - 3ms/step - loss: 3.7606e-05
Epoch 19/100
75/75 - 0s - 3ms/step - loss: 3.5584e-05
Epoch 20/100
75/75 - 0s - 3ms/step - loss: 3.3761e-05
Epoch 21/100
75/75 - 0s - 3ms/step - loss: 3.0703e-05
Epoch 22/100
75/75 - 0s - 3ms/step - loss: 2.7733e-05
Epoch 23/100
75/75 - 0s - 3ms/step - loss: 2.6464e-05
Epoch 24/100
75/75 - 0s - 3ms/step - loss: 2.3906e-05
Epoch 25/100
75/75 - 0s - 3ms/step - loss: 2.1697e-05
Epoch 26/100
75/75 - 0s - 3ms/step - loss: 2.0032e-05
Epoch 27/100
75/75 - 0s - 3ms/step - loss: 1.9242e-05
Epoch 28/100
75/75 - 0s - 4ms/step - loss: 1.7329e-05
Epoch 29/100
```

```
75/75 - 0s - 4ms/step - loss: 1.6898e-05
Epoch 30/100
75/75 - 0s - 3ms/step - loss: 1.4304e-05
Epoch 31/100
75/75 - 0s - 4ms/step - loss: 1.2320e-05
Epoch 32/100
75/75 - 0s - 3ms/step - loss: 1.0995e-05
Epoch 33/100
75/75 - 0s - 4ms/step - loss: 9.9641e-06
Epoch 34/100
75/75 - 0s - 4ms/step - loss: 9.1142e-06
Epoch 35/100
75/75 - 0s - 3ms/step - loss: 9.0215e-06
Epoch 36/100
75/75 - 0s - 4ms/step - loss: 7.8083e-06
Epoch 37/100
75/75 - 0s - 3ms/step - loss: 7.1923e-06
Epoch 38/100
75/75 - 0s - 4ms/step - loss: 6.5405e-06
Epoch 39/100
75/75 - 0s - 4ms/step - loss: 5.8997e-06
Epoch 40/100
75/75 - 0s - 4ms/step - loss: 5.3762e-06
Epoch 41/100
75/75 - 0s - 5ms/step - loss: 5.1136e-06
Epoch 42/100
75/75 - 0s - 4ms/step - loss: 4.8567e-06
Epoch 43/100
75/75 - 0s - 4ms/step - loss: 4.2616e-06
Epoch 44/100
75/75 - 0s - 4ms/step - loss: 4.0360e-06
Epoch 45/100
75/75 - 0s - 4ms/step - loss: 4.3116e-06
Epoch 46/100
75/75 - 0s - 4ms/step - loss: 4.6580e-06
Epoch 47/100
75/75 - 0s - 4ms/step - loss: 3.7234e-06
Epoch 48/100
75/75 - 0s - 4ms/step - loss: 4.1186e-06
Epoch 49/100
75/75 - 1s - 8ms/step - loss: 3.8011e-06
Epoch 50/100
75/75 - 1s - 7ms/step - loss: 3.5955e-06
Epoch 51/100
75/75 - 0s - 4ms/step - loss: 3.6650e-06
Epoch 52/100
75/75 - 0s - 3ms/step - loss: 4.5642e-06
Epoch 53/100
```

```
75/75 - 0s - 3ms/step - loss: 3.5183e-06
Epoch 54/100
75/75 - 0s - 4ms/step - loss: 3.0605e-06
Epoch 55/100
75/75 - 0s - 4ms/step - loss: 3.0772e-06
Epoch 56/100
75/75 - 0s - 3ms/step - loss: 3.0437e-06
Epoch 57/100
75/75 - 0s - 4ms/step - loss: 3.3019e-06
Epoch 58/100
75/75 - 0s - 3ms/step - loss: 3.1938e-06
Epoch 59/100
75/75 - 0s - 4ms/step - loss: 3.0197e-06
Epoch 60/100
75/75 - 0s - 3ms/step - loss: 3.8462e-06
Epoch 61/100
75/75 - 0s - 4ms/step - loss: 3.0926e-06
Epoch 62/100
75/75 - 0s - 3ms/step - loss: 2.8492e-06
Epoch 63/100
75/75 - 0s - 3ms/step - loss: 2.9147e-06
Epoch 64/100
75/75 - 0s - 4ms/step - loss: 3.2415e-06
Epoch 65/100
75/75 - 0s - 3ms/step - loss: 2.6862e-06
Epoch 66/100
75/75 - 0s - 4ms/step - loss: 3.6062e-06
Epoch 67/100
75/75 - 0s - 3ms/step - loss: 3.2983e-06
Epoch 68/100
75/75 - 0s - 3ms/step - loss: 2.7829e-06
Epoch 69/100
75/75 - 0s - 3ms/step - loss: 2.4313e-06
Epoch 70/100
75/75 - 0s - 3ms/step - loss: 2.5500e-06
Epoch 71/100
75/75 - 0s - 4ms/step - loss: 2.9622e-06
Epoch 72/100
75/75 - 0s - 3ms/step - loss: 2.7100e-06
Epoch 73/100
75/75 - 0s - 4ms/step - loss: 2.8492e-06
Epoch 74/100
75/75 - 0s - 3ms/step - loss: 2.2645e-06
Epoch 75/100
75/75 - 0s - 4ms/step - loss: 2.1371e-06
Epoch 76/100
75/75 - 0s - 4ms/step - loss: 2.2920e-06
Epoch 77/100
```

```
75/75 - 0s - 4ms/step - loss: 2.6917e-06
Epoch 78/100
75/75 - 0s - 3ms/step - loss: 3.1397e-06
Epoch 79/100
75/75 - 0s - 4ms/step - loss: 2.7327e-06
Epoch 80/100
75/75 - 0s - 3ms/step - loss: 2.2082e-06
Epoch 81/100
75/75 - 0s - 4ms/step - loss: 2.1711e-06
Epoch 82/100
75/75 - 0s - 3ms/step - loss: 2.6289e-06
Epoch 83/100
75/75 - 0s - 4ms/step - loss: 2.1869e-06
Epoch 84/100
75/75 - 0s - 3ms/step - loss: 2.3470e-06
Epoch 85/100
75/75 - 0s - 3ms/step - loss: 2.3699e-06
Epoch 86/100
75/75 - 0s - 4ms/step - loss: 2.7937e-06
Epoch 87/100
75/75 - 0s - 4ms/step - loss: 3.1060e-06
Epoch 88/100
75/75 - 0s - 4ms/step - loss: 2.4484e-06
Epoch 89/100
75/75 - 1s - 9ms/step - loss: 4.3447e-06
Epoch 90/100
75/75 - 0s - 4ms/step - loss: 2.9187e-06
Epoch 91/100
75/75 - 0s - 4ms/step - loss: 2.0998e-06
Epoch 92/100
75/75 - 0s - 4ms/step - loss: 2.0514e-06
Epoch 93/100
75/75 - 0s - 4ms/step - loss: 2.7080e-06
Epoch 94/100
75/75 - 0s - 4ms/step - loss: 2.7631e-06
Epoch 95/100
75/75 - 0s - 4ms/step - loss: 2.7574e-06
Epoch 96/100
75/75 - 1s - 7ms/step - loss: 2.2761e-06
Epoch 97/100
75/75 - 0s - 3ms/step - loss: 1.5641e-06
Epoch 98/100
75/75 - 0s - 4ms/step - loss: 2.0869e-06
Epoch 99/100
75/75 - 0s - 3ms/step - loss: 2.5679e-06
Epoch 100/100
75/75 - 0s - 4ms/step - loss: 2.2954e-06
75/75            1s 5ms/step
```

```
9/9              0s 2ms/step
Train Score: 3.44 RMSE
Test Score: 2.17 RMSE
```



[ ]:

```
# prompt: calculate rmse rma r2 etc for the above model

# Assuming 'testY' and 'testPredict' are already defined from the previous code

# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(testY, testPredict))

# Calculate MSE (Mean Squared Error)
mse = mean_squared_error(testY, testPredict)

# Calculate MAE (Mean Absolute Error)
mae = mean_absolute_error(testY, testPredict)

# Calculate R-squared
r2 = r2_score(testY, testPredict)

print(f'RMSE: {rmse}')
print(f'MSE: {mse}')
print(f'MAE: {mae}')
print(f'R-squared: {r2}')

# You can also calculate other metrics like:
# - MAPE (Mean Absolute Percentage Error)
# - Adjusted R-squared (for multiple regression)
```

```
RMSE: 2.1741278727698337
MSE: 4.726832007154681
MAE: 1.7263560355513206
R-squared: 0.9995692692561896
```

[ ]:
```python
# prompt: calculate rmse for train and test

# Assuming 'trainY', 'trainPredict', 'testY', and 'testPredict' are already␣
↪defined

# Calculate the training and testing RMSE
train_rmse = np.sqrt(mean_squared_error(trainY, trainPredict))
test_rmse = np.sqrt(mean_squared_error(testY, testPredict))

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")
```

```
Training RMSE: 3.4372081668182934
Testing RMSE: 2.1741278727698337
```

[ ]:
```python
# prompt: check for overfitting

# Assuming 'trainY', 'trainPredict', 'testY', and 'testPredict' are already␣
↪defined

# Calculate the training and testing RMSE
train_rmse = np.sqrt(mean_squared_error(trainY, trainPredict))
test_rmse = np.sqrt(mean_squared_error(testY, testPredict))

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")

# Define a threshold for significance
some_threshold = 5  # You can adjust this value based on your data and model

# Check for overfitting by comparing training and testing RMSE
if test_rmse > train_rmse and (test_rmse - train_rmse) > some_threshold:
  print("Warning: The model might be overfitting.")
  print("The testing RMSE is considerably higher than the training RMSE,␣
↪indicating the model is performing poorly on unseen data.")
else:
  print("The model doesn't appear to be overfitting significantly.")


# Check for underfitting by comparing training and testing RMSE and the␣
↪baseline RMSE
# Create an array of baseline predictions with the same length as test_data
```

```
baseline_predictions = np.repeat(np.mean(trainY), len(testY))  # Repeat the␣
↪mean for each test data point
baseline_rmse = np.sqrt(mean_squared_error(testY, baseline_predictions))  #␣
↪Calculate RMSE using baseline predictions
print(f"Baseline RMSE: {baseline_rmse}")

if train_rmse > baseline_rmse and test_rmse > baseline_rmse:
    print("Warning: The model might be underfitting.")
    print("Both training and testing RMSE are higher than the baseline RMSE,␣
↪indicating the model is not learning effectively.")
elif train_rmse < baseline_rmse and test_rmse > baseline_rmse:
    print("The model is performing better than the baseline on the training␣
↪data but not on the testing data.")
    print("This might indicate that it's not generalizing well or that the␣
↪training data is not representative enough.")
else:
    print("The model doesn't appear to be underfitting significantly.")
```

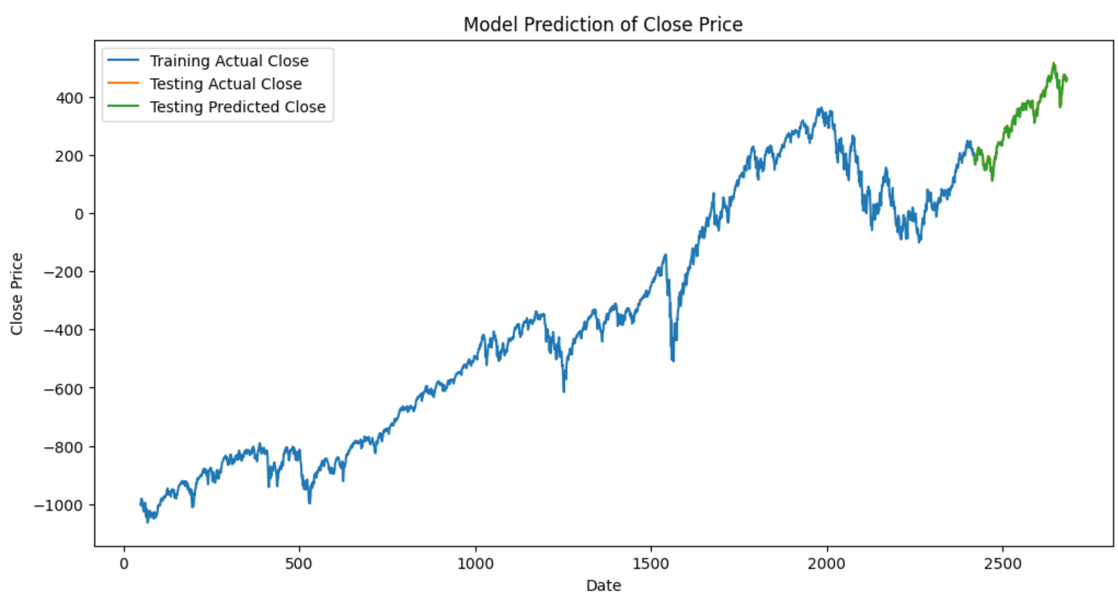
```
Training RMSE: 3.4372081668182934
Testing RMSE: 2.1741278727698337
The model doesn't appear to be overfitting significantly.
Baseline RMSE: 713.4333018458588
The model doesn't appear to be underfitting significantly.
```

LINEAR REGRESSION

[ ]:
```
# prompt: write a code to apply Linear regression on NSEI dataset to predict␣
↪close using all attributes. use min max scalar for pre-processing on all␣
↪numeric attributes. use 90% training data and 10% testing data. add all␣
↪required libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Assuming 'NSEI' DataFrame is already loaded and prepared
# Extract relevant features for prediction (all attributes except 'Date')
X = NASDAQ.drop('Date', axis=1)
y = NASDAQ['Close']

# Normalize the data using MinMaxScaler for numeric attributes
scaler = MinMaxScaler()
numeric_cols = X.select_dtypes(include=np.number).columns
X[numeric_cols] = scaler.fit_transform(X[numeric_cols])
```

```
# Split the data into training and testing sets (90% train, 10% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
↪random_state=42)

# Create and fit the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

[ ]:
```
# prompt: generate graph for predicted test values and training values and␣
↪actual values against time

plt.figure(figsize=(12, 6))
plt.plot(NASDAQ.index[:train_size], trainY, label='Training Actual Close')
plt.plot(NASDAQ.index[train_size:], testY, label='Testing Actual Close')
plt.plot(NASDAQ.index[train_size:], testPredict, label='Testing Predicted␣
↪Close')
plt.title('Model Prediction of Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



[ ]:
```
# prompt: calculate rmse rma r2 etc for the above model

# Assuming 'y_test' and 'y_test_pred' are already defined from the previous code

# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Calculate MSE (Mean Squared Error)
mse = mean_squared_error(y_test, y_test_pred)

# Calculate MAE (Mean Absolute Error)
mae = mean_absolute_error(y_test, y_test_pred)

# Calculate R-squared
r2 = r2_score(y_test, y_test_pred)

print(f'RMSE: {rmse}')
print(f'MSE: {mse}')
print(f'MAE: {mae}')
print(f'R-squared: {r2}')

# Calculate the training and testing RMSE
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")

# Define a threshold for significance
some_threshold = 5  # You can adjust this value based on your data and model

# Check for overfitting by comparing training and testing RMSE
if test_rmse > train_rmse and (test_rmse - train_rmse) > some_threshold:
  print("Warning: The model might be overfitting.")
  print("The testing RMSE is considerably higher than the training RMSE,␣
↪indicating the model is performing poorly on unseen data.")
else:
  print("The model doesn't appear to be overfitting significantly.")


# Check for underfitting by comparing training and testing RMSE and the␣
↪baseline RMSE
# Create an array of baseline predictions with the same length as test_data
baseline_predictions = np.repeat(np.mean(y_train), len(y_test))  # Repeat the␣
↪mean for each test data point
baseline_rmse = np.sqrt(mean_squared_error(y_test, baseline_predictions))  #␣
↪Calculate RMSE using baseline predictions
```

```
print(f"Baseline RMSE: {baseline_rmse}")

if train_rmse > baseline_rmse and test_rmse > baseline_rmse:
    print("Warning: The model might be underfitting.")
    print("Both training and testing RMSE are higher than the baseline RMSE,␣
↪indicating the model is not learning effectively.")
elif train_rmse < baseline_rmse and test_rmse > baseline_rmse:
    print("The model is performing better than the baseline on the training␣
↪data but not on the testing data.")
    print("This might indicate that it's not generalizing well or that the␣
↪training data is not representative enough.")
else:
    print("The model doesn't appear to be underfitting significantly.")
```

```
RMSE: 1.071183478194952e-15
MSE: 1.1474340439578354e-30
MAE: 6.459479416000911e-16
R-squared: 1.0
Training RMSE: 1.003719531511294e-15
Testing RMSE: 1.071183478194952e-15
The model doesn't appear to be overfitting significantly.
Baseline RMSE: 0.4244991928347415
The model doesn't appear to be underfitting significantly.
```

RANDOM FORREST

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

# Function to preprocess data
def preprocess_data(df):
    # Use only the 'Close' price for prediction
    NASDAQ = df[['Close']]

    # Initialize the MinMaxScaler to normalize the data
    scaler = MinMaxScaler(feature_range=(0, 1))

    # Scale the 'Close' price data
    scaled_data = scaler.fit_transform(NASDAQ)

    return scaled_data, scaler
```

[ ]:
```
nsei_data, nsei_scaler = preprocess_data(NASDAQ)
```

[ ]:
```
import numpy as np
def create_dataset(data):
    X, y = [], []
    # Loop through the dataset, using each point as input and the next point as␣
↪the target
```

```
    for i in range(len(data) - 1):
        X.append(data[i, 0])  # Current day's value
        y.append(data[i + 1, 0])  # Next day's value as target
    return np.array(X).reshape(-1, 1), np.array(y).reshape(-1, 1)


nsei_X, nsei_y = create_dataset(nsei_data)
```

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Function to perform grid search
def perform_grid_search(X, y):
    model = RandomForestRegressor(random_state=42)
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
 ↪n_jobs=-1, verbose=2)
    grid_search.fit(X, y)
    return grid_search.best_estimator_


nsei_rf_best = perform_grid_search(nsei_X, nsei_y)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
```

```
/usr/local/lib/python3.10/dist-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
/usr/local/lib/python3.10/dist-
packages/sklearn/model_selection/_validation.py:540: FitFailedWarning:
540 fits failed out of a total of 1620.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
```

```
540 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-
packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1466, in
wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 666, in
_validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-
packages/sklearn/utils/_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestRegressor must be an int in the range [1, inf), a float
in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning:
invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:1103:
UserWarning: One or more of the test scores are non-finite: [       nan
nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.74928933 0.74644047 0.74442425
 0.75409213 0.75419933 0.75224858 0.73948623 0.73912381 0.7376271
 0.75230836 0.75291382 0.75193009 0.75024569 0.75083044 0.74954817
 0.74013264 0.73924339 0.73818485 0.72651832 0.72766098 0.72697761
 0.72651832 0.72766098 0.72697761 0.72411545 0.72561319 0.7250922
 0.74928933 0.74644047 0.74442425 0.75409213 0.75419933 0.75224858
 0.73948623 0.73912381 0.7376271  0.75230836 0.75291382 0.75193009
 0.75024569 0.75083044 0.74954817 0.74013264 0.73924339 0.73818485
 0.72651832 0.72766098 0.72697761 0.72651832 0.72766098 0.72697761
 0.72411545 0.72561319 0.7250922         nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.74886367 0.74574158 0.74349338 0.75390898 0.75401792 0.752013
 0.73944895 0.73908603 0.73757937 0.7522253  0.7528267  0.75183543
 0.75017892 0.75076396 0.749475   0.74012418 0.73923334 0.73817239
 0.72651827 0.7276605  0.72697695 0.72651827 0.7276605  0.72697695
```

```
 0.72411585 0.72561303 0.72509185 0.74886367 0.74574158 0.74349338
 0.75390898 0.75401792 0.752013   0.73944895 0.73908603 0.73757937
 0.7522253  0.7528267  0.75183543 0.75017892 0.75076396 0.749475
 0.74012418 0.73923334 0.73817239 0.72651827 0.7276605  0.72697695
 0.72651827 0.7276605  0.72697695 0.72411585 0.72561303 0.72509185
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.74886362 0.74574153 0.74349332
 0.75390899 0.75401792 0.75201299 0.73944895 0.73908603 0.73757937
 0.7522253  0.7528267  0.75183543 0.75017892 0.75076396 0.749475
 0.74012418 0.73923334 0.73817239 0.72651827 0.7276605  0.72697695
 0.72651827 0.7276605  0.72697695 0.72411585 0.72561303 0.72509185
 0.74886362 0.74574153 0.74349332 0.75390899 0.75401792 0.75201299
 0.73944895 0.73908603 0.73757937 0.7522253  0.7528267  0.75183543
 0.75017892 0.75076396 0.749475   0.74012418 0.73923334 0.73817239
 0.72651827 0.7276605  0.72697695 0.72651827 0.7276605  0.72697695
 0.72411585 0.72561303 0.72509185        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.74886362 0.74574153 0.74349332 0.75390899 0.75401792 0.75201299
 0.73944895 0.73908603 0.73757937 0.7522253  0.7528267  0.75183543
 0.75017892 0.75076396 0.749475   0.74012418 0.73923334 0.73817239
 0.72651827 0.7276605  0.72697695 0.72651827 0.7276605  0.72697695
 0.72411585 0.72561303 0.72509185 0.74886362 0.74574153 0.74349332
 0.75390899 0.75401792 0.75201299 0.73944895 0.73908603 0.73757937
 0.7522253  0.7528267  0.75183543 0.75017892 0.75076396 0.749475
 0.74012418 0.73923334 0.73817239 0.72651827 0.7276605  0.72697695
 0.72651827 0.7276605  0.72697695 0.72411585 0.72561303 0.72509185]
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  return fit_method(estimator, *args, **kwargs)
```

[ ]:
```python
import numpy as np
from sklearn.metrics import r2_score, mean_squared_error

# Define evaluation metrics
def rmse(y_true, y_pred):
    return np.sqrt(np.mean((y_pred - y_true) ** 2))

def mape(y_true, y_pred):
```

```
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def mbe(y_true, y_pred):
    return np.mean(y_pred - y_true)

# Function to evaluate the model
def evaluate_model(model, X, y, scaler):
    # Predict using the model
    predicted = model.predict(X)

    # Inverse transform the predictions and true values to the original scale
    predicted = scaler.inverse_transform(predicted.reshape(-1, 1))
    y = scaler.inverse_transform(y.reshape(-1, 1))

    # Calculate evaluation metrics
    rmse_val = rmse(y, predicted)
    mape_val = mape(y, predicted)
    mbe_val = mbe(y, predicted)
    mse_val = mean_squared_error(y, predicted)
    r2_val = r2_score(y, predicted)

    # Print metrics
    print(f"Evaluation Metrics:")
    print(f"RMSE: {rmse_val}")
    print(f"MAPE: {mape_val}")
    print(f"MBE: {mbe_val}")
    print(f"MSE: {mse_val}")
    print(f"R²: {r2_val}")

    return rmse_val, mape_val, mbe_val, mse_val, r2_val
```

[ ]:
```
nsei_rmse, nsei_mape, nsei_mbe, nsei_mse, nsei_rsquare =␣
 ↪evaluate_model(nsei_rf_best, nsei_X, nsei_y, nsei_scaler)
```

```
Evaluation Metrics:
RMSE: 90.78732683699333
MAPE: 0.6575203566437353
MBE: 0.781100280562942
MSE: 8242.33871420705
R²: 0.9994676677653671
```

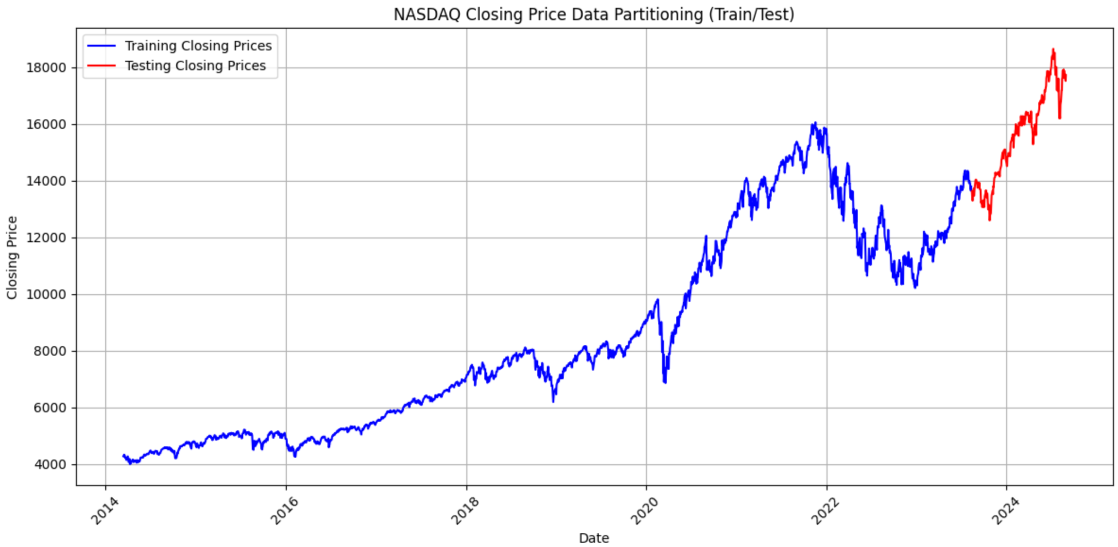
[ ]:
```
# prompt: find evaluation matrix r2, rmse, mse, mae

# Assuming you have already trained your model and have nsei_rf_best, nsei_X,␣
 ↪nsei_y, and nsei_scaler defined as in your provided code.

# You can directly call the evaluate_model function to get the desired metrics.
```

```
nsei_rmse, nsei_mape, nsei_mbe, nsei_mse, nsei_rsquare =␣
↪evaluate_model(nsei_rf_best, nsei_X, nsei_y, nsei_scaler)

# Print the individual metrics
print(f"R-squared: {nsei_rsquare}")
print(f"RMSE: {nsei_rmse}")
print(f"MSE: {nsei_mse}")
print(f"MAE: {nsei_rmse}")  # Assuming you are using RMSE as a proxy for MAE,␣
↪as MAE is not calculated directly in the provided code.
```

```
Evaluation Metrics:
RMSE: 90.78732683699333
MAPE: 0.6575203566437353
MBE: 0.781100280562942
MSE: 8242.33871420705
R²: 0.9994676677653671
R-squared: 0.9994676677653671
RMSE: 90.78732683699333
MSE: 8242.33871420705
MAE: 90.78732683699333
```

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Assuming 'NSEI' DataFrame exists with 'Date' and 'Close' columns
# Also assuming 'nsei_X', 'nsei_y', and 'nsei_scaler' are defined

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(nsei_X, nsei_y, test_size=0.
↪1, random_state=42, shuffle=False)

# Ensure the dates align with the data split
train_dates = NASDAQ['Date'][:len(y_train)].reset_index(drop=True)
test_dates = NASDAQ['Date'][len(y_train):len(y_train) + len(y_test)].
↪reset_index(drop=True)

# Plotting
plt.figure(figsize=(12, 6))

# Plot training data
plt.plot(train_dates, nsei_scaler.inverse_transform(y_train.reshape(-1, 1)),␣
↪label='Training Closing Prices', color='blue')

# Plot testing data
plt.plot(test_dates, nsei_scaler.inverse_transform(y_test.reshape(-1, 1)),␣
↪label='Testing Closing Prices', color='red')
```

```python
# Set labels, title, legend, and grid
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('NASDAQ Closing Price Data Partitioning (Train/Test)')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```



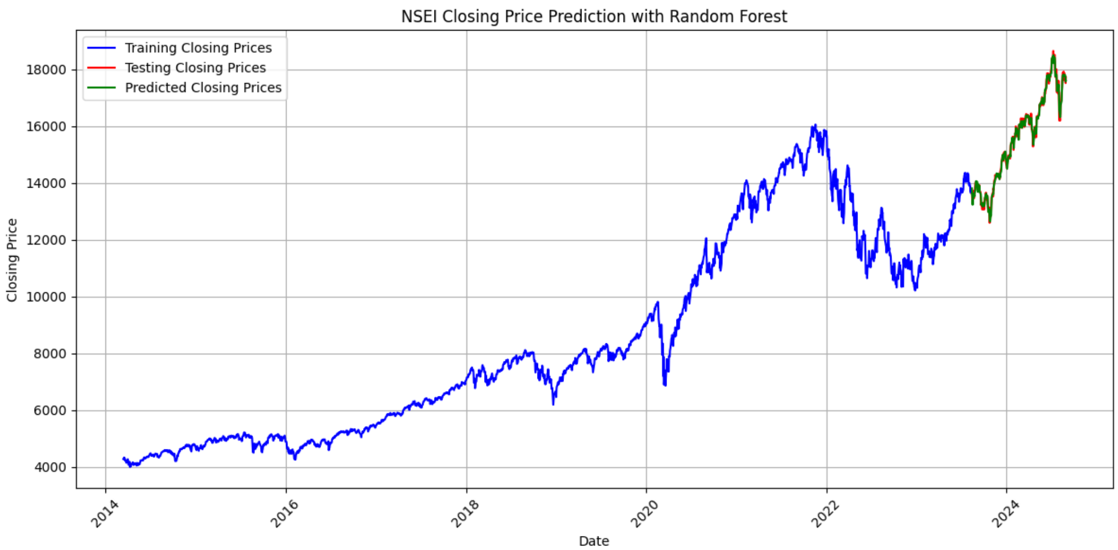
[ ]:

```python
# prompt: generate graph for training data, testing data and predicted data

# Assuming you have already trained your model and have nsei_rf_best, nsei_X,␣
↪nsei_y, and nsei_scaler defined as in your provided code.

# You can directly call the evaluate_model function to get the desired metrics.
nsei_rmse, nsei_mape, nsei_mbe, nsei_mse, nsei_rsquare =␣
↪evaluate_model(nsei_rf_best, nsei_X, nsei_y, nsei_scaler)

# Print the individual metrics
print(f"R-squared: {nsei_rsquare}")
print(f"RMSE: {nsei_rmse}")
print(f"MSE: {nsei_mse}")
print(f"MAE: {nsei_rmse}")  # Assuming you are using RMSE as a proxy for MAE,␣
↪as MAE is not calculated directly in the provided code.

# Assuming 'NSEI' DataFrame exists with 'Date' and 'Close' columns
# Also assuming 'nsei_X', 'nsei_y', and 'nsei_scaler' are defined
```

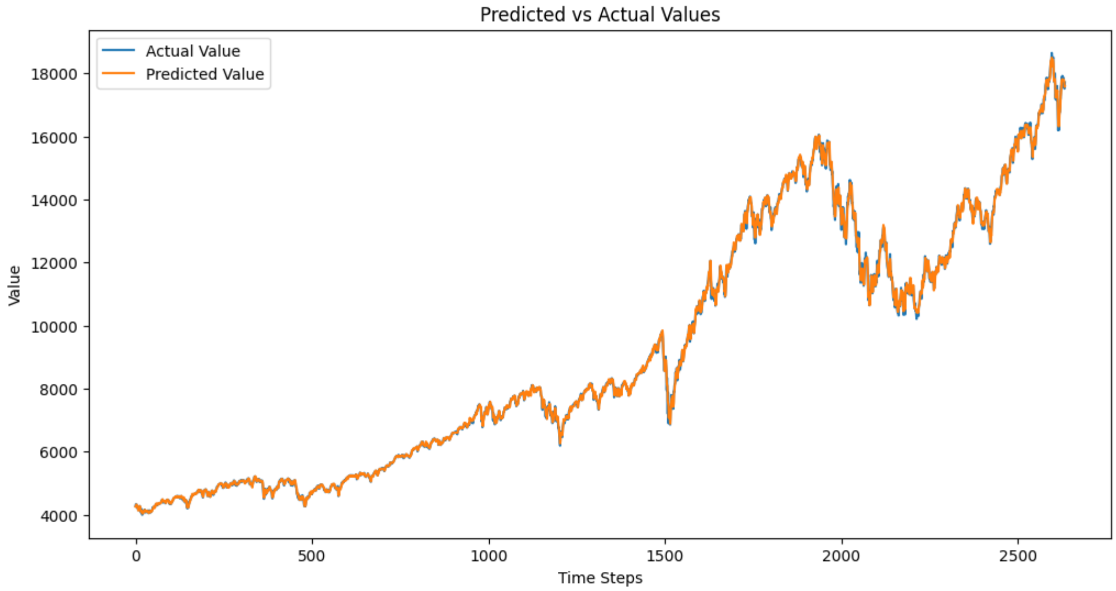
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(nsei_X, nsei_y, test_size=0.
    ↪1, random_state=42, shuffle=False)

# Predict on the test set
y_pred = nsei_rf_best.predict(X_test)

# Ensure the dates align with the data split
train_dates = NASDAQ['Date'][:len(y_train)].reset_index(drop=True)
test_dates = NASDAQ['Date'][len(y_train):len(y_train) + len(y_test)].
    ↪reset_index(drop=True)

# Inverse transform the predictions and true values to the original scale
y_pred_original = nsei_scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_original = nsei_scaler.inverse_transform(y_test.reshape(-1, 1))

# Plotting
plt.figure(figsize=(12, 6))

# Plot training data
plt.plot(train_dates, nsei_scaler.inverse_transform(y_train.reshape(-1, 1)),␣
    ↪label='Training Closing Prices', color='blue')

# Plot testing data
plt.plot(test_dates, y_test_original, label='Testing Closing Prices',␣
    ↪color='red')

# Plot predicted data
plt.plot(test_dates, y_pred_original, label='Predicted Closing Prices',␣
    ↪color='green')

# Set labels, title, legend, and grid
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('NSEI Closing Price Prediction with Random Forest')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```

```
Evaluation Metrics:
RMSE: 90.78732683699333
MAPE: 0.6575203566437353
MBE: 0.781100280562942
MSE: 8242.33871420705
```

```
R²: 0.9994676677653671
R-squared: 0.9994676677653671
RMSE: 90.78732683699333
MSE: 8242.33871420705
MAE: 90.78732683699333
```




```
[ ]: import matplotlib.pyplot as plt
import seaborn as sns

# Predicted vs. Actual values plot
def plot_predicted_vs_actual(y_true, y_pred, title):
    plt.figure(figsize=(12, 6))
    plt.plot(y_true, label="Actual Value")
    plt.plot(y_pred, label="Predicted Value")
    plt.title('Predicted vs Actual Values')
    plt.xlabel('Time Steps')
    plt.ylabel('Value')
    plt.legend()
    plt.show()


plot_predicted_vs_actual(
    nsei_scaler.inverse_transform(nsei_y.reshape(-1, 1)),
    nsei_scaler.inverse_transform(nsei_rf_best.predict(nsei_X).reshape(-1, 1)),
    "NIFTY: Predicted vs Actual"
)
```



[ ]:
```
# prompt: check for underfitting using rmse of train and test data

# Calculate RMSE for training data
y_train_pred = nsei_rf_best.predict(X_train)
train_rmse = rmse(y_train, y_train_pred)

# Calculate RMSE for testing data
y_test_pred = nsei_rf_best.predict(X_test)
test_rmse = rmse(y_test, y_test_pred)

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")

# Check for underfitting
if train_rmse > test_rmse:
  print("Possible underfitting detected.")
elif train_rmse == test_rmse:
  print("The model might be very simple or the dataset might be too small.")
else:
  print("The model is likely not underfitting.")
```

```
Training RMSE: 0.33271205525292735
Testing RMSE: 0.1516687430310018
Possible underfitting detected.
```

[ ]:
```
# prompt: check for overfitting using rmse of train and test data
```

```python
# Calculate RMSE for training data
y_train_pred = nsei_rf_best.predict(X_train)
train_rmse = rmse(y_train, y_train_pred)

# Calculate RMSE for testing data
y_test_pred = nsei_rf_best.predict(X_test)
test_rmse = rmse(y_test, y_test_pred)

print(f"Training RMSE: {train_rmse}")
print(f"Testing RMSE: {test_rmse}")

# Check for overfitting
if test_rmse > train_rmse:
    print("Possible overfitting detected.")
    print("The model is performing significantly better on the training data␣
 ↪than on the test data.")
else:
    print("The model is likely not overfitting.")
```

```
Training RMSE: 0.33271205525292735
Testing RMSE: 0.1516687430310018
The model is likely not overfitting.
```

SVM

[ ]:

```

```