all-models-gdax

November 10, 2024

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

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

[ ]:
```python
GDAX=pd.read_csv('/content/GDAX.csv')
```

[ ]:
```python
GDAX=GDAX.iloc[:,0:11]
```

[ ]:
```python
GDAX.shape
```

[ ]: (2712, 11)

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

[ ]:
```python
GDAX.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2443 entries, 49 to 2711
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          2443 non-null   object
 1   Open          2443 non-null   float64
 2   High          2443 non-null   float64
 3   Low           2443 non-null   float64
 4   Close         2443 non-null   float64
 5   Adj Close     2443 non-null   float64
 6   Volume        2443 non-null   float64
 7   MA_50         2443 non-null   float64
 8   Daily_Return  2443 non-null   float64
 9   Volatility    2443 non-null   float64
 10  Change        2443 non-null   float64
dtypes: float64(10), object(1)
memory usage: 229.0+ KB
```

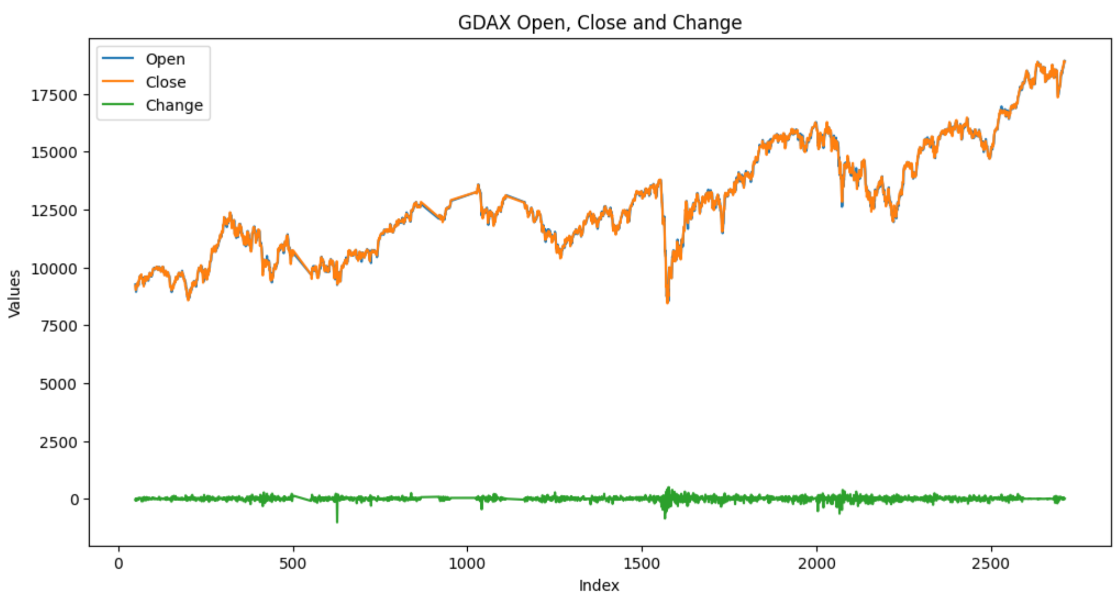
[ ]:
```python
GDAX['Date'] = pd.to_datetime(GDAX['Date']).dt.date
```

[ ]:
```python
GDAX.head()
```

[ ]:
```
          Date         Open         High          Low        Close  \
49  2014-03-12  9257.129883  9267.099609  9142.540039  9188.690430
50  2014-03-13  9200.120117  9226.959961  9017.349609  9017.790039
51  2014-03-14  8939.179688  9094.240234  8913.269531  9056.410156
52  2014-03-17  9047.490234  9197.809570  9047.490234  9180.889648
53  2014-03-18  9172.049805  9315.070313  9105.690430  9242.549805

      Adj Close       Volume        MA_50  Daily_Return  Volatility     Change
49  9188.690430  107430600.0  9493.495645     -0.012796    0.011875 -50.660156
50  9017.790039  113773100.0  9485.850645     -0.018599    0.012296  11.429687
51  9056.410156  141175500.0  9478.275840      0.004283    0.012244 -78.610351
52  9180.889648   86964500.0  9473.333633      0.013745    0.012600  -8.919922
53  9242.549805   99301200.0  9468.060625      0.006716    0.012751  -8.839843
```

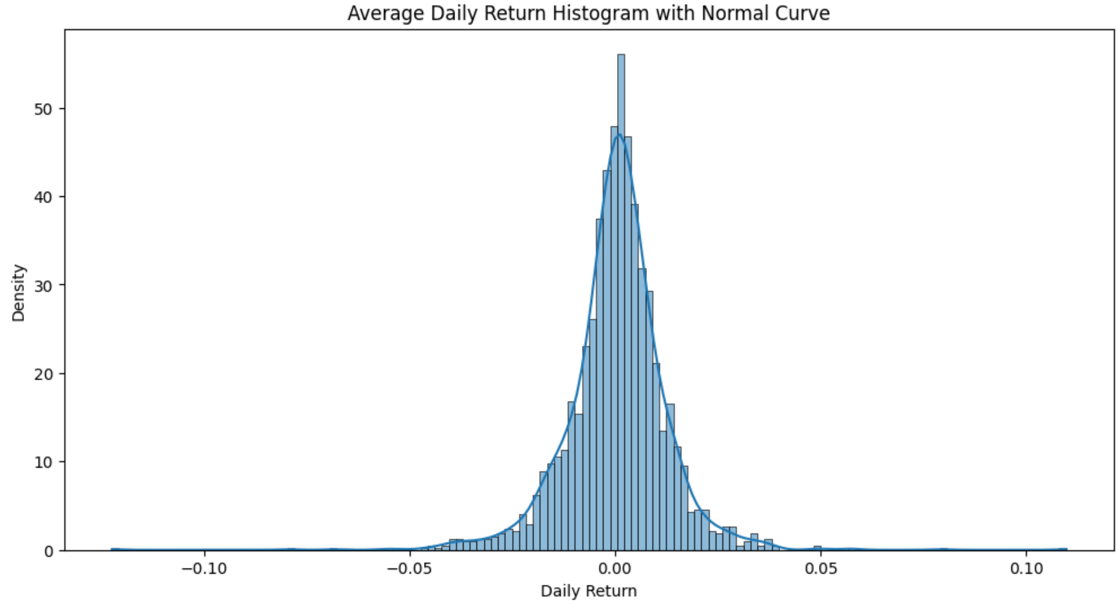
[ ]:
```python
gdata = GDAX[['Open', 'Close', 'Change']]
```

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

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



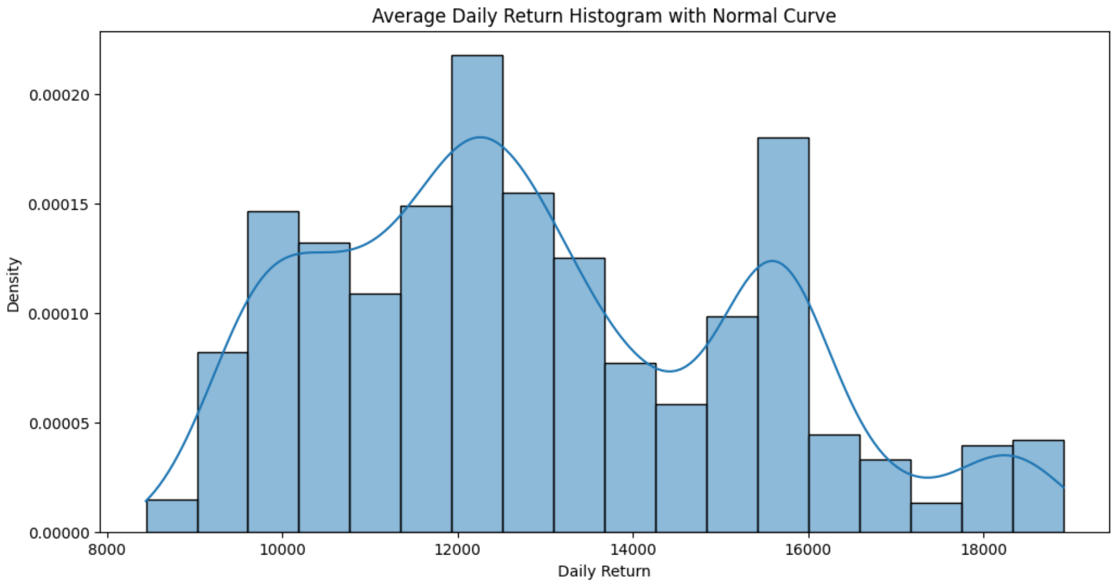
```
[ ]: import numpy as np
     import seaborn as sns
     from scipy.stats import norm

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

     mu, std = norm.fit(GDAX['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()
```

Average Daily Return Histogram with Normal Curve




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

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

     mu, std = norm.fit(GDAX['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()
```



[ ]:
```
#CHECKING OUTLIERS
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(GDAX)
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: 127
MA_50: 0
Daily_Return: 144
Volatility: 134
Change: 150
```

[ ]:
```
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(GDAX)
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: 5.20%
MA_50: 0.00%
Daily_Return: 5.89%
Volatility: 5.49%
Change: 6.14%
```

[ ]:
```
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(GDAX)
```

ARIMA

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

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

[ ]:
```
          Date         Open         High          Low     Close    Adj Close  \
49  2014-03-12  9257.129883  9267.099609  9142.540039  9.125729  9188.690430
50  2014-03-13  9200.120117  9226.959961  9017.349609  9.106955  9017.790039
51  2014-03-14  8939.179688  9094.240234  8913.269531  9.111228  9056.410156
52  2014-03-17  9047.490234  9197.809570  9047.490234  9.124879  9180.889648
53  2014-03-18  9172.049805  9315.070313  9105.690430  9.131573  9242.549805

         Volume        MA_50  Daily_Return  Volatility     Change
49  107430600.0  9493.495645     -0.012796    0.011875 -50.660156
50  113773100.0  9485.850645     -0.018599    0.012296  11.429687
51  141175500.0  9478.275840      0.004283    0.012244 -78.610351
52   86964500.0  9473.333633      0.013745    0.012600  -8.919922
53   99301200.0  9468.060625      0.006716    0.012751  -8.839843
```

[ ]:
```
!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
24.9 MB/s eta 0:00:00
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

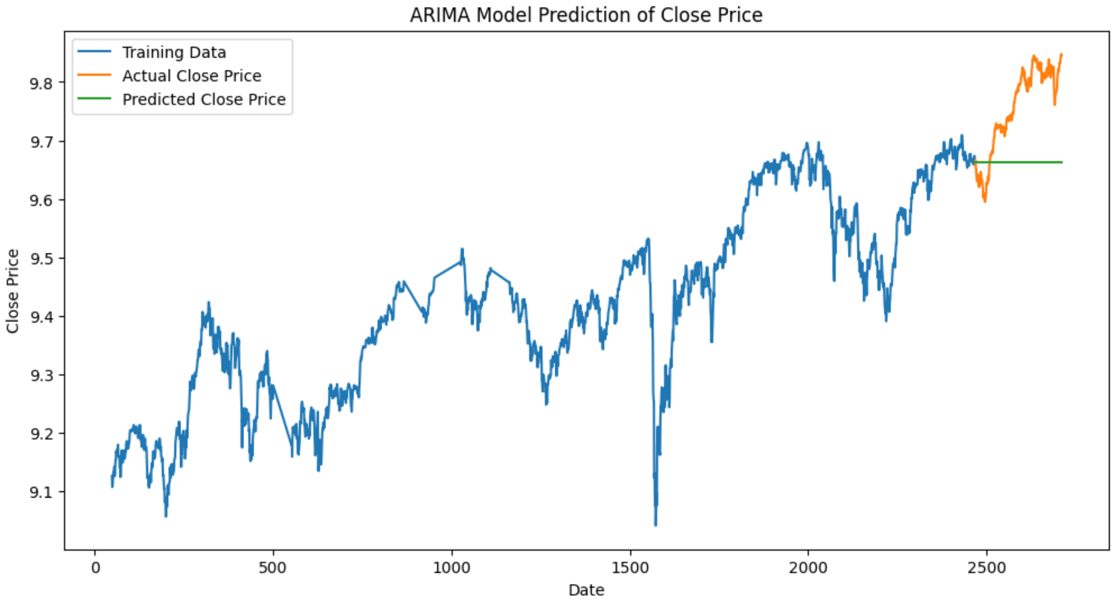
[ ]:
```python
# prompt: write code for arima using autoarima to predict closing price in
 ↪given dataset. use training as 90% data

from pmdarima import auto_arima
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 = GDAX['Close'][:-int(len(GDAX) * 0.1)]
test_data = GDAX['Close'][-int(len(GDAX) * 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=-12882.257, Time=2.94 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-12883.872, Time=1.61 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-12883.127, Time=0.67 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-12883.026, Time=2.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-12885.082, Time=0.30 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 7.640 seconds
RMSE: 0.1162730892731706
```

```
/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(
```

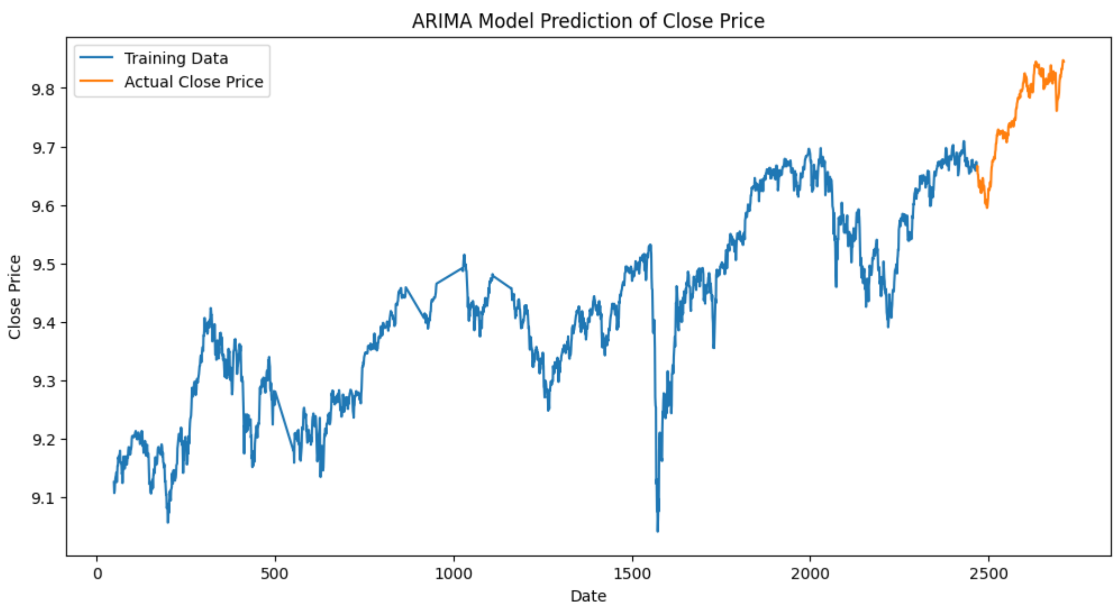


[ ]:
```
# prompt: find out confusion matrix,rmse,mse and other evaluation matrics for
↪the above fir

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)
```

```python
# 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: 0.1162730892731706
MSE: 0.013519431289126698
MAE: 0.10300458225595228
R-squared: -1.6030721853345544
```

[ ]:
```
# 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
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: 0.1950323538540301
Testing RMSE: 0.1162730892731706
The model doesn't appear to be overfitting significantly.
```

[ ]:
```
# 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.")
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: 0.1950323538540301
Testing RMSE: 0.1162730892731706
Baseline RMSE: 0.34421635537804307
The model doesn't appear to be underfitting significantly.
```

GRU

[ ]:
```
!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 = NSEI.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
```

```python
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))
```

```
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)
```

```
58/58 - 5s - 78ms/step - loss: 0.0293
Epoch 2/100
58/58 - 0s - 3ms/step - loss: 3.1997e-04
Epoch 3/100
58/58 - 0s - 5ms/step - loss: 1.4808e-04
Epoch 4/100
58/58 - 0s - 5ms/step - loss: 1.0100e-04
Epoch 5/100
58/58 - 0s - 4ms/step - loss: 7.3367e-05
Epoch 6/100
58/58 - 0s - 4ms/step - loss: 5.4226e-05
Epoch 7/100
58/58 - 0s - 4ms/step - loss: 4.3725e-05
Epoch 8/100
58/58 - 0s - 4ms/step - loss: 3.6452e-05
Epoch 9/100
58/58 - 0s - 5ms/step - loss: 3.0382e-05
Epoch 10/100
58/58 - 0s - 5ms/step - loss: 2.6776e-05
Epoch 11/100
58/58 - 0s - 5ms/step - loss: 2.2229e-05
Epoch 12/100
58/58 - 0s - 6ms/step - loss: 1.8753e-05
```

```
Epoch 13/100
58/58 - 0s - 4ms/step - loss: 1.5741e-05
Epoch 14/100
58/58 - 0s - 5ms/step - loss: 1.2968e-05
Epoch 15/100
58/58 - 0s - 3ms/step - loss: 1.0742e-05
Epoch 16/100
58/58 - 0s - 4ms/step - loss: 9.0818e-06
Epoch 17/100
58/58 - 0s - 4ms/step - loss: 7.5173e-06
Epoch 18/100
58/58 - 0s - 5ms/step - loss: 6.3704e-06
Epoch 19/100
58/58 - 0s - 5ms/step - loss: 5.5985e-06
Epoch 20/100
58/58 - 0s - 5ms/step - loss: 4.6992e-06
Epoch 21/100
58/58 - 0s - 3ms/step - loss: 4.4876e-06
Epoch 22/100
58/58 - 0s - 5ms/step - loss: 3.7279e-06
Epoch 23/100
58/58 - 0s - 3ms/step - loss: 3.3162e-06
Epoch 24/100
58/58 - 0s - 4ms/step - loss: 3.0945e-06
Epoch 25/100
58/58 - 0s - 6ms/step - loss: 2.9275e-06
Epoch 26/100
58/58 - 0s - 5ms/step - loss: 2.6967e-06
Epoch 27/100
58/58 - 0s - 6ms/step - loss: 2.6167e-06
Epoch 28/100
58/58 - 0s - 5ms/step - loss: 2.4640e-06
Epoch 29/100
58/58 - 0s - 6ms/step - loss: 2.5660e-06
Epoch 30/100
58/58 - 1s - 11ms/step - loss: 2.4811e-06
Epoch 31/100
58/58 - 1s - 10ms/step - loss: 2.2361e-06
Epoch 32/100
58/58 - 0s - 3ms/step - loss: 2.3012e-06
Epoch 33/100
58/58 - 0s - 6ms/step - loss: 2.2529e-06
Epoch 34/100
58/58 - 0s - 4ms/step - loss: 2.2965e-06
Epoch 35/100
58/58 - 0s - 3ms/step - loss: 2.0858e-06
Epoch 36/100
58/58 - 0s - 4ms/step - loss: 2.1576e-06
```

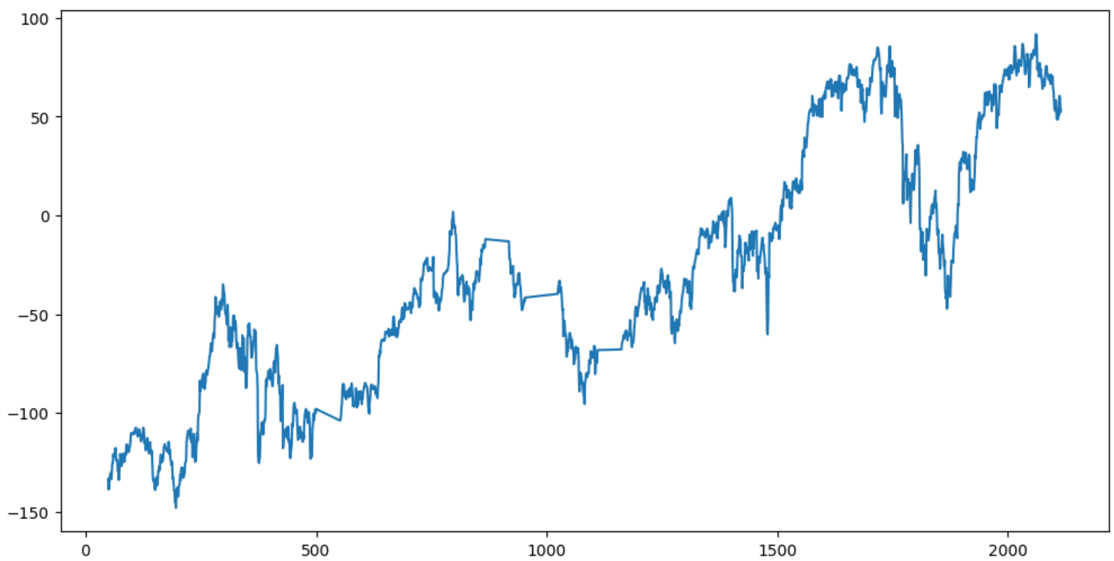
```
Epoch 37/100
58/58 - 0s - 4ms/step - loss: 2.6648e-06
Epoch 38/100
58/58 - 0s - 5ms/step - loss: 2.2893e-06
Epoch 39/100
58/58 - 0s - 5ms/step - loss: 2.0703e-06
Epoch 40/100
58/58 - 0s - 5ms/step - loss: 2.1210e-06
Epoch 41/100
58/58 - 0s - 5ms/step - loss: 1.9458e-06
Epoch 42/100
58/58 - 0s - 5ms/step - loss: 2.2708e-06
Epoch 43/100
58/58 - 0s - 5ms/step - loss: 2.2012e-06
Epoch 44/100
58/58 - 0s - 4ms/step - loss: 2.2213e-06
Epoch 45/100
58/58 - 0s - 4ms/step - loss: 2.1843e-06
Epoch 46/100
58/58 - 0s - 3ms/step - loss: 2.0168e-06
Epoch 47/100
58/58 - 0s - 5ms/step - loss: 1.9648e-06
Epoch 48/100
58/58 - 0s - 3ms/step - loss: 2.5911e-06
Epoch 49/100
58/58 - 0s - 4ms/step - loss: 2.2269e-06
Epoch 50/100
58/58 - 0s - 3ms/step - loss: 2.1684e-06
Epoch 51/100
58/58 - 0s - 5ms/step - loss: 2.0010e-06
Epoch 52/100
58/58 - 0s - 3ms/step - loss: 2.0391e-06
Epoch 53/100
58/58 - 0s - 6ms/step - loss: 1.9961e-06
Epoch 54/100
58/58 - 0s - 5ms/step - loss: 1.8764e-06
Epoch 55/100
58/58 - 0s - 5ms/step - loss: 1.9603e-06
Epoch 56/100
58/58 - 0s - 6ms/step - loss: 2.2231e-06
Epoch 57/100
58/58 - 0s - 5ms/step - loss: 1.9270e-06
Epoch 58/100
58/58 - 0s - 5ms/step - loss: 2.1527e-06
Epoch 59/100
58/58 - 0s - 6ms/step - loss: 1.9720e-06
Epoch 60/100
58/58 - 0s - 5ms/step - loss: 1.9746e-06
```

```
Epoch 61/100
58/58 - 0s - 5ms/step - loss: 2.0177e-06
Epoch 62/100
58/58 - 0s - 5ms/step - loss: 2.5135e-06
Epoch 63/100
58/58 - 0s - 4ms/step - loss: 1.9025e-06
Epoch 64/100
58/58 - 0s - 5ms/step - loss: 2.5595e-06
Epoch 65/100
58/58 - 0s - 5ms/step - loss: 2.0724e-06
Epoch 66/100
58/58 - 0s - 6ms/step - loss: 2.1470e-06
Epoch 67/100
58/58 - 0s - 5ms/step - loss: 1.8341e-06
Epoch 68/100
58/58 - 0s - 4ms/step - loss: 2.6878e-06
Epoch 69/100
58/58 - 0s - 7ms/step - loss: 2.0096e-06
Epoch 70/100
58/58 - 1s - 10ms/step - loss: 2.0372e-06
Epoch 71/100
58/58 - 0s - 5ms/step - loss: 2.3261e-06
Epoch 72/100
58/58 - 1s - 12ms/step - loss: 2.6709e-06
Epoch 73/100
58/58 - 0s - 6ms/step - loss: 1.9887e-06
Epoch 74/100
58/58 - 0s - 6ms/step - loss: 1.6917e-06
Epoch 75/100
58/58 - 0s - 8ms/step - loss: 2.2988e-06
Epoch 76/100
58/58 - 0s - 4ms/step - loss: 2.7470e-06
Epoch 77/100
58/58 - 0s - 5ms/step - loss: 3.8786e-06
Epoch 78/100
58/58 - 0s - 5ms/step - loss: 2.4301e-06
Epoch 79/100
58/58 - 0s - 5ms/step - loss: 1.8835e-06
Epoch 80/100
58/58 - 0s - 4ms/step - loss: 2.2489e-06
Epoch 81/100
58/58 - 0s - 4ms/step - loss: 4.0479e-06
Epoch 82/100
58/58 - 0s - 5ms/step - loss: 3.7111e-06
Epoch 83/100
58/58 - 0s - 4ms/step - loss: 5.8541e-06
Epoch 84/100
58/58 - 0s - 4ms/step - loss: 3.0934e-06
```

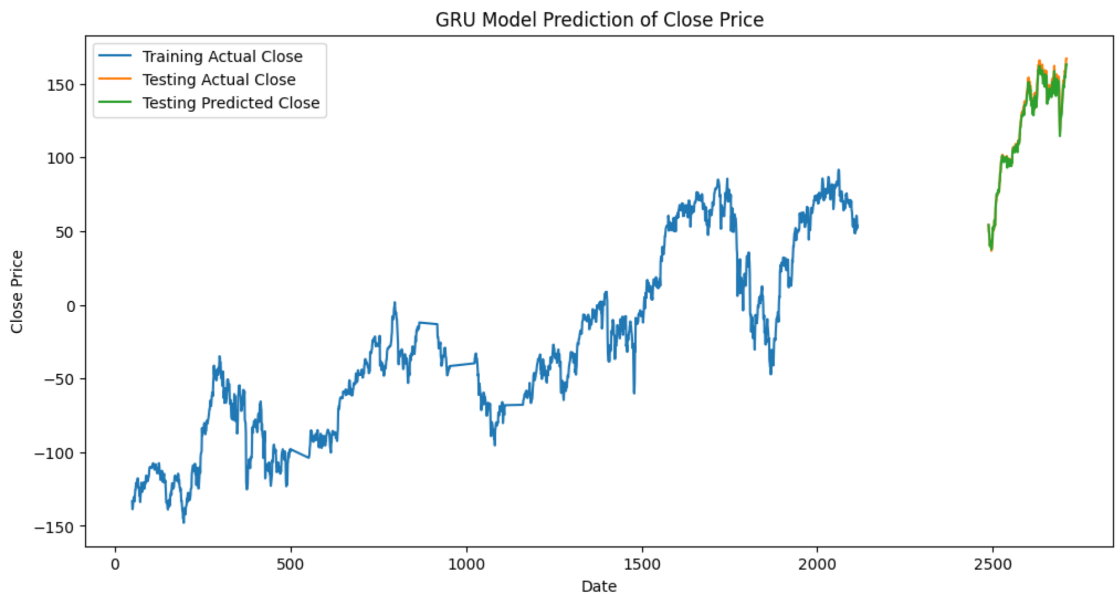
```
Epoch 85/100
58/58 - 0s - 5ms/step - loss: 2.4395e-06
Epoch 86/100
58/58 - 0s - 3ms/step - loss: 3.6931e-06
Epoch 87/100
58/58 - 0s - 5ms/step - loss: 2.1994e-06
Epoch 88/100
58/58 - 0s - 6ms/step - loss: 1.9340e-06
Epoch 89/100
58/58 - 0s - 4ms/step - loss: 4.1362e-06
Epoch 90/100
58/58 - 0s - 4ms/step - loss: 5.5146e-06
Epoch 91/100
58/58 - 0s - 6ms/step - loss: 1.9529e-06
Epoch 92/100
58/58 - 0s - 4ms/step - loss: 7.8184e-06
Epoch 93/100
58/58 - 0s - 4ms/step - loss: 1.6788e-06
Epoch 94/100
58/58 - 0s - 4ms/step - loss: 2.1453e-06
Epoch 95/100
58/58 - 0s - 4ms/step - loss: 1.8164e-06
Epoch 96/100
58/58 - 0s - 4ms/step - loss: 2.7590e-06
Epoch 97/100
58/58 - 0s - 5ms/step - loss: 1.9067e-06
Epoch 98/100
58/58 - 0s - 5ms/step - loss: 2.9335e-06
Epoch 99/100
58/58 - 0s - 4ms/step - loss: 3.6461e-06
Epoch 100/100
58/58 - 0s - 3ms/step - loss: 4.4418e-06
58/58                 1s 8ms/step
7/7               0s 3ms/step
Train Score: 0.36 RMSE
Test Score: 2.19 RMSE
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-50-17a70c60a3e6> in <cell line: 60>()
     58 plt.figure(figsize=(12, 6))
     59 plt.plot(GDAX.index[:train_size], trainY, label='Training Actual Close')
---> 60 plt.plot(GDAX.index[train_size:], testY, label='Testing Actual Close')
     61 plt.plot(GDAX.index[train_size:], testPredict, label='Testing Predicted
  ↪Close')
     62 plt.title('GRU Model Prediction of Close Price')
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py in plot(scalex,␣
↪scaley, data, *args, **kwargs)
   3576     **kwargs,
   3577 ) -> list[Line2D]:
-> 3578     return gca().plot(
   3579         *args,
   3580         scalex=scalex,

/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_axes.py in plot(self,␣
↪scalex, scaley, data, *args, **kwargs)
   1719         """
   1720         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1721         lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1722         for line in lines:
   1723             self.add_line(line)

/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_base.py in␣
↪__call__(self, axes, data, *args, **kwargs)
    301                 this += args[0],
    302                 args = args[1:]
--> 303             yield from self._plot_args(
    304                 axes, this, kwargs,␣
↪ambiguous_fmt_datakey=ambiguous_fmt_datakey)
    305

/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_base.py in␣
↪_plot_args(self, axes, tup, kwargs, return_kwargs, ambiguous_fmt_datakey)
    497
    498         if x.shape[0] != y.shape[0]:
--> 499             raise ValueError(f"x and y must have same first dimension,␣
↪but "
    500                              f"have shapes {x.shape} and {y.shape}")
    501         if x.ndim > 2 or y.ndim > 2:

ValueError: x and y must have same first dimension, but have shapes (596,) and␣
↪(206,)
```



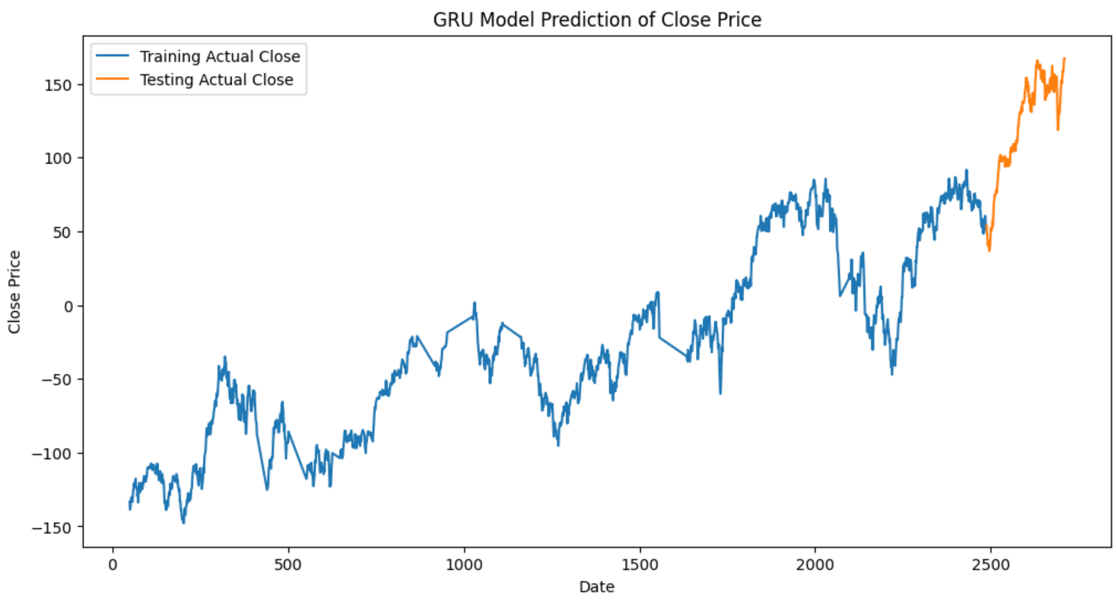
[ ]:
```
#Plot the results
plt.figure(figsize=(12, 6))
plt.plot(GDAX.index[:train_size], trainY, label='Training Actual Close')
plt.plot(data.index[train_size:], testY, label='Testing Actual Close')
plt.plot(data.index[train_size:], testPredict, label='Testing Predicted Close')
plt.title('GRU Model Prediction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



[ ]:
```python
# 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: 2.1924512370669698
MSE: 4.806842426916486
MAE: 1.80155511499374
R-squared: 0.995944103467981
```

[ ]:
```python
plt.figure(figsize=(12, 6))
plt.plot(data.index[:train_size], trainY, label='Training Actual Close')
plt.plot(data.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))

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: 0.35813716098382387
Testing RMSE: 2.1924512370669698
The model doesn't appear to be overfitting significantly.
Baseline RMSE: 153.89794093431553
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 'NSEI' DataFrame is already loaded and prepared
# Extract relevant features for prediction (all attributes except 'Date')
data = GDAX.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)]
```

```python
# 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(GDAX.index[:train_size], trainY, label='Training Actual Close')
plt.plot(GDAX.index[train_size:], testY, label='Testing Actual Close')
```

```python
plt.plot(GDAX.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)
```

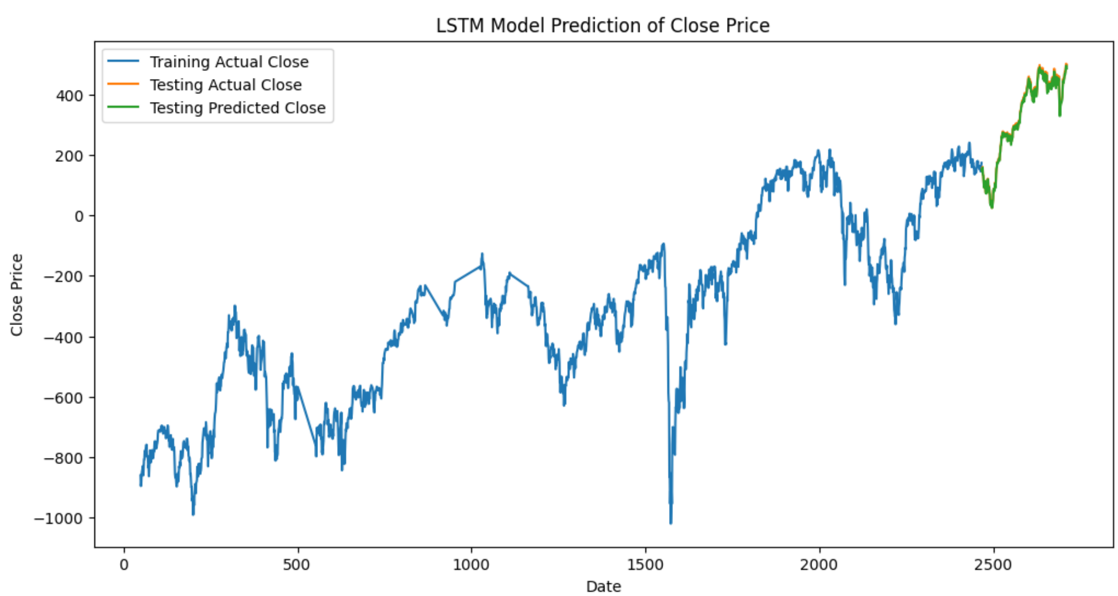
```
69/69 - 3s - 46ms/step - loss: 0.0704
Epoch 2/100
69/69 - 0s - 5ms/step - loss: 0.0030
Epoch 3/100
69/69 - 0s - 4ms/step - loss: 0.0012
Epoch 4/100
69/69 - 0s - 4ms/step - loss: 0.0010
Epoch 5/100
69/69 - 0s - 3ms/step - loss: 8.9606e-04
Epoch 6/100
69/69 - 0s - 4ms/step - loss: 7.6451e-04
Epoch 7/100
69/69 - 0s - 5ms/step - loss: 6.1827e-04
Epoch 8/100
69/69 - 0s - 4ms/step - loss: 4.8226e-04
Epoch 9/100
69/69 - 0s - 4ms/step - loss: 3.5537e-04
Epoch 10/100
69/69 - 0s - 3ms/step - loss: 2.5398e-04
Epoch 11/100
69/69 - 0s - 4ms/step - loss: 1.7955e-04
Epoch 12/100
69/69 - 0s - 5ms/step - loss: 1.3009e-04
Epoch 13/100
69/69 - 0s - 4ms/step - loss: 1.0493e-04
Epoch 14/100
69/69 - 0s - 3ms/step - loss: 8.9216e-05
Epoch 15/100
69/69 - 0s - 3ms/step - loss: 7.7265e-05
Epoch 16/100
69/69 - 0s - 3ms/step - loss: 6.8434e-05
Epoch 17/100
69/69 - 0s - 3ms/step - loss: 5.8283e-05
```

```
Epoch 18/100
69/69 - 0s - 3ms/step - loss: 5.1367e-05
Epoch 19/100
69/69 - 0s - 3ms/step - loss: 4.4511e-05
Epoch 20/100
69/69 - 0s - 3ms/step - loss: 3.9080e-05
Epoch 21/100
69/69 - 0s - 6ms/step - loss: 3.4241e-05
Epoch 22/100
69/69 - 0s - 5ms/step - loss: 3.0631e-05
Epoch 23/100
69/69 - 1s - 9ms/step - loss: 2.6442e-05
Epoch 24/100
69/69 - 0s - 5ms/step - loss: 2.3656e-05
Epoch 25/100
69/69 - 0s - 5ms/step - loss: 2.0190e-05
Epoch 26/100
69/69 - 1s - 10ms/step - loss: 1.8409e-05
Epoch 27/100
69/69 - 1s - 8ms/step - loss: 1.6790e-05
Epoch 28/100
69/69 - 0s - 3ms/step - loss: 1.6182e-05
Epoch 29/100
69/69 - 0s - 3ms/step - loss: 1.4197e-05
Epoch 30/100
69/69 - 0s - 3ms/step - loss: 1.3472e-05
Epoch 31/100
69/69 - 0s - 5ms/step - loss: 1.2301e-05
Epoch 32/100
69/69 - 0s - 4ms/step - loss: 1.0707e-05
Epoch 33/100
69/69 - 0s - 4ms/step - loss: 1.0545e-05
Epoch 34/100
69/69 - 0s - 3ms/step - loss: 9.4696e-06
Epoch 35/100
69/69 - 0s - 4ms/step - loss: 9.4561e-06
Epoch 36/100
69/69 - 0s - 3ms/step - loss: 8.3996e-06
Epoch 37/100
69/69 - 0s - 4ms/step - loss: 8.1869e-06
Epoch 38/100
69/69 - 0s - 4ms/step - loss: 7.9295e-06
Epoch 39/100
69/69 - 0s - 4ms/step - loss: 7.4945e-06
Epoch 40/100
69/69 - 0s - 5ms/step - loss: 6.4791e-06
Epoch 41/100
69/69 - 0s - 4ms/step - loss: 6.5159e-06
```

```
Epoch 42/100
69/69 - 0s - 3ms/step - loss: 6.0222e-06
Epoch 43/100
69/69 - 0s - 4ms/step - loss: 6.0296e-06
Epoch 44/100
69/69 - 0s - 5ms/step - loss: 5.6278e-06
Epoch 45/100
69/69 - 0s - 4ms/step - loss: 5.1230e-06
Epoch 46/100
69/69 - 0s - 3ms/step - loss: 5.0362e-06
Epoch 47/100
69/69 - 0s - 5ms/step - loss: 4.8824e-06
Epoch 48/100
69/69 - 0s - 3ms/step - loss: 4.7039e-06
Epoch 49/100
69/69 - 0s - 4ms/step - loss: 4.8322e-06
Epoch 50/100
69/69 - 0s - 4ms/step - loss: 4.7055e-06
Epoch 51/100
69/69 - 0s - 4ms/step - loss: 4.1727e-06
Epoch 52/100
69/69 - 0s - 4ms/step - loss: 3.9043e-06
Epoch 53/100
69/69 - 0s - 4ms/step - loss: 3.8152e-06
Epoch 54/100
69/69 - 0s - 3ms/step - loss: 4.1249e-06
Epoch 55/100
69/69 - 0s - 4ms/step - loss: 3.5045e-06
Epoch 56/100
69/69 - 0s - 3ms/step - loss: 3.6980e-06
Epoch 57/100
69/69 - 0s - 5ms/step - loss: 3.2372e-06
Epoch 58/100
69/69 - 0s - 4ms/step - loss: 3.3727e-06
Epoch 59/100
69/69 - 0s - 4ms/step - loss: 3.1740e-06
Epoch 60/100
69/69 - 0s - 3ms/step - loss: 3.1027e-06
Epoch 61/100
69/69 - 0s - 3ms/step - loss: 2.9781e-06
Epoch 62/100
69/69 - 0s - 3ms/step - loss: 2.9849e-06
Epoch 63/100
69/69 - 0s - 4ms/step - loss: 3.1166e-06
Epoch 64/100
69/69 - 0s - 4ms/step - loss: 3.0578e-06
Epoch 65/100
69/69 - 0s - 6ms/step - loss: 3.0109e-06
```

```
Epoch 66/100
69/69 - 1s - 8ms/step - loss: 3.5530e-06
Epoch 67/100
69/69 - 1s - 9ms/step - loss: 3.9000e-06
Epoch 68/100
69/69 - 0s - 5ms/step - loss: 3.0639e-06
Epoch 69/100
69/69 - 1s - 9ms/step - loss: 2.3590e-06
Epoch 70/100
69/69 - 0s - 7ms/step - loss: 2.6745e-06
Epoch 71/100
69/69 - 0s - 3ms/step - loss: 2.5831e-06
Epoch 72/100
69/69 - 0s - 3ms/step - loss: 2.2229e-06
Epoch 73/100
69/69 - 0s - 5ms/step - loss: 2.4044e-06
Epoch 74/100
69/69 - 0s - 4ms/step - loss: 3.2553e-06
Epoch 75/100
69/69 - 0s - 4ms/step - loss: 2.2967e-06
Epoch 76/100
69/69 - 0s - 3ms/step - loss: 2.3726e-06
Epoch 77/100
69/69 - 0s - 4ms/step - loss: 2.3839e-06
Epoch 78/100
69/69 - 0s - 3ms/step - loss: 2.6425e-06
Epoch 79/100
69/69 - 0s - 5ms/step - loss: 3.4441e-06
Epoch 80/100
69/69 - 0s - 3ms/step - loss: 2.7001e-06
Epoch 81/100
69/69 - 0s - 3ms/step - loss: 3.0327e-06
Epoch 82/100
69/69 - 0s - 3ms/step - loss: 3.0949e-06
Epoch 83/100
69/69 - 0s - 5ms/step - loss: 2.6115e-06
Epoch 84/100
69/69 - 0s - 4ms/step - loss: 2.8787e-06
Epoch 85/100
69/69 - 0s - 3ms/step - loss: 2.2024e-06
Epoch 86/100
69/69 - 0s - 4ms/step - loss: 2.3126e-06
Epoch 87/100
69/69 - 0s - 3ms/step - loss: 3.7842e-06
Epoch 88/100
69/69 - 0s - 3ms/step - loss: 2.5407e-06
Epoch 89/100
69/69 - 0s - 4ms/step - loss: 2.8597e-06
```

```
Epoch 90/100
69/69 - 0s - 4ms/step - loss: 2.5828e-06
Epoch 91/100
69/69 - 0s - 3ms/step - loss: 2.3016e-06
Epoch 92/100
69/69 - 0s - 4ms/step - loss: 2.6982e-06
Epoch 93/100
69/69 - 0s - 3ms/step - loss: 3.1930e-06
Epoch 94/100
69/69 - 0s - 3ms/step - loss: 3.0266e-06
Epoch 95/100
69/69 - 0s - 4ms/step - loss: 3.3530e-06
Epoch 96/100
69/69 - 0s - 3ms/step - loss: 2.3566e-06
Epoch 97/100
69/69 - 0s - 3ms/step - loss: 3.8497e-06
Epoch 98/100
69/69 - 0s - 4ms/step - loss: 4.0642e-06
Epoch 99/100
69/69 - 0s - 4ms/step - loss: 2.0915e-06
Epoch 100/100
69/69 - 0s - 4ms/step - loss: 2.5290e-06
69/69            1s 6ms/step
8/8            0s 2ms/step
Train Score: 2.46 RMSE
Test Score: 5.31 RMSE
```



[ ]:
```python
# 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: 5.305712681035156
MSE: 28.150587053697265
MAE: 4.77745404791346
R-squared: 0.9984778725139708
```

[ ]:
```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: 2.463002573765655
Testing RMSE: 5.305712681035156
```

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

```python
# 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: 2.463002573765655
Testing RMSE: 5.305712681035156
The model doesn't appear to be overfitting significantly.
Baseline RMSE: 648.3161865304495
```

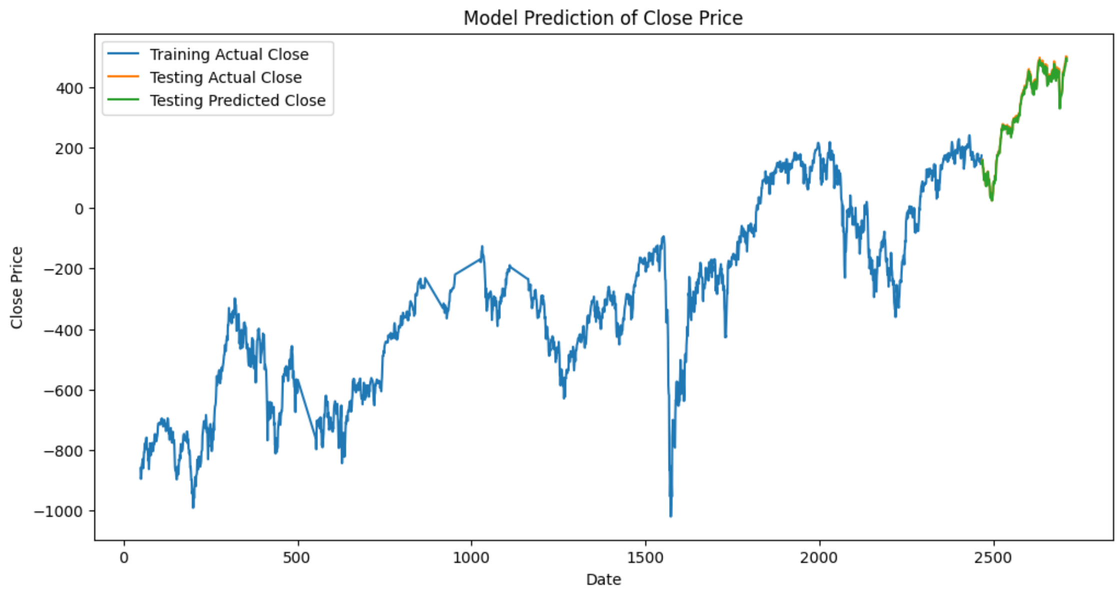
The model doesn't appear to be underfitting significantly.

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 = GDAX.drop('Date', axis=1)
     y = GDAX['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(GDAX.index[:train_size], trainY, label='Training Actual Close')
     plt.plot(GDAX.index[train_size:], testY, label='Testing Actual Close')
     plt.plot(GDAX.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))
```

```python
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: 1.1119436698492558e-15
MSE: 1.2364187249178307e-30
MAE: 6.960418636017308e-16
R-squared: 1.0
Training RMSE: 1.1429758771645269e-15
Testing RMSE: 1.1119436698492558e-15
The model doesn't appear to be overfitting significantly.
Baseline RMSE: 0.16732792585319914
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
         gdax = 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(gdax)

         return scaled_data, scaler
```

```
[ ]: gdax_data, gdax_scaler = preprocess_data(GDAX)
```

```
[ ]: 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)


     gdax_X, gdax_y = create_dataset(gdax_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_


gdax_rf_best = perform_grid_search(gdax_X, gdax_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:
--------------------------------------------------------------------------------
270 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 {'sqrt', 'log2'} or None. Got 'auto'
instead.

--------------------------------------------------------------------------------
270 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/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 0.8572851  0.85805401 0.85802324
 0.86284577 0.86256298 0.86244653 0.86767591 0.86823437 0.86827532
 0.86500822 0.86544973 0.86559003 0.86632562 0.86649958 0.8665341
 0.86834032 0.86874284 0.86874341 0.86801841 0.86889176 0.86881958
 0.86801841 0.86889176 0.86881958 0.867753   0.86844609 0.8683794
 0.8572851  0.85805401 0.85802324 0.86284577 0.86256298 0.86244653
 0.86767591 0.86823437 0.86827532 0.86500822 0.86544973 0.86559003
 0.86632562 0.86649958 0.8665341  0.86834032 0.86874284 0.86874341
 0.86801841 0.86889176 0.86881958 0.86801841 0.86889176 0.86881958
 0.867753   0.86844609 0.8683794         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
 0.854993   0.85575821 0.85573163 0.86199178 0.86170388 0.86158477
 0.86746164 0.86803298 0.8680745  0.86456879 0.86498728 0.8651259
 0.86595734 0.86612292 0.86615285 0.86823619 0.86863595 0.86863613
 0.86797532 0.86885201 0.86877836 0.86797532 0.86885201 0.86877836
 0.86772561 0.86842088 0.86835326 0.854993   0.85575821 0.85573163
 0.86199178 0.86170388 0.86158477 0.86746164 0.86803298 0.8680745
 0.86456879 0.86498728 0.8651259  0.86595734 0.86612292 0.86615285
 0.86823619 0.86863595 0.86863613 0.86797532 0.86885201 0.86877836
 0.86797532 0.86885201 0.86877836 0.86772561 0.86842088 0.86835326
```

```
         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  0.85499314  0.85575824  0.85573144
 0.86199182  0.86170384  0.86158474  0.86746167  0.86803298  0.8680745
 0.86456861  0.8649872   0.86512584  0.86595717  0.86612284  0.86615279
 0.86823619  0.86863595  0.86863613  0.86797532  0.86885201  0.86877836
 0.86797532  0.86885201  0.86877836  0.86772561  0.86842088  0.86835326
 0.85499314  0.85575824  0.85573144  0.86199182  0.86170384  0.86158474
 0.86746167  0.86803298  0.8680745   0.86456861  0.8649872   0.86512584
 0.86595717  0.86612284  0.86615279  0.86823619  0.86863595  0.86863613
 0.86797532  0.86885201  0.86877836  0.86797532  0.86885201  0.86877836
 0.86772561  0.86842088  0.86835326         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
 0.85499314  0.85575824  0.85573144  0.86199182  0.86170384  0.86158474
 0.86746167  0.86803298  0.8680745   0.86456861  0.8649872   0.86512584
 0.86595717  0.86612284  0.86615279  0.86823619  0.86863595  0.86863613
 0.86797532  0.86885201  0.86877836  0.86797532  0.86885201  0.86877836
 0.86772561  0.86842088  0.86835326  0.85499314  0.85575824  0.85573144
 0.86199182  0.86170384  0.86158474  0.86746167  0.86803298  0.8680745
 0.86456861  0.8649872   0.86512584  0.86595717  0.86612284  0.86615279
 0.86823619  0.86863595  0.86863613  0.86797532  0.86885201  0.86877836
 0.86797532  0.86885201  0.86877836  0.86772561  0.86842088  0.86835326]
  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)
```

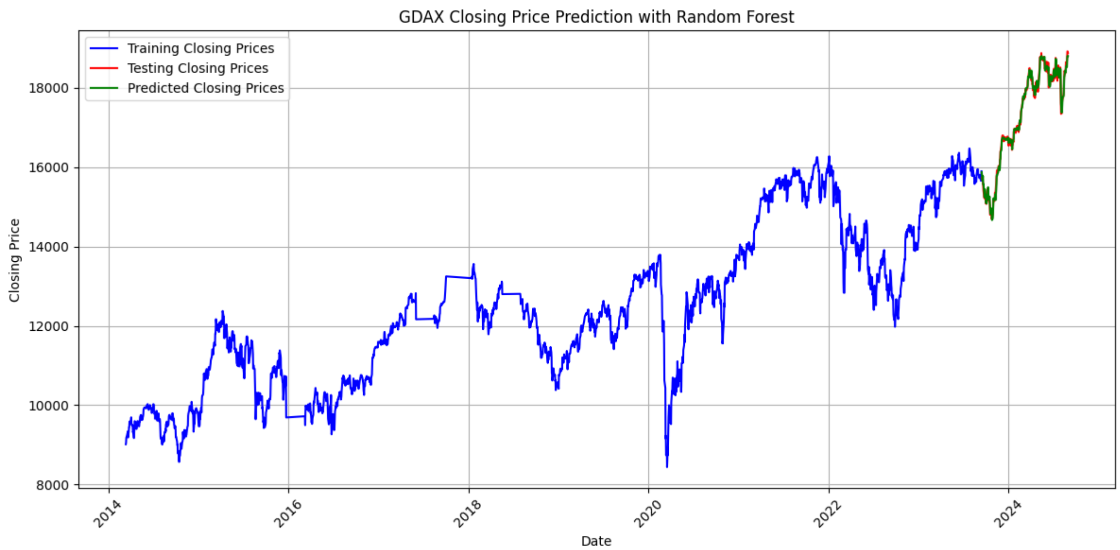
```
# Function to evaluate the model
def evaluate_model(model, X, y, scaler):
    # Predict using the model
    predicted = model.predict(X)

    # 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
```

[ ]:
```
gdax_rmse, gdax_mape, gdax_mbe, _msegdax, gdax_rsquare =␣
↪evaluate_model(gdax_rf_best, gdax_X, gdax_y, gdax_scaler)
```

```
Evaluation Metrics:
RMSE: 126.02044023535666
MAPE: 0.7167361274882209
MBE: -0.3950433729947939
MSE: 15881.1513571131
R²: 0.9972742173567148
```

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

# Split the data into training and testing sets (assuming `gdax_X` and `gdax_y`␣
↪are defined)
X_train, X_test, y_train, y_test = train_test_split(gdax_X, gdax_y, test_size=0.
↪1, random_state=42, shuffle=False)

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

# Ensure the dates align with the data split (assuming NSEI is sorted by date)
```

```
train_dates = GDAX['Date'][:len(y_train)].reset_index(drop=True)
test_dates = GDAX['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 = gdax_scaler.inverse_transform(y_pred.reshape(-1, 1))
y_train_original = gdax_scaler.inverse_transform(y_train.reshape(-1, 1))
y_test_original = gdax_scaler.inverse_transform(y_test.reshape(-1, 1))

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

# Plot training data
plt.plot(train_dates, y_train_original, 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('GDAX 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()
```



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

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

# Calculate RMSE for testing data
y_test_pred = gdax_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: 0.012264217999278833
Testing RMSE: 0.009745667916111226
Possible underfitting detected.
```

[ ]:
```
# Calculate RMSE for training data
y_train_pred = gdax_rf_best.predict(X_train)
train_rmse = rmse(y_train, y_train_pred)
```

```
# Calculate RMSE for testing data
y_test_pred = gdax_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: 0.01226421799927883
Testing RMSE: 0.009745667916111226
The model is likely not overfitting.
```

[ ]: