```
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
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv (r'/content/npa1.csv')
data.head()
```

	S.N.	Open	High	Low	Close	Change	Per Change (%)	Turnover	Dat
0	1	2,772.23	2,805.57	2,742.53	2,755.62	-5.29	-0.19	21,713,431,007.73	8/1/202
1	2	2,708.24	2,791.02	2,707.57	2,760.90	64.25	2.38	21,911,503,161.05	7/31/202
2	3	2,669.41	2,709.89	2,609.02	2,696.65	35.55	1.33	17,266,391,542.96	7/30/202
3	1	2,706.73	2,737.68	2,655.15	2,661.09	-20.46	-0.76	19,609,177,106.29	7/29/202
4	2	2,606.42	2,708.58	2,606.40	2,681.56	113.42	4.41	15,810,596,235.45	7/28/202

Next steps: Generate code with data View recommended plots New interactive sheet

```
# Convert the Date column to datetime format
data['Date'] = pd.to_datetime(data['Date'], format='%m/%d/%Y')

data = data.sort_values(by='Date')
data.head()
```

	S.N.	Open	High	Low	Close	Change	Per Change (%)	Turnover	Date
2196	314	904	904	904	904	1.35	0.15	0	2015-01-01
2195	313	918	918	918	918	13.82	1.53	0	2015-01-04
2194	312	920	920	920	920	2.27	0.25	0	2015-01-05
2193	311	917	917	917	917	-2.36	-0.26	0	2015-01-06
2192	310	925	925	925	925	7.13	0.78	0	2015-01-07

Next steps: Generate code with data View recommended plots New interactive sheet

```
# Remove commas from the numerical columns and convert them to float
data['Open'] = data['Open'].str.replace(',', '').astype(float)
data['High'] = data['High'].str.replace(',', '').astype(float)
data['Low'] = data['Low'].str.replace(',', '').astype(float)
data['Close'] = data['Close'].str.replace(',', '').astype(float)
data['Turnover'] = data['Turnover'].str.replace(',', '').astype(float)

# Create moving averages
data['MA7'] = data['Close'].rolling(window=7).mean()
data['MA30'] = data['Close'].rolling(window=30).mean()

# Create lag features
data['Lag1'] = data['Close'].shift(1)
data['Lag2'] = data['Close'].shift(2)
data['Lag3'] = data['Close'].shift(3)

# Extract date features
data['Day'] = data['Date'].dt.day
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
data['DayOfWeek'] = data['Date'].dt.dayofweek

# Drop rows with NaN values that were created by rolling window and lagging
data = data.dropna()

# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
test_size = len(data) - train_size
train, test = data.iloc[:train_size], data.iloc[train_size:]

# Select the features for modeling
features = ['Close', 'MA7', 'MA30', 'Lag1', 'Lag2', 'Lag3']
train_X = train[features].values
train_y = train['Close'].values
test_X = test[features].values
test_y = test['Close'].values

# Scale the features
scaler_X = MinMaxScaler(feature_range=(0, 1))
train_X_scaled = scaler_X.fit_transform(train_X)
test_X_scaled = scaler_X.transform(test_X)

# Scale the target (Close price)
scaler_y = MinMaxScaler(feature_range=(0, 1))
train_y_scaled = scaler_y.fit_transform(train_y.reshape(-1, 1))
test_y_scaled = scaler_y.transform(test_y.reshape(-1, 1))

# Reshape the data to be compatible with LSTM [samples, time steps, features]
train_X_scaled = train_X_scaled.reshape((train_X_scaled.shape[0], 1, train_X_scaled.shape[1
test_X_scaled = test_X_scaled.reshape((test_X_scaled.shape[0], 1, test_X_scaled.shape[1]))
```

```
# Build the GRU model
model = Sequential()
model.add(GRU(50, return_sequences=True, input_shape=(1, len(features))))
model.add(GRU(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history=model.fit(train_X_scaled, train_y_scaled, batch_size=1, epochs=10)

# Make predictions
train_predict_scaled = model.predict(train_X_scaled)
test_predict_scaled = model.predict(test_X_scaled)

# Inverse transform the predictions
train_predict = scaler_y.inverse_transform(train_predict_scaled)
test_predict = scaler_y.inverse_transform(test_predict_scaled)

# Calculate RMSE
train_rmse = np.sqrt(mean_squared_error(train_y, train_predict))
test_rmse = np.sqrt(mean_squared_error(test_y, test_predict))

print(f'Train RMSE: {train_rmse}')
print(f'Test RMSE: {test_rmse}')
```

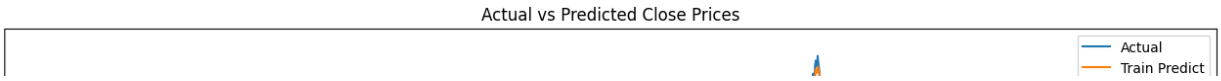
```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning:
  super().__init__(**kwargs)
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 0.0040
Epoch 2/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 4.5809e-04
Epoch 3/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 3.4407e-04
Epoch 4/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - loss: 2.0472e-04
Epoch 5/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 2.2943e-04
Epoch 6/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 1.9946e-04
Epoch 7/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - loss: 2.3010e-04
Epoch 8/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 1.2656e-04
Epoch 9/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 1.1796e-04
Epoch 10/10
1734/1734 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - loss: 1.3180e-04
55/55 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Train RMSE: 15.8160186990448
Test RMSE: 12.293165870837914
```

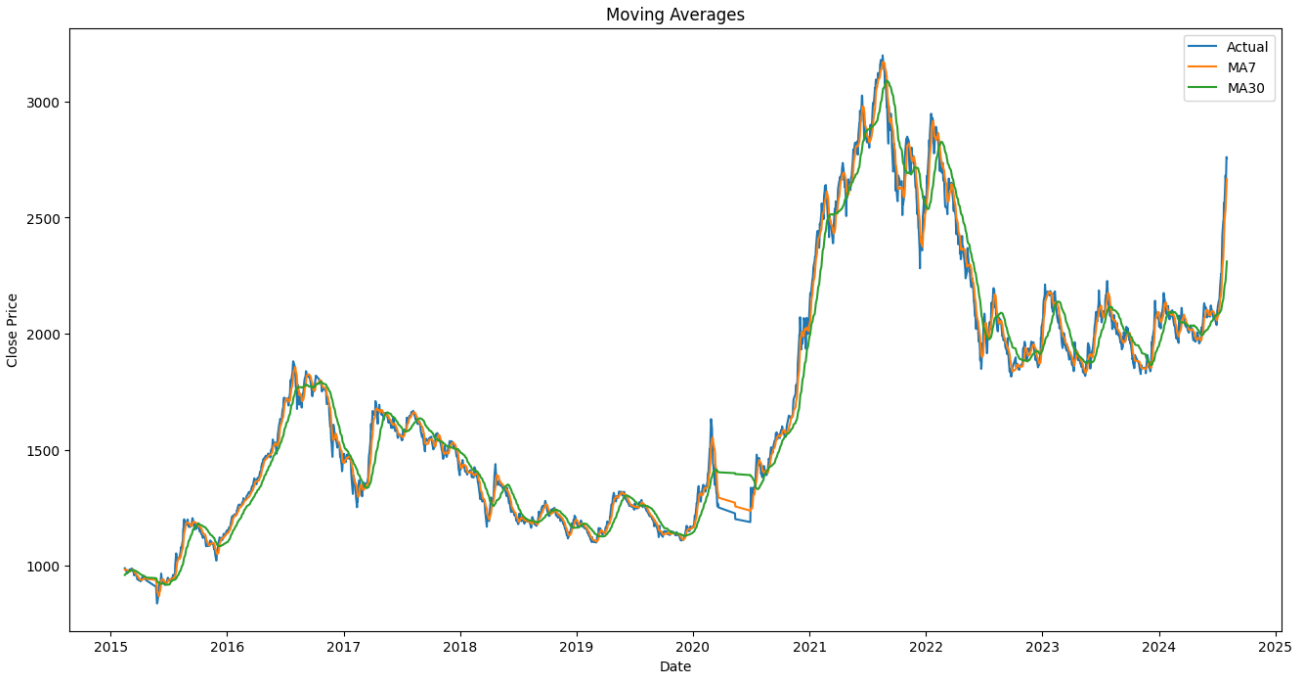
```
# Calculate MSE
train_mse = mean_squared_error(train_y, train_predict)
test_mse = mean_squared_error(test_y, test_predict)
print(f'Train MSE: {train_mse}')
print(f'Test MSE: {test_mse}')
```

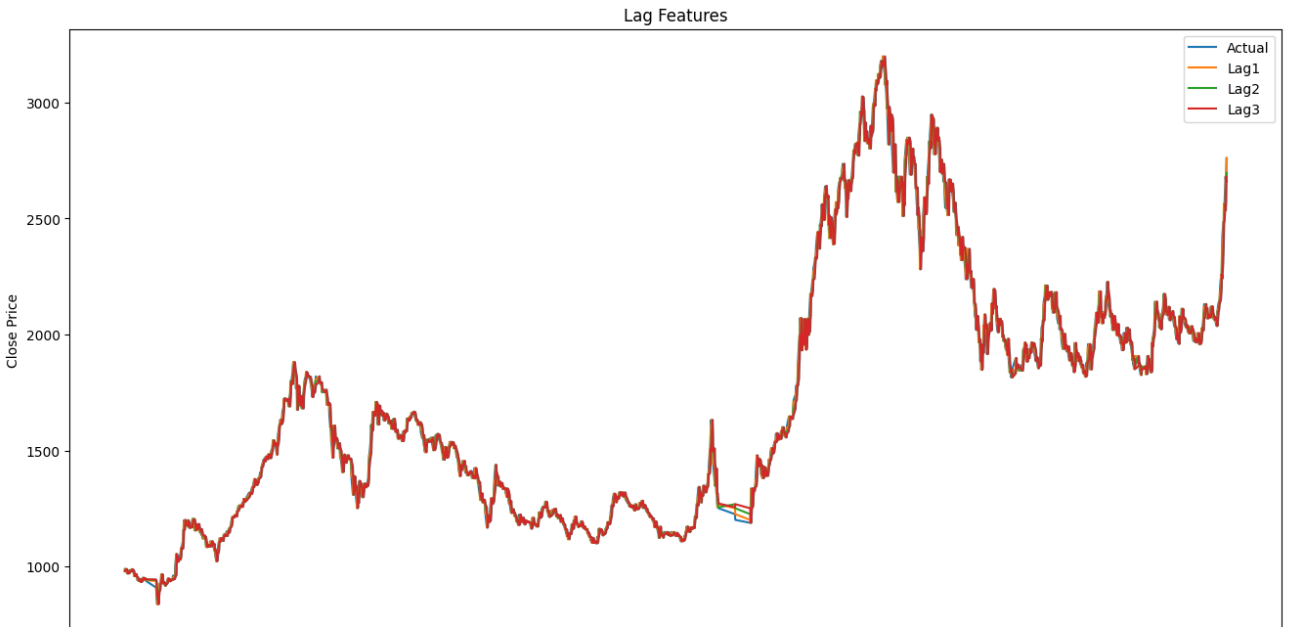
```
Train MSE: 250.1464474885348
Test MSE: 151.1219271279341
```

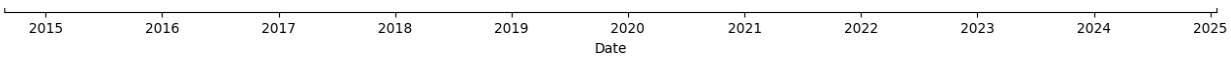
```
# Plot the results: Actual vs Predicted Close Prices
plt.figure(figsize=(16,8))
plt.plot(data['Date'], data['Close'], label='Actual')
plt.plot(data['Date'][:train_size], train_predict, label='Train Predict')
plt.plot(data['Date'][train_size:], test_predict, label='Test Predict')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.title('Actual vs Predicted Close Prices')
plt.show()

# Plot the Moving Averages
plt.figure(figsize=(16,8))
plt.plot(data['Date'], data['Close'], label='Actual')
plt.plot(data['Date'], data['MA7'], label='MA7')
plt.plot(data['Date'], data['MA30'], label='MA30')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.title('Moving Averages')
plt.show()

# Plot the Lag Features
plt.figure(figsize=(16,8))
plt.plot(data['Date'], data['Close'], label='Actual')
plt.plot(data['Date'], data['Lag1'], label='Lag1')
plt.plot(data['Date'], data['Lag2'], label='Lag2')
plt.plot(data['Date'], data['Lag3'], label='Lag3')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.title('Lag Features')
plt.show()
```




Test Predict
Close Price
Date
3000
2500
2000
1500
1000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025


Moving Averages
Actual
MA7
MA30
Close Price
Date
3000
2500
2000
1500
1000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025


Lag Features
Actual
Lag1
Lag2
Lag3
Close Price
3000
2500
2000
1500
1000



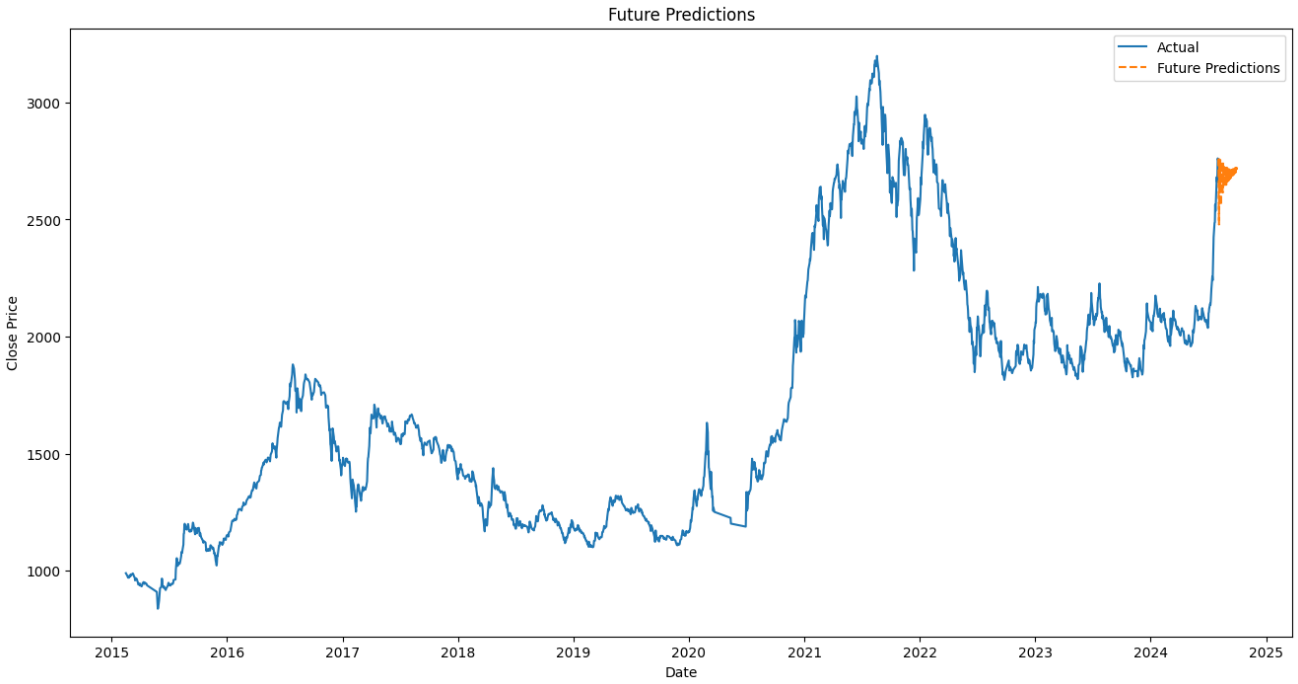

```
# Prepare the last available data point for future prediction
last_data_point = data[features].iloc[-1].values.reshape(1, -1)
scaled_last_data_point = scaler_X.transform(last_data_point)

# Initialize lists to store future predictions
future_predictions = []
num_future_days = 60  # Number of future days to predict

# Predict future values
for _ in range(num_future_days):
    scaled_last_data_point = scaled_last_data_point.reshape((1, 1, len(features)))
    future_pred_scaled = model.predict(scaled_last_data_point)
    future_pred = scaler_y.inverse_transform(future_pred_scaled)

    # Append the prediction to the future_predictions list
    future_predictions.append(future_pred[0][0])

    # Update the scaled_last_data_point with the new predicted value
    new_data_point = np.append(scaled_last_data_point.flatten()[1:], future_pred_scaled)
    scaled_last_data_point = new_data_point.reshape(1, -1)

# Create a DataFrame for future predictions
future_dates = pd.date_range(data['Date'].iloc[-1] + pd.Timedelta(days=1), periods=num_fu
future_df = pd.DataFrame({'Date': future_dates, 'Predicted Close': future_predictions})
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

```
# Plot the future predictions
plt.figure(figsize=(16,8))
plt.plot(data['Date'], data['Close'], label='Actual')
plt.plot(future_df['Date'], future_df['Predicted Close'], label='Future Predictions', lines
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.title('Future Predictions')
plt.show()

# Create a DataFrame for future predictions
future_dates = pd.date_range(data['Date'].iloc[-1] + pd.Timedelta(days=1), periods=num_futu
print(pd.DataFrame({'Date': future_dates, 'Predicted Close': future_predictions}))

# Plot the actual data and future predictions
plt.figure(figsize=(16, 8))
plt.plot(data['Date'], data['Close'], label='Actual')
plt.plot(future_df['Date'], future_df['Predicted Close'], label='Future Predictions', lines
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.title('Future Predictions')
plt.savefig('future_predictions.png')  # Save the plot
plt.show(block=True)  # Prevent the plot from disappearing immediately
```




```
        Date  Predicted Close
0 2024-08-02      2760.116699
1 2024-08-03      2647.025635
```

2	2024-08-04	2478.880615
3	2024-08-05	2757.185303
4	2024-08-06	2673.291992
5	2024-08-07	2647.793701
6	2024-08-08	2754.145264
7	2024-08-09	2634.630371
8	2024-08-10	2560.777100
9	2024-08-11	2749.364746
10	2024-08-12	2658.825928
11	2024-08-13	2654.667236
12	2024-08-14	2744.266602
13	2024-08-15	2633.749756
14	2024-08-16	2615.051025
15	2024-08-17	2739.619629
16	2024-08-18	2653.319336
17	2024-08-19	2668.237061
18	2024-08-20	2733.857666
19	2024-08-21	2639.386963
20	2024-08-22	2651.628662
21	2024-08-23	2729.968018
22	2024-08-24	2654.742676
23	2024-08-25	2682.317383
24	2024-08-26	2724.678223
25	2024-08-27	2648.618652
26	2024-08-28	2676.430176
27	2024-08-29	2721.751221
28	2024-08-30	2660.748779
29	2024-08-31	2694.437256
30	2024-09-01	2717.564941
31	2024-09-02	2659.567627
32	2024-09-03	2693.226807
33	2024-09-04	2715.649170
34	2024-09-05	2669.325439
35	2024-09-06	2703.956055
36	2024-09-07	2712.789795
37	2024-09-08	2670.983398
38	2024-09-09	2704.566895
39	2024-09-10	2711.843750
40	2024-09-11	2678.966553
41	2024-09-12	2711.067383
42	2024-09-13	2710.274414
43	2024-09-14	2682.050781
44	2024-09-15	2712.239014
45	2024-09-16	2710.193115
46	2024-09-17	2688.654785
47	2024-09-18	2716.277100
48	2024-09-19	2709.738281
49	2024-09-20	2692.280518
50	2024-09-21	2717.518066
51	2024-09-22	2710.376465
52	2024-09-23	2697.772217
53	2024-09-24	2720.140381
54	2024-09-25	2710.807129
55	2024-09-26	2701.425537
56	2024-09-27	2721.310303

```
56 2024-09-27      2721.516565
57 2024-09-28      2712.001221
58 2024-09-29      2705.999512
59 2024-09-30      2723.145996
```