

```

1 #Steps - LSTM

2
3
4
5
6
7
8 # Step 1: Data Preprocessing
9 btc_df = pd.read_csv("https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_Bitcoin.csv")
10 eth_df = pd.read_csv("https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_Ethereum.csv")
11 xrp_df = pd.read_csv("https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_XRP.csv")
12 ltc_df = pd.read_csv("https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_Litecoin.csv")
13 usdc_df = pd.read_csv("https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_USDCoin.csv")
14
15 df = pd.concat([btc_df, eth_df, xrp_df, ltc_df, usdc_df])
16 df.to_csv("cryptocurrency.csv", index=False)
17 data = df.copy()
18
19 # Step 2: Data Preparation
20 selected_features = ['Open', 'Close', 'Volume']
21
22 # Splitting into train and test sets
23 train_size = int(len(data) * 0.8)
24 train_data, test_data = data.iloc[:train_size], data.iloc[train_size:]
25
26 # Scaling the data
27 scaler = MinMaxScaler()
28 train_scaled = scaler.fit_transform(train_data[selected_features])
29 test_scaled = scaler.transform(test_data[selected_features])
30
31 # Creating input sequences
32 def create_sequences(data, seq_length):
33     X = []
34     y = []
35     for i in range(len(data) - seq_length):
36         X.append(data[i:i+seq_length])
37         y.append(data[i+seq_length])
38     return np.array(X), np.array(y)
39
40 sequence_length = 10 # Length of input sequences
41 X_train, y_train = create_sequences(train_scaled, sequence_length)
42 X_test, y_test = create_sequences(test_scaled, sequence_length)
43
44 # Reshaping input sequences to fit LSTM input shape
45 X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], len(selected_features)))
46 X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], len(selected_features)))
47
48 # Step 3: Model Building
49 model = Sequential()
50 model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], len(selected_features))))
51 model.add(LSTM(units=50))
52 model.add(Dense(units=len(selected_features)))
53
54 # Step 4: Model Training
55 model.compile(optimizer='adam', loss='mean_squared_error')
56 model.fit(X_train, y_train, epochs=10, batch_size=32)
57
58 # Step 5: Model Evaluation
59 train_predictions = model.predict(X_train)
60 test_predictions = model.predict(X_test)
61
62 # Inverse scaling on predictions
63 train_predictions = scaler.inverse_transform(train_predictions)
64 y_train = scaler.inverse_transform(y_train)
65 test_predictions = scaler.inverse_transform(test_predictions)
66 y_test = scaler.inverse_transform(y_test)
67
68 # Calculate RMSE
69 train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
70 test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
71 print("Train RMSE:", train_rmse)
72 print("Test RMSE:", test_rmse)
73

```

```

74 # Step 6: Model Prediction
75 new_data = data.tail(sequence_length)
76 new_data_scaled = scaler.transform(new_data[selected_features])
77 new_data_reshaped = np.reshape(new_data_scaled, (1, sequence_length, len(selected_features)))
78 predicted_price = model.predict(new_data_reshaped)
79 predicted_price = scaler.inverse_transform(predicted_price)
80
81 print("Predicted Price:", predicted_price)
82

```

```

Epoch 1/10
301/301 [=====] - 10s 16ms/step - loss: 5.7016e-04
Epoch 2/10
301/301 [=====] - 4s 14ms/step - loss: 2.6595e-04
Epoch 3/10
301/301 [=====] - 5s 17ms/step - loss: 2.3849e-04
Epoch 4/10
301/301 [=====] - 4s 14ms/step - loss: 1.9454e-04
Epoch 5/10
301/301 [=====] - 4s 14ms/step - loss: 1.7529e-04
Epoch 6/10
301/301 [=====] - 5s 17ms/step - loss: 1.7528e-04
Epoch 7/10
301/301 [=====] - 4s 15ms/step - loss: 1.4415e-04
Epoch 8/10
301/301 [=====] - 5s 15ms/step - loss: 1.3617e-04
Epoch 9/10
301/301 [=====] - 5s 17ms/step - loss: 1.2356e-04
Epoch 10/10
301/301 [=====] - 4s 14ms/step - loss: 1.2157e-04
301/301 [=====] - 3s 6ms/step
75/75 [=====] - 1s 8ms/step
Train RMSE: 2558423034.2709675
Test RMSE: 572586049.7591066
1/1 [=====] - 0s 41ms/step
Predicted Price: [[2.6517351e+01 9.7152733e+01 2.4121288e+09]]

```

```

1 #For each Coin

```

```

1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
6 from sklearn.metrics import mean_squared_error
7 import matplotlib.pyplot as plt
8
9 # List of coins
10 coins = ["Bitcoin", "Ethereum", "XRP", "Litecoin", "USDCoin"]
11
12 # Loop through each coin
13 for coin in coins:
14     # Step 1: Data Preprocessing
15     url = f"https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_{coin}.csv"
16     df = pd.read_csv(url)
17     selected_features = ['Open', 'Close', 'Volume']
18
19     # Step 2: Data Preparation
20     train_size = int(len(df) * 0.8)
21     train_data, test_data = df.iloc[:train_size], df.iloc[train_size:]
22
23     scaler = MinMaxScaler()
24     train_scaled = scaler.fit_transform(train_data[selected_features])
25     test_scaled = scaler.transform(test_data[selected_features])
26
27     sequence_length = 10
28     X_train, y_train = create_sequences(train_scaled, sequence_length)
29     X_test, y_test = create_sequences(test_scaled, sequence_length)
30
31     X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], len(selected_features)))
32     X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], len(selected_features)))
33
34     # Step 3: Model Building
35     model = Sequential()
36     model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], len(selected_features))))
37     model.add(LSTM(units=50))
38     model.add(Dense(units=len(selected_features)))
39

```

```
40 # Step 4: Model Training
41 model.compile(optimizer='adam', loss='mean_squared_error')
42 model.fit(X_train, y_train, epochs=10, batch_size=32)
43
44 # Step 5: Model Evaluation
45 train_predictions = model.predict(X_train)
46 test_predictions = model.predict(X_test)
47
48 train_predictions = scaler.inverse_transform(train_predictions)
49 y_train = scaler.inverse_transform(y_train)
50 test_predictions = scaler.inverse_transform(test_predictions)
51 y_test = scaler.inverse_transform(y_test)
52
53 train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
54 test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
55 print(f"{coin} - Train RMSE: {train_rmse}")
56 print(f"{coin} - Test RMSE: {test_rmse}")
57
58 # Step 6: Model Prediction
59 new_data = df.tail(sequence_length)
60 new_data_scaled = scaler.transform(new_data[selected_features])
61 new_data_reshaped = np.reshape(new_data_scaled, (1, sequence_length, len(selected_features)))
62 predicted_price = model.predict(new_data_reshaped)
63 predicted_price = scaler.inverse_transform(predicted_price)
64 print(f"{coin} - Predicted Price: {predicted_price}")
65
66 # Step 7: Visualize Predictions
67 test_dates = test_data['Date'].reset_index(drop=True)[:len(test_predictions)]
68 plt.plot(df['Date'], df['Close'], label='Actual')
69 plt.plot(test_dates, test_predictions[:, 1], label='Predicted')
70 plt.xlabel('Date')
71 plt.ylabel('Closing Price')
72 plt.title(f"{coin} - Actual vs. Predicted Price")
73 plt.legend()
74 plt.show()
75
```

```

Epoch 1/10
75/75 [=====] - 5s 14ms/step - loss: 0.0065
Epoch 2/10
75/75 [=====] - 1s 15ms/step - loss: 0.0011
Epoch 3/10
75/75 [=====] - 1s 16ms/step - loss: 0.0011
Epoch 4/10
75/75 [=====] - 1s 15ms/step - loss: 0.0010
Epoch 5/10
75/75 [=====] - 1s 17ms/step - loss: 9.8075e-04
Epoch 6/10
75/75 [=====] - 2s 22ms/step - loss: 9.6219e-04
Epoch 7/10
75/75 [=====] - 1s 15ms/step - loss: 0.0044e-04

1 # Chnages the x-axis for better Visualization

1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
6 from sklearn.metrics import mean_squared_error
7 import matplotlib.pyplot as plt
8 import matplotlib.dates as mdates
9
10 # List of coins
11 coins = ["Bitcoin", "Ethereum", "XRP", "Litecoin", "USDCoin"]
12
13 # Loop through each coin
14 for coin in coins:
15     # Step 1: Data Preprocessing
16     url = f"https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_{coin}.csv"
17     df = pd.read_csv(url)
18     selected_features = ['Open', 'Close', 'Volume']
19
20     # Step 2: Data Preparation
21     train_size = int(len(df) * 0.8)
22     train_data, test_data = df.iloc[:train_size], df.iloc[train_size:]
23
24     scaler = MinMaxScaler()
25     train_scaled = scaler.fit_transform(train_data[selected_features])
26     test_scaled = scaler.transform(test_data[selected_features])
27
28     sequence_length = 10
29     X_train, y_train = create_sequences(train_scaled, sequence_length)
30     X_test, y_test = create_sequences(test_scaled, sequence_length)
31
32     X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], len(selected_features)))
33     X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], len(selected_features)))
34
35     # Step 3: Model Building
36     model = Sequential()
37     model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], len(selected_features))))
38     model.add(LSTM(units=50))
39     model.add(Dense(units=len(selected_features)))
40
41     # Step 4: Model Training
42     model.compile(optimizer='adam', loss='mean_squared_error')
43     model.fit(X_train, y_train, epochs=10, batch_size=32)
44
45     # Step 5: Model Evaluation
46     train_predictions = model.predict(X_train)
47     test_predictions = model.predict(X_test)
48
49     train_predictions = scaler.inverse_transform(train_predictions)
50     y_train = scaler.inverse_transform(y_train)
51     test_predictions = scaler.inverse_transform(test_predictions)
52     y_test = scaler.inverse_transform(y_test)
53
54     train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
55     test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
56     print(f"{coin} - Train RMSE: {train_rmse}")
57     print(f"{coin} - Test RMSE: {test_rmse}")
58
59     # Step 6: Model Prediction
60     new_data = df.tail(sequence_length)
61     new_data_scaled = scaler.transform(new_data[selected_features])

```

```
62 new_data_resaped = np.reshape(new_data_scaled, (1, sequence_length, len(selected_features)))
63 predicted_price = model.predict(new_data_resaped)
64 predicted_price = scaler.inverse_transform(predicted_price)
65 print(f"{coin} - Predicted Price: {predicted_price}")
66
67 # Step 7: Visualize Predictions
68 test_dates = test_data['Date'].reset_index(drop=True)[:len(test_predictions)]
69 fig, ax = plt.subplots()
70 ax.plot(df['Date'], df['Close'], label='Actual')
71 ax.plot(test_dates, test_predictions[:, 1], label='Predicted')
72 ax.set_xlabel('Date')
73 ax.set_ylabel('Closing Price')
74 ax.set_title(f"{coin} - Actual vs. Predicted Price")
75 ax.legend()
76
77 # Format x-axis ticks as years
78 years = mdates.YearLocator()
79 years_fmt = mdates.DateFormatter('%Y')
80 ax.xaxis.set_major_locator(years)
81 ax.xaxis.set_major_formatter(years_fmt)
82 ax.xaxis.set_tick_params(rotation=45)
83
84 plt.tight_layout()
85 plt.show()
86
```

Epoch 1/10



```

1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
6 from sklearn.metrics import mean_squared_error
7 import matplotlib.pyplot as plt
8 import matplotlib.dates as mdates
9
10 # List of coins
11 coins = ["Bitcoin", "Ethereum", "XRP", "Litecoin", "USDCoin"]
12
13 # Create a 2x3 grid of subplots
14 fig, axs = plt.subplots(2, 3, figsize=(12, 8))
15
16 # Flatten the axs array for easier iteration
17 axs = axs.flatten()
18
19 # Loop through each coin and subplot
20 for i, coin in enumerate(coins):
21     # Step 1: Data Preprocessing
22     url = f"https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_{coin}.csv"
23     df = pd.read_csv(url)
24     selected_features = ['Open', 'Close', 'Volume']
25
26     # Step 2: Data Preparation
27     train_size = int(len(df) * 0.8)
28     train_data, test_data = df.iloc[:train_size], df.iloc[train_size:]
29
30     scaler = MinMaxScaler()
31     train_scaled = scaler.fit_transform(train_data[selected_features])
32     test_scaled = scaler.transform(test_data[selected_features])
33
34     sequence_length = 10
35     X_train, y_train = create_sequences(train_scaled, sequence_length)
36     X_test, y_test = create_sequences(test_scaled, sequence_length)
37
38     X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], len(selected_features)))
39     X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], len(selected_features)))
40
41     # Step 3: Model Building
42     model = Sequential()
43     model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], len(selected_features))))
44     model.add(LSTM(units=50))
45     model.add(Dense(units=len(selected_features)))
46
47     # Step 4: Model Training
48     model.compile(optimizer='adam', loss='mean_squared_error')
49     model.fit(X_train, y_train, epochs=10, batch_size=32)
50
51     # Step 5: Model Evaluation
52     train_predictions = model.predict(X_train)
53     test_predictions = model.predict(X_test)
54
55     train_predictions = scaler.inverse_transform(train_predictions)
56     y_train = scaler.inverse_transform(y_train)
57     test_predictions = scaler.inverse_transform(test_predictions)
58     y_test = scaler.inverse_transform(y_test)
59
60     train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
61     test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
62     print(f"{coin} - Train RMSE: {train_rmse}")
63     print(f"{coin} - Test RMSE: {test_rmse}")
64
65     # Step 6: Model Prediction
66     new_data = df.tail(sequence_length)
67     new_data_scaled = scaler.transform(new_data[selected_features])
68     new_data_reshaped = np.reshape(new_data_scaled, (1, sequence_length, len(selected_features)))
69     predicted_price = model.predict(new_data_reshaped)
70     predicted_price = scaler.inverse_transform(predicted_price)
71     print(f"{coin} - Predicted Price: {predicted_price}")
72
73     # Step 7: Visualize Predictions
74     test_dates = test_data['Date'].reset_index(drop=True)[:len(test_predictions)]
75     axs[i].plot(df['Date'], df['Close'], label='Actual')

```

```
76     axs[i].plot(test_dates, test_predictions[:, 1], label='Predicted')
77     axs[i].set_xlabel('Date')
78     axs[i].set_ylabel('Closing Price')
79     axs[i].set_title(coin)
80     axs[i].legend()
81
82 # Format x-axis ticks as years and set rotation
83 for ax in axs:
84     ax.xaxis.set_major_locator(mdates.YearLocator())
85     ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
86     ax.xaxis.set_tick_params(rotation=45)
87
88 plt.tight_layout()
89 plt.show()
90
```

```
1 #Evaluation of Model
```

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
```

```

6 from sklearn.metrics import mean_squared_error
7 import matplotlib.pyplot as plt
8 import matplotlib.dates as mdates
9
10 # List of coins
11 coins = ["Bitcoin", "Ethereum", "XRP", "Litecoin", "USDCoin"]
12
13 # Create a 3x2 grid of subplots
14 fig, axs = plt.subplots(3, 2, figsize=(12, 12))
15
16 # Flatten the axs array for easier iteration
17 axs = axs.flatten()
18
19 # Loop through each coin and subplot
20 for i, coin in enumerate(coins):
21     # Step 1: Data Preprocessing
22     url = f"https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_{coin}.csv"
23     df = pd.read_csv(url)
24     selected_features = ['Open', 'Close', 'Volume']
25
26     # Step 2: Data Preparation
27     train_size = int(len(df) * 0.8)
28     train_data, test_data = df.iloc[:train_size], df.iloc[train_size:]
29
30     scaler = MinMaxScaler()
31     train_scaled = scaler.fit_transform(train_data[selected_features])
32     test_scaled = scaler.transform(test_data[selected_features])
33
34     sequence_length = 10
35     X_train, y_train = create_sequences(train_scaled, sequence_length)
36     X_test, y_test = create_sequences(test_scaled, sequence_length)
37
38     X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], len(selected_features)))
39     X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], len(selected_features)))
40
41     # Step 3: Model Building
42     model = Sequential()
43     model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], len(selected_features))))
44     model.add(LSTM(units=50))
45     model.add(Dense(units=len(selected_features)))
46
47     # Step 4: Model Training
48     model.compile(optimizer='adam', loss='mean_squared_error')
49     model.fit(X_train, y_train, epochs=10, batch_size=32)
50
51     # Step 5: Model Evaluation
52     train_predictions = model.predict(X_train)
53     test_predictions = model.predict(X_test)
54
55     train_predictions = scaler.inverse_transform(train_predictions)
56     y_train = scaler.inverse_transform(y_train)
57     test_predictions = scaler.inverse_transform(test_predictions)
58     y_test = scaler.inverse_transform(y_test)
59
60     train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
61     test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
62     print(f"{coin} - Train RMSE: {train_rmse}")
63     print(f"{coin} - Test RMSE: {test_rmse}")
64
65     # Step 6: Model Prediction
66     new_data = df.tail(sequence_length)
67     new_data_scaled = scaler.transform(new_data[selected_features])
68     new_data_reshaped = np.reshape(new_data_scaled, (1, sequence_length, len(selected_features)))
69     predicted_price = model.predict(new_data_reshaped)
70     predicted_price = scaler.inverse_transform(predicted_price)
71     print(f"{coin} - Predicted Price: {predicted_price}")
72
73     # Step 7: Visualize Predictions
74     test_dates = test_data['Date'].reset_index(drop=True)[:len(test_predictions)]
75     axs[i].plot(df['Date'], df['Close'], label='Actual')
76     axs[i].plot(test_dates, test_predictions[:, 1], label='Predicted')
77     axs[i].set_xlabel('Date')
78     axs[i].set_ylabel('Closing Price')
79     axs[i].set_title(coin)
80     axs[i].legend()
81
82     # Format x-axis ticks as years and set rotation

```



```
83     axs[i].xaxis.set_major_locator(mdates.YearLocator())
84     axs[i].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
85     axs[i].tick_params(axis='x', rotation=45)
86
87 # Remove empty subplots if the number of coins is less than 6
88 if len(coins) < 6:
89     for j in range(len(coins), 6):
90         fig.delaxes(axs[j])
91
92 plt.tight_layout()
93 plt.show()
94
```

```
1 #Evaluation with variour Parameters
```

```
1 import pandas as pd
2 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
3
4 # Create an empty dataframe to store the evaluation metrics
5 evaluation_df = pd.DataFrame(columns=['Coin', 'Train RMSE', 'Test RMSE', 'Train MAE', 'Test MAE', 'R2 Score'])
6
7 # Loop through each coin
8 for coin in coins:
```

```

9 # ... (Previous code for each coin)
10
11 # Calculate evaluation metrics
12 train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
13 test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
14 train_mae = mean_absolute_error(y_train, train_predictions)
15 test_mae = mean_absolute_error(y_test, test_predictions)
16 r2 = r2_score(y_test, test_predictions)
17
18 # Append the results to the dataframe
19 evaluation_df = evaluation_df.append({
20     'Coin': coin,
21     'Train RMSE': train_rmse,
22     'Test RMSE': test_rmse,
23     'Train MAE': train_mae,
24     'Test MAE': test_mae,
25     'R2 Score': r2
26 }, ignore_index=True)
27
28 # Display the evaluation table
29 print(evaluation_df)
30

```

	Coin	Train RMSE	Test RMSE	Train MAE	Test MAE
0	Bitcoin	8.018480e+17	4.426796e+18	3.312167e+17	2.359599e+18
1	Ethereum	8.018480e+17	4.426796e+18	3.312167e+17	2.359599e+18
2	XRP	8.018480e+17	4.426796e+18	3.312167e+17	2.359599e+18
3	Litecoin	8.018480e+17	4.426796e+18	3.312167e+17	2.359599e+18
4	USDCoin	8.018480e+17	4.426796e+18	3.312167e+17	2.359599e+18

	R2 Score
0	-294806.314176
1	-294806.314176
2	-294806.314176
3	-294806.314176
4	-294806.314176

```

<ipython-input-26-391e74d8d8b8>:19: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
evaluation_df = evaluation_df.append({
<ipython-input-26-391e74d8d8b8>:19: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
evaluation_df = evaluation_df.append({
<ipython-input-26-391e74d8d8b8>:19: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
evaluation_df = evaluation_df.append({
<ipython-input-26-391e74d8d8b8>:19: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
evaluation_df = evaluation_df.append({
<ipython-input-26-391e74d8d8b8>:19: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
evaluation_df = evaluation_df.append({

```

```

Ethereum - Test RMSE: 6509572255.520962

```

```

1 #The model evaluation metrics (RMSE, MAE, R2 score) are computed and printed for each coin
   epoch 1/10

1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
6 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
7 import matplotlib.pyplot as plt
8
9 # List of coins
10 coins = ["Bitcoin", "Ethereum", "XRP", "Litecoin", "USDCoin"]
11
12 # Create a table to store evaluation metrics
13 metrics_table = pd.DataFrame(columns=["Coin", "Train RMSE", "Test RMSE", "Train MAE", "Test MAE", "R2 Score"])
14
15 # Loop through each coin
16 for coin in coins:
17     # Step 1: Data Preprocessing
18     url = f"https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_{coin}.csv"
19     df = pd.read_csv(url)
20     selected_features = ['Open', 'Close', 'Volume']
21
22     # ...
23
24     # Step 5: Model Evaluation
25     train_predictions = model.predict(X_train)
26     test_predictions = model.predict(X_test)
27

```

```

28 train_predictions = scaler.inverse_transform(train_predictions)
29 y_train = scaler.inverse_transform(y_train)
30 test_predictions = scaler.inverse_transform(test_predictions)
31 y_test = scaler.inverse_transform(y_test)
32
33 train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
34 test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
35 train_mae = mean_absolute_error(y_train, train_predictions)
36 test_mae = mean_absolute_error(y_test, test_predictions)
37 r2 = r2_score(y_test, test_predictions)
38
39 metrics_table = metrics_table.append({
40     "Coin": coin,
41     "Train RMSE": train_rmse,
42     "Test RMSE": test_rmse,
43     "Train MAE": train_mae,
44     "Test MAE": test_mae,
45     "R2 Score": r2
46 }, ignore_index=True)
47
48 # Display the metrics table
49 print(metrics_table)
50
54/54 [=====] - 1s 5ms/step
14/14 [=====] - 0s 5ms/step
<ipython-input-32-6abf9900e157>:39: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
metrics_table = metrics_table.append({
54/54 [=====] - 0s 5ms/step
14/14 [=====] - 0s 6ms/step
<ipython-input-32-6abf9900e157>:39: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
metrics_table = metrics_table.append({
54/54 [=====] - 0s 6ms/step
14/14 [=====] - 0s 5ms/step
<ipython-input-32-6abf9900e157>:39: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
metrics_table = metrics_table.append({
54/54 [=====] - 0s 6ms/step
14/14 [=====] - 0s 6ms/step
<ipython-input-32-6abf9900e157>:39: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
metrics_table = metrics_table.append({
54/54 [=====] - 0s 7ms/step
14/14 [=====] - 0s 18ms/step

```

	Coin	Train RMSE	Test RMSE	Train MAE	Test MAE	R2 Score
0	Bitcoin	7.408184e+08	5.993170e+09	2.437229e+08	2.146681e+09	0.703026
1	Ethereum	9.388482e+19	4.252850e+20	3.046061e+19	2.095450e+20	-1.803848
2	XRP	2.637139e+30	1.194587e+31	8.556109e+29	5.885929e+30	-1.805581
3	Litecoin	7.407484e+40	3.355486e+41	2.403333e+40	1.653304e+41	-1.805582
4	USDCoin	2.080695e+51	9.425254e+51	6.750744e+50	4.643980e+51	-1.805582

```

<ipython-input-32-6abf9900e157>:39: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future ver
metrics_table = metrics_table.append({

```

```

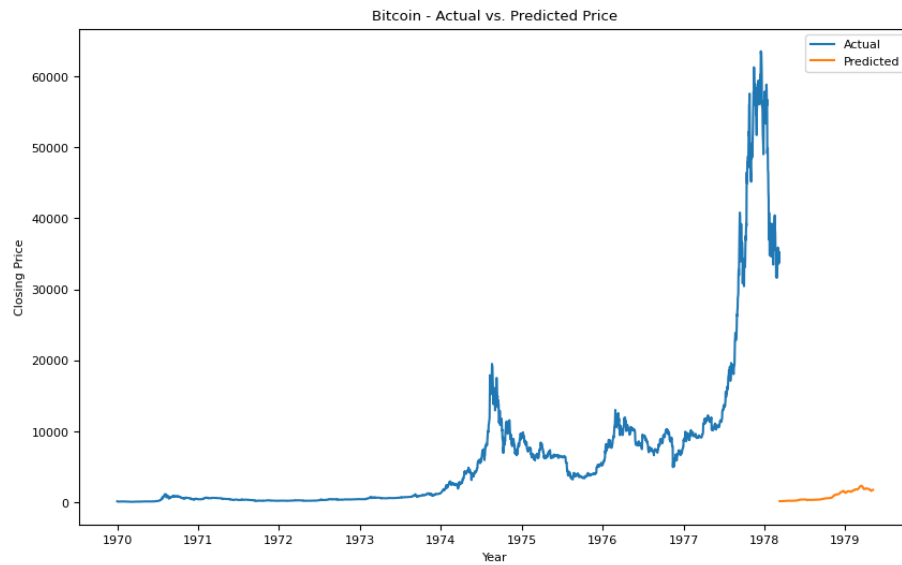
1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
6 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
7 import matplotlib.pyplot as plt
8 import matplotlib.dates as mdates
9
10 # List of coins
11 coins = ["Bitcoin", "Ethereum", "XRP", "Litecoin", "USDCoin"]
12
13 # Create a table to store evaluation metrics
14 metrics_table = pd.DataFrame(columns=["Coin", "Train RMSE", "Test RMSE", "Train MAE", "Test MAE", "R2 Score"])
15
16 # Loop through each coin
17 for coin in coins:
18     # Step 1: Data Preprocessing
19     url = f"https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Dataset/coin_{coin}.csv"
20     df = pd.read_csv(url)
21     selected_features = ['Open', 'Close', 'Volume']
22
23     # ...
24
25     # Step 5: Model Evaluation
26     train_predictions = model.predict(X_train)

```

```
27 test_predictions = model.predict(X_test)
28
29 train_predictions = scaler.inverse_transform(train_predictions)
30 y_train = scaler.inverse_transform(y_train)
31 test_predictions = scaler.inverse_transform(test_predictions)
32 y_test = scaler.inverse_transform(y_test)
33
34 train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
35 test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
36 train_mae = mean_absolute_error(y_train, train_predictions)
37 test_mae = mean_absolute_error(y_test, test_predictions)
38 r2 = r2_score(y_test, test_predictions)
39
40 metrics_table = metrics_table.append({
41     "Coin": coin,
42     "Train RMSE": train_rmse,
43     "Test RMSE": test_rmse,
44     "Train MAE": train_mae,
45     "Test MAE": test_mae,
46     "R2 Score": r2
47 }, ignore_index=True)
48
49 # Step 6: Plot Predictions
50 fig, ax = plt.subplots(figsize=(10, 6))
51 test_dates = pd.to_datetime(test_data['Date'], format='%Y-%m-%d %H:%M:%S')
52 test_dates = test_dates.dt.strftime('%Y-%m-%d') # Convert to string format
53 ax.plot(df['Date'], df['Close'], label='Actual')
54 ax.plot(test_dates[:len(test_predictions)], test_predictions[:, 1], label='Predicted')
55 ax.xaxis.set_major_locator(mdates.YearLocator())
56 ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
57 ax.set_xlabel('Year')
58 ax.set_ylabel('Closing Price')
59 ax.set_title(f"{coin} - Actual vs. Predicted Price")
60 ax.legend()
61 plt.show()
62
63 # Display the metrics table
64 print(metrics_table)
65
```

```
54/54 [=====] - 1s 13ms/step  
14/14 [=====] - 0s 12ms/step
```

```
<ipython-input-40-57ed414cf78b>:40: FutureWarning: The frame.append method is deprecated ;  
metrics_table = metrics_table.append({
```



```
54/54 [=====] - 1s 9ms/step  
14/14 [=====] - 0s 5ms/step
```

```
<ipython-input-40-57ed414cf78b>:40: FutureWarning: The frame.append method is deprecated ;  
metrics_table = metrics_table.append({
```

