In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as web
import datetime as dt
import datetime as dt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import Dense, Dropout, LSTM
from tensorflow.keras.models import Sequential
from sklearn.metrics import r2_score
```

In [3]:

```
# The crypto currencies we will study
crypto_currency1 = 'BTC'
crypto_currency2 = 'ETH'
crypto_currency3 = 'LTC'
against_currency = 'USD'

acc = []

def create_and_fit__model(x_train, y_train):

  # Sequential models are a simple way to stack layers one after another.
  model = Sequential()
  model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape
  model.add(Dropout(0.2))

  model.add(LSTM(units=50, return_sequences=True))
  model.add(Dropout(0.2))

  model.add(LSTM(units=50))
  model.add(Dropout(0.2))

  model.add(Dense(units=1))

  model.compile(optimizer='adam', loss='mean_squared_error')
  model.fit(x_train, y_train, epochs=25, batch_size=32)

  return model

def prediction_function(crypto_currency):
    # The dataset should be read/downloaded from the 1st of Jan, 2018
    start_date = dt.datetime(2018, 1, 1)
    # The dataset should be read/downloaded till today
    end_date = dt.datetime.now()

    # Read the entire CSV file
    df = pd.read_csv('Preprocess.csv')

    # Convert 'Date' column to datetime if it's not already in datetime form
    df['d'] = pd.to_datetime(df['d'])
```

```
    # Filter DataFrame to include only data within the specified date range
    df = df[(df['d'] >= start_date) & (df['d'] <= end_date)]
    scaler = MinMaxScaler(feature_range=(0, 1))

    df_scaled = scaler.fit_transform(df[['c', 'v']].values.reshape(-1, 2))

  # Looks back on 60 days of data to predict the values of 61st day
    lookback = 60
    x_train, y_train, vol, = [], [], []

  # Filling up the x_train and y_train with the scaled data
    for i in range(lookback, len(df_scaled)):

      # Finding the consolidated Volume for the past lookback days
      com_vol = 0
      for j in range(i - lookback, i):
          com_vol += df_scaled[j, 1]

      # Re-Scaling it to the range [0, 1]
      com_vol = com_vol / 60
      vol.append(com_vol)

      # The value f Closing Price for the last 'lookback' days should be use
      x_train.append(df_scaled[i - lookback: i, 0])

      # The value of Closing price at i is the the required output/label
      y_train.append(df_scaled[i, 0])


  # Converting the data set we have created into a numpy array
    x_train = np.array(x_train)
    y_train = np.array(y_train)
    vol = np.array(vol)
    print("\n\n The number of samples in our training data = " + str(len(x_t

    # ********************** Testing Data ************************

  # Start Date of Testing data
    test_start = dt.datetime(2021, 1, 1)
    test_end = dt.datetime.now()

# Read the entire CSV file
    df = pd.read_csv('Preprocess.csv')

# Convert 'Date' column to datetime if it's not already in datetime format
    df['d'] = pd.to_datetime(df['d'])

# Filter DataFrame to include only data within the specified test date range
    df_test = df[(df['d'] >= test_start) & (df['d'] <= test_end)]

# Print the first few rows of the filtered DataFrame
    print(df_test.head())
```

```
  actual_prices = df_test['c'].values

# Creating a combined (Test + Train data set)
  df_total = pd.concat((df['c'], df_test['c']), axis=0)
# The inputs to the model for testing will be the test data set - lookback
  model_inputs = df_total[len(df_total) - len(df_test) - lookback:].values
  model_inputs = model_inputs.reshape(-1, 1)
# The test data has not been scaled, so scaling the test data to the range
  model_inputs = scaler.fit_transform(model_inputs)


  x_test, y_test = [], []

# Creating an 2D array of of our data where each data item has 'lookback'
  for i in range(lookback, len(model_inputs)):
    x_test.append(model_inputs[i - lookback: i, 0])
    y_test.append(model_inputs[i, 0])

  x_test = np.array(x_test)
  x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

  print("\n\n The number of samples in our testing data = " + str(len(x_te
  x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

# Creating the model
  model = create_and_fit__model(x_train, y_train)

# Predicting the value and inverting the scaling
  prediction_prices = model.predict(x_test)
  prediction_prices = scaler.inverse_transform(prediction_prices)
  acc.append(r2_score(actual_prices, prediction_prices))

    # Plotting the training, test and prediction data
  plt.plot(actual_prices, color='black', label='Actual Prices')
  plt.plot(prediction_prices, color='green', label='Predicted Prices')
  plt.title("{} Price Prediction".format(crypto_currency))
  plt.xlabel("Time")
  plt.ylabel("Price")
  plt.legend(loc='upper left')
  plt.show()

# Predict next day

  real_data = [model_inputs[len(model_inputs) - lookback:len(model_inputs)
  real_data = np.array(real_data)

  prediction = []
#real_data[0].shape

  for i in range(7):
    rd = np.reshape(real_data, (real_data.shape[0], real_data.shape[1], 1)
    t = model.predict(rd)
    price = scaler.inverse_transform(t)
```

```
        prediction.append(price[0][0])
        n_real_data = []
        for i in range(1, len(real_data[0])):
            n_real_data.append([real_data[0][i]])
        n_real_data.append(t[0])
        n_real_data = np.array(n_real_data)
        n_real_data = np.transpose(n_real_data)
        real_data = n_real_data

    # prediction
        prediction = np.array(prediction)
        prediction = np.reshape(prediction, (len(prediction), 1))

        final_prediction_prices = prediction_prices
        final_prediction_prices = np.row_stack((final_prediction_prices, predi

        plt.plot(actual_prices, color='black', label='Actual Prices')
        plt.plot(final_prediction_prices, color='green', label='Predicted Pric
        plt.title("{} Price Prediction with 7 days forecast".format(crypto_cur
        plt.xlabel("Time")
        plt.ylabel("Price")
        plt.legend(loc='upper left')
        plt.show()

        price_today = actual_prices[len(actual_prices)-1]
        pred_price_today = prediction_prices[len(prediction_prices)-1][0]

        max_price = prediction[0][0]
        min_price = prediction[0][0]
        for i in range(len(prediction)):
         max_price = max(max_price, prediction[i][0])
         min_price = min(min_price, prediction[i][0])

        upside = (((max_price - pred_price_today)*100)/pred_price_today)
        downside = (((min_price - pred_price_today)*100)/pred_price_today)

        return [upside, downside]

sides1 = prediction_function(crypto_currency1)
sides2 = prediction_function(crypto_currency2)
sides3 = prediction_function(crypto_currency3)
```

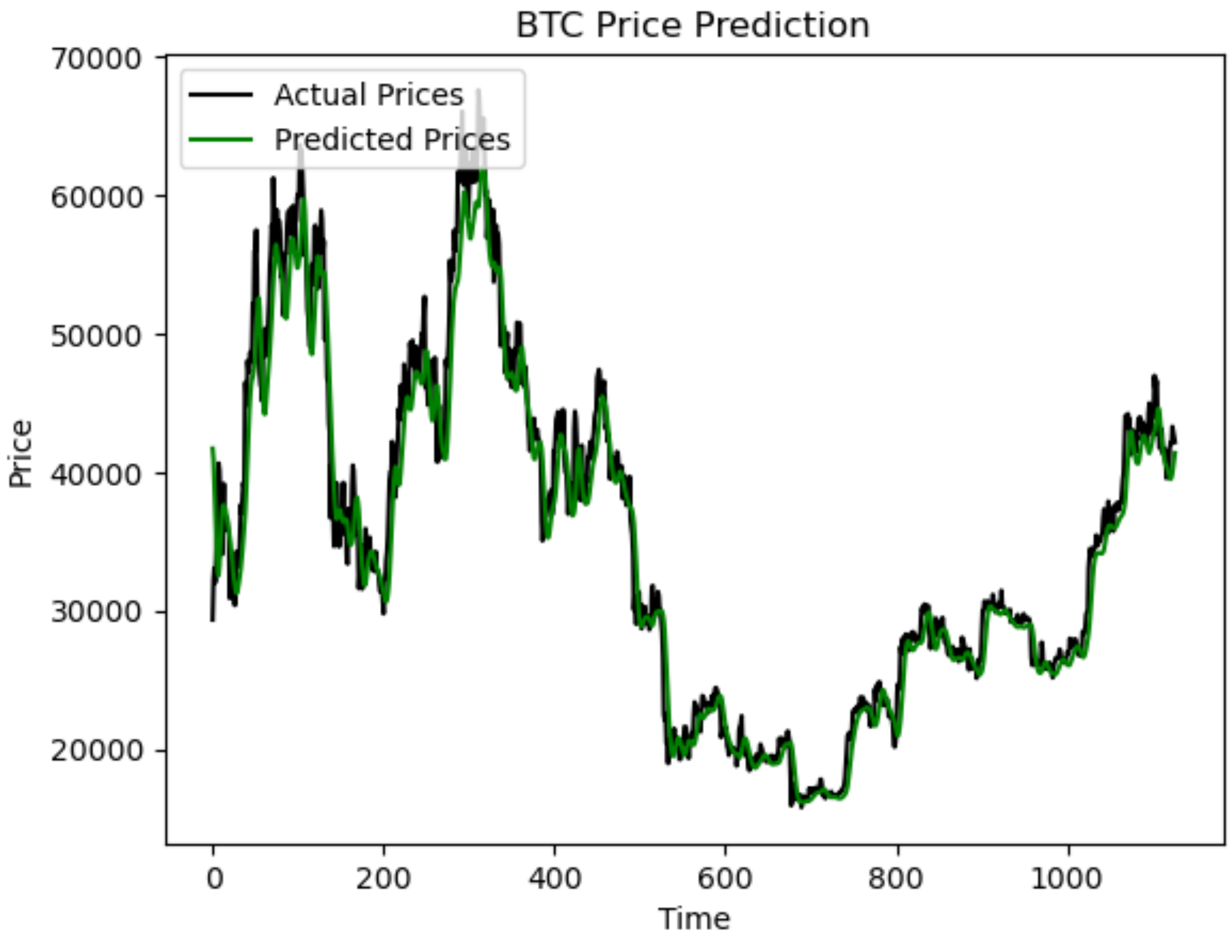
```
 The number of samples in our training data = 1548
```

	d	o	h	l	c	v
481	2021-01-01	29326.55	29350.00	29326.55	29337.16	81.585
482	2021-01-02	32208.71	32222.00	32172.18	32199.91	291.592
483	2021-01-03	32984.17	33077.34	32965.54	33054.53	535.856
484	2021-01-04	32049.71	32068.00	32022.56	32031.07	251.813
485	2021-01-05	33944.26	34016.13	33937.37	33999.52	141.828

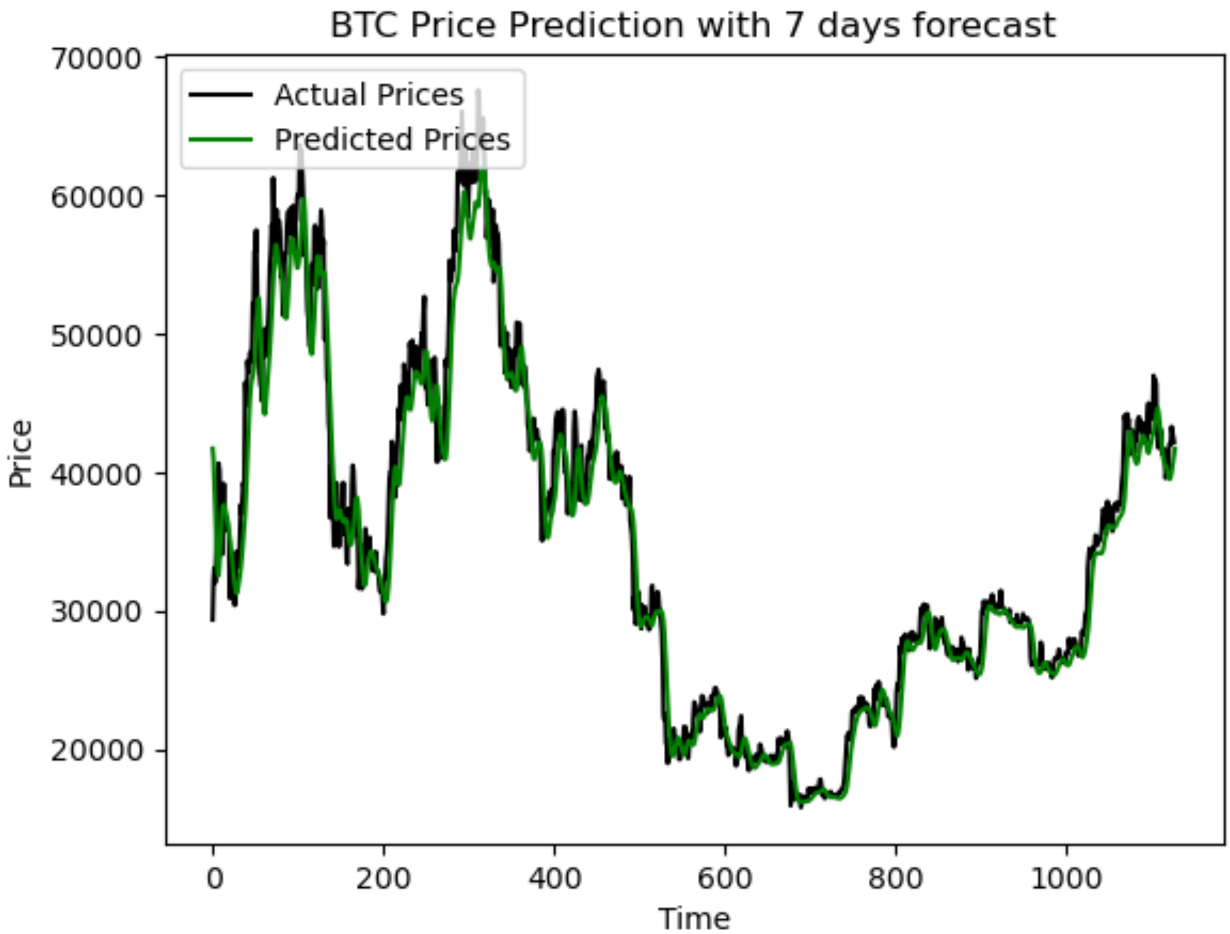
```
 The number of samples in our testing data = 1127
```

```
Epoch 1/25
/Users/charan/anaconda3/lib/python3.11/site-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)
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - loss: 0.0673
Epoch 2/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 0.0049
Epoch 3/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0049
Epoch 4/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0043
Epoch 5/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0037
Epoch 6/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0038
Epoch 7/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0040
Epoch 8/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0038
Epoch 9/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0035
Epoch 10/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 0.0035
Epoch 11/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0031
Epoch 12/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0030
Epoch 13/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0034
Epoch 14/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0029
Epoch 15/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0026
Epoch 16/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0029
Epoch 17/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0029
Epoch 18/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0024
Epoch 19/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0023
Epoch 20/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0022
Epoch 21/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0026
Epoch 22/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0023
Epoch 23/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0025
Epoch 24/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0022
Epoch 25/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0023
36/36 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```




```
 The number of samples in our training data = 1548
```

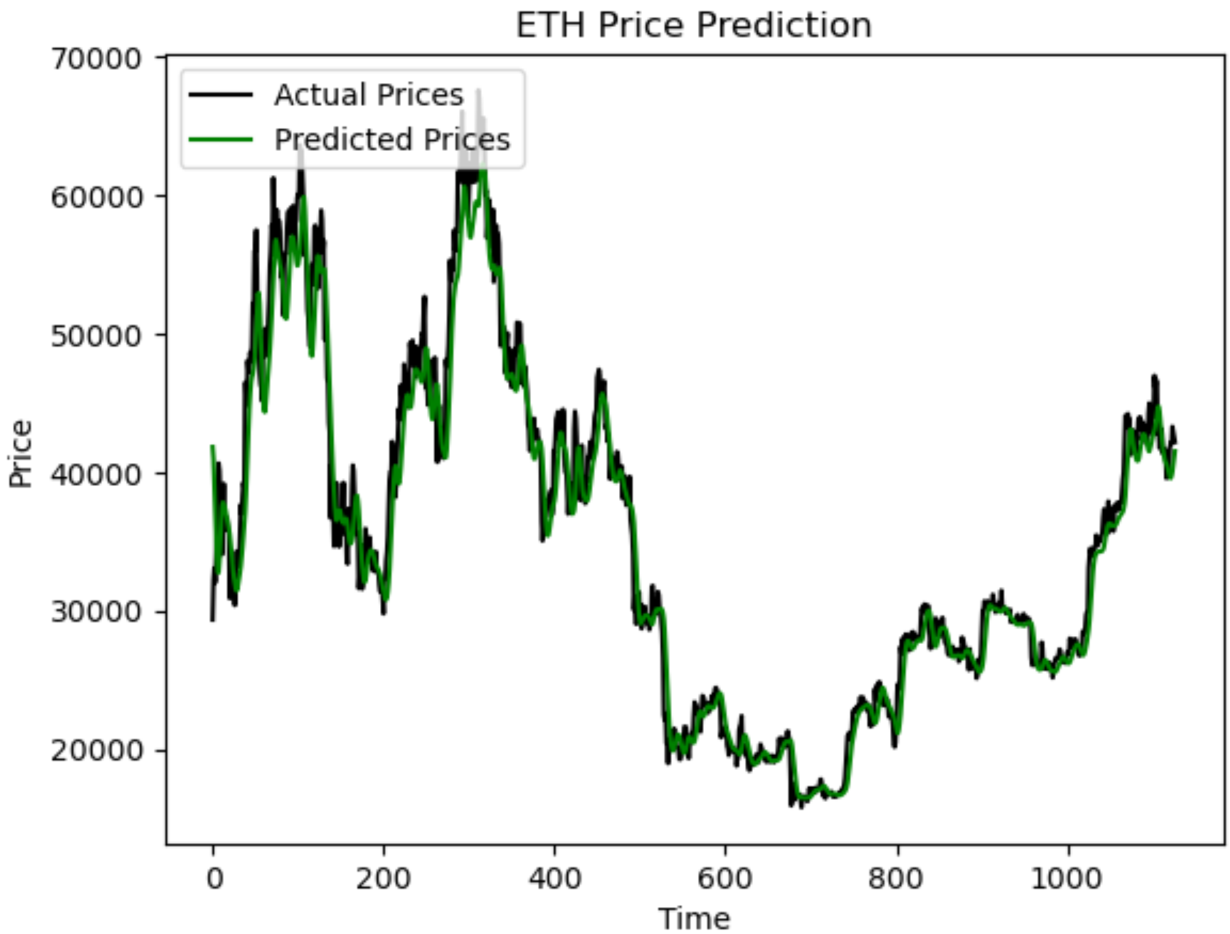
```
             d         o         h         l         c        v
481 2021-01-01  29326.55  29350.00  29326.55  29337.16   81.585
482 2021-01-02  32208.71  32222.00  32172.18  32199.91  291.592
483 2021-01-03  32984.17  33077.34  32965.54  33054.53  535.856
484 2021-01-04  32049.71  32068.00  32022.56  32031.07  251.813
485 2021-01-05  33944.26  34016.13  33937.37  33999.52  141.828
```

```
 The number of samples in our testing data = 1127
```

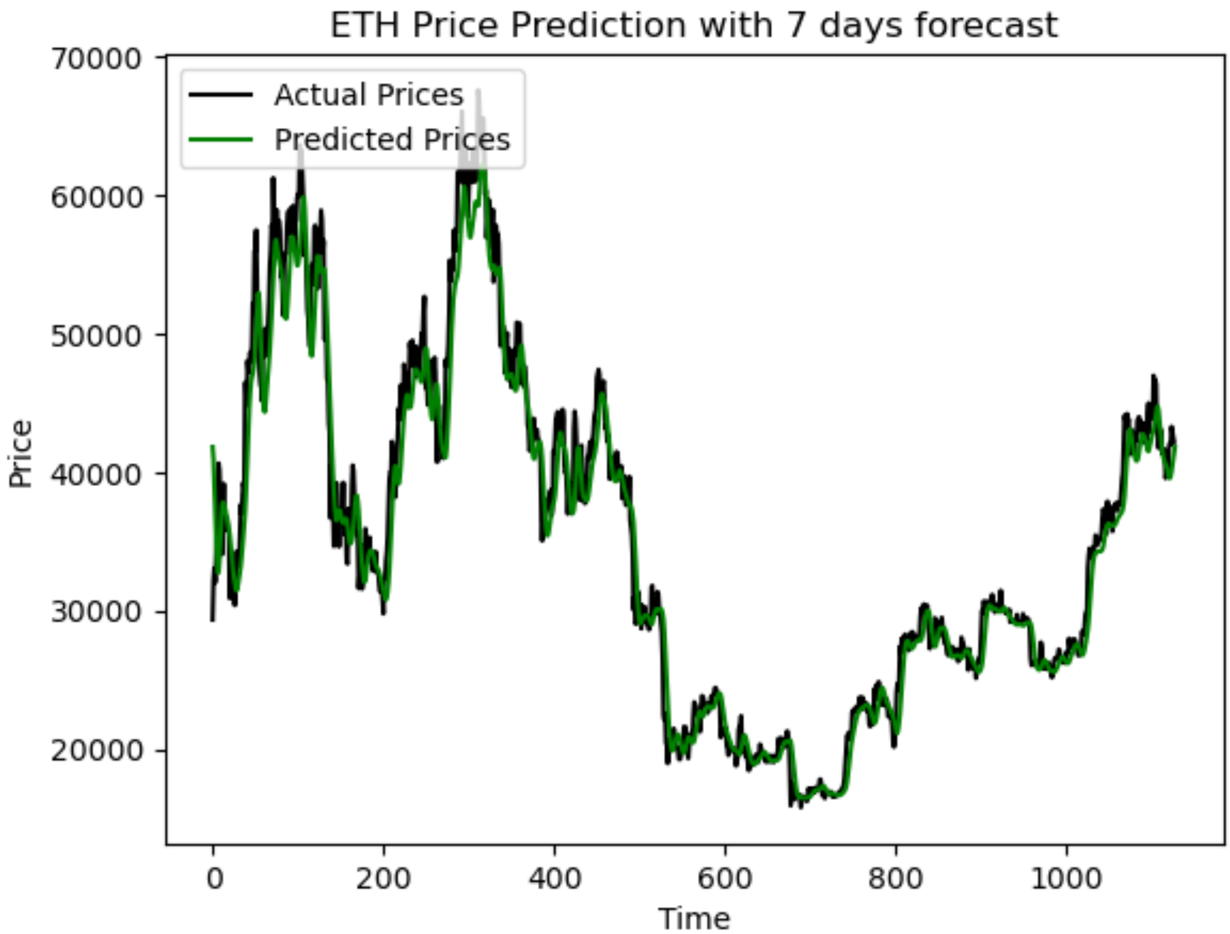
```
Epoch 1/25
```

```
/Users/charan/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rn
n.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)
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - loss: 0.0530
Epoch 2/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 0.0057
Epoch 3/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0043
Epoch 4/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0041
Epoch 5/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0041
Epoch 6/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0035
Epoch 7/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0039
Epoch 8/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0038
Epoch 9/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0032
Epoch 10/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0033
Epoch 11/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0032
Epoch 12/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0032
Epoch 13/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0033
Epoch 14/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0035
Epoch 15/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0026
Epoch 16/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0027
Epoch 17/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0031
Epoch 18/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0024
Epoch 19/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0022
Epoch 20/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0024
Epoch 21/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0025
Epoch 22/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0026
Epoch 23/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0022
Epoch 24/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0020
Epoch 25/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0021
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
```



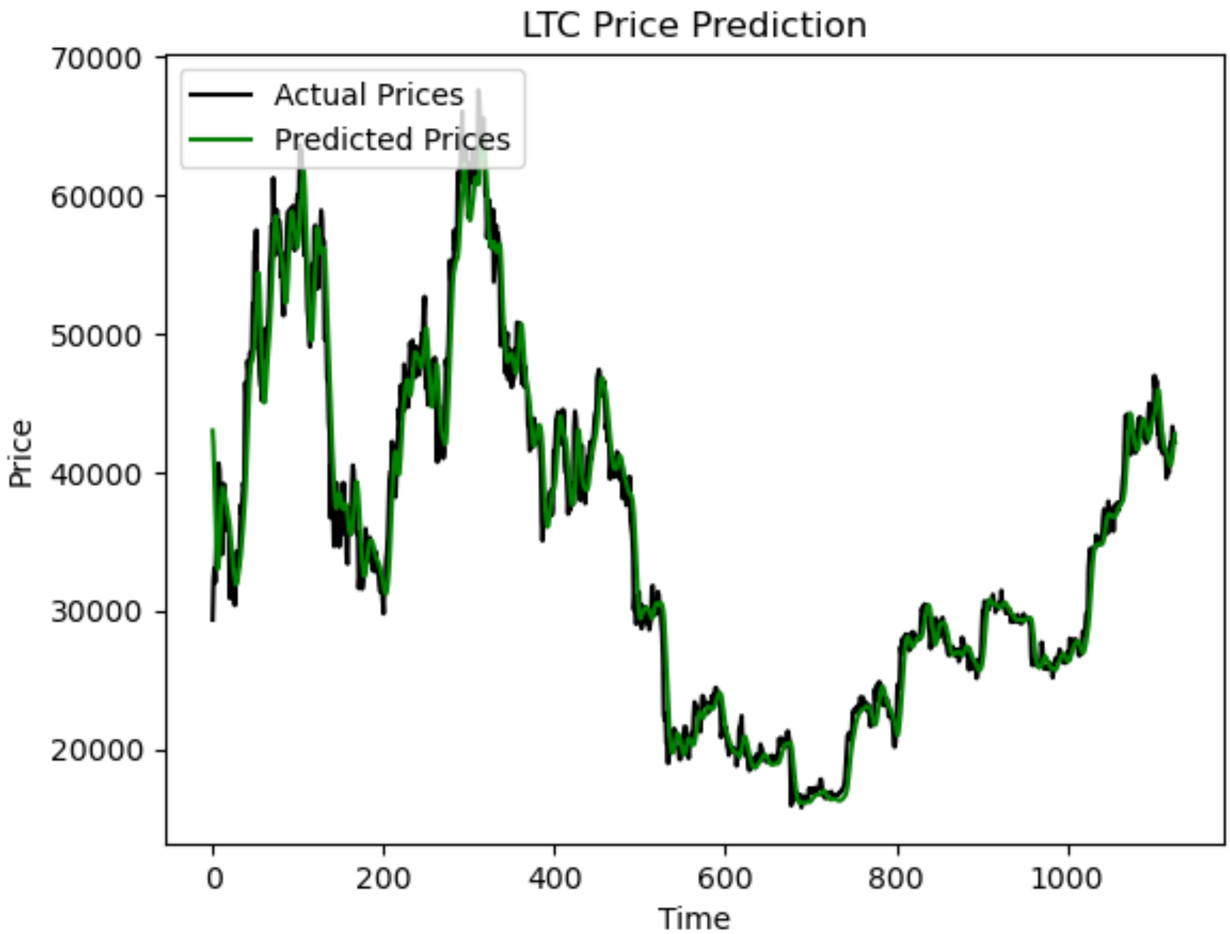

```
 The number of samples in our training data = 1548
```

```
              d         o         h         l         c        v
481 2021-01-01  29326.55  29350.00  29326.55  29337.16   81.585
482 2021-01-02  32208.71  32222.00  32172.18  32199.91  291.592
483 2021-01-03  32984.17  33077.34  32965.54  33054.53  535.856
484 2021-01-04  32049.71  32068.00  32022.56  32031.07  251.813
485 2021-01-05  33944.26  34016.13  33937.37  33999.52  141.828
```

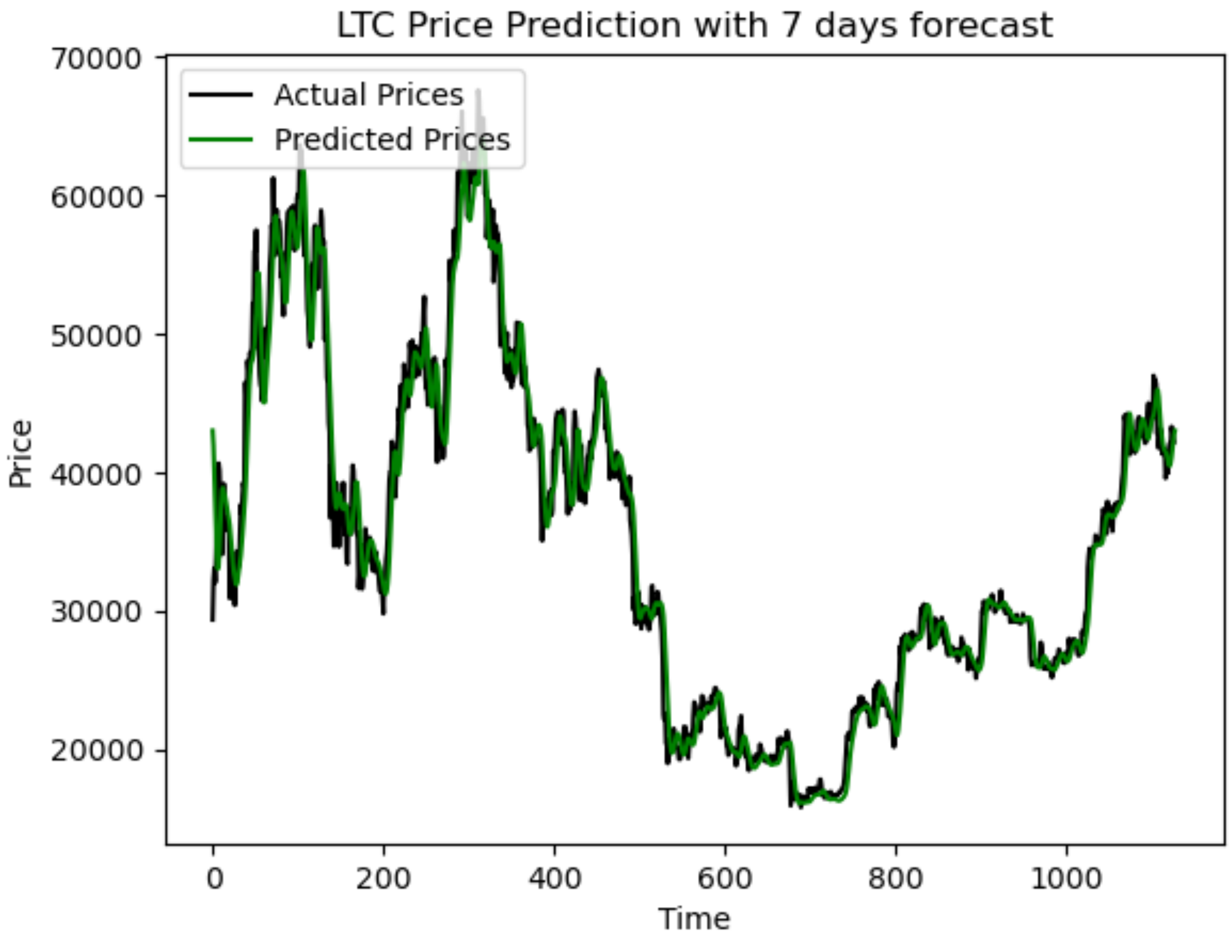
```
 The number of samples in our testing data = 1127
```

```
Epoch 1/25
/Users/charan/anaconda3/lib/python3.11/site-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)
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - loss: 0.0560
Epoch 2/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0049
Epoch 3/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 0.0048
Epoch 4/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0038
Epoch 5/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0039
Epoch 6/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0042
Epoch 7/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0036
Epoch 8/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0033
Epoch 9/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 0.0030
Epoch 10/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - loss: 0.0026
Epoch 11/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0028
Epoch 12/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - loss: 0.0027
Epoch 13/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0030
Epoch 14/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0024
Epoch 15/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0027
Epoch 16/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0024
Epoch 17/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0025
Epoch 18/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0020
Epoch 19/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0022
Epoch 20/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 0.0022
Epoch 21/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 0.0025
Epoch 22/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0025
Epoch 23/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0026
Epoch 24/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0020
Epoch 25/25
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0019
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
```



In [4]:

```
print("The R2 accuracy score for BTC = {} %".format(acc[0]*100))
print("The R2 accuracy score for ETH = {} %".format(acc[1]*100))
print("The R2 accuracy score for LTC = {} %".format(acc[2]*100))

print("Average R2 accuracy score = {} %".format((acc[0]+acc[1]+acc[2])*100/3
```

```
The R2 accuracy score for BTC = 95.99586459685221 %
The R2 accuracy score for ETH = 96.22229285920754 %
The R2 accuracy score for LTC = 96.90199949622296 %
Average R2 accuracy score = 96.3733856507609 %
```

In [7]:
```
amount = np.float64(input("Enter the amount: "))

if sides1[0] < 0:
  sides1[0] = 0

if sides2[0] < 0:
  sides2[0] = 0

if sides3[0] < 0:
  sides3[0] = 0

if sides1[0] == 0 and sides2[0] == 0 and sides3[0] == 0:
  print("This is not the right time to invest! All the cryptocurrencies are

else:
  BTC_share = amount*(sides1[0]/(sides1[0] + sides2[0] + sides3[0]))
  BTC_upside = (BTC_share*sides1[0])/100
  BTC_downside = (BTC_share*sides1[1])/100

  ETH_share = amount*(sides2[0]/(sides1[0] + sides2[0] + sides3[0]))
  ETH_upside = (ETH_share*sides2[0])/100
  ETH_downside = (ETH_share*sides2[1])/100

  LTC_share = amount*(sides3[0]/(sides1[0] + sides2[0] + sides3[0]))
  LTC_upside = (LTC_share*sides3[0])/100
  LTC_downside = (LTC_share*sides3[1])/100

  total_upside = BTC_upside + ETH_upside + LTC_upside
  total_downside = BTC_downside + ETH_downside + LTC_downside

  BTC_share = round(BTC_share, 2)
  ETH_share = round(ETH_share, 2)
  LTC_share = round(LTC_share, 2)

  print("You should invest\n {} {} in {}\n {} {} in {}\n {} {} in {} \nFor a
                                                                   agai
                                                                   agai
                                                                   agai
                                                                   roun
                                                                   roun
```

```
You should invest
 USD 892.07 in BTC
 USD 843.91 in ETH
 USD 764.02 in LTC
For an upside of 0.72% and downside of 0.72% !
```

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```