```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout,GRU
from keras import optimizers
from sklearn.metrics import mean_squared_error
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

data_raw = pd.read_csv("BTC.csv", index_col="Date",
parse_dates=["Date"])
data_raw
```

```
                   Open          High           Low         Close  \
Date
2019-06-01   8573.839844   8625.600586   8481.578125   8564.016602
2019-06-02   8565.473633   8809.303711   8561.235352   8742.958008
2019-06-03   8741.747070   8743.500000   8204.185547   8208.995117
2019-06-04   8210.985352   8210.985352   7564.488770   7707.770996
2019-06-05   7704.343262   7901.849121   7668.668457   7824.231445
...                  ...           ...           ...           ...
2022-05-28  28622.625000  28814.900391  28554.566406  28814.900391
2022-05-29  29019.867188  29498.009766  28841.107422  29445.957031
2022-05-30  29443.365234  31949.630859  29303.572266  31726.390625
2022-05-31  31723.865234  32249.863281  31286.154297  31792.310547
2022-06-01  31792.554688  31957.285156  29501.587891  29799.080078

               Adj Close       Volume
Date
2019-06-01   8564.016602  22488303544
2019-06-02   8742.958008  20266216022
2019-06-03   8208.995117  22004511436
2019-06-04   7707.770996  24609731549
2019-06-05   7824.231445  21760923463
...                  ...          ...
2022-05-28  28814.900391  35519577634
2022-05-29  29445.957031  18093886409
2022-05-30  31726.390625  39277993274
2022-05-31  31792.310547  33538210634
2022-06-01  29799.080078  41135817341

[1097 rows x 6 columns]
```

```
dataset = pd.DataFrame(data_raw["Close"])
```

```
from sklearn.preprocessing import MinMaxScaler
dataset_norm = data_raw.copy()
data_raw[['Close']]
scaler = MinMaxScaler()
dataset_norm['Close'] = scaler.fit_transform(data_raw[['Close']])
dataset_norm
```

```
                    Open          High           Low     Close     Adj
Close  \
Date

2019-06-01   8573.839844   8625.600586   8481.578125  0.057403
8564.016602
2019-06-02   8565.473633   8809.303711   8561.235352  0.060262
8742.958008
2019-06-03   8741.747070   8743.500000   8204.185547  0.051732
8208.995117
2019-06-04   8210.985352   8210.985352   7564.488770  0.043725
7707.770996
2019-06-05   7704.343262   7901.849121   7668.668457  0.045585
7824.231445
...                  ...           ...           ...       ...
...
2022-05-28  28622.625000  28814.900391  28554.566406  0.380920
28814.900391
2022-05-29  29019.867188  29498.009766  28841.107422  0.391002
29445.957031
2022-05-30  29443.365234  31949.630859  29303.572266  0.427433
31726.390625
2022-05-31  31723.865234  32249.863281  31286.154297  0.428486
31792.310547
2022-06-01  31792.554688  31957.285156  29501.587891  0.396643
29799.080078

                Volume
Date
2019-06-01  22488303544
2019-06-02  20266216022
2019-06-03  22004511436
2019-06-04  24609731549
2019-06-05  21760923463
...                 ...
2022-05-28  35519577634
2022-05-29  18093886409
2022-05-30  39277993274
2022-05-31  33538210634
2022-06-01  41135817341

[1097 rows x 6 columns]
```

```
x =  dataset_norm.drop(["Volume"],axis=1).values
x = dataset_norm.drop(["Close"],axis=1).values
print(x)
```

```
[[8.57383984e+03 8.62560059e+03 8.48157812e+03 8.56401660e+03
  2.24883035e+10]
 [8.56547363e+03 8.80930371e+03 8.56123535e+03 8.74295801e+03
  2.02662160e+10]
 [8.74174707e+03 8.74350000e+03 8.20418555e+03 8.20899512e+03
  2.20045114e+10]
 ...
 [2.94433652e+04 3.19496309e+04 2.93035723e+04 3.17263906e+04
  3.92779933e+10]
 [3.17238652e+04 3.22498633e+04 3.12861543e+04 3.17923105e+04
  3.35382106e+10]
 [3.17925547e+04 3.19572852e+04 2.95015879e+04 2.97990801e+04
  4.11358173e+10]]
```

```
y = dataset_norm["Close"].values
print(y)
```

```
[0.05740345 0.06026212 0.05173182 ... 0.42743283 0.42848593
0.39664317]
```

```
totaldata = data_raw.values
totaldatatrain = int(len(totaldata)*0.8)
totaldatatest = int(len(totaldata)*0.2)

training_set = dataset_norm[0:totaldatatrain]
test_set = dataset_norm[totaldatatrain:]

#Sliding windows
lag = 2
# sliding windows function
def create_sliding_windows(data,len_data,lag):
    x=[]
    y=[]
    for i in range(lag,len_data):
        x.append(data[i-lag:i,0])
        y.append(data[i,0])
    return np.array(x),np.array(y)

# Formating data into array for create sliding windows
array_training_set = np.array(training_set)
array_test_set = np.array(test_set)

# Create sliding windows into training data
x_train, y_train =
create_sliding_windows(array_training_set,len(array_training_set),
lag)
# Create sliding windows into test data
```

```
x_test,y_test =
create_sliding_windows(array_test_set,len(array_test_set),lag)

from sklearn.linear_model import LinearRegression
ml=LinearRegression()
ml.fit(x_train, y_train)

LinearRegression()

y_pred=ml.predict(x_test)
print(y_pred)

[62986.70680613 60599.85358598 58668.63243394 60576.63930552
 62225.02565961 61958.69324884 61437.03027932 61069.0613143
 63199.06720959 63045.89384072 61635.03923121 61176.82036232
 61609.36983841 63319.59666735 67379.37295926 67076.88159913
 65185.85063034 64956.95095037 64286.93471521 64522.87897642
 65542.25059498 63916.82585641 60442.55655016 60430.23842803
 57190.19206443 58122.85587962 59714.82184701 58852.32694543
 56533.26403543 57570.74843269 57271.9200099  58933.20875309
 54136.68946708 54827.82003543 57222.06282914 57879.44856814
 57046.29892413 57280.40925644 56634.18509967 53978.48237816
 49556.71115061 49478.47277564 50588.8256037  50741.01459348
 50541.92417991 47891.24901239 47364.63518078 49304.73987515
 50146.52495765 46995.02481937 48354.88708786 48946.27657554
 47807.1887948  46383.47662738 46891.64707617 46792.64291622
 46951.78824216 48889.16897457 48722.95677041 50751.19947577
 50930.6445517  50533.30678043 50858.39375893 50766.01054402
 47888.16379813 46636.51318293 47204.60821659 46440.67375343
 47674.12573505 47441.12732055 46589.51323825 46009.80874258
 43783.47158707 43253.56642931 41732.94814461 41798.13814572
 41973.59499739 41899.13687677 42760.19472391 43948.10620046
 42755.92447766 43145.80946628 43242.50838173 43196.16351242
 42377.55012942 42440.79998952 41856.65883747 40837.08963411
 36801.9512089  35205.2048883  36271.69166894 36702.99244119
 37003.962841   36920.04387012 37182.54436682 37812.74360343
 38201.35819764 38006.53718735 38519.71741355 38800.1534231
 37126.5826462  37208.40584871 41308.73504495 41518.69512289
 42421.98986318 43841.06188221 44157.11568102 44407.78827484
 43693.68629491 42557.67582506 42321.61861843 42236.49345709
 42634.64206625 44531.73296966 44051.51633991 40832.94028215
 40131.40487604 40185.6551472  38599.37664556 37223.37948199
 38282.99956912 37411.97974235 38341.37387929 39231.93598491
 39178.37767473 37863.28187971 42933.01137627 44361.73217045
 44026.84333621 42622.42374908 39423.92411005 39461.35752154
 38561.36066102 38154.67078825 38773.46276108 41850.24736832
 39668.07466722 38906.52756783 38951.75669717 37981.96384143
 39625.82994408 39428.50025644 41103.95926608 41030.46916654
 41816.64004381 42241.35965092 41377.82359308 41158.33231516
 42359.67722336 42929.2734254  43973.68076987 44401.62482526
 44571.28138241 46756.5690285  47160.15743939 47512.0305629
```

```
 47163.10275888 45722.54819778 46317.06136274 45961.40147866
 46485.81264298 46690.06991449 45686.52205398 43425.45990602
 43561.8391834  42431.32143766 42825.08180309 42310.84473356
 39770.03131967 40160.62878566 41170.43651367 40087.30265881
 40588.18276774 40499.36958085 39836.87176738 40833.91402483
 41534.3847657  41453.33470921 40651.07977929 39859.94915597
 39567.19185464 39545.82180886 40462.0003926  38335.1074966
 39245.48400858 39809.33254352 38749.51625592 37839.91067785
 38498.03771746 38597.05992218 37867.7822193  39649.3895858
 36835.86811038 36146.11530024 35606.81935646 34218.49844489
 30573.76661975 31039.20103456 29131.57922544 29092.62936352
 29337.65586204 30116.8170072  31299.15765822 30018.67203185
 30458.24784301 28892.09706992 30281.80416972 29335.52040579
 29485.91670408 30323.81607505 29242.67895276 29687.12352419
 29637.95478057 29337.959112   28728.41209722 29063.0123767
 29485.11972025 31653.01816307]
```

```
set_test = dataset["Close"]
set_test
```

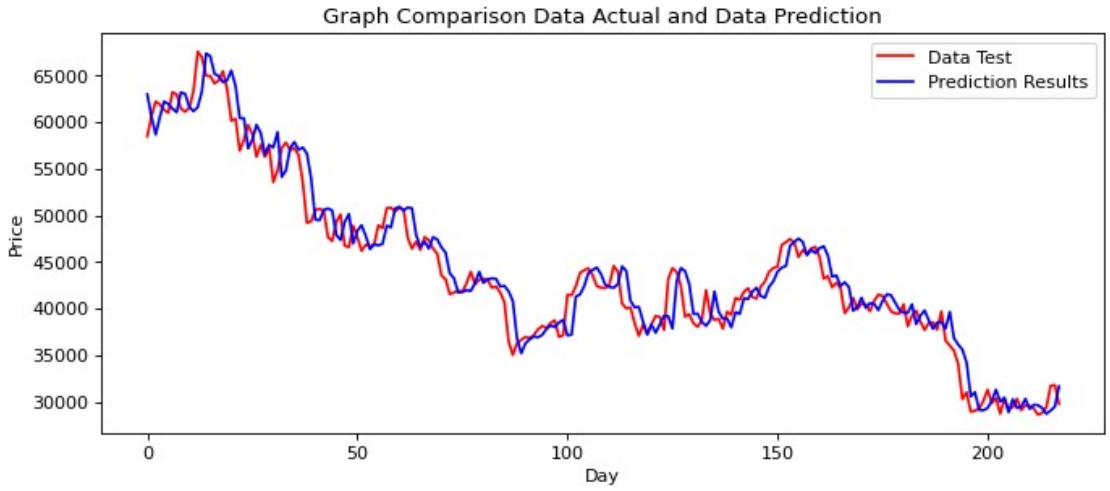
```
Date
2019-06-01     8564.016602
2019-06-02     8742.958008
2019-06-03     8208.995117
2019-06-04     7707.770996
2019-06-05     7824.231445
                  ...
2022-05-28    28814.900391
2022-05-29    29445.957031
2022-05-30    31726.390625
2022-05-31    31792.310547
2022-06-01    29799.080078
Name: Close, Length: 1097, dtype: float64
```

```
datacompare = pd.DataFrame()
datatest=np.array(set_test[totaldatatrain+lag:])
datapred= y_pred

datacompare['Data Test'] = datatest
datacompare['Prediction Results'] = datapred
datacompare
```

```
        Data Test  Prediction Results
0    58482.386719        62986.706806
1    60622.136719        60599.853586
2    62227.964844        58668.632434
3    61888.832031        60576.639306
4    61318.957031        62225.025660
..            ...                 ...
213  28814.900391        29337.959112
214  29445.957031        28728.412097
```

```
215  31726.390625        29063.012377
216  31792.310547        29485.119720
217  29799.080078        31653.018163

[218 rows x 2 columns]
```

```
plt.figure(num=None, figsize=(10, 4), dpi=80,facecolor='w',
edgecolor='k')
plt.title('Graph Comparison Data Actual and Data Prediction')
plt.plot(datacompare['Data Test'], color='red',label='Data Test')
plt.plot(datacompare['Prediction Results'],
color='blue',label='Prediction Results')
plt.xlabel('Day')
plt.ylabel('Price')
plt.legend()
plt.show()
```




```
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
MAPE(datatest, datapred)
```

```
3.6906909475836627
```

```
from sklearn.metrics import mean_squared_error
import math
MSE = mean_squared_error(datatest, datapred)
RMSE = math.sqrt(MSE)
print(RMSE)
```

```
2131.1683262280403
```

```
APE = []

# Iterate over the list values
for day in range(5):
```

```
    # Calculate percentage error
    per_err = (datatest[day] - datapred[day]) / datatest[day]

    # Take absolute value of
    # the percentage error (APE)
    per_err = abs(per_err)

    # Append it to the APE list
    APE.append(per_err)

# Calculate the MAPE
MAPE = sum(APE)/len(APE)
print(MAPE)
```

```
0.034112940547860376
```