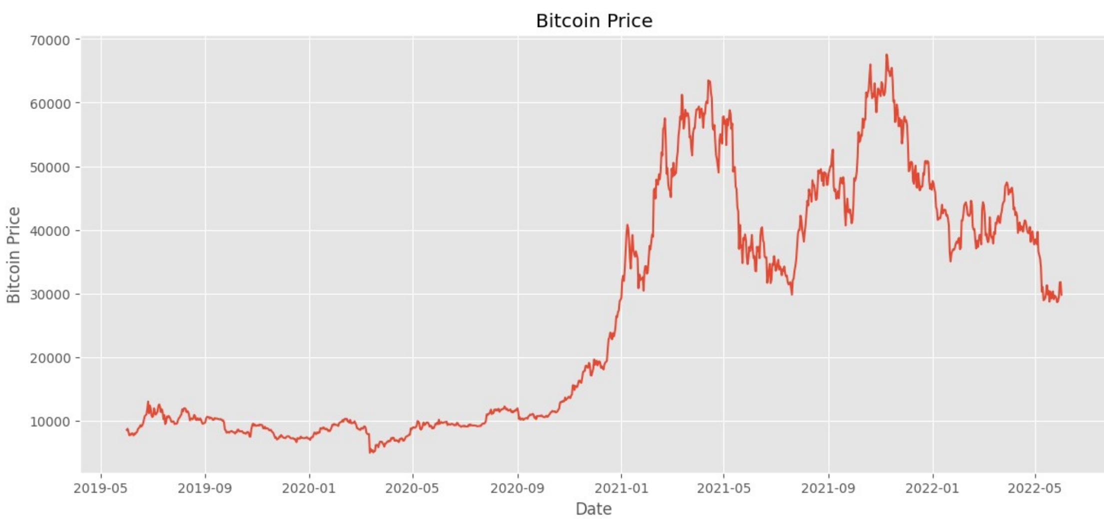
```python
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout,GRU
from keras import optimizers
from sklearn.metrics import mean_squared_error

seed = 1234
np.random.seed(seed)
plt.style.use('ggplot')

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]
```

```
# use feature 'Date' & 'Close'
dataset = pd.DataFrame(data_raw['Close'])
print(' Count row of data: ',len(dataset))

fig = plt.figure(figsize=(14, 6))
plt.plot(dataset)
plt.xlabel('Date')
plt.ylabel('Bitcoin Price')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m"))
plt.title('Bitcoin Price')
plt.show()
```

```
 Count row of data:  1097
```

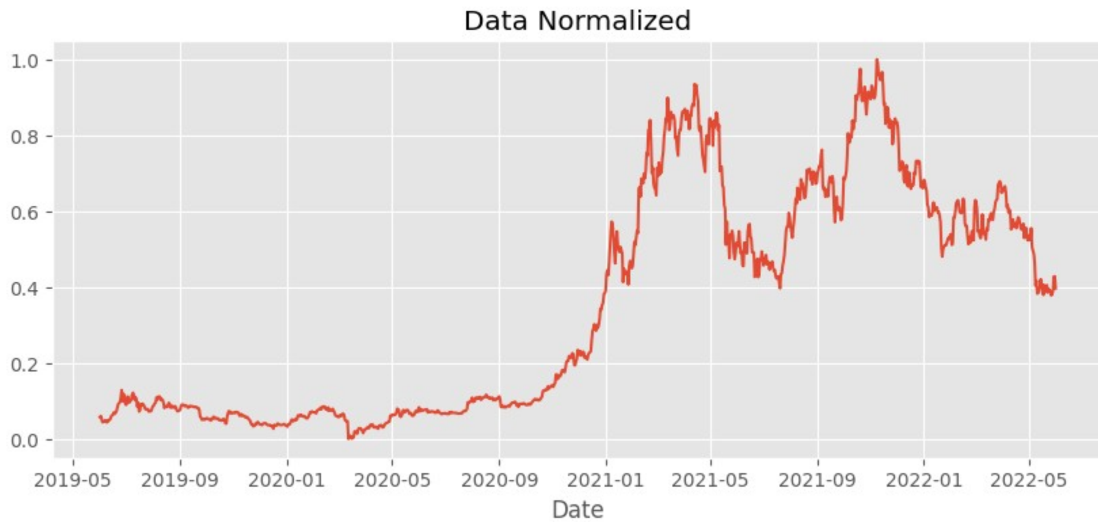



```
#Min-Max Normalization
dataset_norm = dataset.copy()
dataset[['Close']]
scaler = MinMaxScaler()
dataset_norm['Close'] = scaler.fit_transform(dataset[['Close']])
dataset_norm
```

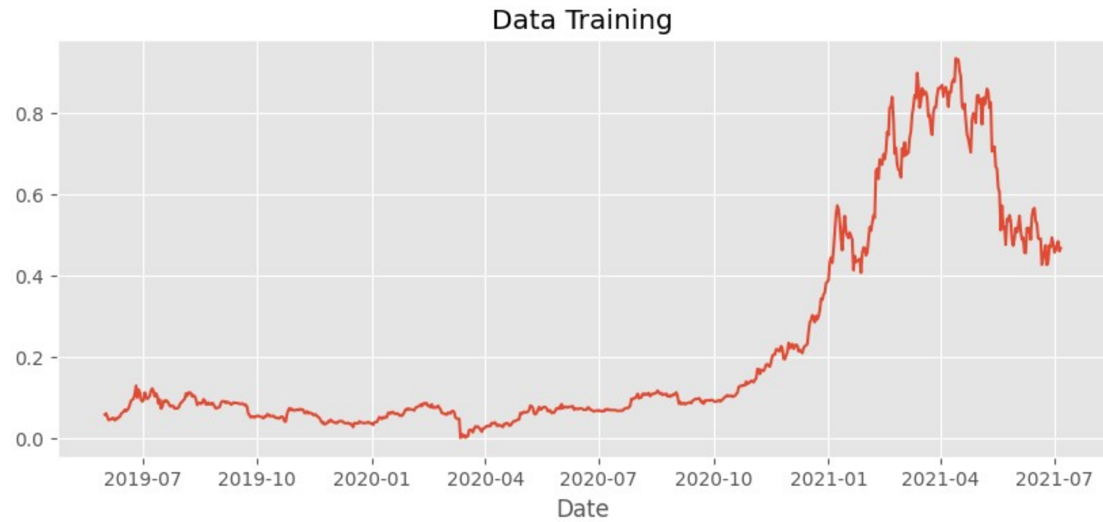
```
               Close
Date
2019-06-01  0.057403
2019-06-02  0.060262
2019-06-03  0.051732
2019-06-04  0.043725
2019-06-05  0.045585
...              ...
2022-05-28  0.380920
```

```
2022-05-29  0.391002
2022-05-30  0.427433
2022-05-31  0.428486
2022-06-01  0.396643

[1097 rows x 1 columns]
```

```
fig = plt.figure(figsize=(10, 4))
plt.plot(dataset_norm)
plt.xlabel('Date')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m"))
plt.title('Data Normalized')
plt.show()
```

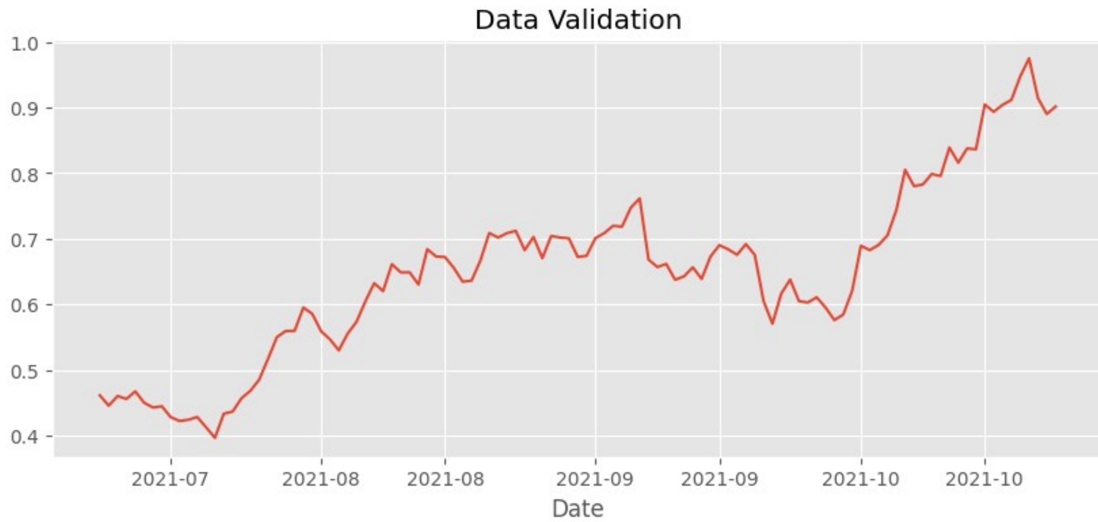



```
# Partition data into data train, val & test
totaldata = dataset.values
totaldatatrain = int(len(totaldata)*0.7)
totaldataval = int(len(totaldata)*0.1)
totaldatatest = int(len(totaldata)*0.2)

# Store data into each partition
training_set = dataset_norm[0:totaldatatrain]
val_set=dataset_norm[totaldatatrain:totaldatatrain+totaldataval]
test_set = dataset_norm[totaldatatrain+totaldataval:]

# graph of data training
fig = plt.figure(figsize=(10, 4))
plt.plot(training_set)
plt.xlabel('Date')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m"))
plt.title('Data Training')
plt.show()
```

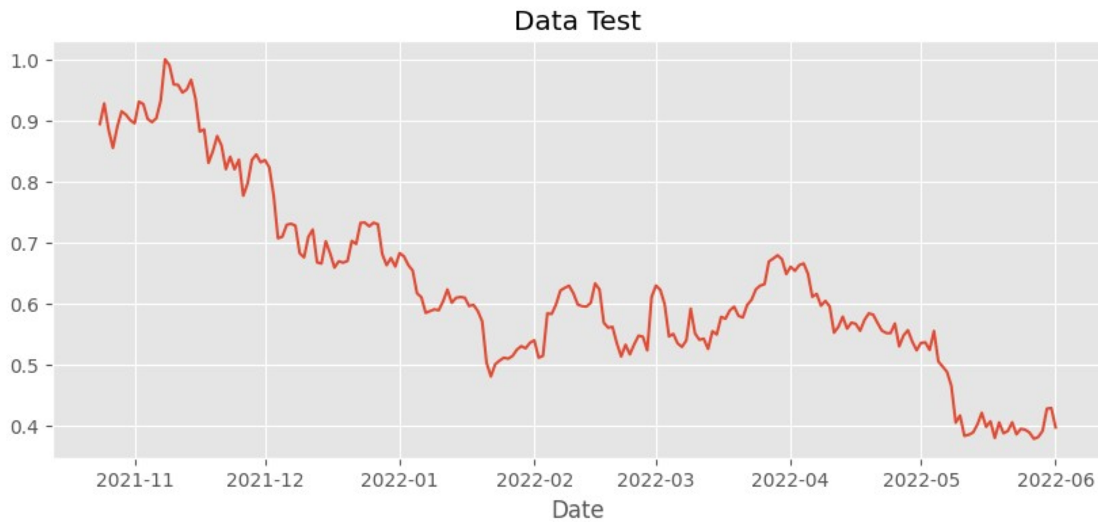



```
# graph of data validation
fig = plt.figure(figsize=(10, 4))
plt.plot(val_set)
plt.xlabel('Date')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m"))
plt.title('Data Validation')
val_set
```

```
               Close
Date
2021-07-07  0.461444
2021-07-08  0.445820
2021-07-09  0.460528
2021-07-10  0.456095
2021-07-11  0.467592
...              ...
2021-10-19  0.947204
2021-10-20  0.974855
2021-10-21  0.914425
2021-10-22  0.890176
2021-10-23  0.901380

[109 rows x 1 columns]
```




```
# graph of data test
fig = plt.figure(figsize=(10, 4))
plt.plot(test_set)
plt.xlabel('Date')
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m"))
plt.title('Data Test')
plt.show()
test_set
```




```
               Close
Date
2021-10-24  0.893987
2021-10-25  0.927679
2021-10-26  0.884928
2021-10-27  0.854872
2021-10-28  0.889055
...              ...
```

```
2022-05-28  0.380920
2022-05-29  0.391002
2022-05-30  0.427433
2022-05-31  0.428486
2022-06-01  0.396643

[221 rows x 1 columns]
```

```
# Initiaton value of lag
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_val_set = np.array(val_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)
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
# Create sliding windows into validation data
x_val,y_val =
create_sliding_windows(array_val_set,len(array_val_set),lag)
x_val = np.reshape(x_val, (x_val.shape[0],x_val.shape[1],1))
# Create sliding windows into test data
x_test,y_test =
create_sliding_windows(array_test_set,len(array_test_set),lag)
x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))

# Hyperparameters
learning_rate = 0.0001
hidden_unit = 64
batch_size=256
epoch = 100

# Architecture Gated Recurrent Unit
regressorGRU = Sequential()

# First GRU layer with dropout
regressorGRU.add(GRU(units=hidden_unit, return_sequences=True,
input_shape=(x_train.shape[1],1), activation = 'tanh'))
```

```python
regressorGRU.add(Dropout(0.2))
# Second GRU layer with dropout
regressorGRU.add(GRU(units=hidden_unit, return_sequences=True,
activation = 'tanh'))
regressorGRU.add(Dropout(0.2))
# Third GRU layer with dropout
regressorGRU.add(GRU(units=hidden_unit, return_sequences=False,
activation = 'tanh'))
regressorGRU.add(Dropout(0.2))

# Output layer
regressorGRU.add(Dense(units=1))

# Compiling the Gated Recurrent Unit
regressorGRU.compile(optimizer=tf.keras.optimizers.Adam(lr=learning_ra
te),loss='mean_squared_error')

# Fitting ke data training dan data validation
pred = regressorGRU.fit(x_train, y_train,
validation_data=(x_val,y_val), batch_size=batch_size, epochs=epoch)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead,
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

Epoch 1/100
3/3 [==============================] - 6s 491ms/step - loss: 0.1062 -
val_loss: 0.3149
Epoch 2/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0699 -
val_loss: 0.1968
Epoch 3/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0469 -
val_loss: 0.0927
Epoch 4/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0380 -
val_loss: 0.0310
Epoch 5/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0355 -
val_loss: 0.0168
Epoch 6/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0261 -
val_loss: 0.0197
Epoch 7/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0152 -
val_loss: 0.0253
Epoch 8/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0086 -
val_loss: 0.0212
Epoch 9/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0058 -
```

```
val_loss: 0.0057
Epoch 10/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0029 -
val_loss: 0.0032
Epoch 11/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0047 -
val_loss: 0.0092
Epoch 12/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0050 -
val_loss: 0.0034
Epoch 13/100
3/3 [==============================] - 0s 32ms/step - loss: 0.0038 -
val_loss: 0.0015
Epoch 14/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0024 -
val_loss: 0.0025
Epoch 15/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0023 -
val_loss: 0.0023
Epoch 16/100
3/3 [==============================] - 0s 32ms/step - loss: 0.0020 -
val_loss: 0.0019
Epoch 17/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0025 -
val_loss: 0.0021
Epoch 18/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0027 -
val_loss: 0.0031
Epoch 19/100
3/3 [==============================] - 0s 35ms/step - loss: 0.0022 -
val_loss: 0.0034
Epoch 20/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0023 -
val_loss: 0.0018
Epoch 21/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0021 -
val_loss: 0.0011
Epoch 22/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0021 -
val_loss: 0.0013
Epoch 23/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0023 -
val_loss: 0.0012
Epoch 24/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0021 -
val_loss: 0.0017
Epoch 25/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0022 -
val_loss: 0.0016
Epoch 26/100
```

```
3/3 [==============================] - 0s 31ms/step - loss: 0.0022 -
val_loss: 0.0013
Epoch 27/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0021 -
val_loss: 0.0013
Epoch 28/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0019 -
val_loss: 0.0015
Epoch 29/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0022 -
val_loss: 0.0019
Epoch 30/100
3/3 [==============================] - 0s 32ms/step - loss: 0.0019 -
val_loss: 0.0015
Epoch 31/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0018 -
val_loss: 0.0011
Epoch 32/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0017 -
val_loss: 0.0011
Epoch 33/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0017 -
val_loss: 0.0012
Epoch 34/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0017 -
val_loss: 0.0012
Epoch 35/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0018 -
val_loss: 0.0014
Epoch 36/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0016 -
val_loss: 0.0015
Epoch 37/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0018 -
val_loss: 0.0011
Epoch 38/100
3/3 [==============================] - 0s 38ms/step - loss: 0.0017 -
val_loss: 0.0010
Epoch 39/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0018 -
val_loss: 0.0011
Epoch 40/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0018 -
val_loss: 0.0011
Epoch 41/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0017 -
val_loss: 0.0011
Epoch 42/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0015 -
val_loss: 0.0010
```

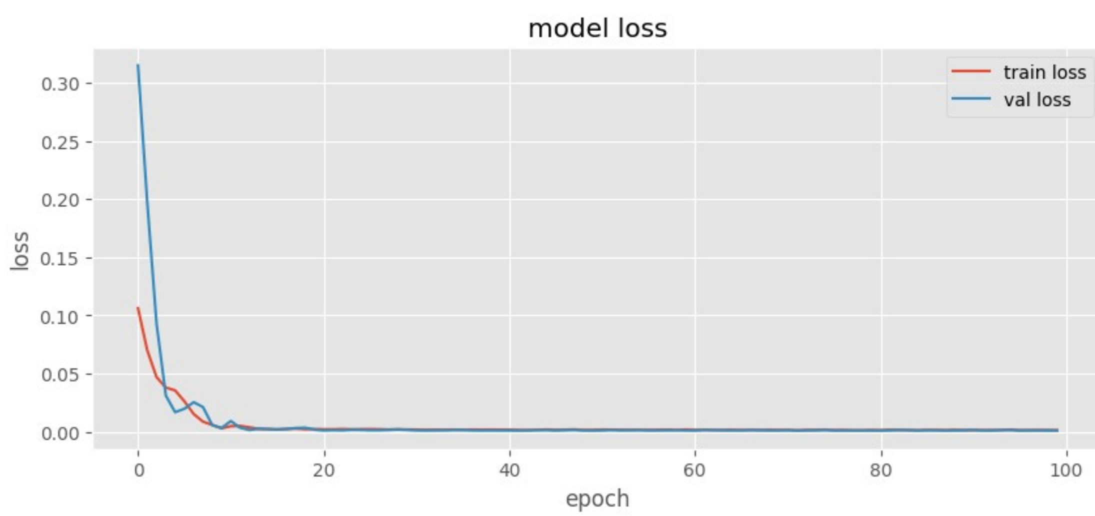
```
Epoch 43/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0014 -
val_loss: 0.0011
Epoch 44/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0015 -
val_loss: 0.0014
Epoch 45/100
3/3 [==============================] - 0s 39ms/step - loss: 0.0019 -
val_loss: 0.0013
Epoch 46/100
3/3 [==============================] - 0s 41ms/step - loss: 0.0016 -
val_loss: 0.0010
Epoch 47/100
3/3 [==============================] - 0s 40ms/step - loss: 0.0018 -
val_loss: 0.0013
Epoch 48/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0020 -
val_loss: 0.0015
Epoch 49/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0015 -
val_loss: 0.0011
Epoch 50/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0015 -
val_loss: 0.0010
Epoch 51/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0018 -
val_loss: 0.0011
Epoch 52/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0017 -
val_loss: 0.0014
Epoch 53/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0016 -
val_loss: 0.0012
Epoch 54/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0016 -
val_loss: 0.0012
Epoch 55/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0016 -
val_loss: 0.0014
Epoch 56/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0015 -
val_loss: 0.0011
Epoch 57/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0016 -
val_loss: 0.0011
Epoch 58/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0018 -
val_loss: 0.0012
Epoch 59/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0014 -
```

```
val_loss: 0.0013
Epoch 60/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0019 -
val_loss: 0.0011
Epoch 61/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0016 -
val_loss: 0.0010
Epoch 62/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0016 -
val_loss: 0.0014
Epoch 63/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0013 -
val_loss: 0.0015
Epoch 64/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0015 -
val_loss: 0.0012
Epoch 65/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0017 -
val_loss: 0.0011
Epoch 66/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0014 -
val_loss: 0.0012
Epoch 67/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0015 -
val_loss: 0.0014
Epoch 68/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0013 -
val_loss: 0.0013
Epoch 69/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0016 -
val_loss: 0.0010
Epoch 70/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0014 -
val_loss: 0.0012
Epoch 71/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0015 -
val_loss: 0.0012
Epoch 72/100
3/3 [==============================] - 0s 33ms/step - loss: 0.0011 -
val_loss: 0.0011
Epoch 73/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0015 -
val_loss: 0.0010
Epoch 74/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0016 -
val_loss: 0.0011
Epoch 75/100
3/3 [==============================] - 0s 37ms/step - loss: 0.0016 -
val_loss: 0.0015
Epoch 76/100
```

```
3/3 [==============================] - 0s 30ms/step - loss: 0.0013 -
val_loss: 0.0011
Epoch 77/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0015 -
val_loss: 0.0010
Epoch 78/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0012 -
val_loss: 0.0011
Epoch 79/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0012 -
val_loss: 0.0011
Epoch 80/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0015 -
val_loss: 0.0010
Epoch 81/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0012 -
val_loss: 0.0010
Epoch 82/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0017 -
val_loss: 0.0011
Epoch 83/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0014 -
val_loss: 0.0015
Epoch 84/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0014 -
val_loss: 0.0013
Epoch 85/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0012 -
val_loss: 0.0010
Epoch 86/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0014 -
val_loss: 0.0013
Epoch 87/100
3/3 [==============================] - 0s 36ms/step - loss: 0.0015 -
val_loss: 0.0012
Epoch 88/100
3/3 [==============================] - 0s 34ms/step - loss: 0.0013 -
val_loss: 0.0010
Epoch 89/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0016 -
val_loss: 0.0010
Epoch 90/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0013 -
val_loss: 0.0012
Epoch 91/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0012 -
val_loss: 0.0016
Epoch 92/100
3/3 [==============================] - 0s 30ms/step - loss: 0.0013 -
val_loss: 0.0010
```

```
Epoch 93/100
3/3 [==============================] - 0s 32ms/step - loss: 0.0014 -
val_loss: 0.0010
Epoch 94/100
3/3 [==============================] - 0s 31ms/step - loss: 0.0014 -
val_loss: 0.0014
Epoch 95/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0018 -
val_loss: 0.0014
Epoch 96/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0013 -
val_loss: 0.0010
Epoch 97/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0013 -
val_loss: 0.0011
Epoch 98/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0014 -
val_loss: 0.0011
Epoch 99/100
3/3 [==============================] - 0s 29ms/step - loss: 0.0014 -
val_loss: 0.0010
Epoch 100/100
3/3 [==============================] - 0s 28ms/step - loss: 0.0013 -
val_loss: 0.0010
```

```python
fig = plt.figure(figsize=(10, 4))
plt.plot(pred.history['loss'], label='train loss')
plt.plot(pred.history['val_loss'], label='val loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()
```



```
learningrate_parameter = learning_rate
train_loss=pred.history['loss'][-1]
validation_loss=pred.history['val_loss'][-1]
learningrate_parameter=pd.DataFrame(data=[[learningrate_parameter,
train_loss, validation_loss]],
                                    columns=['Learning Rate',
'Training Loss', 'Validation Loss'])
learningrate_parameter.set_index('Learning Rate')
```

```
               Training Loss  Validation Loss
Learning Rate
0.0001              0.001301          0.00101
```

```
# Implementation model into data test
y_pred_test = regressorGRU.predict(x_test)

# Invert normalization min-max
y_pred_invert_norm = scaler.inverse_transform(y_pred_test)
```

```
7/7 [==============================] - 1s 3ms/step
```

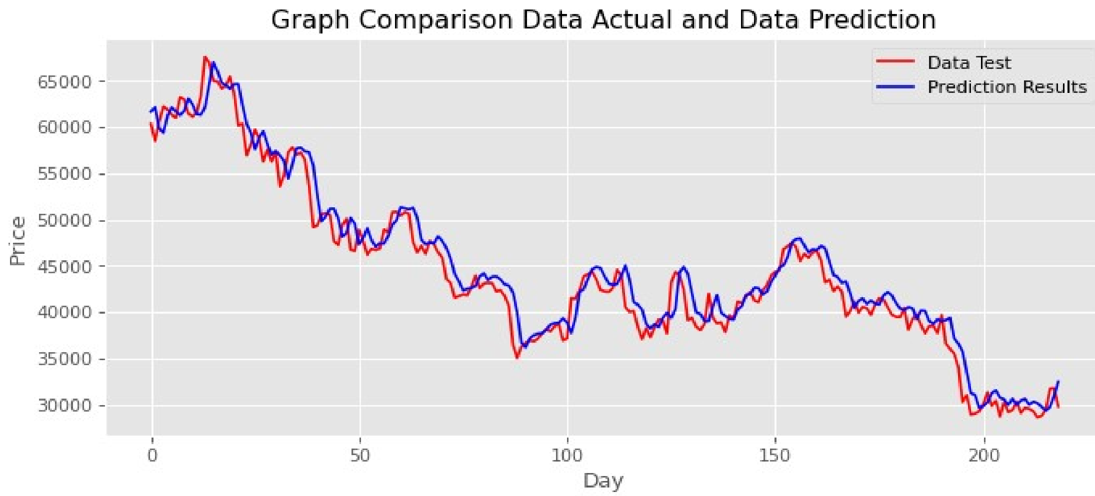
```
set_test = dataset["Close"]
```

```
# Comparison data test with data prediction
datacompare = pd.DataFrame()
datatest=np.array(set_test[totaldatatrain+totaldataval+lag:])
datapred= y_pred_invert_norm

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

```
        Data Test  Prediction Results
0    60363.792969        61678.617188
1    58482.386719        62133.730469
2    60622.136719        59867.835938
3    62227.964844        59379.949219
4    61888.832031        61227.941406
..            ...                 ...
214  28814.900391        29751.156250
215  29445.957031        29388.191406
216  31726.390625        29731.173828
217  31792.310547        30938.320312
218  29799.080078        32480.166016

[219 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)
```

```
23.614980114918378
```

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

```
1942.334182293659
```

```
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)
```

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

```
[0.03062108]
```