```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#loading the dataset
data = pd.read_csv("Amazon.csv")
data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1997-05-15	2.437500	2.500000	1.927083	1.958333	1.958333	72156000
1	1997-05-16	1.968750	1.979167	1.708333	1.729167	1.729167	14700000
2	1997-05-19	1.760417	1.770833	1.625000	1.708333	1.708333	6106800
3	1997-05-20	1.729167	1.750000	1.635417	1.635417	1.635417	5467200
4	1997-05-21	1.635417	1.645833	1.375000	1.427083	1.427083	18853200

```
# information about data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6297 entries, 0 to 6296
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       6297 non-null   object 
 1   Open       6297 non-null   float64
 2   High       6297 non-null   float64
 3   Low        6297 non-null   float64
 4   Close      6297 non-null   float64
 5   Adj Close  6297 non-null   float64
 6   Volume     6297 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 344.5+ KB
```

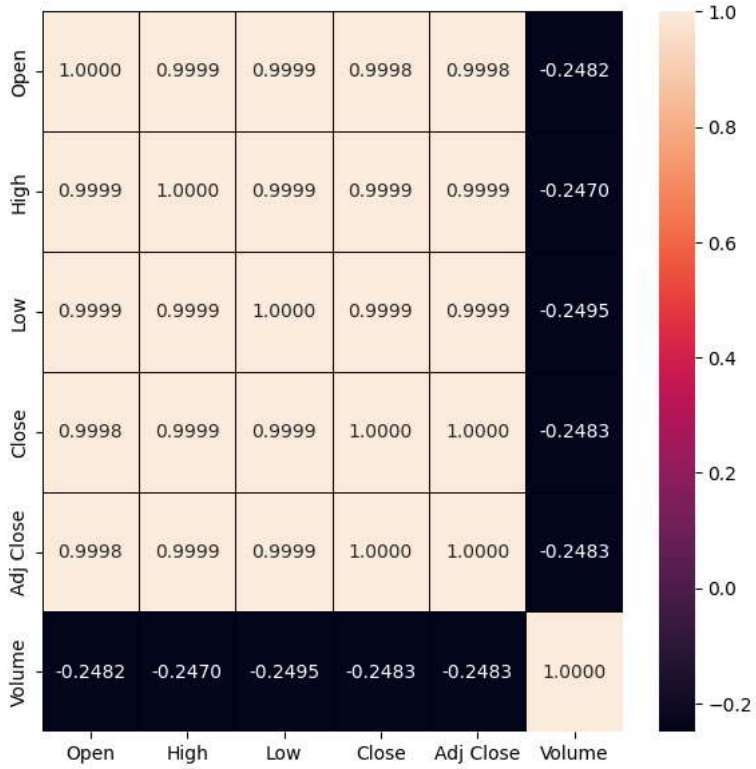
```
#Dataset Values
data.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	6297.000000	6297.000000	6297.000000	6297.000000	6297.000000	6.297000e+03
mean	579.165241	585.734350	571.812049	578.874775	578.874775	7.251173e+06
std	932.832876	943.056593	921.105051	931.982802	931.982802	7.092000e+06
min	1.406250	1.447917	1.312500	1.395833	1.395833	4.872000e+05
25%	39.130001	39.919998	38.590000	39.180000	39.180000	3.520200e+06
50%	117.370003	118.870003	115.519997	117.389999	117.389999	5.405400e+06
75%	626.169983	636.549988	620.799988	628.349976	628.349976	8.205600e+06
max	3744.000000	3773.080078	3696.790039	3731.409912	3731.409912	1.043292e+08

```
# Data Correlation
data.corr()
```

```
#Visualization of correlation
f, ax = plt.subplots(figsize = (7,7))
sns.heatmap(data.corr(), annot = True, linewidths=0.5, linecolor = "black", fmt = ".4f", ax = ax)
plt.show()
```

```
<ipython-input-6-50d2480b27e9>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fu
  sns.heatmap(data.corr(), annot = True, linewidths=0.5, linecolor = "black", fmt = ".4f", ax = ax)
```




```
# Dataset column names
data.columns
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

```
data_set = data.loc[:, ["Open"]].values
print(data_set)
print(data_set.shape)
```

```
[[2.43750000e+00]
 [1.96875000e+00]
 [1.76041700e+00]
 ...
 [2.22879004e+03]
 [2.12561011e+03]
 [2.19137012e+03]]
(6297, 1)
```

```
train = data_set[:len(data_set) - 50]
test = data_set[len(train):]
print(train.shape[0])
train.reshape(train.shape[0],1)
print(train)
print(train.shape)
print(test.shape)
```

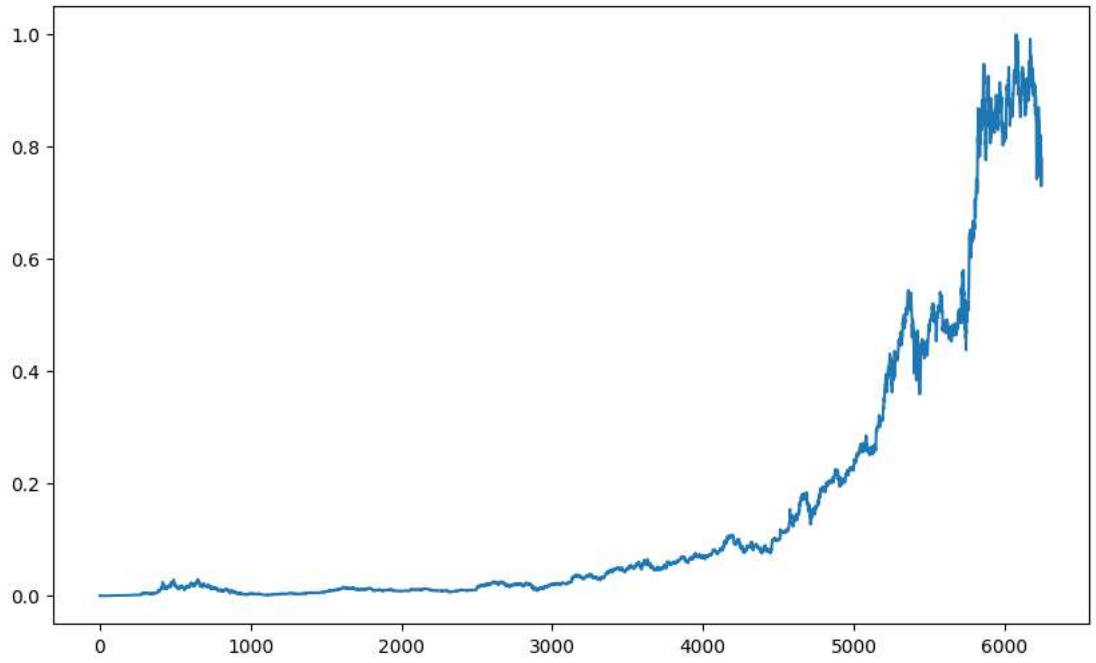
```
6247
[[2.43750000e+00]
 [1.96875000e+00]
 [1.76041700e+00]
 ...
 [2.73366992e+03]
 [2.79000000e+03]
 [2.91369995e+03]]
```

```
(6247, 1)
(50, 1)
```

```
# Normalization the data with MinmaxScaler from scikit learn library
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0,1))
train_scaler = scaler.fit_transform(train)
train_scaler
```

```
array([[2.75544200e-04],
       [1.50296836e-04],
       [9.46314304e-05],
       ...,
       [7.30045486e-01],
       [7.45096566e-01],
       [7.78148497e-01]])
```

```
plt.figure(figsize=(10, 6));
plt.plot(train_scaler)
plt.show()
```




```
X_train = []
Y_train = []
timesteps = 50

for i in range(timesteps, len(train_scaler)):
    X_train.append(train_scaler[i - timesteps:i, 0])
    Y_train.append(train_scaler[i,0])

X_train, Y_train = np.array(X_train), np.array(Y_train)

# reshaping the training data
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1]))
```

```
X_train.shape
```

```
(6197, 50)
```

```
df = pd.DataFrame(X_train)
df1 = pd.DataFrame(Y_train)
df
```

	0	1	2	3	4	5	6	7	8	9	...	40	41
0	0.000276	0.000150	0.000095	0.000086	0.000061	0.000008	0.000000	0.000028	0.000058	0.000036	...	0.000245	0.000195
1	0.000150	0.000095	0.000086	0.000061	0.000008	0.000000	0.000028	0.000058	0.000036	0.000025	...	0.000195	0.000278
2	0.000095	0.000086	0.000061	0.000008	0.000000	0.000028	0.000058	0.000036	0.000025	0.000028	...	0.000278	0.000250
3	0.000086	0.000061	0.000008	0.000000	0.000028	0.000058	0.000036	0.000025	0.000028	0.000033	...	0.000250	0.000209
4	0.000061	0.000008	0.000000	0.000028	0.000058	0.000036	0.000025	0.000028	0.000033	0.000019	...	0.000209	0.000206
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6192	0.896598	0.904184	0.910372	0.913627	0.909060	0.912574	0.906482	0.902506	0.894993	0.910426	...	0.844739	0.830596
6193	0.904184	0.910372	0.913627	0.909060	0.912574	0.906482	0.902506	0.894993	0.910426	0.891428	...	0.830596	0.803764
6194	0.910372	0.913627	0.909060	0.912574	0.906482	0.902506	0.894993	0.910426	0.891428	0.873085	...	0.803764	0.810027
6195	0.913627	0.909060	0.912574	0.906482	0.902506	0.894993	0.910426	0.891428	0.873085	0.875161	...	0.810027	0.746900
6196	0.909060	0.912574	0.906482	0.902506	0.894993	0.910426	0.891428	0.873085	0.875161	0.857775	...	0.746900	0.804147

```
df1
```

	0
0	0.000220
1	0.000256
2	0.000292
3	0.000276
4	0.000250
...	...
6192	0.786025
6193	0.776858
6194	0.730045
6195	0.745097
6196	0.778148

6197 rows × 1 columns

```
X_train.reshape(X_train.shape[0],X_train.shape[1],1)
print(X_train)

```

```
[[2.75544200e-04 1.50296836e-04 9.46314304e-05 ... 2.08745606e-04
  2.39361539e-04 2.17095430e-04]
 [1.50296836e-04 9.46314304e-05 8.62816062e-05 ... 2.39361539e-04
  2.17095430e-04 2.19878794e-04]
 [9.46314304e-05 8.62816062e-05 6.12321335e-05 ... 2.17095430e-04
  2.19878794e-04 2.56061188e-04]
 ...
 [9.10372334e-01 9.13626743e-01 9.09060368e-01 ... 8.20079292e-01
  7.86025382e-01 7.76857992e-01]
 [9.13626743e-01 9.09060368e-01 9.12574013e-01 ... 7.86025382e-01
  7.76857992e-01 7.30045486e-01]
 [9.09060368e-01 9.12574013e-01 9.06481969e-01 ... 7.76857992e-01
  7.30045486e-01 7.45096566e-01]]
```

```
print(Y_train)
```

```
[2.19878794e-04 2.56061188e-04 2.92243848e-04 ... 7.30045486e-01
 7.45096566e-01 7.78148497e-01]
```

```
# Import Library
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout

# Initialising the RNN
regressor = Sequential()

# Add the first RNN layer and some Dropout (regularisation)
```

```
regressor.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding the Second RNN layer and some Dropout (regularisation)
regressor.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True))
regressor.add(Dropout(0.2))

# Adding the Third RNN layer and some Dropout (regularisation)
regressor.add(SimpleRNN(units = 50, activation = "tanh", return_sequences = True))
regressor.add(Dropout(0.2))

# Adding the Fourth RNN layer and some Dropout (regularisation)
regressor.add(SimpleRNN(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = "adam", loss = "mean_squared_error")

# Fitting the RNN to the training set
regressor.fit(X_train, Y_train, epochs = 150, batch_size = 32)
```

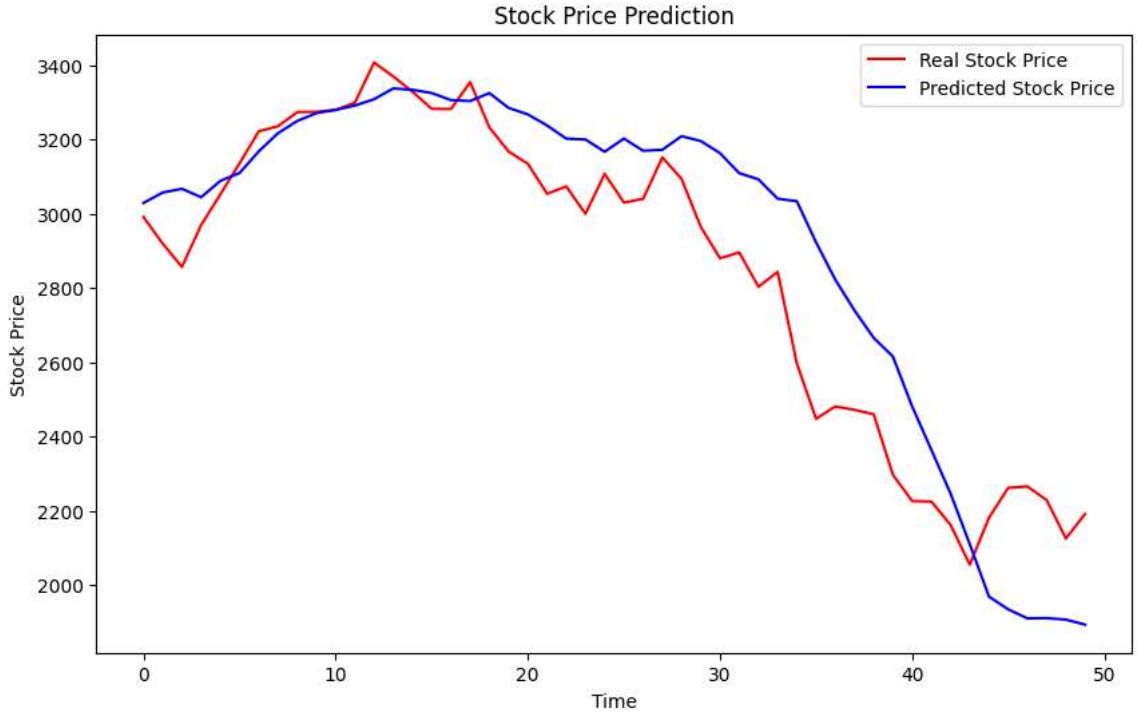
```
Epoch 1/150
194/194 [==============================] - 42s 185ms/step - loss: 0.0980
Epoch 2/150
194/194 [==============================] - 35s 181ms/step - loss: 0.0183
Epoch 3/150
194/194 [==============================] - 36s 185ms/step - loss: 0.0102
Epoch 4/150
194/194 [==============================] - 36s 186ms/step - loss: 0.0070
Epoch 5/150
194/194 [==============================] - 35s 183ms/step - loss: 0.0053
Epoch 6/150
194/194 [==============================] - 35s 182ms/step - loss: 0.0044
Epoch 7/150
194/194 [==============================] - 35s 183ms/step - loss: 0.0036
Epoch 8/150
194/194 [==============================] - 36s 185ms/step - loss: 0.0031
Epoch 9/150
194/194 [==============================] - 36s 183ms/step - loss: 0.0025
Epoch 10/150
194/194 [==============================] - 36s 186ms/step - loss: 0.0023
Epoch 11/150
194/194 [==============================] - 35s 181ms/step - loss: 0.0019
Epoch 12/150
194/194 [==============================] - 36s 185ms/step - loss: 0.0018
Epoch 13/150
194/194 [==============================] - 36s 183ms/step - loss: 0.0017
Epoch 14/150
194/194 [==============================] - 35s 180ms/step - loss: 0.0016
Epoch 15/150
194/194 [==============================] - 36s 183ms/step - loss: 0.0016
Epoch 16/150
194/194 [==============================] - 36s 186ms/step - loss: 0.0014
Epoch 17/150
194/194 [==============================] - 36s 185ms/step - loss: 0.0012
Epoch 18/150
194/194 [==============================] - 35s 182ms/step - loss: 0.0013
Epoch 19/150
194/194 [==============================] - 35s 179ms/step - loss: 0.0012
Epoch 20/150
194/194 [==============================] - 35s 179ms/step - loss: 0.0010
Epoch 21/150
194/194 [==============================] - 36s 186ms/step - loss: 0.0011
Epoch 22/150
194/194 [==============================] - 36s 185ms/step - loss: 9.6849e-04
Epoch 23/150
194/194 [==============================] - 36s 184ms/step - loss: 0.0010
Epoch 24/150
194/194 [==============================] - 36s 183ms/step - loss: 0.0010
Epoch 25/150
194/194 [==============================] - 36s 183ms/step - loss: 0.0011
Epoch 26/150
194/194 [==============================] - 34s 175ms/step - loss: 9.8643e-04
Epoch 27/150
194/194 [==============================] - 35s 180ms/step - loss: 0.0010
Epoch 28/150
194/194 [==============================] - 37s 189ms/step - loss: 8.8368e-04
Epoch 29/150
194/194 [==============================] - 35s 181ms/step - loss: 0.0010
```

```
inputs = data_set[len(data_set) - len(test) - timesteps:]
inputs = scaler.transform(inputs)
```

```
X_test = []
for i in range(timesteps, inputs.shape[0]):
    X_test.append(inputs[i - timesteps:i, 0])

X_test_rnn = np.array(X_test)
X_test_rnn = np.reshape(X_test_rnn, (X_test_rnn.shape[0], X_test_rnn.shape[1], 1))
predicted_stock_price = regressor.predict(X_test_rnn)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)
```

```
2/2 [==============================] - 1s 29ms/step
```

```
# visualising the results
plt.figure(figsize=(10, 6));
plt.plot(test, color = "red", label = "Real Stock Price")
plt.plot(predicted_stock_price, color = "blue", label = "Predicted Stock Price")
plt.title("Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```




```
trainX = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
```

```
# importing Library For LSTM Model
import math
from keras.models import Sequential
from keras.layers import Dense, LSTM, GRU
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
# Initialising the LSTM
model = Sequential()
model.add(LSTM(10, input_shape = (1, timesteps)))
model.add(Dense(1))
model.compile(loss = "mean_squared_error", optimizer = "adam")
model.fit(trainX, Y_train, epochs = 100, batch_size = 1)
```

```
Epoch 1/100
6197/6197 [==============================] - 23s 3ms/step - loss: 4.8055e-04
Epoch 2/100
```

```
6197/6197 [==============================] - 20s 3ms/step - loss: 2.7311e-04
Epoch 3/100
6197/6197 [==============================] - 21s 3ms/step - loss: 2.4399e-04
Epoch 4/100
6197/6197 [==============================] - 21s 3ms/step - loss: 2.3809e-04
Epoch 5/100
6197/6197 [==============================] - 21s 3ms/step - loss: 2.1596e-04
Epoch 6/100
6197/6197 [==============================] - 20s 3ms/step - loss: 2.1770e-04
Epoch 7/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.7896e-04
Epoch 8/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.7226e-04
Epoch 9/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.7111e-04
Epoch 10/100
6197/6197 [==============================] - 20s 3ms/step - loss: 1.7125e-04
Epoch 11/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.6905e-04
Epoch 12/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.5505e-04
Epoch 13/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.5813e-04
Epoch 14/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.5553e-04
Epoch 15/100
6197/6197 [==============================] - 20s 3ms/step - loss: 1.5297e-04
Epoch 16/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.4361e-04
Epoch 17/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.3526e-04
Epoch 18/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.3534e-04
Epoch 19/100
6197/6197 [==============================] - 20s 3ms/step - loss: 1.3024e-04
Epoch 20/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.4624e-04
Epoch 21/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.3516e-04
Epoch 22/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.2747e-04
Epoch 23/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.3384e-04
Epoch 24/100
6197/6197 [==============================] - 20s 3ms/step - loss: 1.1810e-04
Epoch 25/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.2548e-04
Epoch 26/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.2484e-04
Epoch 27/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.2421e-04
Epoch 28/100
6197/6197 [==============================] - 21s 3ms/step - loss: 1.3385e-04
Epoch 29/100
```

```
# Prepare test dataset
testX = np.array(X_test)
testX = testX.reshape(testX.shape[0], 1, testX.shape[1])

# Predict with testX
predict_lstm = model.predict(testX)
predict_lstm = scaler.inverse_transform(predict_lstm)
```

```
2/2 [==============================] - 0s 6ms/step
```

```
# visualising the results
plt.figure(figsize=(10, 6));
plt.plot(test, color = "red", label = "Real Stock Price")
plt.plot(predict_lstm, color = "blue", label = "Predicted Stock Price")
plt.title("Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```


```
# Initialising the GRU
model1 = Sequential()
model1.add(GRU(10,input_shape = (1, timesteps)))
model1.add(Dense(1))
model1.compile(loss = "mean_squared_error", optimizer = "adam")
model1.fit(trainX, Y_train, epochs = 100, batch_size = 1)
```

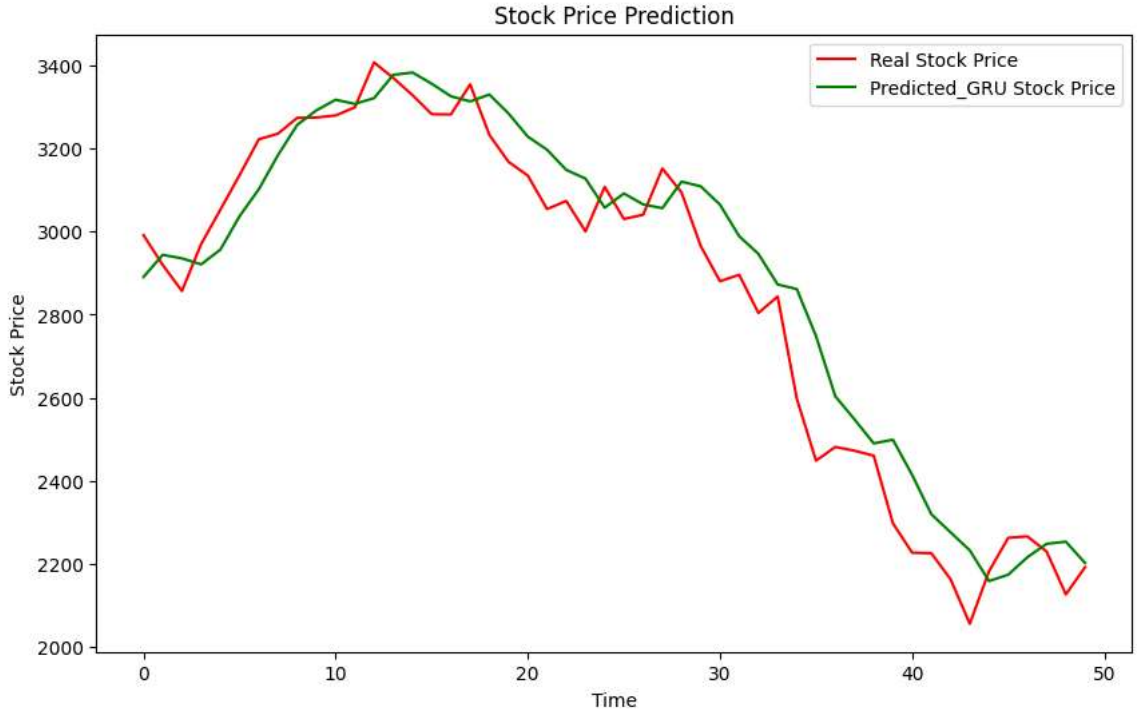
```
Epoch 1/100
6197/6197 [==============================] - 21s 3ms/step - loss: 0.0011
Epoch 2/100
6197/6197 [==============================] - 19s 3ms/step - loss: 3.7055e-04
Epoch 3/100
6197/6197 [==============================] - 19s 3ms/step - loss: 2.8057e-04
Epoch 4/100
6197/6197 [==============================] - 19s 3ms/step - loss: 2.6465e-04
Epoch 5/100
6197/6197 [==============================] - 19s 3ms/step - loss: 2.4861e-04
Epoch 6/100
6197/6197 [==============================] - 19s 3ms/step - loss: 2.0581e-04
Epoch 7/100
6197/6197 [==============================] - 19s 3ms/step - loss: 2.0134e-04
Epoch 8/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.9655e-04
Epoch 9/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.8176e-04
Epoch 10/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.7498e-04
Epoch 11/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.6409e-04
Epoch 12/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.6323e-04
Epoch 13/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.6340e-04
Epoch 14/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.5946e-04
Epoch 15/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.6822e-04
Epoch 16/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.4797e-04
Epoch 17/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.4665e-04
Epoch 18/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.4734e-04
Epoch 19/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.4093e-04
Epoch 20/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.4083e-04
Epoch 21/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.1848e-04
Epoch 22/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.2762e-04
Epoch 23/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.2373e-04
Epoch 24/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.2863e-04
Epoch 25/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.2344e-04
```

```
Epoch 26/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.2055e-04
Epoch 27/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.1551e-04
Epoch 28/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.1431e-04
Epoch 29/100
6197/6197 [==============================] - 19s 3ms/step - loss: 1.2260e-04
```

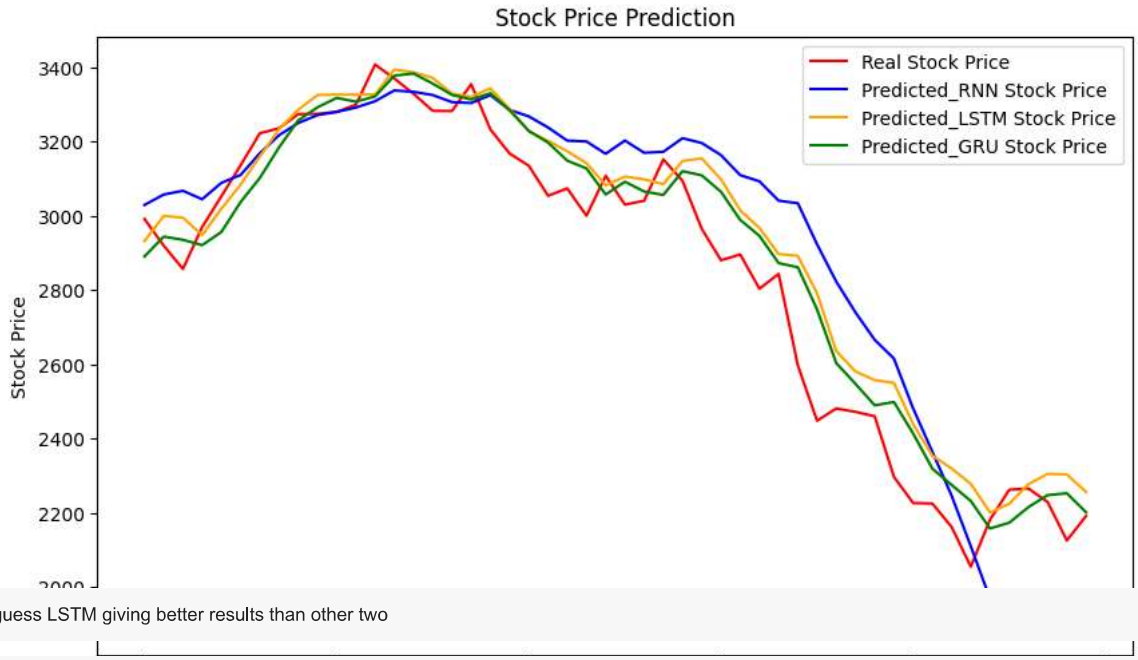
```
# Prepare test dataset
testX1 = np.array(X_test)
testX1 = testX1.reshape(testX1.shape[0], 1, testX1.shape[1])

# Predict with testX
predict_gru = model1.predict(testX1)
predict_gru = scaler.inverse_transform(predict_gru)
```

```
2/2 [==============================] - 0s 7ms/step
```

```
# visualising the results
plt.figure(figsize=(10, 6));
plt.plot(test, color = "red", label = "Real Stock Price")
plt.plot(predict_gru, color = "green", label = "Predicted_GRU Stock Price")
plt.title("Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```




```
# visualising the results
plt.figure(figsize=(10, 6));
plt.plot(test, color = "red", label = "Real Stock Price")
plt.plot(predicted_stock_price, color = "blue", label = "Predicted_RNN Stock Price")
plt.plot(predict_lstm, color = "orange", label = "Predicted_LSTM Stock Price")
plt.plot(predict_gru, color = "green", label = "Predicted_GRU Stock Price")
plt.title("Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```




```
# i guess LSTM giving better results than other two
```

```
from sklearn.metrics import accuracy_score,mean_squared_error,mean_absolute_error,r2_score
import numpy as np
mae = []
mse = []
rtwo = []
rmse = []
```

```
def scale(data):
    newdata = (data- np.min(data))/(np.max(data) - np.min(data))
    return newdata
```

```
newtest = scale(test)
newpredicted_stock_price = scale(predicted_stock_price)
newpredict_lstm = scale(predict_lstm)
newpredict_gru = scale(predict_gru)
```

```
def metri(actual,predicted):
    # r = (r2_score(actual,predicted)- np.min(predicted))/(np.max(predicted) - np.min(predicted))
    # me = (mean_absolute_error(actual,predicted) - np.min(predicted))/(np.max(predicted) - np.min(predicted))
    # me2 = (mean_squared_error(actual,predicted)- np.min(predicted))/(np.max(predicted) - np.min(predicted))
    # rme = (np.sqrt(mean_squared_error(actual,predicted))- np.min(predicted))/(np.max(predicted) - np.min(predicted))
    r = r2_score(actual,predicted)
    me = mean_absolute_error(actual,predicted)
    me2 = mean_squared_error(actual,predicted)
    rme = np.sqrt(mean_squared_error(actual,predicted))
    rtwo.append(r)
    mae.append(me)
    mse.append(me2)
    rmse.append(rme)
```

```
metri(newtest,newpredicted_stock_price)
metri(newtest,newpredict_lstm)
metri(newtest,newpredict_gru)
```

```
import plotly.graph_objects as go
model =['RNN', 'LSTM', 'GRU']

fig = go.Figure(data=[
    go.Bar(name='Mean Absolute error', x=model, y=mae),
    go.Bar(name='Mean Squared error', x=model, y=mse),
    go.Bar(name='Root Mean Squared error', x=model, y=rmse),
    go.Bar(name='R2_Score', x=model, y=rtwo)
])
# Change the bar mode
fig.update_layout(barmode='group',title = "Comparing RNN,LSTM,GRU using Regression Evalution Metrics ")
fig.show()
```


```
import pandas as pd
df = pd.DataFrame()
df['MAE'] = mae
df['MSE'] = mse
df['RMSE'] = rmse
df['R2'] = rtwo
```

```
df.to_csv("test_rnn_lstm_gru.csv")
```

```
import pandas as pd
test_rnn_lstm_gru = pd.DataFrame()
```

```
test_rnn_lstm_gru['test'] = test.tolist()
test_rnn_lstm_gru['RNN'] = predicted_stock_price
test_rnn_lstm_gru['LSTM'] = predict_lstm
test_rnn_lstm_gru['GRU'] = predict_gru
```

```
test_rnn_lstm_gru.to_csv("test_rnn_lstm_gru.csv")
```

```
#saving the model reduces the time for training the model again and again
RNNa_json = regressor.to_json()
with open("RNNa.json", "w") as json_file:
    json_file.write(RNNa_json)
regressor.save_weights("RNNa.h5")
print("model saved in disk")
```

```
 #loading the model
 #from keras.models import model_from_json

 #json_file = open('RNNa.json','r')
 #loaded_model_json = json_file.read()
 #json_file.close()

 #regressor1 = model_from_json(loaded_model_json)
 #regressor1.load_weights("RNNa.h5")
 #print("model loaded from disk")

```

```
lstma_json =model.to_json()
with open("lstma.json", "w") as json_file:
    json_file.write(lstma_json)
model.save_weights("lstma.h5")
print("lstm model saved in disk")
```

```
#loading the model
#from keras.models import model_from_json

#json_file = open('lstma.json','r')
#loaded_model_json = json_file.read()
#json_file.close()

#regressor1 = model_from_json(loaded_model_json)
#regressor1.load_weights("lstma.h5")
#print("model loaded from disk")

```

```
grua_json = model1.to_json()
with open("grua.json", "w") as json_file:
    json_file.write(grua_json)
model1.save_weights("grua.h5")
print("gru model saved in disk")
```

```
#loading the model
#from keras.models import model_from_json

#json_file = open('grua.json','r')
#loaded_model_json = json_file.read()
#json_file.close()

#regressor1 = model_from_json(loaded_model_json)
#regressor1.load_weights("grua.h5")
#print("model loaded from disk")
```

```

```

```

```