Importing Libraries

In [2]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from keras.layers import LSTM
from keras.layers import GRU
import tensorflow as tf
import math
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from sklearn.metrics import mean_squared_error
```

```
Using TensorFlow backend.
```

Define Some Functions

In [3]:
```
def plot_predictions(test,predicted):
    plt.figure(figsize = (10,6))
    plt.plot(test, color='red',label='Real Stock Price')
    plt.plot(predicted, color='blue',label='Predicted Stock Price')
    plt.title('Google Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('Google Stock Price')
    plt.legend()
    plt.show()
```

In [4]:
```
def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

Data reading

In [5]:
```
dataset = pd.read_csv('goog2.csv', index_col='Date', parse_dates=['Date'])
dataset.tail()
```

Out[5]:

	Close	High	Low	Open	Adj Close
Date					
2019-04-29	1287.579956	1289.270020	1266.295044	1274.000000	1287.579956
2019-04-30	1188.479980	1192.810059	1175.000000	1185.000000	1188.479980
2019-05-01	1168.079956	1188.050049	1167.180054	1188.050049	1168.079956
2019-05-02	1162.609985	1174.189941	1155.001953	1167.760010	1162.609985
2019-05-03	1185.400024	1186.800049	1169.000000	1173.650024	1185.400024

In [6]:
```
dataset.tail(3)
```
Out[6]:

Date	Close	High	Low	Open	Adj Close
2019-05-01	1168.079956	1188.050049	1167.180054	1188.050049	1168.079956
2019-05-02	1162.609985	1174.189941	1155.001953	1167.760010	1162.609985
2019-05-03	1185.400024	1186.800049	1169.000000	1173.650024	1185.400024

Define Training and Test Set

We want to train upto 2017's data to predict 2018 and 2019's stock movement

In [7]:
```
training_set = dataset[:'2017'].iloc[:,0:5]
test_set = dataset['2018':].iloc[:,0:1]
training_set.head(2)
```
Out[7]:

Date	Close	High	Low	Open	Adj Close
2005-05-05	112.75663	113.571327	112.210182	113.571327	112.75663
2005-05-06	113.27327	113.884293	112.503273	113.462036	113.27327

In [8]:
```
training_set.head(2)
```
Out[8]:

Date	Close	High	Low	Open	Adj Close
2005-05-05	112.75663	113.571327	112.210182	113.571327	112.75663
2005-05-06	113.27327	113.884293	112.503273	113.462036	113.27327

In [9]:
```
test_set.tail(2)
```
Out[9]:

Date	Close
2019-05-02	1162.609985
2019-05-03	1185.400024

In [10]:
```
len(training_set)
```
Out[10]: 3187

In [11]:
```
len(test_set)
```
Out[11]: 336

Plotting Stock movement with test and train data

In [12]:
```python
from matplotlib import pyplot as plt
plt.figure(figsize=(16,4))

plt.plot(training_set["Close"])
plt.plot(test_set["Close"])

plt.title('Google stock exchange stock price history')
plt.ylabel('Price (USD)')
plt.xlabel('Days')
plt.legend(['Train_set','Test_set'], loc='upper left')
plt.show()
```

Reshaping the multidimentional series array to 1d array

In [25]:
```python
train= training_set.values.reshape(-1, 1)
test = test_set.values.reshape(-1, 1)
```

In [26]:
```python
train.shape
```

Out[26]:
```
(15935, 1)
```

Normalization

In [27]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_sc = scaler.fit_transform(train)
test_sc = scaler.fit_transform(test)
```

Split Data set

In [28]:
```python
X_train = train_sc[:-1]
y_train = train_sc[1:]
X_test = test_sc[:-1]
y_test = test_sc[1:]
```

Moving Average

In [ ]:
```python

```

In [268]:
```
#make predictions
preds = []
for i in range(0,len(test_set)):
    a = training_set['Close'][len(training_set)-len(test_set)+i:].sum() + sum(pr
    b = a/248
    preds.append(b)
```

In [271]:
```
#calculate rmse
rms=np.sqrt(np.mean(np.power((np.array(test_set['Close'])-preds),2)))
rms
```

Out[271]:
```
566.4007490687118
```

In [273]:
```
#plot
test_set['Predictions'] = 0
test_set['Predictions'] = preds
plt.plot(training_set['Close'])
plt.plot(test_set[['Close', 'Predictions']])
```

Out[273]:
```
[<matplotlib.lines.Line2D at 0x1c5c1f5b38>,
 <matplotlib.lines.Line2D at 0x1c5c1f5550>]
```

ARIMA

In [13]:
```
from pmdarima import auto_arima
```

In [ ]:
```
_q=1,max_p=3, max_q=3, m=12,start_P=0, seasonal=True,d=1, D=1, trace=True,error_a
```

In [16]:
```
model.fit(training_set['Close'])

forecast = model.predict(n_periods=len(test_set))
forecast = pd.DataFrame(forecast,index = test_set.index,columns=['Prediction'])
```

In [21]:
```
rms=np.sqrt(np.mean(np.power((np.array(test_set['Close'])-np.array(forecast['Pre
rms
```

Out[21]:
```
90.4133997958666
```

In [24]:
```
#plot
plt.figure(figsize = (10,6))
plt.plot(training_set['Close'])
plt.plot(test_set['Close'])
plt.plot(forecast['Prediction'])
```

Recurrent Nueral Network Model

Long-Short Term Memory (LSTM)

Measuring Partial Autocorrelation

In [228]:
```
from statsmodels.tsa.stattools import pacf
```

In [229]:
```
dataset_pacf = pacf(dataset, nlags = 5, method = 'ols')
```

In [230]:
```
dataset_pacf
```

Out[230]:
```
array([1.0,
       array([1.00016535, 1.00773737, 0.99181986, 0.99991287, 1.00016535]),
       array([ 5.81860138e-02,  2.65145371e-01, -2.27261777e-01, -1.28103749e-
04,
        5.81860138e-02]),
       array([ 0.12716142, -0.18153613,  0.29826482,  0.00821393,  0.1271614
2]),
       array([-0.21303591, -0.25709012, -0.10919941, -0.13918812, -0.2130359
1]),
       array([0.35429788, 0.35608232, 0.46276931, 0.4521478 , 0.35429788])],
      dtype=object)
```

Making Time Series at Lag - 2

In [239]:
```
XL_train = []
yL_train = []
for i in range(2,3187+2):
    XL_train.append(train_sc[i-2:i,0])
    yL_train.append(train_sc[i,0])
XL_train, yL_train = np.array(XL_train), np.array(yL_train)
```

In [240]:
```
XL_train = np.reshape(XL_train, (XL_train.shape[0],XL_train.shape[1],1))
```

Applying LSTM

In [255]:
```
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=60, activation = 'tanh' , inner_activation = 'hard_sigm
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=60, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=60, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=60))

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

# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
```

```
/Users/san/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: User
Warning: Update your `LSTM` call to the Keras 2 API: `LSTM(units=60, activatio
n="tanh", return_sequences=True, input_shape=(2, 1), recurrent_activation="har
d_sigmoid")`
  after removing the cwd from sys.path.
```

In [256]:
```
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='loss', patience=50, verbose=1)
history = regressor.fit(XL_train, yL_train, epochs=100, batch_size=256, verbose=1
```

```
Epoch 91/100
3187/3187 [==============================] - 2s 582us/step - loss: 0.0021
Epoch 92/100
3187/3187 [==============================] - 2s 721us/step - loss: 0.0021
Epoch 93/100
3187/3187 [==============================] - 2s 525us/step - loss: 0.0022
Epoch 94/100
3187/3187 [==============================] - 1s 452us/step - loss: 0.0019
Epoch 95/100
3187/3187 [==============================] - 2s 585us/step - loss: 0.0022
Epoch 96/100
3187/3187 [==============================] - 2s 482us/step - loss: 0.0020
Epoch 97/100
3187/3187 [==============================] - 2s 524us/step - loss: 0.0022
Epoch 98/100
3187/3187 [==============================] - 2s 532us/step - loss: 0.0021
Epoch 99/100
3187/3187 [==============================] - 1s 456us/step - loss: 0.0020
Epoch 100/100
3187/3187 [==============================] - 2s 654us/step - loss: 0.0017
```

In [266]:
```
dataset_total = pd.concat((dataset["Close"][:'2018'],dataset["Close"]['2019':]),
inputs = dataset_total[len(dataset_total)-len(test_set) - 2:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)
```

In [267]:
```
XL_test = []
yL_test = []
for i in range(2,335+2): #lenght of y_test+2
    XL_test.append(inputs[i-2:i,0])
    yL_test.append(inputs[i,0])
XL_test = np.array(XL_test)
XL_test = np.reshape(XL_test, (XL_test.shape[0],XL_test.shape[1],1))


predicted_stock_price = regressor.predict(XL_test)
```

In [253]:
```
return_rmse(y_test,predicted_stock_price)
```

```
The root mean squared error is 0.8735664443653334.
```

In [254]:
```
plot_predictions(y_test,predicted_stock_price)
```

In [ ]:

Gated Recurrent Unit (GRU)

In [ ]:

In [257]:
```
;TM architecture
r = Sequential()
 GRU layer
r.add(GRU(units=60, activation = 'tanh' , inner_activation = 'hard_sigmoid',retu
l GRU layer
r.add(GRU(units=60, return_sequences=True))
 GRU layer
r.add(GRU(units=60, return_sequences=True))
 GRU layer
r.add(GRU(units=60))

tput layer
r.add(Dense(units=1))

ing the RNN
r.compile(optimizer='rmsprop',loss='mean_squared_error')
```

```
/Users/san/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: User
Warning: Update your `GRU` call to the Keras 2 API: `GRU(units=60, activation
="tanh", return_sequences=True, input_shape=(2, 1), recurrent_activation="hard
_sigmoid")`
  after removing the cwd from sys.path.
```

In [259]:
```
rom keras.callbacks import EarlyStopping

arly_stop = EarlyStopping(monitor='loss', patience=10, verbose=1)
istory = regressor.fit(XL_train, yL_train, epochs=250, batch_size=256, verbose=1,
```

```
Epoch 1/250
3187/3187 [==============================] - 1s 411us/step - loss: 0.0022
Epoch 2/250
3187/3187 [==============================] - 2s 607us/step - loss: 0.0019
Epoch 3/250
3187/3187 [==============================] - 2s 571us/step - loss: 0.0020
Epoch 4/250
3187/3187 [==============================] - 2s 712us/step - loss: 0.0019
Epoch 5/250
3187/3187 [==============================] - 2s 568us/step - loss: 0.0019
Epoch 6/250
3187/3187 [==============================] - 1s 468us/step - loss: 0.0020
Epoch 7/250
3187/3187 [==============================] - 1s 438us/step - loss: 0.0019
Epoch 8/250
3187/3187 [==============================] - 2s 511us/step - loss: 0.0017
Epoch 9/250
3187/3187 [==============================] - 2s 581us/step - loss: 0.0019
Epoch 10/250
3187/3187 [==============================] - 2s 474us/step - loss: 0.0020
Epoch 11/250
3187/3187 [==============================] - 1s 460us/step - loss: 0.0018
Epoch 12/250
3187/3187 [==============================] - 2s 510us/step - loss: 0.0019
Epoch 13/250
3187/3187 [==============================] - 1s 455us/step - loss: 0.0018
Epoch 14/250
3187/3187 [==============================] - 1s 462us/step - loss: 0.0018
Epoch 15/250
3187/3187 [==============================] - 2s 475us/step - loss: 0.0018
Epoch 16/250
3187/3187 [==============================] - 1s 345us/step - loss: 0.0018
Epoch 17/250
3187/3187 [==============================] - 2s 485us/step - loss: 0.0018
Epoch 18/250
3187/3187 [==============================] - 1s 363us/step - loss: 0.0017
Epoch 19/250
3187/3187 [==============================] - 2s 477us/step - loss: 0.0016
Epoch 20/250
3187/3187 [==============================] - 1s 387us/step - loss: 0.0020
Epoch 21/250
3187/3187 [==============================] - 2s 547us/step - loss: 0.0018
Epoch 22/250
3187/3187 [==============================] - 2s 509us/step - loss: 0.0017
Epoch 23/250
3187/3187 [==============================] - 2s 520us/step - loss: 0.0017
Epoch 24/250
3187/3187 [==============================] - 1s 391us/step - loss: 0.0016
Epoch 25/250
3187/3187 [==============================] - 2s 479us/step - loss: 0.0016
Epoch 26/250
3187/3187 [==============================] - 1s 376us/step - loss: 0.0017
Epoch 27/250
3187/3187 [==============================] - 2s 598us/step - loss: 0.0016
```

```
Epoch 28/250
3187/3187 [==============================] - 3s 844us/step - loss: 0.0017
Epoch 29/250
3187/3187 [==============================] - 4s 1ms/step - loss: 0.0017
Epoch 30/250
3187/3187 [==============================] - 5s 2ms/step - loss: 0.0017
Epoch 31/250
3187/3187 [==============================] - 3s 1ms/step - loss: 0.0017
Epoch 32/250
3187/3187 [==============================] - 3s 966us/step - loss: 0.0017
Epoch 33/250
3187/3187 [==============================] - 2s 508us/step - loss: 0.0016
Epoch 34/250
3187/3187 [==============================] - 2s 517us/step - loss: 0.0016
Epoch 35/250
3187/3187 [==============================] - 3s 881us/step - loss: 0.0015
Epoch 36/250
3187/3187 [==============================] - 3s 845us/step - loss: 0.0016
Epoch 37/250
3187/3187 [==============================] - 3s 818us/step - loss: 0.0017
Epoch 38/250
3187/3187 [==============================] - 2s 728us/step - loss: 0.0016
Epoch 39/250
3187/3187 [==============================] - 3s 1ms/step - loss: 0.0016
Epoch 40/250
3187/3187 [==============================] - 2s 502us/step - loss: 0.0017
Epoch 41/250
3187/3187 [==============================] - 2s 732us/step - loss: 0.0014
Epoch 42/250
3187/3187 [==============================] - 2s 634us/step - loss: 0.0014
Epoch 43/250
3187/3187 [==============================] - 2s 519us/step - loss: 0.0016
Epoch 44/250
3187/3187 [==============================] - 1s 373us/step - loss: 0.0015
Epoch 45/250
3187/3187 [==============================] - 1s 452us/step - loss: 0.0016
Epoch 46/250
3187/3187 [==============================] - 2s 479us/step - loss: 0.0016
Epoch 47/250
3187/3187 [==============================] - 1s 427us/step - loss: 0.0015
Epoch 48/250
3187/3187 [==============================] - 1s 453us/step - loss: 0.0015
Epoch 49/250
3187/3187 [==============================] - 1s 466us/step - loss: 0.0016
Epoch 50/250
3187/3187 [==============================] - 2s 584us/step - loss: 0.0014
Epoch 51/250
3187/3187 [==============================] - 2s 709us/step - loss: 0.0015
Epoch 00051: early stopping
```

In [260]:

```
dataset_total = pd.concat((dataset["Close"][:'2018'],dataset["Close"]['2019':]),
inputs = dataset_total[len(dataset_total)-len(test_set) - 2:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)
```

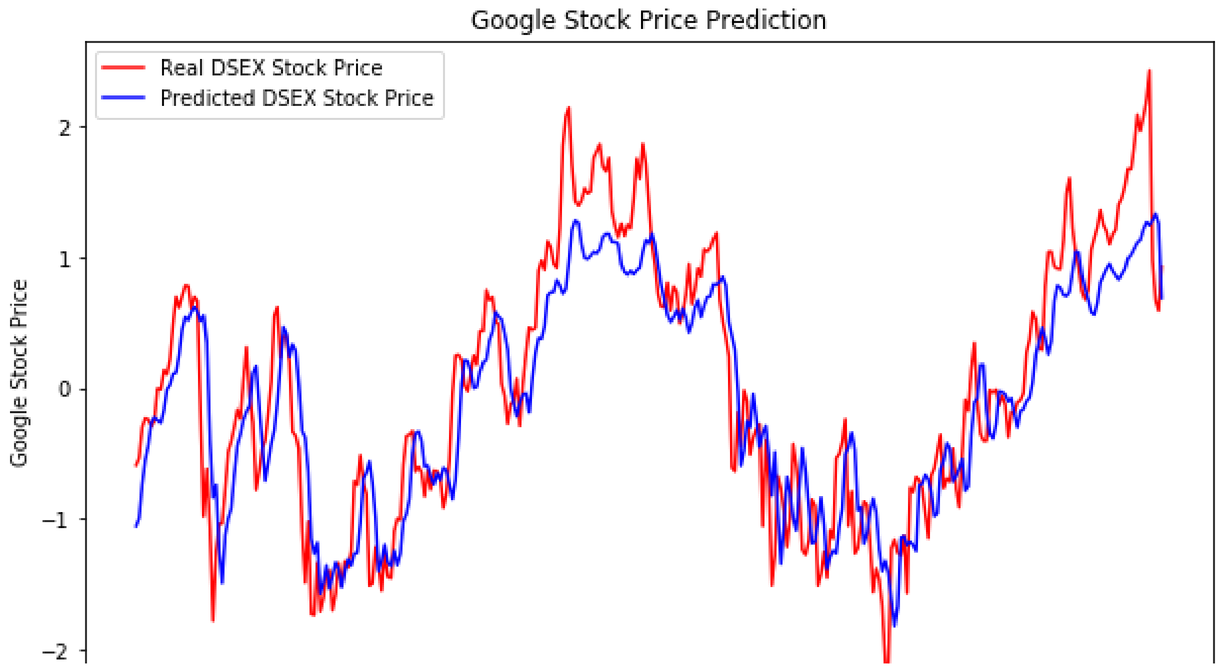
In [261]:
```
XL_test = []
yL_test = []
for i in range(2,335+2): #lenght of y_test+2
    XL_test.append(inputs[i-2:i,0])
    yL_test.append(inputs[i,0])
XL_test = np.array(XL_test)
XL_test = np.reshape(XL_test, (XL_test.shape[0],XL_test.shape[1],1))


predicted_stock_price = regressor.predict(XL_test)
```

In [262]:
```
return_rmse(y_test,predicted_stock_price)
```

```
The root mean squared error is 0.4543938977445383.
```

In [265]:
```
plot_predictions(y_test,predicted_stock_price)
```



In [ ]:

In [ ]:

Linear Regression

In [275]:
```
#implement linear regression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

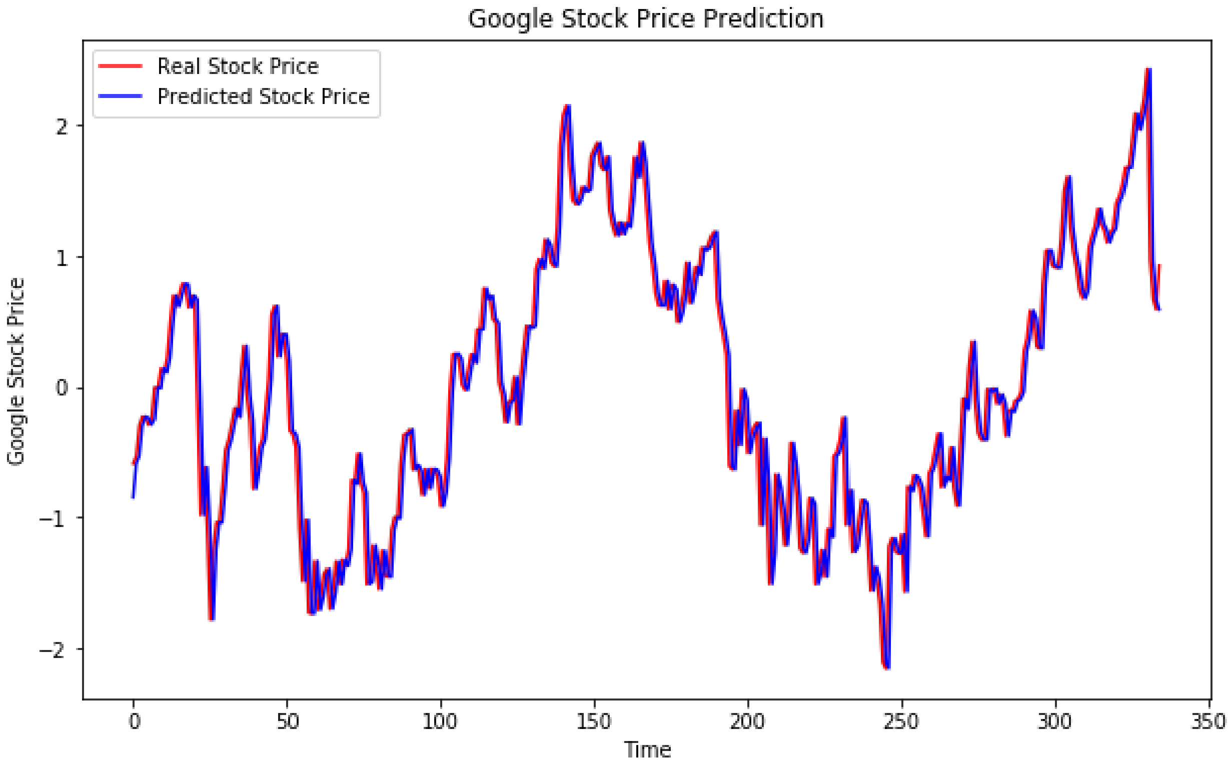
Out[275]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [276]:
```
#make predictions and find the rmse
preds = model.predict(X_test)
rms=np.sqrt(np.mean(np.power((np.array(y_test)-np.array(preds)),2)))
rms
```

Out[276]:
```
0.2807608194138677
```

In [ ]:

In [282]:
```
plot_predictions(y_test,preds)
```



In [283]:
```
y_testp = scaler.inverse_transform(y_test)
```

In [286]:
```
predsS = scaler.inverse_transform(preds)
```

In [59]:
```
return_rmse(y_testp,predsS)
```

```
The root mean squared error is 19.035010363852734.
```

In [69]:
```
def getAccuracy1(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if RMSD(testSet[x][-1], predictions[x]) < 50:
            correct += 1
    return (correct/float(len(testSet))) * 100.0


def RMSD(X, Y):
    return math.sqrt(pow(Y - X, 2))
```

In [70]:
```
getAccuracy1(y_testp,predsS)
```

Out[70]:
```
97.31343283582089
```

Applying ANN model

In [29]:
```
n_cols = X_train.shape[1]
```

```
In [30]: nn_model = Sequential()

         nn_model.add(Dense(12, activation='relu', input_shape=(n_cols,))) #Layer1
         nn_model.add(Dense(12, activation='sigmoid')) #Layer 2

         nn_model.add(Dense(1)) #Output Layer
         nn_model.compile(loss='mean_squared_error', optimizer='adam',metrics=['accuracy'
         early_stop = EarlyStopping(monitor='loss', patience=4, verbose=1)
         history = nn_model.fit(X_train, y_train, epochs=100, batch_size=128, verbose=1,
```

```
cc: 0.0000e+00
Epoch 35/100
15934/15934 [==============================] - 1s 37us/step - loss: 0.0040 - a
cc: 0.0000e+00
Epoch 36/100
15934/15934 [==============================] - 1s 41us/step - loss: 0.0038 - a
cc: 0.0000e+00
Epoch 37/100
15934/15934 [==============================] - 1s 43us/step - loss: 0.0035 - a
cc: 0.0000e+00
Epoch 38/100
15934/15934 [==============================] - 1s 53us/step - loss: 0.0033 - a
cc: 0.0000e+00
Epoch 39/100
15934/15934 [==============================] - 1s 46us/step - loss: 0.0031 - a
cc: 0.0000e+00
Epoch 40/100
15934/15934 [==============================] - 1s 40us/step - loss: 0.0028 - a
cc: 0.0000e+00
Epoch 41/100
```

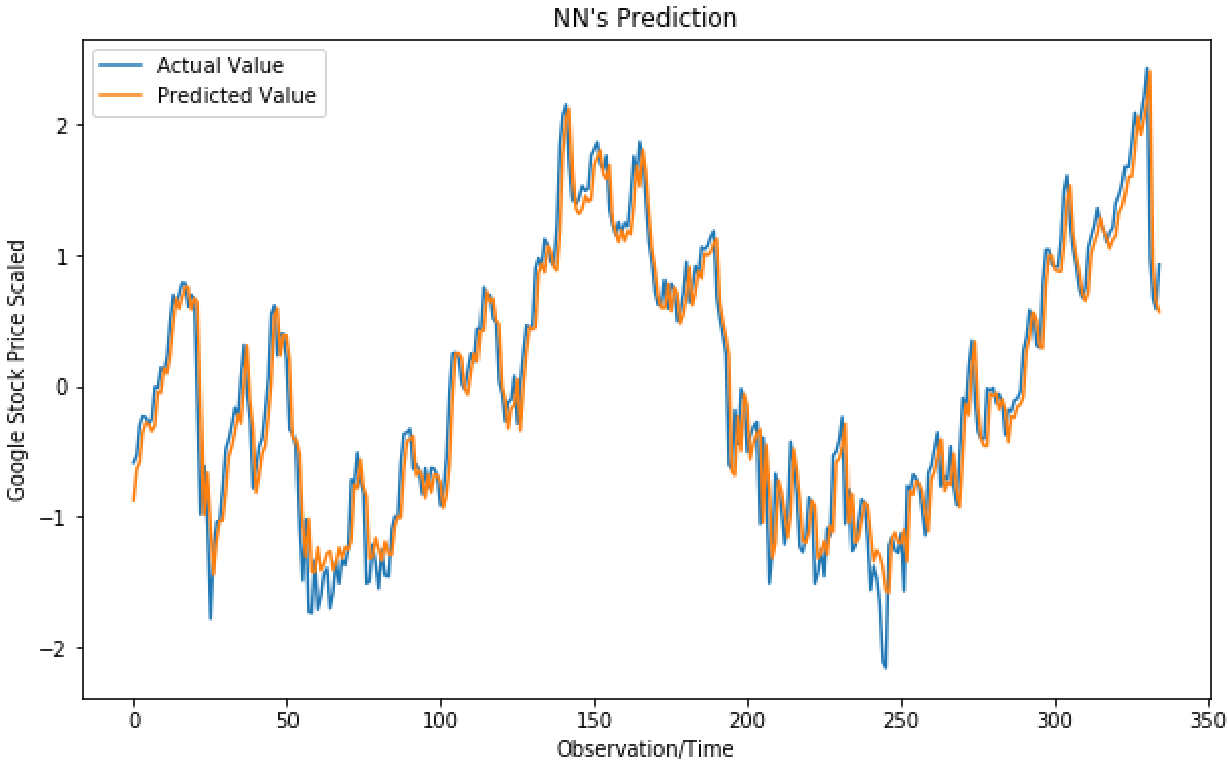
```
In [31]: nn_y_pred_test = nn_model.predict(X_test)
```

Calculating Root Mean Square and Plotting Actual vs Predicted

```
In [32]: return_rmse(y_test,nn_y_pred_test)
```

```
The root mean squared error is 0.2768267332918585.
```

In [33]:
```
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual Value')
plt.plot(nn_y_pred_test, label='Predicted Value')
plt.title("NN's Prediction")
plt.xlabel('Observation/Time')
plt.ylabel('Google Stock Price Scaled')
plt.legend()
plt.show();
```



In [ ]:

Return to Original Values

In [35]:
```
predicted_stock_price = scaler.inverse_transform(nn_y_pred_test)
```

In [36]:
```
y_testp = scaler.inverse_transform(y_test)
```

In [211]:
```
return_rmse(y_testp,predicted_stock_price)
```

```
The root mean squared error is 19.08415603702718.
```

k-Nearest Neighbors Algorithm (KNN)

In [46]:
```
#importing libraries
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
```

In [47]:
```
#using gridsearch to find the best parameter
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)
```

In [48]:
```
model
```

Out[48]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='m
inkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

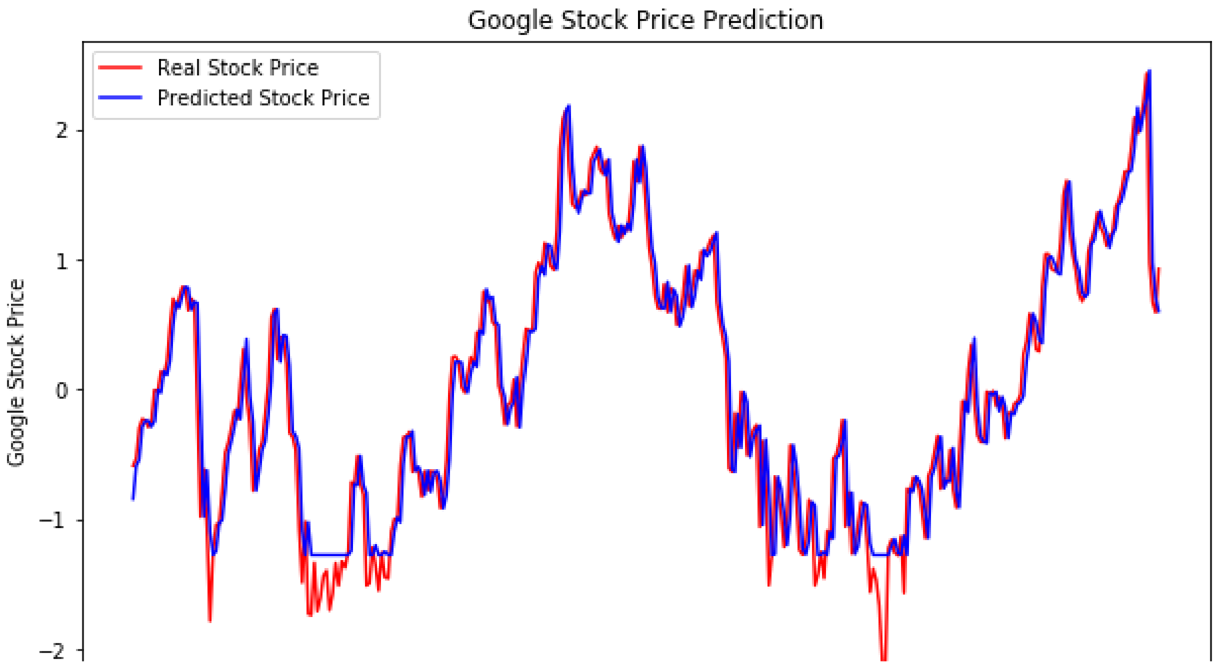
In [49]:
```
#fit the model and make predictions
model.fit(X_train,y_train)
predsk = model.predict(X_test)
```

```
/Users/san/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_sear
ch.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will chan
ge numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

In [51]:
```
rms=np.sqrt(np.mean(np.power((np.array(y_test)-np.array(predsk)),2)))
rms
```

Out[51]:
```
0.2801851383591317
```

In [52]:
```
plot_predictions(y_test,predsk)
```



In [53]:
```
y_testp = scaler.inverse_transform(y_test)
predsS = scaler.inverse_transform(predsk)
return_rmse(y_testp,predsS)
```

```
The root mean squared error is 19.035010363852734.
```

In [ ]: