In [ ]:
```
'''
Time Series Prediction with LSTM Recurrent Neural Networks in Python
Recurrent neural networks (RNN) are a class of neural networks that are helpfu
l in
modeling sequence data. Derived from feedforward networks, RNNs exhibit simila
r behavior to
how human brains function. Simply put: recurrent neural networks produce
predictive results in sequential data that other algorithms can't.
Recurrent Neural Networks or RNNs are a special type of neural network designe
d for sequence problems.
'''
```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

In [2]:
```
data_set = pd.read_csv(r'C:\Users\admin\Downloads\lmst\Foreign_Exchange_Rates.
csv', na_values='ND')
```

In [3]:
```
data_set.shape
```

Out[3]: (5217, 24)

In [4]:
```
data_set.head()
```

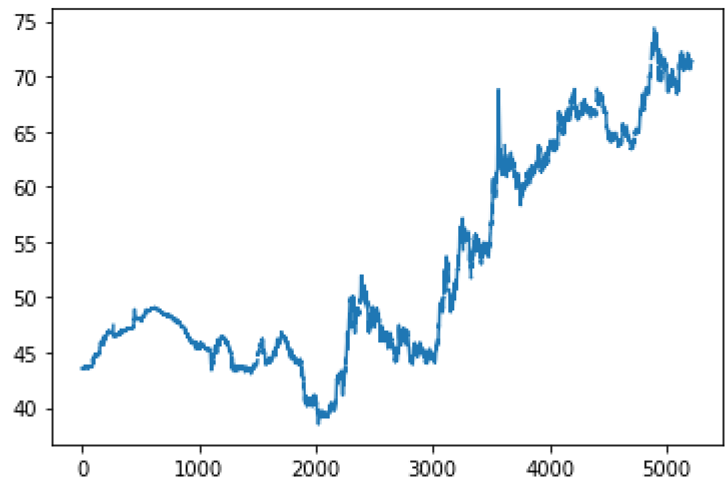
Out[4]:

	Unnamed: 0	Time Serie	AUSTRALIA - AUSTRALIAN DOLLAR/US$	EURO AREA - EURO/US$	NEW ZEALAND - NEW ZELAND DOLLAR/US$	UNITED KINGDOM - UNITED KINGDOM POUND/US$	BRAZIL - REAL/US$	CANAD CANADI DOLLAR/U
0	0	2000-01-03	1.5172	0.9847	1.9033	0.6146	1.8050	1.4
1	1	2000-01-04	1.5239	0.9700	1.9238	0.6109	1.8405	1.4
2	2	2000-01-05	1.5267	0.9676	1.9339	0.6092	1.8560	1.4
3	3	2000-01-06	1.5291	0.9686	1.9436	0.6070	1.8400	1.4
4	4	2000-01-07	1.5272	0.9714	1.9380	0.6104	1.8310	1.4

5 rows × 24 columns

In [5]:
```
plt.plot(data_set['INDIA - INDIAN RUPEE/US$'])
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1e5cea1c8d0>]
```



In [6]:
```
df = data_set['INDIA - INDIAN RUPEE/US$']
df
```

Out[6]:
```
0       43.55
1       43.55
2       43.55
3       43.55
4       43.55
        ...
5212      NaN
5213    71.28
5214    71.45
5215    71.30
5216    71.36
Name: INDIA - INDIAN RUPEE/US$, Length: 5217, dtype: float64
```

In [7]:
```
#Preprocessing data set
df = np.array(df).reshape(-1,1)
```

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

df = scaler.fit_transform(df)
print(df)
```

```
[[0.14142259]
 [0.14142259]
 [0.14142259]
 ...
 [0.91966527]
 [0.91548117]
 [0.91715481]]
```

In [9]:
```
#Training and test sets
train = df[:4800]
test = df[4800:]
```

In [10]:
```
print(train.shape)
print(test.shape)
```

```
(4800, 1)
(417, 1)
```

In [11]:
```
def get_data(data, look_back):
    data_x, data_y = [],[]
    for i in range(len(data)-look_back-1):
        data_x.append(data[i:(i+look_back),0])
        data_y.append(data[i+look_back,0])
    return np.array(data_x) , np.array(data_y)
```

In [12]:
```
look_back = 1
```

In [13]:
```
x_train , y_train = get_data(train, look_back)
```

In [14]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(4798, 1)
(4798,)
```

In [15]:
```
x_test , y_test = get_data(test,look_back)

print(x_test.shape)
print(y_test.shape)
```

```
(415, 1)
(415,)
```

In [16]:
```
#Processing train and test sets for LSTM model
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)
```

In [17]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(4798, 1, 1)
(415, 1, 1)
```

In [18]:
```
#Defining the LSTM model
n_features=x_train.shape[1]
model=Sequential()
model.add(LSTM(100,activation='relu',input_shape=(1,1)))
model.add(Dense(n_features))
```

In [27]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm (LSTM)                  (None, 100)               40800     
_________________________________________________________________
dense (Dense)                (None, 1)                 101       
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
model.compile(optimizer='adam', loss = 'mse')
```

In [29]:
```
model.fit(x_train,y_train, epochs = 5, batch_size=1)
```

```
Epoch 1/5
4798/4798 [==============================] - 17s 3ms/step - loss: nan
Epoch 2/5
4798/4798 [==============================] - 18s 4ms/step - loss: nan
Epoch 3/5
4798/4798 [==============================] - 18s 4ms/step - loss: nan
Epoch 4/5
4798/4798 [==============================] - 18s 4ms/step - loss: nan
Epoch 5/5
4798/4798 [==============================] - 18s 4ms/step - loss: nan
```

Out[29]:
```
<tensorflow.python.keras.callbacks.History at 0x1e5d591e710>
```

In [25]:
```
#Prediction using the trained model
scaler.scale_
```

Out[25]:
```
array([0.027894])
```

In [32]:
```
#Prediction using the trained model
scaler.scale_
y_pred = model.predict(x_test)
y_pred = scaler.inverse_transform(y_pred)
print(y_pred[:10])
```

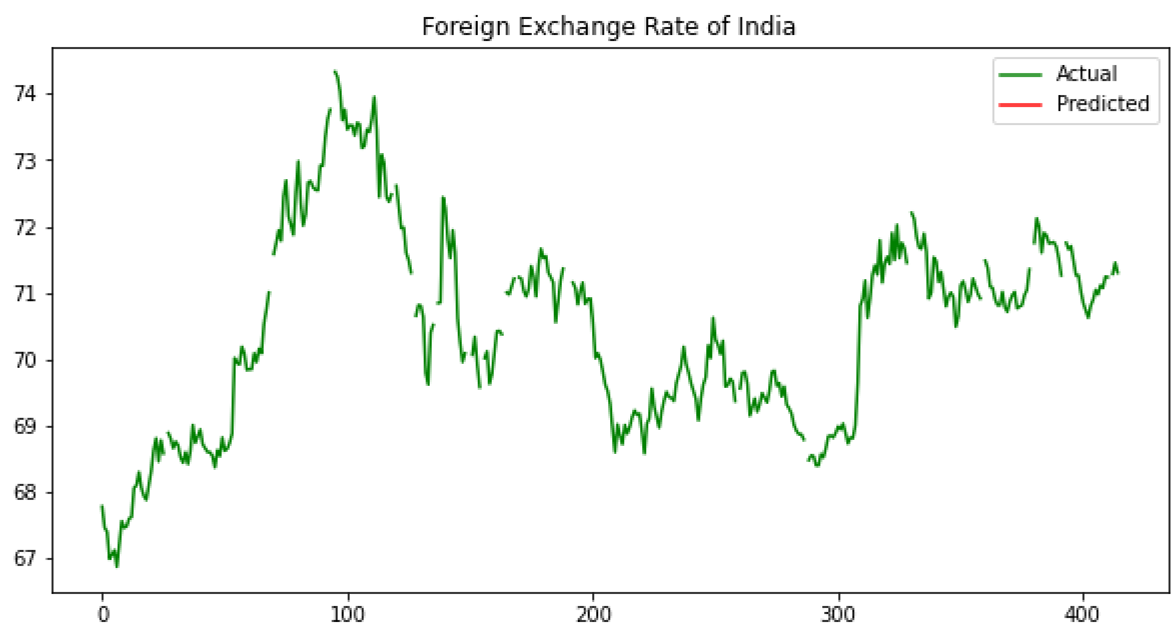
```
[[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]
```

In [30]:
```
y_test = np.array(y_test).reshape(-1,1)
y_test = scaler.inverse_transform(y_test)
print(y_test[:10])
```

```
[[67.78]
 [67.46]
 [67.4 ]
 [66.99]
 [67.05]
 [67.12]
 [66.87]
 [67.2 ]
 [67.56]
 [67.46]]
```

In [31]:
```
plt.figure(figsize=(10,5))
plt.title('Foreign Exchange Rate of India')
plt.plot(y_test , label = 'Actual', color = 'g')
plt.plot(y_pred , label = 'Predicted', color = 'r')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x1e5d6af4d68>



In [ ]: