Stock Price Prediction Using LSTM

Introduction:

In this task, the future stock of local shop of Bangladesh are predicted using the **LSTM** Recurrent Neural Network. Our task is to predict stock prices, sale prices and sale quantity for a few days, which is a time series problem. The **LSTM** model is very popular in time-series forecasting, and this is the reason why this model is chosen in this task. The historical prices of local shops are collected manually. We have used 8 years of historical price data, from 27.06.2015 to 01.02.2022.

This data set contains 87026 observations with 18 attributes. After preprocessing, only Date, Amount, Rate and Qty columns, a total of 4 columns, are taken as these columns have main significance in the dataset. The **LSTM** model is trained on this entire dataset. The stock prices for this new duration will be predicted by the already trained **LSTM** model, and the predicted prices will be plotted against the original prices to visualize the model's accuracy.

Long short-term memory (LSTM): LSTM is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points, but also entire sequences of data (such as speech or video inputs). LSTM models are able to store information over a period of time.

Modules: Keras, Tensorflow, Pandas, Scikit-Learn & Numpy.

Work target:

Implementation:

1. Let's import the required Libraries:

```
# import libaray
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math

from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM,Dropout
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.metrics import mean_squared_error
from numpy import array
import seaborn as sns
```

2. Load the data:

```
itemdata = pd.read_excel("StockTables.xlsx", sheet_name='Item')
categorydata = pd.read_excel("StockTables.xlsx", sheet_name='Category')
companydata = pd.read_excel("StockTables.xlsx", sheet_name='Company')
masterdata= pd.read_excel("StockTables.xlsx", sheet_name='Master')
detailsdata= pd.read_excel("StockTables.xlsx", sheet_name='Details')
```

3. Preprocessing the data:

```
detailsdata.drop(['Id','Unnamed: 17','Unnamed: 18','Unnamed: 20','Unnamed: 21','Unnamed: 22','Unnamed: 23','Unnamed: 24','Unnamed
```

```
detailsdata = pd.concat([detailsdata, data1], axis=1)
detailsdata
```

	CompanyId	Type	Date	Year	TransactionNo	StoreFr	StoreTo	PartyId	ItemSl	ItemId	Qty	Rate	Amount	DiscountRate	DiscountAmo
0	101	BUY	2017-12-31	2017	20170001	NaN	10102.0	1.012030e+09	1	101030183	823.0	0.0	0.0	0.0	
1	101	BUY	2017-12-31	2017	20170001	NaN	10102.0	1.012030e+09	3	101030360	548.5	0.0	0.0	0.0	

4. Visualize the dataset:

Index	Date	Amount	Qty	Rate
0	2001-02-07	64000	190	1030
1	2010-05-01	8931	2	8931
2	2015-06-27	0	1	0
3	2015-09-02	35000	1	35000
4	2015-09-03	325000	1	325000
5	2015-09-09	2000	1	2000
6	2015-09-10	17000	1	17000
7	2015-09-13	80000	1	80000
8	2015-09-15	230000	1	230000
9	2015-09-16	17000	1	17000
10	2015-09-17	29000	41	17600

5. Normalize the data:

	0	1	2
3	-0.0821744	-0.244161	-0.235761
4	-0.0602975	-0.244161	-0.061162
5	-0.0846638	-0.244161	-0.255629
6	-0.0835322	-0.244161	-0.246598
7	-0.0787797	-0.244161	-0.208668
8	-0.067464	-0.244161	-0.118358
9	-0.0835322	-0.244161	-0.246598
10	-0.082627	-0.235686	-0.246237
11	-0.0692745	-0.243525	-0.136119
12	-0.0832682	-0.235686	-0.251535

6. Split data trainX and trainY:

	0	1	2
0	-0.0799867	-0.204116	-0.256213
1	-0.0841409	-0.243949	-0.251456
2	-0.0848147	-0.244161	-0.256833
3	-0.0821744	-0.244161	-0.235761
4	-0.0602975	-0.244161	-0.061162
5	-0.0846638	-0.244161	-0.255629
6	-0.0835322	-0.244161	-0.246598
7	-0.0787797	-0.244161	-0.208668

Axis: 0 Shape: (1953, 14, 3) Index: 5 Slicing: [0, :, :]

	0
0	-0.243949
1	-0.24289
2	-0.244161
3	-0.244161
4	-0.24289
5	-0.243314
6	-0.244161
7	-0.240135
8	-0.241618
9	-0.237593

7. Create LSTM model:

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_30 (LSTM)              (None, 14, 50)            10800

 lstm_31 (LSTM)              (None, 14, 50)            20200

 lstm_32 (LSTM)              (None, 50)                20200

 dense_10 (Dense)            (None, 1)                 51

=================================================================
Total params: 51,251
Trainable params: 51,251
Non-trainable params: 0
_________________________________________________________________
```

8. Compile and train the model:

```
Epoch 1/100
28/28 [==============================] - 6s 56ms/step - loss: 0.6033 - mean_absolute_error:
0.1847 - val_loss: 4.5164 - val_mean_absolute_error: 1.2659
Epoch 2/100
28/28 [==============================] - 1s 34ms/step - loss: 0.5940 - mean_absolute_error:
0.1286 - val_loss: 4.2054 - val_mean_absolute_error: 1.2114
Epoch 3/100
28/28 [==============================] - 1s 36ms/step - loss: 0.5882 - mean_absolute_error:
0.1331 - val_loss: 4.1558 - val_mean_absolute_error: 1.2078
Epoch 4/100
28/28 [==============================] - 1s 37ms/step - loss: 0.5847 - mean_absolute_error:
0.1266 - val_loss: 3.8799 - val_mean_absolute_error: 1.1616
Epoch 5/100
28/28 [==============================] - 1s 38ms/step - loss: 0.5790 - mean_absolute_error:
0.1282 - val_loss: 3.7562 - val_mean_absolute_error: 1.1440
Epoch 6/100
28/28 [==============================] - 1s 37ms/step - loss: 0.5795 - mean_absolute_error:
0.1381 - val_loss: 3.5590 - val_mean_absolute_error: 1.1170
Epoch 7/100
28/28 [==============================] - 1s 36ms/step - loss: 0.5769 - mean_absolute_error:
0.1342 - val_loss: 3.4413 - val_mean_absolute_error: 1.0939
Epoch 8/100
```

9. Prediction the data:

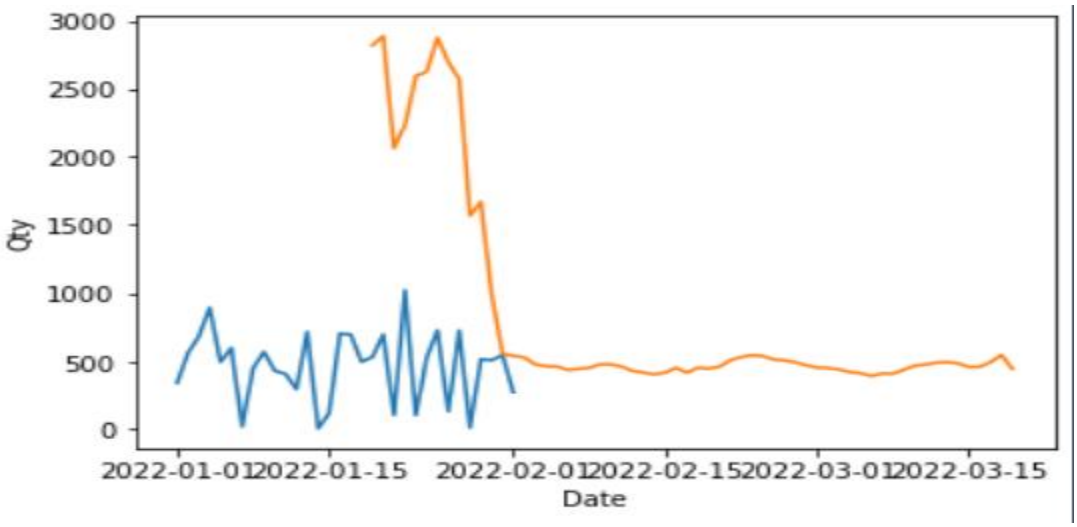
```
predict_period_dates = pd.date_range(list(train_dates)[-n_past], periods=n_days_fo
prediction = model.predict(trainX[-n_days_for_prediction:])


prediction_copies = np.repeat(prediction, df_for_training.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)[:,1]
```

Result:

1. Per day sale Qty prediction:

Index	Date	Qty
51	2022-03-11 00:00:00	466.06247
52	2022-03-12 00:00:00	473.33798
53	2022-03-13 00:00:00	491.85312
54	2022-03-14 00:00:00	595.98083
55	2022-03-15 00:00:00	488.0179
56	2022-03-16 00:00:00	644.47394
57	2022-03-17 00:00:00	773.4202
58	2022-03-18 00:00:00	867.201
59	2022-03-19 00:00:00	614.0149

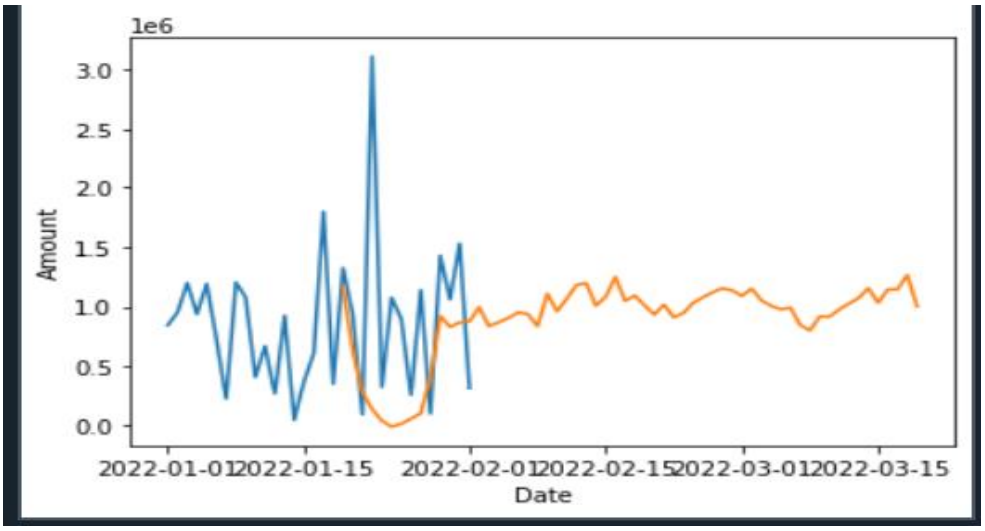
Visualize:



2. Per day amount prediction:

Index	Date	Amount
24	2022-02-12 00:00:00	1182875.6
25	2022-02-13 00:00:00	1197781.5
26	2022-02-14 00:00:00	1009063.1
27	2022-02-15 00:00:00	1083506.5
28	2022-02-16 00:00:00	1250007.6
29	2022-02-17 00:00:00	1049370.1
30	2022-02-18 00:00:00	1093608.0
31	2022-02-19 00:00:00	1011773.25
32	2022-02-20 00:00:00	934657.1

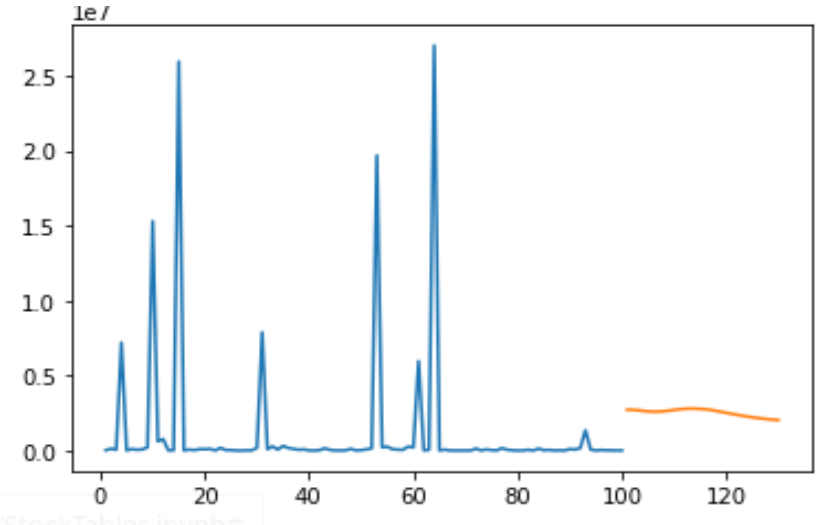
Visualize:



3. Per day investment prediction:

```
[[2200000.0],
 [32499.999999999996],
 [11000.0],
 [1500000.0],
 [367452.0],
 [2200.0],
 [21000.0],
 [54000.0],
 [5000.0],
 [11000.0],
 [11000.0],
 [108500.0],
 [11000.0],
 [81000.0],
 [3400.0],
 [13000.0],
 [29543824.999999996],
 [525000.0],
 [2197000.0],
```

Visualize:



4. Yearly investment prediction:

```
2022  orginal amount 2082036.07tk
2022 predict amount  [1732228.24232365] tk of Linear Rgression
2022 predict amount  [30488351.2012] tk of Random Forest Regression
2022 predict amount [81754030.94783925] tk of Support Vector Machines
2022 predict amount [4204152.] tk of Decision Tree Regression
```

5. Yearly profit prediction:

```
: print("2022  orginal amount 2.530280e+07tk ")
  print("2022 predict amount ", lr_predict ,"tk of Linear Rgression")
  print("2022 predict amount ", rnf_predict, "tk of Random Forest Regression")
  print("2022 predict amount", dr_predict, "tk of Decision Tree Regression")
```

```
2022  orginal amount 2.530280e+07tk
2022 predict amount  [91429103.27857208] tk of Linear Rgression
2022 predict amount  [65075194.09140001] tk of Random Forest Regression
2022 predict amount [71031486.38] tk of Decision Tree Regression
```