UNIVERSITY OF TECHNOLOGY

ASSIGNMENT REPORT

DISCRETE STRUCTURE

PHAN GIA BAO - 2153210

Class: CC02

November 2022

All by myself

Contents

1 Trend analysis

1.1 Group 1

I select three stocks in three different sectors (VNINDEX). They are ASG from transportation, ACC from construction and DSN from services sector. I analyse the movement of their daily closing price from 01/07/2021 to 31/12/2021. The data is collected using Vnquant package and the data visualization is done by Matplotlib.

```
import vnquant.data as dt
import matplotlib.pyplot as plt

company_list = [('ASG','red'),('ACC','green'),('DSN','blue')]
start = '2021-07-01'
end = '2021-12-31'

plt.figure(figsize=(16, 9))

for company,color in company_list:
    data = dt.DataLoader(company,start,end,minimal=True,data_source='vnd')
```

```python
    data = data.download()
    plt.plot(data['close'],label=company,color=color)
```

Figure 1: Data visualisation of group 1

The overall trend of them are a decrease followed by a steady increase toward the end of 2021. This is understandable because after being negatively impacted by COVID-19 in the middle of the year, the economy gradually recovered.

1.2 Group 2 - greatest growth rate

To select three indexes with the greatest growth rate (defined as $\frac{xf-xi}{x_i}$ where x_f, x_i is the closing price of that index in the final and initial day of the period, I write a small script as follow:

```python
# Library to scrawl data from excel
from openpyxl import load_workbook

wb = load_workbook(filename = 'HOSE_today.xlsx')
ws = wb.get_sheet_by_name('Sheet1')
column = ws['A']
vni_list = [column[x].value for x in range(len(column))]
```

The file HOSE_today.xlsx contains the data on HOSE in a single day. Next I compute the rate of return of all the symbols and sort them to get the greatest and smallest value.

```python
# Compute rate of return of all symbols
rate_of_return = dict()
for company in vni_list:
    try:
        data = dt.DataLoader(company,start,end,minimal=True,data_source='vnd')
        data = data.download()
```

```python
        rate_of_return[company] = data['close'].pct_change(periods = len(data.index) - 1).iloc[-1].values[0
        print(rate_of_return[company])
    except:
        rate_of_return[company] = float(0)
# Sort
sorted_rate_of_return = sorted(rate_of_return.items(), key=lambda x:x[1])
converted_dict = dict(sorted_rate_of_return)

#'FRT': 2.6370235934664246, 'DIG': 2.7263969171483624, 'VRC': 2.817733990147784 greatest growth
#'VPB': -0.49147727272727728, 'SHI': -0.3855185909980431, 'APH': -0.3649572649572655 greatest drop
```

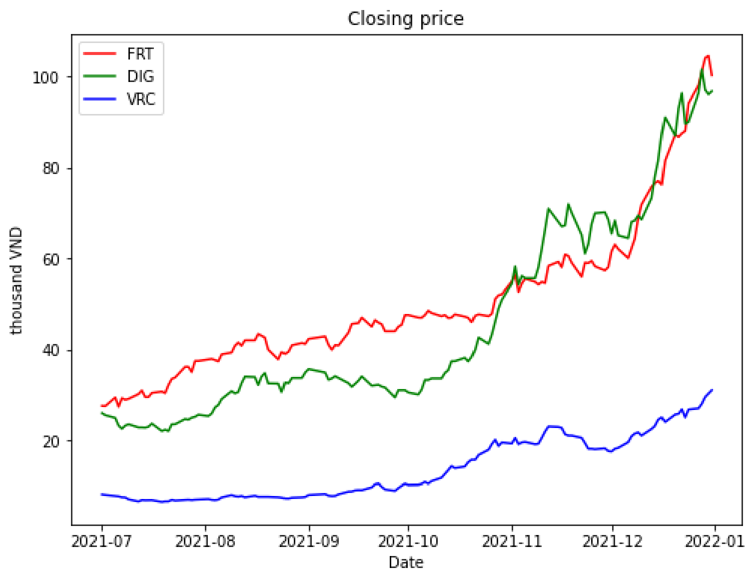


Figure 2: Data visualisation of group 2

1.3 Group 3 - greatest drop rate

Figure 3: Data visualisation of group 3

The random walk hypothesis states that stock market prices evolve according to a random walk (so price changes are random) and thus cannot be predicted.
However, for educational purpose, it is interesting to build a simple predictive model to get some hands-on experience about data science.

2 Predictive model

Since I have some experience with deep learning, I chose LSTM - Long short term memory to build the model. The reason was that it was possible to use LSTM for time series analysis.
The model was built with the following steps:

2.1 Data preprocessing

The data has already been collected by vnquant package.

```
#Load data by vnquant
import vnquant.data as dt

company = 'ASG'
start = '2012-01-01'
end = '2021-12-31'

data = dt.DataLoader(company,start,end,minimal=True,data_source='vnd')
data = data.download()
# Prepare data
```

```python
scaler = MinMaxScaler(feature_range = (0,1)) # Normalize the data
scaled_data = scaler.fit_transform(data['close'].values.reshape(-1,1))
# The next day is predicted from the previous 60 days
prediction_days = 60
x_train = []
y_train = []

for x in range(prediction_days, len(scaled_data)):
  x_train.append(scaled_data[x - prediction_days:x,0])
  y_train.append(scaled_data[x, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
```

2.2 Construct the model

The architecture of the model includes 2 Long short term memory (LSTM) layers and they consist of 128 and 64 hidden units respectively while dropout rate is set to 0.5.

```python
# Build the model
model = Sequential()

model.add(LSTM(units=128,return_sequences=True,input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=64))
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(optimizer='adam',loss='mean_absolute_error')

```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_14 (LSTM)              (None, 60, 128)           66560

 lstm_15 (LSTM)              (None, 64)                49408

 dropout_7 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 1)                 65

=================================================================
Total params: 116,033
Trainable params: 116,033
Non-trainable params: 0
```

Figure 4: Detail information of the model

2.3 Training and testing

2.3.1 Training

```python
save_model = "save_model.hdf5"
# Save the best_model when find out one
```

```python
best_model = tf.keras.callbacks.ModelCheckpoint(save_model,
                              monitor='loss',
                              verbose=2,
                              save_best_only=True,
                              mode='auto')
model.fit(x_train,y_train,epochs=100, batch_size = 32,verbose=2,callbacks=[best_model])
```

```
Epoch 97: loss did not improve from 0.03056
6/6 - 1s - loss: 0.0312 - 651ms/epoch - 109ms/step
Epoch 98/100

Epoch 98: loss did not improve from 0.03056
6/6 - 1s - loss: 0.0310 - 696ms/epoch - 116ms/step
Epoch 99/100

Epoch 99: loss improved from 0.03056 to 0.02906, saving model to save_model.hdf5
6/6 - 1s - loss: 0.0291 - 716ms/epoch - 119ms/step
Epoch 100/100

Epoch 100: loss did not improve from 0.02906
6/6 - 1s - loss: 0.0331 - 674ms/epoch - 112ms/step
```

Figure 5: Training process

2.3.2 Testing

```python
# Load test data
test_start = '2022-01-01'
test_end = '2022-03-31'
test_data = dt.DataLoader(company,test_start,test_end,minimal=True,data_source='vnd').download()

actual_prices = test_data['close'].values

total_dataset = pd.concat((data['close'], test_data['close']),axis=0)
model_inputs = total_dataset[
    len(total_dataset)-len(test_data)-prediction_days:].values
model_inputs = model_inputs.reshape(-1,1)
model_inputs = scaler.transform(model_inputs)

y_train = scaler.inverse_transform(y_train) #real data
final_model = tf.keras.models.load_model('save_model.hdf5')
train_predicted_prices = final_model.predict(x_train)
train_predicted_prices = scaler.inverse_transform(train_predicted_prices) #train predicted price

# Predict on Test Data
x_test = []
# Sliding window technique, use 60 past days to predict the next day
for x in range(prediction_days, len(model_inputs)):
    x_test.append(model_inputs[x-prediction_days:x,0])

x_test = np.array(x_test)
x_test = np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))

test_predicted_prices = model.predict(x_test) #predict 60 days in the test period
test_predicted_prices = scaler.inverse_transform(test_predicted_prices)
```

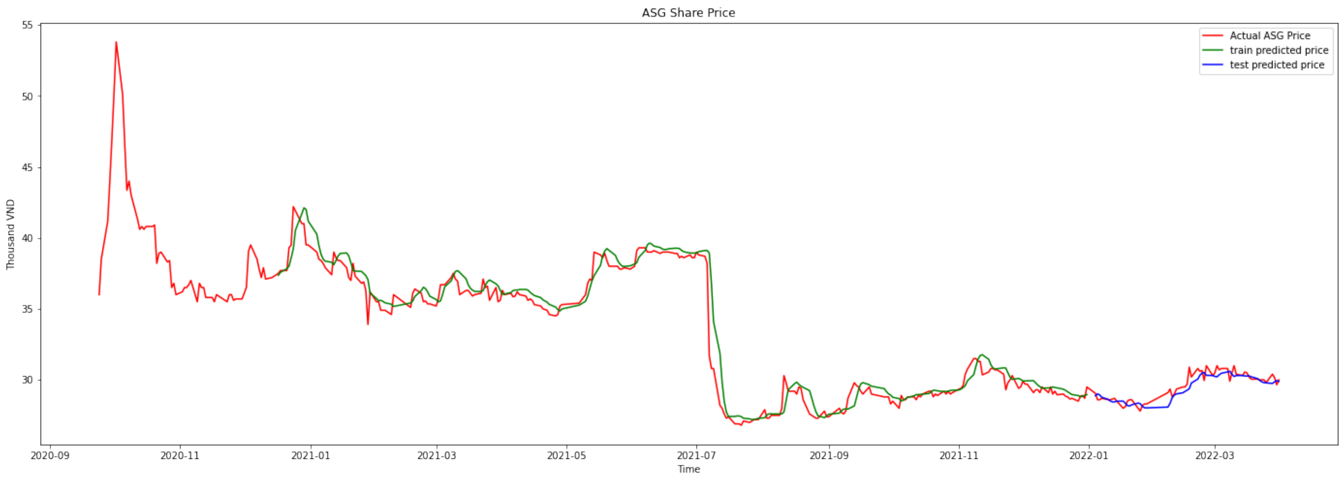


Figure 6: Visualization of the result

Apart from the visualisation, it is necessary to provide some statistical data to evaluate the performance of the model on this data set.

```
def _metric_measure(actual, predicted):
    mape = mean_absolute_percentage_error(actual,predicted)
    mae = mean_absolute_error(actual,predicted) * 1000
    print('Mean Absolute Percentage Error: {}'.format(mape))
    print('Mean Absolute Error: {}'.format(mae))

_metric_measure(y_train, train_predicted_prices,"train")
_metric_measure(test_data['close'].values,test_predicted_prices,"test")


--Model evaluation on train data--
Mean Absolute Percentage Error: 0.017670053924057592
Mean Absolute Error: 584.8572824515549


--Model evaluation on test data--
Mean Absolute Percentage Error: 0.011131483330843768
Mean Absolute Error: 330.9131819626381
```

That is the result obtained from data of ASG. For the remaining companies, I just feed the data in the model and show the result.

```
--Model evaluation on train data--
Mean Absolute Percentage Error: 0.01616987361647167
Mean Absolute Error: 359.3838194264964


--Model evaluation on test data--
Mean Absolute Percentage Error: 0.046761330294357924
Mean Absolute Error: 1054.9451762232286
```

Figure 8: Evaluation on ACC data

3 Result from other stock symbols

3.1 The remaining of group 1

3.1.1 ACC

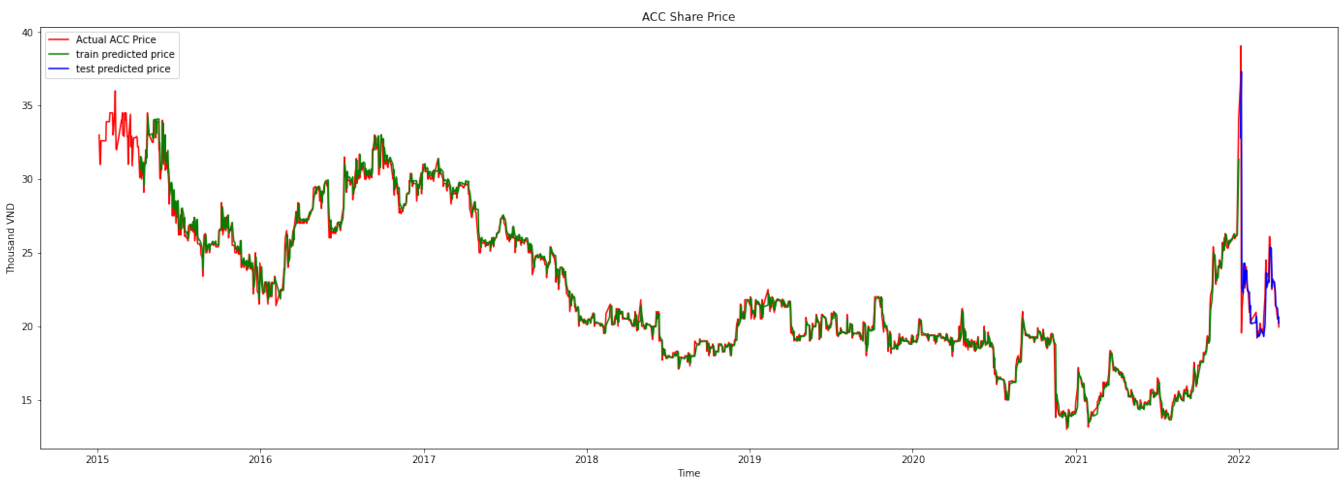


Figure 7: result on ACC data

3.1.2 DSN

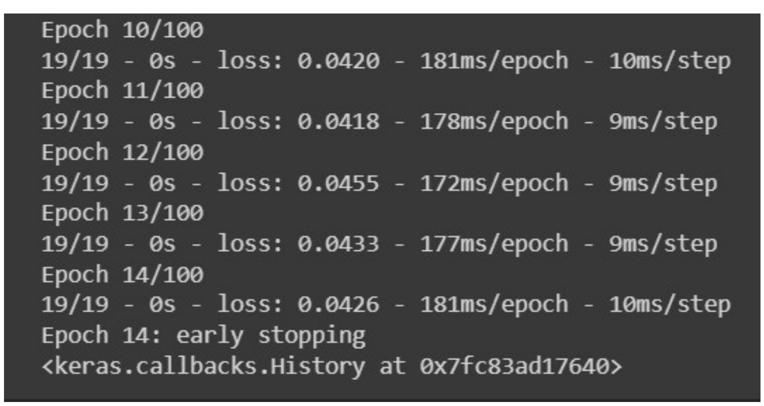
Figure 9: result on DSN data

3.2 Group 2

The result of group 1 is acceptable but takes so much time in comparison with the scope of the data. On training the model for the data of group 2, I make an adjustment to reduce training time. Rather than use **ModelCheckpoint** to select the best model when loss function no longer decreases, I employ Tensorflow **EarlyStopping** to terminate training when there is no progress.

```
best_model = tf.keras.callbacks.EarlyStopping(monitor = 'loss',
                                              patience=3,
                                              verbose=2
                                              )
```

3.2.1 FRT



```
Epoch 10/100
19/19 - 0s - loss: 0.0420 - 181ms/epoch - 10ms/step
Epoch 11/100
19/19 - 0s - loss: 0.0418 - 178ms/epoch - 9ms/step
Epoch 12/100
19/19 - 0s - loss: 0.0455 - 172ms/epoch - 9ms/step
Epoch 13/100
19/19 - 0s - loss: 0.0433 - 177ms/epoch - 9ms/step
Epoch 14/100
19/19 - 0s - loss: 0.0426 - 181ms/epoch - 10ms/step
Epoch 14: early stopping
<keras.callbacks.History at 0x7fc83ad17640>
```

Figure 10: The training terminates at epoch 14

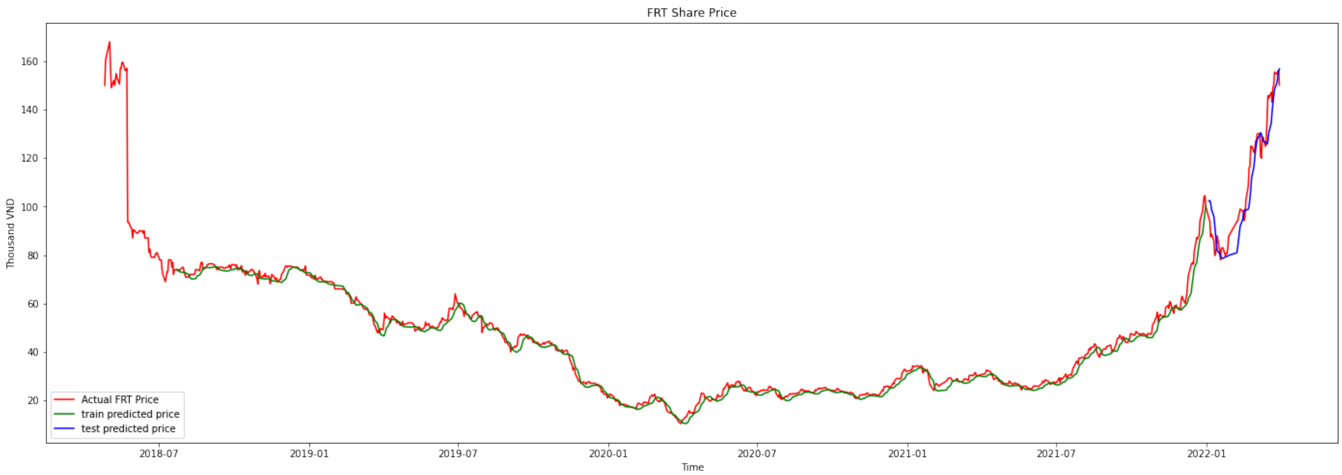


Figure 11: result on FRT data

3.2.2 DIG

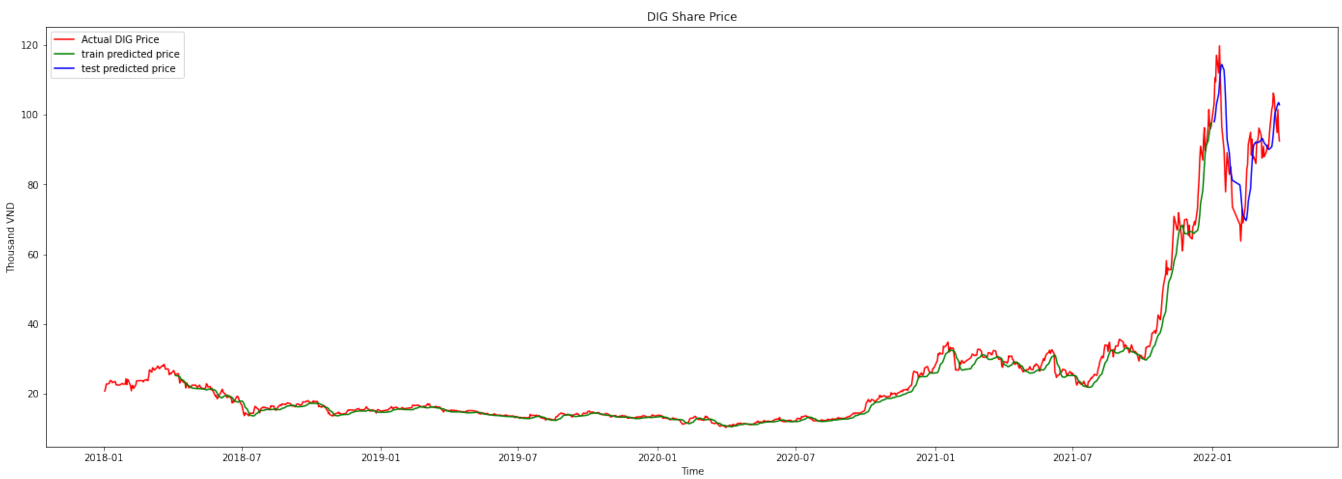


Figure 12: result on DIG data

3.2.3 VRC

Figure 13: result on VRC data

3.3 Group 3

3.3.1 VPB

Figure 14: result on VPB data

3.3.2 SHI

Figure 15: result on SHI data

3.3.3 APH

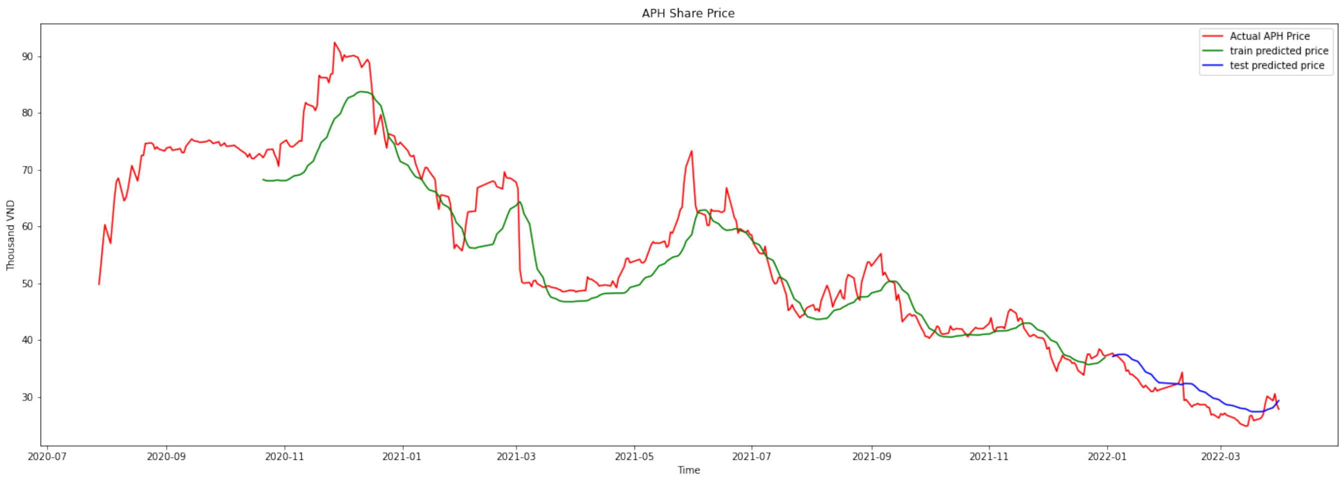


Figure 16: result on APH data

	Train data		Test data	
	MAPE	MAE	MAPE	MAE
ASG	0.0177	584.86	0.0111	330.91
ACC	0.0162	359.38	0.0468	1054.95
DSN	0.0225	1205.96	0.0197	966.75
FRT	0.0512	1895.66	0.0843	9654.11
DIG	0.043	1102.87	0.0835	7448.07
VRC	0.046	589.05	0.0784	2401.19
VPB	0.0365	1218.83	0.0209	756.18
SHI	0.0279	355.012	0.0412	684.02
APH	0.057	3410.88	0.0711	2066.97

Table 1: Summary of all the stock symbols

4 Are these results really close to reality as they seem?

Since stock market prices evolve according to a random walk, it is impossible to predict the changes of them. There are a huge number of factors that can affect stock prices.
Finance sectors are most likely to be negatively impacted by economic crisis. Indeed, VPB saw the greatest drop rate in their stock price within the second half of 2021. As of now, many financial companies are suffering from the economic crisis, where the most famous case recently is the scandal of Saigon Bank (SCB).

5 Conclusion

There is no point predicting stock price like this because the result is not useful in term of business potential. However, I learnt quite a number of interesting things from doing this project.
First, I know how to crawl data from the internet using API. In general, I learnt to build tools that fit my purpose when there is no ready-made tools that help. Next, I can get some hands-on experience about data science. I studied about ARIMA model and understood more about data analysis. I also reinforced my deep learning knowledge by building a model that actually works. There were hard times when stagnation occured and I did not know how to escape that stage. That was even worse when I chose to do everything by myself. But anyway, I did overcome all the difficulties.