VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ODD GROUP (A)

DISCRETE STRUCTURE FOR COMPUTING

ASSIGNMENT

VÕ TRUNG KIÊN (Leader)
LÊ VĂN TIẾN
NGUYỄN PHÚC TOÀN
PHẠM ĐỨC TRUNG

MEMBER CONTRIBUTION

Number	Name	Student's ID	Contribution
1.	Võ Trung Kiên(Leader)	2153502	25%
2.	Lê Văn Tiến	2153886	25%
3.	Nguyễn Phúc Toàn	2153902	25%
4.	Phạm Đức Trung	2153928	25%

CONTENT

1 INTRODUCTION

Vietnam accomplished 2021 with a 2.58% GDP growth rate, despite witnessing one of the harshest COVID lockdowns in the world during the second half of 2021. Nevertheless, our last two consecutive year still has a stable growth among any countries suffered from pandemic. 2022 as we can see so far is quite challenging for Viet Nam, the Ukraine wars, petrol price dramatically increased, the economic crisis is approaching, and state banks continuously increase interest rates. This leads to a significant impact on the stock market.

The stock price fluctuations might be vary, combining many unlinked reasons: umemployment rate, monetary policies, inflation, immigration, foreign trade...The hardest task is gathering a huge multifaceted information and know how and when is the right time to make a decision.

In this assignment, we will use Python to illustrate the movement of the stock indexes based on the efficiency, ease of learning and implementing. Explain some stocks if they were followed to any common distributions.

With the data collected from the Lightning Price Board and Investing.com, we will predict the trend of the stock market by RRN (Recurrent Neural Network) and LSTM (Long Short-term Memory), an algorithm used in Deep Learning.

By comparing with another machine learning algorithm Decision-tree, using the model's evaluation, we can able to conclude which models is the best and closest to reality.

2 Big Project

2.1 Question 1

2.1.1 Overview about VNIndex and HNX-Index

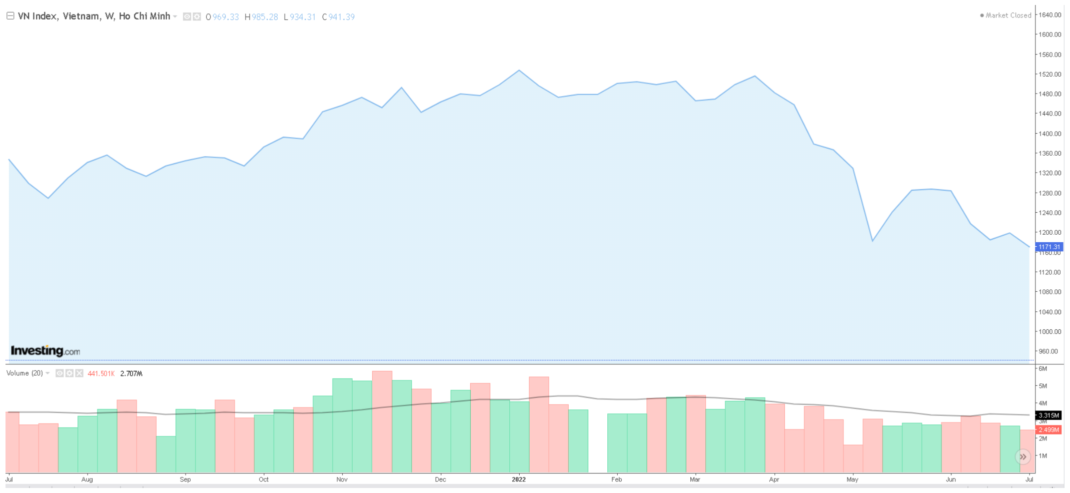
VNIndex is a market index showing the price movement of stocks currently traded on the Ho Chi Minh City Stock Exchange also known as HOSE. Based on this index, we can know the specific size and value of stocks at the present time, compared to the price based on the base date of July 28, 2000.

Similar to VNIndex, HNX-Index is also a market index which reflects the price movement of stocks on the Ha Noi Stock Exchange. HNX-Index was formerly known as HASTC-Index, because the Hanoi Stock Exchange used to be the Hanoi Stock Exchange Center (HASTC) before it was reorganized.

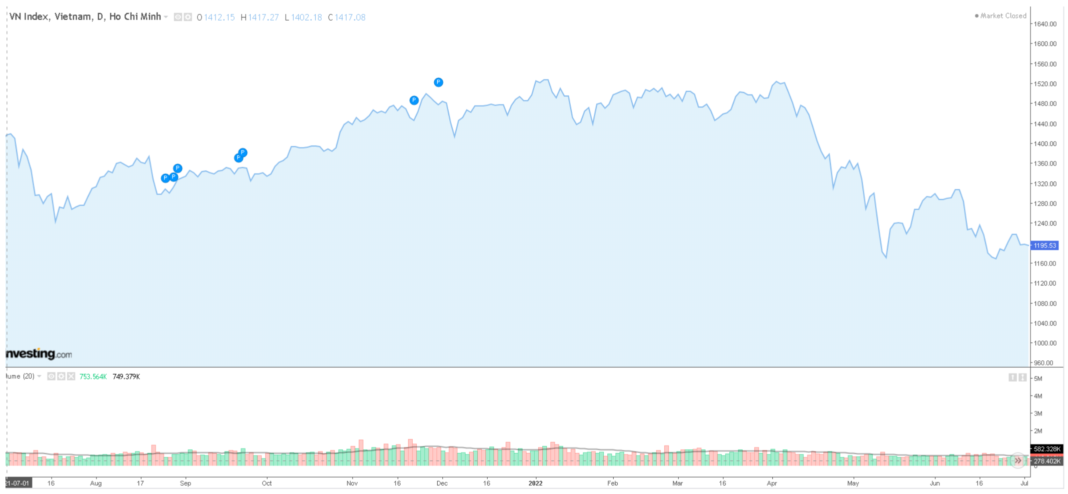
2.1.2 Stock data during the past 1 year (07/2021 - 07/2022)

The data of Vietnam's stock indexes (VNIndex, HNX-Index) are collected via this website : https://www.investing.com/indices/vn for VNIndex and https://www.investing.com/indices/hnx for HNX-Index

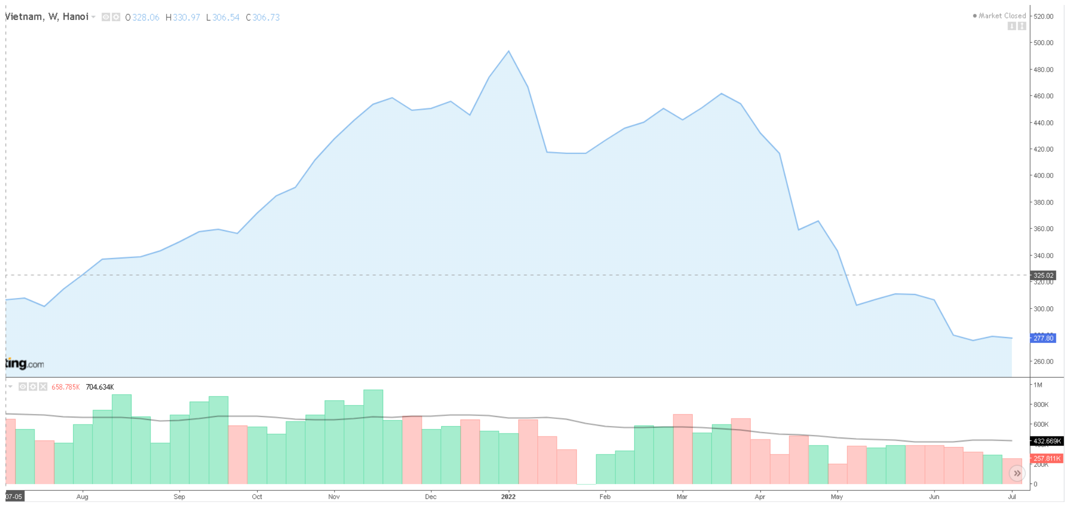
After accessing those links and adjusting the suitable period (07/2021-07/2022), we get these result :



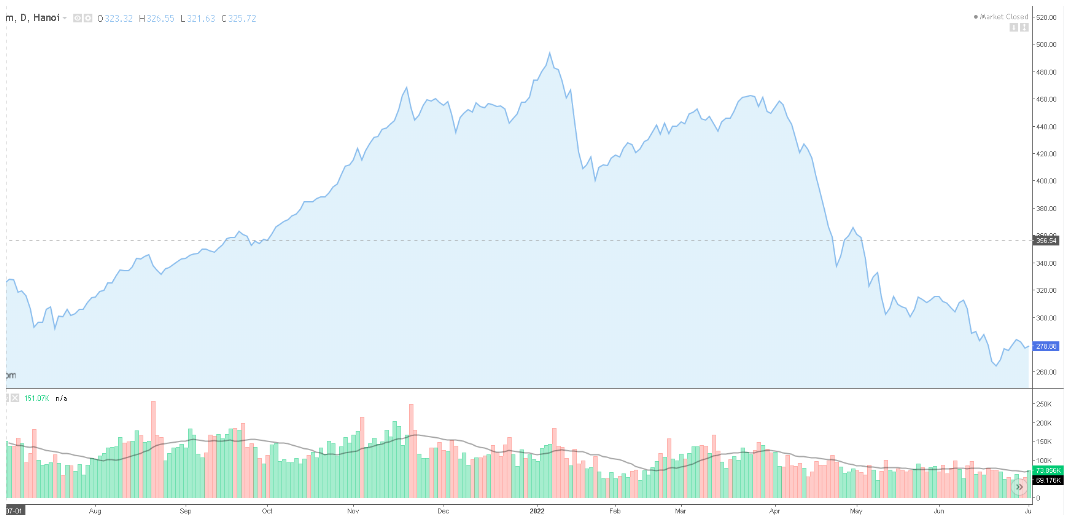
Price change by week of VNIndex



Price change by day of VNIndex



Price change by week of HNX-Index



Price change by day of HNX-Index

2.2 Question 2

2.2.1 How to firgue out 3 stocks with the strongest drop rate and highest growth rate

a. Adding data from file excel to python

Listing 1: Adding data from files

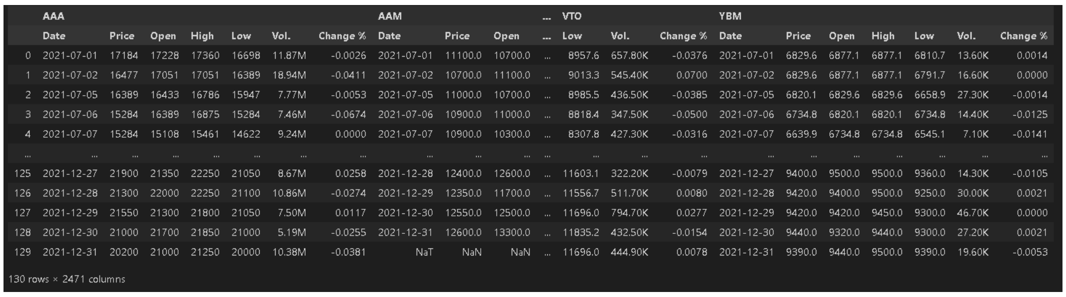
```python
import math
import pandas as pd
from pandas import ExcelFile
import cufflinks as cf
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio
import numpy as np
import datetime
#Put all stocks into an array
label = ['AAA','AAM','AAT','ABR','ABS','ABT','ACB','ACC','ACL','ADG','ADS','AGG','AGM','AGR','AMD','ANV','APC','APG','APH','ASG'
,'ASM','ASP','AST','BAF','BBC','BCE','BCG','BCM','BFC','BHN','BIC','BID','BKG','BMC','BMI','BMP','BRC','BSI','BTP','BTT','BVH','BWE','C32'
,'C47','CAV','CCI','CCL','CDC','CHP','CIG','CII','CKG','CLC','CLL','CLW','CMG','CMV','CMX','CNG','COM','CRC','CRE','CSM','CSV','CTD','CTF'
,'CTG','CTI','CTR','CVT','D2D','DAG','DAH','DAT','DBC','DBT','DC4','DCL','DCM','DGC','DGW','DHA','DHC','DHG','DHM','DIG','DLG','DMC','DPG'
,'DQC','DRC','DRH','DRL','DSN','DTA','DTL','DTT','DVP','DXG','DXS','DXV','EIB','ELC','EMC','EVE','EVF','EVG','FMC','FPT','FRT','FTS'
,'GAB','GAS','GDT','GEG','GEX','GIL','GMD','GSP','GTA','GVR','HAG','HAH','HAI','HAP','HAR','HAS','HAX','HBC','HCD','HCM','HDB','HDC','HDG'
,'HHP','HHS','HID','HII','HMC','HNG','HOT','HPG','HPX','HQC','HRC','HSG','HSL','HT1','HTI','HTL','HTN','HTV','HU1','HU3','HUB','HVH'
,'HVX','IBC','ICT','IDI','IJC','ILB','IMP','ITA','ITC','ITD','JVC','KBC','
    KDC','KHG','KHP','KMR','KOS','KPF','KSB','L10','LAF','LBM','LCG'
,'LDG','LGL','LHG','LIX','LM8','LPB','LSS','MBB','MCG','MCP','MDG','MHC','
    MIG','MSB','MSH','MSN','MWG','NAF','NAV','NCT','NHA','NHH'
,'NHT','NKG','NLG','NNC','NSC','NT2','NTL','NVL','NVT','OCB','OGC','OPC','
    ORS','PAC','PAN','PC1','PDN','PDR','PET','PGC','PGD','PGI','PGV'
,'PHC','PHR','PIT','PJT','PLP','PLX','PMG','PNC','PNJ','POM','POW','PPC','
    PSH','PTB','PTC','PTL','PVD','PVT','QBS','QCG','RAL','RDP','REE'
,'S4A','SAB','SAM','SAV','SBA','SBT','SBV','SC5','SCD','SCR','SCS','SFC','
    SFG','SFI','SGN','SGR','SGT','SHA','SHB','SHI','SHP','SII','SJD'
,'SJF','SJS','SKG','SMA','SMB','SMC','SPM','SRC','SRF','SSB','SSC','SSI','
    ST8','STG','STK','SVC','SVD','SVI','SVT','SZC','SZL','TBC','TCB'
,'TCD','TCH','TCL','TCM','TCO','TCR','TCT','TDC','TDG','TDH','TDM','TDP','
    TDW','TEG','TGG','THG','THI','TIP','TIX','TLD','TLG','TLH','TMP'
,'TMS','TMT','TN1','TNA','TNC','TNH','TNI','TNT','TPB','TPC','TRA','TRC','
    TSC','TTA','TTB','VJC','VMD','VND','VNE','VNG','VNL','VNM','VNS'
,'VOS','VPB','VPD','VPH','VPI','VPS','VRC','VRE','VSC','VSH','VSI','VTB','
    VTO','YBM']
print(len(label))
```

Listing 2: Make python read data from xlxs file

```
#example for first 5 stocks
AAA = pd.read_excel('stock data\AAA Historical Data.xlsx')
AAM = pd.read_excel('stock data\AAM Historical Data.xlsx')
AAT = pd.read_excel('stock data\AAT Historical Data.xlsx')
ABR = pd.read_excel('stock data\ABR Historical Data.xlsx')
ABS = pd.read_excel('stock data\ABS Historical Data.xlsx')
ABT = pd.read_excel('stock data\ABT Historical Data.xlsx')
```

Listing 3: Put data of all stocks into a landsacpe array

```
 stocks = pd.concat([AAA, AAM, AAT, ABR, ABS, ABT, ACB, ACC, ACL, ADG, ADS,
     AGG, AGM, AGR, AMD, ANV, APC, APG, APH, ASG, ASM, ASP, AST, BAF,
BBC, BCE, BCG, BCM, BFC, BHN, BIC, BID, BKG, BMC, BMI, BMP, BRC, BSI, BTP,
    BTT, BVH, BWE, C32, C47, CAV, CCI, CCL, CDC, CHP, CIG, CII, CKG,
CLC, CLL, CLW, CMG, CMV, CMX, CNG, COM, CRC, CRE, CSM, CSV, CTD, CTF, CTG,
    CTI, CTR, CVT, D2D, DAG, DAH, DAT, DBC, DBT, DC4, DCL, DCM, DGC,
DGW, DHA, DHC, DHG, DHM, DIG, DLG, DMC, DPG, DQC, DRC, DRH, DRL, DSN, DTA,
    DTL, DTT, DVP, DXG, DXS, DXV, EIB, ELC, EMC, EVE, EVF, EVG, FMC,
FPT, FRT, FTS, GAB, GAS, GDT, GEG, GEX, GIL, GMD, GSP, GTA, GVR, HAG, HAH,
    HAI, HAP, HAR, HAS, HAX, HBC, HCD, HCM, HDB, HDC, HDG, HHP, HHS,
HID, HII, HMC, HNG, HOT, HPG, HPX, HQC, HRC, HSG, HSL, HT1, HTI, HTL, HTN,
    HTV, HU1, HU3, HUB, HVH, HVX, IBC, ICT, IDI, IJC, ILB, IMP, ITA,
ITC, ITD, JVC, KBC, KDC, KHG, KHP, KMR, KOS, KPF, KSB, L10, LAF, LBM, LCG,
    LDG, LGL, LHG, LIX, LM8, LPB, LSS, MBB, MCG, MCP, MDG, MHC, MIG,
MSB, MSH, MSN, MWG, NAF, NAV, NCT, NHA, NHH, NHT, NKG, NLG, NNC, NSC, NT2,
    NTL, NVL, NVT, OCB, OGC, OPC, ORS, PAC, PAN, PC1, PDN, PDR, PET,
PGC, PGD, PGI, PGV, PHC, PHR, PIT, PJT, PLP, PLX, PMG, PNC, PNJ, POM, POW,
    PPC, PSH, PTB, PTC, PTL, PVD, PVT, QBS, QCG, RAL, RDP, REE, S4A,
SAB, SAM, SAV, SBA, SBT, SBV, SC5, SCD, SCR, SCS, SFC, SFG, SFI, SGN, SGR,
    SGT, SHA, SHB, SHI, SHP, SII, SJD, SJF, SJS, SKG, SMA, SMB, SMC,
SPM, SRC, SRF, SSB, SSC, SSI, ST8, STG, STK, SVC, SVD, SVI, SVT, SZC, SZL,
    TBC, TCB, TCD, TCH, TCL, TCM, TCO, TCR, TCT, TDC, TDG, TDH, TDM,
TDP, TDW, TEG, TGG, THG, THI, TIP, TIX, TLD, TLG, TLH, TMP, TMS, TMT, TN1,
    TNA, TNC, TNH, TNI, TNT, TPB, TPC, TRA, TRC, TSC, TTA, TTB, VJC,
VMD, VND, VNE, VNG, VNL, VNM, VNS, VOS, VPB, VPD, VPH, VPI, VPS, VRC, VRE,
    VSC, VSH, VSI, VTB, VTO, YBM],axis=1, keys = label)
stocks
```



	AAA							AAM			...	VTO			YBM						
	Date	Price	Open	High	Low	Vol.	Change %	Date	Price	Open	...	Low	Vol.	Change %	Date	Price	Open	High	Low	Vol.	Change %
0	2021-07-01	17184	17228	17360	16698	11.87M	-0.0026	2021-07-01	11100.0	10700.0	...	8957.6	657.80K	-0.0376	2021-07-01	6829.6	6877.1	6877.1	6810.7	13.60K	0.0014
1	2021-07-02	16477	17051	17051	16389	18.94M	-0.0411	2021-07-02	10700.0	11100.0	...	9013.3	545.40K	0.0700	2021-07-02	6829.6	6877.1	6877.1	6791.7	16.60K	0.0000
2	2021-07-05	16389	16433	16786	15947	7.77M	-0.0053	2021-07-05	11000.0	10700.0	...	8985.5	436.50K	-0.0385	2021-07-05	6820.1	6829.6	6829.6	6658.9	27.30K	-0.0014
3	2021-07-06	15284	16389	16875	15284	7.46M	-0.0674	2021-07-06	10900.0	11000.0	...	8818.4	347.50K	-0.0500	2021-07-06	6734.8	6820.1	6820.1	6734.8	14.40K	-0.0125
4	2021-07-07	15284	15108	15461	14622	9.24M	0.0000	2021-07-07	10900.0	10300.0	...	8307.8	427.30K	-0.0316	2021-07-07	6639.9	6734.8	6734.8	6545.1	7.10K	-0.0141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
125	2021-12-27	21900	21350	22250	21050	8.67M	0.0258	2021-12-28	12400.0	12600.0	...	11603.1	322.20K	-0.0079	2021-12-27	9400.0	9500.0	9500.0	9360.0	14.30K	-0.0105
126	2021-12-28	21300	22000	22250	21100	10.86M	-0.0274	2021-12-29	12350.0	11700.0	...	11556.7	511.70K	0.0080	2021-12-28	9420.0	9400.0	9500.0	9250.0	30.00K	0.0021
127	2021-12-29	21550	21300	21800	21050	7.50M	0.0117	2021-12-30	12550.0	12500.0	...	11696.0	794.70K	0.0277	2021-12-29	9420.0	9420.0	9450.0	9300.0	46.70K	0.0000
128	2021-12-30	21000	21700	21850	21000	5.19M	-0.0255	2021-12-31	12600.0	13300.0	...	11835.2	432.50K	-0.0154	2021-12-30	9440.0	9320.0	9440.0	9300.0	27.20K	0.0021
129	2021-12-31	20200	21000	21250	20000	10.38M	-0.0381	NaT	NaN	NaN	...	11696.0	444.90K	0.0078	2021-12-31	9390.0	9440.0	9500.0	9390.0	19.60K	-0.0053

130 rows × 2471 columns

Result of above code

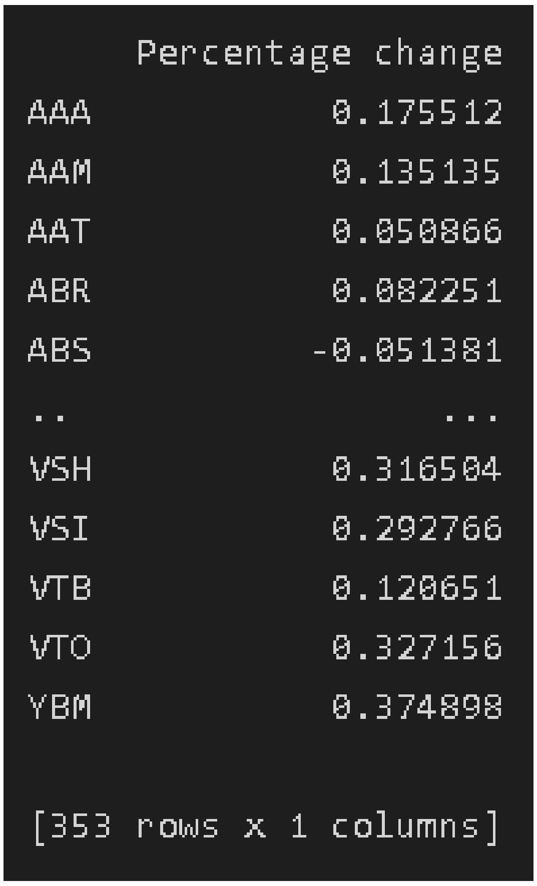
b. Find the percentage changes of each stocks
In this step, we calculate the percentage changes by this formula :

$$\text{Percentage change} = \frac{P_f - P_i}{P_i}$$

where: P_f is final price, P_i is initial price

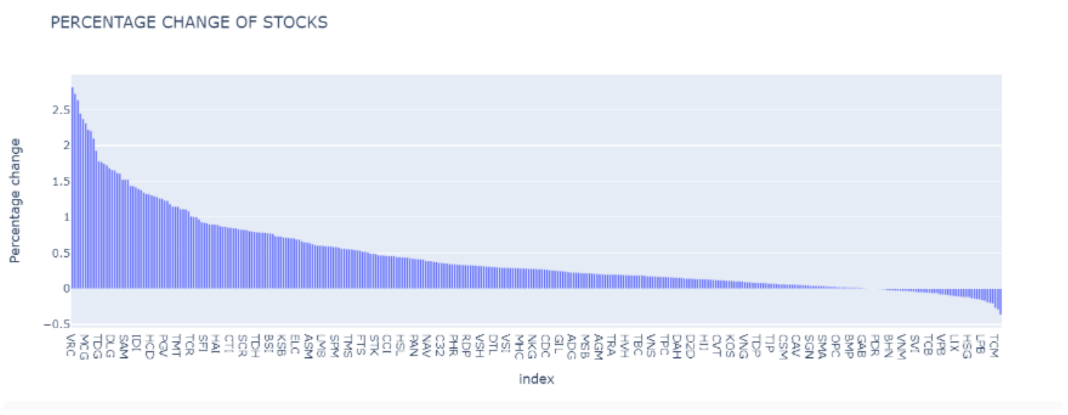
Listing 4: Find the percentage change of each stock

```
    array = np.array([1.1 for i in range(0,len(label))])
    i = 0
    for name in label:
    j = len(stocks[name]['Price']) - 1
    while(j > - 1):
        if(np.isnan(stocks[name]['Price'][j]) == 0):
            array[i] = (stocks[name]['Price'][j] - stocks[name]['Price'][0])
                /stocks[name]['Price'][0]
            break
        j = j - 1
    i = i + 1
percentage_change = pd.DataFrame(data = array, index= [name for name in
    label], columns= ['Percentage change'])
print(percentage_change)

#Put the value into a graph of percentage change
fig = px.bar(percentage_change,y= percentage_change['Percentage change'],
    title='PERCENTAGE CHANGE OF STOCKS')
fig.show()
```



```
      Percentage change
AAA            0.175512
AAM            0.135135
AAT            0.050866
ABR            0.082251
ABS           -0.051381
..                  ...
VSH            0.316504
VSI            0.292766
VTB            0.120651
VTO            0.327156
YBM            0.374898

[353 rows x 1 columns]
```

Percentage changes of stocks



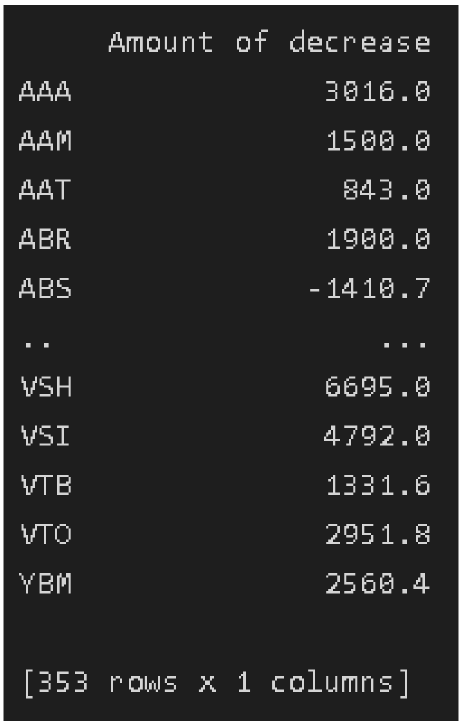
Percentage changes of stocks Graph

c. Amount of decrease
We use this method to firgue out the amount of decrease:

$$\text{Amount of decrease} = P_i \times \text{ percentage change } (P_i \text{ is initial price})$$

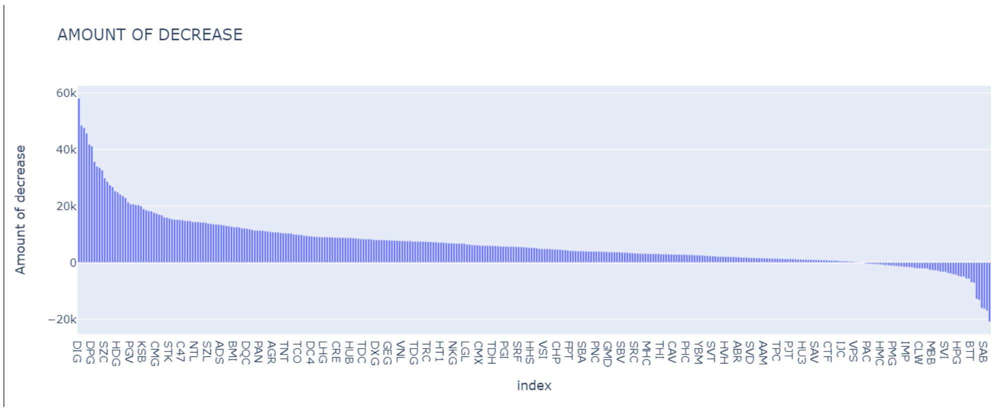
Listing 5: Find the amount of decrease of each stock

```python
 decrease = np.array([1.1 for i in range(0,len(label))])
i = 0
for name in label:
    decrease[i] = stocks[name]['Price'][0] * percentage_change['Percentage
        change'][i]
    i = i + 1
decrease_data = pd.DataFrame(data = decrease, index= [name for name in label
    ], columns= ['Amount of decrease'])
print(decrease_data)
#Show result in Bar Chart
fig = px.bar(decrease_data,y= decrease_data['Amount of decrease'],title='
    AMOUNT OF DECREASE')
fig.show()
```



```
      Amount of decrease
AAA               3016.0
AAM               1500.0
AAT                843.0
ABR               1900.0
ABS              -1410.7
..                   ...
VSH               6695.0
VSI               4792.0
VTB               1331.6
VTO               2951.8
YBM               2560.4

[353 rows x 1 columns]
```

Amount of decrease



Bar chart shows amount of decrease

d. Conclusion

According to the bar chart showing percentage of change, we conclude that :

- 3 stocks with the highest growth rate on VNIndex in the period from July to December 2021 are : Vrc Real Estate and Investment JSC (VRC), Development Investment Construction JSC (DIG) and FPT Digital Retail JSC (FRT).
- 3 stocks with strongest drop rate on VNIndex in the period from July to December 2021 are : An Phat Holdings JSC (APH), Hanoi Plastics JSC (NHH) and Thanh Cong Textile Garment Investment Trading JSC (TCM).

2.2.2 Distribution of Stocks

a. Simple return
The simple return is calculated by $R = \frac{P_i - P_j}{P_j}$

where R is return rate, P_i is before price, P_j is after price.
And to find the total return over a period T with n sub-periods, we need to compound the growth/drop of each sub-period : $P_f = P_i(1 + R_1)(1 + R_2)...(1 + R_n) = P_i(1 + R_n)^n$
With :

- P_f - Final Price
- P_i - Initial Price
- R_x - Return for each sub period

However, we can recognize that the return rate of a sub-period is different from each other, so the compound manner can create many errors. For this reason, we turn our attention to logarithmic return.

b. Logarithmic return

Logarithmic return measures the rate of exponential growth. In this method, we calculate the exponent of its natural growth/drop during each sub-period. After that, adding each sub-period's exponential growth/drop to get the total for the period T.
In order to calculate the change of stock value by day, we usually use the Logarithmic returns

instead of simple return.
Let's take an example, you invest an amount P1 with a 4% invest rate for one year, so $P_2 = 1.04P_1$ (calculated by simple return). Sometimes, your interest might be compounded, you get 2% at the end of 6 months, and then can earn 2% interest on both original principal as well as on the first six months' interest. Therefore, after a year, the amount P_2 you can get will be $P_2 = P_1(1 + \frac{r}{2})(1 + \frac{r}{2})$ $=P_1(1 + \frac{r}{2})^2$

Similarly, if it is compounded quarterly, you will get: $P_2 = P_1(1 + \frac{r}{4})^4$

It turns to formula and when n gets arbitrarily large, $lim_{x\to\infty}(1 + \frac{r}{n})^n = e^r$

Thus, if you earn $r\%$ interest that is compounded continuously, after a year you will get: $P_2 = P_1 lim_{x\to\infty}(1 + \frac{r}{n})^n = P_1 e^r$

e is a mathematical constants whose value is approximately 2.71828. Taking the natural logarithm we can receive $r = ln(\frac{P_2}{P_1})$

c. Some well-known distributions

- **Normal Distribution**

  Normal distribution, also known as the Gaussian distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.

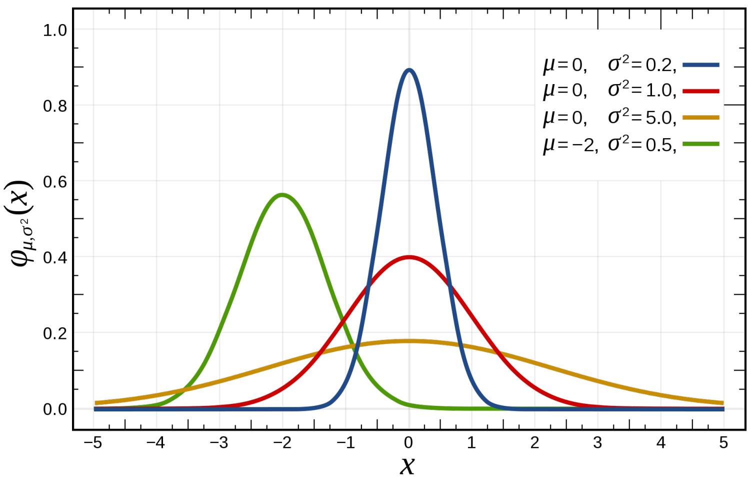
  The normal distribution is the most common type of distribution assumed in technical stock market analysis and in other types of statistical analyses. The standard normal distribution has two parameters: the mean and the standard deviation.

  Although the normal distribution is an extremely important statistical concept, its applications in finance can be limited because financial phenomena—such as expected stock-market returns do not fall neatly within a normal distribution

  In normal distribution, we pay attention to two parameters. The parameter μ is the mean or expectation of the distribution (and also its median and mode), while the parameter σ is its standard deviation. The variance of the distribution is σ^2. A random variable with a Gaussian distribution is said to be normally distributed, and is called a normal deviate.

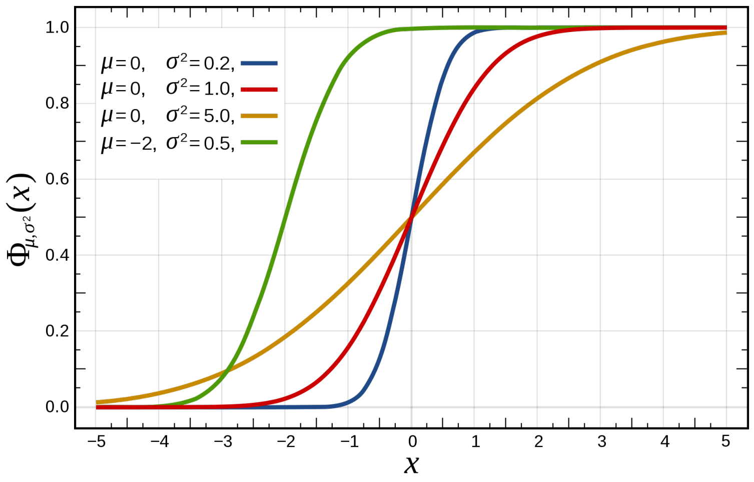
  We can see the feature of a normal distribution through *cumulative distribution function* (CDF) and *probability density function* (PDF).

  In probability theory and statistics, PDF is used to represent an integral probability distribution. The probability density function is always non-negative, and its integral from $-\infty$ to $+\infty$ is equal to 1. If a probability distribution has a density $f(x)$, then the interval is intuitively The (infinitely small) fraction $[x, x + dx]$ has a probability equal to $f(x)dx$.



Probability density function of 4 different parameter sets

The CDF of a real-valued random variable X, or just distribution function of X , evaluated at x, is the probability that X will take a value less than or equal to x.The cumulative distribution function of a real-valued random variable x is the function given by: $F_X(x) = P(X \leq x)$ where the right-hand side represents the probability that the random variable X takes on a value less than or equal to x.



[h]

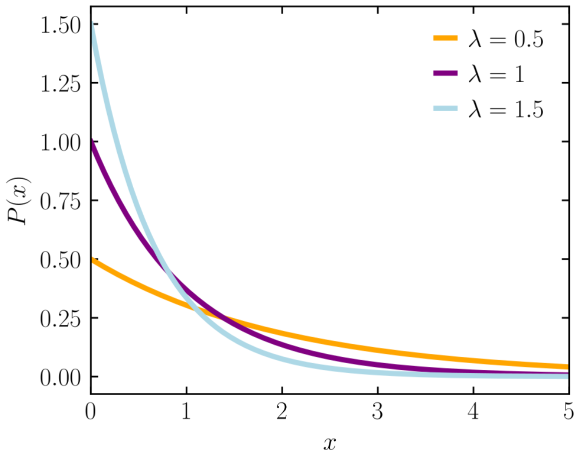
Cumulative Distribution Function of 4 different parameter sets

- **Exponential distribution**

  In probability theory and statistics, the exponential distribution is the probability distribution of the time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate.

The probability density function (pdf) of an exponential distribution is

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} \text{ when } x \geq 0 \\ 0 \text{ when } x < 0 \end{cases}$$

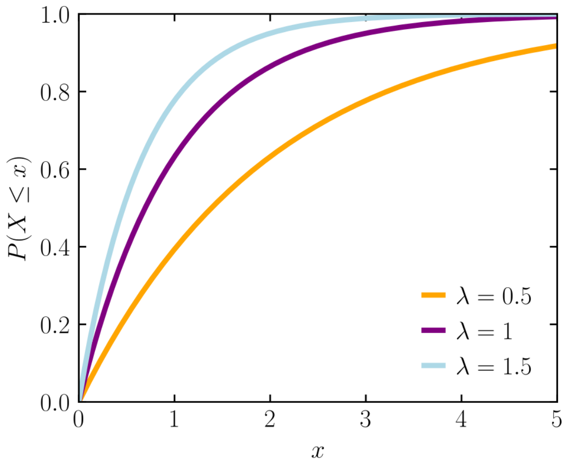


[h]

Probability density function

The cumulative distribution function is given by :

$$F(x;\lambda) = \begin{cases} 1 - e^{-x/\beta} \text{ when } x \geq 0 \\ 0 \text{ when } x < 0 \end{cases}$$



[h]

Cumulative distribution function

- **Poisson distribution**

  In probability theory and statistics, the Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant mean rate and independently of the time since the last event.The Poisson distribution can also be used for the number of events in other specified interval types such as distance, area, or volume.

  For instance, a call center receives an average of 180 calls per hour, 24 hours a day. The calls are independent; receiving one does not change the probability of when the next one will arrive. The number of calls received during any minute has a Poisson probability distribution with mean 3: the most likely numbers are 2 and 3 but 1 and 4 are also likely and there is a small probability of it being as low as zero and a very small probability it could be 10.

  A discrete random variable X is said to have a Poisson distribution, with parameter $\lambda > 0$, if it has a probability mass function given by :

  $f(k;\lambda) = Pr(X = k) = \frac{\lambda^k e^{-k}}{k!}$

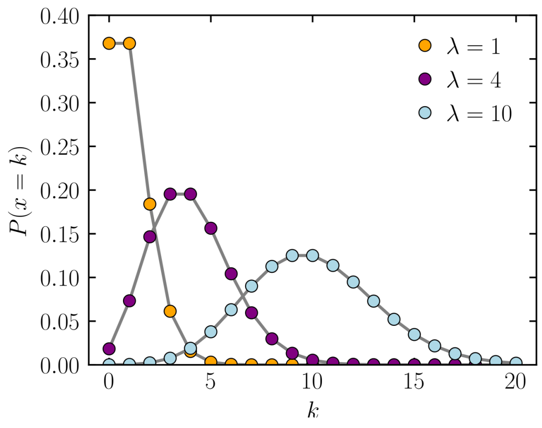
  where

  · k is the number of occurrences (k=0,1,2,3,...)

  · e is Euler number ($\approx$ 2.71828)

  · ! is factorial function

[h]



Probability mass function

d. Apply distribution to stock
In this section we use Python to determine the distribution of these stocks

- VNINDEX
- 3 stocks with highest increase rate : VRC, DIG, FRT.
- 3 stocks with strongest drop rate : APH, NHH, TCM.
- Stock in construction : Vingroup JSC (VIC)

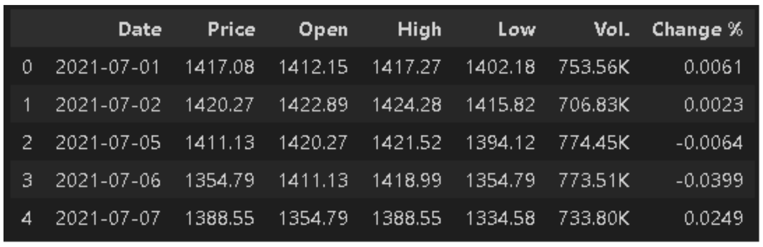
- Stock in service : Vietnam Exhibition Fair Centre JSC (VEF)
- Stock in transportation : Gemadept Corp (GMD)

i. Source Code
In this part, we will show code to determine the distribution of VNIndex (apply the same code but change the stock's name for others)

Listing 6: Distribution of VNIndex

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

from pandas import ExcelFile

df = pd.read_excel('VN Index Historical Data Distribution.xlsx')
df.head()
```

After this first step the excel file will be read and the result will be shown by the picture below:

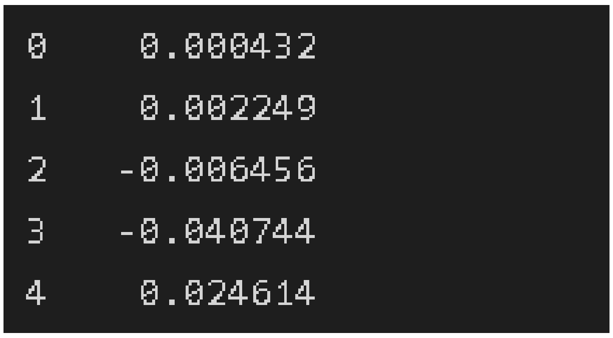


	Date	Price	Open	High	Low	Vol.	Change %
0	2021-07-01	1417.08	1412.15	1417.27	1402.18	753.56K	0.0061
1	2021-07-02	1420.27	1422.89	1424.28	1415.82	706.83K	0.0023
2	2021-07-05	1411.13	1420.27	1421.52	1394.12	774.45K	-0.0064
3	2021-07-06	1354.79	1411.13	1418.99	1354.79	773.51K	-0.0399
4	2021-07-07	1388.55	1354.79	1388.55	1334.58	733.80K	0.0249

Listing 7: Find ...

```
r_t = np.log(df['Price']/df['Price'].shift(1))
mean = np.nanmean(r_t)
r_t[0]=mean
r_t[:5]
```

About the second step, we will calculate the log return of each day by the formula as we mention before. However, at the first day it doesn't have data of the previous day so the calculation in this day will be Nan so we calculate mean of all data and replace the Nan row by this data in order to standardized the data so the first of five data in log return column is as the image:

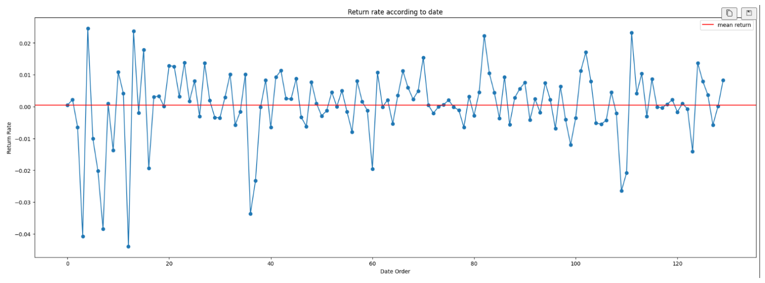


```
0     0.000432
1     0.002249
2    -0.006456
3    -0.040744
4     0.024614
```

Listing 8: Return rate according to date

```python
#The reutrn rate by day is showned in a graph
plt.figure(figsize=(24, 8))
plt.plot(np.arange(r_t.shape[0]), r_t, '-o')
plt.axhline(y=mean, label='mean return', c='red')
plt.title('Return rate according to date')
plt.xlabel('Date Order')
plt.ylabel('Return Rate')
plt.legend()
plt.show()
```

To get better understand about the trend of price, we use the yield chart of Return rate

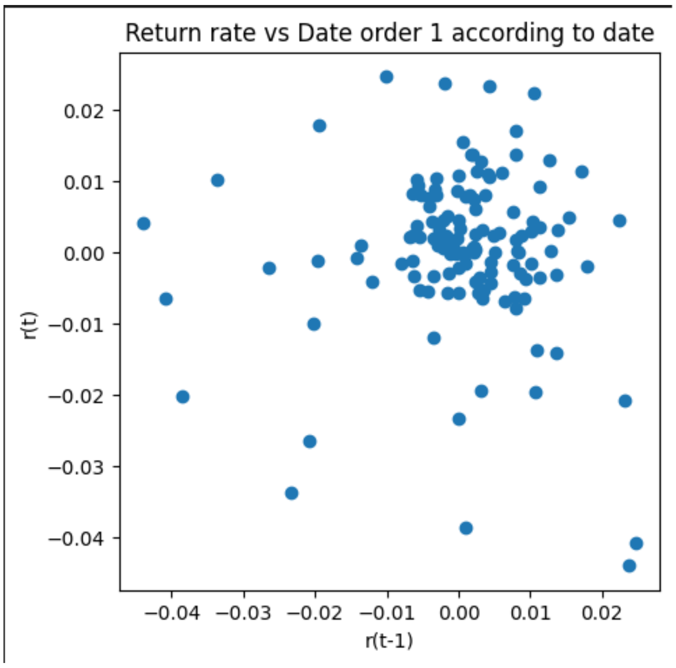


Comment: The yield series plot shows it as a white noise stochastic, with mean return $\mu \approx 0$ and constant variance of the distribution.

Listing 9: Linear relationship between Return rate vs Date order

```python
plt.figure(figsize=(5, 5))
plt.scatter(x=r_t[1:], y=r_t[:-1])
plt.title('Return rate vs Date order 1 according to date')
plt.xlabel('r(t-1)')
plt.ylabel('r(t)')
plt.show()
```

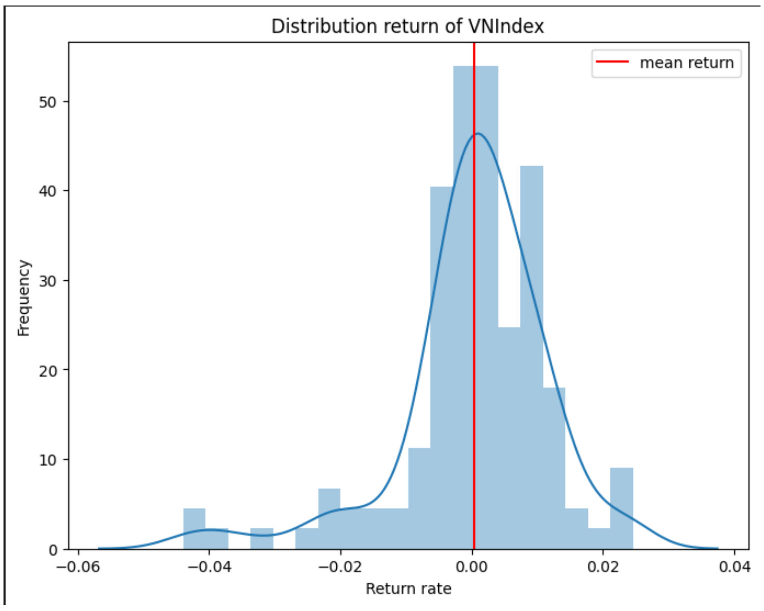
We can plot the representation of Return rate against the Date order to see whether they have linear or random relationship.



Grpah shows that Return rate and Rate Order have no correlation relationship. The graph consists of many points that do not follow a particular trend

Listing 10: Probability distribution chart

```
plt.figure(figsize = (8, 6))
sns.distplot(r_t, bins = 20)
plt.axvline(x=mean, label='mean return', c='red')
plt.title('Distribution return of VNIndex')
plt.legend()
plt.xlabel('Return rate')
plt.ylabel('Frequency')
```



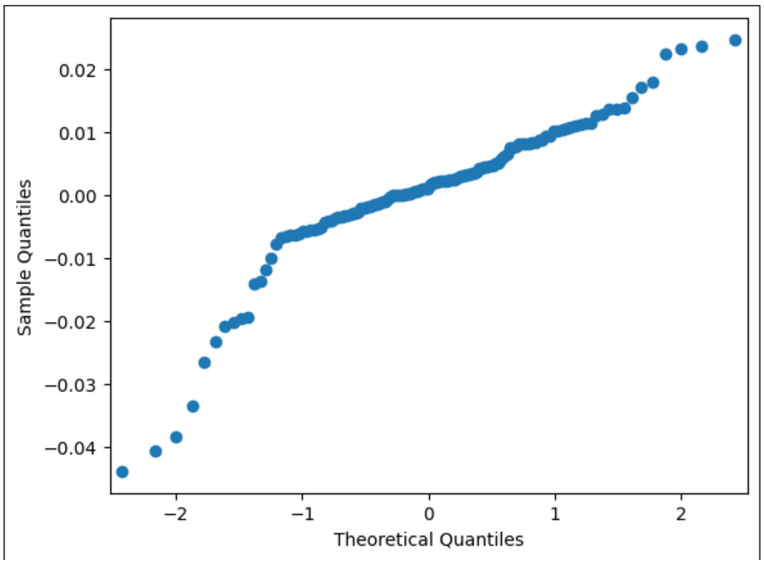
As we see in the graph, the distribution of VNI is quite similar to a blue line(Probability density function of normal distribution) so we can conclude that the stock of VNIndex follows normal distribution

We can test the normal distribution through quantiles quantiles plot.

Listing 11: Graph of Quantile-Quantile plot

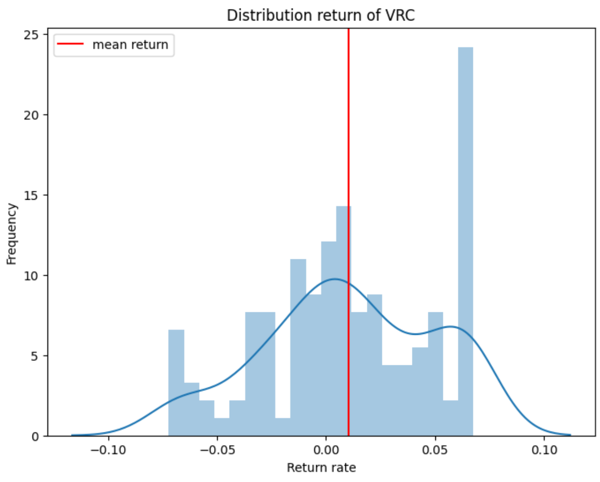
```python
#Using directly sm.qqplot()
sm.qqplot(r_t)
plt.show()
```

Graph of Quantile-Quantile plot is one of the non-parametric testing methods in statistics to test whether a quantity follows a certain distribution based on comparing the shapes of two emperical and theoretical probability distribution

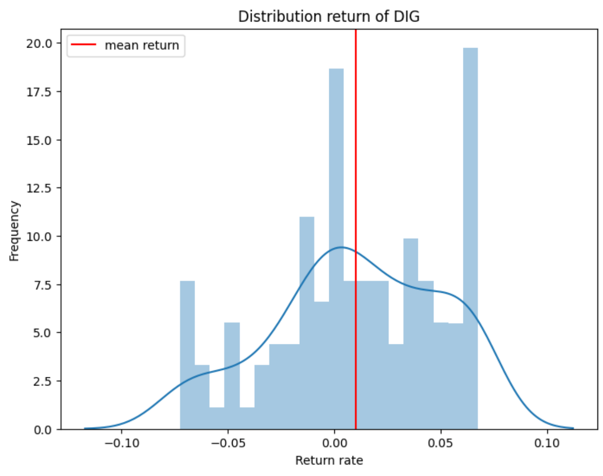


ii. Results of others stocks

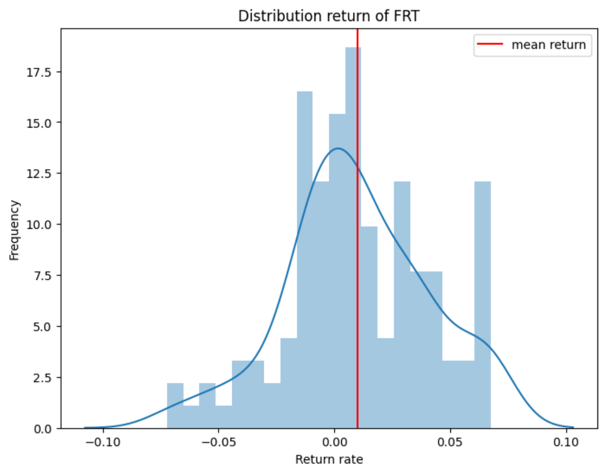
a. Distribution of 3 stocks with the highest growth rate on VNINDEX in the period from July to December 2021 (VRC, DIG, FRT)



It follows normal distribution

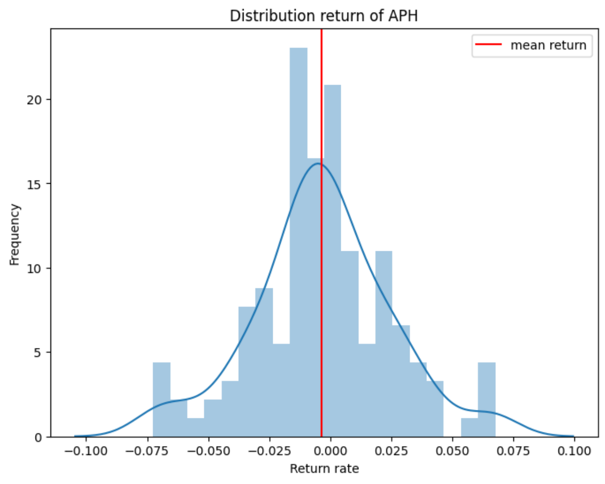


It follows normal distribution

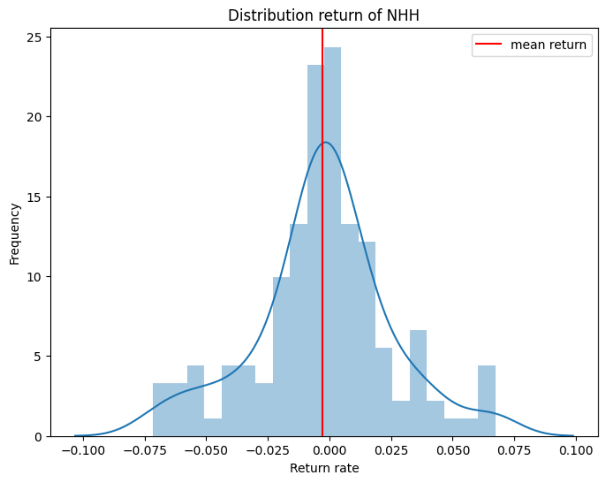


It follows normal distribution

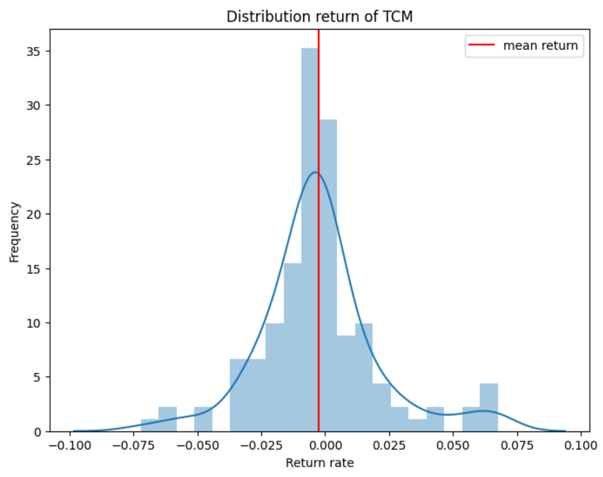
b. Distribution of 3 stocks with the strongest drop rate on VNINDEX in the period from July to December 2021 (APH, NHH, TCM)



It follows normal distribution

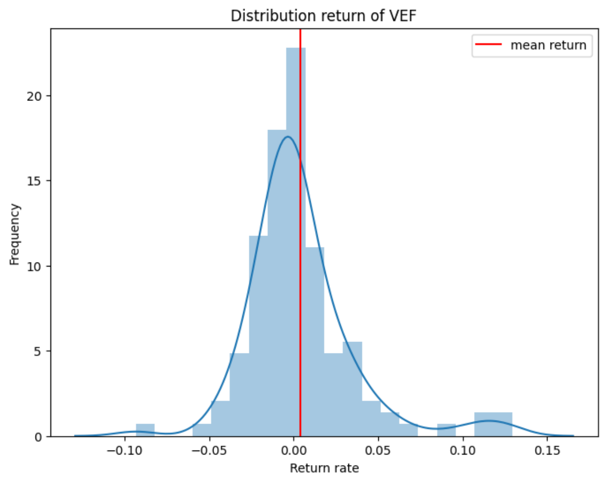


It follows normal distribution



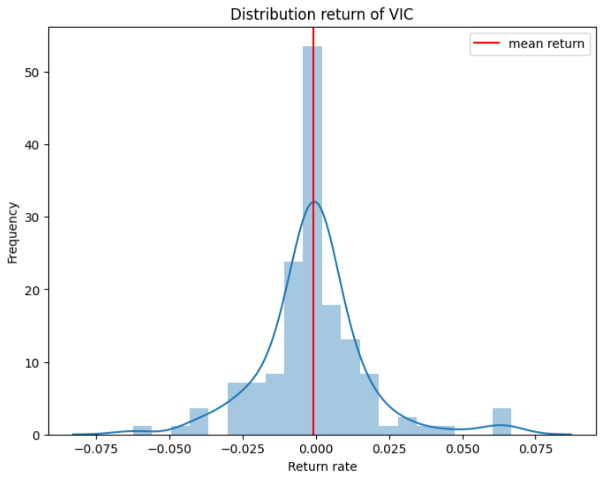
It follows normal distribution

c. Distribution of stock in service : VEF



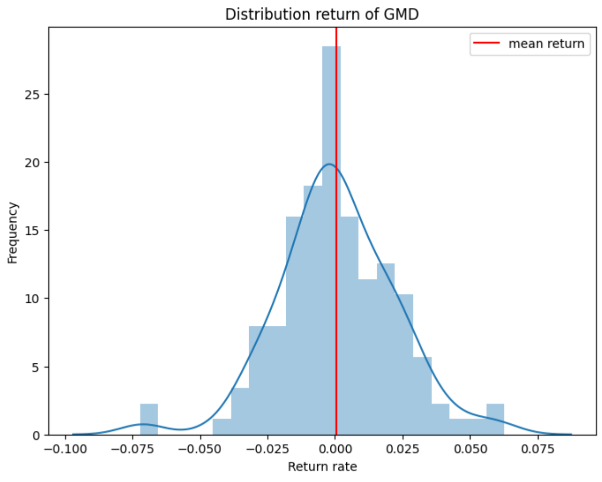
It follows normal distribution

d. Distribution of stock in construction : VIC



It follows normal distribution

e. Distribution of stock in transportation : GMD



It follows normal distribution

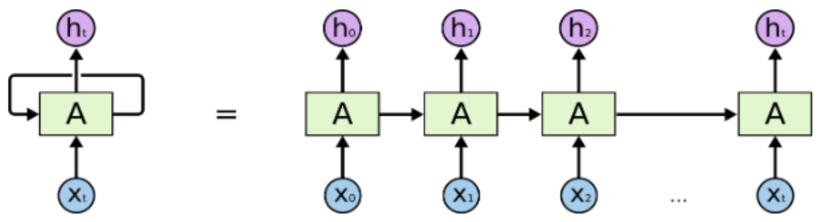
2.3 Question 3: Predict the model for the next 3 months

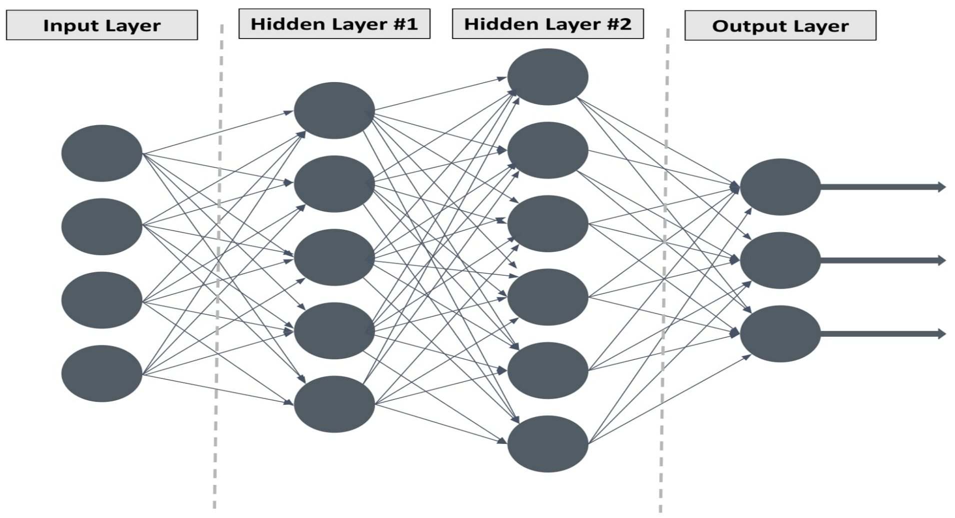
2.3.1 RRN and LSTM

RRN (Recurrent Neural Network) is a wonderful algorithm in Deep Learning, which is a class of artificial neural networks where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes. This model is commonly used in language translation, natural language processing, speech recognition, image captioning. Some popular applications such as Siri, Google Translate, voice search becoming indispensable nowadays based anonymously on this algorithm.

In the definition of language, the meaning of a sentence is formulated by the connection of word(vocab) based on a particular grammar structure. If consider each single word, we cannot fully understand its meaning unless we have to look at its relation. RRN was built for this problem

In term of stock market prediction, RRN consists of loops, allowing data being passed over time, a variant of the conventional feedforward artificial neural networks can properly deal with a huge load of sequential data such as stock price.





In order to make a predictive model, LSTM (Long Short-Term Memory) with denote each recurrnet node as an *internal state*, which can overcome the vanishing gradient problem-an ordinary issue in RRN. Moreover, LSTM is created for avoiding *long-term dependency*. Conducting on three main functions:

i) sigmod function: $\frac{1}{1+e^{-x}}$

ii) tanh function: $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

iii) relu function: max(0, x)

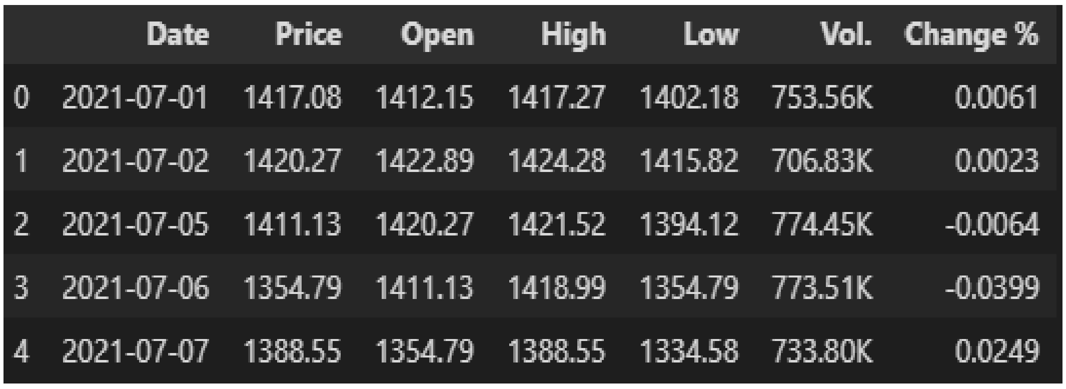
2.3.2 Conducting LSTM model on VNI-index

Python has provided with a wide range of libraries which can used to infrastructure the LSTM model.

Listing 12: predict-VNI.ipynb

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime as dt
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense , Dropout , LSTM
```

At the beginning, we got a variable *df* = pd.readexcel('VN Index Historical Data.xlsx)

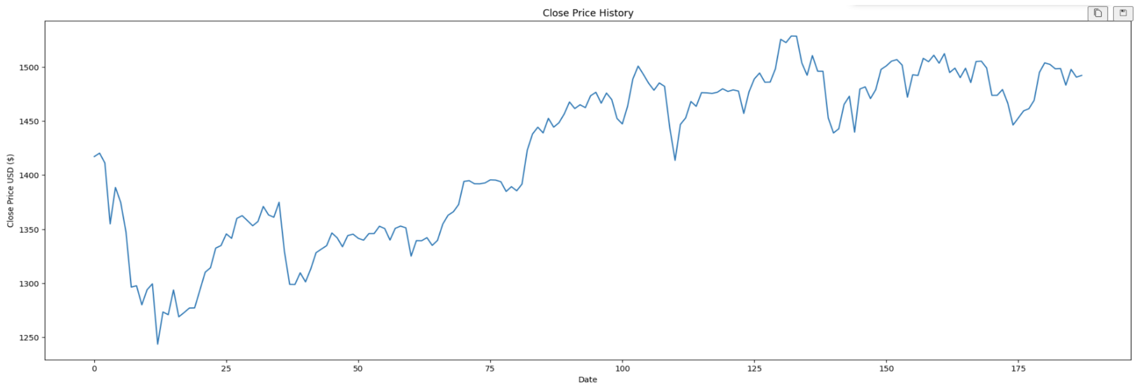


	Date	Price	Open	High	Low	Vol.	Change %
0	2021-07-01	1417.08	1412.15	1417.27	1402.18	753.56K	0.0061
1	2021-07-02	1420.27	1422.89	1424.28	1415.82	706.83K	0.0023
2	2021-07-05	1411.13	1420.27	1421.52	1394.12	774.45K	-0.0064
3	2021-07-06	1354.79	1411.13	1418.99	1354.79	773.51K	-0.0399
4	2021-07-07	1388.55	1354.79	1388.55	1334.58	733.80K	0.0249

After having the stock data stored in the variable *df*, using *plt* in order to make graph of the VNI-index from 01-07-2021 to 31-12-2021

Listing 13: graph-VNI.ipynb

```python
plt.figure(figsize=(24,8))
plt.title('Close Price History')
plt.plot(df['Price'])
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.show()
```



In predicting the trend of the stock, we care mostly on the the column 'Price', so that we have to make a dataset contains only the 'Price' value daily.

Listing 14: collect-'Price'.ipynb

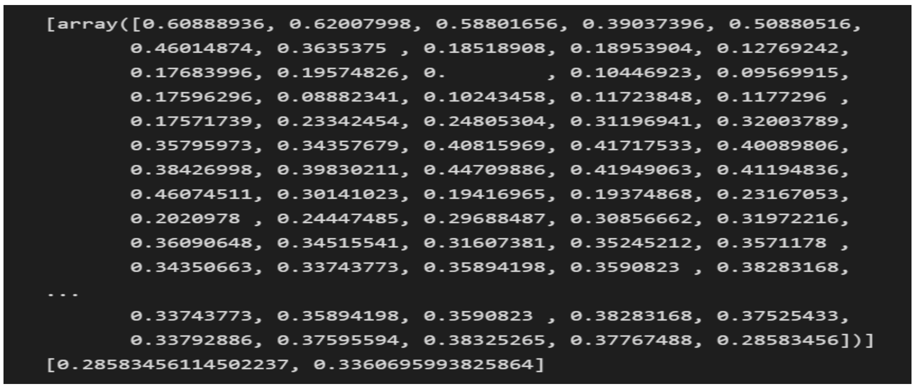
```python
data = df.filter(['Price'])
dataset = data.values
training_data_len = int(np.ceil( len(dataset)) - 62)

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
scaled_data
```

To fit in a neural network, we will extract the *numpy* array from the dataframe and convert the integer values into floating point values. The scaler imported from *sklearn.processing* will transform the 'Price' value into a floating number in the range [0, 1]. Scaling the data from the *dataset*, we will have an array with a desired data.

Listing 15: collect-'Price'.ipynb

```python
train_data = scaled_data[0:int(training_data_len), :]
x_train = []
y_train = []

predictions_days = 60

for i in range(predictions_days, len(train_data)):
    x_train.append(train_data[i-predictions_days:i, 0])
    y_train.append(train_data[i, 0])
    if i<= (predictions_days + 1):
        print(x_train)
        print(y_train)
        print()

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))
```

Prediction-day = 60 stands for the amount of days in the past we want to look for in order to fit in the model, the less the amount of days we looking back, the less accuracy we get in the prediction, however, the more amount of days we looking back, the more time-consuming and complexity of the model.
After compling the code, we got a numpy array *x-train* below.



```
[array([0.60888936, 0.62007998, 0.58801656, 0.39037396, 0.50880516,
        0.46014874, 0.3635375 , 0.18518908, 0.18953904, 0.12769242,
        0.17683996, 0.19574826, 0.        , 0.10446923, 0.09569915,
        0.17596296, 0.08882341, 0.10243458, 0.11723848, 0.1177296 ,
        0.17571739, 0.23342454, 0.24805304, 0.31196941, 0.32003789,
        0.35795973, 0.34357679, 0.40815969, 0.41717533, 0.40089806,
        0.38426998, 0.39830211, 0.44709886, 0.41949063, 0.41194836,
        0.46074511, 0.30141023, 0.19416965, 0.19374868, 0.23167053,
        0.2020978 , 0.24447485, 0.29688487, 0.30856662, 0.31972216,
        0.36090648, 0.34515541, 0.31607381, 0.35245212, 0.3571178 ,
        0.34350663, 0.33743773, 0.35894198, 0.3590823 , 0.38283168,
...
        0.33743773, 0.35894198, 0.3590823 , 0.38283168, 0.37525433,
        0.33792886, 0.37595594, 0.38325265, 0.37767488, 0.28583456])]
[0.28583456114502237, 0.3360695993825864]
```

After preparing all the data done, we now build the neural network model for the prediction. By adding respectively a LSTM layer then a dropout layer until the end, we put one last Dense layer, this is the stock price prediction we looking for.

Listing 16: LSTM-model.ipynb

```python
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1))
    )
model.add(LSTM(128, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(x_train, y_train, batch_size=1, epochs=25)
```

As we can notice below, the LSTM layer is model that we execute from the x-train that we have extracted previously. Then we must return it to the Sequence because RRN demand the model to feed back the information that is executed before.

Getting the model finished, before plotting the graph we have to test the accuracy of the model.

Listing 17: testing-model.ipynb

```python
test_data = scaled_data[training_data_len - predictions_days: , :]
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(predictions_days, len(test_data)):
    x_test.append(test_data[i-predictions_days:i, 0])
```

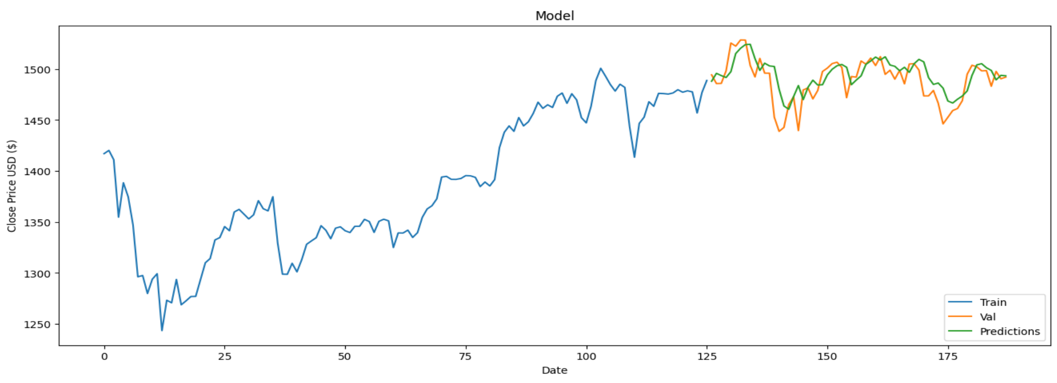
```

x_test = np.array(x_test)

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

predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
rmse
```

The *scaled-data* used for loading the data into the variable *test-data*. Then we transform it into numpy for more accuracy in Python because we do not care about the actual 'Price' in the prediction, we just need to know its new graph. Then put it in the *model*, noted in our mind is that the predicted data is scaled so we need to reverse them using *sklearn*, now we will get the actual prices.

Now we will illustrate it in a graph by *plt*, change some noticed figures so that we can see it more easily.

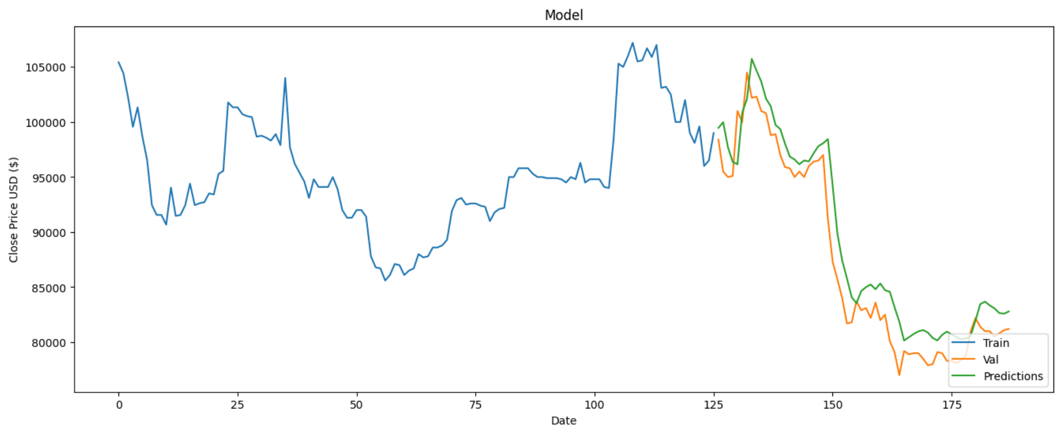
Listing 18: visualizing-model.ipynb

```
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(train['Price'])
plt.plot(valid[['Price', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

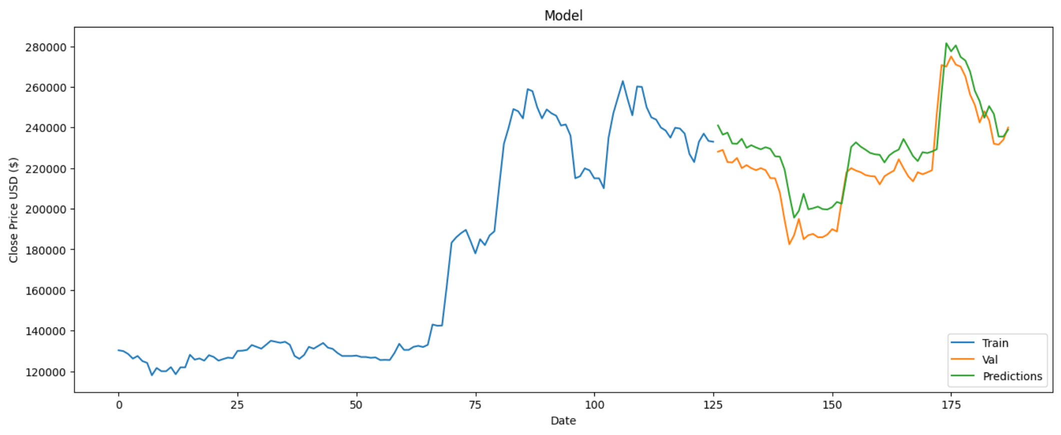


2.3.3 Applying the model into 10 given stock

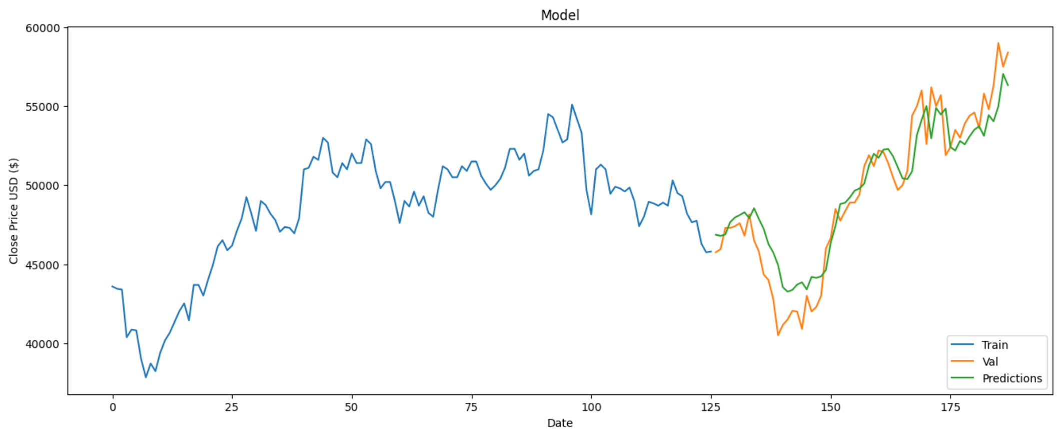
2.3.3.a Three stocks in three different sectors: transportation, construction, services



Construction

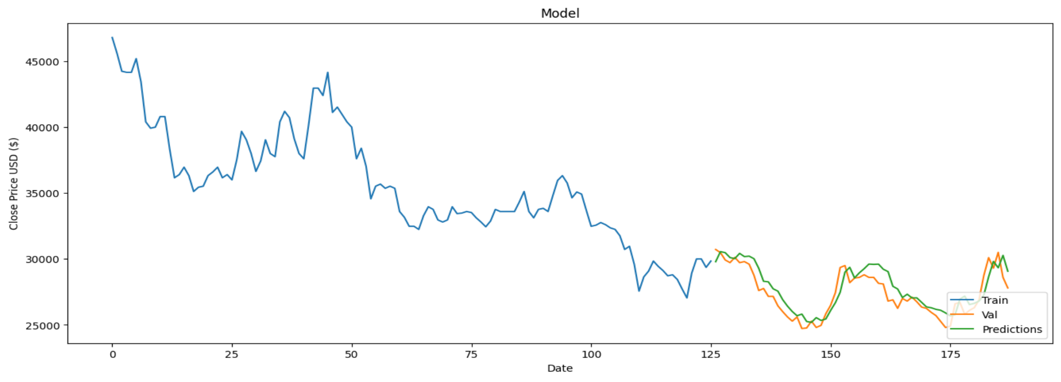


Service

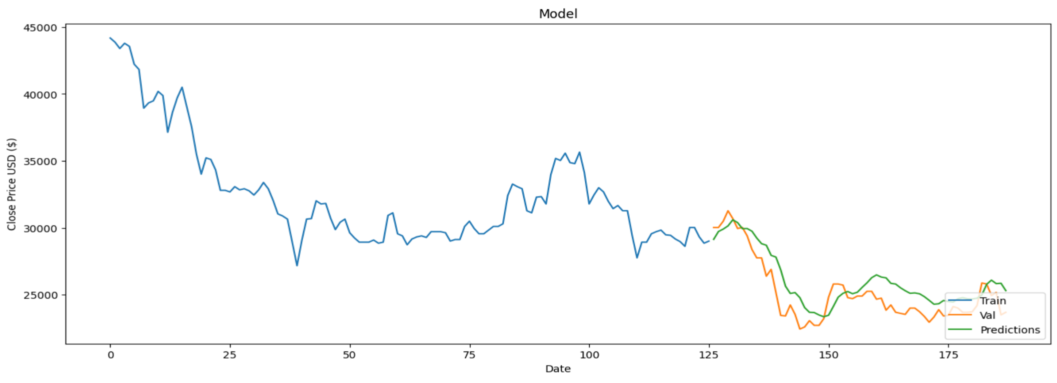


Transportation

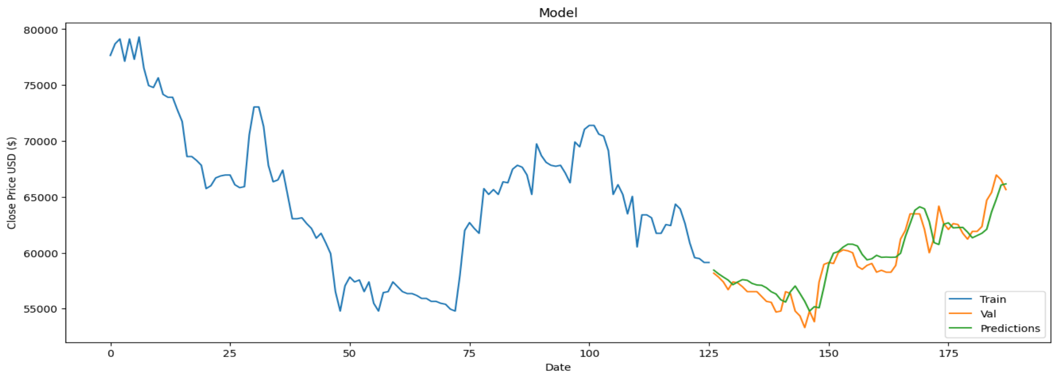
2.3.3.b Three stocks with the strongest drop rate on VNINDEX (APH, NHH, TCM)



Stock prediction of APH

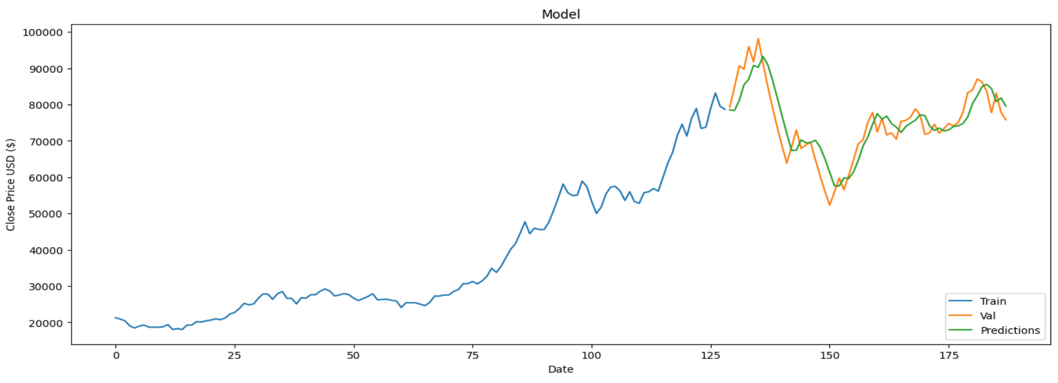


Stock prediction of NHH

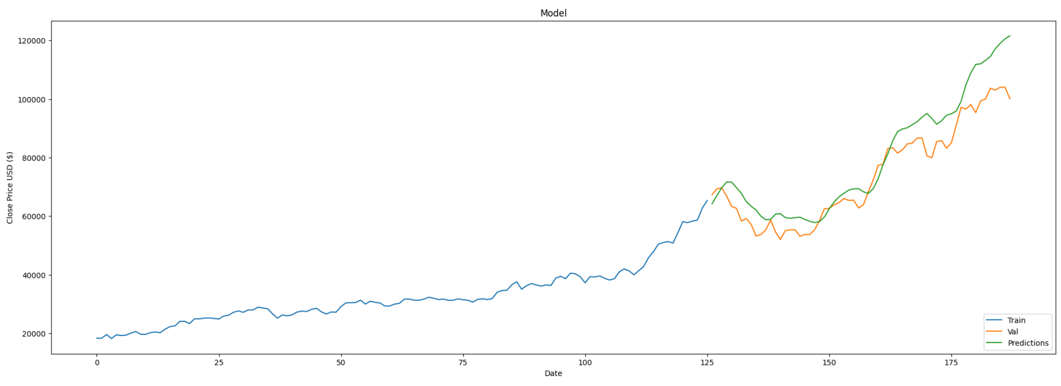


Stock prediction of TCM

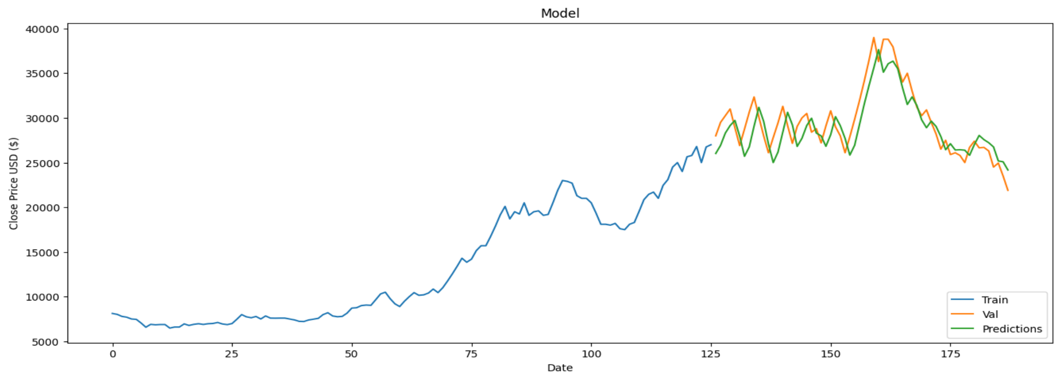
2.3.3.c Three stocks with the highest grow rate on VNINDEX (DIG, FRT, VRC)



Stock prediction of DIG



Stock prediction of FRT



Stock prediction of VRC

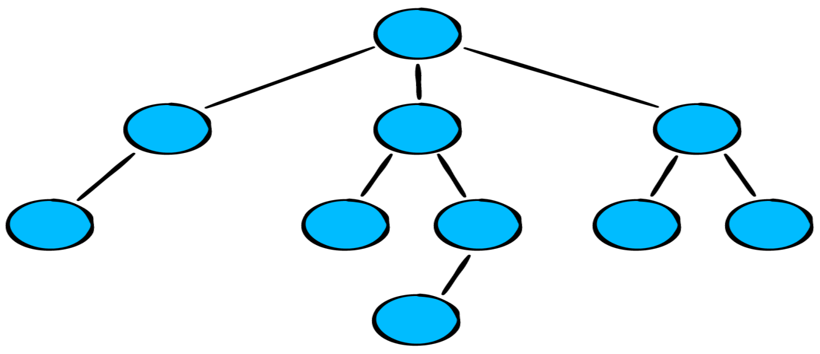
2.4 Question 4

In term of predictional models, there is a wide range of methods and procedures, of course our selection depends on the problems and the kinds of data we deal with. Stock market for instance, the reason why LSTM is suitable and fit in is that it can track back a huge amount of data without vanishing gradients. And in Python, there are many libraries that we can use in order to build our model properly.

In this section, we will use Decision Tree, a supervised learning in Machine Learning, used for handling regression and classification.

2.4.1 Mathematical background

A Decision Tree is a flowchart-like system, where each node denotes an attribute, and each branch shows an output and so on.



A load of data is learned by *splitting* into subsets based on their characteristic and value. This process is called *recursive partitioning*, it comes to an end whenever the subset at a node contains the same value of the target.

A decision tree is constructed by a decision tree inducer (classifier) such as: ID3, C4.5, CART... In stock market, the dataset comprises of Open, High, Low, Close and Volume (OHLCV), which is using CART algorithm in order to handle both classification and regression tasks.

Gini impurity is also used for splitting the dataset into a decision tree. Computation will be more accurate and less time-consuming, working on categorical variables such as OHLCV in stock market, it can provide outcomes either ’1’ or ’0’.

2.4.2 Code, implementation and model’s evaluation algorithm

We will take VNindex stock market as our testing model, after analysis and consider every factors or usability that affects to the prediction, we will apply it to other indicators and determine which models is the best and closest to reality.

First of all, we will prepare everything carefully for the model. We will take the VNIndex stock data for our model testing.

Listing 19: decision-tree-VNIndex.ipynb

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

from pandas import ExcelFile

df = pd.read_excel('VN Index Historical Data Tree.xlsx')
df2 = df['Price']
df2 = pd.DataFrame(df2)

future_days = 58
df2['Prediction'] = df2['Price'].shift(-future_days)

X = np.array(df2.drop(['Prediction'], 1))[:-future_days]
y = np.array(df2['Prediction'])[:-future_days]
```

Now the variable df2 contains the ’Price’ data of the VNIndex stock market. Using *numpy* in order to make an array contains every values after shifting 3 months prediction for the future.

Now the variables X, Y will contain the array which take the value of ’Price’ from oldest to newest.

Listing 20: decision-tree-VNIndex.ipynb

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
```

From library *sklearn*, we use train.test.split for transforming two arrays X, Y into test subsets. test_size stands for percentage of data we will test, the remaining used for training.

Then we using the sklearn in Python which is provided with Decision Tree and Linear Regression.

Listing 21: decision-tree-VNIndex.ipynb

```python
x_future = df2.drop(['Prediction'], 1)[:-future_days]
x_future = x_future.tail(future_days)
x_future = np.array(x_future)
tree_prediction = tree.predict(x_future)
lr_prediction = lr.predict(x_future)

predictions = tree_prediction
predictions_2 = lr_prediction
valid = df2[X.shape[0]:]
valid['Predictions'] = predictions
```

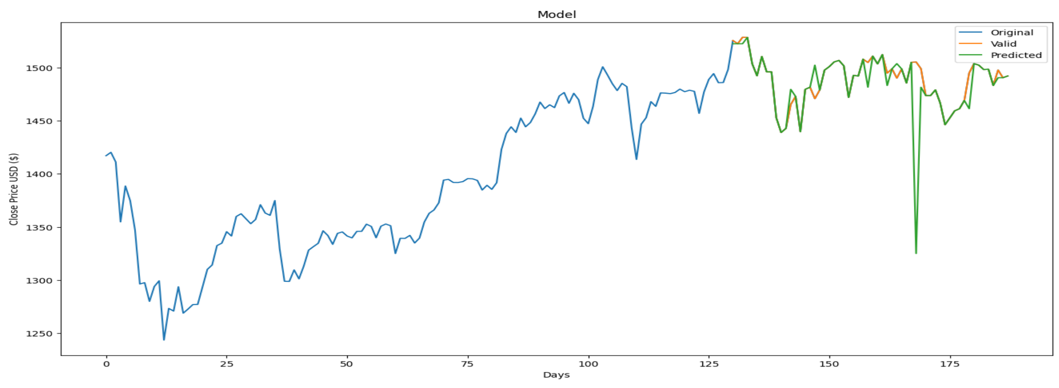
x_future represents the data we will predict by our models, as we has make a variable future_days, after loading it using *numpy*, we will store the value of predicted data into x_future. After executing the model into the x_future, the data will be hold in the variable predictions.

Now we plotting our result into a graph using mathplotlib.pyplot library, which the green line ('Predicted') represents the prediction of the model.

Listing 22: decision-tree-VNIndex.ipynb

```python
plt.figure(figsize=(16,8))
plt.title("Model")
plt.xlabel('Days')
plt.ylabel('Close Price USD ($)')
plt.plot(df2['Price'])
plt.plot(valid[['Price', 'Predictions']])
plt.legend(["Original", "Valid", 'Predicted'])
plt.show()
```

Each time the model is executed, the differences the prediction its become, this is because of the *overfitting*, when we try to predict as much precisely as possible, the leaf node will recursively tracking back again and again, so it can lead to overfitting and maybe sometime we will get a result like this.

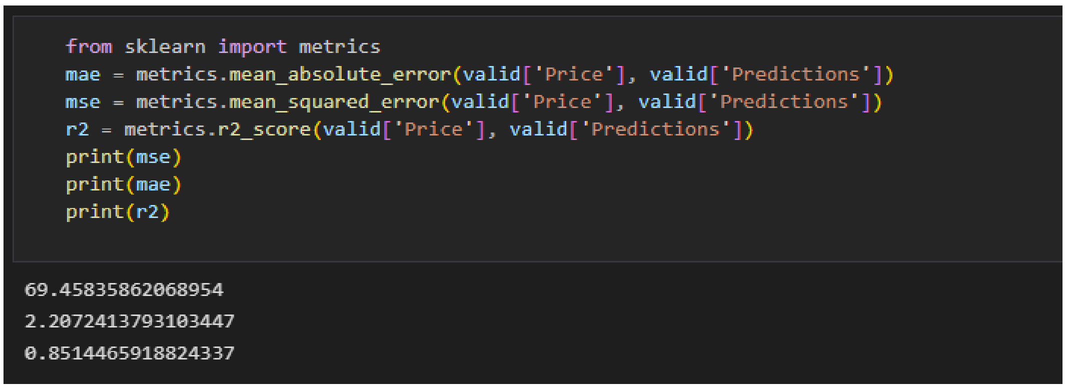


Tackling this problem, we can use *metrics* to evaluate the errors between the actual price and prediction.

Listing 23: decision-tree-VNIndex.ipynb

```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

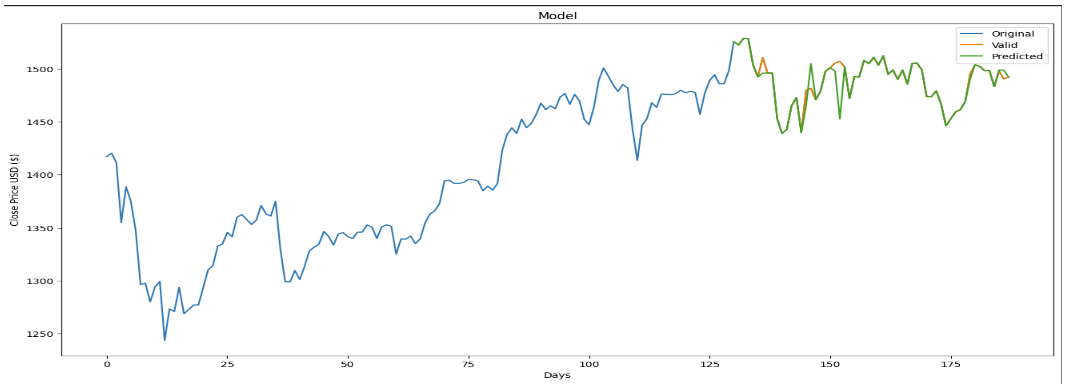
Executing over and over until the value r2 has reaches almost 1, its meaning that the models has been conducted with high accuracy.



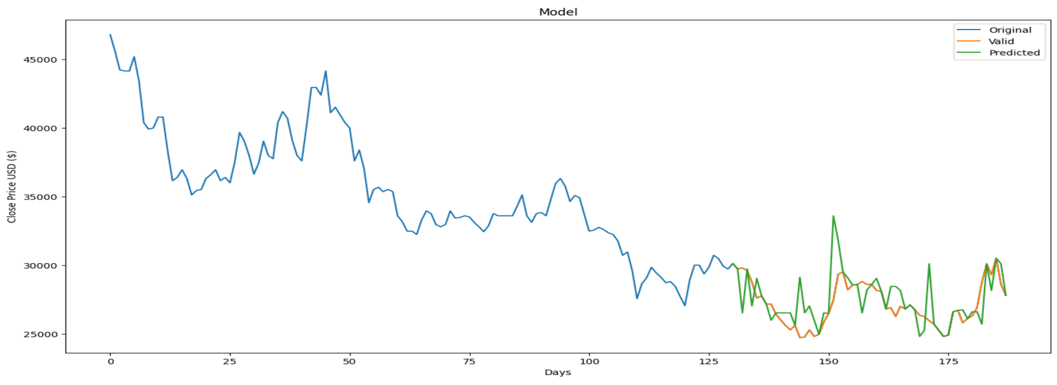
```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

```
69.45835862068954
2.2072413793103447
0.8514465918824337
```

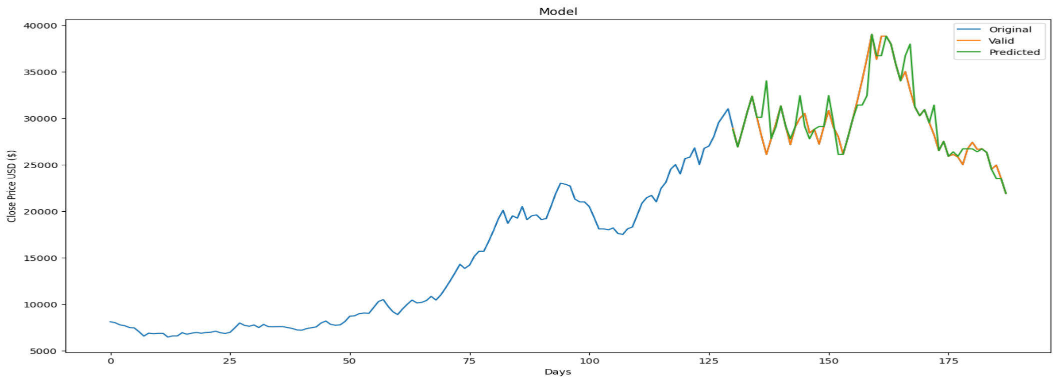
0.85 for instance, after a several execution, we got the result of the VNIndex stock market prediction as the picture below.



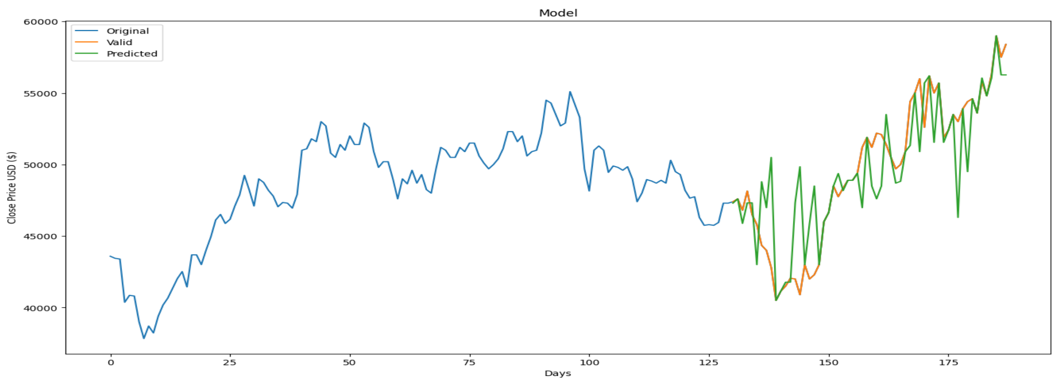
2.4.3 Applying the model into another indicators



Stock prediction of APH (highest drop indicator)



Stock prediction of VRC (highest growth indicator)



Stock prediction of Transportation

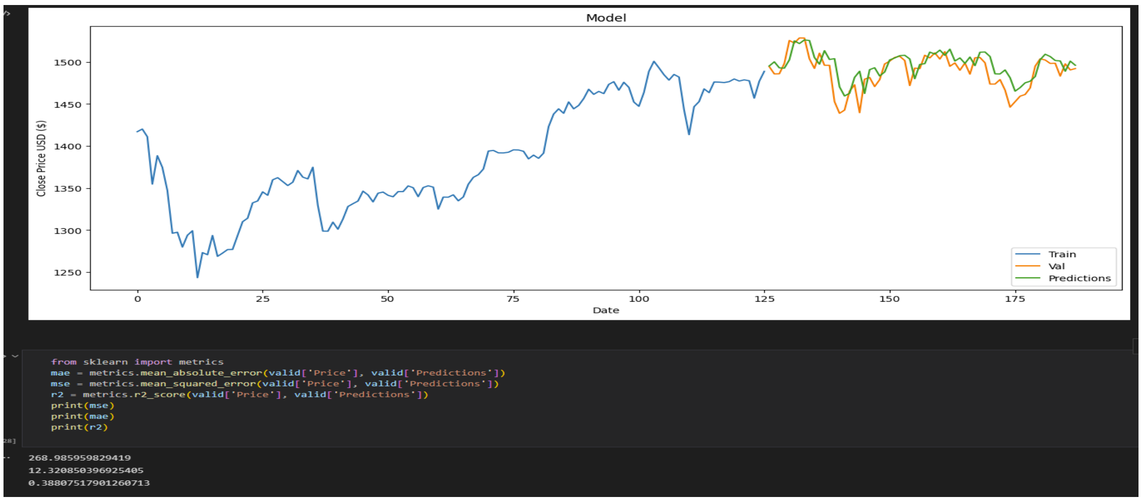
2.4.4 LSTM and Decision-tree, which one is better ?

In term of prediction accuracy, it is clear that Decision-tree prediction is not stable, because this model works by dividing the data in each node, so that there are a bunch of cases that the data is splitted (overfitting), using the model's evaluation algorithm we can easily find the flaw of the model.

Meanwhile, the model's evaluation algorithm when we use to tracking the accuracy of LSTM, the value is r_2 is always positive, which means the model has a proper prediction in compared with the Decision-tree model.

Now we going to make a comparison between two models using the evaluation algorithm or simply just look at the graph.

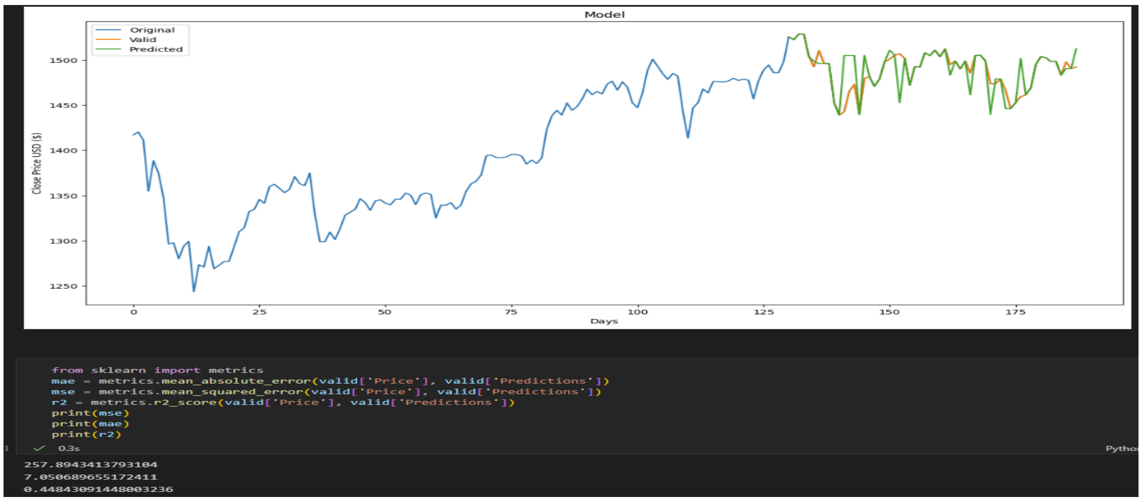
Three stocks are used for conducting: VNIndex, APH and VRC.



```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

```
268.985959829419
12.320850396925405
0.38807517901260713
```

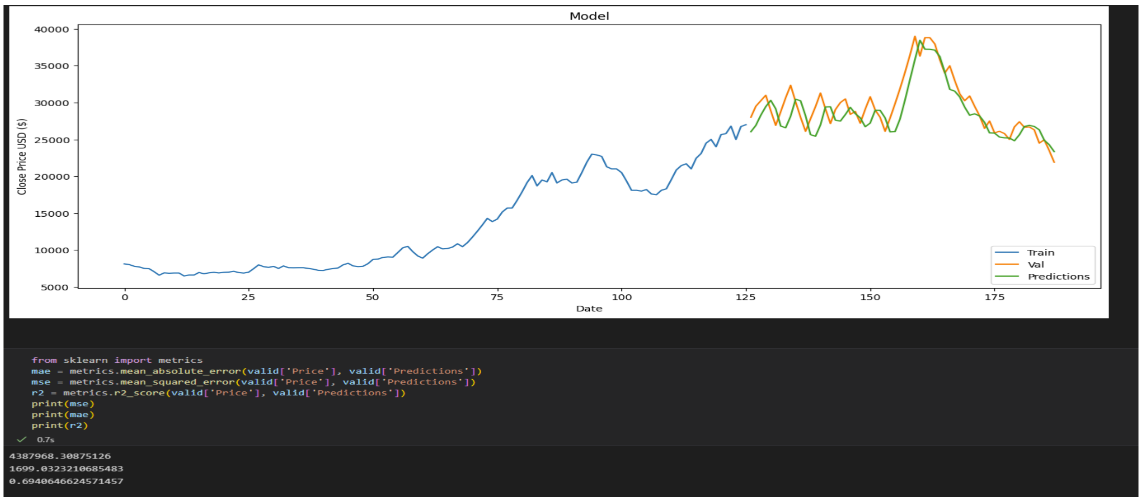
VNIndex using LSTM model



```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

```
257.8943413793104
7.050689655172411
0.44843091448003236
```

VNIndex using Decision-tree

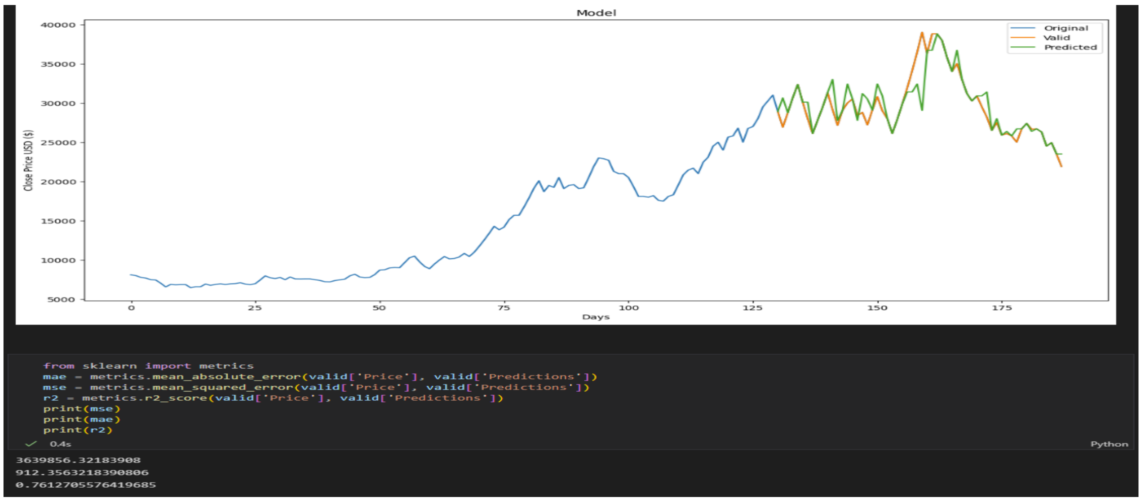


```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

✓ 0.7s

```
4387968.30875126
1699.0323210685483
0.6940646624571457
```

VRC using LSTM model

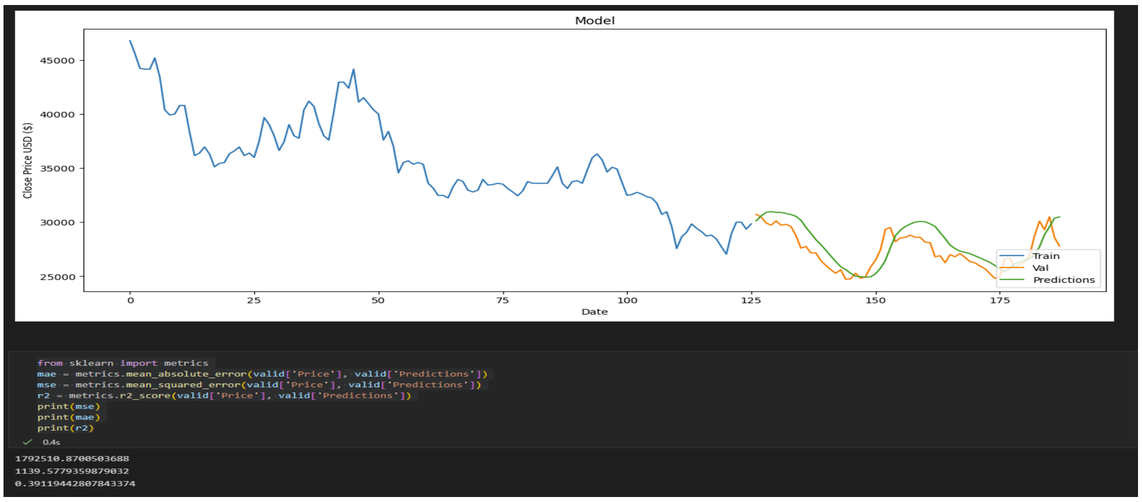


```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

✓ 0.4s

```
3639856.32183908
912.3563218390806
0.7612705576419685
```

VRC using Decision-tree

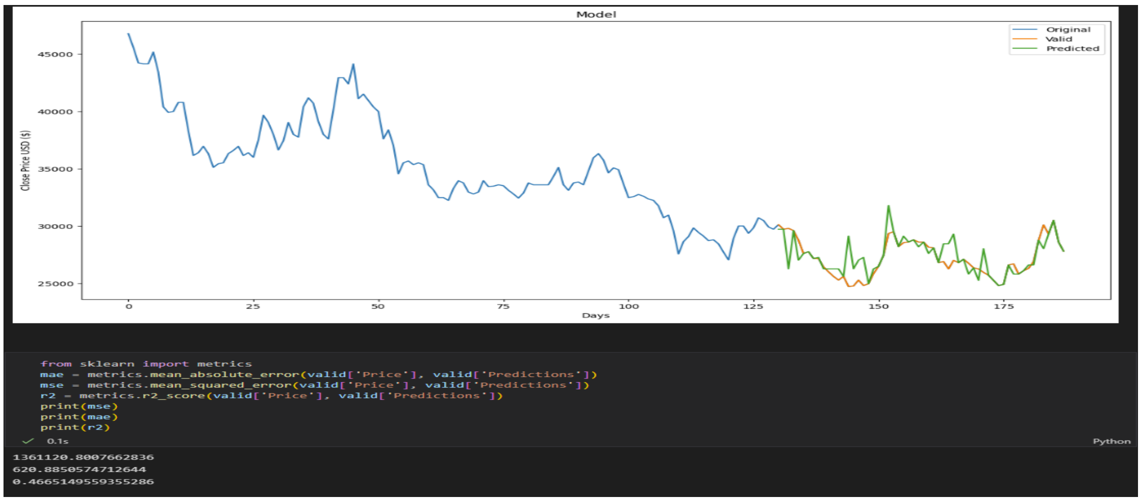


```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

✓ 0.4s

```
1792510.8700503688
1139.5779359879032
0.39119442807843374
```

APH using LSTM model



```
from sklearn import metrics
mae = metrics.mean_absolute_error(valid['Price'], valid['Predictions'])
mse = metrics.mean_squared_error(valid['Price'], valid['Predictions'])
r2 = metrics.r2_score(valid['Price'], valid['Predictions'])
print(mse)
print(mae)
print(r2)
```

✓ 0.1s

```
1361120.8007662836
620.8850574712644
0.4665149559355286
```

APH using Decision-tree

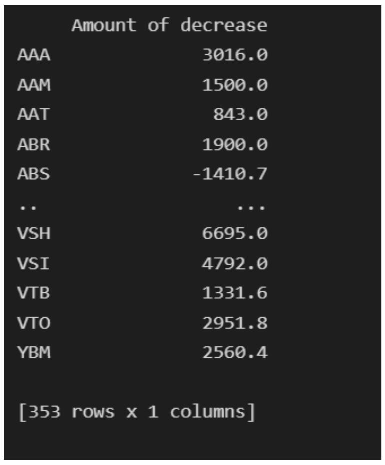
2.5 Question 5

2.5.1 How to find out 3 stocks affected negatively by the Economic Crisis

Using the formula: Amount of change = percentage change * the first day's price

Listing 24: Create dataframe for amount of decrease data

```
        decrease = np.array([1.1 for i in range(0,len(label))])
        i = 0
        for name in label:
            decrease[i] = stocks[name]['Price'][0] * percentage_change['
                Percentage change'][i]
            i = i + 1
        decrease_data = pd.DataFrame(data = decrease, index = [name for name
            in label], columns= ['Amount of decrease'])
        print(decrease_data)
```

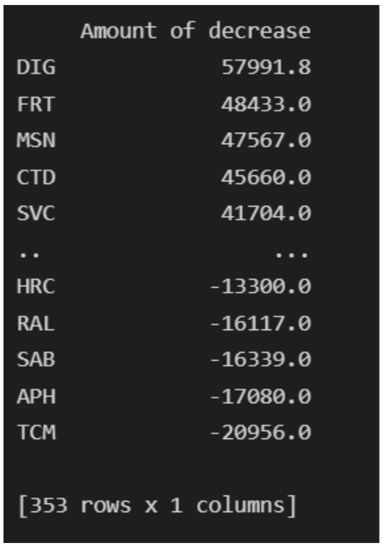


```
     Amount of decrease
AAA              3016.0
AAM              1500.0
AAT               843.0
ABR              1900.0
ABS             -1410.7
..                  ...
VSH              6695.0
VSI              4792.0
VTB              1331.6
VTO              2951.8
YBM              2560.4

[353 rows x 1 columns]
```

Hình 45: Amount of decrease dataframe

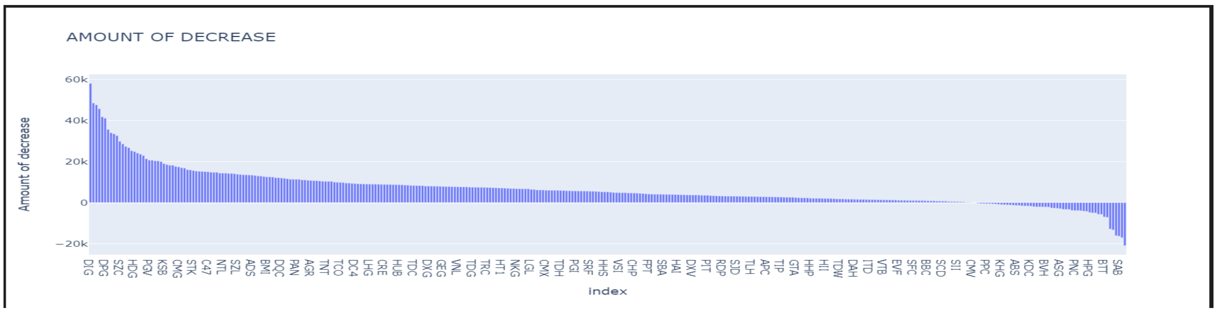
Listing 25: Sorting the amount of decrease dataframe

```
        new_label = np.copy(label)
        new_decrease = np.copy(decrease)
        for i in range(0,len(label) - 1):
        for j in range(i+1, len(label)):
        if(new_decrease[i] < new_decrease[j]):
            swap = new_decrease[i]
            new_decrease[i] = new_decrease[j]
            new_decrease[j] = swap
            new_swap = new_label[i]
            new_label[i] = new_label[j]
            new_label[j] = new_swap
        sorted_decrease_data = pd.DataFrame(data = new_decrease, index = [name
             for name in new_label], columns= ['Amount of decrease'])
        print(sorted_decrease_data)
```



```
     Amount of decrease
DIG             57991.8
FRT             48433.0
MSN             47567.0
CTD             45660.0
SVC             41704.0
..                  ...
HRC            -13300.0
RAL            -16117.0
SAB            -16339.0
APH            -17080.0
TCM            -20956.0

[353 rows x 1 columns]
```

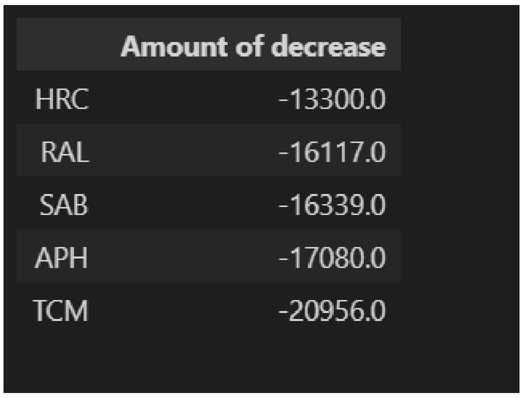
Hình 46: Sorted amount of decrease dataframe



Hình 47: Sorted amount of decrease dataframe chart

Listing 26: Printing out the 5 lowest

```
sorted_decrease_data.tail(5)
```



	Amount of decrease
HRC	-13300.0
RAL	-16117.0
SAB	-16339.0
APH	-17080.0
TCM	-20956.0

Hình 48: The 5 biggest amount of drop

- Through the data we find out:
TCM: Thanh Cong Textile - Investment - Trading Joint Stock Company (Personal good)
APH: An Phat Holdings Group Joint Stock Company (Industrial service)
SAB: Saigon Beer - Alcohol - Beverage Joint Stock Corporation (Food and beverage)
- Their product are all not necessary good during COVID 19 pandemic and the economic was also being damaged a lot so investing business of those company decrease heavily. Overall service and non-necessary good are the most drop
- The chance for those companies is after pandemic, they will grow rapidly again because of the high demand of market and the recovery of the economic