1 Exploring Data

First We have to analyze the data to see what's happening inside it.

In [372]:

```
import os
import sys
import time
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
from statsmodels.graphics.tsaplots import plot_acf
import datetime
import seaborn as sns
import statsmodels.api as sm
import csv
from tqdm import tqdm, tqdm_notebook
from numpy import loadtxt
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Activatio
from tensorflow.python.keras import backend as k
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStoppi
from tensorflow.keras.optimizers import Adam, SGD, Nadam
from tensorflow.python.keras.layers.normalization import BatchNorma
from tensorflow.python.keras.layers.advanced_activations import Lea
exch = pd.read_csv('EXCHANGE.csv')
kos1 = pd.read_csv('KOSPI_1.csv')

import platform
from matplotlib import font_manager, rc
if platform.system() == 'Windows':
    font_name = font_manager.FontProperties(fname="C:/Windows/Fonts
    rc('font', family = font_name)
else:
    rc('font', family = "AppleGothic")
```

KOSPI_1 is dataset that has stock market price and other related variables. EXCHANGE dataset is about the price of foreign currency compared with KRW.

In [225]:

```
kos1.tail()
```

Out[225]:

	날짜	지수시가	지수고가	지수저가	지수종가	거래량	거래대금	싱
4444	12/21/18	2052.70	2061.51	2049.76	2061.49	311388800	5492537	51
4445	12/24/18	2050.38	2059.94	2046.18	2055.01	285275000	3843849	51
4446	12/26/18	2028.81	2037.83	2014.28	2028.01	321499300	5424078	52
4447	12/27/18	2032.09	2035.57	2021.39	2028.44	398021300	5351003	52
4448	12/28/18	2036.70	2046.97	2035.41	2041.04	352677700	4120695	52

5 rows × 30 columns

Column 2 to 5 is about the KOSPI price of that day. Open, High, Low, End price from left to right.

In [226]:

```
kos1.iloc[:,0] = pd.to_datetime(kos1.iloc[:,0])
```

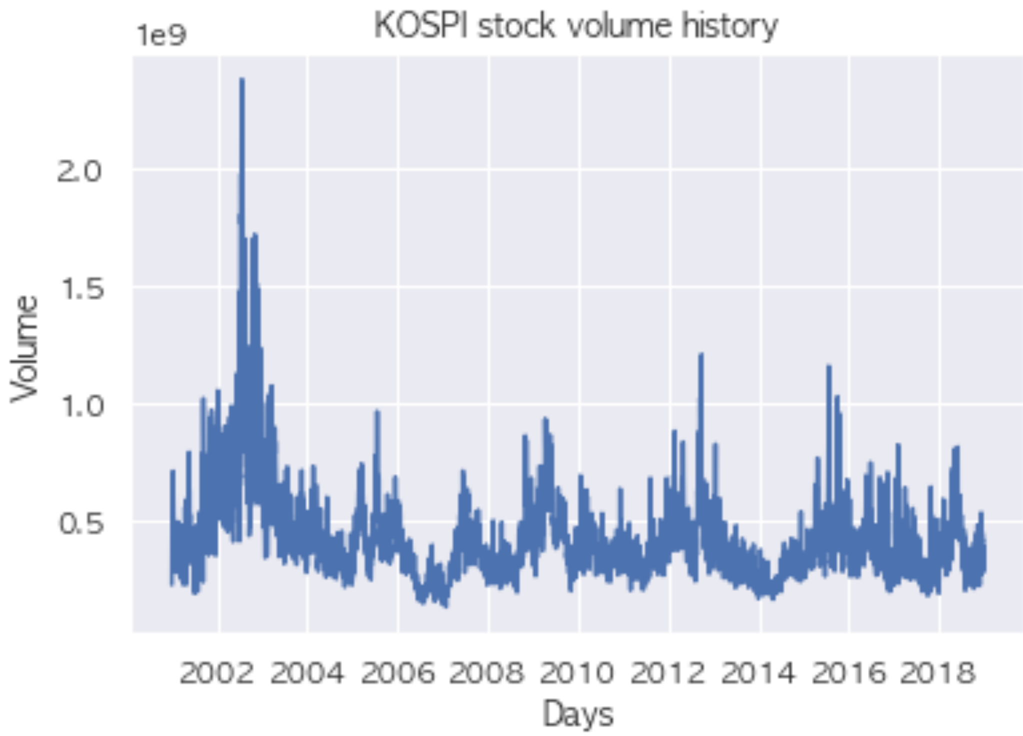
In [227]:

```
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.plot(kos1['날짜'],kos1['지수시가'], label = 'Open', color = 'black'
ax1.plot(kos1['날짜'],kos1['지수고가'], label = 'High', color = 'red')
ax1.plot(kos1['날짜'],kos1['지수저가'], label = 'Low', color = 'blue')
ax1.plot(kos1['날짜'],kos1['지수종가'], label = 'End', color = 'green')
ax1.legend(loc = 'best')
plt.show()
```

As you can see in the plot, prices of each day don't vary too much from each other.

In [228]:

```
plt.figure()
plt.plot(kos1['날짜'],kos1['거래량'])
plt.title('KOSPI stock volume history')
plt.ylabel('Volume')
plt.xlabel('Days')
plt.show()
```



There is a significant increase of volume in 2002 to 2003. However, this doesn't seem to be related to the change of price.

In [229]:

```
print("checking if any null values are present\n", kos1.isna().sum(
```

```
checking if any null values are present
 날짜              0
지수시가            0
지수고가            0
지수저가            0
지수종가            0
거래량             0
거래대금            0
상장주식수           0
시가총액            0
자본금             0
외국인보유주식수        0
외국인보유시가총액       0
신용거래종목수         0
신용가능종목거래량       0
신용자료일자          1
전체종목수           0
회사수             0
거래형성종목수         0
상승종목수           0
하락종목수           0
보합종목수           0
상한종목수           0
하한종목수           0
연중최고가종목수        0
연중최저가종목수        0
25일이평상회종목건수     0
25일이평하회종목건수     0
52주신고가종목수       0
52주신저가종목수       0
배당 수익율          0
dtype: int64
```

In [230]:

```
print("checking if any 0 values are present\n", (kos1 == 0).sum(axi
```

```
checking if any 0 values are present
 날짜              0
지수시가            0
지수고가            0
지수저가            0
지수종가            0
거래량             0
거래대금            0
상장주식수           0
시가총액            0
자본금             0
외국인보유주식수        0
외국인보유시가총액       0
신용거래종목수        41
신용가능종목거래량      41
신용자료일자          0
전체종목수           0
회사수             0
거래형성종목수         0
상승종목수           0
하락종목수           0
보합종목수           0
상한종목수         300
하한종목수        1521
연중최고가종목수     3549
연중최저가종목수     3551
25일이평상회종목건수      0
25일이평하회종목건수      0
52주신고가종목수      149
52주신저가종목수      250
배당 수익율       2949
dtype: int64
```

There is only one NA value which is good. There are lots of 0 values in the data, but they are not equivalent to NA values. So, I'll just omit the columns that has more than 2000 counts of 0.

In [231]:

```
exch.tail()
```

Out[231]:

	날짜	USD	EUR	CNY	JPY	GBP
4762	12/27/18	1125.6	1278.29	163.20	1012.23	1422.93
4763	12/28/18	1121.3	1281.81	162.60	1010.95	1417.88
4764	12/29/18	1121.3	1281.81	162.60	1010.95	1417.88
4765	12/30/18	1121.3	1281.81	162.60	1010.95	1417.88
4766	12/31/18	1118.1	1279.16	162.76	1013.18	1420.32

In [232]:

```
print("checking if any null values are present\n", exch.isna().sum(
```

```
checking if any null values are present
 날짜         0
USD        0
EUR      220
CNY     1193
JPY       72
GBP     1012
dtype: int64
```

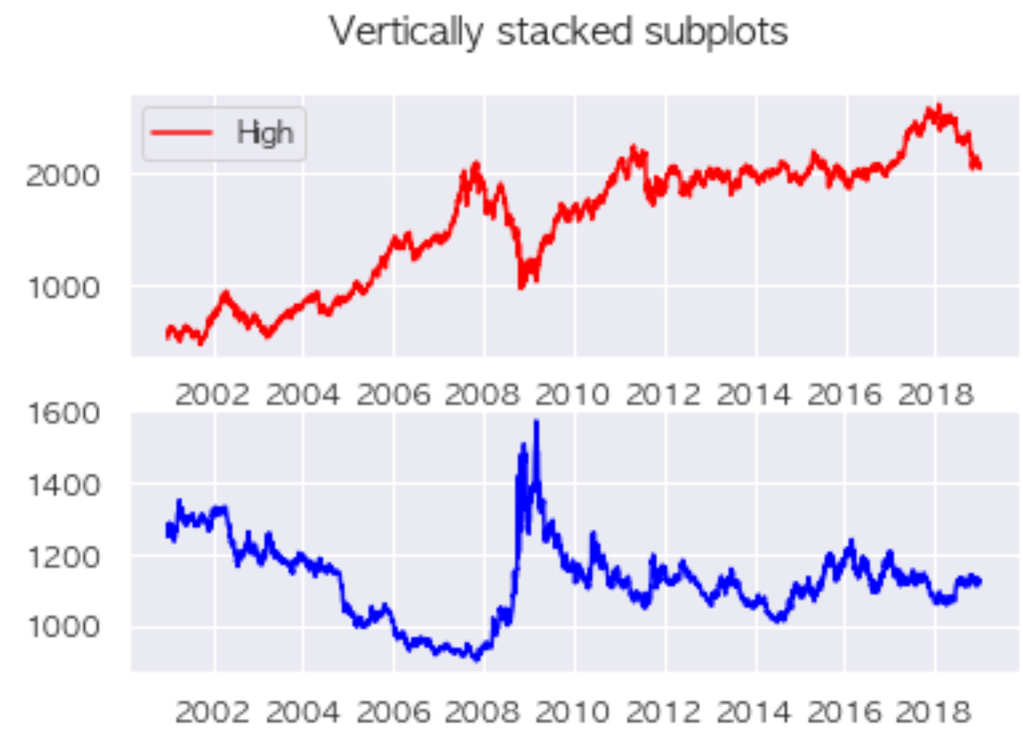
There are so many NA values in the dataset which makes me confuse. I'll just use USD that has no NA values.

In [233]:

```
exch.iloc[:,0] = pd.to_datetime(exch.iloc[:,0])
plt.figure()
plt.plot(exch['날짜'], exch['USD'])
plt.title('USD history')
plt.ylabel('USD')
plt.xlabel('Days')
plt.show()
```

In [234]:

```
fig, (ax1, ax2) = plt.subplots(2, 1)
fig.suptitle('Vertically stacked subplots')
ax1.plot(kos1['날짜'], kos1['지수고가'], label = 'High', color = 'red')
ax2.plot(exch['날짜'], exch['USD'],  label = 'USD', color = 'blue')
ax1.legend(loc = 'best')
plt.show()
```



There is a clear negative correlation between KOSPI and USD. As USD value gets higher, KOSPI value gets lower.

2 Feature Engineering

In [235]:

```
kos1 = kos1.drop(['연중최고가종목수','연중최저가종목수','배당 수익율','신용자료일ᄌ
```

Here I first dropped the columns that have too many 0 values.

In [236]:

```
exch = exch.drop(['EUR','CNY','JPY','GBP'], axis = 1)
exch.iloc[:,0] = pd.to_datetime(exch.iloc[:,0])
```

Since USD is a main currency that has major effect on Korea's economy, I'll just use USD for the currency variable.

In [282]:

```
merged_kos = pd.merge(left=kos1,right=exch, left_on='날짜', right_on=
merged_kos.isnull().any()
```

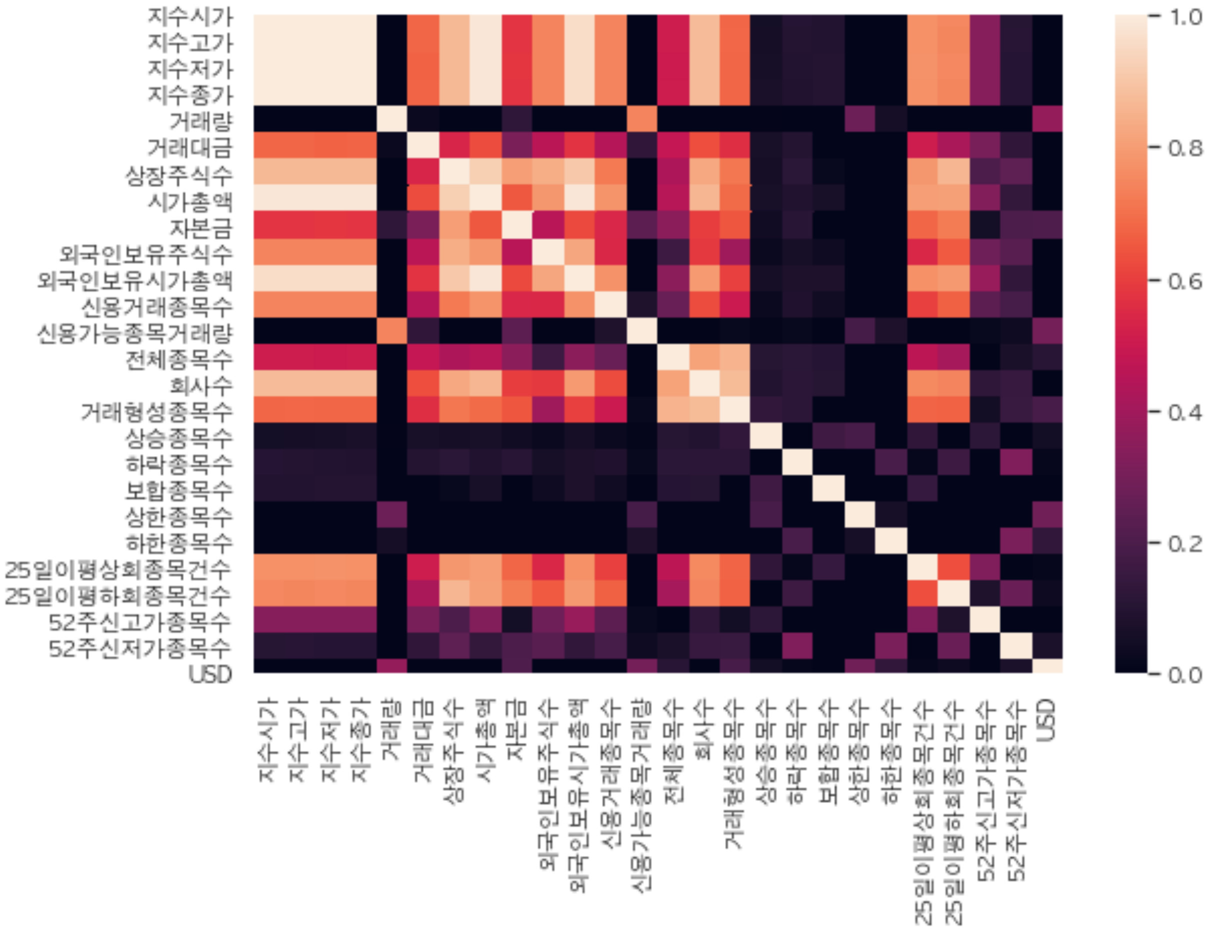
Out[282]:

```
날짜               False
지수시가             False
지수고가             False
지수저가             False
지수종가             False
거래량              False
거래대금             False
상장주식수            False
시가총액             False
자본금              False
외국인보유주식수         False
외국인보유시가총액        False
신용거래종목수          False
신용가능종목거래량        False
전체종목수            False
회사수              False
거래형성종목수          False
상승종목수            False
하락종목수            False
보합종목수            False
상한종목수            False
하한종목수            False
25일이평상회종목건수      False
25일이평하회종목건수      False
52주신고가종목수        False
52주신저가종목수        False
USD              False
dtype: bool
```

And merge the USD column into the main dataset with equivalent date.

In [283]:

```
sns.set(font = "AppleGothic")
f, ax = plt.subplots(figsize=(9, 6))
ax = sns.heatmap(merged_kos.corr(), vmin= 0, vmax=1)
```



In [284]:

```
merged_kos = merged_kos.drop(['거래량','신용가능종목거래량','상승종목수','하락
```

As the color gets darker, it means that the correlation gets smaller. So I dropped the dark columns except USD.

In [285]:

```
merged_kos.index = merged_kos['날짜']
merged_kos = merged_kos.drop(['날짜'], axis = 1)
```

Make a time column as an index.

In [286]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
```

3 Modeling

LSTM Model with time step = 5

I'm going to make a model that retrieve past 5 values and predict from it.

In [243]:

```
scaler = MinMaxScaler()
kos1[['지수종가']] = scaler.fit_transform(kos1[['지수종가']])
```

In [244]:

```
price = kos1[['지수종가']].values.tolist() #지수종가 데이터를 price라는 변수
```

First normalize the end price.

In [245]:

```
window_size = 5
x = []
y = []

for i in range(len(price) - window_size):
    x.append([price[i + j] for j in range(window_size)])
    y.append(price[window_size + i])
```

Make time step as 5.

In [246]:

```
x = np.asarray(x)
y = np.asarray(y)
```

In [253]:

```
x.shape
```

Out[253]:

```
(4444, 5, 1)
```

In [247]:

```
len(x)
```

Out[247]:

```
4444
```

In [248]:

```
train_test_split = 4205

x_train = x[:train_test_split, :]
y_train = y[:train_test_split]


x_test = x[train_test_split:, :]
y_test = y[train_test_split :]

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

print(x_test.shape)

print(x_train[0])
```

```
(239, 1, 5)
[[0.02450891 0.02473432 0.04191732 0.0526385  0.055362
23]]
```

In [249]:

```
model = Sequential()
model.add(LSTM(50, return_sequences = True, input_shape = (1, 5)))
model.add(LSTM(64, return_sequences = False))
model.add(Dense(1, activation = 'linear'))
model.compile(loss = 'mse', optimizer = 'rmsprop')
model.summary()
```

```
Model: "sequential_17"
______________________________________________________________________________
Layer (type)                 Output Shape
Param #
=================================================================
lstm_41 (LSTM)               (None, 1, 50)
11200
______________________________________________________________________________
lstm_42 (LSTM)               (None, 64)
29440
______________________________________________________________________________
dense_91 (Dense)             (None, 1)
65
=================================================================
Total params: 40,705
Trainable params: 40,705
Non-trainable params: 0
______________________________________________________________________________
```

In [250]:

```
model.fit(x_train, y_train, epochs = 100, batch_size = 1)
```

```
Train on 4205 samples
Epoch 1/100
4205/4205 [==============================] - 31s 7ms/s
ample - loss: 0.0018
Epoch 2/100
4205/4205 [==============================] - 23s 6ms/s
ample - loss: 3.8913e-04
Epoch 3/100
4205/4205 [==============================] - 25s 6ms/s
ample - loss: 3.3435e-04
Epoch 4/100
4205/4205 [==============================] - 22s 5ms/s
ample - loss: 2.8415e-04
Epoch 5/100
4205/4205 [==============================] - 25s 6ms/s
ample - loss: 2.5454e-04
Epoch 6/100
4205/4205 [==============================] - 22s 5ms/s
ample - loss: 2.3186e-04
Epoch 7/100
```

In [251]:

```
test_predict = model.predict(x_test)

plt.figure(figsize=(20,10))
plt.plot(price[train_test_split :])

split_pt = train_test_split + window_size
plt.plot(test_predict, color='r')
```

Out[251]:

```
[<matplotlib.lines.Line2D at 0x1abf927c88>]
```

It seems like predicted value is almost as same as actual value. Since I've tried time step as 5, let's see if there's any difference as we change time step as 30.

LSTM Model with time step = 30

I'm going to extract past 30 values and predict from it.

In [254]:

```
window_size = 30
x = []
y = []

for i in range(len(price) - window_size):
    x.append([price[i + j] for j in range(window_size)])
    y.append(price[window_size + i])
```

In [255]:

```
x = np.asarray(x)
y = np.asarray(y)
```

In [256]:

```
x.shape
```

Out[256]:

```
(4419, 30, 1)
```

In [257]:

```
len(x)
```

Out[257]:

```
4419
```

In [258]:

```
train_test_split = 4175

x_train = x[:train_test_split, :]
y_train = y[:train_test_split]



x_test = x[train_test_split:, :]
y_test = y[train_test_split :]

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

print(x_test.shape)

print(x_train[0])
```

```
(244, 1, 30)
[[0.02450891 0.02473432 0.04191732 0.0526385  0.055362
23 0.05689786
  0.04322753 0.04368775 0.05593516 0.06116191 0.063237
58 0.05967325
  0.06353343 0.07092039 0.07452229 0.05774785 0.060006
67 0.0575647
  0.07004222 0.06740771 0.0656138  0.05184486 0.055329
36 0.05045012
  0.05767271 0.05950419 0.06125583 0.06105859 0.063430
12 0.0633362 ]]
```

In [261]:

```
model = Sequential()
model.add(LSTM(50, return_sequences = True, input_shape = (1, 30)))
model.add(LSTM(64, return_sequences = False))
model.add(Dense(1, activation = 'linear'))
model.compile(loss = 'mse', optimizer = 'rmsprop')
model.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_45 (LSTM)               (None, 1, 50)             16200
_________________________________________________________________
lstm_46 (LSTM)               (None, 64)                29440
_________________________________________________________________
dense_93 (Dense)             (None, 1)                 65
=================================================================
Total params: 45,705
Trainable params: 45,705
Non-trainable params: 0
_________________________________________________________________
```

In [262]:

```python
model.fit(x_train, y_train, epochs = 100, batch_size = 1)
```

```
Train on 4175 samples
Epoch 1/100
4175/4175 [==============================] - 28s 7ms/s
ample - loss: 0.0016
Epoch 2/100
4175/4175 [==============================] - 19s 4ms/s
ample - loss: 7.1309e-04
Epoch 3/100
4175/4175 [==============================] - 18s 4ms/s
ample - loss: 5.8189e-04
Epoch 4/100
4175/4175 [==============================] - 19s 4ms/s
ample - loss: 5.0523e-04
Epoch 5/100
4175/4175 [==============================] - 18s 4ms/s
ample - loss: 4.5896e-04
Epoch 6/100
4175/4175 [==============================] - 19s 4ms/s
ample - loss: 4.3146e-04
Epoch 7/100
```

In [266]:

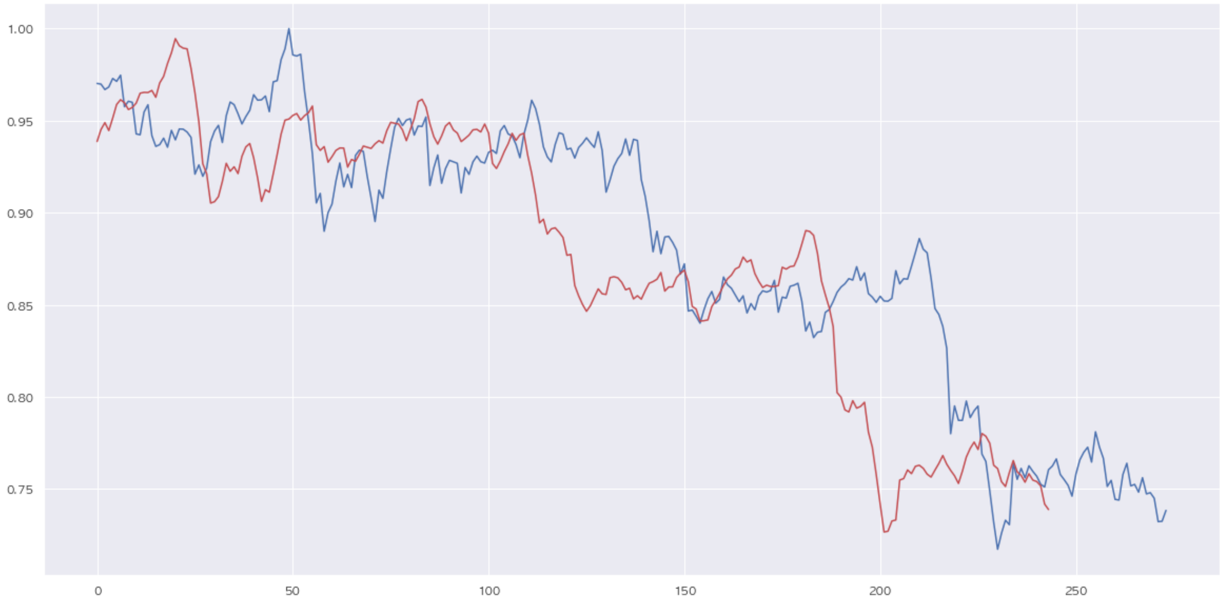
```
test_predict = model.predict(x_test)

plt.figure(figsize=(20,10))
plt.plot(price[train_test_split :])

split_pt = train_test_split + window_size
plt.plot(test_predict, color='r')
```

Out[266]:

```
[<matplotlib.lines.Line2D at 0x1ab70036d8>]
```



The prediction from time step = 30 almost seems like the same as prediction from the previous model.

In [ ]:

```

```