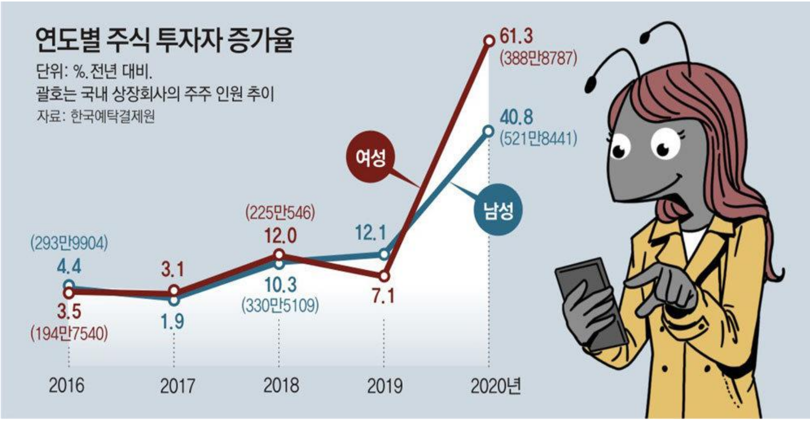
Forecasting the stock price of Apple lnc.

20201181 오지환

Introduction

Stocks are a type of investment, and many people now invest in them. As the importance of investment increases further in the future, the number of people investing in stocks is increasing rapidly. I am interested in the stock market because I like the economy and finance. Stock prices are driven by a wide variety of factors. People analyze a company's financial statements, news articles, etc. and decide whether to buy its stocks or not. Of course, some people buy stocks without any analysis. In some cases, people just buy stocks cause others buy same stocks. Like this, people's psychological factors are combined to determine stock prices. For this reason, I often wonder that "Can we really predict the price of a stock that changes from time to time?“

So, I decided to forecast Apple's stock price using various forecasting methods that I learned in class. In addition, I applied Deep Learning (LSTM)models that I learned from AIP2 with Python to this topic. The reason why I chose Apple Inc. among many companies is that it would have the highest market capitalization in the world, so the stock fluctuation rate would be small and easy to predict. I will use Root Mean Squared Error (RMSE) as an indicator to determine Forecasting accuracy. Because I think RMSE is the most reasonable indicator of error as mean squared deviation and it has simple form.

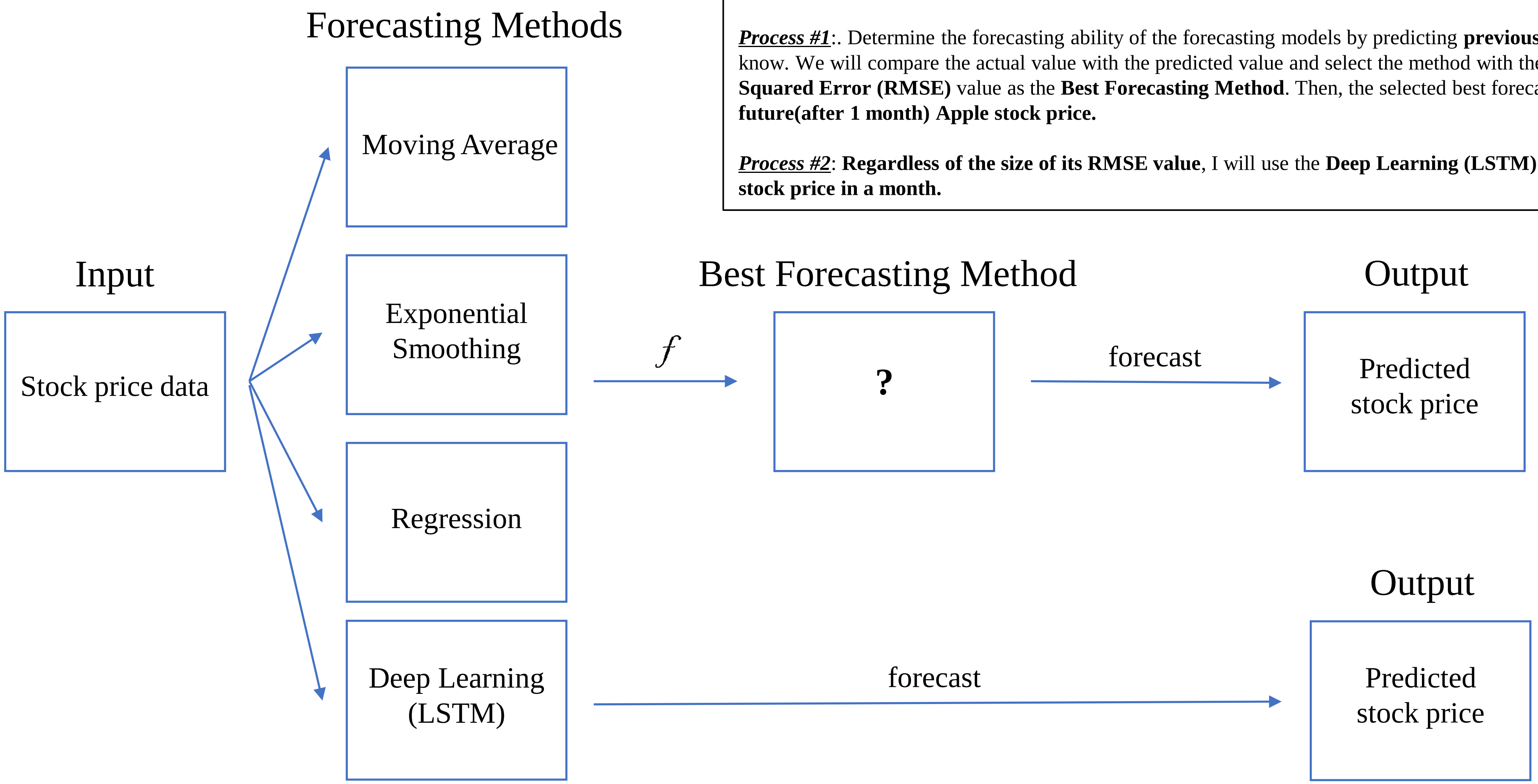


Project Process

A total of four forecasting models, **Moving Average, Exponential Smoothing, Regression, and Deep Learning (LSTM: Long Short-Term Memory)**, will be used to forecasting. It can be divided into two main processes.

Process #1:. Determine the forecasting ability of the forecasting models by predicting **previous stock data** that we already know. We will compare the actual value with the predicted value and select the method with the smallest **Root Mean Squared Error (RMSE)** value as the **Best Forecasting Method**. Then, the selected best forecasting method will **forecast future(after 1 month) Apple stock price.**

Process #2: **Regardless of the size of its RMSE value**, I will use the **Deep Learning (LSTM)** method **to forecast Apple's stock price in a month.**



f : Find the best forecasting model with the smallest RMSE(Root Mean Squared Error) value. $RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$

Problem & Used Data

<u>Problem definition</u>

: The purpose of this project is to **forecast the stock price of Apple Inc. in the future(after 1 month)**. To improve the accuracy of the forecasting, I will **find best forecasting model with the smallest RMSE value**.

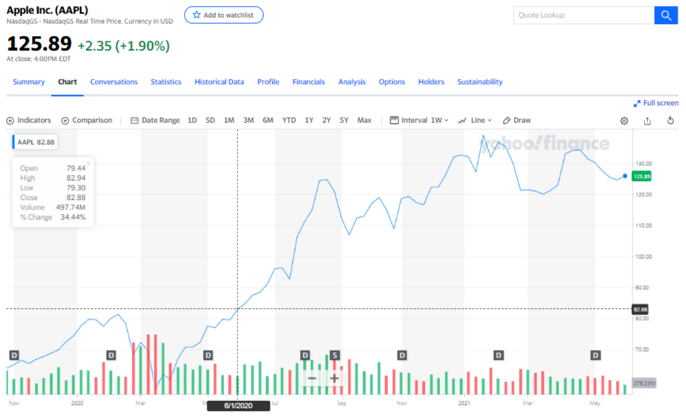
<u>Used Data</u>

: Apple Inc. stock price data from https://finance.yahoo.com/quote/AAPL?p=AAPL&.tsrc=fin-srch

In Moving Average, Exponential Smoothing and Regression models, I used **monthly** Apple Inc. stock price(close) data from **2010-01-01 to 2021-06-01. =** total 140 stock price data

In LSTM: Long Short-Term Memory model, I used **daily** Apple Inc. stock price data(close) from **2015-01-01 to 2021-06-06**. =total 2372 stock price data

* Stock price(close) means the price at the close of the stock market.



Daily data

Time Period: Jan 01, 2012 - Jun 06, 2021 Show: Historical Prices Frequency: Daily Apply

Currency in USD Download

Date	Open	High	Low	Close*	Adj Close**	Volume
Jun 04, 2021	124.07	126.16	123.85	125.89	125.89	75,087,300
Jun 03, 2021	124.68	124.85	123.13	123.54	123.54	76,229,200
Jun 02, 2021	124.28	125.24	124.05	125.06	125.06	59,278,900
Jun 01, 2021	125.08	125.35	123.94	124.28	124.28	67,637,100
May 28, 2021	125.57	125.80	124.55	124.61	124.61	71,311,100
May 27, 2021	126.44	127.64	125.08	125.28	125.28	94,625,600
May 26, 2021	126.96	127.39	126.42	126.85	126.85	56,575,900
May 25, 2021	127.82	128.32	126.32	126.90	126.90	72,009,500
May 24, 2021	126.01	127.94	125.94	127.10	127.10	63,092,900
May 21, 2021	127.82	128.00	125.21	125.43	125.43	79,295,400
May 20, 2021	125.23	127.72	125.10	127.31	127.31	76,857,100
May 19, 2021	123.16	124.92	122.86	124.69	124.69	92,612,000
May 18, 2021	126.56	126.99	124.78	124.85	124.85	63,342,900
May 17, 2021	126.82	126.93	125.17	126.27	126.27	74,244,600
May 14, 2021	126.25	127.89	125.85	127.45	127.45	81,806,500
May 13, 2021	124.58	126.15	124.26	124.97	124.97	105,861,300
May 12, 2021	123.40	124.64	122.25	122.77	122.77	112,172,300
May 11, 2021	123.50	126.27	122.77	125.91	125.91	126,142,800

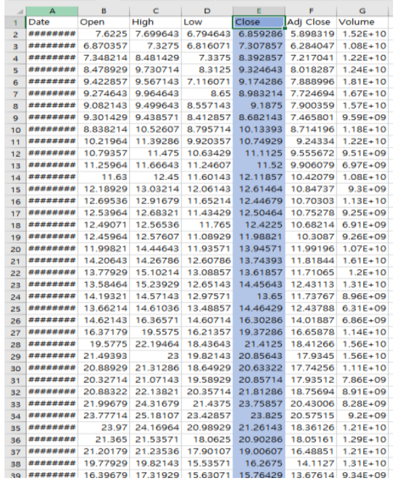
Monthly data

Time Period: Jan 01, 2012 - Jun 06, 2021 Show: Historical Prices Frequency: Monthly Apply

Currency in USD Download

Date	Open	High	Low	Close*	Adj Close**	Volume
Jun 04, 2021	124.07	126.16	123.85	125.89	125.89	75,169,343
Jun 01, 2021	125.08	126.16	123.13	125.89	125.89	278,232,500
May 07, 2021	**0.22** Dividend					
May 01, 2021	132.04	134.07	122.25	124.61	124.40	1,711,742,800
Apr 01, 2021	123.66	137.07	122.49	131.46	131.24	1,889,731,200
Mar 01, 2021	123.75	128.72	116.21	122.15	121.94	2,650,418,200
Feb 05, 2021	**0.205** Dividend					
Feb 01, 2021	133.75	137.88	118.39	121.26	120.87	1,833,855,600
Jan 01, 2021	133.52	145.09	126.38	131.96	131.54	2,240,262,000
Dec 01, 2020	121.01	138.79	120.01	132.69	132.27	2,322,189,600
Nov 06, 2020	**0.205** Dividend					
Nov 01, 2020	109.11	121.99	107.32	119.05	118.47	2,123,077,300
Oct 01, 2020	117.64	125.39	107.72	108.86	108.33	2,894,666,500
Sep 01, 2020	132.76	137.98	103.10	115.81	115.24	3,885,245,100
Aug 31, 2020	**4:1** Stock Split					
Aug 07, 2020	**0.205** Dividend					
Aug 01, 2020	108.20	131.00	107.89	129.04	128.18	4,070,061,100
Jul 01, 2020	91.28	106.42	89.14	106.26	105.55	3,020,283,200
Jun 01, 2020	79.44	93.10	79.30	91.20	90.59	3,243,375,600

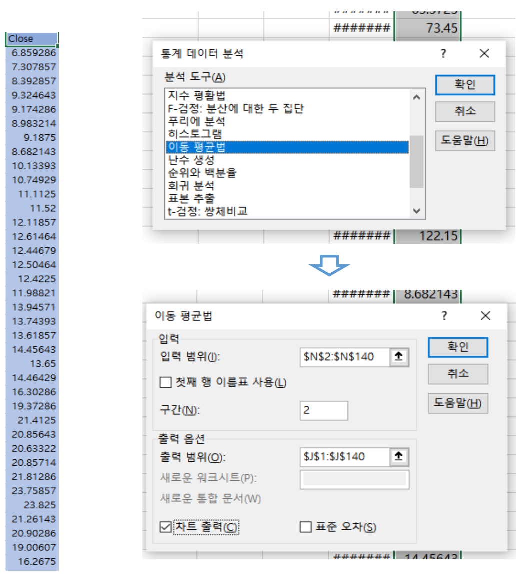
Stock price(close)



	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	########	7.6225	7.699643	6.794643	6.859286	5.898319	1.52E+10
3	########	6.870357	7.3275	6.816071	7.307857	6.284047	1.08E+10
4	########	7.348214	8.481429	7.3375	8.392857	7.217041	1.22E+10
5	########	8.478929	9.730714	8.3125	9.324643	8.018287	1.24E+10
6	########	9.422857	9.567143	7.116071	9.174286	7.888996	1.81E+10
7	########	9.274643	9.964643	8.65	8.983214	7.724694	1.67E+10
8	########	9.082143	9.499643	8.557143	9.1875	7.900359	1.57E+10
9	########	9.301429	9.438571	8.412857	8.682143	7.465801	9.59E+09
10	########	8.838214	10.52607	8.795714	10.13393	8.714196	1.18E+10
11	########	10.21964	11.39286	9.920357	10.74929	9.24334	1.22E+10
12	########	10.79357	11.475	10.63429	11.1125	9.555672	9.51E+09
13	########	11.25964	11.66643	11.24607	11.52	9.906079	6.97E+09
14	########	11.63	12.45	11.60143	12.11857	10.42079	1.08E+10
15	########	12.18929	13.03214	12.06143	12.61464	10.84737	9.3E+09
16	########	12.69536	12.91679	11.65214	12.44679	10.70303	1.13E+10
17	########	12.53964	12.68321	11.43429	12.50464	10.75278	9.25E+09
18	########	12.49071	12.56536	11.765	12.4225	10.68214	6.91E+09
19	########	12.45964	12.57607	11.08929	11.98821	10.3087	9.26E+09
20	########	11.99821	14.44643	11.93571	13.94571	11.99196	1.07E+10
21	########	14.20643	14.26786	12.60786	13.74393	11.81844	1.61E+10
22	########	13.77929	15.10214	13.08857	13.61857	11.71065	1.2E+10
23	########	13.58464	15.23929	12.65143	14.45643	12.43113	1.31E+10
24	########	14.19321	14.57143	12.97571	13.65	11.73767	8.96E+09
25	########	13.66214	14.61036	13.48857	14.46429	12.43788	6.31E+09
26	########	14.62143	16.36571	14.60714	16.30286	14.01887	6.86E+09
27	########	16.37179	19.5575	16.21357	19.37286	16.65878	1.14E+10
28	########	19.5775	22.19464	18.43643	21.4125	18.41266	1.56E+10
29	########	21.49393	23	19.82143	20.85643	17.9345	1.56E+10
30	########	20.88929	21.31286	18.64929	20.63322	17.74256	1.11E+10
31	########	20.32714	21.07143	19.58929	20.85714	17.93512	7.86E+09
32	########	20.88322	22.13821	20.35714	21.81286	18.75694	8.91E+09
33	########	21.99679	24.31679	21.4375	23.75857	20.43006	8.28E+09
34	########	23.77714	25.18107	23.42857	23.825	20.57515	9.2E+09
35	########	23.97	24.16964	20.98929	21.26143	18.36126	1.21E+10
36	########	21.365	21.53571	18.0625	20.90286	18.05161	1.29E+10
37	########	21.20179	21.23536	17.90107	19.00607	16.48851	1.21E+10
38	########	19.77929	19.82143	15.53571	16.2675	14.1127	1.31E+10
39	########	16.39679	17.31929	15.63071	15.76429	13.67614	9.34E+09

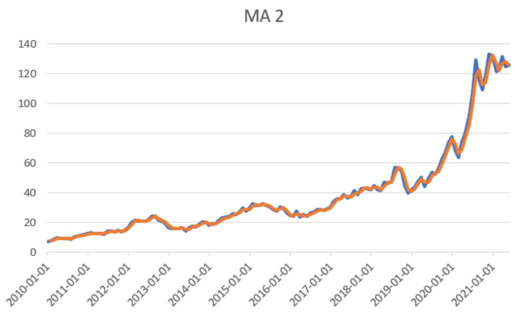
Analysis Details

Moving Average

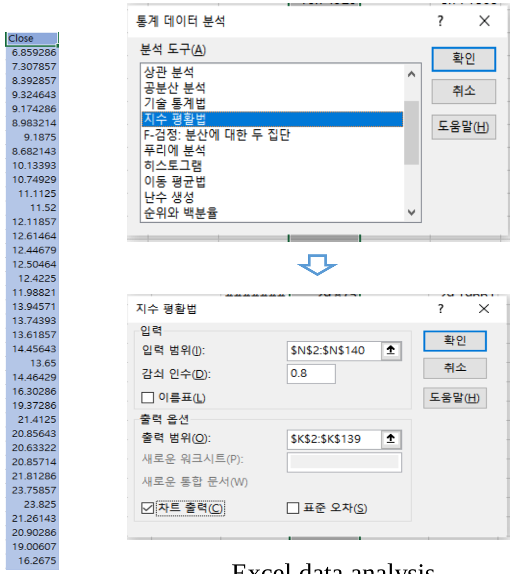


Actual data

Excel data analysis

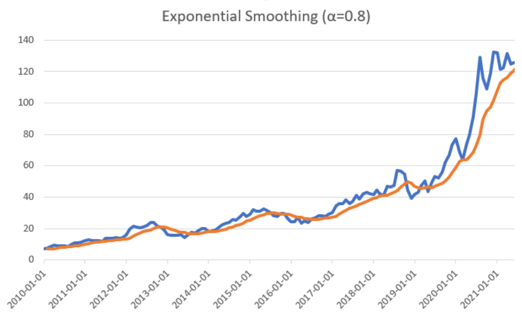


Exponential Smoothing

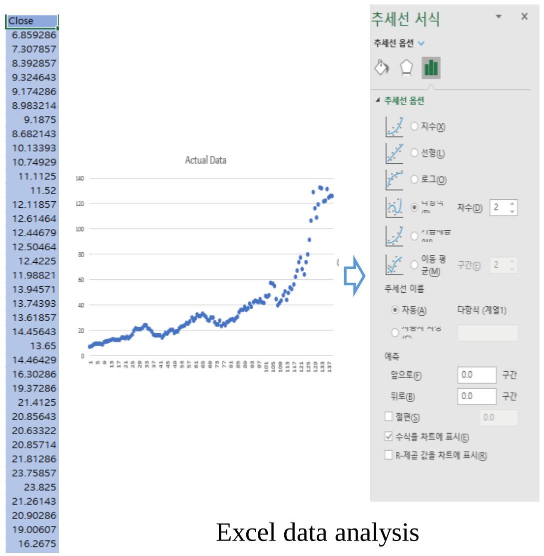


Excel data analysis

Actual data

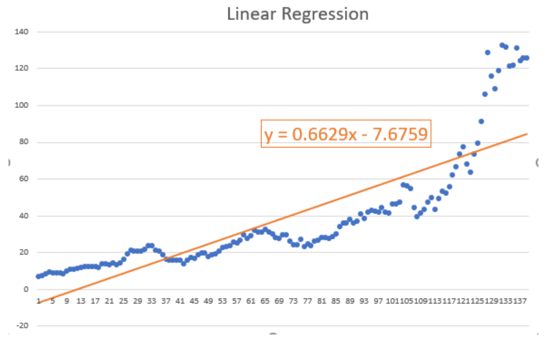


Regression



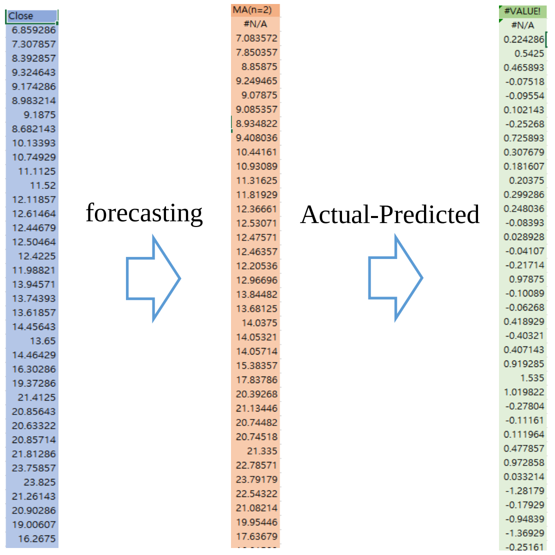
Excel data analysis

Actual data



Calculation of RMSE

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$



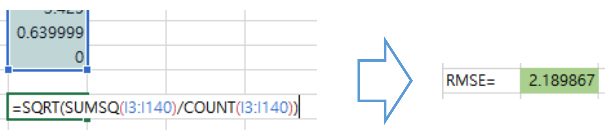
forecasting

Actual-Predicted

Actual data

Predicted data

Deviation data



RMSE

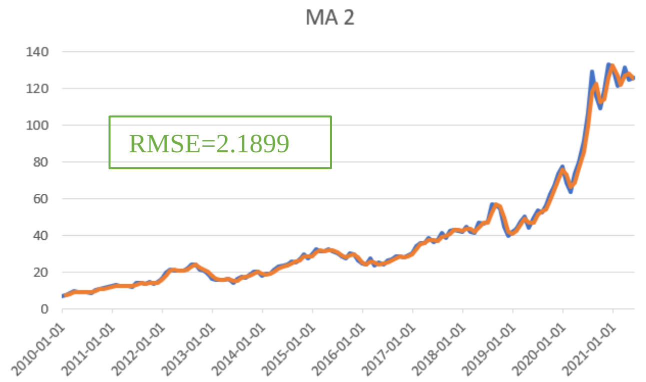
*LSTM detailed on 7th slide

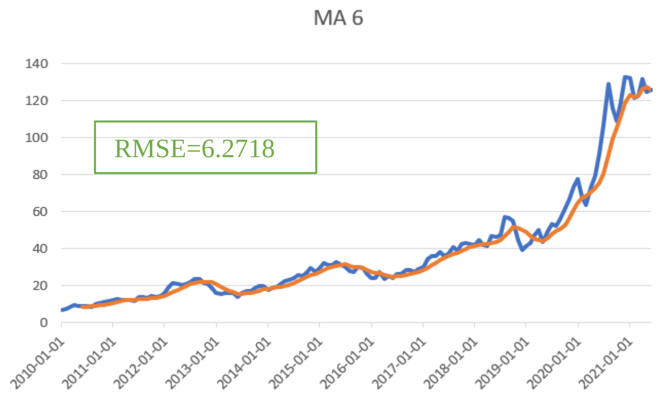
Moving Average & Exponential Smoothing

Moving Average

$$MA_n = \frac{\sum_{i=1}^{n} D_i}{n}$$

n = number of periods in the moving average
D_i = demand in period i



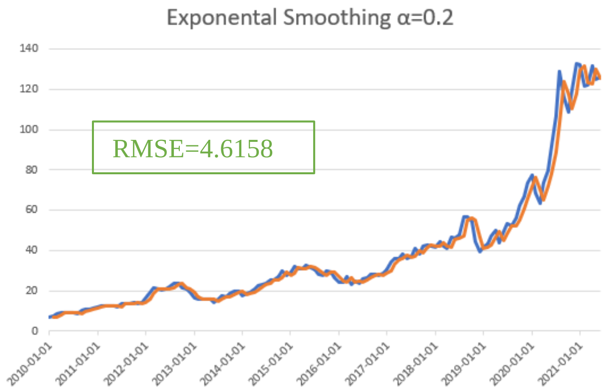


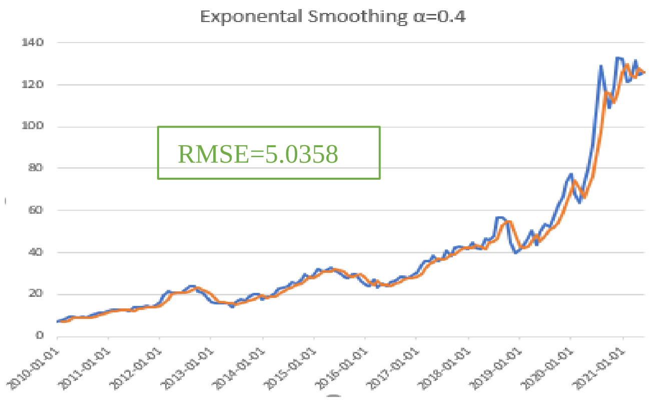
Exponential Smoothing

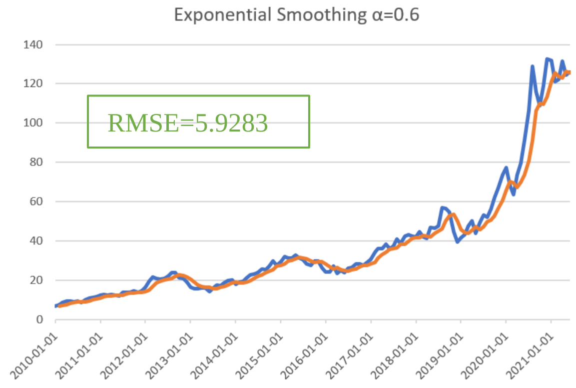
$$F_{t+1} = \alpha D_t + (1 - \alpha)F_t$$

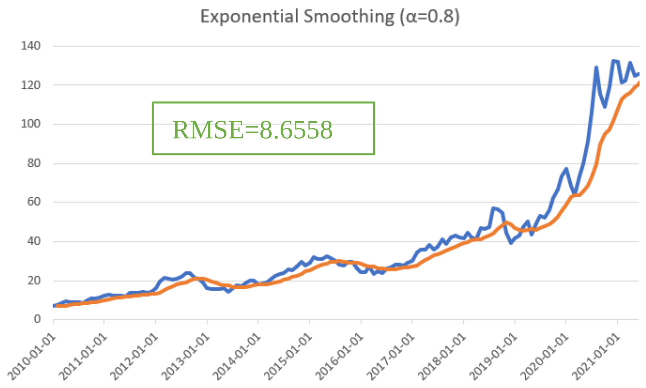
F_{t+1} = the forecast for the next period
D_t = actual demand in the present period
F_t = the previously determined forecast for the present period
α = a weighting factor referred to as the **smoothing constant**

Actual Stock Price
Predicted Stock Price









Deep Learning(LSTM)

Full Code(Python)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os

%matplotlib inline
warnings.filterwarnings('ignore')

plt.rcParams['font.family'] = 'NanumGothic'
```

```python
[3] !pip install finance-datareader
```

```python
[4] import FinanceDataReader as fdr
```

```python
[5] # Apple(AAPL), 애플
    apple = fdr.DataReader('AAPL')
```

```python
[16] STOCK_CODE = 'AAPL'
```

```python
[17] stock = fdr.DataReader(STOCK_CODE)
```

```python
[19] plt.figure(figsize=(16, 9))
     sns.lineplot(y=stock['Close'], x=stock.index)
     plt.xlabel('time')
     plt.ylabel('price')
```

```python
[20] from sklearn.preprocessing import MinMaxScaler

     scaler = MinMaxScaler()
     scale_cols = ['Open', 'High', 'Low', 'Close', 'Volume']
     scaled = scaler.fit_transform(stock[scale_cols])
     scaled
```

```python
[21] df = pd.DataFrame(scaled, columns=scale_cols)
```

```python
[22] from sklearn.model_selection import train_test_split
```

```python
[23] x_train, x_test, y_train, y_test = train_test_split(df.drop('Close', 1), df['Close'], test_size=0.2, random_state=0, shuffle=False)
```

```python
[24] import tensorflow as tf
```

```python
[25] def windowed_dataset(series, window_size, batch_size, shuffle):
         series = tf.expand_dims(series, axis=-1)
         ds = tf.data.Dataset.from_tensor_slices(series)
         ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
         ds = ds.flat_map(lambda w: w.batch(window_size + 1))
         if shuffle:
             ds = ds.shuffle(1000)
         ds = ds.map(lambda w: (w[:-1], w[-1]))
         return ds.batch(batch_size).prefetch(1)
```

```python
[26] WINDOW_SIZE=20
     BATCH_SIZE=32
```

```python
[27] train_data = windowed_dataset(y_train, WINDOW_SIZE, BATCH_SIZE, True)
     test_data = windowed_dataset(y_test, WINDOW_SIZE, BATCH_SIZE, False)
```

```python
[28] from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, LSTM, Conv1D, Lambda
     from tensorflow.keras.losses import Huber
     from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


     model = Sequential([
         Conv1D(filters=32, kernel_size=5,
                padding="causal",
                activation="relu",
                input_shape=[WINDOW_SIZE, 1]),
         # LSTM
         LSTM(16, activation='tanh'),
         Dense(16, activation="relu"),
         Dense(1),
     ])
```

```python
loss = Huber()
optimizer = Adam(0.0005)
model.compile(loss=Huber(), optimizer=optimizer, metrics=['mse'])
```

```python
[30] earlystopping = EarlyStopping(monitor='val_loss', patience=10)
     filename = os.path.join('tmp', 'ckeckpointer.ckpt')
     checkpoint = ModelCheckpoint(filename,
                                  save_weights_only=True,
                                  save_best_only=True,
                                  monitor='val_loss',
                                  verbose=1)
```

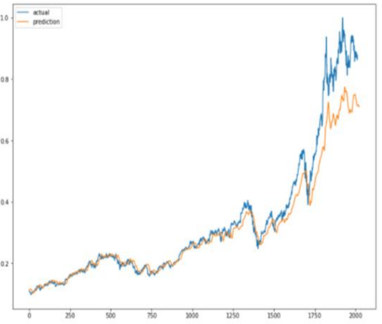
```python
[31] history = model.fit(train_data,
                         validation_data=(test_data),
                         epochs=30,
                         callbacks=[checkpoint, earlystopping])
```

```
255/255 [==============================] - 2s 7ms/step - loss: 9.0865e-07 - mse: 1.8173e-06 - val_loss: 0.0037 - val_mse: 0.0075

Epoch 00015: val_loss improved from 0.00404 to 0.00375, saving model to tmp/ckeckpointer.ckpt
Epoch 16/30
255/255 [==============================] - 2s 7ms/step - loss: 8.9395e-07 - mse: 1.7879e-06 - val_loss: 0.0035 - val_mse: 0.0069

Epoch 00016: val_loss improved from 0.00375 to 0.00346, saving model to tmp/ckeckpointer.ckpt
Epoch 17/30
255/255 [==============================] - 2s 7ms/step - loss: 1.0379e-06 - mse: 2.0758e-06 - val_loss: 0.0037 - val_mse: 0.0074

Epoch 00017: val_loss did not improve from 0.00346
Epoch 18/30
255/255 [==============================] - 2s 7ms/step - loss: 8.4366e-07 - mse: 1.6873e-06 - val_loss: 0.0033 - val_mse: 0.0066

Epoch 00018: val_loss improved from 0.00346 to 0.00329, saving model to tmp/ckeckpointer.ckpt
Epoch 19/30
255/255 [==============================] - 2s 7ms/step - loss: 8.1257e-07 - mse: 1.6251e-06 - val_loss: 0.0029 - val_mse: 0.0057

Epoch 00019: val_loss improved from 0.00329 to 0.00287, saving model to tmp/ckeckpointer.ckpt
Epoch 20/30
255/255 [==============================] - 2s 7ms/step - loss: 1.0948e-06 - mse: 2.1897e-06 - val_loss: 0.0027 - val_mse: 0.0055

Epoch 00020: val_loss improved from 0.00287 to 0.00274, saving model to tmp/ckeckpointer.ckpt
Epoch 21/30
255/255 [==============================] - 2s 7ms/step - loss: 8.1990e-07 - mse: 1.6398e-06 - val_loss: 0.0030 - val_mse: 0.0059

Epoch 00021: val_loss did not improve from 0.00274
Epoch 22/30
255/255 [==============================] - 2s 7ms/step - loss: 7.5867e-07 - mse: 1.5173e-06 - val_loss: 0.0028 - val_mse: 0.0055

Epoch 00022: val_loss did not improve from 0.00274
Epoch 23/30
255/255 [==============================] - 2s 7ms/step - loss: 9.4200e-07 - mse: 1.8840e-06 - val_loss: 0.0026 - val_mse: 0.0052

Epoch 00023: val_loss improved from 0.00274 to 0.00259, saving model to tmp/ckeckpointer.ckpt
Epoch 24/30
255/255 [==============================] - 2s 7ms/step - loss: 9.4511e-07 - mse: 1.8902e-06 - val_loss: 0.0023 - val_mse: 0.0045

Epoch 00024: val_loss improved from 0.00259 to 0.00227, saving model to tmp/ckeckpointer.ckpt
Epoch 25/30
255/255 [==============================] - 2s 7ms/step - loss: 1.1849e-06 - mse: 2.3699e-06 - val_loss: 0.0027 - val_mse: 0.0054

Epoch 00025: val_loss did not improve from 0.00227
Epoch 26/30
255/255 [==============================] - 2s 7ms/step - loss: 7.2476e-07 - mse: 1.4495e-06 - val_loss: 0.0024 - val_mse: 0.0048

Epoch 00026: val_loss did not improve from 0.00227
Epoch 27/30
255/255 [==============================] - 2s 7ms/step - loss: 7.1776e-07 - mse: 1.4355e-06 - val_loss: 0.0022 - val_mse: 0.0044

Epoch 00027: val_loss improved from 0.00227 to 0.00218, saving model to tmp/ckeckpointer.ckpt
Epoch 28/30
255/255 [==============================] - 2s 7ms/step - loss: 7.7652e-07 - mse: 1.5530e-06 - val_loss: 0.0020 - val_mse: 0.0040

Epoch 00028: val_loss improved from 0.00218 to 0.00199, saving model to tmp/ckeckpointer.ckpt
Epoch 29/30
255/255 [==============================] - 2s 7ms/step - loss: 7.5056e-07 - mse: 1.5011e-06 - val_loss: 0.0017 - val_mse: 0.0034

Epoch 00029: val_loss improved from 0.00199 to 0.00168, saving model to tmp/ckeckpointer.ckpt
Epoch 30/30
255/255 [==============================] - 2s 7ms/step - loss: 7.8261e-07 - mse: 1.5652e-06 - val_loss: 0.0017 - val_mse: 0.0034
```

```python
[32] model.load_weights(filename)
```

```python
[33] pred = model.predict(test_data)
```

```python
plt.figure(figsize=(12, 9))
plt.plot(np.asarray(y_test)[30:], label='actual')
plt.plot(pred, label='prediction')
plt.legend()
plt.show()
```




```python
[51] print(pred)
```

```
[[0.11348004]
 [0.11379438]
 [0.11400901]
 ...
 [0.7132945 ]
 [0.7193576 ]
 [0.7099639 ]]
```

Main Code(Python)

```python
#주식 데이터를 편하게 다룰 수 있는 파이썬 패키지
import FinanceDataReader as fdr
```

```python
train_data = windowed_dataset(y_train, WINDOW_SIZE, BATCH_SIZE, True) #학습용 데이터
test_data = windowed_dataset(y_test, WINDOW_SIZE, BATCH_SIZE, False) #검증용 데이터
```

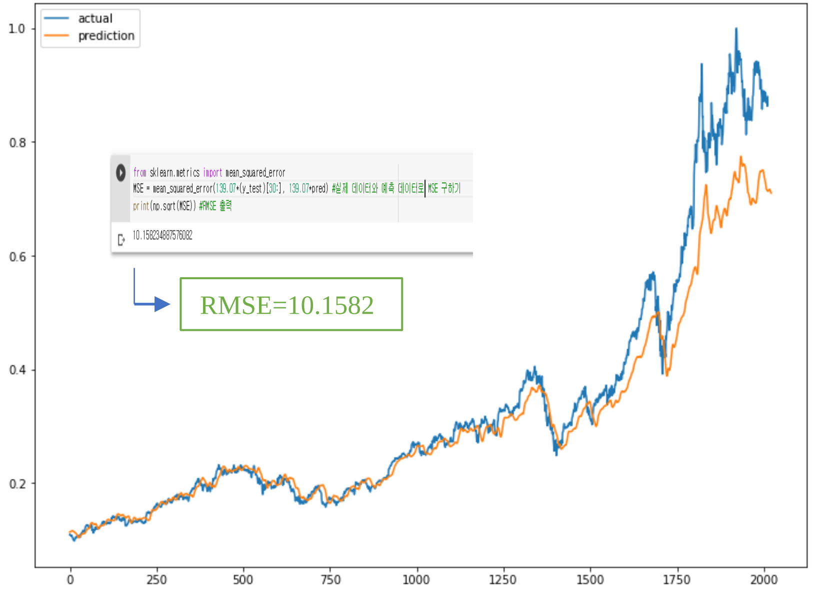
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Conv1D, Lambda
from tensorflow.keras.losses import Huber
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


model = Sequential([
    Conv1D(filters=32, kernel_size=5,
           padding="causal",
           activation="relu",
           input_shape=[WINDOW_SIZE, 1]),
    # LSTM
    LSTM(16, activation='tanh'),
    Dense(16, activation="relu"),
    Dense(1),
])
```

```python
plt.figure(figsize=(12, 9))
#30일(한달)후의 데이터를 예측
plt.plot(np.asarray(y_test)[30:], label='actual')
plt.plot(pred, label='prediction')
plt.legend()
plt.show()
```

```python
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(139.07*(y_test)[30:], 139.07*pred) #실제 데이터와 예측 데이터로 MSE 구하기
print(np.sqrt(MSE)) #RMSE 출력
```

```
10.158234887576082
```

Result and Sub-conclusion




```python
print(139.07*pred)
```

↳ correction coefficient : maximum value of Apple Inc. stock price.

```
[[15.78167 ]
 [15.825385]
 [15.855233]
 ...
 [99.197876]
 [99.00891 ]
 [98.73469 ]]
```

※ The date I made this ppt is 2021.6.6

→ This is a forecast for 2021.7.6. (after 30 days from today)

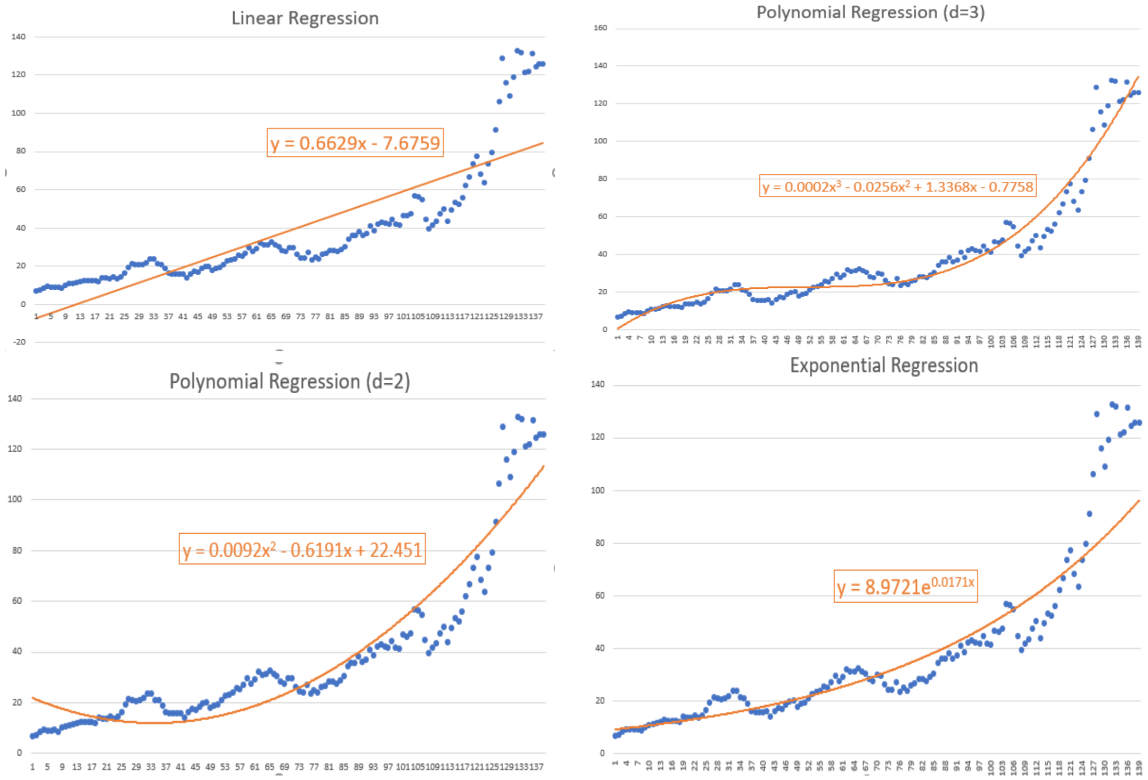
*Apple Inc's stock price on July 6, 2021, is expected to be about **$98.73** by LSTM*

Regression & Result

Regression

Actual Stock Price

Predicted Stock Price



The graph of the Regression model shows that the error is much larger than the previous models. Therefore, **I will omit the RMSE calculation.** (In polynomial regression, the error is also seen to decrease as the order increases.)

Result and Conclusion

	MA_2	MA_6	F_{t+1} (α=0.2)	F_{t+1} (α=0.4)	F_{t+1} (α=0.6)	F_{t+1} (α=0.8)	LSTM
RMSE	2.189867	6.271788	4.615809	5.035775	5.928298	8.655819	10.1582

The best forecasting method is **MA_2**

Date	Stock Price(close)
2021-05-01	124.61
2021-06-01	125.89

$$\xrightarrow[\mathbf{MA_2}]{} \frac{124.61 + 125.89}{2} = 125.25$$

*Apple Inc's stock price on 2021-07-01 is expected to be about **$125.25** by MA_2*

Evaluation & Discussion

Evaluation

Innovativeness: Already many people around the world have forecasted stock prices in a variety of models. Thus, in terms of models, MA, Exponential Smoothing, and Regression are not very innovative. However, Deep Learning (LSTM) model has only been applied to the stock market for a short time. In addition, unlike classical models that produce the same results, LSTM models that produce different results depending on code and computer (GPU) performance can be said to be innovative and creative.

Feasibility: In terms of the feasibility of stock forecasting, it is more realistic than any other topic. The stock market opens every day and can be traded on mobile phones and computers in seconds without limit. There are no time or space constraints. Also, stocks are time series data that change from day to day, so they are meaningful at every point in making predictions. There are various stocks, prices change every day, and volatility is relatively large, and cash profits can be made immediately if predictions are successful. In this regard, forecasting stock prices is of great significance. If you look at the RMSE value of my MA2 model, it is only 2.189. It is a very small error and shows that stock price predictions can be successful.

Practical application: My research is unlikely to have a significant impact on other people's research. My prediction is likely to be used as a one-off due to the nature of the stock is a time series data that changes from day to day. However, if you made a forecasting model to find specific patterns or rules of stock graphs, the research will be very influential around the worldwide.

Different from relative work: I tried to make good results by adjusting batch size and epoch several times while creating an LSTM model. My LSTM programming code and process will be different from anyone else. But I don't know if it's because of my bad code (not finding the best learning conditions) or bad computer performance (or both), but my LSTM model showed a poor RMSE of 10.1582.

Discussion

During this project, I could learn about the principles of polynomial regression, MAPE, MPE, MSLE, and other statistical knowledge that I did not learn in class. Furthermore, while coding LSTM, I was able to know the data preprocessing process, various statistical computation-related Python packages. In particular, I spent a lot of time due to errors in the data preprocessing process, but I learned that it is important to matrixing the data exactly as required by Python package functions. While I was studying forecasting to do this project this time, I found out that R language is an optimized programming language for statistical analysis. Next time, I want to do statistical analysis using R language.

I wondered if run LSTM as a supercomputer would really bring stock forecasting closer to perfection. Of course, it's not such a complicated algorithm, but as I agonized over this topic, I thought there would be a rule of stock graph that people haven't yet discovered.

References

[1] Murtaza Roondiwala , Harshal Patel , Shraddha Varma “Predicting Stock Prices Using LSTM”, IJSR(2015)

[2] Handanhal V. Ravinder, “Determining The Optimal Values Of Exponential Smoothing Constants – Does Solver Really Work?”, American Journal Of Business Education Volume 6, Number 3 (2013)

[3] S Abdulsalam Sulaiman Olaniyi, Adewole, Kayode S. , Jimoh, R. G , “Stock Trend Prediction Using Regression Analysis – A Data Mining Approach” , ARPN Journal of Systems and Software, Volume 1 No. 4 (2011)

[4] https://finance.yahoo.com/quote/AAPL?p=AAPL&.tsrc=fin-srch

[5] https://statkclee.github.io/statistics/stat-time-series-forecast.html

[6] https://danbi-ncsoft.github.io/study/2018/05/04/study-regression_model_summary.html

[7] https://teddylee777.github.io/tensorflow/LSTM%EC%9C%BC%EB%A1%9C-%EC%98%88%EC%B8%A1%ED%95%B4%EB%B3%B4%EB%8A%94-%EC%82%BC%EC%84%B1%EC%A0%84%EC%9E%90-%EC%A3%BC%EA%B0%80