Kaggle 股價預測

[參考來源]

Multiple Stock Prediction Using Single NN

```
!rm -rf /content/stock/
!mkdir /content/stock

# upload dataset yourself.
```

```
%tensorflow_version 1.x
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
plt.style.use('seaborn-whitegrid')

from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

```
# Some functions to help out with
def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

```
import os
fileList = os.listdir("/content/stock")
```

```
companyList = []
for file in fileList:
    companyName = file.split("_")[0]
    if companyName != "all":
        companyList.append(companyName)
print(companyList)
```

```
['2882', '2330', '2317', '1301', '2412', '6505']
```

```
# First, we get the data
stockList = ['2330', '2317', '6505', '2412', '1301', '2882']
df_ = {}
for i in stockList:
    df_[i] = pd.read_csv("/content/stock/" + i + "_2007-04-23_to_2019-06-28.cs
    df_[i]
```

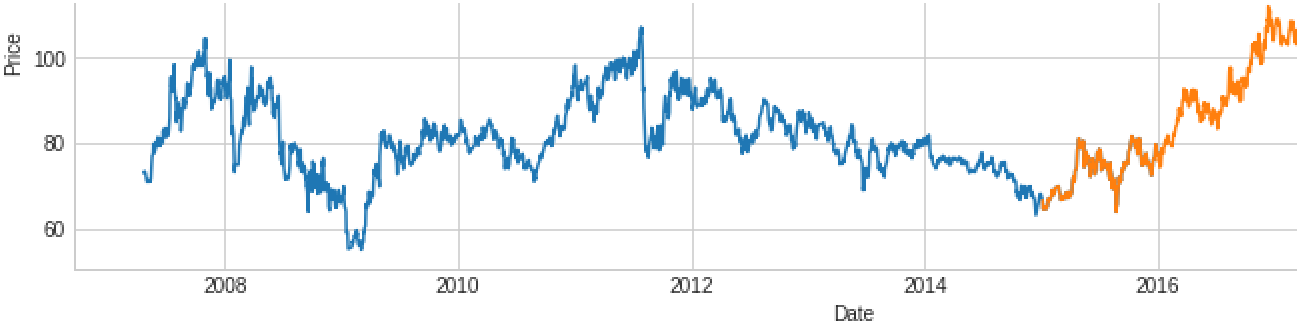
```
df_
```

```
...
2007-04-23  2007-04-23   68.0   68.6   69.2   67.9   8152.0  2330
2007-04-24  2007-04-24   68.8   69.8   70.0   68.6  11240.0  2330
2007-04-25  2007-04-25   69.4   69.3   69.7   68.7   4485.0  2330
2007-04-26  2007-04-26   70.4   69.9   70.5   69.7   8124.0  2330
2007-04-27  2007-04-27   69.9   69.0   70.0   68.7   5254.0  2330
...                ...    ...    ...    ...    ...      ...   ...
2019-06-24  2019-06-24  241.0  241.0  242.0  240.0  17780.0  2330
2019-06-25  2019-06-25  241.0  238.5  241.5  237.0  11130.0  2330
2019-06-26  2019-06-26  235.0  234.5  236.5  234.0  13535.0  2330
2019-06-27  2019-06-27  236.0  240.5  241.5  236.0  14983.0  2330
2019-06-28  2019-06-28  241.5  239.0  241.5  238.0   8987.0  2330

[3014 rows x 7 columns],
'2412':                   date   Open   High    Low  Close  Volume  Name
Date
2007-04-23  2007-04-23   64.5   64.6   65.3   64.4  1619.0  2412
2007-04-24  2007-04-24   64.6   64.6   64.9   64.0  1222.0  2412
2007-04-25  2007-04-25   64.5   63.8   64.5   63.7  1682.0  2412
2007-04-26  2007-04-26   64.0   63.1   64.0   63.1  1940.0  2412
2007-04-27  2007-04-27   63.3   63.3   63.8   63.1  1038.0  2412
...                ...    ...    ...    ...    ...     ...   ...
2019-06-24  2019-06-24  113.5  114.0  114.0  113.0  2658.0  2412
2019-06-25  2019-06-25  113.5  114.0  114.0  113.0  2418.0  2412
2019-06-26  2019-06-26  114.0  114.0  114.0  113.5  2366.0  2412
2019-06-27  2019-06-27  114.0  113.5  114.0  113.0  3783.0  2412
2019-06-28  2019-06-28  113.5  113.0  114.0  113.0  1596.0  2412

[3014 rows x 7 columns],
'2882':                   date   Open   High    Low  Close  Volume  Name
Date
2007-04-23  2007-04-23  69.80  70.20  70.20  69.70  3253.0  2882
2007-04-24  2007-04-24  69.10  69.50  69.90  69.10  2265.0  2882
2007-04-25  2007-04-25  69.00  68.50  69.10  68.30  3612.0  2882
2007-04-26  2007-04-26  69.00  68.60  69.50  68.60  2561.0  2882
2007-04-27  2007-04-27  69.40  68.30  69.40  68.30  2091.0  2882
...                ...    ...    ...    ...    ...     ...   ...
2019-06-24  2019-06-24  42.30  42.55  42.55  42.05  3068.0  2882
2019-06-25  2019-06-25  42.55  42.60  42.70  42.45  2855.0  2882
2019-06-26  2019-06-26  42.70  42.75  42.90  42.60  4054.0  2882
2019-06-27  2019-06-27  42.75  43.20  43.35  42.75  7511.0  2882
2019-06-28  2019-06-28  43.20  43.00  43.40  43.00  7514.0  2882

[3014 rows x 7 columns],
'6505':                   date   Open   High    Low  Close  Volume  Name
Date
2007-04-23  2007-04-23   73.0   73.6   74.4   72.9   958.0  6505
2007-04-24  2007-04-24   73.6   73.6   74.1   73.4   819.0  6505
2007-04-25  2007-04-25   73.5   73.0   73.5   73.0   352.0  6505
2007-04-26  2007-04-26   73.0   73.4   73.7   73.0   640.0  6505
2007-04-27  2007-04-27   73.6   73.0   73.6   73.0   527.0  6505
...                ...    ...    ...    ...    ...     ...   ...
2019-06-24  2019-06-24  112.5  113.0  113.5  111.5  1682.0  6505
2019-06-25  2019-06-25  113.0  111.5  113.0  111.0  2242.0  6505
2019-06-26  2019-06-26  111.5  111.0  112.5  111.0  1009.0  6505
2019-06-27  2019-06-27  111.5  111.5  112.5  111.0  1561.0  6505
2019-06-28  2019-06-28  111.5  110.5  111.5  110.0  1444.0  6505

[3014 rows x 7 columns]}
```

```
def split(dataframe, border, col):
    return dataframe.loc[:border,col], dataframe.loc[border:,col]

df_new = {}
for i in stockList:
    df_new[i] = {}
    df_new[i]["Train"], df_new[i]["Test"] = split(df_[i], "2015", "Close")
df_new
```

```
2007-04-25    63.7
2007-04-26    63.1
2007-04-27    63.1
              ...
2015-12-25    99.3
2015-12-28    99.2
2015-12-29    99.0
2015-12-30    99.0
2015-12-31    99.0
Name: Close, Length: 2162, dtype: float64}, '2882': {'Test': Date
2015-01-05    46.20
2015-01-06    44.80
2015-01-07    44.85
2015-01-08    45.35
2015-01-09    45.70
              ...
2019-06-24    42.05
2019-06-25    42.45
2019-06-26    42.60
2019-06-27    42.75
2019-06-28    43.00
Name: Close, Length: 1096, dtype: float64, 'Train': Date
2007-04-23    69.70
2007-04-24    69.10
2007-04-25    68.30
2007-04-26    68.60
2007-04-27    68.30
              ...
2015-12-25    43.50
2015-12-28    43.75
2015-12-29    43.55
2015-12-30    42.70
2015-12-31    42.65
Name: Close, Length: 2162, dtype: float64}, '6505': {'Test': Date
2015-01-05     66.6
2015-01-06     65.1
2015-01-07     64.5
2015-01-08     65.0
2015-01-09     65.3
              ...
2019-06-24    111.5
2019-06-25    111.0
2019-06-26    111.0
2019-06-27    111.0
2019-06-28    110.0
Name: Close, Length: 1096, dtype: float64, 'Train': Date
2007-04-23    72.9
2007-04-24    73.4
2007-04-25    73.0
2007-04-26    73.0
2007-04-27    73.0
              ...
2015-12-25    79.7
2015-12-28    78.7
2015-12-29    77.5
2015-12-30    77.1
2015-12-31    77.8
Name: Close, Length: 2162, dtype: float64}}
```

```
for i in stockList:
    plt.figure(figsize=(14,4))
    plt.plot(df_new[i]["Train"])
    plt.plot(df_new[i]["Test"])
    plt.ylabel("Price")
    plt.xlabel("Date")
    plt.legend(["Training Set", "Test Set"])
    plt.title(i + " Closing Stock Price")
```


Price
100
80
60
2008
2010
2012
2014
2016
Date


2412 Closing Stock Price
Training Set
Test Set
120
110
100
90
80
70
60
50
Price
2008
2010
2012
2014
2016
Date


1301 Closing Stock Price
Training Set
Test Set
120
110
100
90
80
70
60
50
40
Price
2008
2010
2012
2014
2016
Date


2882 Closing Stock Price
90
80
70
60
50
40
30
Price
2008
2010
2012
2014
2016
Date

```
# Scaling the training set
transform_train = {}
transform_test = {}
scaler = {}

for num, i in enumerate(stockList):
    sc = MinMaxScaler(feature_range=(0,1))
    a0 = np.array(df_new[i]["Train"])
    a1 = np.array(df_new[i]["Test"])
    a0 = a0.reshape(a0.shape[0],1)
    a1 = a1.reshape(a1.shape[0],1)
    transform_train[i] = sc.fit_transform(a0)
    transform_test[i] = sc.fit_transform(a1)
    scaler[i] = sc

del a0
del a1
```

```
for i in transform_train.keys():
    print(i, transform_train[i].shape)
print("\n")
for i in transform_test.keys():
```

```
for i in transform_test.keys():
    print(i, transform_test[i].shape)
```

```
2330 (2162, 1)
2317 (2162, 1)
6505 (2162, 1)
2412 (2162, 1)
1301 (2162, 1)
2882 (2162, 1)


2330 (1096, 1)
2317 (1096, 1)
6505 (1096, 1)
2412 (1096, 1)
1301 (1096, 1)
2882 (1096, 1)
```

```
trainset = {}
testset = {}
for j in stockList:
    trainset[j] = {}
    X_train = []
    y_train = []
    for i in range(60,2162):
        X_train.append(transform_train[j][i-60:i,0])
        y_train.append(transform_train[j][i,0])
    X_train, y_train = np.array(X_train), np.array(y_train)
    trainset[j]["X"] = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],
    trainset[j]["y"] = y_train

    testset[j] = {}
    X_test = []
    y_test = []
    for i in range(60, 1096):
        X_test.append(transform_test[j][i-60:i,0])
        y_test.append(transform_test[j][i,0])
    X_test, y_test = np.array(X_test), np.array(y_test)
    testset[j]["X"] = np.reshape(X_test, (X_test.shape[0], X_train.shape[1], 1
    testset[j]["y"] = y_test
```

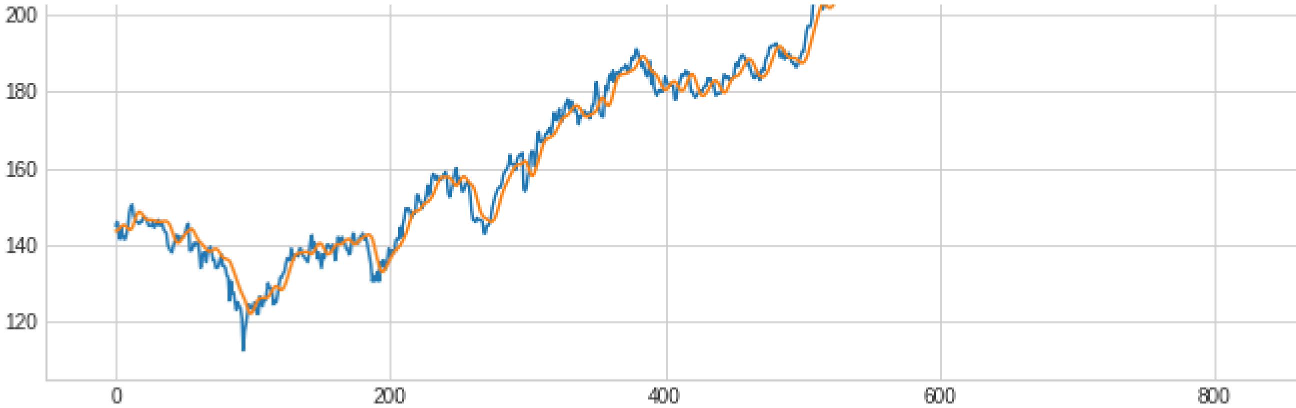
```
arr_buff = []
for i in stockList:
    buff = {}
    buff["X_train"] = trainset[i]["X"].shape
    buff["y_train"] = trainset[i]["y"].shape
    buff["X_test"] = testset[i]["X"].shape
    buff["y_test"] = testset[i]["y"].shape
    arr_buff.append(buff)

pd.DataFrame(arr_buff, index=stockList)
```

	X_train	y_train	X_test	y_test
2330	(2102, 60, 1)	(2102,)	(1036, 60, 1)	(1036,)
2317	(2102, 60, 1)	(2102,)	(1036, 60, 1)	(1036,)
6505	(2102, 60, 1)	(2102,)	(1036, 60, 1)	(1036,)
2412	(2102, 60, 1)	(2102,)	(1036, 60, 1)	(1036,)

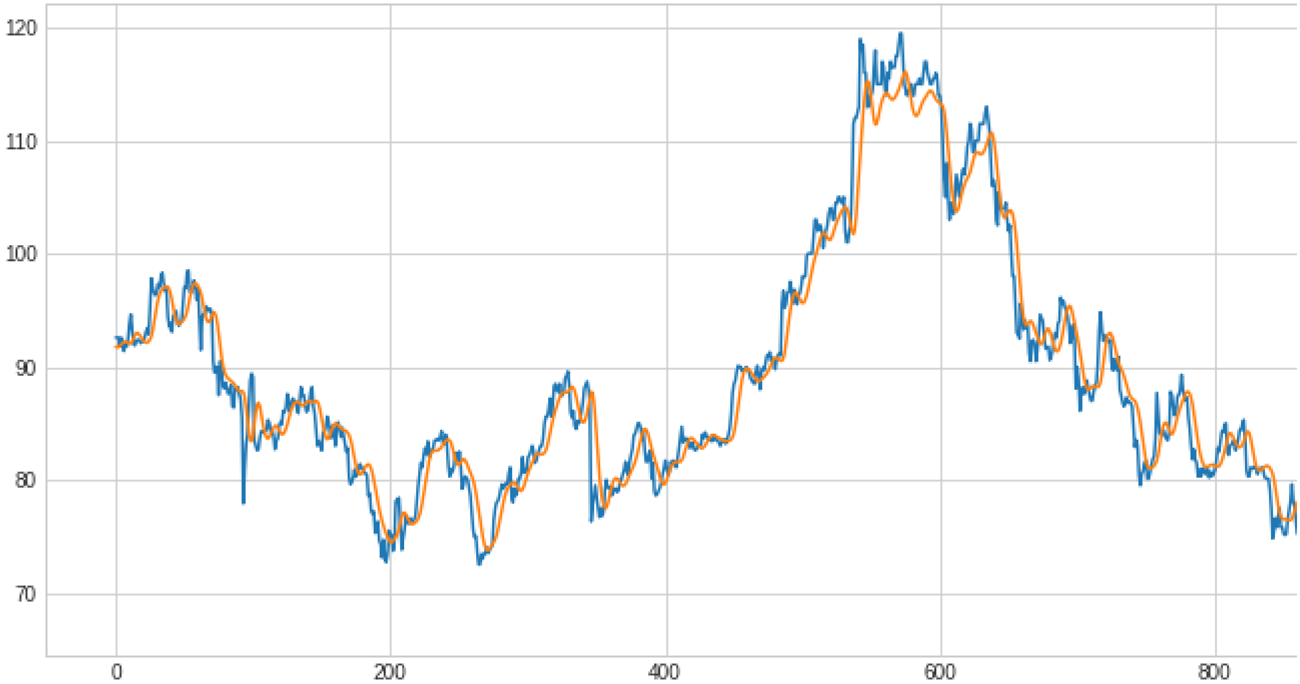
```
%%time
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.5))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.5))
# The output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='rmsprop', loss='mean_squared_error')
# Fitting to the training set
for i in stockList:
    print("Fitting to", i)
    regressor.fit(trainset[i]["X"], trainset[i]["y"], epochs=10, batch_size=2(
```

```
Epoch 4/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0117
Epoch 5/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0104
Epoch 6/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0110
Epoch 7/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0116
Epoch 8/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0116
Epoch 9/10
2102/2102 [==============================] - 6s 3ms/step - loss: 0.0101
Epoch 10/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0108
Fitting to 1301
Epoch 1/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0060
Epoch 2/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0078
Epoch 3/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0059
Epoch 4/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0064
Epoch 5/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0070
Epoch 6/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0063
Epoch 7/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0068
Epoch 8/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0063
Epoch 9/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0064
Epoch 10/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0055
Fitting to 2882
Epoch 1/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0032
Epoch 2/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0038
Epoch 3/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0040
Epoch 4/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0041
Epoch 5/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0036
Epoch 6/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0036
Epoch 7/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0034
Epoch 8/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0033
Epoch 9/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0041
Epoch 10/10
2102/2102 [==============================] - 5s 2ms/step - loss: 0.0032
CPU times: user 8min 37s, sys: 24.1 s, total: 9min 1s
Wall time: 4min 50s
```

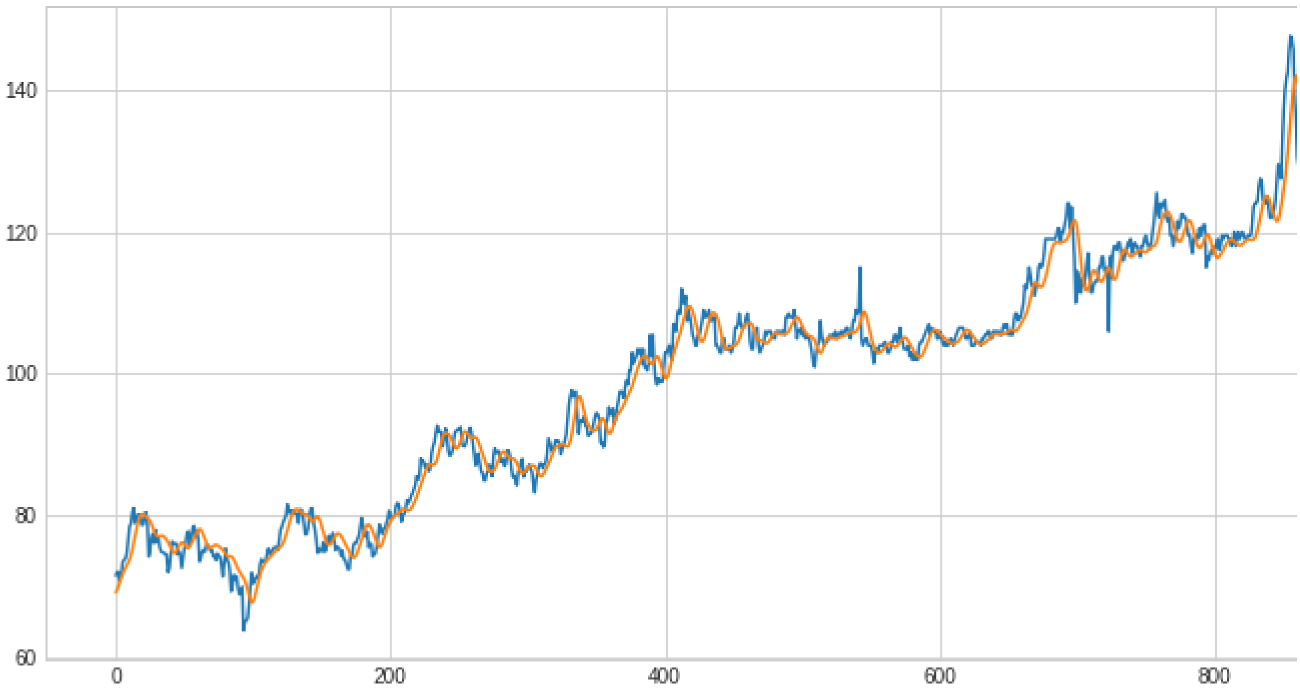
```
import math

pred_result = {}
for i in stockList:
    try:
        y_true = scaler[i].inverse_transform(testset[i]["y"].reshape(-1,1))
        y_pred = scaler[i].inverse_transform(regressor.predict(testset[i]["X"]

        MSE = mean_squared_error(y_true, y_pred)
        pred_result[i] = {}
        pred_result[i]["True"] = y_true
        pred_result[i]["Pred"] = y_pred

        plt.figure(figsize=(14,6))
        plt.title("{} with MSE {:10.4f}".format(i,MSE))
        plt.plot(y_true)
        plt.plot(y_pred)
    except:
        print('Nan ')
```



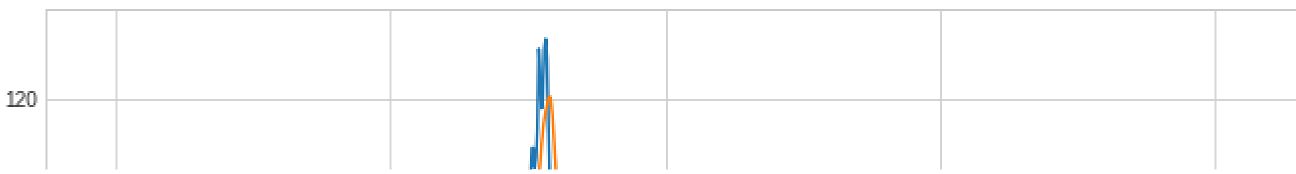
2317 with MSE 6.4174

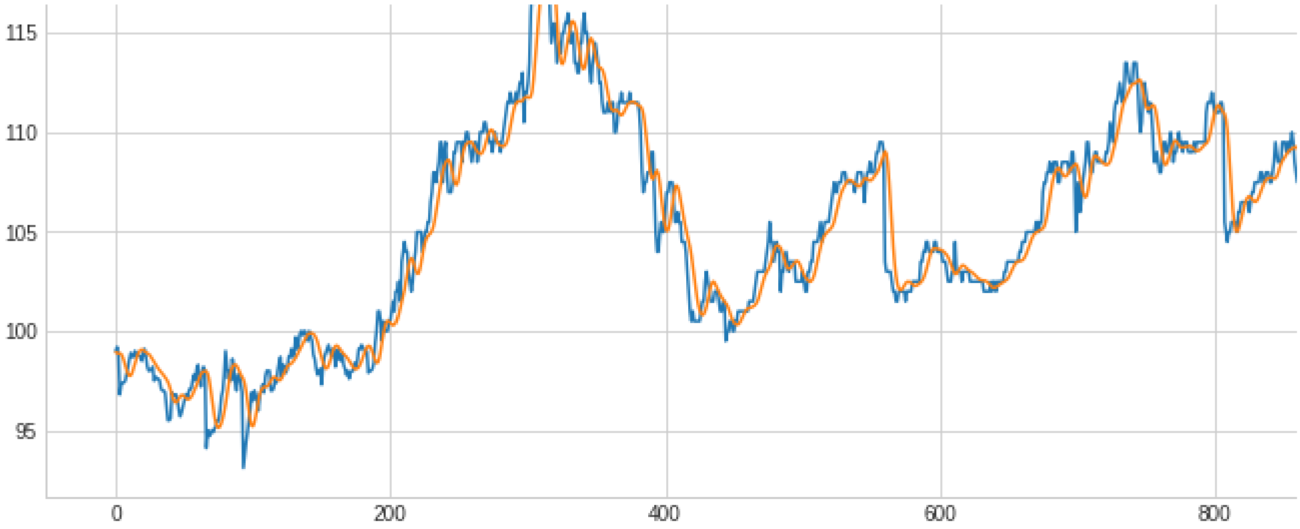


6505 with MSE 6.8107

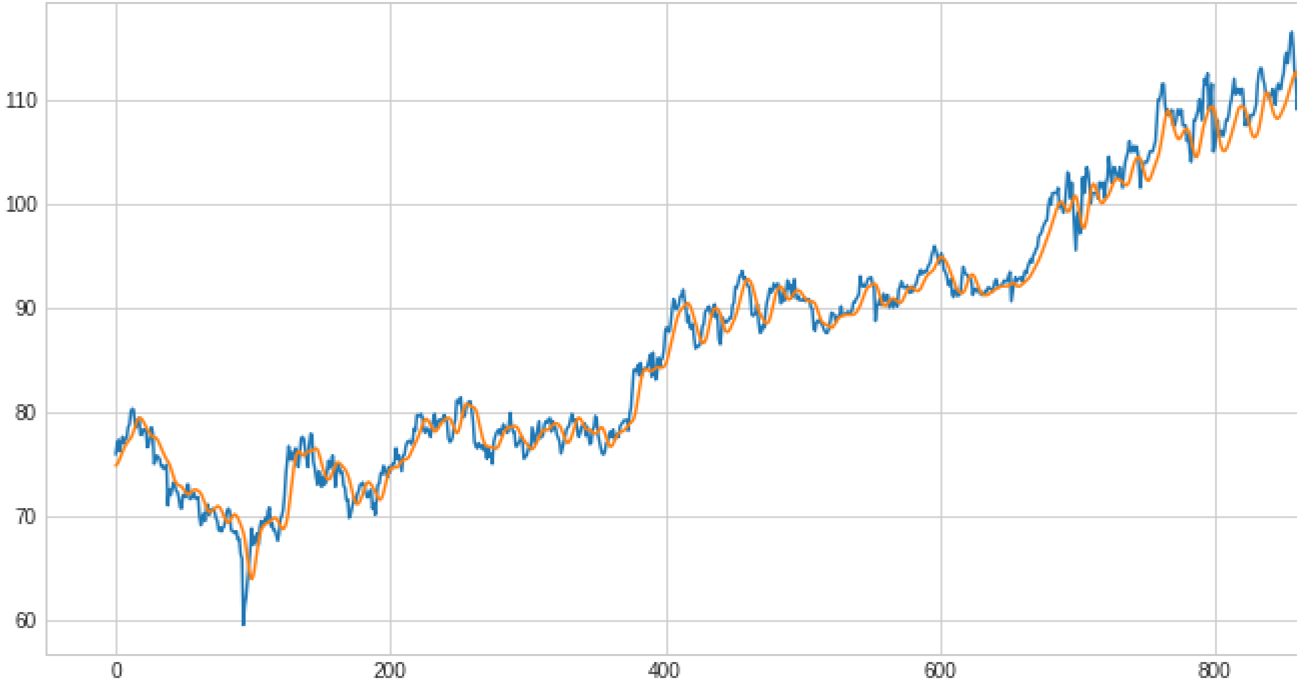


2412 with MSE 1.5139

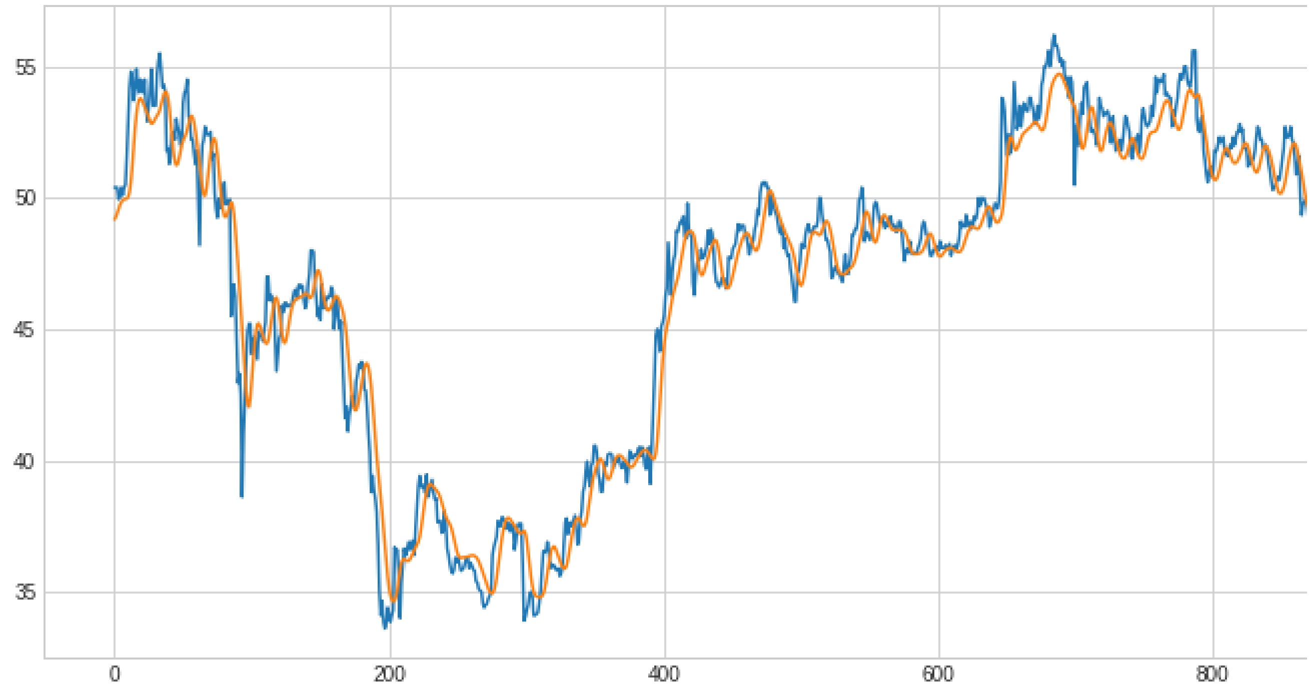




1301 with MSE 3.3848



2882 with MSE 1.4115



```
time_index = df_new["2317"]["Test"][60:].index
def lagging(df, lag, time_index):
    df_pred = pd.Series(df["Pred"].reshape(-1), index=time_index)
    df_true = pd.Series(df["True"].reshape(-1), index=time_index)

    df_pred_lag = df_pred.shift(lag)

    print("MSE without Lag", mean_squared_error(np.array(df_true), np.array(df
    print("MSE with Lag 5", mean_squared_error(np.array(df_true[:-5]), np.arra

    plt.figure(figsize=(14,4))
    plt.title("Prediction without Lag")
    plt.plot(df_true)
    plt.plot(df_pred)

    MSE_lag = mean_squared_error(np.array(df_true[:-5]), np.array(df_pred_lag[
    plt.figure(figsize=(14,4))
    plt.title("Prediction with Lag")
    plt.plot(df_true)
    plt.plot(df_pred_lag)
```

```
dff = pred_result["2330"]
pd.Series(dff["Pred"].reshape(-1), index=time_index)
```

```
Date
2015-04-10    143.644073
2015-04-13    143.683884
2015-04-14    144.014786
2015-04-15    144.479568
2015-04-16    144.856232
                 ...
2019-06-24    234.187149
2019-06-25    234.824478
2019-06-26    235.536789
2019-06-27    235.887161
2019-06-28    235.773041
Length: 1036, dtype: float32
```

```
lagging(pred_result["2330"], -5, time_index)
```