Se

０．Analysis environment

Data editing was done in R, and post-editing analysis was done in Python. I would like to thank Rstudio.Cloud and Google Colaboratory for their help.

１．Data to be analyzed

1.1. Nikkei 225

After looking into it, it seems that there are various peculiarities as a stock price index, but it is the most major, so I adopted it.

1.2. dow industrial stocks

There is also the S&P500. Somehow decided with a major feeling.

1.3. Japanese government bond 10 years

The reason I used 10-year government bonds as an interest rate indicator was because I was taught that when I was working at a stock market at the end of the market.

It's knowledge from 20 years ago, so I don't know if that feeling still remains.

1.4. 10 year US treasury bond

Since I chose a 10-year Japanese government bond, I set it to 10 years to match the consistency of the maturity.

1.5. Yen US dollar exchange

Once I got the US and UK stocks and interest rates, I wanted a currency exchange that connects them.

1.6. others

I thought about gold as an escape from financial assets, but I didn't want to increase variables, so I gave up.

1.7. data editing program

Since the author is unfamiliar with python, I edited it with R. I'm not good at R anymore.

1.7.1. Joining method

In terms of SQL, the date is used as a join key, and it is a simple Inner Join, and it is a body that throws away all the days that are closed even if it is one. It's just a preliminary preliminary analysis, so I'm fine with this.

１．8．basic statistics

<Python-Code-Begin>

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

import statsmodels.api as sm

import datetime

#Data

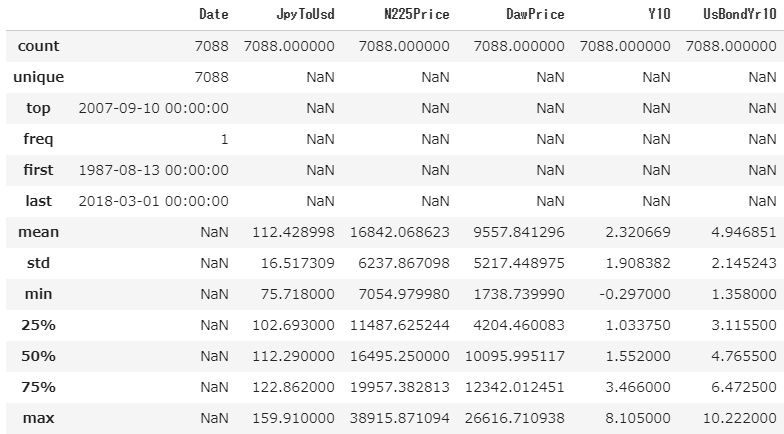
!wget https://www.dropbox.com/s/\*\*\*\*\*\*\*\*/TsData002.csv

df000 = pd.read\_csv( 'TsData002.csv', parse\_dates=['Date'])

df000.head(5)



df000.describe(include='all')



<Python-Code-End>

２．Univariate time series

２．１．fbprophet

It is a time series analysis tool made by facebook, which has already been tried by various people and reported on qiita. It seems that it is explained as a developed form of the Holt Winters method. It seems to work in both R and Python. It worked in my local environment in R, but it didn't work in Rstudio could. Is it because you use Stan? Also, it supports univariate, but does not seem to support multivariate. This time, I tried it with Python on Google Colaboratory.

First, prepare. It's easier to put the data edited in R in "share" mode on dropbox and use "!wget" instead of putting it on google drive.

<code\_below>

#Install

!pip install pandas

!pip install numpy

!pip install pystan

!pip install matplotlib

!pip install fbprophet

#Import Data

!wget [https://www.dropbox.com/\*\*\*\*\*\*\*（省略）\*\*\*\*\*\*\*\*\*\*\*\*/TsData002.csv](https://www.dropbox.com/*******（省略）************/TsData002.csv)

#Load Data

import pandas as pd

import numpy as np

df = pd.read\_csv('TsData002.csv')

<code\_above>

I tried all five series, but I will write about the Nikkei 225, which is the target variable that I want to predict as a representative.

Nikkei 225. Since the code has not been cleaned of redundant parts after copy and paste, it is not beautiful because it is imported twice, but please forgive me.

<code\_below>

#N225Price

df2 = df[['Date','N225Price']].copy()

df2 = df2.rename(columns={'Date': 'ds', 'N225Price':'y'})

from fbprophet import Prophet

model = Prophet()

model.fit(df2)

future\_df2 = model.make\_future\_dataframe(365)

future\_df2.tail()

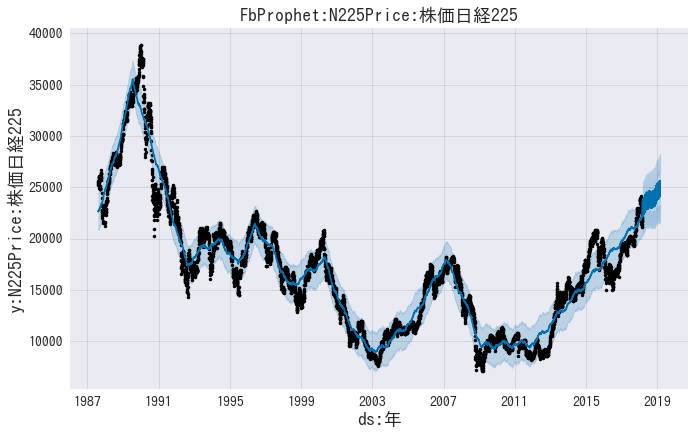
forecast\_df2 = model.predict(future\_df2)

forecast\_df2[['ds','yhat']].tail()

from matplotlib import pyplot as plt

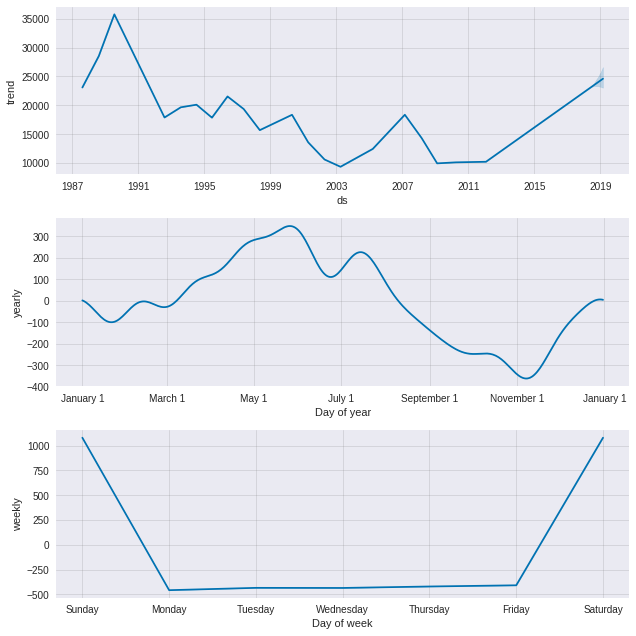
model.plot(forecast\_df2)

plt.show()



model.plot\_components(forecast\_df2)

plt.show()



# Calculate root mean squared error.

print('RMSE: %f' % np.sqrt(np.mean((forecast\_df2.loc[:len(df2), 'yhat']-df2['y'])\*\*2)) )

>>>RMSE: 1399.172808

<code\_above>

Since the Great East Japan Earthquake, the annual trend has been on an upward trend. In terms of monthly periodicity, it can be read that the bottom will be around November. It was so beautiful that I began to wonder if it was true. If you think about it with a little statistics, you can think that this graph itself is "point estimation" in statistical inference. In other words, I think it would be even better if you could understand the confidence interval elements like the first graph. That way, you can analyze the situation based on the “certainty” of trends and periodicity.

Perhaps such a function is also implemented and can be used with option commands etc.

I would like to investigate this in the future.

２．２．ARIMA changed to ARMA

Next, let's start working on ARIMA.

ARIMA is ARIMA. If you predict the number of daily page requests for a Japanese airline company in the past 20 months for one month in the future, and if the difference between the measured value and the actual value is within 5%, then it is OK. I remember that I was able to handle the increase in the number of accesses to . but···.

So, I have good memories of ARIMA, but I wonder if it will fit well and make predictions this time.

It was quite difficult, so only Nikkei 225.

Postscript: When the degree is automatically estimated, it becomes (2, 0, 2) and the sum is zero, so it is ARMA.

<code\_below>

# Load Libraries

import numpy as np

import pandas as pd

from scipy import stats

# Draw Graph

from matplotlib import pylab as plt

import seaborn as sns

%matplotlib inline

# make graph horizontal

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 15, 6

# Statistical model

import statsmodels.api as sm

#Read Data

!get [https://www.dropbox.com/\*\*\*\*\*\*\*TsData002.csv](https://www.dropbox.com/*******TsData002.csv)

#Load data to pandas

df000 = pd.read\_csv( 'TsData002.csv', index\_col='Date', parse\_dates=['Date'])

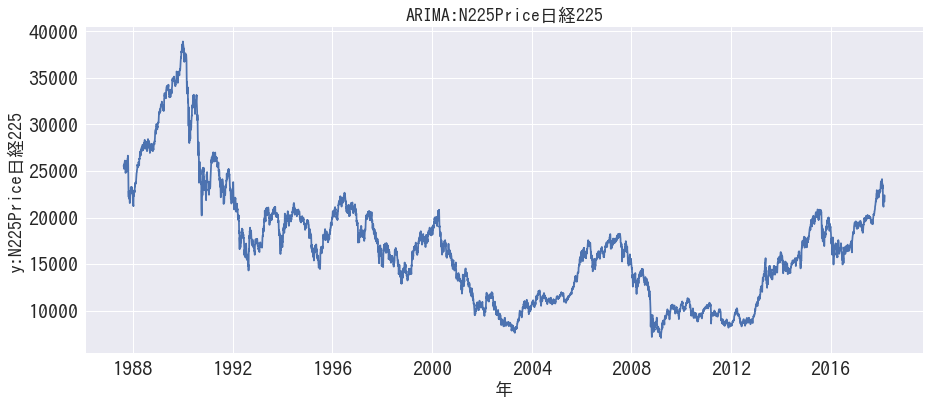
df000.head(5)



#Eliminate other than N225Pricew

df001 = df000[["N225Price"]]

plt.plot(df001)

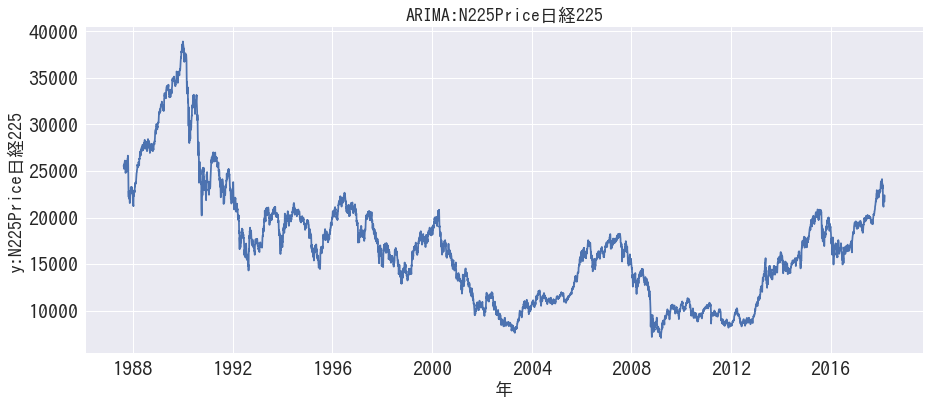


# fix date type & delete null data

ts = df001['N225Price']

ts =ts.dropna(how='any')

plt.plot(ts)



# find the autocorrelation

ts\_acf = sm.tsa.stattools.acf(ts, nlags=40)

ts\_acf

array([1. , 0.99903304, 0.99807604, 0.99716146, 0.99628441,  
 0.99541629, 0.99456571, 0.99368822, 0.99272498, 0.99174111,  
 0.9907543 , 0.98977139, 0.98876962, 0.98776901, 0.98681383,  
 0.98584892, 0.98481014, 0.98377661, 0.98275179, 0.98173483,  
 0.98073477, 0.97978321, 0.97884645, 0.97786696, 0.97685134,  
 0.97588177, 0.97492482, 0.97393909, 0.97294024, 0.97193549,  
 0.97092823, 0.96989444, 0.96882841, 0.96776806, 0.96671785,  
 0.96567044, 0.96458989, 0.96346446, 0.96231421, 0.96119644,  
 0.96008845])

# partial autocorrelation

ts\_pacf = sm.tsa.stattools.pacf(ts, nlags=40, method='ols')

ts\_pacf

array([ 1. , 0.9991194 , 0.00818315, 0.0280301 , 0.02071028,  
 0.0013183 , 0.013607 , -0.01082195, -0.05106348, -0.01921664,  
 -0.00417685, -0.00259338, -0.01256579, 0.00143656, 0.03637976,  
 -0.00937997, -0.05032874, 0.01730417, -0.00133815, 0.00239207,  
 0.00287304, 0.02941132, 0.01823222, -0.02035991, -0.02390654,  
 0.0302897 , 0.01058523, -0.02155854, -0.01185044, -0.00838821,  
 0.01829222, -0.00940945, -0.02990229, -0.00365188, 0.01271348,  
 0.00676229, -0.02760578, -0.03823805, -0.01223098, 0.02425152,  
 0.00206539])

# graph of autocorrelation

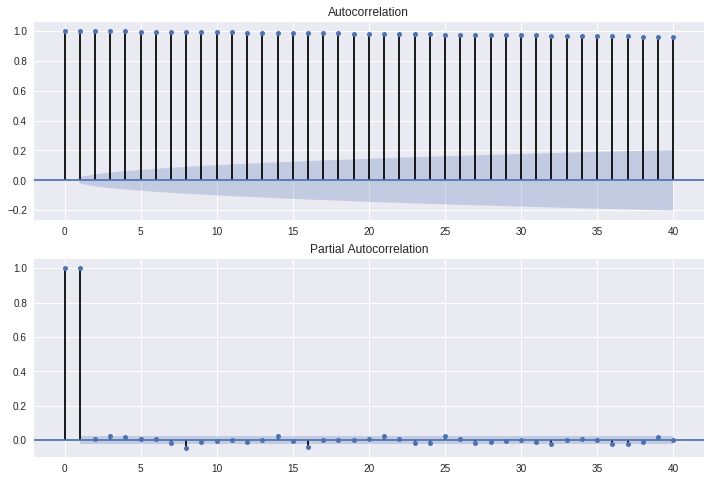
fig = plt.figure(figsize=(12,8))

ax1 = fig.add\_subplot(211)

fig = sm.graphics.tsa.plot\_acf(ts, lags=40, ax=ax1)

ax2 = fig.add\_subplot(212)

fig = sm.graphics.tsa.plot\_pacf(ts, lags=40, ax=ax2)



<code\_above>

Stationarity is not maintained at all, so take the difference and check again.

<code\_below>

# Find Autocorrelation -- Difference Series

diff\_acf = sm.tsa.stattools.acf(diff, nlags=40)

diff\_acf

array([ 1.00000000e+00, -8.56775204e-03, -2.83136053e-02, -2.05382687e-02,  
 -4.78423689e-04, -1.27270025e-02, 1.11971751e-02, 5.11723143e-02,  
 1.76716610e-02, 5.70848634e-05, -8.64011700e-04, 1.08580893e-02,  
 -3.39246954e-03, -3.64525666e-02, 1.24154735e-02, 5.34053751e-02,  
 -1.73294737e-02, -1.33542265e-03, -2.02518812e-03, -3.51557212e-03,  
 -3.37212820e-02, -1.57650077e-02, 2.68959547e-02, 2.48436907e-02,  
 -3.21667294e-02, -1.14623499e-02, 2.27293990e-02, 6.84961528e-03,  
 1.21088533e-03, -1.45360878e-02, 1.27128918e-02, 2.36163318e-02,  
 1.57196961e-03, -1.01756053e-02, -5.62132802e-03, 2.12029236e-02,  
 3.36617272e-02, 1.74116668e-02, -2.19415752e-02, -8.91676238e-03,  
 -1.82411280e-02])

# Partial Autocorrelation -- Difference Series

diff\_pacf = sm.tsa.stattools.pacf(diff, nlags=40, method='ols')

diff\_pacf

array([ 1.00000000e+00, -8.57009208e-03, -2.84061467e-02, -2.10647711e-02,  
 -1.65929682e-03, -1.39443431e-02, 1.04836376e-02, 5.07049519e-02,  
 1.88309961e-02, 3.77958746e-03, 2.19482478e-03, 1.21608835e-02,  
 -1.84976411e-03, -3.67866197e-02, 8.99370605e-03, 4.99269490e-02,  
 -1.77402781e-02, 9.12158614e-04, -2.81688612e-03, -3.28928769e-03,  
 -2.98177973e-02, -1.86244868e-02, 1.99689538e-02, 2.34995752e-02,  
 -3.06998449e-02, -1.09813533e-02, 2.11635527e-02, 1.14406190e-02,  
 7.96925286e-03, -1.87216666e-02, 8.98208497e-03, 2.94635883e-02,  
 3.19930522e-03, -1.31667808e-02, -7.20997457e-03, 2.71532271e-02,  
 3.77610347e-02, 1.17298516e-02, -2.47507718e-02, -2.55091804e-03,  
 -1.77858076e-02])

# Autocorrelation Graph -- Difference Series

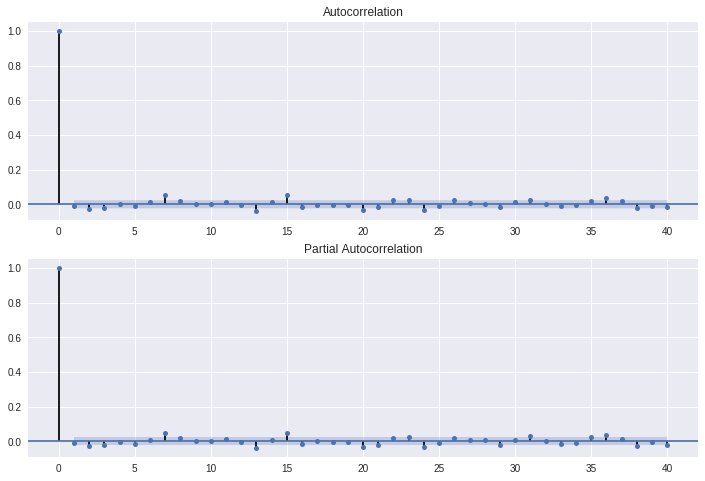
fig = plt.figure(figsize=(12,8))

ax1 = fig.add\_subplot(211)

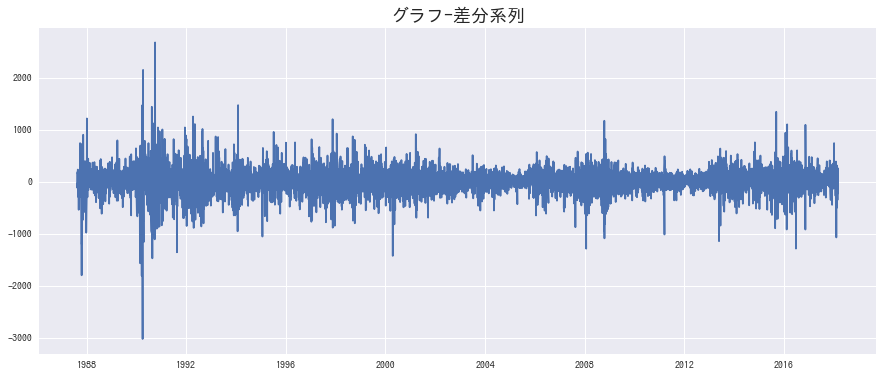
fig = sm.graphics.tsa.plot\_acf(diff, lags=40, ax=ax1)

ax2 = fig.add\_subplot(212)

fig = sm.graphics.tsa.plot\_pacf(diff, lags=40, ax=ax2)



plt.plot(diff)



<code\_above>

Stationarity is maintained... I feel.

So, let's try ARIMA parameter estimation for the difference series (diff).

<code\_below>

# Execution of automatic ARMA estimation function for differential series (first-order differential version)--Eliminates disturbing NaN with dropna

resDiff = sm.tsa.arma\_order\_select\_ic(diff, ic='aic', trend='nc')

resDiff

>>>

{'aic': 0 1 2  
 0 NaN 98218.905477 98215.050764  
 1 98218.936947 98216.949365 98214.920125  
 2 98215.221971 98214.741907 98201.920126  
 3 98214.081072 98215.997405 98217.804338  
 4 98216.061694 98217.371673 98218.641840, 'aic\_min\_order': (2, 2)}

<code\_above>

I got "'aic\_min\_order': (2, 2)}" in the difference series, so I assumed that (2,0,2) would be good in the original series.

To be precise, ARMA instead of ARIMA because it does not take the difference

<code\_below>

# 'aic\_min\_order': (2, 2)} in the first order difference sequence, so ARIMA(2,0,2) is the best, so simply apply it

# P-2, q=2 was the best, so I modeled it -- but I couldn't calculate it with difference 1, but it worked with difference 2

#In the graph of ACF, PACF, p=0,7,13,15,q=0,7,13,15 seems to be good. So 7, 1, 7 = NG assuming sum = 1

from statsmodels.tsa.arima\_model import ARIMA

ARIMA = ARIMA(ts, order=(2,0,2)).fit(dist=False)

print(ARIMA.params)

#mod = sm.tsa.ARIMA(ts, order=(2,2,2)).fit(trend='nc')

#mod.params

# It's not SARIMA, but I don't see any periodicity

resid = ARIMA.resid

fig = plt.figure(figsize=(12,8))

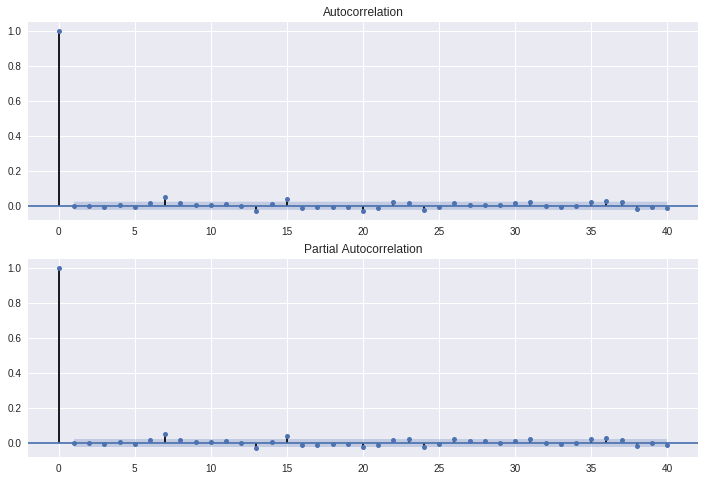
ax1 = fig.add\_subplot(211)

fig = sm.graphics.tsa.plot\_acf(resid.values.squeeze(), lags=40, ax=ax1)

ax2 = fig.add\_subplot(212)

fig = sm.graphics.tsa.plot\_pacf(resid, lags=40, ax=ax2)

>>>

const -0.521828  
ar.L1.D.N225Price -0.867974  
ar.L2.D.N225Price -0.170238  
ar.L3.D.N225Price 0.533285  
ma.L1.D.N225Price 0.850380  
ma.L2.D.N225Price 0.143486  
ma.L3.D.N225Price -0.564095  
dtype: float64  


<code\_above>

I feel that the scale of the vertical axis of the graph is deceiving ("how to trick people with statistics"),

Continue.

<>code\_below>

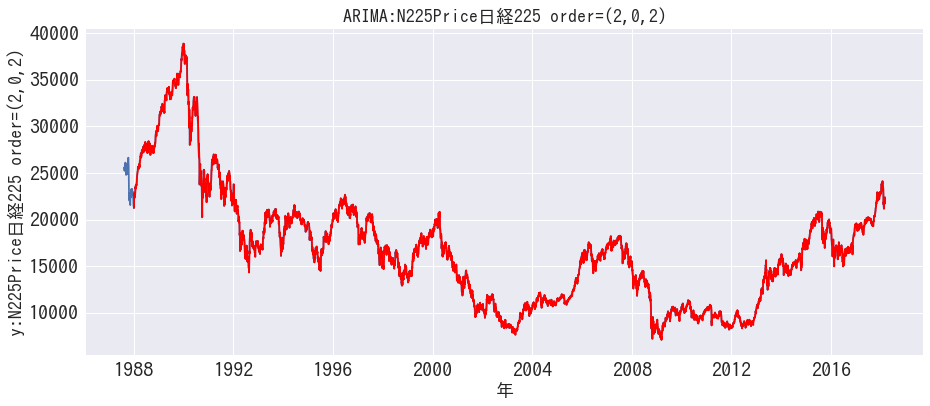
# Prediction

pred = ARIMA.predict('1988-01-04', '2018-03-01')

# Visualization of actual data and prediction results--nice

plt.plot(ts)

plt.plot(pred, "r")



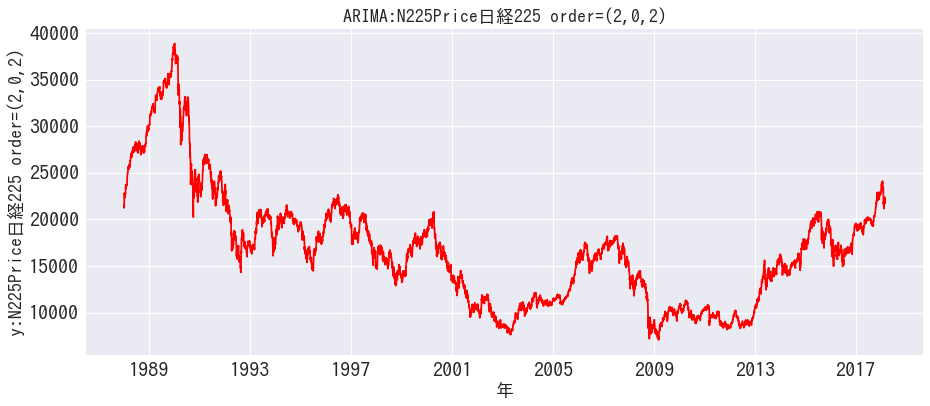
<code\_above>

It fits so well that I was surprised if it was a drawing mistake. So, let's forecast the original series and the forecast separately.

original series



predictor series



For real? I'm afraid that there are some big pitfalls to be FIT.

#validation

ts\_pred = ts[len(ts)-len(pred):len(ts)]

#index for validation

# http://docs.w3cub.com/statsmodels/generated/statsmodels.tools.eval\_measures.vare/

import statsmodels

rmse = statsmodels.tools.eval\_measures.rmse(ts\_pred, pred, axis=0)

mse = statsmodels.tools.eval\_measures.mse(ts\_pred, pred, axis=0)

print(mse)

print(rmse)

>>>59880.04082859314

244.70398613139332

２．３．RNN

I'm going to try RNN (LSTM) with keras because it's Nora deep learning.

This is also univariate, only Nikkei 225.

I used the code from https://qiita.com/yukiB/items/5d5b202af86e3c587843. Thank you.

<code\_below>

# library

%matplotlib inline

import seaborn as sns

import numpy as np

import pandas as pd

import statsmodels.api as sm

import datetime

import matplotlib.pyplot as plt

import math

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from sklearn import preprocessing

from keras.layers.core import Dense, Activation

from keras.layers.recurrent import LSTM

#data

!wget https://www.dropbox.com/\*\*\*\*\*\*\*\*\*/TsData002.csv

#

df000 = pd.read\_csv( 'TsData002.csv', index\_col='Date', parse\_dates=['Date'])

#save alias

df001 = df000.copy()

#Eliminate other than the Nikkei 225, which is the target variable

df002 = df001[["N225Price"]]

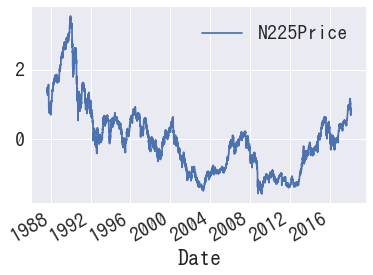
df002.head()

df003 = df002.apply(lambda x: (x-x.mean())/x.std(), axis=0).fillna(0)

df003.head()

df003["N225Price"].head()

df003.plot()



def \_load\_data(data, n\_prev = 100):

"""

data should be pd.DataFrame()

"""

docX, docY = [], []

for i in range(len(data)-n\_prev):

docX.append(data.iloc[i:i+n\_prev].as\_matrix())

docY.append(data.iloc[i+n\_prev].as\_matrix())

alsX = np.array(docX)

alsY = np.array(docY)

return alsX, alsY

def train\_test\_split(df, test\_size=0.1, n\_prev = 100):

"""

This just splits data to training and testing parts

"""

ntrn = round(len(df) \* (1 - test\_size))

ntrn = int(ntrn)

X\_train, y\_train = \_load\_data(df.iloc[0:ntrn], n\_prev)

X\_test, y\_test = \_load\_data(df.iloc[ntrn:], n\_prev)

return (X\_train, y\_train), (X\_test, y\_test)

length\_of\_sequences = 365

(X\_train, y\_train), (X\_test, y\_test) = train\_test\_split(df003[["N225Price"]], n\_prev =length\_of\_sequences)

in\_out\_neurons = 1

hidden\_neurons = 300

model = Sequential()

model.add(LSTM(hidden\_neurons, batch\_input\_shape=(None, length\_of\_sequences, in\_out\_neurons), return\_sequences=False))

model.add(Dense(in\_out\_neurons))

model.add(Activation("linear"))

model.compile(loss="mean\_squared\_error", optimizer="rmsprop")

# early stopping

early\_stopping = keras.callbacks.EarlyStopping(monitor='val\_loss', patience=2)

model.fit(X\_train, y\_train, batch\_size=60, nb\_epoch=15, validation\_split=0.05, callbacks=[early\_stopping])

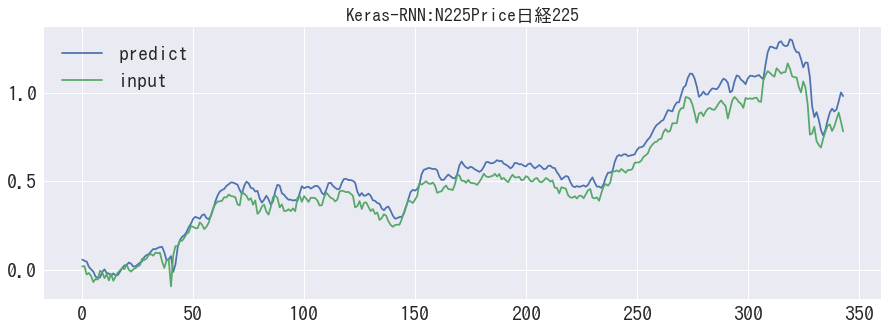
predicted = model.predict(X\_test)

dataf = pd.DataFrame(predicted[:365])

dataf.columns = ["predict"]

dataf["input"] = y\_test[:365]

dataf.plot(figsize=(15, 5))



#MSE

import statsmodels

statsmodels.tools.eval\_measures.mse(dataf['input'],dataf['predict'], axis=0)

>>>0.004287546624338439

#RMSE

statsmodels.tools.eval\_measures.rmse(dataf['input'],dataf['predict'], axis=0)

>>>0.06547936029267878

<code\_above>

The kedo that I did by looking at it, the shape seems to be similar. Feels like it's vertical.

Some degree of deviation seems to be a realistic prediction.

３．Multivariate time series

From here, it is a multivariate time series that finally makes sense of data with 5 multivariate series.

３．１．VAR

I used Python halfway through (order estimation of VAR) and then used R.

<Python\_code\_below>

# Preparation

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

import statsmodels.api as sm

#import quandl # API wrapper for downloading from data site Quandl. You can install it with pip install quandl.

import datetime

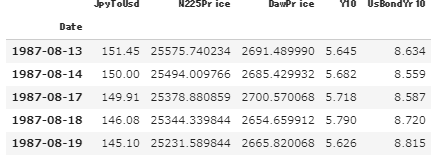
#data

!wget https://www.dropbox.com/s/gbpbbifrtnoyaq6/TsData002.csv

!ls

df000 = pd.read\_csv( 'TsData002.csv', index\_col='Date', parse\_dates=['Date'])

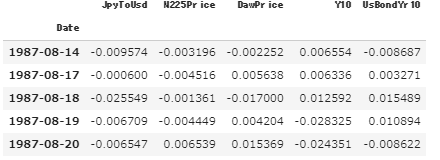
df000.head(5)

#Convert to differential series to achieve stationarity

df001 = df000.pct\_change()

df002 = df001.dropna(how='any')

df002.head()



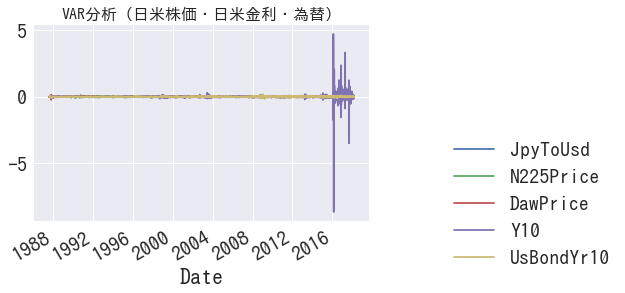
# Take a look at descriptive statistics

df002.describe()



# try plotting. -The exchange rate has been fluctuating recently

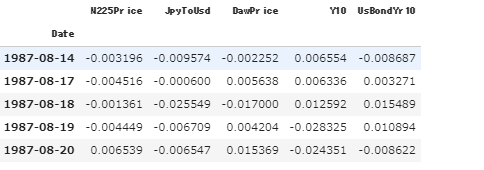
df002.plot().legend(bbox\_to\_anchor=(1.2, 0.5))



#column sorting. Make the leftmost dependent variable N225Price.

df003= pd.concat([df002['N225Price'],df002['JpyToUsd'],df002['DawPrice'],df002['Y10'],df002['UsBondYr10']], axis=1)

df003.head()



# make a VAR model

model = VAR(df003)

#Lag order selection

model.select\_order(365)

>>>\* Minimum  
{'aic': 18, 'bic': 1, 'fpe': 18, 'hqic': 2}

#Set to 18 based on the result of model.select\_order(365) above.

# -> Error could not broadcast input array from shape (18,5,5) into shape (11,5,5)

#I get an error, so change the order to 11

results = model.fit(11)

<Python\_code\_above>

For some reason, I got an error at degree 18, so I decided to try it with R. (Unable to solve due to lack of hacking ability of small job)

<R\_code\_below>

#setwd("~/project/20180304/20180304")

library(KFAS)

library(xts)

library(forecast)

library(urca)

library(ggplot2)

library(ggfortify)

library(tseries)

library(sqldf)

library(lubridate)

library(vars)

library(readr)

### read csv

TsData002 <- read\_csv("TsData002.csv")

#data transformation

TsData002ts <- as.ts(TsData002, start=c(1987,8,13),freq=365)

XtsData002 <- as.xts(TsData002ts, start=c(1987,8,13),freq=365)

head(XtsData002)

#difference sequence for stationarity

XtsData002diff1=diff.default(XtsData002, lag=1, differences=1)

XtsData002diff1$Date <- NULL

#VAR(p=18) estimate model p=18 is up to the analysis on google colab

XtsData002diff1var01=VAR(XtsData002diff1,p=18, type="const")

#VAR(p=11) model outline

summary(XtsData002diff1var01)

>>>（omitted as it is very long）

#Test the model VAR(p=11) for Granger causality.

#(Summary of output) Even with N225Price, which has the highest P-value, p-value = 0.009037, so it is done with a 5% rejection region (no null hypothesis)

causality(XtsData002diff1var01,cause="N225Price")

causality(XtsData002diff1var01,cause="JpyToUsd")

causality(XtsData002diff1var01,cause="DawPrice")

causality(XtsData002diff1var01,cause="Y10")

causality(XtsData002diff1var01,cause="UsBondYr10")

#draw the impulse response function of the model VAR(p=11)

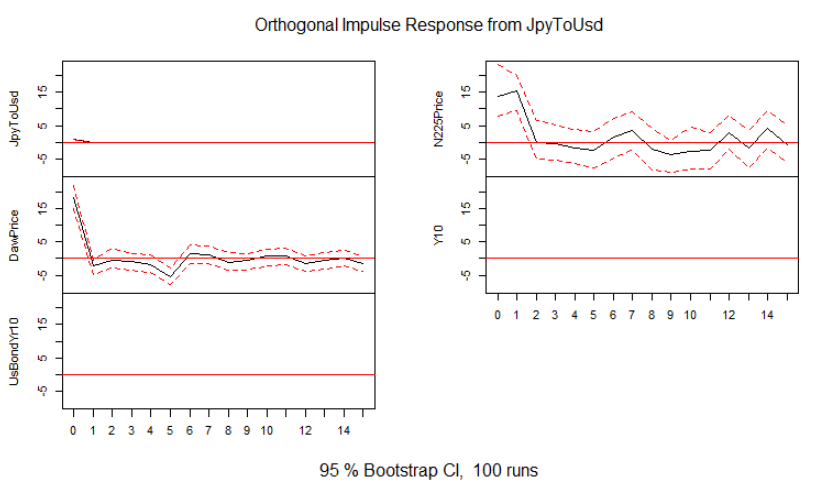
#Since the model has an impulse response function p = 18, draw a lot up to n.ahead = 20

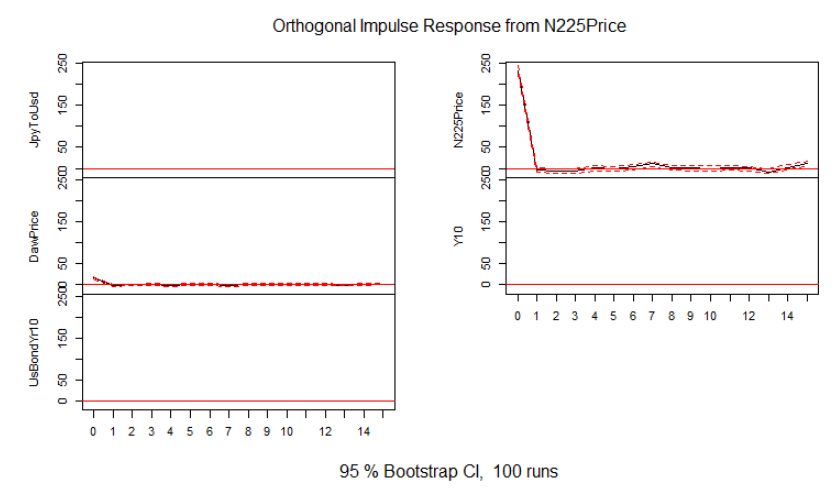
impulse\_func <- irf(XtsData002diff1var01,n.ahead = 20,ci = 0.95,ortho = FALSE)

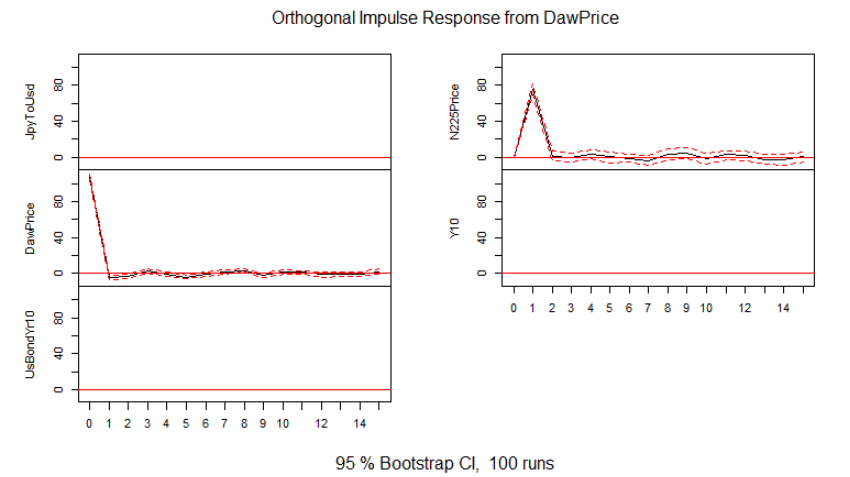
ortho\_impulse\_func <- irf(XtsData002diff1var01,n.ahead = 20,ci = 0.95,ortho = TRUE)

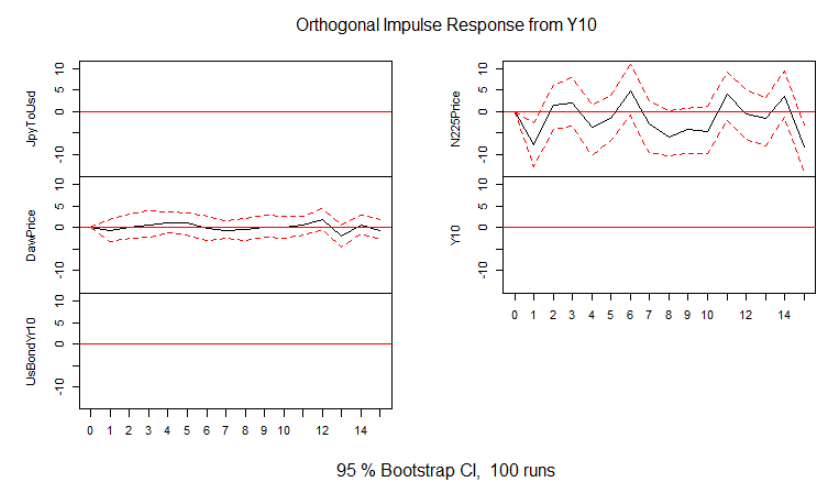
#plot(impulse\_func)

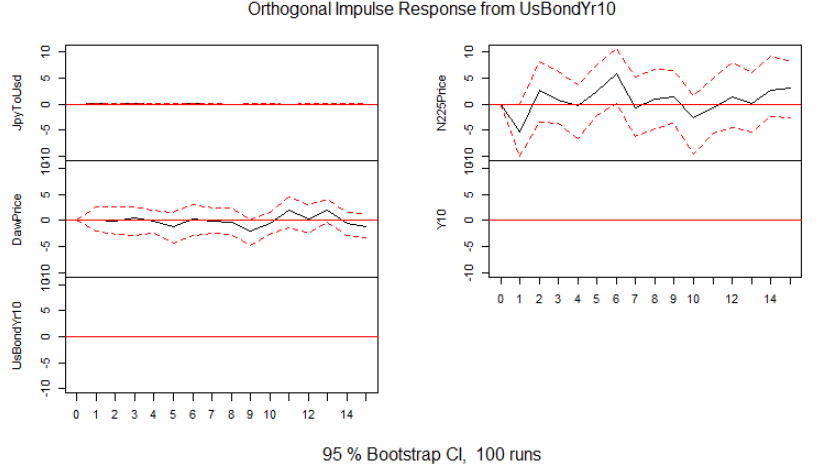
plot(ortho\_impulse\_func)











<R\_code\_above>

The interpretation of the impulse response is

#JpyToUsd Forex: Impacting Dow and N225 Stocks

#N225: Affects self (N225) + slightly affects Dow

#Daw: Affects itself (Daw) and N225

#Y10 (JGB 10 Years): Impacting the Dow and N225

#UsBondY10: Impact on Dow and N225

So, in the end, I feel that modeling with this 5 series with N225 as the objective variable is good.

３．２．Multivariate LSTM

I was allowed to reference.

<https://qiita.com/tizuo/items/b9af70e8cdc7fb69397f>

<code\_below>

# https://keras.io/

!pip install -q keras

# preparation

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

import statsmodels.api as sm

import datetime

import numpy

import matplotlib.pyplot as plt

import pandas

import math

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

#data

!wget https://www.dropbox.com/\*\*\*\*\*\*\*\*\*/TsData002.csv

df000 = pd.read\_csv( 'TsData002.csv', index\_col='Date', parse\_dates=['Date'])

#save alias

df001 = df000.copy()

#Move the target variable Nikkei 225 to the far left

df002 = df001[["N225Price","JpyToUsd","DawPrice","Y10","UsBondYr10"]]

#Standardization

df = df002.values

df = df.astype('float32')

# normalize the dataset

scaler = MinMaxScaler(feature\_range=(0, 1))

df = scaler.fit\_transform(df)

# split into train and test sets

train\_size = int(len(df) \* 0.67)

test\_size = len(df) - train\_size

train, test = df[0:train\_size,:], df[train\_size:len(df),:]

print(len(train), len(test))

#convert an array of values into a dataset matrix

# if you give look\_back 3, a part of the array will be like this: Jan, Feb, Mar

def create\_dataset(dataset, look\_back=1):

dataX, dataY = [], []

for i in range(len(dataset)-look\_back-1):

#Number of rows of data minus look\_back minus 1 (because the last minus 1 means that the python loop counter starts from zero)

xset = []

for j in range(dataset.shape[1]): #Loop range specification meaning over all columns

a = dataset[i:(i+look\_back), j] # From line i to line i + look\_back

xset.append(a)

dataY.append(dataset[i + look\_back, 0])

dataX.append(xset)

return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t and Y=t+1

look\_back = 12

trainX0, trainY0 = create\_dataset(train, look\_back)

testX0, testY0 = create\_dataset(test, look\_back)

print(testX0.shape)

print(testX0.shape[0])

print(testX0.shape[1])

print(testX0.shape[2])

print(testY0.shape)

print(testX0[26,4,])

trainX = numpy.reshape(trainX0, (trainX0.shape[0], trainX0.shape[1], trainX0.shape[2]))

testX = numpy.reshape(testX0, (testX0.shape[0], testX0.shape[1], testX0.shape[2]))

trainY = trainY0.copy()

testY = testY0.copy()

#Developping mode

model = Sequential()

model.add(LSTM(4, input\_shape=(testX.shape[1], look\_back)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

#It takes time, sodation try setting epoch=10 in the trial

#model.fit(trainX, trainY, epochs=1000, batch\_size=1, verbose=2)

model.fit(trainX, trainY, epochs=10, batch\_size=1, verbose=2)

#validation

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

pad\_col = numpy.zeros(df.shape[1]-1)

def pad\_array(val):

return numpy.array([numpy.insert(pad\_col, 0, x) for x in val])

trainPredict = scaler.inverse\_transform(pad\_array(trainPredict))

trainY = scaler.inverse\_transform(pad\_array(trainY))

testPredict = scaler.inverse\_transform(pad\_array(testPredict))

testY = scaler.inverse\_transform(pad\_array(testY))

#Calculation of standard deviation--RMSE varies considerably from run to run

trainScore = math.sqrt(mean\_squared\_error(trainY[:,0], trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

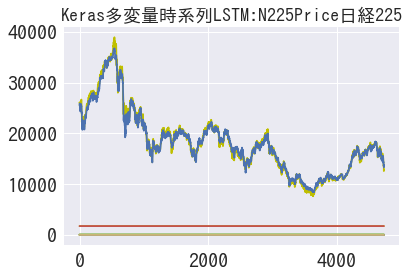
testScore = math.sqrt(mean\_squared\_error(testY[:,0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

Train Score: 577.90 RMSE  
Test Score: 2113.18 RMSE

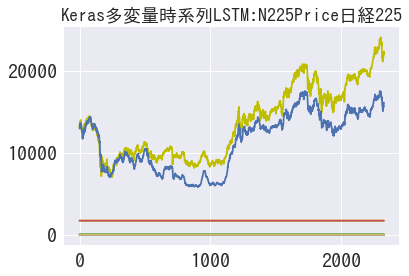
plt.plot(trainY ,"y")

plt.plot(trainPredict)



plt.plot(testY ,"y")

plt.plot(testPredict)



We were able to do reasonable modeling along with univariate RNN.

Summary.

After trying various things, I feel that my Python skills have improved a little. We will continue to learn by collecting, transcribing, interpreting, and modifying code that will serve as samples.

The challenge is performance evaluation comparing different methods. I haven't gotten around to it at all this time.

Next, it seems that I have obtained three years' worth of data on the level of one-ball breaking news in the major leagues (I haven't seen the contents yet), so

I'm wondering whether to try various things there, or to rush to performance evaluation with this topic.