import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import seaborn as sns

%matplotlib inline

import statsmodels.api as sm

from statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from pmdarima import auto\_arima

from statsmodels.graphics.tsaplots import plot\_acf , plot\_pacf

from pandas.plotting import autocorrelation\_plot

from statsmodels.tsa.stattools import adfuller

import statsmodels.tsa.api as smt

from math import sqrt

from sklearn import preprocessing

from sklearn.metrics import r2\_score , mean\_absolute\_error , mean\_absolute\_percentage\_error , mean\_squared\_error

from sklearn.model\_selection import TimeSeriesSplit

from sklearn.linear\_model import LinearRegression

import pymannkendall as mk

import pingouin as pg

from scipy.stats import kruskal

from scipy import stats

import pickle

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv('Merged Data.csv')

df.head()

df.shape

df.isnull().sum()

df.duplicated().sum()

df.dtypes

df.describe()

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace = True)

df.plot(figsize = (10,5)) #Plot Graph

plt.title('Exchange & Policy Rate - THB to USD')

plt.show()

Check\_trend = mk.original\_test(df['Value'])

print(Check\_trend)

Check\_Seasonal = kruskal(df['Policy rate'], df['Value'])

print(Check\_Seasonal)

plt.rcParams['figure.figsize'] = (12,6)

decomposition = seasonal\_decompose(df.Value, period = 12, model = 'additive')

decomposition.plot()

plt.show()

def adf\_check(time\_series):

result = adfuller(time\_series , autolag = 'AIC')

label = pd.Series(result[0:4], index=['Test Statistic','p-value','Number of Lags Used','Number of Observations Used'])

for key,value in result[4].items():

label['Critical Value (%s)'%key] = value

print(label)

if result[1] <= 0.05:

print('Strong evidence against the null hypothesis, hence REJECT null hypothesis and the series is Stationary')

else:

print ('Weak evidence against the null hypothesis, hence ACCEPT null hypothesis and the series is Not Stationary ')

adf\_check(df['Value'])

df1 = df.diff().dropna()

print('Count of value', df1.shape[0])

df1.head()

adf\_check(df1.Value)

fig, (ax1,ax2) = plt.subplots(nrows = 2, ncols = 1, sharex = False, sharey = False, figsize = (12,6))

ax1 = autocorrelation\_plot(df1, ax = ax1)

ax1.set\_title('Non-Stationary Data')

ax2 = autocorrelation\_plot(df1 , ax = ax2)

ax2.set\_title('Stationary Data')

plt.subplots\_adjust(hspace = 0.5)

plt.show()

model\_arimax = auto\_arima(df['Value'],

exog = df['Policy rate'],

m = 12,

d = 1,

seasonal = False,

max\_order = 8,

test = 'adf',

trace = True)

model\_arimax.summary()

model\_arimax1 = SARIMAX(df['Value'],

order = (0,1,1),

seasonal\_order = (0,0,0,0),

exog = df['Policy rate'],

freq = 'M',

enforce\_stationarity = False,

enforce\_invertibility = False)

result = model\_arimax1.fit(disp = False)

result.summary()

Ljingbox = sm.stats.acorr\_ljungbox(result.resid,

lags = 5,

return\_df = True)

print(Ljingbox)

result.plot\_diagnostics(figsize = (12,7))

plt.subplots\_adjust(hspace = 0.5)

plt.show()

predictions = result.predict(typ = 'levels')

print('Evaluation Result for whole data : ','\n')

print('R2 Score for whole data : {0:.2f} %'.format(100\*r2\_score(df['Value'],predictions)),'\n')

print('Mean Squared Error : ',mean\_squared\_error(df['Value'],predictions),'\n')

print('Mean Absolute Error : ',mean\_absolute\_error(df['Value'],predictions),'\n')

print('Root Mean Squared Error : ',sqrt(mean\_squared\_error(df['Value'],predictions)),'\n')

print('Mean Absolute Percentage Error : {0:.2f} %'.format(100\*mean\_absolute\_percentage\_error(df['Value'],predictions)))

Final = pd.concat([df, df1, predictions], axis = 1)

Final.columns = ['Foreign Exchange Rate (monthly)',

'Policr Rate (monthly)',

'Monthly First Difference',

'Predicted Policy Rate',

'Predicted Exchange Rate']

Final.head()

train\_size = 5 # Size of the training data

test\_size = 1 # Size of the test data

def walk\_forward\_optimization(df, train\_size, test\_size, start\_month, last\_model = None):

predictions = []

mse\_values = []

actual\_values = []

current\_month = start\_month

model\_fit = None

for end\_month in range(start\_month + train\_size, len(df) - test\_size + 1):

# Slice training data

value\_data = df['Value'].iloc[start\_month - 1:end\_month]

policy\_rate\_data = df['Policy rate'].iloc[start\_month - 1:end\_month]

# train\_data is used to fit the ARIMA model

train\_data = np.stack((value\_data.values, policy\_rate\_data.values), axis=1)

# Fit ARIMA model

model = ARIMA(train\_data[:, 0], exog=train\_data[:, 1], order=(0, 1, 1))

model\_fit = model.fit()

# Predict only 1 step ahead (test\_size)

next\_month\_policy\_rate = df['Policy rate'].iloc[[end\_month]].values

test\_prediction = model\_fit.forecast(steps=test\_size, exog=next\_month\_policy\_rate)[0]

# Calculate MSE using actual value at next time step

actual\_value = df['Value'].iloc[end\_month]

mse\_values.append(mean\_squared\_error([actual\_value], [test\_prediction]))

predictions.append(test\_prediction)

actual\_values.append(actual\_value)

# Increase current month

current\_month += 1

return predictions, mse\_values, actual\_values, current\_month, model\_fit

forecast\_months = 3

start\_month = 1

rmse\_values\_all = []

predictions\_all = []

actual\_values\_all = []

last\_model = None

# Loop until end of data

while start\_month <= len(df) - forecast\_months:

# Run walk-forward optimization

predictions, mse\_values, actual\_values, current\_month, last\_model = walk\_forward\_optimization(df, train\_size, test\_size, start\_month, last\_model)

# Process and analyze results (calculate RMSE, plot graphs, etc.)

rmse = np.sqrt(np.mean(mse\_values))

rmse\_values\_all.append(rmse)

predictions\_all.extend(predictions) # Extend to keep all predictions

actual\_values\_all.extend(actual\_values) # Extend to keep all actual values

# Update start month for next iteration

start\_month += 1

print('Printing Predictied vs Expected Values....')

print('\n')

for predicted, actual in zip(predictions\_all, actual\_values\_all):

print('Predicted = %f , Actual = %f' % (predicted, actual))

print('Evaluation Result for Test data : ','\n')

print('R2 Score for Test data: {0:.2f} %'.format(100\*r2\_score(actual\_values\_all, predictions\_all)),'\n')

print('Mean Squared Error: ', mean\_squared\_error(actual\_values\_all, predictions\_all),'\n')

print('Mean Absolute Error: ', mean\_absolute\_error(actual\_values\_all, predictions\_all),'\n')

print('Root Mean Squared Error: ', sqrt(mean\_squared\_error(actual\_values\_all, predictions\_all)),'\n')

print('Mean Absolute Percentage Error: {0:.2f} %'.format(100\*mean\_absolute\_percentage\_error(actual\_values\_all, predictions\_all)),'\n')

svr

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

import os

import seaborn as sns

%matplotlib inline

import statsmodels.api as sm

from statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from pmdarima import auto\_arima

from statsmodels.graphics.tsaplots import plot\_acf , plot\_pacf

from pandas.plotting import autocorrelation\_plot

from statsmodels.tsa.stattools import adfuller

from math import sqrt

from sklearn import preprocessing

from sklearn.metrics import r2\_score , mean\_absolute\_error , mean\_absolute\_percentage\_error , mean\_squared\_error

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVR

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

import pickle

import warnings

warnings.filterwarnings('ignore')

import os

data = pd.read\_csv('Merged Data.csv')

data.head()

X = pd.to\_datetime(data['Date']).astype('int64').values.reshape(-1, 1)

policy\_rate = data['Policy rate'].values.reshape(-1, 1)

X = np.concatenate((X, policy\_rate), axis=1) # เพิ่ม Policy rate เข้าไปใน feature

y = data['Value']

# แบ่งข้อมูลเป็นชุดฝึกและชุดทดสอบ

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# ทำการสเกลข้อมูล

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# ฝึกโมเดล SVR

svr\_model = SVR(kernel='rbf', C=100, gamma='auto') # ตั้งค่าพารามิเตอร์ของ SVR

svr\_model.fit(X\_train\_scaled, y\_train)

predictions = []

for i in range(3):

# นับเวลา i เดือนหลังจากเดือนปัจจุบัน

next\_month = pd.to\_datetime('2024-06-01') + pd.DateOffset(months=i)

next\_month = np.array([[next\_month.timestamp()]]) # แปลงให้มีรูปแบบเหมือนกับ next\_month\_policy\_rate

# หาค่าเฉลี่ยของอัตราการเปลี่ยนแปลงของอัตราดอกเบี้ย

your\_policy\_rate\_value = data['Policy rate'].diff().mean()

# ระบุค่า Policy rate ของเดือนถัดไป

next\_month\_policy\_rate = np.array([[your\_policy\_rate\_value]])

# ทำการสเกลข้อมูล

next\_month\_combined = np.concatenate((next\_month, next\_month\_policy\_rate), axis=1)

next\_month\_scaled = scaler.transform(next\_month\_combined)

# ทำการทำนาย

next\_month\_prediction = svr\_model.predict(next\_month\_scaled)

predictions.append(next\_month\_prediction)

print("Predictions for the next 3 months:", predictions)

y\_pred = svr\_model.predict(X\_test\_scaled)

print('Evaluation Result for Test data : ','\n')

print('R2 Score for Test data: {0:.2f} %'.format(100\*r2\_score(y\_test, y\_pred)),'\n')

print('Mean Squared Error: ', mean\_squared\_error(y\_test, y\_pred),'\n')

print('Mean Absolute Error: ', mean\_absolute\_error(y\_test, y\_pred),'\n')

print('Root Mean Squared Error: ', sqrt(mean\_squared\_error(y\_test, y\_pred)),'\n')

print('Mean Absolute Percentage Error: {0:.2f} %'.format(100\*mean\_absolute\_percentage\_error(y\_test, y\_pred)),'\n')

# พล็อตกราฟ

plt.figure(figsize=(10, 6))

plt.plot(data['Date'], data['Value'], label='Actual') # พล็อตข้อมูลจริง

for i, prediction in enumerate(predictions):

next\_month\_date = pd.to\_datetime('2024-06-01') + pd.DateOffset(months=i) # วันที่ของเดือนถัดไป

next\_month\_date\_str = next\_month\_date.strftime('%Y-%m-%d') # แปลงเป็นรูปแบบที่ matplotlib เข้าใจได้

plt.plot(next\_month\_date\_str, prediction, 'ro', label=f'Prediction {i+1} Month Ahead') # พล็อตข้อมูลการทำนาย

plt.xlabel('Date')

plt.ylabel('Value')

plt.title('Actual vs Predicted Values')

plt.legend()

plt.savefig('Foreign Exchange Rate Prediction SVR.png')

plt.show()

plt.rcParams['figure.figsize'] = (12, 6)

fig, ax = plt.subplots(2, 1, sharex=True)

ax[0].plot(data['Date'][:len(y\_pred)], y[:len(y\_pred)], 'o', label='Actual')

ax[0].set\_ylabel('Foreign Exchange Rate')

ax[1].plot(data['Date'][:len(y\_pred)], y\_pred, 'r', label='Forecast')

ax[1].set\_xlabel('Date')

ax[1].set\_ylabel('Foreign Exchange Rate')

for ax\_ in ax:

legend = ax\_.legend(loc='upper left')

legend.get\_frame().set\_facecolor('w')

plt.show()