from datetime import datetime  
from sklearn.model\_selection import train\_test\_split  
from statsmodels.tsa.seasonal import seasonal\_decompose  
import plotly.express as px  
import statsmodels.api as sm  
from statsmodels.tsa.stattools import adfuller  
from statsmodels.tsa.arima\_model import ARIMA  
import pmdarima as pm  
import numpy as np  
import seaborn as sns  
import pandas as pd  
import matplotlib.pyplot as plt  
from statsmodels.graphics.tsaplots import plot\_acf  
from statsmodels.graphics.tsaplots import plot\_pacf  
from scipy.stats.mstats import winsorize  
from pmdarima import auto\_arima  
from statsmodels.tsa.holtwinters import Holt  
from statsmodels.tsa.holtwinters import ExponentialSmoothing  
from scipy.stats import pearsonr, spearmanr, kendalltau

import warnings  
warnings.filterwarnings('ignore')

df1 = pd.read\_csv('EXPCH.csv',parse\_dates=True,index\_col='DATE')  
df2 = pd.read\_csv('IMPCH.csv',parse\_dates=True,index\_col='DATE')  
print(df1)  
print(df2)

df1.shape

df2.shape

#Check the null values in each column   
print(df1.isnull().sum())  
print(df2.isnull().sum())

merged\_df=df1  
merged\_df['IMPCH']=df2['IMPCH']  
merged\_df.head()

correlation = merged\_df['EXPCH'].corr(merged\_df['IMPCH'])  
  
# Print the correlation coefficient  
print(f"Correlation between the two datasets: {correlation\*100}")

merged\_df.plot(figsize=(15,5),legend=True)

df1.info()  
df2.info()

df1.index  
df2.index

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Assuming df1 and df2 are your DataFrames containing the covariate variables  
# For example, df1['covariate1'] and df2['covariate2'] are the covariate variables  
  
# Plotting covariate variables from df1 and df2 against each other  
sns.scatterplot(x=df1['EXPCH'], y=df2['IMPCH'],color='violet')  
plt.xlabel('Covariate 1 from EXPCH')  
plt.ylabel('Covariate 2 from IMPCH')  
plt.title('Scatter Plot of Covariate 1 from EXPCH vs Covariate 2 from IMPCH')  
plt.show()

Seasonality Check

from statsmodels.tsa.seasonal import seasonal\_decompose  
result = seasonal\_decompose(df1['EXPCH'],period=12) # Assuming monthly data with a seasonal period of 12  
plt.figure(figsize=(20, 8))  
result.plot()  
plt.show()

from statsmodels.tsa.seasonal import seasonal\_decompose  
result = seasonal\_decompose(df2['IMPCH'],period=12) # Assuming monthly data with a seasonal period of 12  
plt.figure(figsize=(18, 8))  
result.plot()  
plt.show()

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

fig, ax = plt.subplots(figsize=(12,5))  
plot\_acf(df1['EXPCH'],lags=40,ax=ax);

fig, ax = plt.subplots(figsize=(12,5))  
plot\_pacf(df1['EXPCH'],lags=25,ax=ax);

ACF and PACF for IMPCH

fig, ax = plt.subplots(figsize=(12,5))  
plot\_acf(df2['IMPCH'],lags=40,ax=ax);

fig, ax = plt.subplots(figsize=(12,5))  
plot\_pacf(df2['IMPCH'],lags=25,ax=ax);

from statsmodels.tsa.stattools import adfuller  
  
def adf\_test(series,title=''):  
 """  
 Pass in a time series and an optional title, returns an ADF report  
 """  
 print(f'Augmented Dickey-Fuller Test: {title}')  
 result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles differenced data  
   
 labels = ['ADF test statistic','p-value','# lags used','# observations']  
 out = pd.Series(result[0:4],index=labels)  
  
 for key,val in result[4].items():  
 out[f'critical value ({key})']=val  
   
 print(out.to\_string()) # .to\_string() removes the line "dtype: float64"  
   
 if result[1] <= 0.05:  
 print("Strong evidence against the null hypothesis")  
 print("Reject the null hypothesis")  
 print("Data has no unit root and is stationary")  
 else:  
 print("Weak evidence against the null hypothesis")  
 print("Fail to reject the null hypothesis")  
 print("Data has a unit root and is non-stationary")

result1=adf\_test(df1['EXPCH'],title='')

result2=adf\_test(df2['IMPCH'],title='')

As p value for both the Time series is more then 0.05 we can say its not stationary and need differencing.

from pmdarima import auto\_arima  
  
# Assuming 'values' is the column you want to use for time series forecasting  
values = merged\_df['EXPCH']  
  
# Perform auto\_arima on the numerical values  
stepwise\_fit = auto\_arima(values,exogenous=merged\_df['IMPCH'],test='adf',suppress\_warnings=True,seasonal=True,m=12)  
  
# Print the summary of the fitted model  
print(stepwise\_fit.summary())

from statsmodels.tsa.statespace.sarimax import SARIMAX

train=merged\_df.iloc[:421]  
test=merged\_df.iloc[421:]  
print(train.shape,test.shape)

model = SARIMAX(train['EXPCH'],exog=train['IMPCH'],order=(2,1,2),seasonal\_order=(2,0,2,12),enforce\_invertibility=True)  
results = model.fit()  
  
# Print the model summary  
print(results.summary())

start=len(train)  
end=len(train)+len(test)-1  
exog\_forecast=test['IMPCH']  
prediction=results.predict(start=start,end=end,exog=exog\_forecast).rename('prediction')

prediction.head()

test.head()

from sklearn.metrics import mean\_squared\_error  
error=np.sqrt(mean\_squared\_error(test["EXPCH"],prediction))  
print(error)

test['EXPCH'].plot(figsize=(12,8),legend=True)  
prediction.plot(legend=True)

auto\_arima(merged\_df['IMPCH'],seasonal=True,m=12).summary()

model\_IMPCH = SARIMAX(merged\_df['IMPCH'],order=(3,1,0),seasonal\_order=(2, 0, [1, 2], 12))  
results = model\_IMPCH.fit()  
IMPCH\_PRED = results.predict(len(merged\_df),len(merged\_df)+47).rename('Forecasted\_IMPCH')  
print(IMPCH\_PRED)

len(IMPCH\_PRED)

new\_df=pd.DataFrame({'IMPCH':IMPCH\_PRED,'EXPCH': np.nan})  
df\_new = pd.concat([merged\_df, new\_df])  
df\_new.head()

df\_new.tail()

len(df\_new)

fcast\_start=len(merged\_df)  
exog\_forecast=df\_new[421:]['IMPCH']  
fcast\_val=results.predict(start=fcast\_start,end=fcast\_start+47,exog=exog\_forecast)

fcast\_val

merged\_df.plot(figsize=(12,8),legend=True,color='blue')  
IMPCH\_PRED.plot(legend=True,color='red')