%%% This is the code of Forecasting using ARIMA based on historical wind output power data as ARIMA is of univariant this is for 1 hour and the same is followed for the remaining hours%%%

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import os

import pandas as pd

import numpy as np

import seaborn as sns

from datetime import datetime, timedelta

import matplotlib.pyplot as plt

import itertools

# In[2]:

import random

# In[3]:

pip install pmdarima

# In[4]:

df=pd.read\_csv('D:/CSV files WT/WT1-1h.csv')

# In[5]:

df.head(10)

# In[6]:

df.tail()

# In[7]:

df.describe()

# In[8]:

df['1'].plot()

# In[9]:

df.index = pd.to\_datetime(df.index)

# In[10]:

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

# In[11]:

df.dropna(inplace=True)

# In[12]:

df.fillna(0)

df=df.fillna(df.bfill())

# In[13]:

from pmdarima.arima import ADFTest

# In[14]:

adf\_test = ADFTest(alpha =0.05)

adf\_test.should\_diff(df)

df.fillna(value=0, inplace=True)

# In[15]:

from statsmodels.tsa.stattools import adfuller

# In[16]:

adf, pvalue, usedlag\_, nobs\_, critical\_values\_, icbest\_ = adfuller(df)

print("pvalue=", pvalue, "if above 0.05, data is not stationary")

# In[17]:

from statsmodels.tsa.seasonal import seasonal\_decompose

# In[18]:

decomposed = seasonal\_decompose(df['1'], model='additive', period=2)

trend = decomposed.trend

seasonal = decomposed.seasonal

residual = decomposed.resid

# In[19]:

plt.style.use('dark\_background')

# In[20]:

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

plt.subplot(411)

plt.plot(df, label ='original', color= 'yellow')

plt.legend(loc='upper left')

plt.subplot(412)

plt.plot(trend, label ='trend', color= 'yellow')

plt.legend(loc='upper left')

plt.subplot(413)

plt.plot(seasonal, label ='seasonal', color= 'yellow')

plt.legend(loc='upper left')

plt.subplot(414)

plt.plot(residual, label ='residual', color= 'yellow')

plt.legend(loc='upper left')

plt.show()

# In[21]:

from pmdarima.arima import auto\_arima

# In[22]:

arima\_model = auto\_arima(df['1'], start\_p =1, d=1, start\_q =1, max\_q=5, max\_d =5, m=6, start\_P=0, D=1, start\_Q=0, max\_P=5, max\_D=5, max\_Q=5, seasonal=True, trace=True, error\_action='ignore',suppress\_warnings=True, stepwise=True, n\_fits=50)

# In[23]:

# to print summary

print(arima\_model.summary())

# In[24]:

#Model SARIMAX(4, 1, 0)x(0, 1, 1, 6)

# In[25]:

size = int(len(df)\*0.66)

X\_train, X\_test = df[0:size], df[size:len(df)]

# In[26]:

from statsmodels.tsa.statespace.sarimax import SARIMAX

# In[27]:

import warnings

warnings.filterwarnings('ignore')

# In[28]:

model =SARIMAX(X\_train['1'], order = (4,1,0), seasonal\_order = (0,1,1,6))

# In[29]:

result = model.fit()

result.summary()

# In[30]:

start\_index = 0

end\_index = len(X\_train)-1

train\_prediction = result.predict(start\_index, end\_index)

# In[31]:

start\_index = len(X\_train)

end\_index = len(df)-1

prediction = result.predict(start\_index, end\_index).rename('predicted Time')

# In[32]:

prediction

# In[33]:

prediction.plot(legend=True)

X\_test['1'].plot(legend=True)

# In[34]:

# Calculate residuals from the ARIMA model

residuals = arima\_model.resid

# In[35]:

# Create a DataFrame from the residuals

residuals\_df = pd.DataFrame(residuals, columns=['residuals'])

# In[ ]:

# Align residuals DataFrame index with the original data index

residuals\_df.index = X\_train.index

# In[ ]:

# In[104]:

# Create a DataFrame from the residuals

residuals\_df = pd.DataFrame(residuals, columns=['residuals'])

# In[99]:

# plot residual erros

residuals = pd.dataframe(arima\_model.fit.resid)

residuals.plot()

residuals.plot(kind='kde')

plt.show()

print(residuals.describe())

# In[36]:

import math

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import mean\_absolute\_percentage\_error

# In[37]:

from sklearn.ensemble import StackingRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.linear\_model import LinearRegression

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

# In[38]:

# Initialize Random Forest Regressor

random\_forest = RandomForestRegressor(n\_estimators=100, random\_state=42)

# In[39]:

# Initialize Gradient Boosting Regressor

gradient\_boosting = GradientBoostingRegressor(n\_estimators=100, learning\_rate=0.1, random\_state=42)

# In[40]:

# Initialize meta-regressor (e.g., Linear Regression)

meta\_regressor = LinearRegression()

# In[41]:

# Initialize Stacking Regressor

stacking\_regressor = StackingRegressor(

estimators=[('random\_forest', random\_forest), ('gradient\_boosting', gradient\_boosting)],

final\_estimator=meta\_regressor

)

# In[47]:

# Train the Stacking Regressor

stacking\_regressor.fit(X\_train, y\_train)

# In[42]:

trainScore = math.sqrt(mean\_squared\_error(X\_train, train\_prediction))

print('trainScore:% 2f RMSE'% (trainScore))

testScore = math.sqrt(mean\_squared\_error(X\_test, prediction))

print('testScore:% 2f RMSE'% (testScore))

# In[43]:

# Calculate MAE for training set

train\_mae = mean\_absolute\_error(X\_train, train\_prediction)

print('Train MAE: %.2f' % train\_mae)

test\_mae = mean\_absolute\_error(X\_test, prediction)

print('Test MAE: %.2f' % test\_mae)

# In[44]:

# Calculate MAPE for training set

train\_mape = mean\_absolute\_percentage\_error(X\_train, train\_prediction)

print('Train MAPE: %.2f' % train\_mape)

test\_mape = mean\_absolute\_percentage\_error(X\_test, prediction)

print('Test MAPE: %.2f' % test\_mape)

# In[45]:

from sklearn.metrics import r2\_score

score = r2\_score(X\_test, prediction)

print("R2 score is:", score)

# In[104]:

forecast1 = result.predict(start=len(df)-53, end = (len(df-1))+1\*12, tup = 'levels').rename('forecast1')

# In[87]:

forecast = result.predict(start=len(df)-48, end = (len(df)-1)+1\*12, tup = 'levels').rename('forecast')

# In[88]:

forecast

# In[105]:

print(forecast1)

# In[41]:

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

plt.plot(X\_train, label = 'Training', color = 'green')

plt.plot(X\_test, label = 'Test', color = 'yellow')

plt.plot(forecast, label='forecast', color = 'cyan')

plt.legend(loc='upper left')

plt.show()

# In[69]:

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

plt.plot(X\_train, label = 'Training', color = 'green')

plt.plot(X\_test, label = 'Test', color = 'yellow')

plt.plot(forecast1, label='forecast', color = 'cyan')

plt.legend(loc='upper left')

plt.show()

%%% code for Reducing Scenarios using K-Means Clustering%%%%

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

# In[2]:

import pandas as pd

import seaborn as sns

# In[3]:

df = pd.read\_csv('C:/Users/pavan/OneDrive/Desktop/Var with uncertnity/PV-forecast-kmeans.csv')

# In[4]:

df.isna().sum()

# In[5]:

df.head()

# In[6]:

df.shape

# In[7]:

df.describe()

# In[8]:

print("The shape of data is", df.shape)

# In[9]:

df.tail()

# In[10]:

# Calculate probability for each column

column\_probabilities = {}

total\_rows = len(df)

# In[11]:

for column in df.columns:

value\_counts = df[column].value\_counts()

# In[12]:

# Calculate probability of each unique value

probabilities = value\_counts / total\_rows

# In[13]:

# Store probabilities for the column

column\_probabilities[column] = probabilities

# In[14]:

# Print probabilities for each column

for column, probabilities in column\_probabilities.items():

print(f"Probabilities for column {column}")

print(probabilities)

print()

# In[15]:

print(probabilities)

print()

# In[16]:

# Sample a subset of scenarios

sampled\_data = df.sample(n=5, replace=False)

# In[17]:

# Optionally, you can export the sampled data to a new CSV file

sampled\_data.to\_csv('sampled\_dataset.csv', index=False)

# In[18]:

#Reduced Scenarios

sampled\_data

# In[19]:

from openpyxl import Workbook

# In[20]:

# Create a new Excel workbook

wb = Workbook()

# In[21]:

# Create a new worksheet

ws = wb.active

# In[22]:

# Your print output

print\_output = sampled\_data

# In[23]:

# Save the Excel file

wb.save('print\_output.xlsx')

# In[24]:

#Reduced Scenarios for wind plant

# In[25]:

df1 = pd.read\_csv('C:/Users/pavan/OneDrive/Desktop/Var with uncertnity/WT-Forecast-kmeans.csv')

# In[26]:

df1.isna().sum()

# In[27]:

df1.head()

# In[28]:

print("The shape of data is", df1.shape)

# In[29]:

df1.tail()

# In[30]:

# Calculate probability for each column

column\_probabilities = {}

total\_rows = len(df1)

# In[31]:

for column in df1.columns:

value\_counts = df1[column].value\_counts()

# In[32]:

# Calculate probability of each unique value

probabilities = value\_counts / total\_rows

# In[33]:

# Store probabilities for the column

column\_probabilities[column] = probabilities

# In[34]:

# Print probabilities for each column

for column, probabilities in column\_probabilities.items():

print(f"Probabilities for column {column}")

print(probabilities)

print()

# In[35]:

# Sample a subset of scenarios

sampled\_data1 = df1.sample(n=10, replace=False)

# In[36]:

# Optionally, you can export the sampled data to a new CSV file

sampled\_data1.to\_csv('sampled\_dataset.csv', index=False)

# In[37]:

#Reduced Scenarios

sampled\_data1

# In[38]:

# Sample a subset of scenarios

sampled\_data1 = df1.sample(n=10, replace=False)

# In[39]:

# Specify the number of scenarios you want to retain

num\_samples = 10;

# In[48]:

# Randomly sample scenarios

idx = randperm(size(df, 1), num\_samples);

sampled\_data = df(idx,:);

# In[45]:

import matplotlib.pyplot as plt

# In[103]:

x=df.iloc[:,[3,4,5,6]].values

# In[104]:

from sklearn.cluster import KMeans

# In[105]:

wcss=[]

# In[ ]:

for i in range(1,11)

# In[97]:

#plt.scatter(df['11h'],df['12h'],df['13h'],df['14h'],df['15h'],df['16h'],df['17h'],df['18h'],df['19h'],df['20h'])

# In[47]:

print(df.isnull().sum())

# In[49]:

from sklearn.cluster import KMeans

# In[50]:

from sklearn.impute import SimpleImputer

from sklearn.impute import MissingIndicator

# In[51]:

imputer = SimpleImputer(strategy='mean') # You can use other strategies like 'median' or 'most\_frequent'

df\_imputed = imputer.fit\_transform(df)

# In[52]:

import warnings

warnings.filterwarnings('ignore')

# In[53]:

wcss=[]

# In[54]:

for i in range(1,11):

kmeans=KMeans(n\_clusters=i, init="k-means++", random\_state=42)

kmeans.fit(x)

wcss.append(kmeans.inertia\_)

# In[ ]:

plt.plot(range(1,11),wcss)

# In[ ]:

plt.plot(range(1,11),wcss)

plt.title('The elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('wcss')

plt.show()

# In[55]:

#Fitting kmeans to dataset

# In[56]:

kmeans=KMeans(n\_clusters=5, init="k-means++", random\_state=42)

# In[57]:

y\_kmeans = kmeans.fit\_predict(x)

# In[58]:

y\_kmeans

# In[59]:

x[y\_kmeans == 3,0]

# In[126]:

plt.scatter(x[y\_kmeans == 0,0], x[y\_kmeans == 0,1], color ='blue')

plt.scatter(x[y\_kmeans == 1,0], x[y\_kmeans == 1,1], color ='red')

plt.scatter(x[y\_kmeans == 2,0], x[y\_kmeans == 2,1], color ='green')

plt.scatter(x[y\_kmeans == 3,0], x[y\_kmeans == 3,1], color ='yellow')

plt.scatter(x[y\_kmeans == 4,0], x[y\_kmeans == 4,1], color ='yellow')

# In[ ]:

# In[ ]:

# In[94]:

wcss.append(km.inertia\_)

# In[95]:

wcss

# In[96]:

plt.plot(range(10,11),wcss)

# In[79]:

X=df.iloc[:,i].values

# In[80]:

km=KMeans(n\_clusters=5)

# In[89]:

arr\_2d = X.reshape(-1, 1)

arr\_2d = np.array(X).reshape(-1, 1)

# In[120]:

Y\_means=km.fit\_predict(x)

%%% Optimization using GAMS %%%%

sets

t time /1\*24/

i DGs /1\*5/

s senarios /1\*5/

;

table PV(s,t) 'PV forecast power'

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1 0 0 0 0 0 0 0 0 0 0.23 1.111 1.835 2.287 2.555 2.673 2.57 1.815 1.353 0.944 0.116 0 0 0 0

2 0 0 0 0 0 0 0 0 0 0.92 1.754 2.509 3.704 4.516 4.703 4.541 3.96 3.063 2.138 0.645 0 0 0 0

3 0 0 0 0 0 0 0 0 0 0.82 1.659 2.401 3.632 4.152 4.351 4.212 3.908 3 1.985 0.564 0 0 0 0

4 0 0 0 0 0 0 0 0 0 0.71 1.534 2.266 3.567 3.926 4.126 3.988 3.711 2.805 1.832 0.479 0 0 0 0

5 0 0 0 0 0 0 0 0 0 0.27 1.239 2.071 2.825 3.457 3.631 3.477 2.913 2.103 1.254 0.15 0 0 0 0

;

table WT(s,t) 'WT forecast power'

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1 14.445 10.027 9.598 7.939 11.111 8.211 4.726 4.806 9.932 9.878 15.356 6.959 4.331 4.461 4.037 3.462 6.239 1.935 2.505 2.924 4.247 4.772 7.914 9.55

2 8.861 5.566 5.087 4.173 4.586 3.261 4.91 3.214 7.693 6.794 11.843 6.393 3.132 3.245 3.116 3.16 4.576 1.899 2.221 2.87 3.262 4.904 9.766 11.99

3 11.349 5.949 4.713 3.244 2.148 2.728 3.256 1.83 5.128 3.63 5.842 4.17 1.365 2.01 2.311 1.751 0.679 1.373 0.87 1.199 1.136 1.561 4.094 2.415

4 15.859 9.474 8.635 6.944 8.346 4.644 4.24 3.544 7.602 7.68 12.59 5.895 3.263 3.548 3.252 2.768 4.372 1.473 1.688 2.204 3.35 3.406 5.534 7.942

5 9.283 5.143 4.416 3.516 0.184 1.783 3.857 2.105 4.047 1.106 0.249 1.864 0.027 0.866 0.555 0.819 0.294 1.104 1.062 1.061 1.314 2.959 3.879 2.027

;

table prob(s,t)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1 0.138 0.156 0.169 0.178 0.234 0.236 0.134 0.178 0.163 0.167 0.156 0.128 0.118 0.111 0.108 0.100 0.125 0.078 0.100 0.113 0.153 0.119 0.111 0.123

2 0.085 0.087 0.090 0.093 0.097 0.094 0.140 0.119 0.126 0.128 0.129 0.129 0.121 0.122 0.126 0.128 0.132 0.117 0.126 0.131 0.118 0.122 0.136 0.154

3 0.108 0.093 0.083 0.073 0.045 0.079 0.093 0.068 0.084 0.074 0.071 0.095 0.089 0.097 0.108 0.099 0.071 0.103 0.083 0.066 0.041 0.039 0.057 0.031

4 0.151 0.147 0.152 0.155 0.176 0.134 0.121 0.131 0.125 0.139 0.134 0.119 0.121 0.118 0.119 0.112 0.125 0.101 0.102 0.100 0.121 0.085 0.077 0.102

5 0.089 0.080 0.078 0.079 0.004 0.051 0.110 0.078 0.066 0.023 0.014 0.057 0.051 0.068 0.068 0.071 0.050 0.076 0.067 0.045 0.047 0.074 0.054 0.026

;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Parameters

C\_AGDG1 'AG DG marginal generation cost' /22.5/

C\_AGDG2 /20/

C\_AGDG3 /20/

C\_AGDG4 /20/

C\_AGDG5 /22.5/

P\_AGDG1max 'AG DGs maximum generation' /1.2/

P\_AGDG1min 'AG DGs minimum generation' /0/

P\_AGDG2max 'AG DGs maximum generation' /1.0/

P\_AGDG2min 'AG DGs minimum generation' /0/

P\_AGDG3max 'AG DGs maximum generation' /1.0/

P\_AGDG3min 'AG DGs minimum generation' /0/

P\_AGDG4max 'AG DGs maximum generation' /1.0/

P\_AGDG4min 'AG DGs minimum generation' /0/

P\_AGDG5max 'AG DGs maximum generation' /1.2/

P\_AGDG5min 'AG DGs minimum generation' /0/

E\_AGBS1ini /0.02/

E\_AGBS1min Min amount of energy stored in AGBS / 0.4/

E\_AGBS1max Max amount of energy stored in AGBS / 8/

P\_AGBS1chmin Min charge power of AGBS / 0/

P\_AGBS1chmax Max charge power of AGBS / 6.5/

P\_AGBS1dchmin Min discharge power of AGBS / 0/

P\_AGBS1dchmax Max discharge power of AGBS /2.5/

Neeta\_AGBS1ch / 0.98/

Neeta\_AGBS1dch / 0.98/

P\_AGDGini Initial Gen power of AGs DG

P\_exchLEMmax Max\_exchange power in DA LEM / 20/

Lambda\_LEMmax /30/

RD\_AGDG1 Ramp down rate of AGs DG /0.5/

RU\_AGDG1 Ramp up rate of AGs DG /0.5/

RD\_AGDG2 Ramp down rate of AGs DG /0.6/

RU\_AGDG2 Ramp up rate of AGs DG /0.6/

RD\_AGDG3 Ramp down rate of AGs DG /0.6/

RU\_AGDG3 Ramp up rate of AGs DG /0.6/

RD\_AGDG4 Ramp down rate of AGs DG /0.6/

RU\_AGDG4 Ramp up rate of AGs DG /0.6/

RD\_AGDG5 Ramp down rate of AGs DG /0.5/

RU\_AGDG5 Ramp up rate of AGs DG /0.5/

CI Confidence interval /0.95/

beta Risk aversion factor /0.4/

Lambda\_PV Cost of PV /36/

Lambda\_WT Cost of WT /29/;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variables

P\_exchWEMDA(t) Exchange power in DA WEM

P\_exchLEMDA(s,t) Exchange power in DA LEM

P\_AGDG1(s,t)

P\_AGDG2(s,t)

P\_AGDG3(s,t)

P\_AGDG4(s,t)

P\_AGDG5(s,t)

P\_AGBS1ch

P\_AGBS1dch

E\_AGBS1

P\_SLEM(t) Surplus power

Rev(s,t)

Cos(s,t)

OF

Lambda\_LEMDA(t)

E\_Prof(s,t)

CVaR

VaR

OF1

neeta;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Equations

const1(s,t)

const2(s,t)

const3(s,t)

const4(s,t)

const5(s,t)

const6(s,t)

const7(s,t)

const8(s,t)

const9(s,t)

const10(s,t)

const11(s,t)

const12(s,t)

const13(s,t)

const14(s,t)

const15(s,t)

const16(s,t)

const17(s,t)

const18(s,t)

const19(s,t)

const20(s,t)

const21(s,t)

const22(s,t)

const23(s,t)

const24(s,t)

const25(s,t)

const26(s,t)

const27(s,t)

const28(s,t)

const29(s,t)

const30(s,t)

const31(t)

const32(t)

const33(s,t)

const34(s,t)

const35(s,t)

const36(s,t)

const37(s,t)

;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const1(s,t).. PV(s,t)+WT(s,t)+P\_AGDG1(s,t)+P\_AGDG2(s,t)+P\_AGDG3(s,t)+P\_AGDG4(s,t)+P\_AGDG5(s,t)+(P\_AGBS1ch(s,t)-P\_AGBS1dch(s,t)) =e= P\_exchLEMDA(s,t);

const2(s,t).. P\_exchLEMDA(s,t) =g= 0;

const3(s,t).. P\_exchLEMDA(s,t) =l= P\_exchLEMmax;

const4(s,t).. P\_AGDG1(s,t) =g= P\_AGDG1min;

const5(s,t).. P\_AGDG1(s,t) =l= P\_AGDG1max;

const6(s,t).. P\_AGDG2(s,t) =g= P\_AGDG2min;

const7(s,t).. P\_AGDG2(s,t) =l= P\_AGDG2max;

const8(s,t).. P\_AGDG3(s,t) =g= P\_AGDG3min;

const9(s,t).. P\_AGDG3(s,t) =l= P\_AGDG3max;

const10(s,t).. P\_AGDG4(s,t) =g= P\_AGDG4min;

const11(s,t).. P\_AGDG4(s,t) =l= P\_AGDG4max;

const12(s,t).. P\_AGDG5(s,t) =g= P\_AGDG5min;

const13(s,t).. P\_AGDG5(s,t) =l= P\_AGDG5max;

const14(s,t).. P\_AGDG1(s,t)-P\_AGDG1(s,t-1) =l= RU\_AGDG1;

const15(s,t).. P\_AGDG1(s,t-1)-P\_AGDG1(s,t) =l= RD\_AGDG1;

const16(s,t).. P\_AGDG2(s,t)-P\_AGDG2(s,t-1) =l= RU\_AGDG2;

const17(s,t).. P\_AGDG2(s,t-1)-P\_AGDG2(s,t) =l= RD\_AGDG2;

const18(s,t).. P\_AGDG3(s,t)-P\_AGDG3(s,t-1) =l= RU\_AGDG3;

const19(s,t).. P\_AGDG3(s,t-1)-P\_AGDG3(s,t) =l= RD\_AGDG3;

const20(s,t).. P\_AGDG4(s,t)-P\_AGDG4(s,t-1) =l= RU\_AGDG4;

const21(s,t).. P\_AGDG4(s,t-1)-P\_AGDG4(s,t) =l= RD\_AGDG4;

const22(s,t).. P\_AGDG5(s,t)-P\_AGDG5(s,t-1) =l= RU\_AGDG5;

const23(s,t).. P\_AGDG5(s,t-1)-P\_AGDG5(s,t) =l= RD\_AGDG5;

const24(s,t).. P\_AGBS1dch(s,t) =g= P\_AGBS1dchmin;

const25(s,t).. P\_AGBS1dch(s,t) =l= P\_AGBS1dchmax;

const26(s,t).. P\_AGBS1ch(s,t) =g= P\_AGBS1chmin;

const27(s,t).. P\_AGBS1ch(s,t) =l= P\_AGBS1chmax;

const28(s,t).. E\_AGBS1(s,t) =e= E\_AGBS1(s,t-1)+(P\_AGBS1ch(s,t)\*Neeta\_AGBS1ch)-(P\_AGBS1dch(s,t)/Neeta\_AGBS1dch);

const29(s,t).. E\_AGBS1(s,t) =g= E\_AGBS1min;

const30(s,t).. E\_AGBS1(s,t) =l= E\_AGBS1max;

const31(t).. Lambda\_LEMDA(t) =g= 0;

const32(t).. Lambda\_LEMDA(t) =l= Lambda\_LEMmax;

const33(s,t).. E\_Prof(s,t) =e= ((P\_exchLEMDA(s,t)\*Lambda\_LEMDA(t))-((P\_AGDG1(s,t)\*C\_AGDG1)+(P\_AGDG2(s,t)\*C\_AGDG2)+(P\_AGDG3(s,t))\*C\_AGDG3)+(P\_AGDG4(s,t)\*C\_AGDG4)+(P\_AGDG5(s,t)\*C\_AGDG5));

const34(s,t).. CVaR =e= (VaR) -(1\*(prob(s,t)\*neeta)/(1-0.95));

const35(s,t).. VaR-E\_Prof(s,t)-neeta =l= 0;

const36(s,t).. neeta=g= 0;

const37(s,t).. OF1 =e= (1-beta)\*prob(s,t)\*E\_Prof(s,t)+(beta\*CVaR);

\*const19(t).. P\_SLEM(t) =e= P\_LEMload(t,'P')-P\_exchLEMDA(t);

model NewVaR/ const1, const2,const3,const4,const5,const6,const7,const8,const9,const10,const11,const12,const13,const14,const15,const16, const17, const18,const19,const20,const21,const22,const23,const24,const25,const26, const27,const28, const29,

const30,const31,const32,const33,const34,const35,const36,const37/;

option nlp = conopt;

solve NewVaR maximizing OF1 using nlp;

Display P\_exchLEMDA.l, P\_AGDG1.l, P\_AGDG2.l, P\_AGDG3.l,P\_AGDG4.l,P\_AGDG5.l,neeta.l,Lambda\_LEMDA.l,E\_Prof.l, OF1.l,CVaR.l,VaR.l;

%%%% Code for calculating CVaR and VaR of aggregators%%%

#!/usr/bin/env python

# coding: utf-8

# In[40]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import norm

# In[41]:

df=pd.read\_csv('C:/Users/pavan/OneDrive/Desktop/Var with uncertnity/AG1-Prof\_New.csv')

# In[42]:

df.head()

# In[43]:

df.tail()

# In[44]:

df.describe()

# In[45]:

for column in df.columns:

plt.plot(df.index, df[column], label=column)

# Add legend

plt.legend(labels=df.columns)

# In[46]:

ax = df['Scen1'].pct\_change().plot(kind='hist')

ax.set\_title("Scen1 Percentage Change Histogram")

# In[47]:

ax = df['Scen2'].pct\_change().plot(kind='hist')

ax.set\_title("Scen2 Percentage Change Histogram")

# In[48]:

ax = df['Scen3'].pct\_change().plot(kind='hist')

ax.set\_title("Scen3 Percentage Change Histogram")

# In[49]:

ax = df['Scen4'].pct\_change().plot(kind='hist')

ax.set\_title("Scen4 Percentage Change Histogram")

# In[50]:

# Obtain percentage change per stock

returns = df.pct\_change().dropna()

# Calculate the portfolio returns as the weighted average of the individual asset returns

weights = np.full((5), 0.25) # assuming equal weight

port\_returns = (weights \* returns).sum(axis=1) # weighted sum

# In[51]:

returns

# In[52]:

# Filter out non-finite values from port\_returns

port\_returns\_filtered = port\_returns[np.isfinite(port\_returns)]

# Calculate the range for the histogram based on the minimum and maximum values of port\_returns

hist\_range = (np.min(port\_returns\_filtered), np.max(port\_returns\_filtered))

# In[53]:

plt.hist(port\_returns\_filtered, bins=10, range=hist\_range, density=True, alpha=0.5)

plt.xlabel("Portfolio Returns")

plt.ylabel("Density")

plt.title(f"Portfolio Percentage Returns")

plt.show()

# In[54]:

#Historical method

# Assume initial portfolio value

initial\_portfolio = 100000

# In[55]:

# Obtain percentage change per stock

returns = df.pct\_change()

# In[56]:

# Calculate the portfolio returns as the weighted average of the individual asset returns

weights = np.full((5), 0.1) # assuming equal weight

individual\_port\_returns = weights \* returns # Element-wise multiplication

port\_returns = (weights \* returns).sum(axis=1) # weighted sum

# In[57]:

# Calculate the portfolio's VaR at 95% confidence level

confidence\_level = 0.95

# Calculate P(Return <= VAR) = alpha

var = port\_returns.quantile(q=1-confidence\_level)

# Calculate CVAR by computing the average returns below the VAR level

cvar = port\_returns[port\_returns <= var].mean()

# In[58]:

# Multiply the VaR and CVaR by the initial investment value to get the absolute value

var\_value = var \* initial\_portfolio

cvar\_value = cvar \* initial\_portfolio

# In[59]:

print(f"Historical VaR at {confidence\_level} confidence level: {var\_value:.2f} ({var:.2%})")

print(f"Historical CVaR at {confidence\_level} confidence level: {cvar\_value:.2f} ({cvar:.2%})")

# In[60]:

plt.hist(port\_returns\_filtered, bins=10, range=hist\_range, density=True, alpha=0.5)

# Add VAR CVAR to the histogram

plt.axvline(x=var, color='red', linestyle='--', label=f"VaR: {var:.2%}")

plt.axvline(x=cvar, color='blue', linestyle='--', label=f"CVaR: {cvar:.2%}")

plt.legend()

plt.xlabel("Portfolio Returns")

plt.ylabel("Density")

plt.title(f"Portfolio Percentage Returns for 95% confidence interval")

plt.show()

# In[61]:

# Calculate actual losses

initial\_portfolio\_value = 1000000 # Example initial portfolio value

actual\_losses = initial\_portfolio\_value \* (1 + returns) - initial\_portfolio\_value

# In[62]:

# Compare with VaR estimates

var\_violations = actual\_losses < -var

# In[63]:

# Old behavior (deprecated)

mean\_value = df.mean()

# In[64]:

# Updated behavior (to avoid FutureWarning)

mean\_value = df.mean() # Compute mean along the columns

import warnings

# Suppress FutureWarning

warnings.simplefilter(action='ignore', category=FutureWarning)

# In[65]:

# Calculate violation frequency

violation\_frequency = np.mean(var\_violations)

# In[66]:

selected\_columns = ['Scen1', 'Scen2', 'Scen3', 'Scen4', 'Scen5']

series = df[selected\_columns]

# In[67]:

# Display the selected columns

print(series)

# In[68]:

violation\_frequency\_scalar = violation\_frequency.iloc[0]

# In[69]:

# Print violation frequency with percentage formatting

print(f"Violation Frequency: {violation\_frequency\_scalar:.2%}")

# In[70]:

# Calculate the portfolio's VaR at 90% confidence level

confidence\_level = 0.90

# Calculate P(Return <= VAR) = alpha

var = port\_returns.quantile(q=1-confidence\_level)

# Calculate CVAR by computing the average returns below the VAR level

cvar = port\_returns[port\_returns <= var].mean()

# In[71]:

# Multiply the VaR and CVaR by the initial investment value to get the absolute value

var\_value = var \* initial\_portfolio

cvar\_value = cvar \* initial\_portfolio

# In[72]:

print(f"Historical VaR at {confidence\_level} confidence level: {var\_value:.2f} ({var:.2%})")

print(f"Historical CVaR at {confidence\_level} confidence level: {cvar\_value:.2f} ({cvar:.2%})")

# In[73]:

plt.hist(port\_returns\_filtered, bins=10, range=hist\_range, density=True, alpha=1.0)

# Add VAR CVAR to the histogram

plt.axvline(x=var, color='red', linestyle='--', label=f"VaR: {var:.2%}")

plt.axvline(x=cvar, color='blue', linestyle='--', label=f"CVaR: {cvar:.2%}")

plt.legend()

plt.xlabel("Portfolio Returns")

plt.ylabel("Density")

plt.title(f"Portfolio Percentage Returns for 90%")

plt.show()

# In[74]:

# Calculate the portfolio's VaR at 98% confidence level

confidence\_level = 0.98

# Calculate P(Return <= VAR) = alpha

var = port\_returns.quantile(q=1-confidence\_level)

# Calculate CVAR by computing the average returns below the VAR level

cvar = port\_returns[port\_returns <= var].mean()

# In[75]:

# Multiply the VaR and CVaR by the initial investment value to get the absolute value

var\_value = var \* initial\_portfolio

cvar\_value = cvar \* initial\_portfolio

# In[76]:

print(f"Historical VaR at {confidence\_level} confidence level: {var\_value:.2f} ({var:.2%})")

print(f"Historical CVaR at {confidence\_level} confidence level: {cvar\_value:.2f} ({cvar:.2%})")

# In[77]:

plt.hist(port\_returns\_filtered, bins=10, range=hist\_range, density=True, alpha=0.2)

# Add VAR CVAR to the histogram

plt.axvline(x=var, color='red', linestyle='--', label=f"VaR: {var:.2%}")

plt.axvline(x=cvar, color='blue', linestyle='--', label=f"CVaR: {cvar:.2%}")

plt.legend()

plt.xlabel("Portfolio Returns")

plt.ylabel("Density")

plt.title(f"Portfolio Percentage Returns for 98%")

plt.show()

# In[78]:

import warnings

# Filter out the specific warning

warnings.filterwarnings("ignore", message="invalid value encountered in subtract")

# In[79]:

confidence\_level = 0.95

# In[80]:

# Initialize a dictionary to store VaR values for each column

var\_values = {}

# In[81]:

var\_values = {}

# Loop through each column in the DataFrame

for column in df.columns:

sorted\_data = np.sort(df[column])

var\_index = int((1 - confidence\_level) \* len(sorted\_data))

var\_values[column] = sorted\_data[var\_index]

# In[82]:

# Print VaR values for each column

for column, var in var\_values.items():

print(f"VaR for {column} at {confidence\_level\*100}% confidence level: {var}")

# In[83]:

sorted\_data = np.sort(df)

# In[84]:

num\_samples = len(sorted\_data)

# In[85]:

cvar\_index = int((1 - confidence\_level) \* num\_samples)

# In[86]:

cvar = np.mean(sorted\_data[:cvar\_index])

# Print CVaR result

print(f"CVaR at {confidence\_level\*100}% confidence level: {cvar}")

# In[87]:

# Calculate CVaR for each column

cvar\_values = {}

for column in df.columns:

sorted\_data = np.sort(df[column])

cvar\_index = int(confidence\_level \* len(sorted\_data))

cvar\_values[column] = np.mean(sorted\_data[:cvar\_index])

# In[88]:

# Print CVaR values at confidence\_level for each column

for column, cvar in cvar\_values.items():

print(f"CVaR for {column}: {cvar}")

# In[89]:

confidence\_level\_1 = 0.98

# In[90]:

# Initialize a dictionary to store VaR values for each column

var\_values1 = {}

# In[109]:

# Print VaR values for each column

for column, var in var\_values1.items():

print(f"VaR for {column} at {confidence\_level\_1\*100}% confidence level: {var}")

# In[102]:

var\_values1 = {}

# Loop through each column in the DataFrame

for column in df.columns:

sorted\_data = np.sort(df[column])

var\_index = int((1 - confidence\_level\_1) \* len(sorted\_data))

var\_values1[column] = sorted\_data[var\_index]

# In[103]:

sorted\_data = np.sort(df)

# In[104]:

num\_samples = len(sorted\_data)

# In[105]:

cvar\_index = int((1 - confidence\_level\_1) \* num\_samples)

# In[106]:

cvar = np.mean(sorted\_data[:cvar\_index])

# Print CVaR result

print(f"CVaR at {confidence\_level\_1\*100}% confidence level: {cvar}")

# In[107]:

# Print CVaR values for each column

for column, cvar in cvar\_values.items():

print(f"CVaR for {column}: {cvar}")

# In[108]:

#Calculate mean column-wise

mean\_columns = np.mean(df, axis=0)

# In[99]:

print("Mean of each column:")

print(mean\_columns)

# In[ ]: