

```
!pip3 install pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: Cython<0.29.18,>=0.29 in /usr/local/lib/python3.7/dist
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in /usr/local/lib/python3.7
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pa
```

```
# Import package
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from statsmodels.tsa.api import SimpleExpSmoothing
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from datetime import datetime
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
warnings.simplefilter('ignore')
```

```
data = pd.read_csv('Dataset5_yearssn.csv', names = ['yearssn'])
```

Applying KPSS and ADF test

1. ADF test

```
#define function for ADF test
from statsmodels.tsa.stattools import adfuller

def adf_test(atr):
    #Perform Dickey-Fuller test:
    timeseries = data[atr].dropna()
    print ('Results of Dickey-Fuller Test for ',atr,'\n')
    dfctest = adfuller(timeseries, autolag='AIC')
    dfcoutput = pd.Series(dfctest[0:4], index=['Test Statistic','p-value','#Lags Used','Numb
    for key,value in dfctest[4].items():
        dfcoutput['Critical Value (%)'%key] = value
```

```
dfoutput['Critical Value (%s) %key'] = value
print(dfoutput)
```

```
#apply adf test on the series
adf_test('yearssn')
```

Results of Dickey-Fuller Test for yearssn

```
Test Statistic      -2.845522
p-value             0.052060
#Lags Used          8.000000
Number of Observations Used  305.000000
Critical Value (1%)  -3.451974
Critical Value (5%)  -2.871063
Critical Value (10%) -2.571844
dtype: float64
```

2. KPSS test

```
#define function for kpss test
from statsmodels.tsa.stattools import kpss
#define KPSS
def kpss_test(atr):
    timeseries = data[atr].dropna()
    print ('Results of KPSS Test for ',atr)
    kpsstest = kpss(timeseries, regression='c')
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-value','Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s) %key'] = value
    print (kpss_output)
kpss_test('yearssn')
```

```
Results of KPSS Test for yearssn
Test Statistic      0.414735
p-value             0.070804
Lags Used          16.000000
Critical Value (10%)  0.347000
Critical Value (5%)  0.463000
Critical Value (2.5%) 0.574000
Critical Value (1%)  0.739000
dtype: float64
```

For ADF test, we can see that the p-value is more than 0.05. Thus, from ADF test, we can say that the dataset is non-stationary.

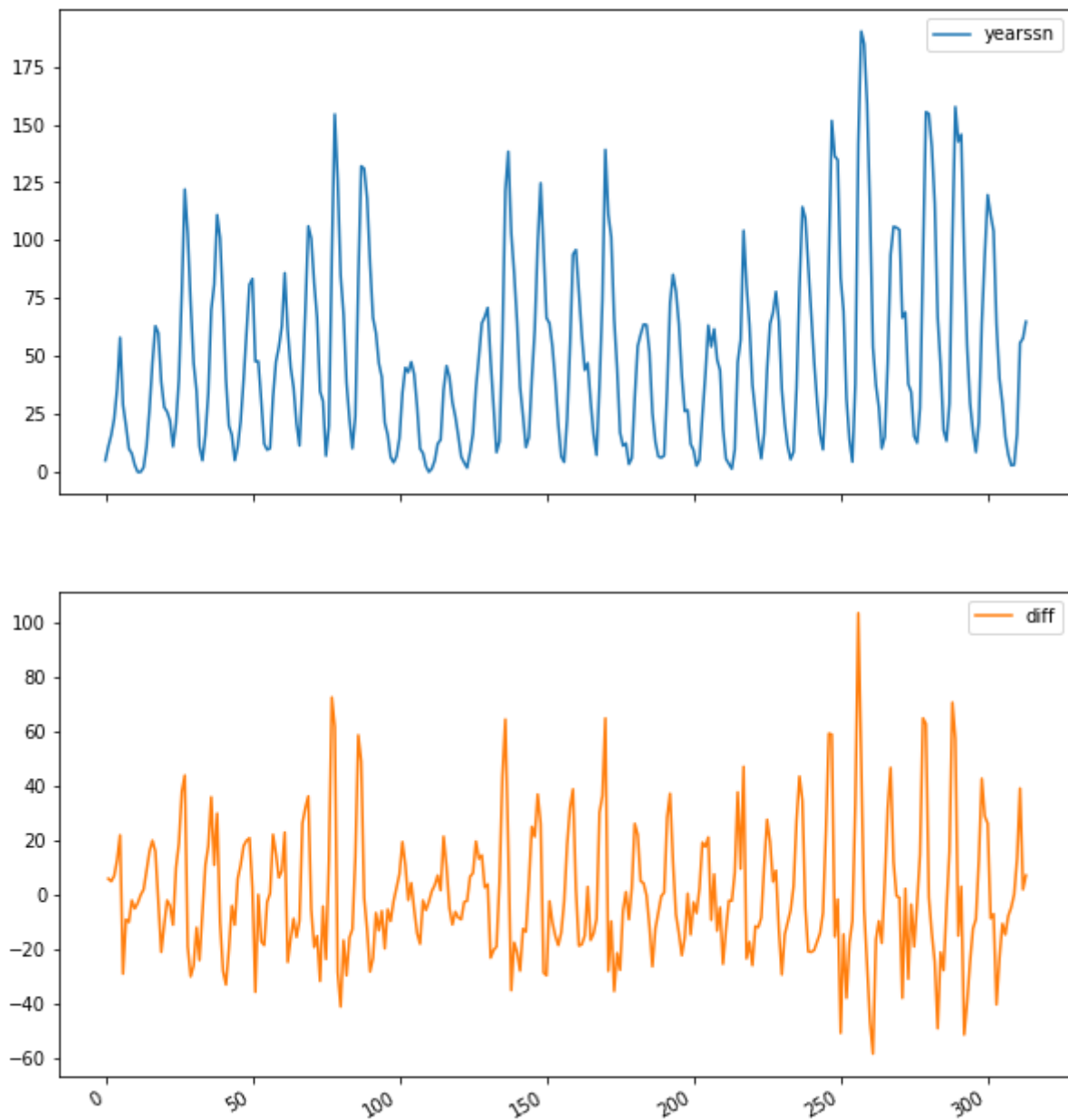
For KPSS test, Test Statistic is less than Critical Value, thus we fail to reject the null hypothesis. Thus, from KPSS test, we can say that the dataset is stationary.

Since, both tests conclude that the series is stationary, therefore, the dataset is concluded as Trend-Stationary.

Making dataset stationary with differencing

```
# Differencing
data['diff'] = data['yearssn'].diff(periods=1)

data.plot(subplots=True, figsize=(10,12))
plt.show()
```



Applying Exponential Smoothing

```
#List of least mse and mae
mseries=[]
msedes=[]
msetes=[]
maeses=[]
```

```
maedes=[]
maetes=[]
```

Single Exponential Smoothing

```
#Defining Single Exponential Smoothing function ses
def ses(arr,alpha):
    arr1 = [arr[0]]
    for i in range(1, len(arr)):
        arr1.append(alpha * arr[i-1] + (1 - alpha) * arr1[i-1])
    return arr1
```

```
#Defining Mean of Squared Error Function mse
def mse(arr1,arr2):
    arr3=[]
    for i, j in zip(arr1, arr2):
        arr3.append(i-j)
    Sum=0
    for i in arr3:
        sqr=i**2
        Sum+=sqr
    mse=Sum/(len(arr2)-1)
    return mse
```

```
#Function to make list of demand with interval 'n'
def dem_n(arr,n):
    arr1=[arr[0]]
    for i in range(1,len(arr)):
        if i%n==0:
            arr1.append(arr[i])
    return arr1
```

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,1)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
forecast1=ses(demand,alpha1)
forecast2=ses(demand,alpha2)
forecast3=ses(demand,alpha3)
```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2 is: ",mse1)
```

```
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)

Mean of Square Errors for alpha = 0.2 is: 1423.7656094103215
Mean of Square Errors for alpha = 0.5 is: 1082.6052976072397
Mean of Square Errors for alpha = 0.8 is: 723.5052144698412
```

```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)
```

```
print("Mean Absolute Errors for alpha = 0.2 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8 is: ",mae3)
```

```
Mean Absolute Errors for alpha = 0.2 is: 31.04575551336873
Mean Absolute Errors for alpha = 0.5 is: 26.475409455479966
Mean Absolute Errors for alpha = 0.8 is: 20.66825223080585
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    df1.plot(style=['-','-'])
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-','-'])
else:
    print('alpha: ',alpha3)
    df3.plot(style=['-','-'])
```

```
alpha= 0.2
```

Double Exponential Smoothing

```
''' | forecast |
```

```
#Defining Double Exponential Smoothing function des
def des(arr,alpha,beta):
    a=[arr[0]]
    l=len(arr)
    b=[(arr[l-1]-arr[0])/(l-1)]
    arr1 = [arr[0]]
    arr1.append(a[0]+b[0])
    for i in range(1,len(arr)-1):
        a.append(alpha * arr[i] + (1 - alpha) * (a[i-1]+b[i-1]))
        b.append(beta * (a[i]-a[i-1]) + (1 - beta) * (b[i-1]))
        arr1.append(a[i]+b[i])
    return arr1

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,1)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

beta1=0.3
beta2=0.6
beta3=0.9

forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)

#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)

    Mean of Square Errors for alpha = 0.2,beta= 0.3 is:  2012.3303657557422
    Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  1871.1571453106742
    Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  651.9662859611615

#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

print("Mean Absolute Errors for alpha = 0.2,beta= 0.3 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6 is: ",mae2)
```

```
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9 is: ",mae3)

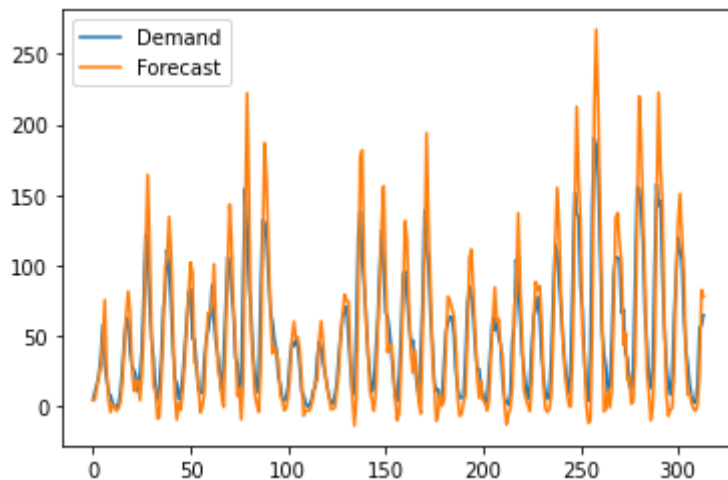
Mean Absolute Errors for alpha = 0.2,beta= 0.3 is: 37.51233810681161
Mean Absolute Errors for alpha = 0.5,beta= 0.6 is: 34.0868149094099
Mean Absolute Errors for alpha = 0.8,beta= 0.9 is: 18.598707057177787
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
```

```
alpha: 0.8
beta: 0.9
```



Triple Exponential Smoothing

```
#Defining initial trend
def initial_trend(arr, slen):
    Sum = 0
    for i in range(slen):
        Sum += float(arr[i+slen] - arr[i]) / slen
    return Sum / slen
```

```

#Defining initial seasonal
def initial_seasonal(arr, slen):
    arr1 = {}
    s_avg = []
    m = int(len(arr)/slen)
    for j in range(m):
        s_avg.append(sum(arr[slen*j:slen*j+slen])/float(slen))
    for i in range(slen):
        Sum = 0
        for j in range(m):
            Sum += arr[slen*j+i]-s_avg[j]
        arr1[i] = Sum/m
    return arr1

#Defining Triple Exponential Smoothing function tes with interval 'n'
def tes(arr, slen, alpha, beta, gamma, n):
    arr1 = []
    seasonals = initial_seasonal(arr, slen)
    for i in range(len(arr)+n):
        if i == 0:
            smooth = arr[0]
            trend = initial_trend(arr, slen)
            arr1.append(arr[0])
            continue
        if i >= len(arr):
            m = i - len(arr) + 1
            arr1.append((smooth + m*trend) + seasonals[i%slen])
        else:
            val = arr[i]
            lsmooth, smooth = smooth, alpha*(val-seasonals[i%slen]) + (1-alpha)*(smooth+tr
            trend = beta * (smooth-lsmooth) + (1-beta)*trend
            seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%slen]
            arr1.append(smooth+trend+seasonals[i%slen])
    return arr1

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,1)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

beta1=0.3
beta2=0.6
beta3=0.9

gamma1=0.4
gamma2=0.7
gamma3=0.95

#Considering season of 1 hours here

```



```

forecast1=tes(demand,1,alpha1,beta1,gamma1,0)
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
forecast3=tes(demand,1,alpha3,beta3,gamma3,0)

#Calculating mean of squared errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: ",mse3)

    Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: 269.3844974223863
    Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: 145.15574297524344
    Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: 418.50971245836934

#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)

    Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: 13.175520156294937
    Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: 10.040754623873244
    Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: 15.89153968520488

#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

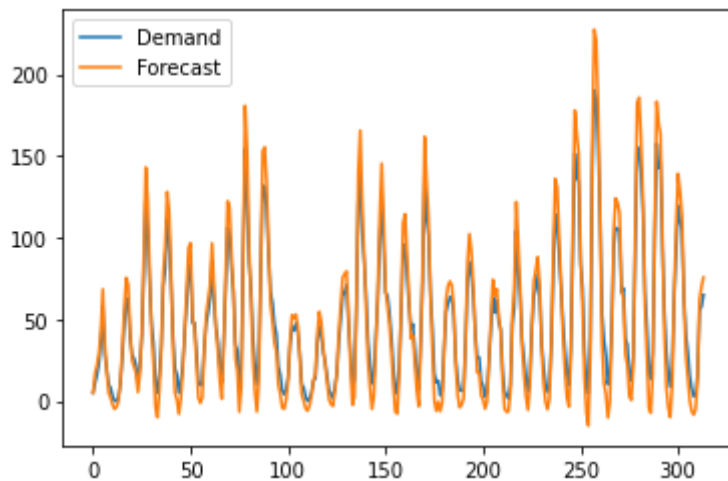
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)

if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    print('gamma: ',gamma1)
    df1.plot(style=['-','-'])
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    print('gamma: ',gamma2)
    df2.plot(style=['-','-'])
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    print('gamma: ',gamma3)

```

```
df3.plot(style=['-', '-'])
```

```
alpha: 0.5  
beta: 0.6  
gamma: 0.7
```



For 1 Unit

Single Exponential Smoothing

```
#Creating demand list in 'n' intervals  
demand=dem_n(data.yearssn,1)
```

```
#Forecasting  
alpha1=0.2  
alpha2=0.5  
alpha3=0.8
```

```
forecast1=ses(demand,alpha1)  
forecast2=ses(demand,alpha2)  
forecast3=ses(demand,alpha3)
```

```
#Calculating Mean of Square Errors  
mse1=mean_squared_error(demand,forecast1)  
mse2=mean_squared_error(demand,forecast2)  
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2 is: ",mse1)  
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)  
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)
```

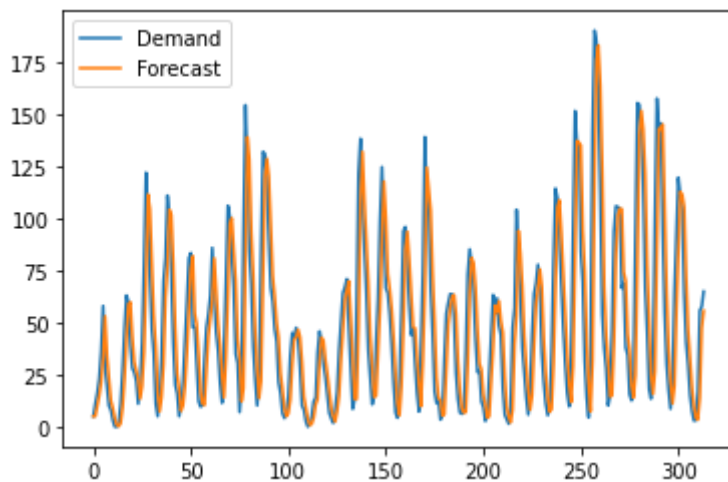
```
Mean of Square Errors for alpha = 0.2 is: 1423.7656094103215  
Mean of Square Errors for alpha = 0.5 is: 1082.6052976072397  
Mean of Square Errors for alpha = 0.8 is: 723.5052144698412
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    df1.plot(style=['-','-'])
    mseses.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-','-'])
    mseses.append(mse2)
else:
    print('alpha: ',alpha3)
    df3.plot(style=['-','-'])
    mseses.append(mse3)
```

alpha: 0.8



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maeses.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maeses.append(mae2)
else:
    maeses.append(mae3)
```

Double Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,1)
```

```
#Forecasting
alpha1=0.2
```

```
alpha2=0.5
alpha3=0.8
```

```
beta1=0.3
beta2=0.6
beta3=0.9
```

```
forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)
```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)
```

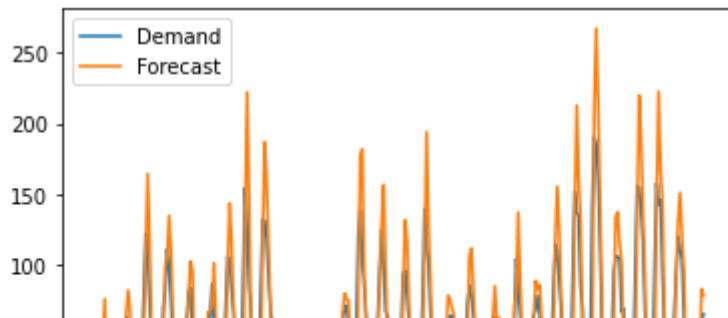
```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is: 2012.3303657557422
Mean of Square Errors for alpha = 0.5,beta= 0.6 is: 1871.1571453106742
Mean of Square Errors for alpha = 0.8,beta= 0.9 is: 651.9662859611615
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
    msedes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
    msedes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
    msedes.append(mse3)
```

```
alpha: 0.8
beta: 0.9
```



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maedes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maedes.append(mae2)
else:
    maedes.append(mae3)
```

Triple Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,1)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
beta1=0.3
beta2=0.6
beta3=0.9
```

```
gamma1=0.4
gamma2=0.7
gamma3=0.95
```

```
#Considering season of 1 hours here
```

```
forecast1=tes(demand,1,alpha1,beta1,gamma1,0)
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
forecast3=tes(demand,1,alpha3,beta3,gamma3,0)
```

```
#Calculating mean of squared errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: ",mse3)
```

```
Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: 269.3844974223863
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: 145.15574297524344
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: 418.50971245836934
```

```
#Calculating Mean Absolute Errors
```

```
mae1=mean_absolute_error(demand,forecast1)
```

```
mae2=mean_absolute_error(demand,forecast2)
```

```
mae3=mean_absolute_error(demand,forecast3)
```

```
print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
```

```
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
```

```
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)
```

```
Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: 13.175520156294937
```

```
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: 10.040754623873244
```

```
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: 15.89153968520488
```

```
#Comparing mse and plotting for least mse
```

```
d1={'Demand':demand,'Forecast':forecast1}
```

```
d2={'Demand':demand,'Forecast':forecast2}
```

```
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
```

```
df2=pd.DataFrame(d2)
```

```
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
```

```
    print('alpha: ',alpha1)
```

```
    print('beta: ',beta1)
```

```
    print('gamma: ',gamma1)
```

```
    df1.plot(style=['-','-'])
```

```
    msetes.append(mse1)
```

```
elif mse2<=mse1 and mse2<=mse3:
```

```
    print('alpha: ',alpha2)
```

```
    print('beta: ',beta2)
```

```
    print('gamma: ',gamma2)
```

```
    df2.plot(style=['-','-'])
```

```
    msetes.append(mse2)
```

```
else:
```

```
    print('alpha: ',alpha3)
```

```
    print('beta: ',beta3)
```

```
    print('gamma: ',gamma3)
```

```
    df3.plot(style=['-','-'])
```

```
    msetes.append(mse3)
```

```
alpha: 0.5
beta: 0.6
gamma: 0.7
```



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maetes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maetes.append(mae2)
else:
    maetes.append(mae3)
```

1	2	3	4	5	6	7
---	---	---	---	---	---	---

For 2 Unit

Single Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,2)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
forecast1=ses(demand,alpha1)
forecast2=ses(demand,alpha2)
forecast3=ses(demand,alpha3)
```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)
```

```
Mean of Square Errors for alpha = 0.2 is: 1592.9627355629634
Mean of Square Errors for alpha = 0.5 is: 1766.7394603329508
Mean of Square Errors for alpha = 0.8 is: 1777.9034247112054
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
```

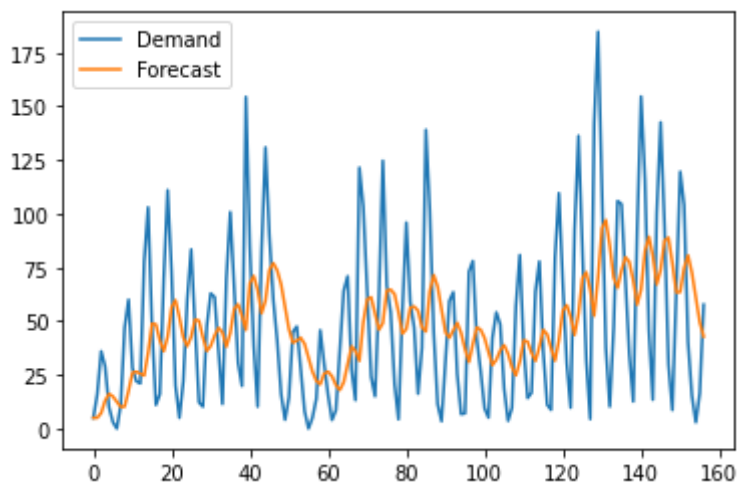
```

df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)

if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    df1.plot(style=['-','-'])
    mseses.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-','-'])
    mseses.append(mse2)
else:
    print('alpha: ',alpha3)
    df3.plot(style=['-','-'])
    mseses.append(mse3)

```

alpha: 0.2



```

#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maeses.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maeses.append(mae2)
else:
    maeses.append(mae3)

```

Double Exponential Smoothing

```

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,2)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

beta1=0.3
beta2=0.6
beta3=0.9

```



```
forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)
```

```
#Calculating Mean of Square Errors
```

```
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)
```

```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is: 1795.1971208439209
Mean of Square Errors for alpha = 0.5,beta= 0.6 is: 2991.0663598929123
Mean of Square Errors for alpha = 0.8,beta= 0.9 is: 3526.6510703318286
```

```
#Comparing mse and plotting for least mse
```

```
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
    msedes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
    msedes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
    msedes.append(mse3)
```

Quarter	Demand	Forecast
Q1 2017	150	150
Q2 2017	150	150
Q3 2017	150	150
Q4 2017	185	175

01

beta2=0.6

```
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
```

```
print("Mean of Square Errors for alpha = 0.5,beta= 0.
```

Mean of Square Errors for alpha = 0.5, beta= 0.6

1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832. 833. 834. 835. 836. 837. 838. 839. 840. 84

```

mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)

    Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  17.469677978838963
    Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  9.16547484168987
    Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  25.386925465863833

```

```

#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

```

```

df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)

```

```

if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    print('gamma: ',gamma1)
    df1.plot(style=['-','-'])
    msetes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    print('gamma: ',gamma2)
    df2.plot(style=['-','-'])
    msetes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    print('gamma: ',gamma3)
    df3.plot(style=['-','-'])
    msetes.append(mse3)

```

heta. 0.6

```
if mae1<=mae2 and mae1<=mae3:
```

```

    elif mae2<=mae1 and mae2<=mae3:

```

else:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

[illegible]

```
demand=dem_n(data.years$sn,4)
```

$$\alpha_1 = 0.2$$
$$\alpha_3 = 0.8$$

```
forecast2=sas(demand,alpha2)
```

#Calculating Mean of Square Errors

```
mse2=mean_squared_error(demand,forecast2)
```

```
print("Mean of Square Errors for alpha = 0.2 is: ",mse1)
```

```
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)
```

```
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)
```

Mean of Square Errors for alpha = 0.5 is: 2134.9658263484116

```
d1={'Demand':demand,'Forecast':forecast1}
```

```
d2={'Demand':demand,'Forecast':forecast2}
```

```
d3={'Demand':demand,'Forecast':forecast3}
```

```
df2=pd.DataFrame(d2)
```

```
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
```

```
print('alpha: ',alpha1)
```

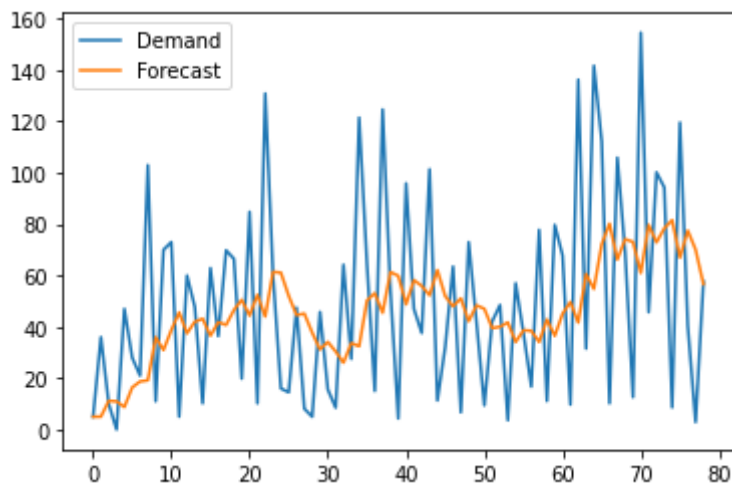
$$df1 = n1 + (ctv1 - 1) + (ctv2 - 1)$$

```

df1.plot(style=['-', '-'])
mseses.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-', '-'])
    mseses.append(mse2)
else:
    print('alpha: ',alpha3)
    df3.plot(style=['-', '-'])
    mseses.append(mse3)

```

alpha: 0.2



```

#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maeses.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maeses.append(mae2)
else:
    maeses.append(mae3)

```

Double Exponential Smoothing

```

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,4)

```

```

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

```

```

beta1=0.3
beta2=0.6
beta3=0.9

```

```

forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)

```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)

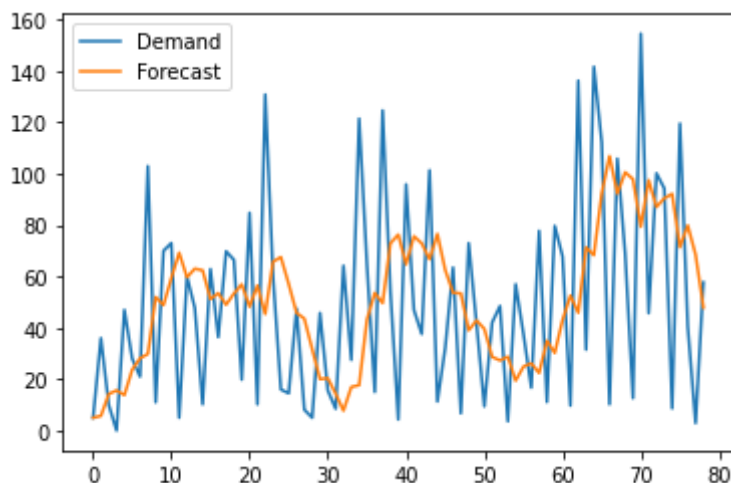
    Mean of Square Errors for alpha = 0.2,beta= 0.3 is: 1861.7609770709907
    Mean of Square Errors for alpha = 0.5,beta= 0.6 is: 2772.722713554847
    Mean of Square Errors for alpha = 0.8,beta= 0.9 is: 6334.244971342228
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
    msedes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
    msedes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
    msedes.append(mse3)
```

```
alpha: 0.2
beta: 0.3
```



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maedes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maedes.append(mae2)
else:
    maedes.append(mae3)
```

Triple Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,4)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
beta1=0.3
beta2=0.6
beta3=0.9
```

```
gamma1=0.4
gamma2=0.7
gamma3=0.95
```

```
#Considering season of 1 hours here
```

```
forecast1=tes(demand,1,alpha1,beta1,gamma1,0)
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
forecast3=tes(demand,1,alpha3,beta3,gamma3,0)
```

```
#Calculating mean of squared errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: ",mse3)
```

```
Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: 661.2049102216006
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: 68.23978479971298
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: 1328.0494076540465
```

```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)
```

```
print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
```

```
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)

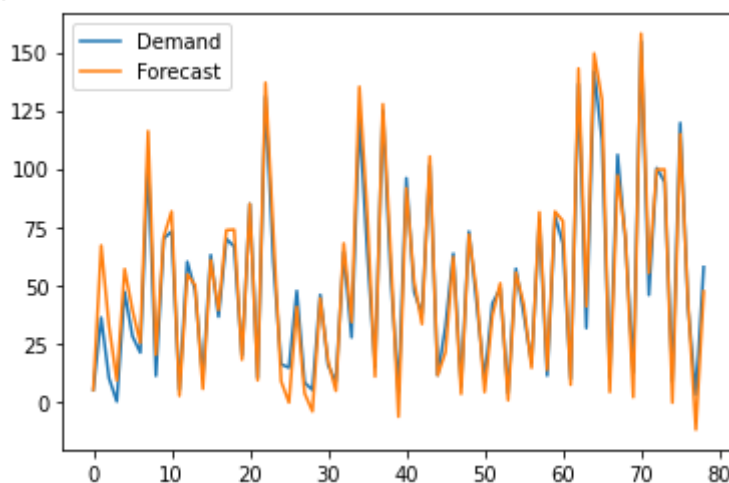
Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: 21.224505551778854
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: 6.345566663687334
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: 30.327630901647133
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    print('gamma: ',gamma1)
    df1.plot(style=['-','-'])
    msetes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    print('gamma: ',gamma2)
    df2.plot(style=['-','-'])
    msetes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    print('gamma: ',gamma3)
    df3.plot(style=['-','-'])
    msetes.append(mse3)
```

```
alpha: 0.5
beta: 0.6
gamma: 0.7
```



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maetes.append(mae1)
```



```

elif mae2<=mae1 and mae2<=mae3:
    maetes.append(mae2)
else:
    maetes.append(mae3)

```

For 8 Unit

Single Exponential Smoothing

```

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,8)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

forecast1=ses(demand,alpha1)
forecast2=ses(demand,alpha2)
forecast3=ses(demand,alpha3)

#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)

    Mean of Square Errors for alpha = 0.2 is: 1934.1154156697291
    Mean of Square Errors for alpha = 0.5 is: 2210.233614294581
    Mean of Square Errors for alpha = 0.8 is: 2567.196785297861

#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)

if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    df1.plot(style=['-','-'])
    mseses.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-','-'])
    mseses.append(mse2)

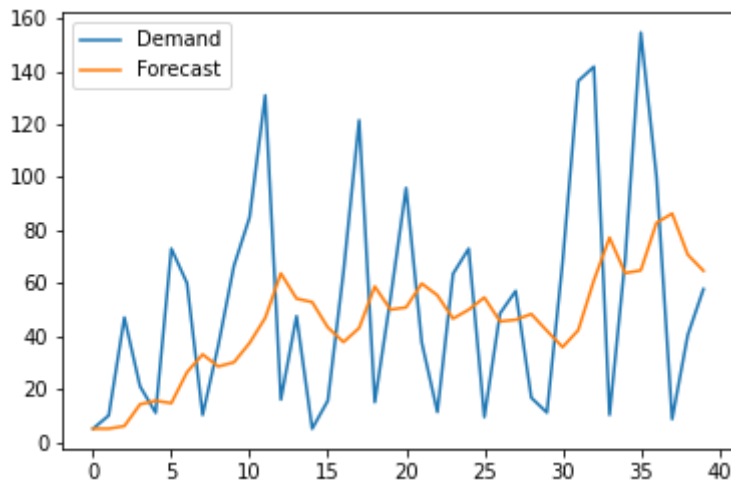
```

```

else:
    print('alpha: ',alpha3)
    df3.plot(style=['-','-'])
    mseses.append(mse3)

```

alpha: 0.2



```

#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maeses.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maeses.append(mae2)
else:
    maeses.append(mae3)

```

Double Exponential Smoothing

```

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,8)

```

```

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

```

```

beta1=0.3
beta2=0.6
beta3=0.9

```

```

forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)

```

```

#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

```

```

print("Mean of Square Errors for alpha = 0.2 beta= 0.3 is: " mse1)

```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)
```

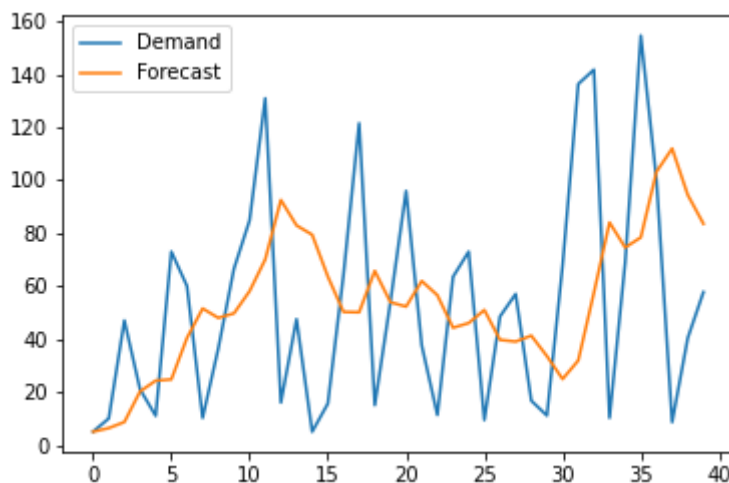
```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is: 2200.2528825736854
Mean of Square Errors for alpha = 0.5,beta= 0.6 is: 3247.7380391296747
Mean of Square Errors for alpha = 0.8,beta= 0.9 is: 5501.475735494226
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
    msedes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
    msedes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
    msedes.append(mse3)
```

```
alpha: 0.2
beta: 0.3
```



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maedes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maedes.append(mae2)
else:
```

```
maedes.append(mae3)
```

Triple Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,8)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

beta1=0.3
beta2=0.6
beta3=0.9

gamma1=0.4
gamma2=0.7
gamma3=0.95

#Considering season of 1 hours here

forecast1=tes(demand,1,alpha1,beta1,gamma1,0)
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
forecast3=tes(demand,1,alpha3,beta3,gamma3,0)

#Calculating mean of squared errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: ",mse3)

    Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is:  508.8331089708787
    Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  113.56662360267781
    Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  1250.4587420536172

#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)

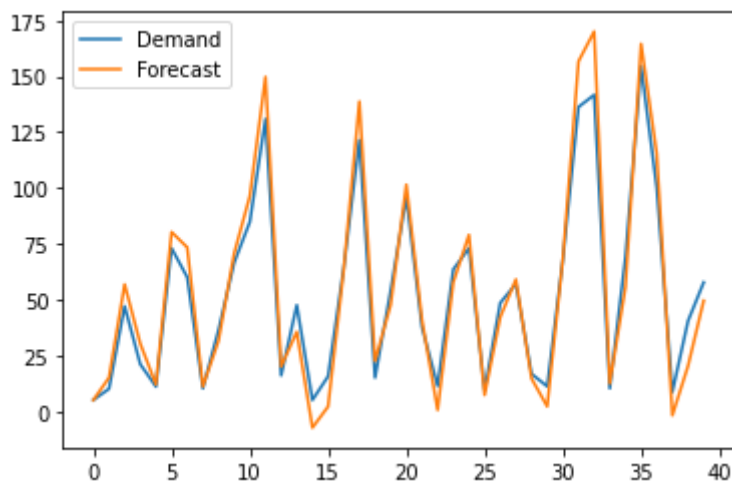
    Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  17.555963491060538
    Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  8.534471665110216
    Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  27.862047208668308
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    print('gamma: ',gamma1)
    df1.plot(style=['-','-'])
    msetes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    print('gamma: ',gamma2)
    df2.plot(style=['-','-'])
    msetes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    print('gamma: ',gamma3)
    df3.plot(style=['-','-'])
    msetes.append(mse3)
```

```
alpha: 0.5
beta: 0.6
gamma: 0.7
```



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maetes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maetes.append(mae2)
else:
    maetes.append(mae3)
```

Single Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,12)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

forecast1=ses(demand,alpha1)
forecast2=ses(demand,alpha2)
forecast3=ses(demand,alpha3)

#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)

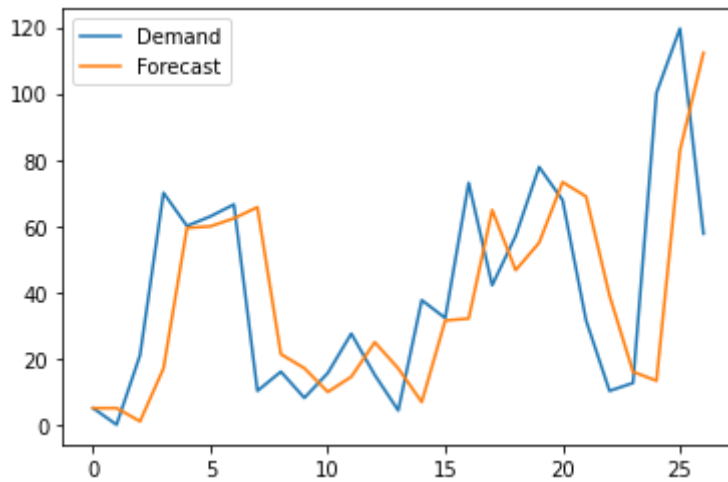
    Mean of Square Errors for alpha = 0.2 is:  1022.7884998909686
    Mean of Square Errors for alpha = 0.5 is:  944.7411881574418
    Mean of Square Errors for alpha = 0.8 is:  919.5444938370305

#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)

if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    df1.plot(style=['-','-'])
    mseses.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-','-'])
    mseses.append(mse2)
else:
    print('alpha: ',alpha3)
    df3.plot(style=['-','-'])
    mseses.append(mse3)
```

alpha: 0.8



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maeses.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maeses.append(mae2)
else:
    maeses.append(mae3)
```

Double Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,12)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
beta1=0.3
beta2=0.6
beta3=0.9
```

```
forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)
```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)
```

Mean of Square Errors for alpha = 0.2,beta= 0.3 is: 1230.7916334585705

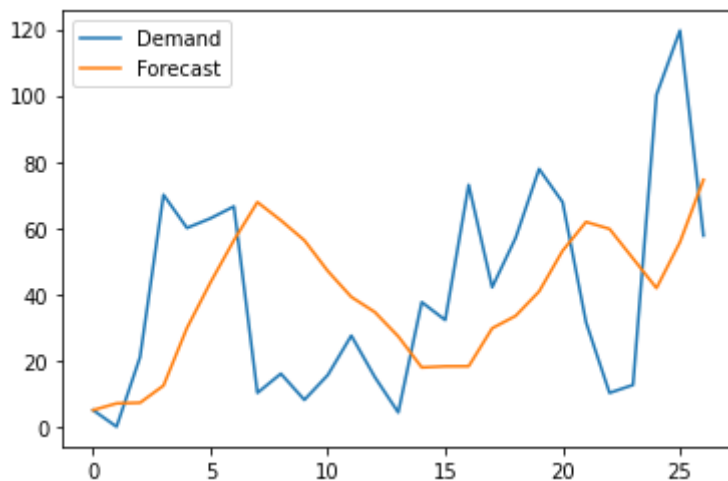
Mean of Square Errors for alpha = 0.5,beta= 0.6 is: 1420.1624307361178
Mean of Square Errors for alpha = 0.8,beta= 0.9 is: 1617.3042101904846

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
    msedes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
    msedes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
    msedes.append(mse3)
```

alpha: 0.2
beta: 0.3



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maedes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maedes.append(mae2)
else:
    maedes.append(mae3)
```

Triple Exponential Smoothing


```

#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,12)

#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

beta1=0.3
beta2=0.6
beta3=0.9

gamma1=0.4
gamma2=0.7
gamma3=0.95

#Considering season of 1 hours here

forecast1=tes(demand,1,alpha1,beta1,gamma1,0)
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
forecast3=tes(demand,1,alpha3,beta3,gamma3,0)

#Calculating mean of squared errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: ",mse3)

    Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is:  226.1821383063702
    Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  75.03413321281637
    Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  470.36206065866025

#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)

    Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  11.280765700782277
    Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  7.028383578100507
    Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  15.738301712859059

#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

```

```
d3={'Demand':demand, 'Forecast':forecast}
```

```
df1=pd.DataFrame(d1)
```

```
df2=pd.DataFrame(d2)
```

```
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
```

```
    print('alpha: ',alpha1)
```

```
    print('beta: ',beta1)
```

```
    print('gamma: ',gamma1)
```

```
    df1.plot(style=['-','-'])
```

```
    msetes.append(mse1)
```

```
elif mse2<=mse1 and mse2<=mse3:
```

```
    print('alpha: ',alpha2)
```

```
    print('beta: ',beta2)
```

```
    print('gamma: ',gamma2)
```

```
    df2.plot(style=['-','-'])
```

```
    msetes.append(mse2)
```

```
else:
```

```
    print('alpha: ',alpha3)
```

```
    print('beta: ',beta3)
```

```
    print('gamma: ',gamma3)
```

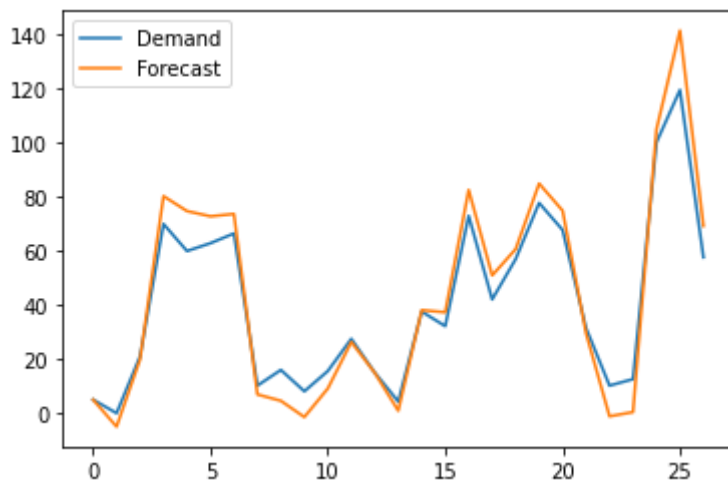
```
    df3.plot(style=['-','-'])
```

```
    msetes.append(mse3)
```

```
alpha: 0.5
```

```
beta: 0.6
```

```
gamma: 0.7
```



```
#Storing least mae values
```

```
if mae1<=mae2 and mae1<=mae3:
```

```
    maetes.append(mae1)
```

```
elif mae2<=mae1 and mae2<=mae3:
```

```
    maetes.append(mae2)
```

```
else:
```

```
    maetes.append(mae3)
```

For 24 Interval

Single Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,24)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
forecast1=ses(demand,alpha1)
forecast2=ses(demand,alpha2)
forecast3=ses(demand,alpha3)
```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8 is: ",mse3)
```

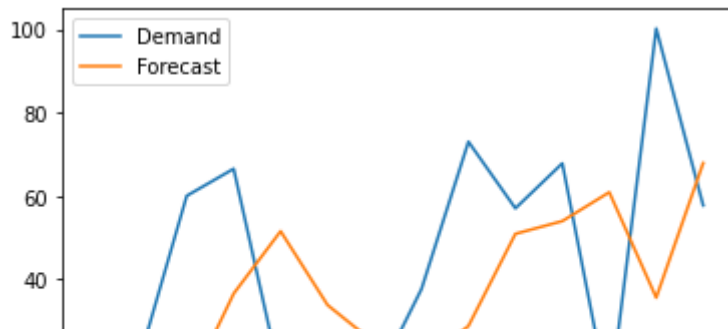
```
Mean of Square Errors for alpha = 0.2 is: 1039.7499684745346
Mean of Square Errors for alpha = 0.5 is: 1029.0628839029584
Mean of Square Errors for alpha = 0.8 is: 1206.0812286782327
```

```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    df1.plot(style=['-','-'])
    mseses.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    df2.plot(style=['-','-'])
    mseses.append(mse2)
else:
    print('alpha: ',alpha3)
    df3.plot(style=['-','-'])
    mseses.append(mse3)
```

alpha: 0.5



```
#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maeses.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maeses.append(mae2)
else:
    maeses.append(mae3)
```

Double Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,24)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
beta1=0.3
beta2=0.6
beta3=0.9
```

```
forecast1=des(demand,alpha1,beta1)
forecast2=des(demand,alpha2,beta2)
forecast3=des(demand,alpha3,beta3)
```

```
#Calculating Mean of Square Errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)
```

```
print("Mean of Square Errors for alpha = 0.2,beta= 0.3 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 is: ",mse3)
```

```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is: 900.4246777371616
Mean of Square Errors for alpha = 0.5,beta= 0.6 is: 1493.115190502541
Mean of Square Errors for alpha = 0.8,beta= 0.9 is: 2359.272425689149
```

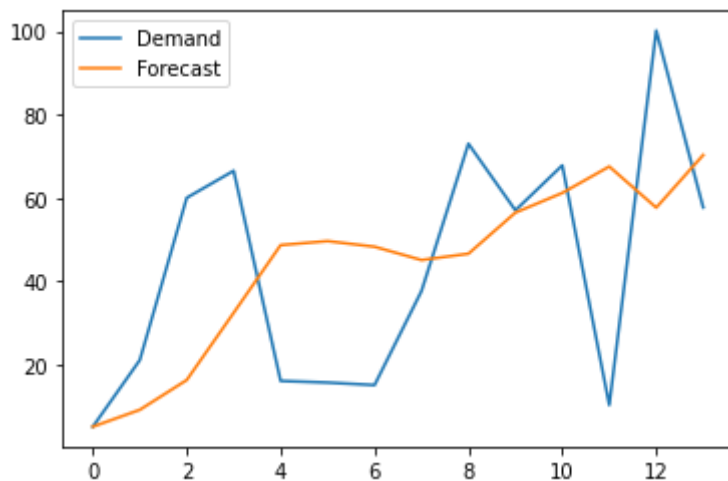
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
```

```
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)
```

```
if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    df1.plot(style=['-','-'])
    msedes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)
    df2.plot(style=['-','-'])
    msedes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    df3.plot(style=['-','-'])
    msedes.append(mse3)
```

```
alpha: 0.2
beta: 0.3
```



Triple Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yearssn,24)
```

```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8
```

```
beta1=0.3
beta2=0.6
beta3=0.9
```

```

gamma1=0.4
gamma2=0.7
gamma3=0.95

#Considering season of 1 hours here

forecast1=tes(demand,1,alpha1,beta1,gamma1,0)
forecast2=tes(demand,1,alpha2,beta2,gamma2,0)
forecast3=tes(demand,1,alpha3,beta3,gamma3,0)

#Calculating mean of squared errors
mse1=mean_squared_error(demand,forecast1)
mse2=mean_squared_error(demand,forecast2)
mse3=mean_squared_error(demand,forecast3)

print("Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: ",mse1)
print("Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: ",mse2)
print("Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: ",mse3)

    Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is: 493.35639407287624
    Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is: 108.60519673656229
    Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is: 586.7919040583805

#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

print("Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: ",mae1)
print("Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: ",mae2)
print("Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: ",mae3)

    Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is: 18.219079669274702
    Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is: 8.096574869244506
    Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is: 19.03964964271159

#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

df1=pd.DataFrame(d1)
df2=pd.DataFrame(d2)
df3=pd.DataFrame(d3)

if mse1<=mse2 and mse1<=mse3:
    print('alpha: ',alpha1)
    print('beta: ',beta1)
    print('gamma: ',gamma1)
    df1.plot(style=['-','-'])
    msetes.append(mse1)
elif mse2<=mse1 and mse2<=mse3:
    print('alpha: ',alpha2)
    print('beta: ',beta2)

```

```

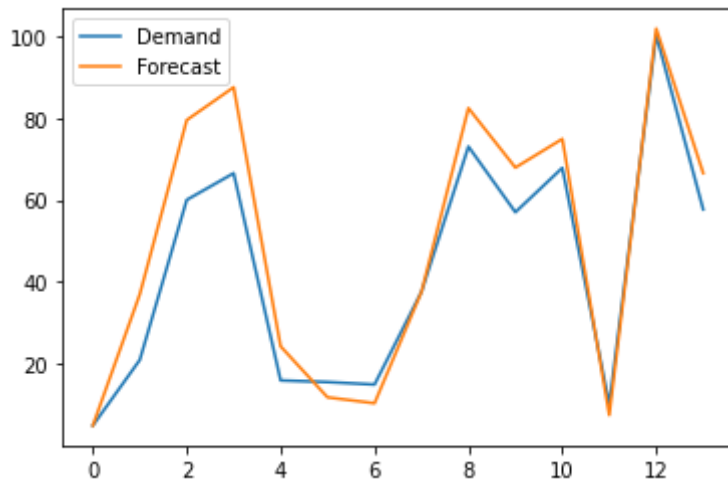
print('gamma: ',gamma2)
df2.plot(style=['-','-'])
msetes.append(mse2)
else:
    print('alpha: ',alpha3)
    print('beta: ',beta3)
    print('gamma: ',gamma3)
    df3.plot(style=['-','-'])
    msetes.append(mse3)

```

```

alpha: 0.5
beta: 0.6
gamma: 0.7

```



```

#Storing least mae values
if mae1<=mae2 and mae1<=mae3:
    maetes.append(mae1)
elif mae2<=mae1 and mae2<=mae3:
    maetes.append(mae2)
else:
    maetes.append(mae3)

```

Least MSE and MAE values are

```

print("Least MSE ses")
print(mseses)
print("Least MSE des")
print(msedes)
print("Least MSE tes")
print(msetes)

```

```

print("Least MAE ses")
print(maeses)
print("Least MAE des")
print(maedes)
print("Least MAE tes")
print(maetes)

```

```

Least MSE ses
[723.5052144698412, 1592.9627355629634, 1693.2449686568807, 1934.1154156697291, 919.5]
Least MSE des
[651.9662859611615, 1795.1971208439209, 1861.7609770709907, 2200.2528825736854, 1230]
Least MSE tes
[145.15574297524344, 122.02948159950819, 68.23978479971298, 113.56662360267781, 75.03]
Least MAE ses
[10.040754623873244, 10.040754623873244, 9.16547484168987, 6.345566663687334, 8.53447]
Least MAE des
[10.040754623873244, 10.040754623873244, 9.16547484168987, 6.345566663687334, 8.53447]
Least MAE tes
[10.040754623873244, 9.16547484168987, 6.345566663687334, 8.534471665110216, 7.02838]

```

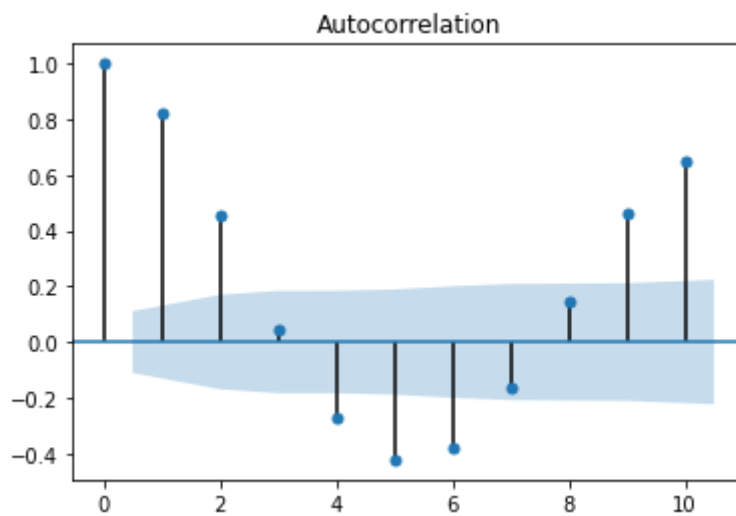
Applying ACF and PACF

```

#Plotting ACF
plot_acf(data.yearssn,lags=10)
plt.show

```

<function matplotlib.pyplot.show>



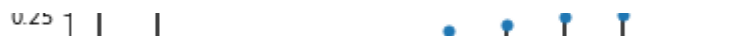
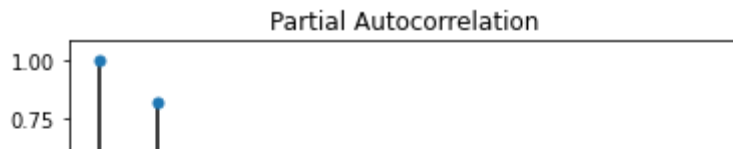
```

#plotting PACF
plot_pacf(data.yearssn,lags=10)
plt.show

```



```
<function matplotlib.pyplot.show>
```



Applying AR, MA, ARIMA Models



```
#AR
```

```
#fit model
```

```
model=ARIMA(data['yearssn'], order=(3,0,0))
```

```
model_fit=model.fit()
```

```
#model summary
```

```
print(model_fit.summary())
```

```
#make prediction
```

```
data['forecastAR'] = model_fit.predict()
```

```
data[['yearssn','forecastAR']].plot()
```

ARMA Model Results

```
=====
Dep. Variable:          yearssn    No. Observations:          314
Model:                  ARMA(3, 0)  Log Likelihood              -1325.144
Method:                 css-mle     S.D. of innovations         16.401
Date:                   Mon, 01 Mar 2021  AIC                2660.287
Time:                   05:35:08      BIC                2679.034
=====
```

```
mse=mean_squared_error(data.yearssn,data.forecastAR.dropna())
print("MSE for AR is:",mse)
```

MSE for AR is: 274.28824978825946

```
CONST          49.6160      2.750      18.045      0.000      44.227      55.005
```

```
mae=mean_absolute_error(data.yearssn,data.forecastAR.dropna())
print("MAE for AR is:",mae)
```

MAE for AR is: 12.526751800100925

```
Real          Imaginary          Modulus          Frequency
```

#MA

```
#fit model
model=ARIMA(data['diff'].dropna(), order=(0,0,2))
model_fit=model.fit()
```

```
#model summary
print(model_fit.summary())
```

```
#make prediction
data['forecastMA'] = model_fit.predict()
data[['diff','forecastMA']].plot()
```

ARMA Model Results

```
=====
Dep. Variable:          diff      No. Observations:          313
Model:                  ARMA(0, 2)  Log Likelihood          -1373.226
Method:                 css-mle     S.D. of innovations      19.445
Date:                   Mon, 01 Mar 2021  AIC          2754.451
Time:                   05:35:08      BIC          2769.436
Sample:                 0            HQIC          2760.439
=====
```

```
=====
              coef      std err          z      P>|z|      [0.025      0.975]
-----
-----
```

```
mse=mean_squared_error(data.yearssn[0:-1],data.forecastMA.dropna())
print("MSE for MA is:",mse)
```

MSE for MA is: 4074.783133969069

```
real      imaginary      modulus      frequency
```

```
mae=mean_absolute_error(data.yearssn[0:-1],data.forecastMA.dropna())
print("MAE for MA is:",mae)
```

MAE for MA is: 49.34682101981897

```
100 |--- diff | |
```

#ARIMA

```
#fit model
model=ARIMA(data['diff'].dropna(), order=(3,0,2))
model_fit=model.fit()
```

```
#model summary
print(model_fit.summary())
```

```
#make prediction
data['forecastARIMA'] = model_fit.predict()
data[['diff','forecastARIMA']].plot()
```

ARMA Model Results

```

=====
Dep. Variable:          diff      No. Observations:          313
Model:                  ARMA(3, 2)  Log Likelihood          -1301.544
Method:                  css-mle    S.D. of innovations      15.404
Date:                   Mon, 01 Mar 2021  AIC                    2617.089
Time:                   05:35:09    BIC                     2643.312
Sample:                 0          HQIC                     2627.568
=====

```

```

=====
              coef      std err          z      P>|z|      [0.025      0.975]
-----
const          0.0106      0.405        0.026      0.979      -0.784      0.805
ar.L1.diff      1.8040      0.084       21.420      0.000       1.639      1.969
ar.L2.diff     -1.2407      0.131       -9.477      0.000      -1.497     -0.984
ar.L3.diff       0.1796      0.077        2.326      0.020       0.028      0.331
ma.L1.diff     -1.6258      0.060     -27.268      0.000      -1.743     -1.509
ma.L2.diff       0.7447      0.060       12.514      0.000       0.628      0.861
=====

```

Roots

```

=====
              Real          Imaginary          Modulus          Frequency
-----
AR.1          0.8665          -0.5701j          1.0372          -0.0926
AR.2          0.8665          -0.5701j          1.0372          -0.0926
=====

```

```

mse=mean_squared_error(data.yearssn[0:-1],data.forecastARIMA.dropna())
print("MSE for MA is:",mse)

```

MSE for MA is: 4854.11167302186

```

mae=mean_absolute_error(data.yearssn[0:-1],data.forecastARIMA.dropna())
print("MAE for MA is:",mae)

```

MAE for MA is: 52.499355713783395

Applying Auto ARIMA

```

import pmdarima as pm
model = pm.auto_arima(data.iloc[:,0], start_p=1, start_q=1,test='adf',max_p=3, max_q=3,m=1)
print(model.summary())

```

Performing stepwise search to minimize aic

```

ARIMA(1,0,1)(0,0,0)[0]      : AIC=2779.644, Time=0.08 sec
ARIMA(0,0,0)(0,0,0)[0]      : AIC=3503.280, Time=0.01 sec
ARIMA(1,0,0)(0,0,0)[0]      : AIC=2878.675, Time=0.03 sec
ARIMA(0,0,1)(0,0,0)[0]      : AIC=3162.801, Time=0.09 sec
ARIMA(2,0,1)(0,0,0)[0]      : AIC=2743.327, Time=0.12 sec
ARIMA(2,0,0)(0,0,0)[0]      : AIC=2744.118, Time=0.05 sec
ARIMA(3,0,1)(0,0,0)[0]      : AIC=2748.118, Time=0.12 sec
ARIMA(2,0,2)(0,0,0)[0]      : AIC=2734.757, Time=0.14 sec
ARIMA(1,0,2)(0,0,0)[0]      : AIC=2738.506, Time=0.10 sec
ARIMA(3,0,2)(0,0,0)[0]      : AIC=2629.302, Time=0.33 sec
ARIMA(3,0,3)(0,0,0)[0]      : AIC=2738.753, Time=0.27 sec
ARIMA(2,0,3)(0,0,0)[0]      : AIC=2736.348, Time=0.27 sec

```

[illegible]

