```
!pip3 install pmdarima
```

```
    Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-packages (1
    Requirement already satisfied: joblib>=0.11 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: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages
    Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages
    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: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-package
    Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-pa
    Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fro
    Requirement already satisfied: pytz>=2017.2 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: patsy>=0.5 in /usr/local/lib/python3.7/dist-packages (
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
```

```
# 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('Dataset2_month_temp.csv',names = ['month_temp'])
```

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')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Numb
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
```

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

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

```
Results of Dickey-Fuller Test for  month_temp

Test Statistic                  -4.044912
p-value                          0.001193
#Lags Used                       3.000000
Number of Observations Used    219.000000
Critical Value (1%)             -3.460567
Critical Value (5%)             -2.874830
Critical Value (10%)            -2.573853
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('month_temp')
```

```
Results of KPSS Test for  month_temp
Test Statistic            0.664381
p-value                   0.016784
Lags Used                15.000000
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 below 0.05. Thus, from ADF test, we can say that the dataset is stationary.

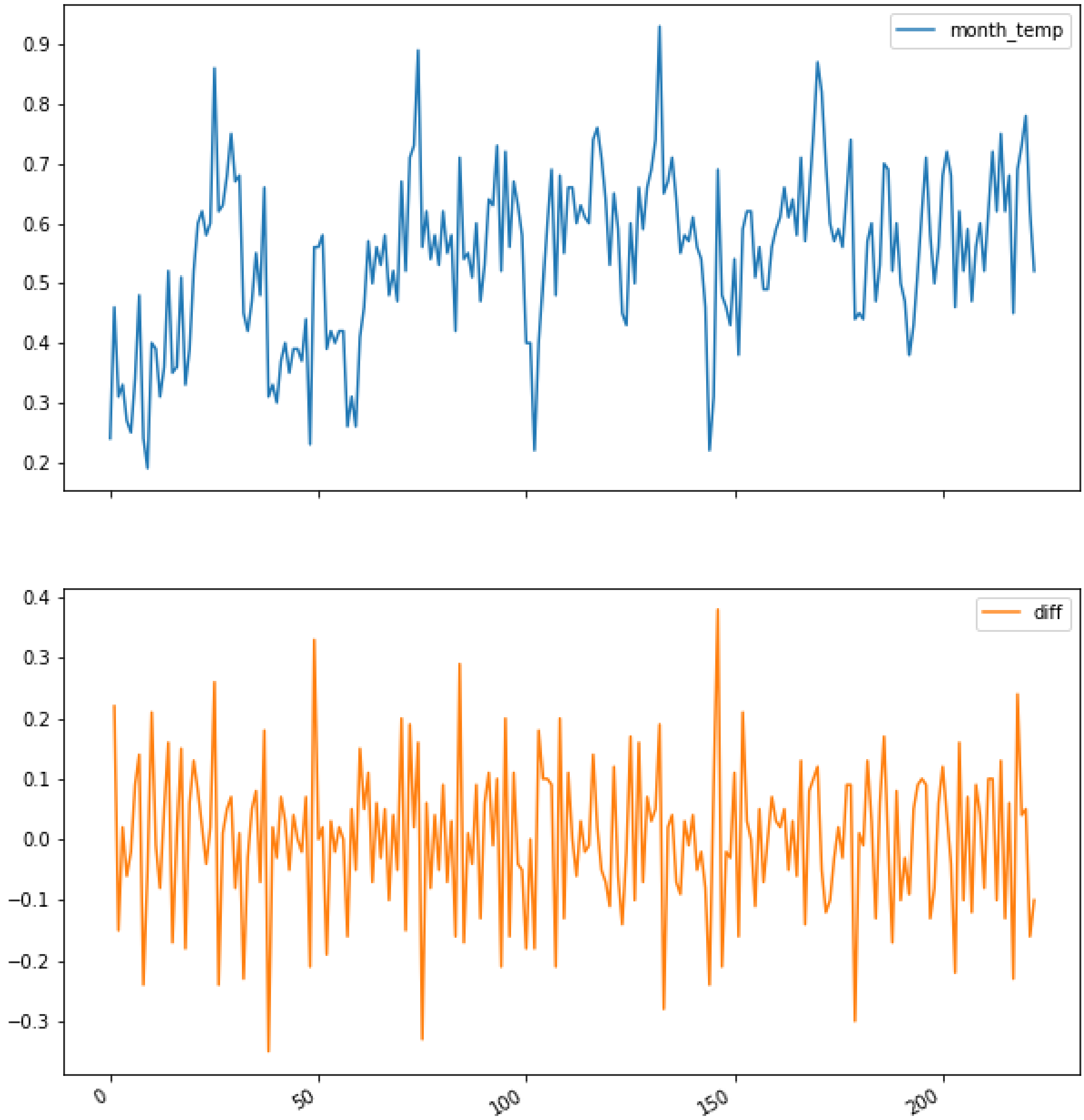
For KPSS test, Test Statistic is more than Critical Value, thus we reject the null hypothesis. Thus, from KPSS test, we can say that the dataset is non-stationary.

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

Making dataset stationary with differencing

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

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



Applying Exponential Smoothening

```
#List of least mse and mae
mseses=[]
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=[0]
  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.month_temp,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:  0.012776241365066766
     Mean of Square Errors for alpha = 0.5 is:  0.011783265150351989
     Mean of Square Errors for alpha = 0.8 is:  0.012865803390005504


#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:  0.08991859036980403
     Mean Absolute Errors for alpha = 0.5 is:  0.087340641452764
     Mean Absolute Errors for alpha = 0.8 is:  0.09010946336491085


#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: 0.5

Double Exponential Smoothing


```
#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.month_temp,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:  0.016344385097743625
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.016752197412308715
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.02435074881446847
```

```
#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)
```

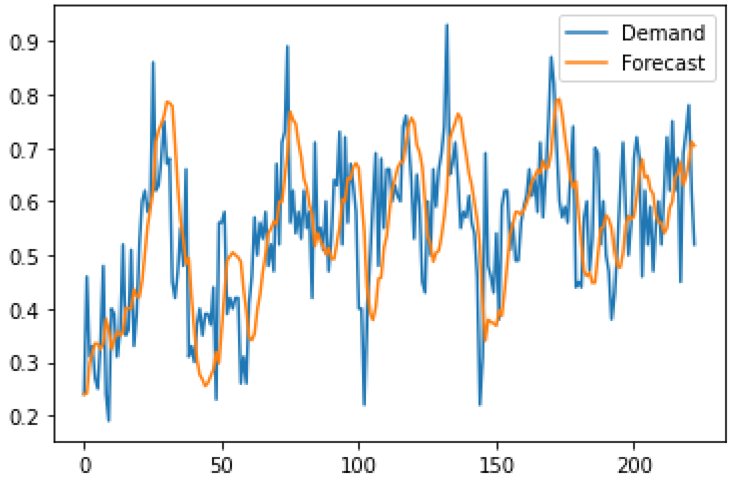
```
Mean Absolute Errors for alpha = 0.2,beta= 0.3 is:  0.10489883193358056
Mean Absolute Errors for alpha = 0.5,beta= 0.6 is:  0.1024506218200433
Mean Absolute Errors for alpha = 0.8,beta= 0.9 is:  0.121427472738696
```

```
#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:  0.2
beta:  0.3
```



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.month_temp,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 sqaured 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:  0.006142178171985968
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.0011599649951772768
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.006329173898406177
```

```
#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:  0.0526621870388914
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.02413624022372913
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.06325534048617418
```

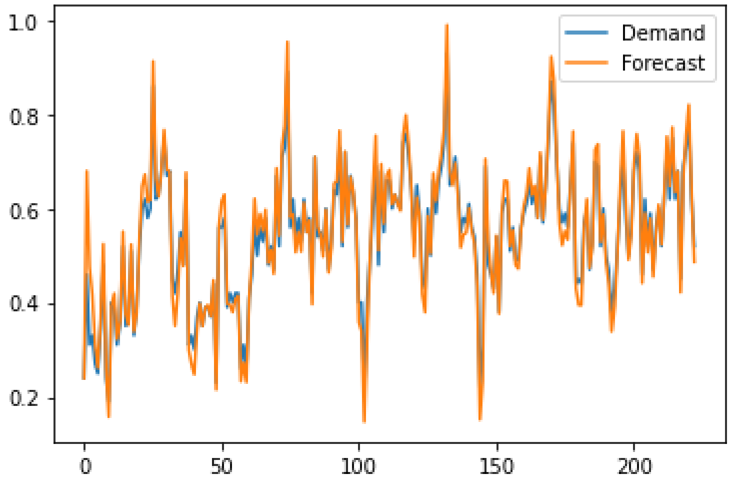
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

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.month_temp,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)
```

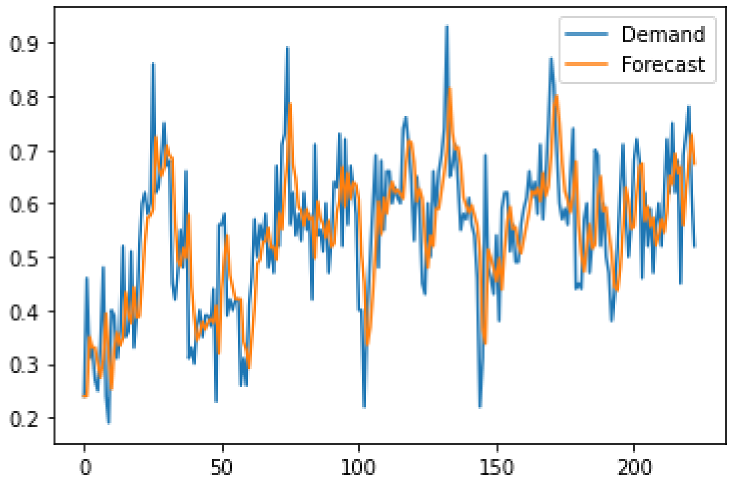
```
Mean of Square Errors for alpha = 0.2 is:  0.012776241365066766
Mean of Square Errors for alpha = 0.5 is:  0.011783265150351989
Mean of Square Errors for alpha = 0.8 is:  0.012865803390005504
```

```
#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.month_temp,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)
```

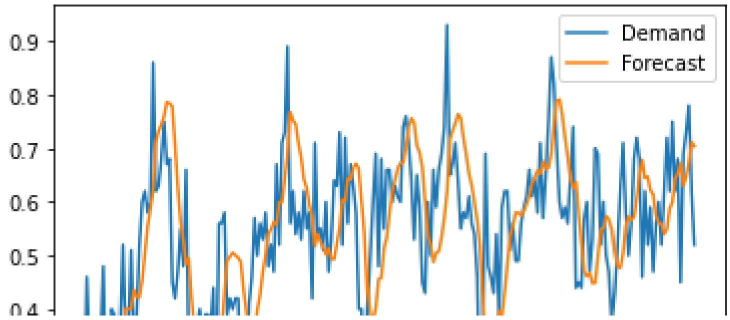
```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is:  0.016344385097743625
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.016752197412308715
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.02435074881446847
```

```
#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.month_temp,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 sqaured 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:  0.006142178171985968
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.0011599649951772768
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.006329173898406177
```

```
#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)
```

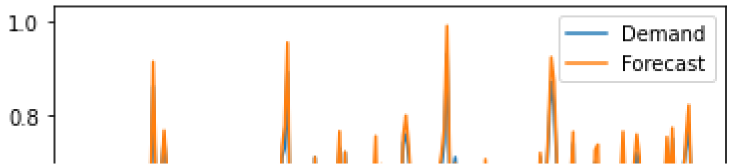
```
Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  0.0526621870388914
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.02413624022372913
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.06325534048617418
```

```
#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 2 Unit

Single Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.month_temp,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:  0.016080961425601354
Mean of Square Errors for alpha = 0.5 is:  0.015120728666467782
Mean of Square Errors for alpha = 0.8 is:  0.016803794493335546
```

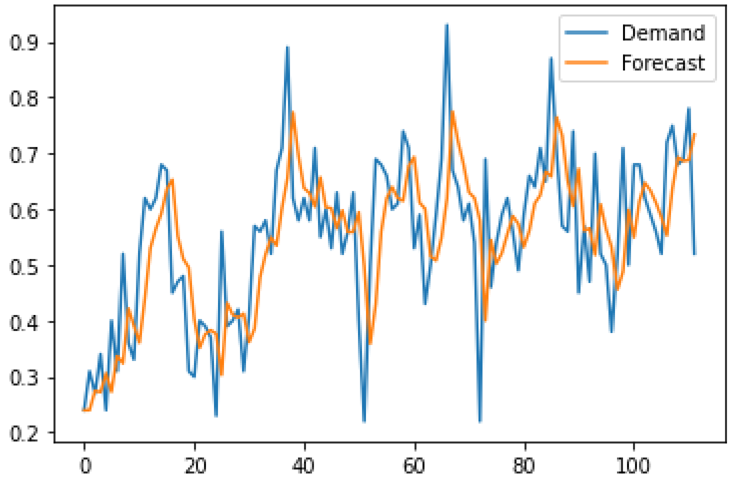
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

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.month_temp,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:  0.021743706689690372
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.021152519593798498
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.032303830991147524
```

```
#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:  0.5
beta:  0.6
```


```
#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.month_temp,2)


#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 sqaured 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:  0.004171315216015895
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.0010530392502876376
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.008109326159268706
```

```
#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:  0.05171209685741829
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.026731750173506715
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.07085378916000452
```

```
#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
```

```
#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 4 Unit

Single Exponential Smoothing

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


#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:  0.019573996145141943
    Mean of Square Errors for alpha = 0.5 is:  0.017969169959880345
    Mean of Square Errors for alpha = 0.8 is:  0.018292660563368656
```

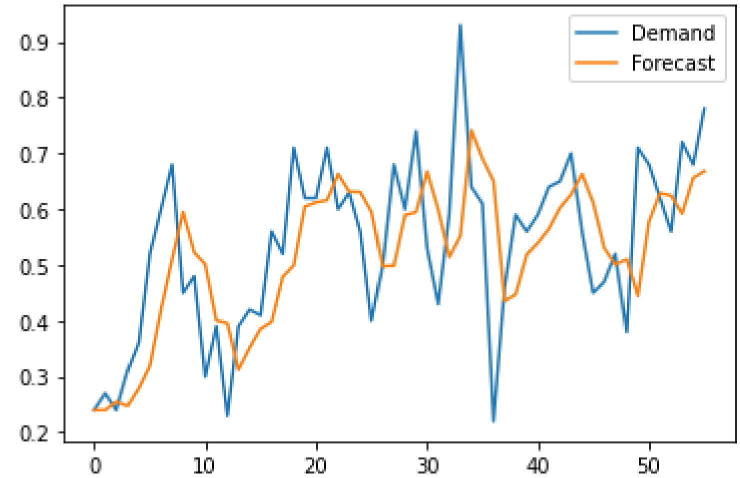
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

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

if mse1<=mse2 and mse1<=mse3:
  print('alpha: ',alpha1)
  df1.plot(style=['-','-'])
```

```
  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.month_temp,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)
```

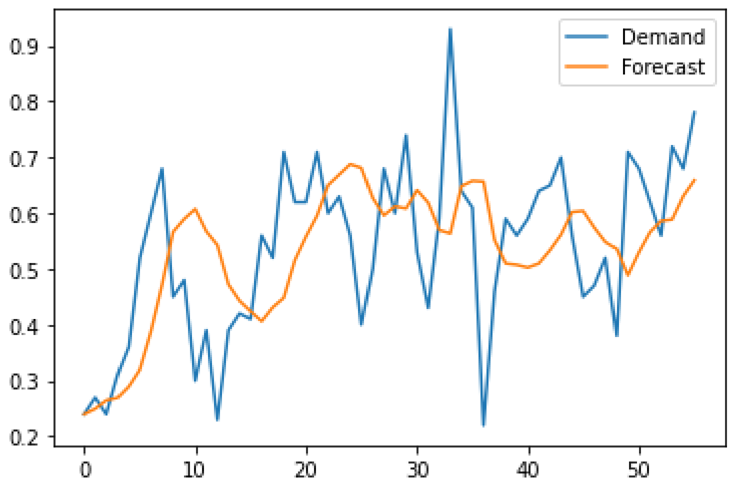
```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is:  0.022918530300947426
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.026714583491085346
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.032875219237949706
```

```
#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.month_temp,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 sqaured 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:  0.004366820789674877
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.0014247816969603806
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.009108135677565483
```

```
#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)
```

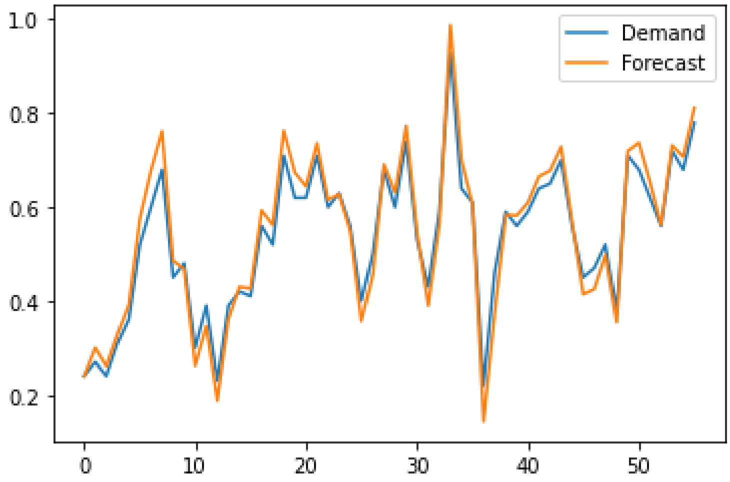
```
Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  0.05049346336504821
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.03173745827232747
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.07327609377920521
```

```
#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.month_temp,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:  0.02195913607989374
Mean of Square Errors for alpha = 0.5 is:  0.02081324209758422
Mean of Square Errors for alpha = 0.8 is:  0.022605624476612792
```

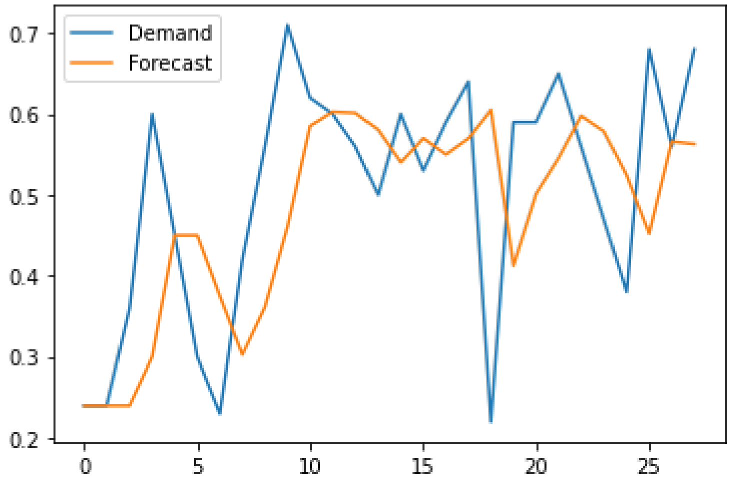
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

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.month_temp,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)
```

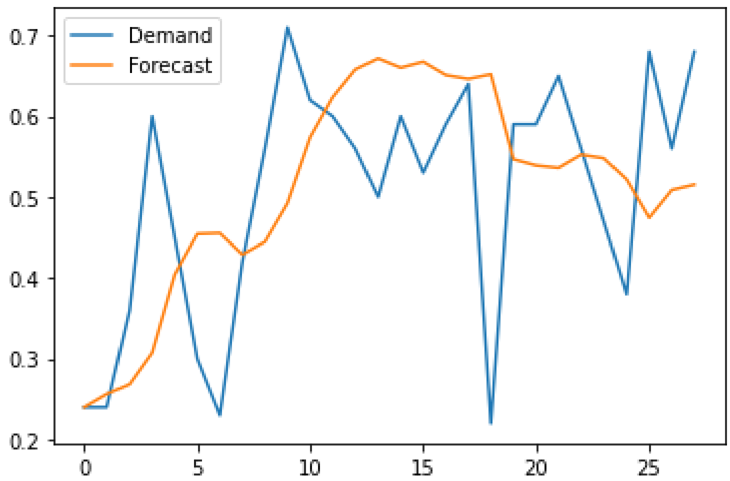
```
Mean of Square Errors for alpha = 0.2,beta= 0.3 is:  0.021507331873940945
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.03126182928406941
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.042888512013966436
```

```
#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.month_temp,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 sqaured 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:  0.004688438880967469
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.001309053829376328
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.011092396420127158
```

```
#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)
```

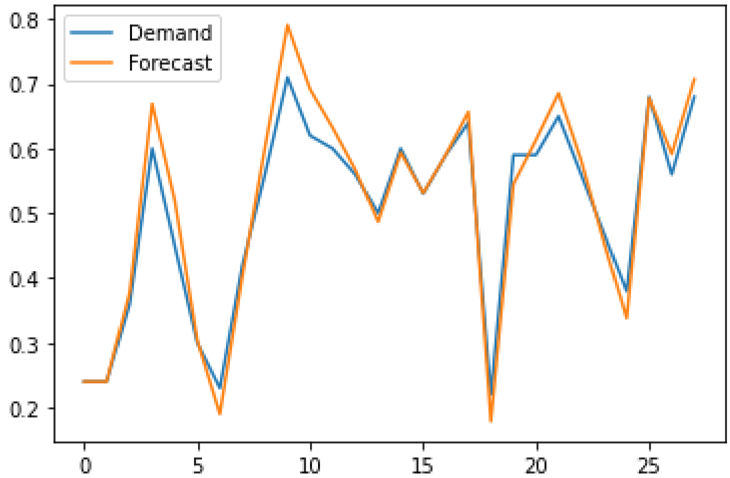
```
Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  0.05491616937825304
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.027618075773923143
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.0823281154744975
```

```
#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 12 Unit

Single Exponential Smoothing

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

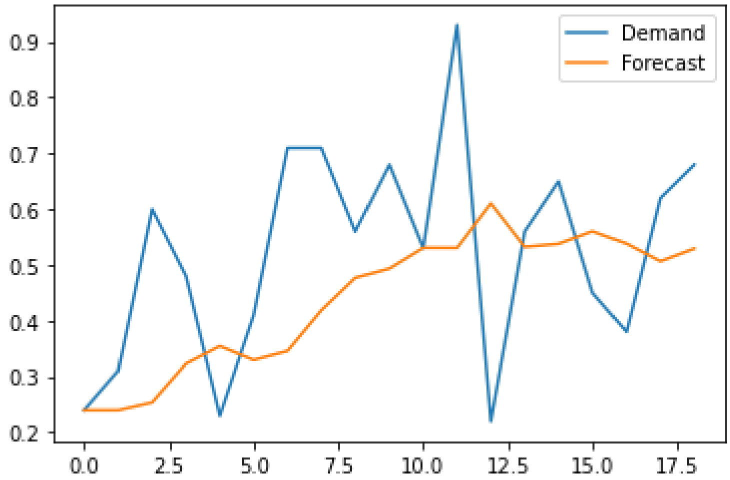
```
Mean of Square Errors for alpha = 0.2 is:  0.04361094545399775
Mean of Square Errors for alpha = 0.5 is:  0.04464140278196214
Mean of Square Errors for alpha = 0.8 is:  0.05507685799681031
```

```
#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.month_temp,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:  0.0439398263063909
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.05824152657925115
```

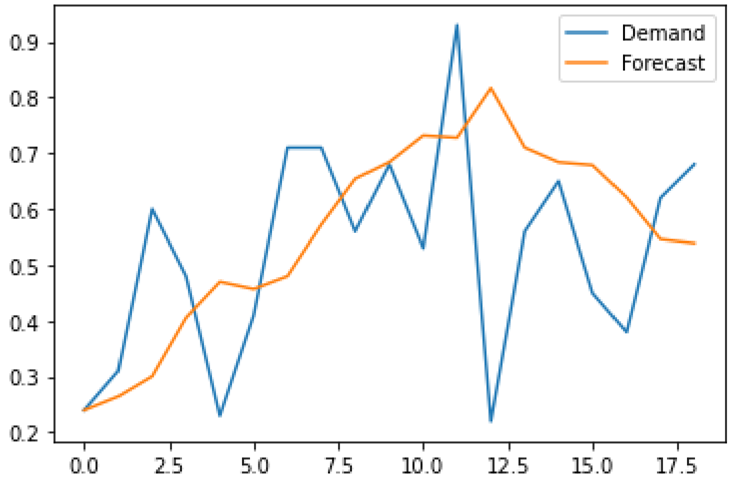
```
    Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.11450367969557813
```

```
#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.month_temp,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 sqaured 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:  0.014130607318364134
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.002819566009035683
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.025998330476397213
```

```
#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)
```

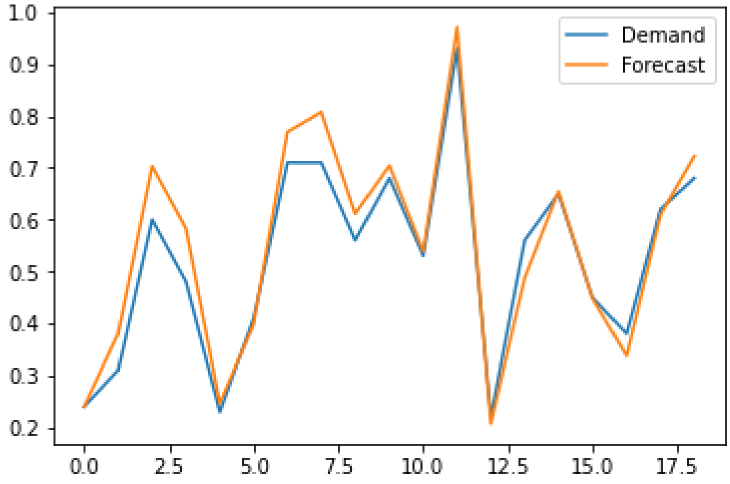
```
Mean Absolute Errors for alpha = 0.2,beta= 0.3, gamma=0.4 is:  0.08899122356016119
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.04068580726768734
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.1283092776849997
```

```
#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 24 Interval

Single Exponential Smoothing

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

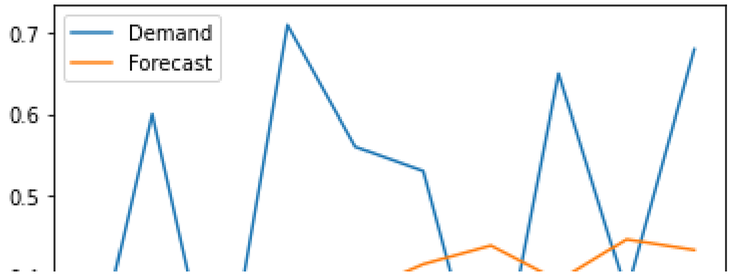
```
Mean of Square Errors for alpha = 0.2 is:  0.053274637834819075
Mean of Square Errors for alpha = 0.5 is:  0.05612971008300782
Mean of Square Errors for alpha = 0.8 is:  0.07436831967254126
```

```
#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.month_temp,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:  0.05175964453756846
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  0.07134782394074757
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  0.1446453483548052
```

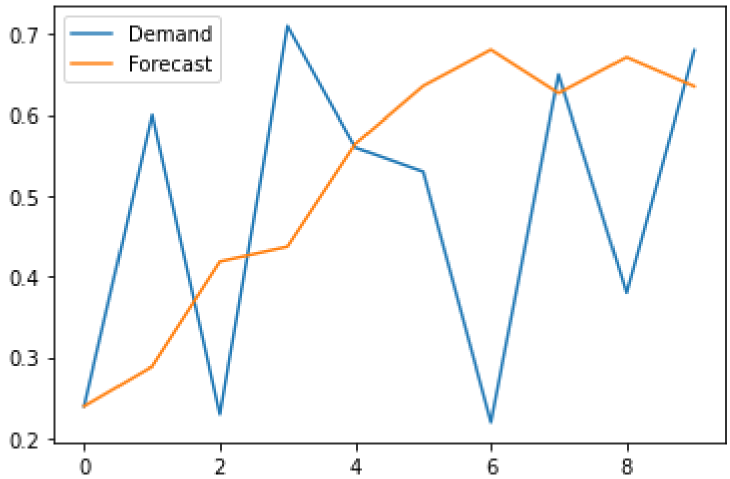
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}
```

```
d3={'Demand':demand,'Forecast':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.month_temp,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
```

```
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 sqaured 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:  0.1873689547447504
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  0.02574568178093526
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  0.03907104292904815
```

```
#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:  0.3890806247672641
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.10894441517707029
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  0.1609046455657641
```

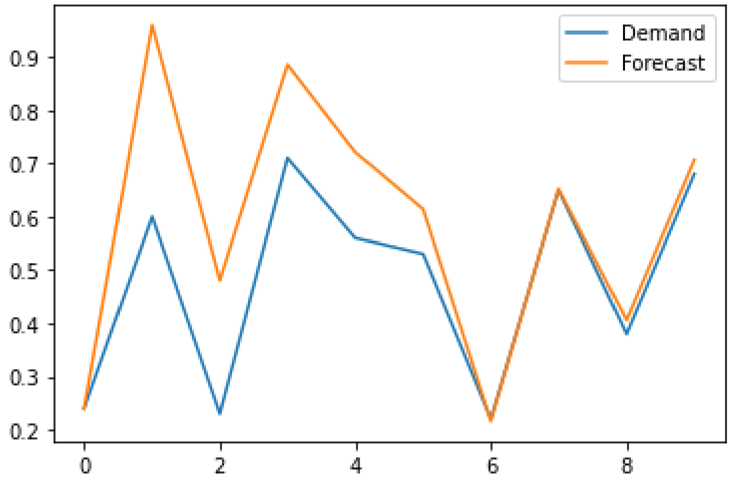
```
#Comparing mse and plotting for least mse
d1={'Demand':demand,'Forecast':forecast1}
d2={'Demand':demand,'Forecast':forecast2}
d3={'Demand':demand,'Forecast':forecast3}

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=['-','-'])
```

```
  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)
```

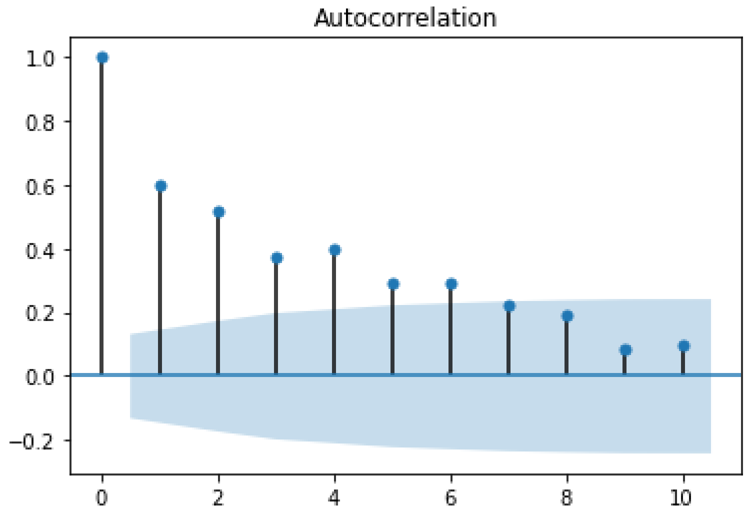
```
Least MSE ses
```

```
[0.011783265150351989, 0.015120728666467782, 0.017969169959880345, 0.020813242097584
Least MSE des
[0.016344385097743625, 0.021152519593798498, 0.022918530300947426, 0.021507331873940
Least MSE tes
[0.0011599649951772768, 0.0010530392502876376, 0.0014247816969603806, 0.00130905382
Least MAE ses
[0.02413624022372913, 0.02413624022372913, 0.026731750173506715, 0.03173745827232747,
Least MAE des
[0.02413624022372913, 0.02413624022372913, 0.026731750173506715, 0.03173745827232747,
Least MAE tes
[0.02413624022372913, 0.026731750173506715, 0.03173745827232747, 0.0276180757739231
```

Applying ACF and PACF

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

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




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

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

Applying AR, MA, ARIMA Models

```
#AR

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

#model summary
print(model_fit.summary())

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

```
                                ARMA Model Results
     ==============================================================================
     Dep. Variable:             month_temp   No. Observations:                  223
```

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

```
     MSE for AR is: 0.011324876395756103

     ------------------------------------------------------------------------------------
```

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

```
     MAE for AR is: 0.08527310148432576

                                      Roots
```

```
#MA

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

#model summary
print(model_fit.summary())

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

```
                              ARMA Model Results
==============================================================================
Dep. Variable:             month_temp   No. Observations:                  223
Model:                     ARMA(0, 3)   Log Likelihood                 175.843
Method:                       css-mle   S.D. of innovations              0.110
Date:            Mon, 01 Mar 2021   AIC                           -341.686
Time:                        05:28:00   BIC                           -324.650
Sample:                             0   HQIC                          -334.808

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

```
mse=mean_squared_error(data.month_temp,data.forecastMA.dropna())
print("MSE for MA is:",mse)
```

```
MSE for MA is: 0.012199643481877933
```

```
                                    Roots
```

```
mae=mean_absolute_error(data.month_temp,data.forecastMA.dropna())
print("MAE for MA is:",mae)
```

```
MAE for MA is: 0.08773075322053521
MA.3           -3.5776           -0.0000j            3.5776           -0.5000
```

```
#ARIMA

#fit model
model=ARIMA(data['diff'].dropna(), order=(2,0,3))
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:                  222
Model:                     ARMA(2, 3)   Log Likelihood                 188.188
Method:                       css-mle   S.D. of innovations              0.103
Date:                Mon, 01 Mar 2021   AIC                           -362.375
Time:                        05:28:01   BIC                           -338.557
Sample:                             0   HQIC                          -352.759

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0009      0.000      2.982      0.003       0.000       0.001
ar.L1.diff    -0.1780      0.110     -1.619      0.106      -0.394       0.038
ar.L2.diff     0.6585      0.109      6.060      0.000       0.446       0.872
ma.L1.diff    -0.3721      0.147     -2.529      0.011      -0.660      -0.084
ma.L2.diff    -0.7833      0.120     -6.501      0.000      -1.019      -0.547
ma.L3.diff     0.1554      0.144      1.081      0.280      -0.126       0.437
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
```

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

```
MSE for MA is: 0.3383298281790968
-----------------------------------------------------------------------------
```

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

```
MAE for MA is: 0.5506916595028443
```

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=-348.834, Time=0.21 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=377.837, Time=0.04 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=-305.484, Time=0.05 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=164.742, Time=0.08 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=-350.560, Time=0.21 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=-337.521, Time=0.08 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=-355.547, Time=0.30 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=inf, Time=0.12 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=-356.991, Time=0.57 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=-347.901, Time=0.36 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=-354.965, Time=0.33 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=-355.692, Time=0.40 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=-363.650, Time=0.66 sec
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=-363.197, Time=0.67 sec
```

```
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=-361.875, Time=0.58 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=-362.613, Time=0.70 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-362.433, Time=0.40 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=-364.620, Time=0.59 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=-360.440, Time=0.69 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=-359.751, Time=0.47 sec

Best model:  ARIMA(2,0,3)(0,0,0)[0] intercept
Total fit time: 7.526 seconds
                               SARIMAX Results                                
==============================================================================
Dep. Variable:                      y   No. Observations:                  223
Model:               SARIMAX(2, 0, 3)   Log Likelihood                 189.310
Date:                Mon, 01 Mar 2021   AIC                           -364.620
Time:                        05:28:44   BIC                           -340.770
Sample:                             0   HQIC                          -354.992
                                - 223                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.1124      0.059      1.896      0.058      -0.004       0.229
ar.L1          0.0463      0.148      0.312      0.755      -0.245       0.337
ar.L2          0.7454      0.115      6.498      0.000       0.521       0.970
ma.L1          0.4373      0.167      2.622      0.009       0.110       0.764
ma.L2         -0.3507      0.117     -3.004      0.003      -0.579      -0.122
ma.L3         -0.1584      0.091     -1.744      0.081      -0.336       0.020
sigma2         0.0107      0.001     10.549      0.000       0.009       0.013
===================================================================================
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):                 0.04
Prob(Q):                              0.84   Prob(JB):                         0.98
Heteroskedasticity (H):               0.72   Skew:                            -0.00
Prob(H) (two-sided):                  0.16   Kurtosis:                         3.06
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step)
```

Best model: ARIMA(2,0,3)(0,0,0)[0]

Final Result

Dataset 2													
Is Dataset Stationery	No												
If yes, why, and if not, why not	For ADF test, we can see that the p-value is below 0.05. Thus, from ADF test, we can say that the dataset is stationary. For KPSS test, Test Statistic is more than Critical Value, thus we reject the null hypothesis. Thus, from KPSS test, we can say that the dataset is non-stationary. Thus, Difference-Stationary.												
If differencing was done, how many times it was done?	One Time												
		Prediction FOR ->											
		MSE						MAE					
		1 UNIT	2 UNIT	4 UNIT	8 UNIT	12 UNIT	24 UNIT	1 UNIT	2 UNIT	4 UNIT	8 UNIT	12 UNIT	24 UNIT
Single Exponential Smoothing	alpha(0.5)	0.011783265	0.015121	0.017969	0.020813	0.043611	0.053275	0.024136	0.024136	0.026732	0.031737	0.027618	0.040686
Double and triple Exponential Smoothing	alpha(0.2), beta(0.3)	0.016344385	0.021153	0.022919	0.021507	0.04394	0.05176	0.024136	0.024136	0.026732	0.031737	0.027618	0.040686
Triple Exponential Smoothing	alpha(0.5), beta(0.6), gamma(0.7)	0.001159965	0.001053	0.001425	0.001309	0.00282	0.025746	0.024136	0.026732	0.031737	0.027618	0.040686	0.108944
Only AR	AR(2)	0.011324876						0.085273					
Only MA	MA(3)	0.012199643						0.087731					
ARIMA	ARIMA(2,0,3)	0.338329828						0.550692					
SARIMA (if the data has seasonality)	season(?)												