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

```
# 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('Dataset4_yahoo304.96.8.14.csv',names = ['yahoo304_96_8_14'])
```

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('yahoo304_96_8_14')
```

```
Results of Dickey-Fuller Test for  yahoo304_96_8_14

Test Statistic                    -3.286733
p-value                            0.015482
#Lags Used                        30.000000
Number of Observations Used     4582.000000
Critical Value (1%)               -3.431778
Critical Value (5%)               -2.862171
Critical Value (10%)              -2.567106
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('yahoo304_96_8_14')
```

```
Results of KPSS Test for  yahoo304_96_8_14
Test Statistic            3.701679
p-value                   0.010000
Lags Used                32.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

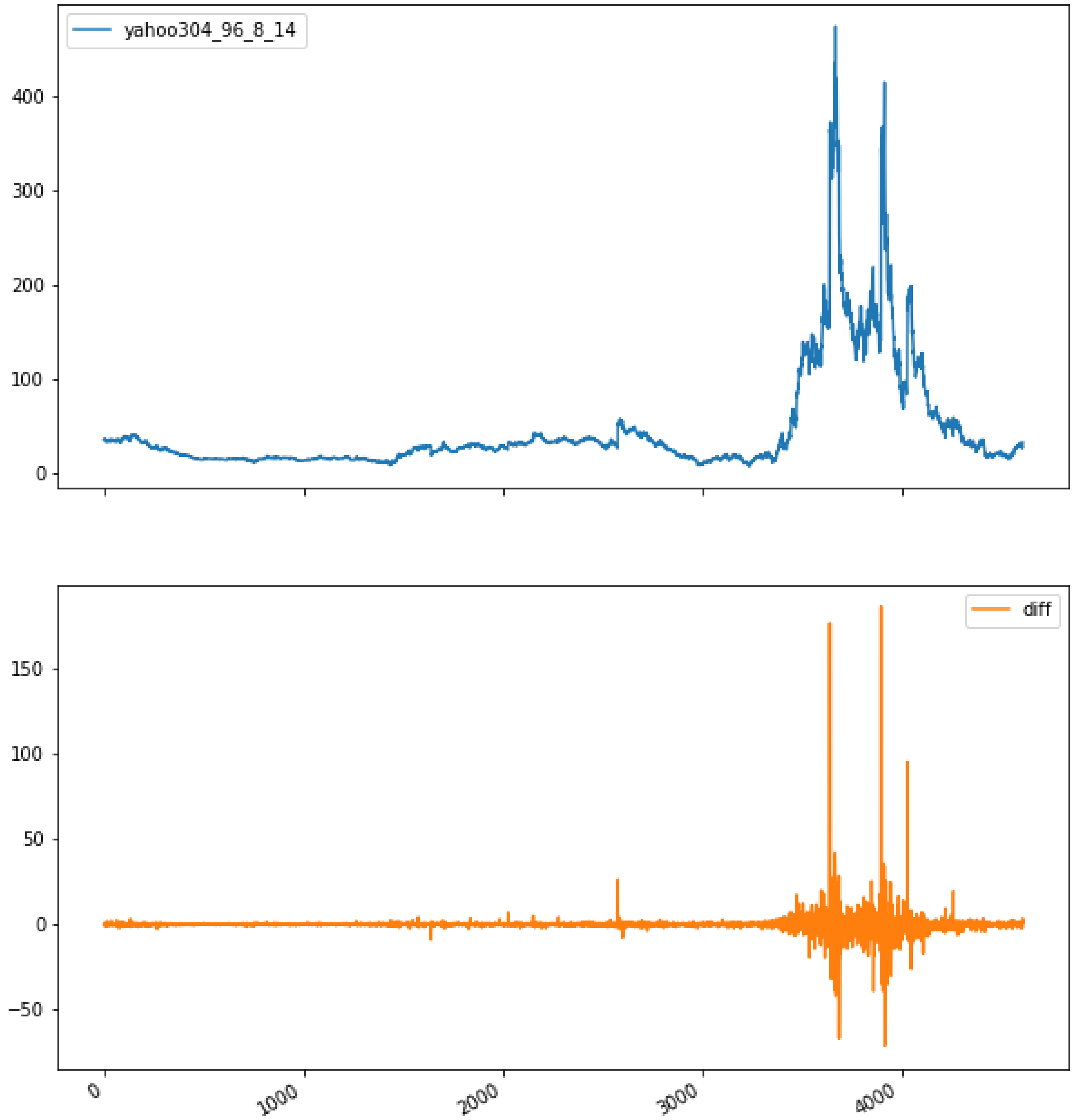
For ADF test, we can see that the p-value is below 0.05. Thus, from ADF test, we can say that the dataset is stationary.

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['yahoo304_96_8_14'].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=[]
```

```
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.yahoo304_96_8_14,1)


#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 = 0.2 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:  96.05545489077616
Mean of Square Errors for alpha = 0.5 is:  47.45126817506155
Mean of Square Errors for alpha = 0.8 is:  36.266977799467504
```

```
#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:  2.977391560154959
Mean Absolute Errors for alpha = 0.5 is:  1.9522762583275515
Mean Absolute Errors for alpha = 0.8 is:  1.6859884166178474
```

```
#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:  0.8
```

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.yahoo304_96_8_14,1)


#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:  98.61340054241282
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  53.99123148363984
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  50.11839964406356
```

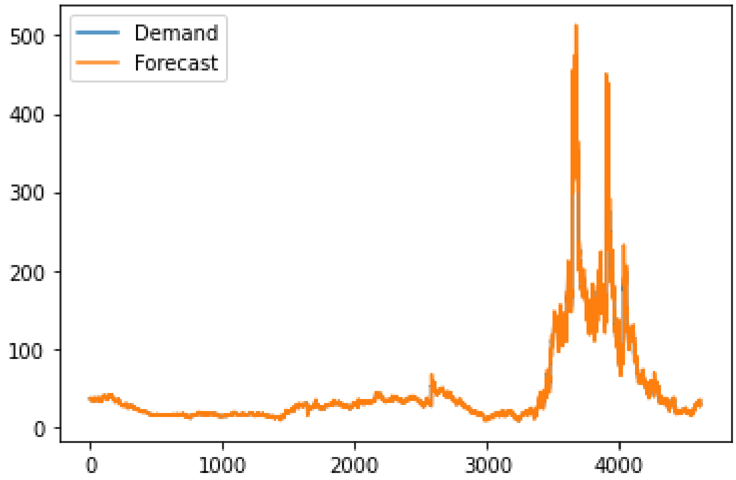
```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

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 = 0.8,beta= 0.9 is: ",mae3)
```

```
Mean Absolute Errors for alpha = 0.2,beta= 0.3 is:  2.964884523576852
Mean Absolute Errors for alpha = 0.5,beta= 0.6 is:  2.152219376160902
Mean Absolute Errors for alpha = 0.8,beta= 0.9 is:  2.0473756875762197
```

```
#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.yahoo304_96_8_14,1)


#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:  11.639338570577138
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  6.348675651210603
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  19.48517402679209
```

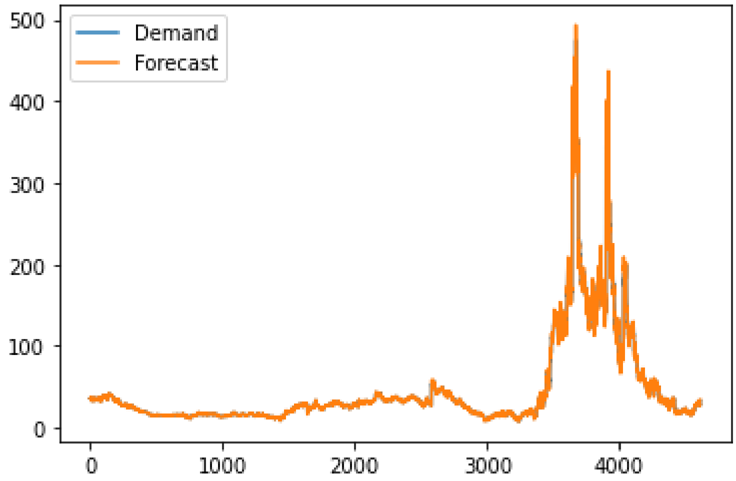
```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

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:  1.032820786741875
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.7323065580072101
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  1.239266078672252
```

```
#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)
```

```
print("gamma: ",gamma3)
df3.plot(style=['-','-'])
```

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



For 1 Unit

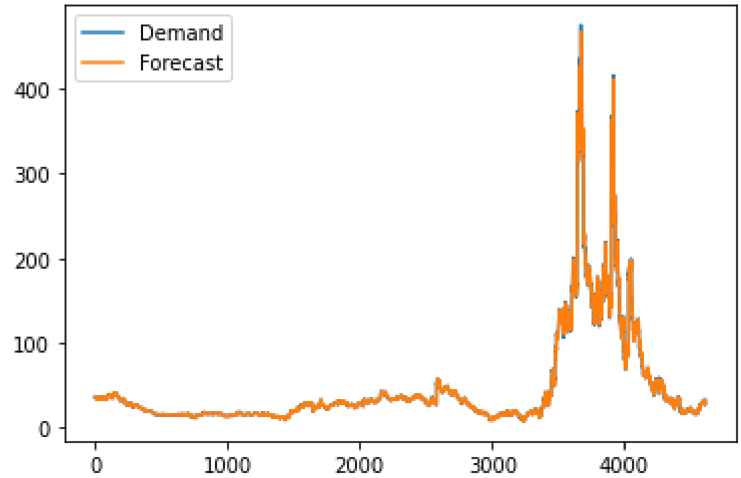
Single Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,1)


#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:  96.05545489077616
Mean of Square Errors for alpha = 0.5 is:  47.45126817506155
Mean of Square Errors for alpha = 0.8 is:  36.266977799467504
```

```
#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.yahoo304_96_8_14,1)
```

```
#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:  98.61340054241282
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  53.99123148363984
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  50.11839964406356
```

```
#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.yahoo304_96_8_14,1)


#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:  11.639338570577138
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  6.348675651210603
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  19.48517402679209
```

```
#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:  1.032820786741875
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  0.7323065580072101
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  1.239266078672252
```

```
#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

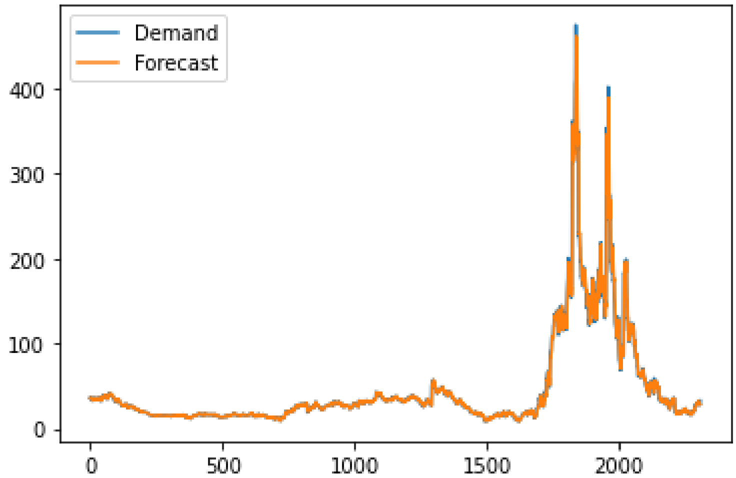
```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,2)


#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:  180.76450708751057
Mean of Square Errors for alpha = 0.5 is:  90.24920445458754
Mean of Square Errors for alpha = 0.8 is:  70.74433216569851
```

```
#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.yahoo304_96_8_14,2)


#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:  189.56455362093845
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  105.91052413310219
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  100.69577635286231
```

```
#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.yahoo304_96_8_14,2)


#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:  22.053348792543282
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  11.733396511437721
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  37.76557145513107
```

```
#Calculating Mean Absolute Errors
```

```
#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:  1.4890138335978669
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  1.0783372848909816
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  1.7916636448595848
```

```
#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 4 Unit

Single Exponential Smoothing

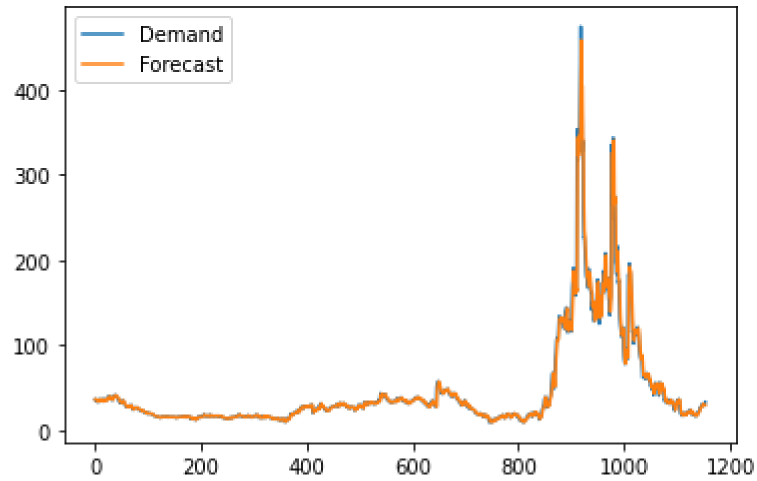
```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,4)


#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:  324.88713214124726
Mean of Square Errors for alpha = 0.5 is:  167.06281335851196
Mean of Square Errors for alpha = 0.8 is:  135.12511739078056
```

```
#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('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.yahoo304_96_8_14,4)
```

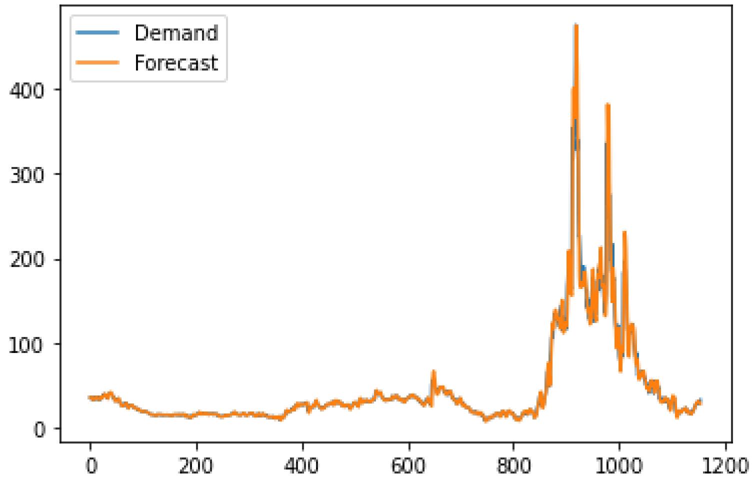
```
#Forecasting
alpha1=0.2
alpha2=0.5
alpha3=0.8

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:  389.5482290439572
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  179.78037941617038
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  200.49519102127783
```

```
#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.yahoo304_96_8_14,4)


#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:  40.92701807206647
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  21.350809413671804
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  70.76855113847974
```

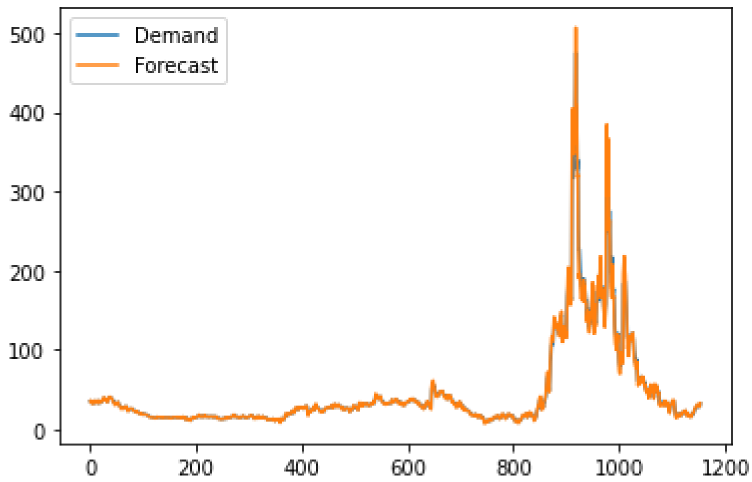
```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

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:  2.1853413729779603
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  1.5846934379909212
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  2.5783465554224296
```

```
#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)
```

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

For 8 Unit

Single Exponential Smoothing

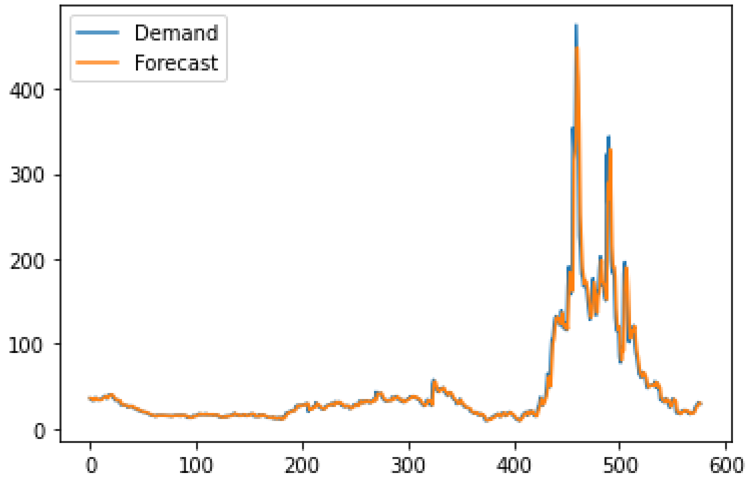
```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,8)


#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:  595.4377000575759
     Mean of Square Errors for alpha = 0.5 is:  354.46155227597797
     Mean of Square Errors for alpha = 0.8 is:  284.3567740733148
```

```
#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

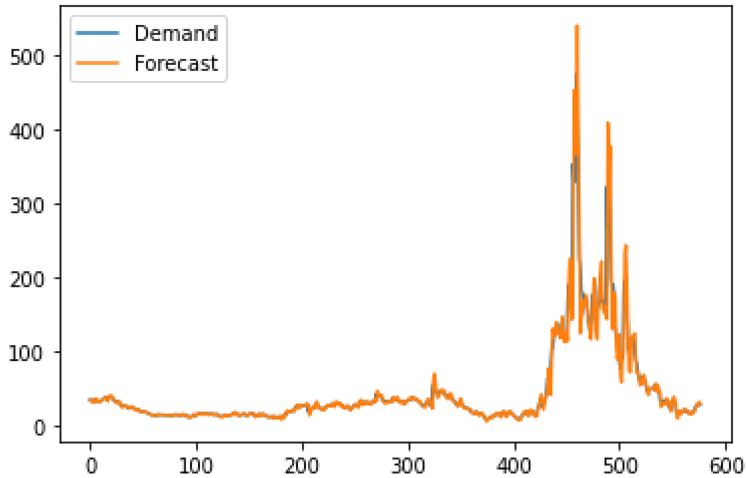
```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,8)


#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 = 0.5,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:  861.5601415083182
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  443.02309370147407
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  401.47865143128286
```

```
#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.yahoo304_96_8_14,8)


#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)
```

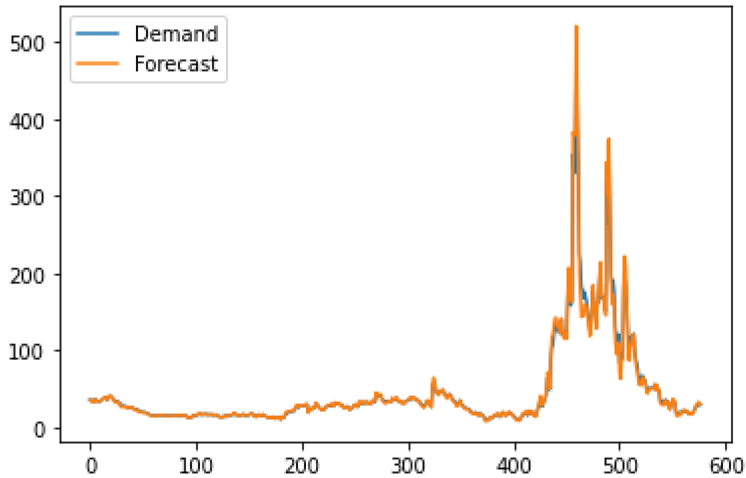
```
Mean of Square Errors for alpha = 0.2,beta= 0.3 gamma=0.4 is:  86.62536866355454
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  43.77633888809179
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  150.47577578775852
```

```
#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:  3.1931530914015007
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  2.444632845486679
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  3.8543419881386067
```

```
#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

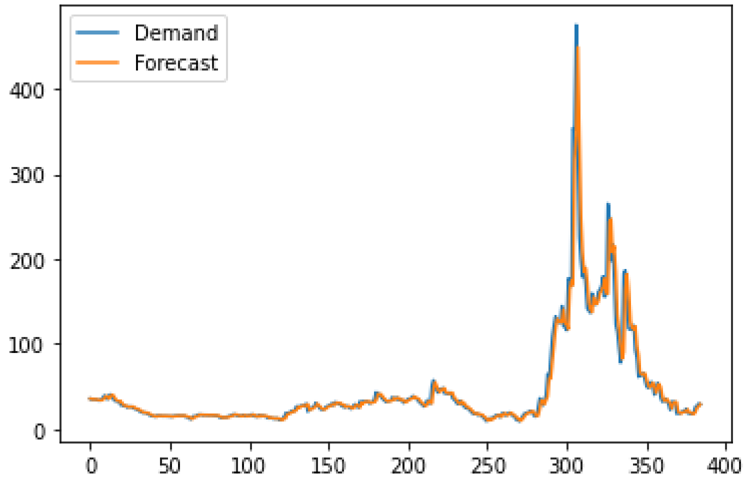
Single Exponential Smoothing

```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,12)


#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:  695.2913065175823
Mean of Square Errors for alpha = 0.5 is:  434.09441905658844
Mean of Square Errors for alpha = 0.8 is:  347.86041803251385
```

```
#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

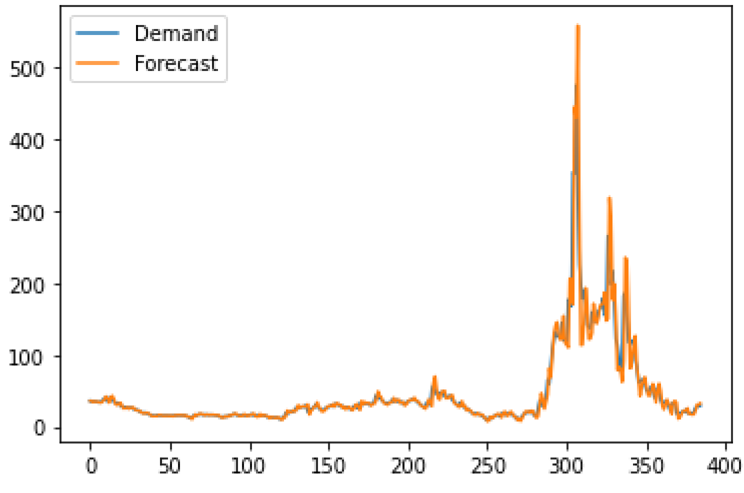
```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,12)


#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:  917.0415874290327
```

```
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  576.8322453010067
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  490.8480707638323
```

```
#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.yahoo304_96_8_14,12)


#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:  105.68945928872901
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  52.452436971990124
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  185.33925379432878
```

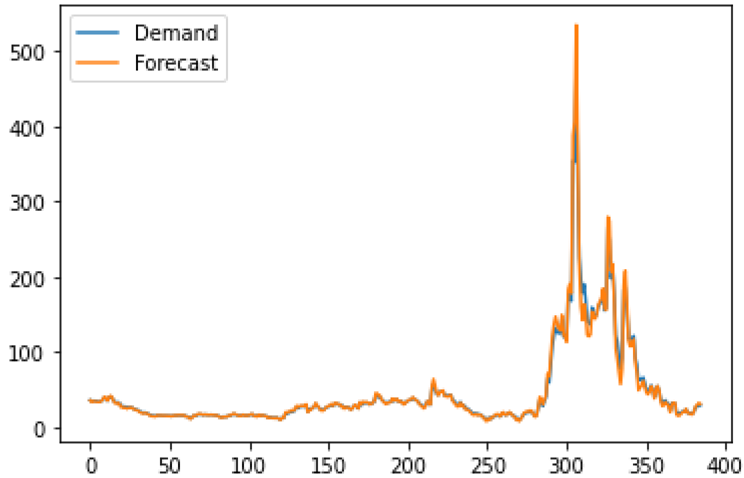
```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

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:  3.718815282718495
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  2.7884595336623166
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  4.5485157158957294
```

```
#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)
  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.yahoo304_96_8_14,24)


#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:  1307.5088358730952
     Mean of Square Errors for alpha = 0.5 is:  945.8271328960523
     Mean of Square Errors for alpha = 0.8 is:  849.7836364017458
```

```
#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

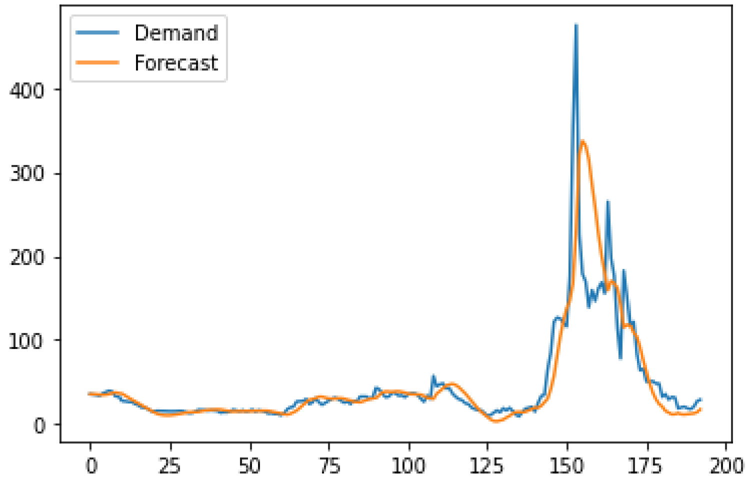
```
#Creating demand list in 'n' intervals
demand=dem_n(data.yahoo304_96_8_14,24)


#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:  1371.9935841256988
Mean of Square Errors for alpha = 0.5,beta= 0.6 is:  1424.4136667011549
Mean of Square Errors for alpha = 0.8,beta= 0.9 is:  1425.973409598658
```

```
#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.yahoo304_96_8_14,24)


#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 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:  223.8033725086531
Mean of Square Errors for alpha = 0.5,beta= 0.6 gamma=0.7 is:  92.95581609431166
Mean of Square Errors for alpha = 0.8,beta= 0.9 gamma=0.95 is:  444.40722954115944
```

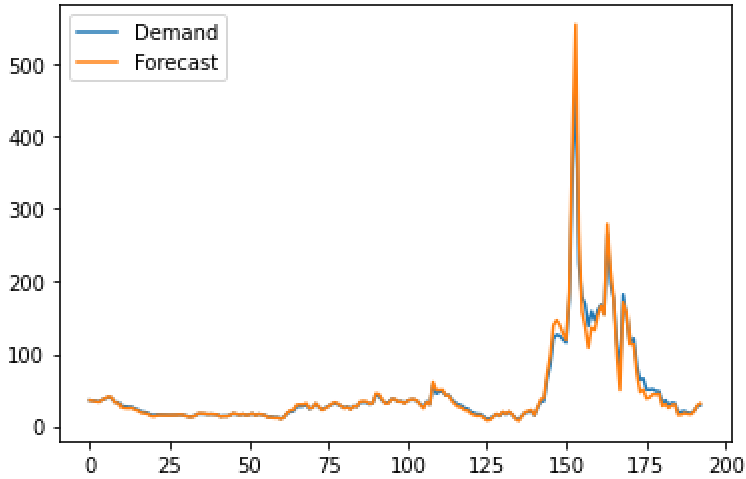
```
#Calculating Mean Absolute Errors
mae1=mean_absolute_error(demand,forecast1)
mae2=mean_absolute_error(demand,forecast2)
mae3=mean_absolute_error(demand,forecast3)

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:  5.802045015904002
Mean Absolute Errors for alpha = 0.5,beta= 0.6, gamma=0.7 is:  3.957796285812707
Mean Absolute Errors for alpha = 0.8,beta= 0.9, gamma=0.95 is:  7.297799596309526
```

```
#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

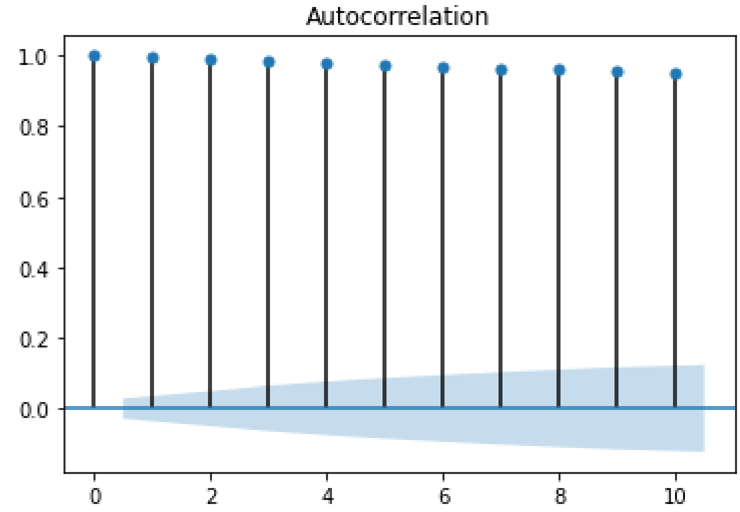
```
print("Least MSE ses")
print(mseses)
print("Least MSE des")
print(msedes)
print("Least MSE tes")
print(msetes)


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

```
Least MSE ses
[36.266977799467504, 70.74433216569851, 135.12511739078056, 284.3567740733148, 347.86
Least MSE des
[50.11839964406356, 100.69577635286231, 179.78037941617038, 401.47865143128286, 490.8
Least MSE tes
[6.348675651210603, 11.733396511437721, 21.350809413671804, 43.77633888809179, 52.452
Least MAE ses
[0.7323065580072101, 0.7323065580072101, 1.0783372848909816, 1.5846934379909212, 2.44
Least MAE des
[0.7323065580072101, 0.7323065580072101, 1.0783372848909816, 1.5846934379909212, 2.44
Least MAE tes
[0.7323065580072101, 1.0783372848909816, 1.5846934379909212, 2.444632845486679, 2.788
```

Applying ACF and PACF

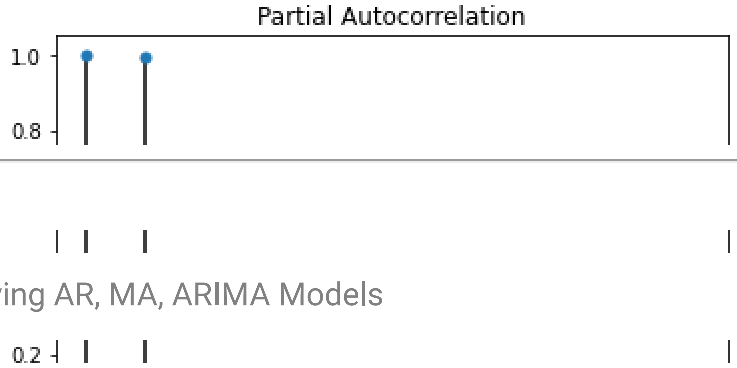
```
#Plotting ACF
plot_acf(data.yahoo304_96_8_14,lags=10)
plt.show
```

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




```
#plotting PACF
plot_pacf(data.yahoo304_96_8_14,lags=10)
plt.show
```

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



Applying AR, MA, ARIMA Models

```
#AR

#fit model
model=ARIMA(data['yahoo304_96_8_14'], order=(3,0,0))
model_fit=model.fit()

#model summary
print(model_fit.summary())

#make prediction
data['forecastAR'] = model_fit.predict()
data[['yahoo304_96_8_14','forecastAR']].plot()
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:       yahoo304_96_8_14   No. Observations:                 4613
Model:                     ARMA(3, 0)   Log Likelihood              -14672.712
Method:                       css-mle   S.D. of innovations              5.820
Date:                Mon, 01 Mar 2021   AIC                          29355.423
Time:                        05:24:04   BIC                          29387.607
Sample:                             0   HQIC                         29366.749
```

```
mse=mean_squared_error(data.yahoo304_96_8_14,data.forecastAR.dropna())
print("MSE for AR is:",mse)
```

```
MSE for AR is: 33.89131601513936
ar.L1.yahoo304_96_8_14     1.0561      0.015     71.546      0.000       1.021
```

```
mae=mean_absolute_error(data.yahoo304_96_8_14,data.forecastAR.dropna())
print("MAE for AR is:",mae)
```

```
MAE for AR is: 1.631457515002413
AR.1            1.0061           +0.0000j            1.0061            0.0000
```

```
#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:                 4612
Model:                     ARMA(0, 2)   Log Likelihood              -14673.764
Method:                       css-mle   S.D. of innovations              5.828
Date:                Mon, 01 Mar 2021   AIC                          29355.529
Time:                        05:24:05   BIC                          29381.275
Sample:                             0   HQIC                         29364.589

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0006      0.093     -0.007      0.995      -0.182       0.181
```

```
mse=mean_squared_error(data.yahoo304_96_8_14[0:-1],data.forecastMA.dropna())
print("MSE for MA is:",mse)
```

```
MSE for MA is: 5509.995684894809
```

```
mae=mean_absolute_error(data.yahoo304_96_8_14[0:-1],data.forecastMA.dropna())
print("MAE for MA is:",mae)
```

```
MAE for MA is: 46.141560834983046
```


```
#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:                 4612
Model:                     ARMA(3, 2)   Log Likelihood              -14636.373
Method:                       css-mle   S.D. of innovations              5.781
Date:                Mon, 01 Mar 2021   AIC                          29286.746
Time:                        05:24:08   BIC                          29331.801
Sample:                             0   HQIC                         29302.602

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0007      0.091     -0.007      0.994      -0.178       0.177
ar.L1.diff    -0.3720      0.017    -21.541      0.000      -0.406      -0.338
ar.L2.diff    -0.8936      0.018    -48.907      0.000      -0.929      -0.858
ar.L3.diff     0.0250      0.015      1.633      0.103      -0.005       0.055
ma.L1.diff     0.4286      0.009     47.771      0.000       0.411       0.446
ma.L2.diff     0.9581      0.012     76.842      0.000       0.934       0.983
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.2209           -1.0279j            1.0514           -0.2837
AR.2           -0.2209           +1.0279j            1.0514            0.2837
AR.3           36.1317           -0.0000j           36.1317           -0.0000
```

```
mse=mean_squared_error(data.yahoo304_96_8_14[0:-1],data.forecastARIMA.dropna())
print("MSE for MA is:",mse)
```

```
MSE for MA is: 5511.605519032832
```

```
mae=mean_absolute_error(data.yahoo304_96_8_14[0:-1],data.forecastARIMA.dropna())
print("MAE for MA is:",mae)
```

```
MAE for MA is: 46.14158164074054
```

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=29358.877, Time=0.40 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=52832.194, Time=0.08 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.19 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=46916.004, Time=0.68 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=29360.811, Time=0.71 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=29357.882, Time=0.48 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=42222.823, Time=2.91 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=29345.809, Time=2.09 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=29290.882, Time=4.59 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=29348.048, Time=2.03 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=29296.227, Time=4.54 sec
```

```
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=29340.140, Time=1.17 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=29295.962, Time=9.00 sec

Best model:  ARIMA(3,0,2)(0,0,0)[0]
Total fit time: 28.895 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 4613
Model:               SARIMAX(3, 0, 2)   Log Likelihood              -14639.441
Date:                Mon, 01 Mar 2021   AIC                          29290.882
Time:                        05:25:11   BIC                          29329.502
Sample:                             0   HQIC                         29304.473
                               - 4613
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6191      0.008     77.475      0.000       0.603       0.635
ar.L2         -0.5112      0.010    -49.807      0.000      -0.531      -0.491
ar.L3          0.8846      0.008    105.040      0.000       0.868       0.901
ma.L1          0.4173      0.007     55.685      0.000       0.403       0.432
ma.L2          0.9431      0.008    123.834      0.000       0.928       0.958
sigma2        33.4673      0.092    363.507      0.000      33.287      33.648
===================================================================================
Ljung-Box (L1) (Q):                   1.46   Jarque-Bera (JB):          36931252.63
Prob(Q):                              0.23   Prob(JB):                         0.00
Heteroskedasticity (H):             442.37   Skew:                            13.32
Prob(H) (two-sided):                  0.00   Kurtosis:                       440.53
===================================================================================

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

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

Final Result

Dataset 4													
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												
		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(0.8)	36.2669778	70.74433	135.1251	284.3568	347.8604	849.7836	0.732307	0.732307	1.078337	1.584693	2.444633	2.78846
Double and triple Exponential Smoothing	alpha(0.8), beta(0.9)	50.11839964	100.6958	179.7804	401.4787	490.8481	1371.994	0.732307	0.732307	1.078337	1.584693	2.444633	2.78846
Triple Exponential Smoothing	alpha(0.5), beta(0.6), gamma(0.7)	6.348675651	11.7334	21.35081	43.77634	52.45244	92.95582	0.732307	1.078337	1.584693	2.444633	2.78846	3.957796
Only AR	AR(3)	33.9						1.63					
Only MA	MA(2)	5509.995685						46.14156					
ARIMA	ARIMA(3,0,2)	5511.605519						46.14156					
SARIMA (if the data has seasonality)	season(?)												