```
import pandas as pd
import numpy as np

data = pd.read_excel("Delhi.xlsx", usecols=('#Data','Unnamed: 1'))
data = data.rename(columns={'#Data':'Date', 'Unnamed: 1': 'PM'})
data = data.drop([0,1])
data.sort_values("Date", inplace = True)
data.reset_index(drop=True, inplace=True)
data.dropna(inplace=True)
data[['PM']] = data[['PM']].replace('-', np.nan)
data['PM'] = data.PM.interpolate(method = 'nearest')
data1 = data.copy()


import datetime as dt
data1['Date'] =  pd.to_datetime(data1['Date'],format='%Y-%m-%d %H:%M:%S')
data1["year"] = data1.Date.dt.strftime("%Y")
data1["month"] = data1.Date.dt.strftime("%b")
data1["Day"] = data1.Date.dt.strftime("%d")
data1['hour'] = data1.Date.dt.strftime("%H")
data1['dayofweek'] = data1.Date.dt.day_name()
data1['month'] = pd.Categorical(data1['month'], categories=data1.month.unique())
data1['dayofweek'] = pd.Categorical(data1['dayofweek'],
categories=data1.dayofweek.unique())
data1.PM.plot(figsize = (24,6))
data2 = data1.copy()

import seaborn as sns
heatmap = pd.pivot_table(data1,values=['PM'],
                         index=["month"],columns=["dayofweek"],aggfunc="mean")
sns.heatmap(heatmap,annot=True,fmt="g")
sns.boxplot(x="hour",y="PM",data=data1)
sns.boxplot(x="year",y="PM",data=data1)
sns.factorplot("month","PM",data=data1,kind="box")
sns.lineplot(x="hour",y="PM",hue="month",data=data1)


from statsmodels.tsa.stattools import adfuller
def adf_test(series):
    result = adfuller(series.dropna())
    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)
    if result[1] <= 0.05:
        print("Reject the null hypothesis")
        print("Data is stationary")
    else:
        print("Fail to reject the null hypothesis")
        print("Data is non-stationary")
adf_test(data1['PM'])
result = adfuller(data1['PM']) # 0.006588878938254434
```

```
import matplotlib.pylab as plt
import statsmodels.graphics.tsaplots as sgt
sgt.plot_acf(data1.PM, lags=24, zero = False)
plt.title("ACF", size=24)
plt.show()

sgt.plot_pacf(data1.PM, lags = 24, zero = False, method = ('ols'))
plt.title("PACF", size=24)
plt.show()

from pmdarima import auto_arima
stepwise_fit = auto_arima(data1['PM'], start_p=0, start_q=0,
                          max_p=4, max_q=4, m=24,
                          seasonal=True,
                          d=None, trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)
stepwise_fit.summary()

Train = data1['PM'][:2326]

import statsmodels.api as sma
from statsmodels.tsa.statespace.sarimax import SARIMAX
mod = sma.tsa.statespace.SARIMAX(Train, trend='ct', order=(2,1,2),
seasonal_order=(2,0,2,4))
results = mod.fit(disp=-1)

data1['forecast1'] = results.predict (0,2326, dynamic=False, typ='levels')
data1['forecast1'] = data1['forecast1'].shift(-1)
data1[['PM', 'forecast1']][0:2325]
rmse_Train =
np.sqrt(np.mean((np.array(data1['PM'][0:2325])-np.array(data1['forecast1'][0:2325]))
**2))
rmse_Train

data1['forecast'] = results.predict (2325,2375, dynamic=False, typ='levels')
data1['forecast'] = data1['forecast'].shift(-1)
data1[['PM', 'forecast']][2324:2375]
rmse_Test =
np.sqrt(np.mean((np.array(data1['PM'][2325:2373])-np.array(data1['forecast'][2325:23
73]))**2))
rmse_Test

data1[['PM', 'forecast1']][:2326].plot()#plotting Test results
data1[['PM', 'forecast']][2325:2373].plot()#plotting Train results

# prediction for full data
mod_full = sma.tsa.statespace.SARIMAX(data1['PM'], trend='ct', order=(2,1,1),
seasonal_order=(2,0,2,4))
results_full = mod_full.fit(disp=-1)
print (results_full.summary())

x = np.arange(2375,2400,1)
future = pd.DataFrame(index=x, columns= data1.columns)
data1 = pd.concat([data1, future])

data1['forecast2'] = results_full.predict (2365, 2400, dynamic=False, typ='levels')
data1['forecast2'] = data1['forecast2'].shift(-1)
data1[['PM', 'forecast2']][2365:2400].plot()

data1['forecast3'] = results.predict (0,2374, dynamic=False, typ='levels')
data1['forecast3'] = data1['forecast3'].shift(-1)
rmse_SARIMAX =
np.sqrt(np.mean((np.array(data1['PM'][:2373])-np.array(data1['forecast3'][:2373]))**
2))
rmse_SARIMAX

##################################################################
dummy= pd.get_dummies(data2.dayofweek)
data2= data2.join(dummy)
data2.columns

data2['t'] = np.arange(0,2374)
data2['t_squre'] = data2.t*data2.t
data2.head(30)
data2['log_PM']= np.log(data2.PM)

Train = data2.head(2350)
Test = data2.tail(24)
Train.columns
####################### L I N E A R ##########################
import statsmodels.formula.api as smf

linear_model = smf.ols('PM~t',data=Train).fit()
pred_linear =  pd.Series(linear_model.predict(pd.DataFrame(Test['t'])))
rmse_linear = np.sqrt(np.mean((np.array(Test['PM'])-np.array(pred_linear))**2))
rmse_linear # 72.2556945797981

##################### Exponential ##############################
Exp = smf.ols('log_PM~t',data=Train).fit()
pred_Exp = pd.Series(Exp.predict(pd.DataFrame(Test['t'])))
rmse_Exp = np.sqrt(np.mean((np.array(Test['PM'])-np.array(np.exp(pred_Exp)))**2))
rmse_Exp # 74.65163477926485

#################### Quadratic ################################
Quad = smf.ols('PM~t+t_squre',data = Train).fit()
pred_Quad = pd.Series(Quad.predict(Test[["t","t_squre"]]))
rmse_Quad = np.sqrt(np.mean((np.array(Test['PM'])-np.array(pred_Quad))**2)
rmse_Quad # 8.65467132431136

################### Additive seasonality ########################

add_sea =
smf.ols('PM~Monday+Tuesday+Wednesday+Thursday+Friday+Saturday+Sunday',data=Train).fi
t()
pred_add_sea = pd.Series(add_sea.predict(Test[['Monday', 'Tuesday', 'Wednesday',
'Thursday',
        'Friday', 'Saturday', 'Sunday']]))
rmse_add_sea = np.sqrt(np.mean((np.array(Test['PM'])-np.array(pred_add_sea))**2))
rmse_add_sea #119.15951554643155

################## Additive Seasonality Quadratic ############################
add_sea_quad =
smf.ols('PM~t+t_squre+Monday+Tuesday+Wednesday+Thursday+Friday+Saturday+Sunday',data
=Train).fit()
pred_add_sea_quad = pd.Series(add_sea_quad.predict(Test[['Monday', 'Tuesday',
'Wednesday', 'Thursday',
        'Friday', 'Saturday', 'Sunday','t','t_squre']]))
rmse_add_sea_quad =
np.sqrt(np.mean((np.array(Test['PM'])-np.array(pred_add_sea_quad))**2))
rmse_add_sea_quad #68.34950376307623

################## Multiplicative Seasonality ##################
Mult_sea =
smf.ols('log_PM~Monday+Tuesday+Wednesday+Thursday+Friday+Saturday+Sunday',data=Train
).fit()
pred_Mult_sea = pd.Series(Mult_sea.predict(Test[['Monday', 'Tuesday', 'Wednesday',
'Thursday',
        'Friday', 'Saturday', 'Sunday']]))
rmse_Mult_sea =
np.sqrt(np.mean((np.array(Test['PM'])-np.array(np.exp(pred_Mult_sea)))**2))
rmse_Mult_sea# 80.78401106995817

#################Multiplicative Additive Seasonality ###########
Mul_Add_sea =
smf.ols('log_PM~t+Monday+Tuesday+Wednesday+Thursday+Friday+Saturday+Sunday',data=Tra
in).fit()
pred_Mult_add_sea = pd.Series(Mul_Add_sea.predict(Test[['Monday', 'Tuesday',
'Wednesday', 'Thursday',
        'Friday', 'Saturday', 'Sunday','t']]))
rmse_Mul_Add_sea =
np.sqrt(np.mean((np.array(Test['PM'])-np.array(pred_Mult_add_sea))**2))
rmse_Mul_Add_sea #116.27264438391724

################## Multiplicative Quadratic trend ################
Mul_Add_sea_quad =
smf.ols('log_PM~t+t_squre+Monday+Tuesday+Wednesday+Thursday+Friday+Saturday+Sunday',
data=Train).fit()
```

```
pred_Mult_add_sea_quad = pd.Series(Mul_Add_sea_quad.predict(Test[['Monday',
'Tuesday', 'Wednesday', 'Thursday','Friday', 'Saturday','Sunday','t','t_squre']]))
rmse_Mul_Add_sea_quad =
np.sqrt(np.mean((np.array(Test['PM'])-np.array(pred_Mult_add_sea_quad))**2))
rmse_Mul_Add_sea_quad #116.20108280459638
################## Testing ########################################
data = {"MODEL":pd.Series(["rmse_linear","rmse_Exp","rmse_Quad","rmse_add_sea",
                            "rmse_add_sea_quad","rmse_Mult_sea","rmse_Mul_Add_sea",
                            "rmse_Mul_Add_sea_quad","rmse_SARIMAX"]),

"RMSE_Values":pd.Series([rmse_linear,rmse_Exp,rmse_Quad,rmse_add_sea,rmse_add_sea_qu
ad,

rmse_Mult_sea,rmse_Mul_Add_sea,rmse_Mul_Add_sea_quad,
                                   rmse_SARIMAX])}

table_rmse=pd.DataFrame(data)
table_rmse
```