Final Project Report
DATS 6313 Time Series Analysis & Modeling

Dr. Reza Jafari

submitted by

Julia Jin

CJ, 05/04/2022

Table of Content

Abstract

In the final project, we would like to develop a model that best represents the value PM2.5 of US embassy in Beijing, China. We begin by preprocess and check the stationarity of the dataset. Then we analyze the trend and seasonality. Throughout the project we develop several different models. To analyze their performance, we check the variance of forecast errors. Once we select the model, we derive a forecast function for it.

Introduction

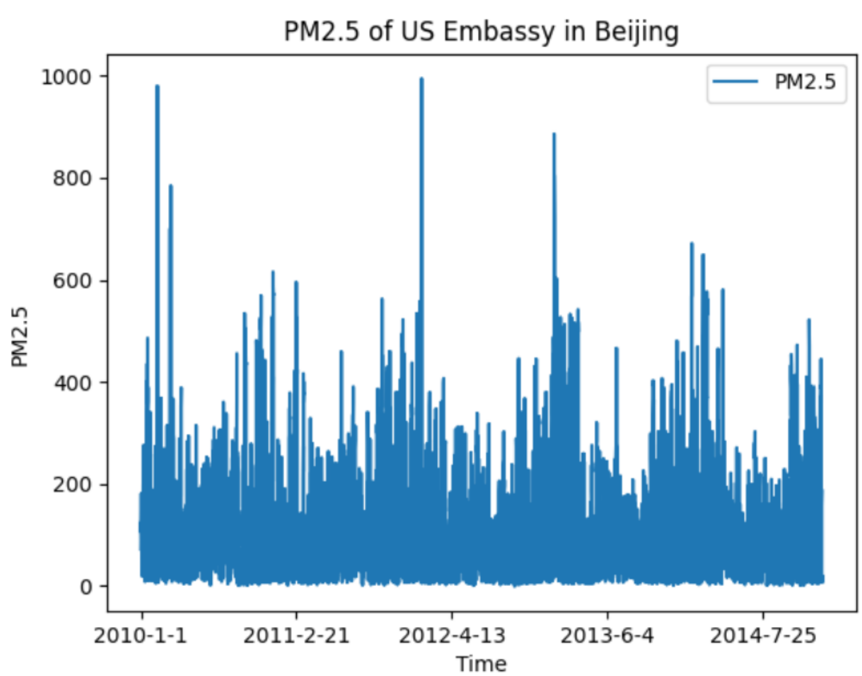
The models we develop in this project include holt-winter's method, average method, naïve method, drift method, simple exponential smoothing, multiple linear regression and autoregressive moving average (ARMA). For linear regression, we apply feature selection, hypothesis tests (t-test, F-test), check AIC, BIC, Adjusted R2 and analyze its residuals. For ARMA model, we plot GPAC table to find the order, use Levenberg Marquardt algorithm to estimate parameters and conduct diagnostic analysis.

Description of dataset

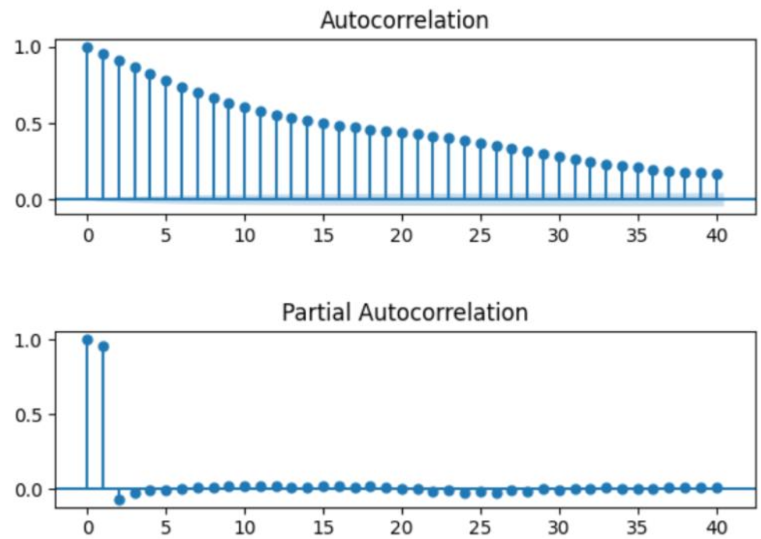
This hourly data set contains the PM2.5 data of US Embassy in Beijing. Meanwhile, meteorological data from Beijing Capital International Airport are also included. The dataset's time period is between Jan 1st, 2010 to Dec 31st, 2014. Missing data are denoted as NA. The dataset has 43824 observations and 7 feature after we preprocessed time related attributes

Attribute Information:

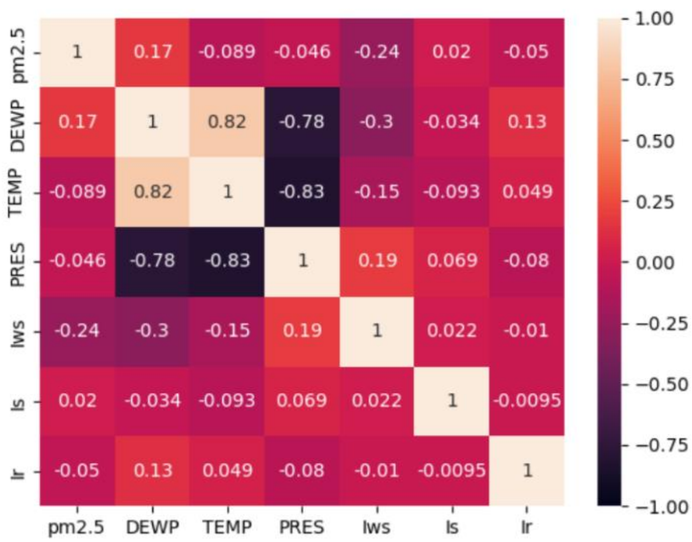
No: row number
year: year of data in this row
month: month of data in this row
day: day of data in this row
hour: hour of data in this row
pm2.5: PM2.5 concentration (ug/m^3)
DEWP: Dew Point
TEMP: Temperature
PRES: Pressure
cbwd: Combined wind direction
Iws: Cumulated wind speed (m/s)
Is: Cumulated hours of snow
Ir: Cumulated hours of rain



6.c



6.d



6.e
see appendix

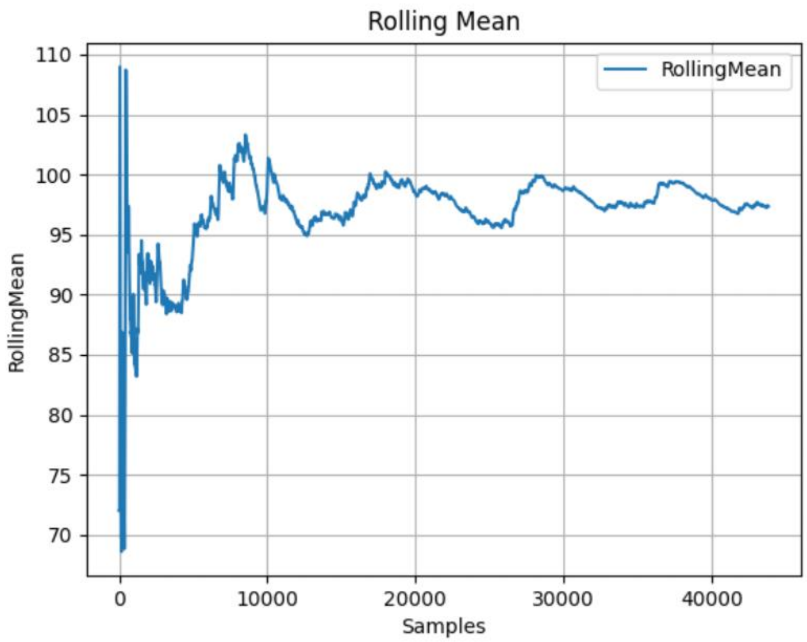
7. Stationarity

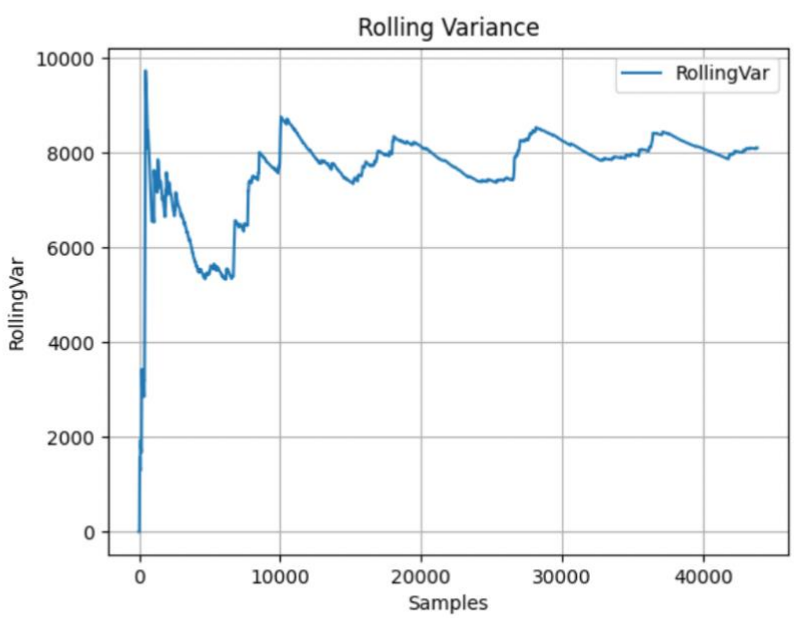
ADF Test

```
ADF Statistic: -21.274057
p-value: 0.000000
Critical Values:
	1%: -3.430
	5%: -2.862
	10%: -2.567
```

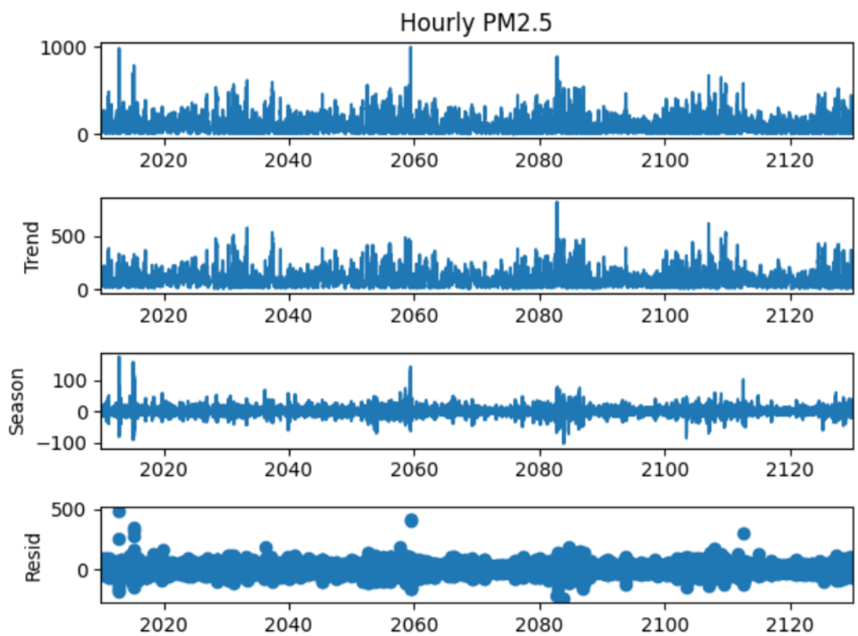
KPSS Test

```
Test Statistic              0.069882
p-value                     0.100000
Lags Used                 114.000000
Critical Value (10%)        0.347000
Critical Value (5%)         0.463000
Critical Value (2.5%)       0.574000
Critical Value (1%)         0.739000
dtype: float64
```





8. Time Series Decomposition




```
Strength of seasonality = 0.28418110877015645
Strength of trend = 0.956533189831818
```

9. Holt-Winter method

```
>>> holt_f
                    0
date
2013-12-31  15.024076
2013-12-31  15.018361
2013-12-31  15.012704
2013-12-31  15.007102
2013-12-31  15.001557
                  ...
2014-12-31  14.452582
2014-12-31  14.452582
2014-12-31  14.452582
2014-12-31  14.452582
2014-12-31  14.452582
```

10. Feature Selection

Singular Values = [4.53141523e+10 1.10410292e+08 1.34600207e+07 1.26300893e+06
 8.54043998e+04 2.49372519e+04]
Conditional Number = 1348.0085094164033
(backward stepwise reduction)

11. Base Models

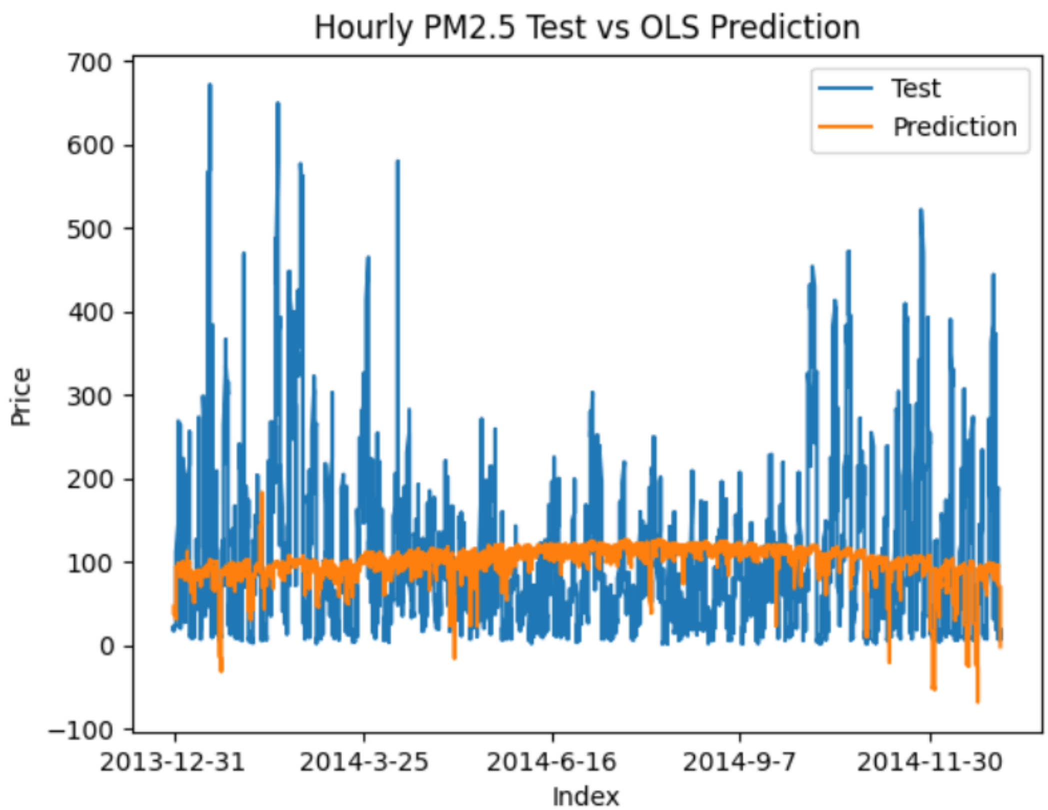
Variance of Average forecast error: 8654.060695897733
Variance of Naive forecast error: 8654.060695897733
Variance of Drift forecast error: 8589.036601482498
Variance of SES forecast error: 8654.060695897733
Variance of ARMA forecast error: 8656.015192378709

ARMA model performance the test set is about the same as our base models.

12. Multiple Linear Regression

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                  pm2.5   R-squared (uncentered):                   0.581
Model:                            OLS   Adj. R-squared (uncentered):              0.581
Method:                 Least Squares   F-statistic:                              9718.
Date:                Wed, 04 May 2022   Prob (F-statistic):                        0.00
Time:                        17:35:12   Log-Likelihood:                     -2.0571e+05
No. Observations:               35059   AIC:                                  4.114e+05
Df Residuals:                   35054   BIC:                                  4.115e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
DEWP           0.9061      0.033     27.230      0.000       0.841       0.971
PRES           0.1037      0.001    200.121      0.000       0.103       0.105
Iws           -0.3578      0.009    -38.299      0.000      -0.376      -0.339
Is             3.6242      0.591      6.135      0.000       2.466       4.782
Ir            -4.1712      0.309    -13.485      0.000      -4.777      -3.565
==============================================================================
Omnibus:                    15186.324   Durbin-Watson:                   0.100
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            87622.327
Skew:                           2.025   Prob(JB):                         0.00
Kurtosis:                       9.602   Cond. No.                     1.32e+03
==============================================================================
```

The final model we have is pm2.5 = .9061*DEWP + .1037*PRES - .3578*Iws + 3.6242* Is – 4.1712*Ir.

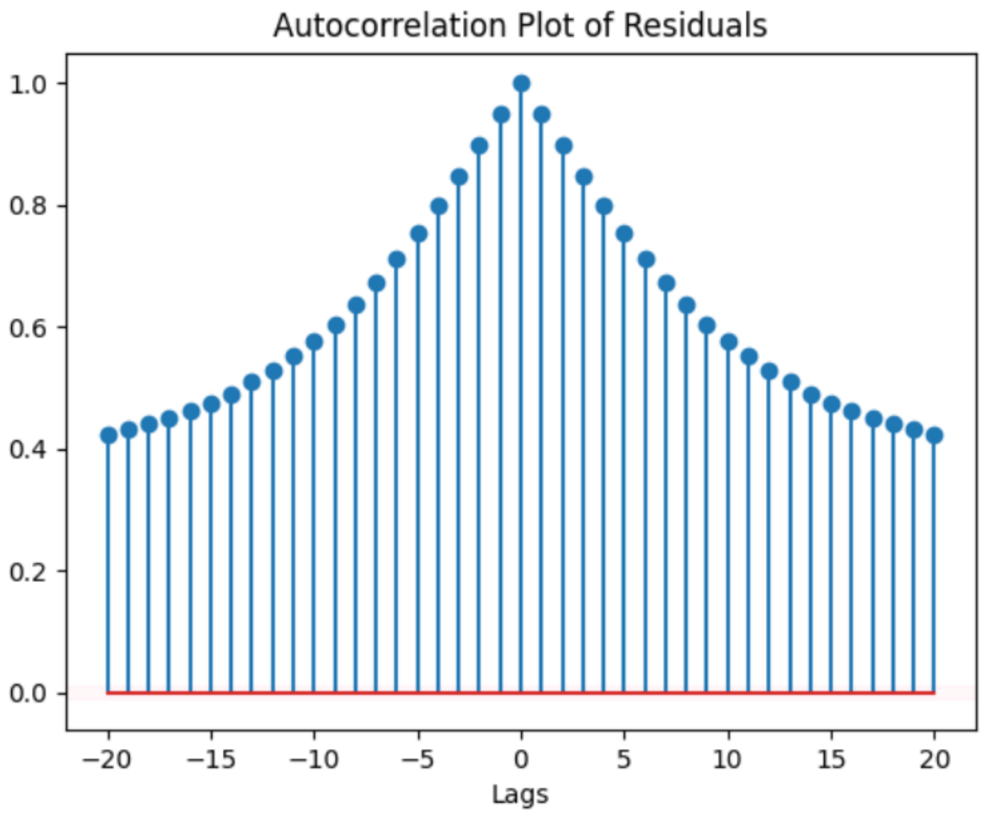


12.b
The p-value of ‘DEWP’ is less than the significance level 0.05. We can reject the null hypothesis and conclude that there is a positive relationship between ‘DEWP’ and ‘pm2.5’. Similarly, we can conclude that there is a positive relationship between ‘PRES’ and ‘pm2.5’, ‘Is’ and ‘pm2.5’, and a negative relationship between ‘Iws’ and ‘pm2.5’, ‘Ir’ and ‘pm2.5’.

The F-statistics is 0 which means we can reject the null hypothesis and conclude that our model is a better fit than the intercept-only model.

12.c
AIC = 4.114e+05
BIC = 4.115e+05
R2 = 0.581
Adjusted R2 = 0.581

12.d



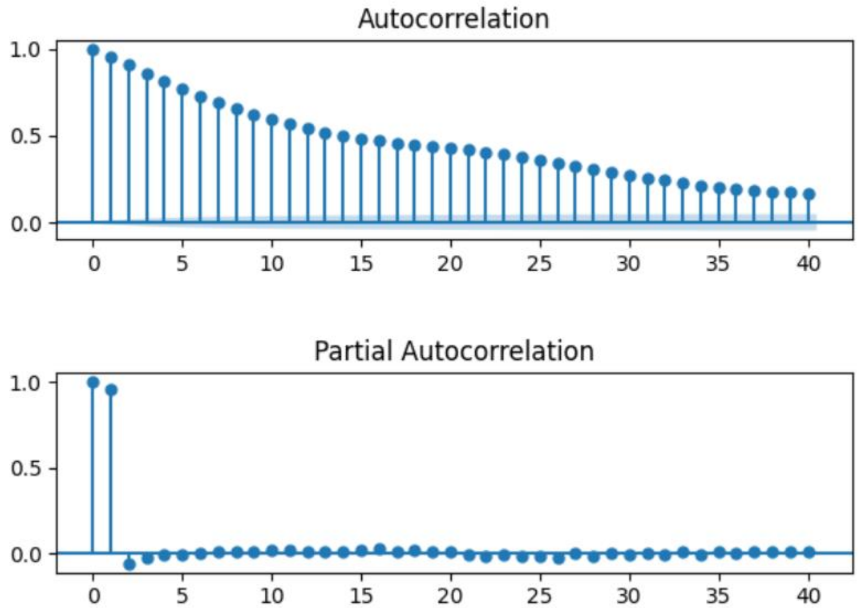
Most residuals are outside the insignificance band, which suggest the residuals are correlated.

12.e
Q-value of residuals: 280489.14463559

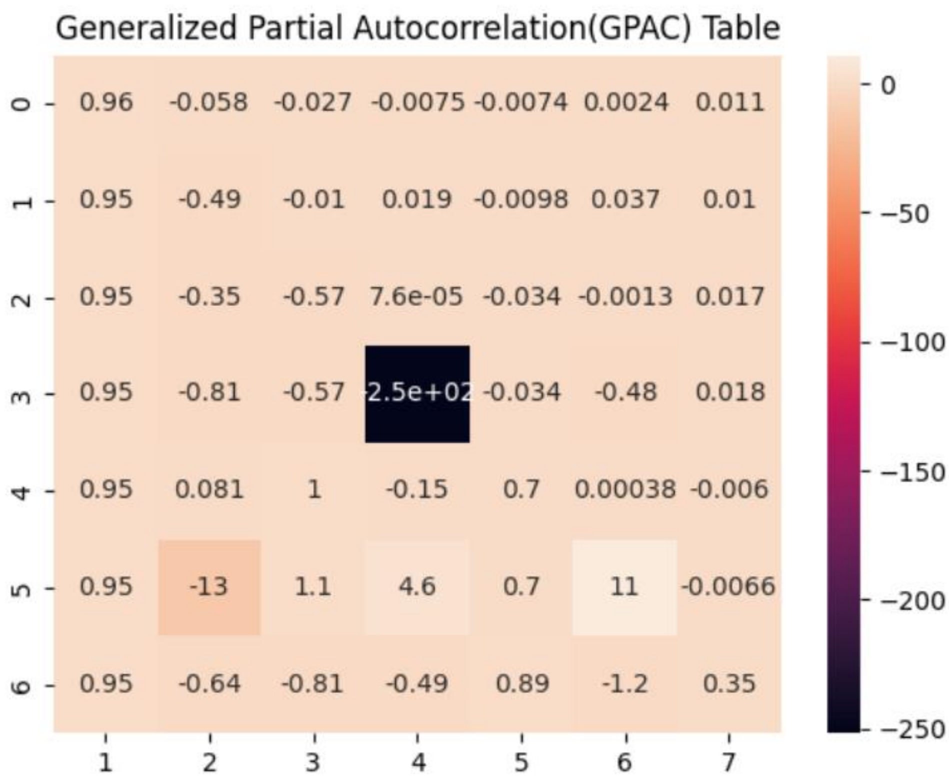
12.f
Mean of residuals: -0.058730578681652205
Variance of residuals: 7310.925167306343

13. ARMA model

ACF/PACF plot


Autocorrelation
Partial Autocorrelation

GPAC Table


Generalized Partial Autocorrelation(GPAC) Table
0.96 -0.058 -0.027 -0.0075 -0.0074 0.0024 0.011
0.95 -0.49 -0.01 0.019 -0.0098 0.037 0.01
0.95 -0.35 -0.57 7.6e-05 -0.034 -0.0013 0.017
0.95 -0.81 -0.57 -2.5e+02 -0.034 -0.48 0.018
0.95 0.081 1 -0.15 0.7 0.00038 -0.006
0.95 -13 1.1 4.6 0.7 11 -0.0066
0.95 -0.64 -0.81 -0.49 0.89 -1.2 0.35

The ACF/PACF plot and the GPAC table both suggest our dataset can be represented by an ARMA(1,0) model.

14. LM Algorithm

The estimated parameter is a1 = -0.97989317.

15. Diagnostic Analysis

15.a
The confidence interval of the parameter a1 is [-48.12140612234565, 46.1616197823834].

Zeros are: []
Poles are: [0.97989317]

The Q value is 35466.22.
The residual is not white.

15.b
Variance of residual error is 690.5107221459118.
Covariance of parameters: 555.580560859488

15.c
Since the variance of error is large, the model is biased.

15.d
Variance of forecast error is 535.7599859025235.

The variance of forecast error is smaller than residual error, meaning our model fits better on the test set.

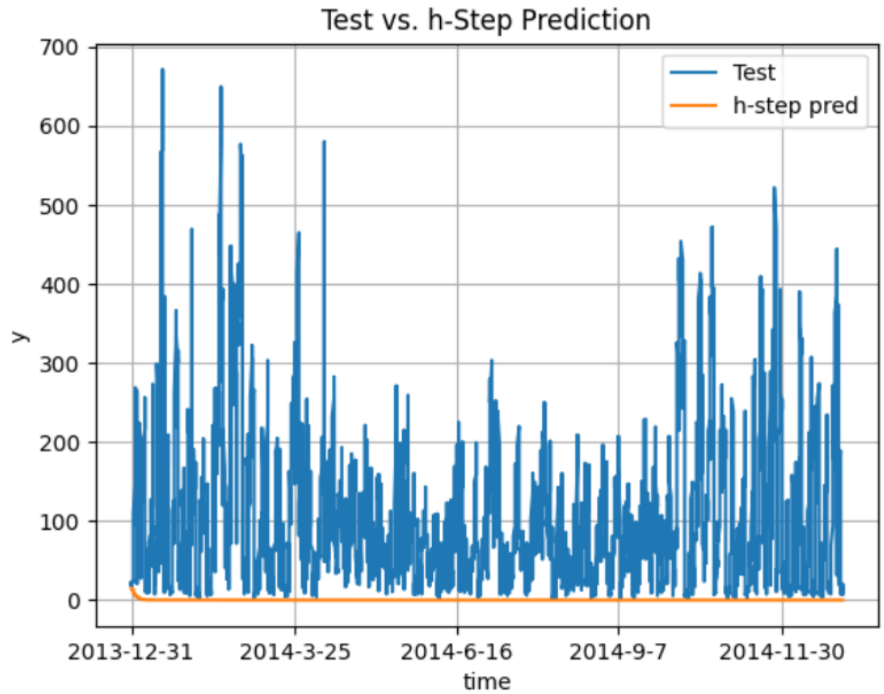
17. Model Selection

We chose ARMA(1,0): y(t) - .9799*y(t-1) = e(t) as our final model.

18. Forecast function

y_hat(t+1|t) = .9799*y(t)

19. h-step ahead prediction



Summary

Although we find the best model to be ARMA(1,0): y(t) – .9799*y(t-1) = e(t), we can see that a simple model as that does not perform very well on our test set. It also has many drawbacks such as being biased and non-white residuals. To reach higher accuracy we would like to explore more options such as deep learning model and a combination of our base models and ARMA model.

Appendix

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 6
# 6.a Preprocess
df = pd.read_csv('BeijingPM2.5.csv',header=0,index_col=0)
med = df['pm2.5'].median()
# Fill in the nan's with the median of the column
df = df.fillna(value=med)
df['date'] = df['year'].astype(str) + '-' \
             + df['month'].astype(str) + '-' \
             + df['day'].astype(str)
df = df.drop(columns=['year','month','day','hour'])

df = df.set_index(['date'])

# 6.b Plot dependent variable
y = df['pm2.5']
```

```
t = df.index

plt.figure()
plt.plot(t, y, label = 'PM2.5')
plt.title('PM2.5 of US Embassy in Beijing')
plt.xlabel('Time')
plt.xticks(t[::10000])
plt.ylabel('PM2.5')
plt.legend()
plt.show()

# 6.c ACF/PACF
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
def ACF_PACF_Plot(y,lags):
    acf = sm.tsa.stattools.acf(y,nlags=lags)
    pacf = sm.tsa.stattools.pacf(y,nlags=lags)
    fig = plt.figure()
    plt.subplot(211)
    plt.title('ACF/PACF of the raw data')
    plot_acf(y,ax=plt.gca(),lags=lags)
    plt.subplot(212)
    plot_pacf(y,ax=plt.gca(),lags=lags)
    fig.tight_layout(pad=3)
    plt.show()

ACF_PACF_Plot(y, 40)

# 6.d Correlation Matrix
import seaborn as sns
sns.heatmap(df.corr(),vmin=-1, vmax=1, annot=True)
plt.show()

# 6.e Split the data
# Remove categorical feature 'cbwd' and features with high correlation
coefficients
X = df.drop(['pm2.5', 'cbwd'],axis=1)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False,
test_size=0.2)

# 7
# ADF Test
from statsmodels.tsa.stattools import adfuller
def ADF_Cal(x):
    result = adfuller(x)
    print("ADF Statistic: %f" %result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

ADF_Cal(y)

# KPSS Test
```

```python
from statsmodels.tsa.stattools import kpss
def kpss_test(timeseries):
    print ('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c', nlags="auto")
    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(y)

# p-value for ADF < 0.05 and > 0.05 for kpss, so the dataset is stationary

# Rolling mean, rolling variance
def Cal_rolling_mean_var(y):
    a=[]
    b=[]
    for i in range(1,len(y)+1):
        roll_mean = np.mean(y[:i])
        roll_var = np.var(y[:i])
        a.append(roll_mean)
        b.append(roll_var)
    return a, b

roll_mean, roll_var = Cal_rolling_mean_var(y)

def Roll_Mean_Var_Plot(roll_mean, roll_var):
    plt.figure()
    plt.plot(roll_mean, label='RollingMean')
    plt.legend()
    plt.title('Rolling Mean')
    plt.xlabel('Samples')
    plt.ylabel('RollingMean')
    plt.grid()
    plt.show()

    plt.figure()
    plt.plot(roll_var, label='RollingVar')
    plt.legend()
    plt.title('Rolling Variance')
    plt.xlabel('Samples')
    plt.ylabel('RollingVar')
    plt.grid()
    plt.show()

Roll_Mean_Var_Plot(roll_mean, roll_var)

# 8
from statsmodels.tsa.seasonal import STL
y = pd.Series(np.array(y), index=pd.date_range('2010-1-1', periods=len(y),
freq='24h'),\
                 name='Hourly PM2.5')
STL = STL(y)
res = STL.fit()
fig = res.plot()
plt.show()
```

```python
T = res.trend
S = res.seasonal
R = res.resid

adj_seasonal = y - S
detrended_y = y - T

# strength of seasonality
F = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(S)+np.array(R)))
print(f'Strength of seasonality = {F}')

# strength of trend
F2 = np.maximum(0,1-np.var(np.array(R))/np.var(np.array(T)+np.array(R)))
print(f'Strength of trend = {F2}')

# 9 Holt-Winter
import statsmodels.tsa.holtwinters as ets
holt_t = ets.ExponentialSmoothing(y_train, trend='add', damped_trend = True,
seasonal=None).fit()
holt_f = holt_t.forecast(steps = len(y_test))
holt_f = pd.DataFrame(holt_f).set_index(y_test.index)

# 10 Feature selection
# Single value decomposition
H = np.matmul(X.T, X)

import numpy.linalg as la
s,d,v = la.svd(H)
print("Singular Values =", d)

# Condition number
kappa = la.cond(X)
print("Conditional Number =", kappa)

LSE = np.matmul(np.matmul(la.inv(np.matmul(X_train.T, X_train)), X_train.T),
y_train)
print("LSE =", LSE)

import statsmodels.api as sm
model = sm.OLS(y_train, X_train).fit()
print(model.summary())

# Backward stepwise reduction

X_train = X_train.drop(['DEWP'],1)
model = sm.OLS(y_train, X_train).fit()
print(model.summary())

X_train = X_train.drop(['TEMP'],1)
model = sm.OLS(y_train, X_train).fit()
print(model.summary())

X_train = X_train.drop(['PRES'],1)
model = sm.OLS(y_train, X_train).fit()
print(model.summary())
```

```python
X_train = X_train.drop(['Iws'],1)
model = sm.OLS(y_train, X_train).fit()
print(model.summary())

X_train = X_train.drop(['Is'],1)
model = sm.OLS(y_train, X_train).fit()
print(model.summary())

# 11 Base models
# Average
yt_avg = []
yt_avg.append(np.nan)
for i in range(1, len(y_train)):
    val = sum(y_train[:i])/i
    yt_avg.append(val)

e_avg = [np.nan]
for i in range(1, len(y_train)):
    err = y_train[i] - yt_avg[i]
    e_avg.append(err)

print('Variance of Average residual error:', np.var(e_avg[1:]))

num = sum(y_train)/len(y_train)
yf_avg = [num]*len(y_test)

ef_avg = []
for i in range(len(y_test)):
    err = y_test[i]-yf_avg[i]
    ef_avg.append(err)

print('Variance of Average forecast error:', np.var(ef_avg))

# Naive
yt_naive = []
yt_naive.append(np.nan)
for i in range(1, len(y_train)):
    val = y_train[i-1]
    yt_naive.append(val)

e_naive = [np.nan]
for i in range(1, len(y_train)):
    err = y_train[i] - yt_naive[i]
    e_naive.append(err)

print('Variance of Naive residual error:', np.var(e_naive[1:]))

yf_naive = [y_train[-1]]*len(y_test)

ef_naive = []
for i in range(len(y_test)):
    err = y_test[i]-yf_naive[i]
    ef_naive.append(err)

print('Variance of Naive forecast error:', np.var(ef_naive))

# Drift
```

```python
yt_drift = [np.nan, np.nan]
for i in range(2, len(y_train)):
    val = (y_train[i-1] - y_train[0])/(i-1) + y_train[i-1]
    yt_drift.append(val)

e_drift = [np.nan, np.nan]
for i in range(2, len(y_train)):
    err = y_train[i] - yt_drift[i]
    e_drift.append(err)

print('Variance of Drift residual error:', np.var(e_drift[2:]))

yf_drift=[]
for i in range(1, len(y_test)+1):
    val = (y_train[-1] - y_train[0])/(len(y_train)-1)*i + y_train[-1]
    yf_drift.append(val)

ef_drift = []
for i in range(len(y_test)):
    err = y_test[i]-yf_drift[i]
    ef_drift.append(err)

print('Variance of Drift forecast error:', np.var(ef_drift))

# SES
# Define alpha = 0.5
alpha = 0.5
yt_ses = [np.nan, y_train[0]]
for i in range(2, len(y_train)):
    val = y_train[i-1]*alpha + yt_ses[i-1]*(1-alpha)
    yt_ses.append(val)

e_ses = [np.nan]

for i in range(1,len(y_train)):
    err = y_train[i] - yt_ses[i]
    e_ses.append(err)

print('Variance of SES residual error:', np.var(e_ses[1:]))

num = y_train[-1]*alpha + yt_ses[-1]*(1-alpha)
yf_ses = [num]*len(y_test)

ef_ses = []
for i in range(len(y_test)):
    err = y_test[i]-yf_ses[i]
    ef_ses.append(err)

print('Variance of SES forecast error:', np.var(ef_ses))

plt.figure()
plt.plot(y_test.index, yf_avg, label='Average')
plt.plot(y_test.index, yf_naive, label='Naive')
plt.plot(y_test.index, yf_drift, label='Drift')
plt.plot(y_test.index, yf_ses, label='SES')
plt.legend()
plt.title('Hourly PM2.5')
```

```
plt.xticks(y_test.index[::2000])
plt.xlabel('time')
plt.ylabel('y')
plt.show()

# 12 Multiple linear regression
X = df.drop(['pm2.5', 'TEMP', 'cbwd'],axis=1)
X_train, X_test = train_test_split(X, shuffle=False, test_size=0.2)

# 12.b Hypothesis test
# 12.c AIC, BIC, Adjusted R2
model = sm.OLS(y_train, X_train).fit()
print(model.summary())

# 12.a One-step prediction
predictions = model.predict(X_test)
y_pred = pd.DataFrame(predictions).set_index(y_test.index)

predictions = model.predict(X_test)
y_pred = pd.DataFrame(predictions).set_index(y_test.index)
plt.plot(y_test, label='Test')
plt.plot(y_pred, label='Prediction')
plt.legend()
plt.ylabel('Price')
plt.xlabel('Index')
plt.xticks(y_test.index[::2000])
plt.title('Hourly PM2.5 Test vs OLS Prediction')
plt.show()

# 12.d Residual analysis
z =
pd.Series(.9061*X_train['DEWP']+.1037*X_train['PRES']-.3578*X_train['Iws']\
                 +3.6242*X_train['Is']-4.1712*X_train['Ir'])
e_pred = y_train-z

# ACF
def ACF_cal(y, lag):
    y_bar = sum(y)/len(y)
    num=0
    denom = 0
    for i in range(lag, len(y)):
        num += (y[i]-y_bar)*(y[i-lag]-y_bar)
    for j in range(len(y)):
        denom += (y[j]-y_bar)**2
    r = num / denom
    return r

l1=[]
for i in range(21):
    l1.append(ACF_cal(e_pred.values, i))
L1 = l1[::-1][:-1] + l1
x1 = np.linspace(-20, 20, 41)

plt.title("Autocorrelation Plot of Residuals")
plt.xlabel("Lags")
plt.stem(x1, L1)
m = 1.96 / np.sqrt(len(e_pred))
```

```python
plt.axhspan(-m, m, alpha=.1, color='pink')
plt.show()

# 12.e Q-value
lbvalue, pvalue = sm.stats.acorr_ljungbox(e_pred.values,lags=[20])
print('Q-value of residuals:', lbvalue)
print(pvalue)

# 12.f Mean & Variance of residuals
print('Mean of residuals:', np.mean(e_pred))
print('Variance of residuals:', np.var(e_pred))

# 13

ACF_PACF_Plot(y_train, 40)

# GPAC
def GPAC_Cal(p,ry):
#np.zeros(shape=(7, 7))
    gpac = [[0]*p for i in range(p)]
    for k in range(1,p+1):
        M = [[0]*k for i in range(k)]
        for j in range(p):
            # The matrix excluding the last column
            for a in range(k):
                for b in range(0,k-1):
                    M[a][b] = ry[abs(j+a-b)]
            # Fill in the last col of numerator
            for a in range(k):
                M[a][k-1] = ry[abs(j+a+1)]
            num = np.linalg.det(M)
            # Fill in the last col of denominator
            for a in range(k):
                M[a][k-1] = ry[abs(j+a+1-k)]
            den = np.linalg.det(M)
            # Compute psi
            psi = num/den
            if abs(psi)<1e-6:
                psi=0
            gpac[j][k-1]=psi
    return gpac

def Plot_GPAC(y):
    # My GPAC becomes very messy if I use stattools.acf()
    ry = sm.tsa.stattools.acf(y)
    df = pd.DataFrame(GPAC_Cal(7,ry),columns=range(1,8))
    sns.heatmap(df,annot=True)
    plt.title('Generalized Partial Autocorrelation(GPAC) Table')
    plt.show()

Plot_GPAC(y_train)

# The plot suggests ARMA(1,0) or ARMA(1,1)

# ACF
l2=[]
for i in range(21):
```

```python
    l2.append(ACF_cal(y_train, i))
L2 = l2[::-1][:-1] + l2
x2 = np.linspace(-20, 20, 41)

plt.title("Autocorrelation Plot of Train set")
plt.xlabel("Lags")
plt.stem(x2, L2)
m = 1.96 / np.sqrt(len(y_train))
plt.axhspan(-m, m, alpha=.1, color='pink')
plt.show()

# 14 Levenberg Marquardt Algorithm
from scipy import signal

def Cal_e(theta,y,na):
    den = np.r_[1,theta[:na].flatten()].tolist()
    num = np.r_[1,theta[na:].flatten()].tolist()
    if len(den) > len(num):
        while len(den) > len(num):
            num.append(0.0)
    elif len(den) < len(num):
        while len(den) < len(num):
            den.append(0.0)
    sys = (den,num,1)
    _,e = signal.dlsim(sys, y)
    return e.flatten()

def grad_desc(theta,y,na,n,delta):
    X = np.array([])
    for i in range(n):
        theta_new = theta.copy()
        theta_new[i] = theta[i] + delta
        e = Cal_e(theta,y,na)
        e_new = Cal_e(theta_new,y,na)
        x = (e-e_new) / delta
        X = np.append(X,x.T)
    X = X.reshape(n,len(y))
    A = np.dot(X, X.T)
    g = np.dot(X, e)
    return X, A, g

def SSE_Cal(e):
    return np.matmul(e.T, e)

def delta_theta(A,g,n,u):
    l = np.identity(n)
    theta_change = np.matmul(np.linalg.inv(A + u * l), g)
    return theta_change

# The following code applies for ARMA(1,0)

def LM_algorithm(y,na,nb):
    miu_max = 10 ** 6
    miu = 0.01
    ite_max = 50
    N = 10000
    n = na + nb
```

```
    delta = 10 ** (-5)
    epsilon = 10**(-3)
    theta = np.zeros(shape=(n, 1))
    k = 0
    SSE_list = []
    while k < ite_max:
        # step 1
        e = Cal_e(theta, y, na)
        SSE_old = SSE_Cal(e)
        X, A, g = grad_desc(theta, y, na, n, delta)
        SSE_list.append(SSE_old)
        # step 2
        theta_change = delta_theta(A, g, n, miu)
        theta_new = (theta.flatten() + theta_change.flatten()).reshape(n,1)
        e_new = Cal_e(theta_new, y, na)
        SSE_new = SSE_Cal(e_new)
        # step 3
        if SSE_new < SSE_old:
            theta_norm = np.dot(theta_change.T, theta_change)**(0.5)
            if theta_norm < epsilon:
                theta_hat = theta_new
                var = SSE_new/(N-n)
                cov = np.dot(var, np.linalg.inv(A))
                break
                print(f'Estimated parameters are {theta_hat}.')
            else:
                theta = theta_new
                miu = miu/10
        else:
            miu = miu*10
            if miu > miu_max:
                theta_hat = theta_new
                break
                print(f'Estimated parameters are {theta_hat}.')
                print('Error!')
    k += 1
    if k > ite_max:
        print(f'Estimated parameters are {theta}.')
        print('Error!')
    return theta_new, SSE_list

theta_new, SSE_list= LM_algorithm(y_train,1,0)

print(theta_new)

# 15 Diagnostic Analysis
# Confidence Interval
cov = SSE_list[-1]/(len(y)-1)
high = theta_new[0][0] + 2*(cov**(0.5))
low = theta_new[0][0] - 2*(cov**(0.5))
print(f'The confidence interval of the parameter is {[low,high]}.')

# Zero/pole cancellation
def zero_pole_cancellation(theta,na):
    den = np.r_[1, theta[:na].flatten()].tolist()
    num = np.r_[1, theta[na:].flatten()].tolist()
    zeros = np.roots(num)
```

```python
    poles = np.roots(den)
    print('Zeros are: ', zeros)
    print('Poles are: ', poles)

zero_pole_cancellation(theta_new, 1)

def ARMA_prediction(theta, y, na, nb):
    e = Cal_e(theta, y, na)
    y_hat = []
    for k in range(1,len(y)):
        AR = []
        MA = []
        for i in range(na):
            AR.append(theta[i]*y[k-i-1])
        for j in range(nb):
            MA.append(theta[j+na]*e[k-j-1])
        y_hat.extend(sum(MA) - sum(AR))
    return y_hat

y_hat = ARMA_prediction(theta_new, y_train, 1, 0)

# Chi2 test
def Q_value(y,y_hat,lags):
    residual_error = y[1:] - y_hat
    acf = sm.tsa.acf(residual_error)
    Q = len(y)*np.sum(np.square(acf[:lags+1]))
    print(f'The Q value is {np.round(Q, 2)}.')
    return Q,residual_error

Q, residual_error = Q_value(y_train, y_hat, 20)

from scipy.stats import chi2
def chi2_test(Q,na,nb):
    dof = 20 - na - nb
    alfa = 0.01
    chi_critical = chi2.ppf(1 - alfa, dof)
    if Q < chi_critical:
        print('The residual is white.')
    if Q > chi_critical:
        print('The residual is not white.')

chi2_test(Q, 1, 0)

# 15.b
# Variance
print(f'Variance of residual error is {np.var(residual_error)}.')

# Covariance
print('Covariance of parameters:', cov)

# 15.c
# The model is biased.

# 15.d
# Variance of residuals vs forecast errors
# Variance of forecast errors
yf_hat = ARMA_prediction(theta_new, y_test, 1, 0)
```

```python
forecast_error = y_test[1:] - yf_hat
print(f'Variance of forecast error is {np.var(forecast_error)}.')

# 16 LSTM

# 17 Model selection

# 18 Forecast function

# One-step prediction function
y_hat_t_1 = []
for i in range(len(y_train)):
    yt = .9799*y_train[i]
    y_hat_t_1.append(yt)

et_arma = []
for i in range(len(y_hat_t_1)):
    err = y_train[i] - y_hat_t_1[i]
    et_arma.append(err)

print('Variance of ARMA residual error:', np.var(et_arma))

# h-step prediction function
y_hat_t_h = []
for h in range(len(y_test)):
    if h == 0:
        yh = .9799*y_train[-1]
        y_hat_t_h.append(yh)
    else:
        yh = .9799*y_hat_t_h[h-1]
        y_hat_t_h.append(yh)

ef_arma = []
for i in range(len(y_hat_t_h)):
    err = y_test[i] - y_hat_t_h[i]
    ef_arma.append(err)

print('Variance of ARMA forecast error:', np.var(ef_arma))

# 19

y_hat_t_h = pd.DataFrame(y_hat_t_h).set_index(y_test.index)

plt.figure()
plt.plot(y_test, label = 'Test')
plt.plot(y_hat_t_h, label = 'h-step pred')
plt.legend()
plt.title('Test vs. h-Step Prediction')
plt.xlabel('time')
plt.xticks(y_test.index[::2000])
plt.ylabel('y')
plt.grid()
plt.show()

# The GPAC table also suggests our model could be ARMA(1,1)
# The following code applies for ARAM(1,1)
def LM_algorithm(y,na,nb):
```

```python
    miu_max = 10 ** 6
    miu = 0.01
    ite_max = 50
    N = 10000
    n = na + nb
    delta = 10 ** (-5)
    epsilon = 10**(-3)
    theta = np.zeros(shape=(n, 1))
    k = 0
    SSE_list = []

    while k < ite_max:
        # step 1
        e = Cal_e(theta, y, na)
        SSE_old = SSE_Cal(e)
        X, A, g = grad_desc(theta, y, na, n, delta)
        SSE_list.append(SSE_old)
        # step 2
        theta_change = delta_theta(A, g, n, miu)
        theta_new = (theta.flatten() + theta_change.flatten()).reshape(n,1)
        e_new = Cal_e(theta_new, y, na)
        SSE_new = SSE_Cal(e_new)
        # step 3
        if SSE_new < SSE_old:
            theta_norm = np.dot(theta_change.T, theta_change)**(0.5)
            if theta_norm < epsilon:
                theta_hat = theta_new
                var = SSE_new/(N-n)
                cov = np.dot(var, np.linalg.inv(A))
                break
                print(f'Estimated parameters are {theta_hat}.')
            else:
                theta = theta_new
                miu = miu/10
        else:
            miu = miu*10
            if miu > miu_max:
                theta_hat = theta_new
                break
                print(f'Estimated parameters are {theta_hat}.')
                print('Error!')
    k += 1
    if k > ite_max:
        print(f'Estimated parameters are {theta}.')
        print('Error!')
    return theta_new, cov, SSE_list

theta_new, cov, SSE_list= LM_algorithm(y_train,1,1)
print(theta_new)

def CI_cal(theta, na, nb, cov):
    n = na + nb
    for i in range(n):
        high = theta[i] + 2*(cov[i][i]**(0.5))
        low = theta[i] - 2*(cov[i][i]**(0.5))
        print(f'The confidence interval of {i+1}th parameter is
{[low[0],high[0]]}.')
```

```python
CI_cal(theta_new, 1, 1, cov)

zero_pole_cancellation(theta_new, 1)

y_hat1 = ARMA_prediction(theta_new, y_train, 1, 1)

Q1, residual_error1 = Q_value(y_train, y_hat1, 20)

chi2_test(Q1, 1, 1)

# Variance
print(f'Variance of residual error is {np.var(residual_error)}.')

# Covariance
print('Covariance of parameters:', cov)

# The model is biased.

# Variance of residuals vs forecast errors
# Variance of forecast errors
yf_hat1 = ARMA_prediction(theta_new, y_test, 1, 1)
forecast_error = y_test[1:] - yf_hat1
print(f'Variance of forecast error is {np.var(forecast_error)}.')

# One-step prediction
y_hat1_t_1 = []
for i in range(len(y_train)):
    if i == 0:
        yt = .978*y_train[i] +.045*y_train[i]
        y_hat1_t_1.append(yt)
    else:
        yt = .978*y_train[i] +.045*(y_train[i] - y_hat1_t_1[i-1])
        y_hat1_t_1.append(yt)

# h-step prediction

y_hat1_t_h = []
for h in range(len(y_test)):
    if h == 0:
        yh = .978*y_train[-1] +.045*(y_train[-1] - y_hat1_t_1[-1])
        y_hat1_t_h.append(yh)
    else:
        yh = .978*y_hat1_t_h[h-1]
        y_hat1_t_h.append(yh)

ef_arma1 = []
for i in range(len(y_hat1_t_h)):
    err = y_test[i] - y_hat1_t_h[i]
    ef_arma1.append(err)

print('Variance of ARMA(1,1) forecast error:', np.var(ef_arma1))

# The result suggests that ARMA(1,1) performs even worse on our dataset
# Hence we would not pick it.
```

Reference

Beijing PM2.5 Data Dataset. https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

README

There is no need for any input when running the python file. All the results in the report can be generated in python console.