THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Air Quality Prediction

Time Series Analysis and Modeling - Final Project

Sisi Zhang

DATS 6450 Dr. Reza Jafari

2021/05/05

Table of content

Abstract

This is the term project of time series analysis and modeling with Dr. Reza Jafari in Spring 2021. The main purpose of this project is to apply the technologies I learned in the class on multivariable time dependent dataset and make effective predictions. The data analysis skills I use in this project will include time series decomposition and feature selection. The data modeling technologies will include base model (average, naïve, drift, Simple Exponential Smoothing), Holt-Winter, Multiple Linear regression and ARMA. All the models motioned before will be applied on the dataset and the final model will be selected based on the comparison of model performance.

Introduction

This project is about the air quality prediction. The dataset is from UCI Machine Learning Repository. It contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensor embedded in an Air Quality Chemical Multisensor Device, which located in a significantly polluted area in Italian city. The data is time dependent, which were record from March 2004 to February 2005. The goal of this project is to use the model I learned to predict the CO (carbon monoxide) amount. I choose CO as my target because it is an important air quality index and it has less missing values than other features.

Description of dataset

The dataset has 9358 observations and 15 attributes, which are listed below: Date, Time, CO(GT), PT08.S1(CO), NMHC(GT), C6H6(GT), PT08.S2(NMHC), NOx(GT), PT08.S3(NOx), NO2(GT), PT08.S4(NO2), PT08.S5, T, RH, AH. The Independent variables: CO(GT) and the rest of them are dependent variables. For data preprocessing, I remove ~~NMHC(GT)~~ from dataset, because it has more than 50 % of the missing values. For other variables I fill the missing value with average. Then split the data into train set (80%) and test set (20%).

Fig 1. Plot of the dependent variable vs time

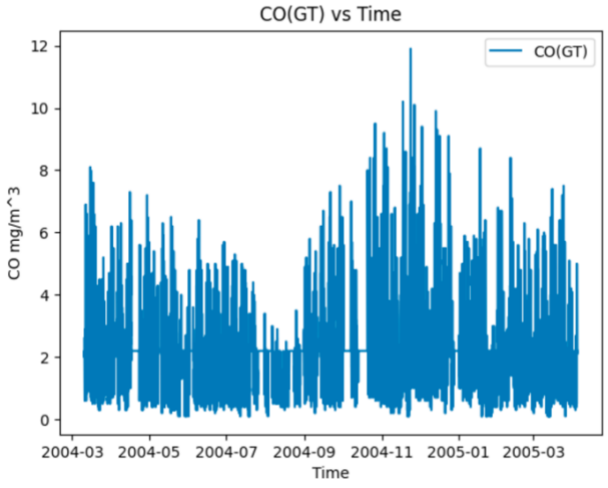
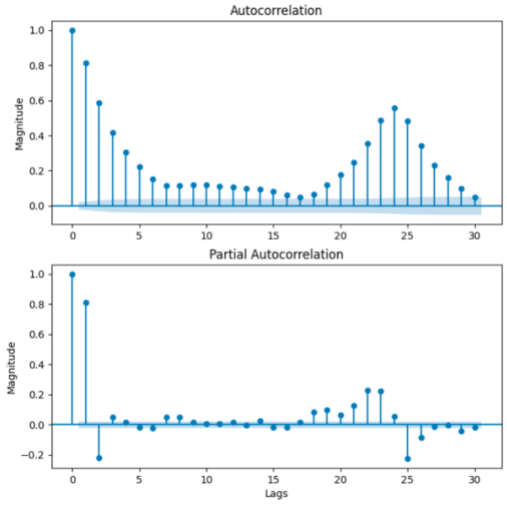
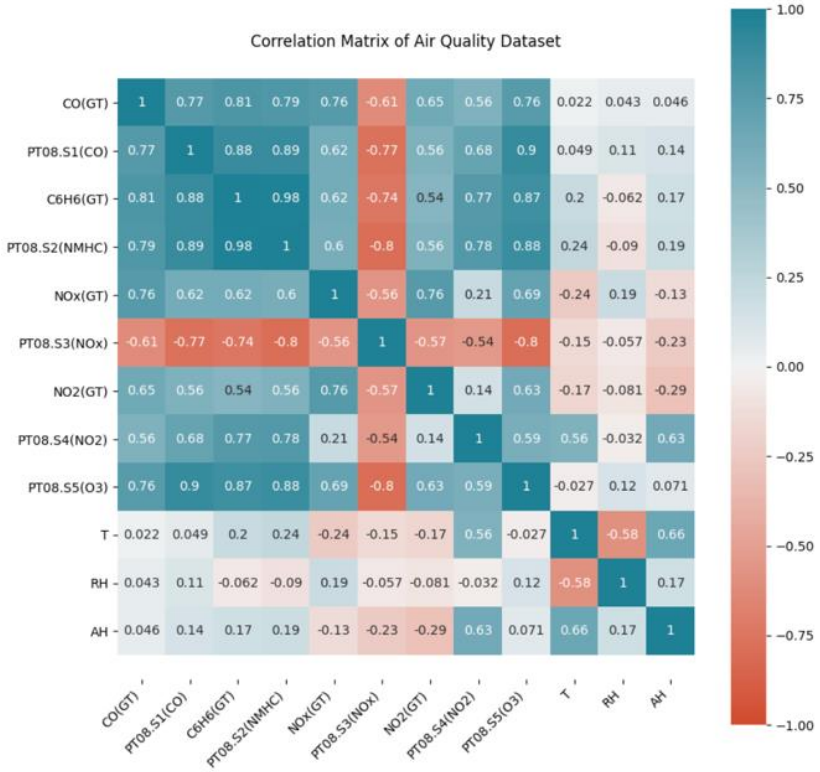


Fig 2. ACF/PACF of the dependent variables



From the Fig 1 and Fig 2, I can see that the data is not white noise, which we can find pattern and build model for prediction.

Fig 3. Correlation Matrix with seaborn heatmap and Pearson's correlation coefficient



From Fig 3. I can see that last three features (T, RH, AH) have weak correlation with target CO(GT). Others have stronger correlation with target. The correlation coefficients are showing in the Fig 3.

Stationarity Check:

Fig 4. Original data -Rolling mean & variance.

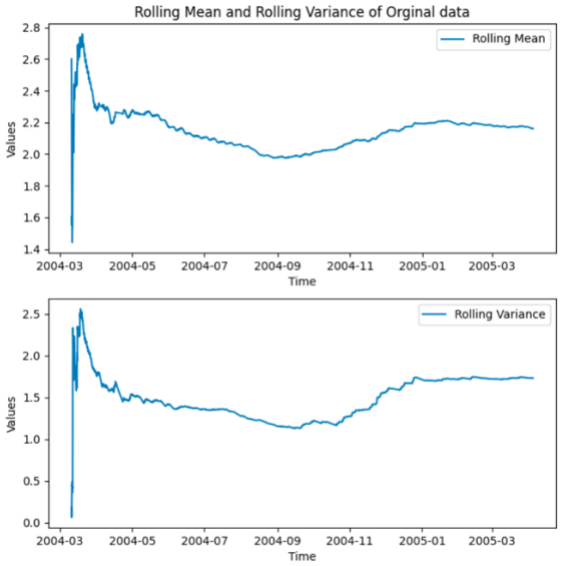
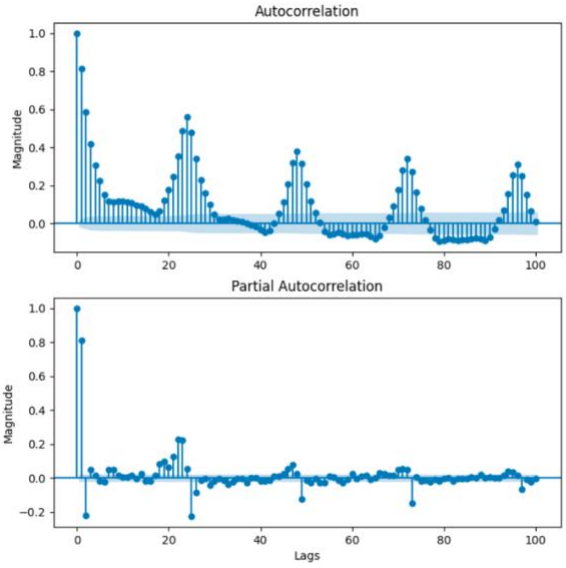


Fig 5. Original data -ACF/PACF




```
ADF Statistic: -11.067815
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

The Fig 4 show that the rolling mean and variance is not significantly changing over time. From the ADF test result, I can see that statistic value is negative and lower than the critical values at 1% level. The p-value is close 0, which indicates the original dataset is stationary. From Fig 5, I can see that the ACF plot is not white noise, which means the original dataset can be modeled.

Differenced data:

Fig 6. Differenced data -Rolling mean & variance.

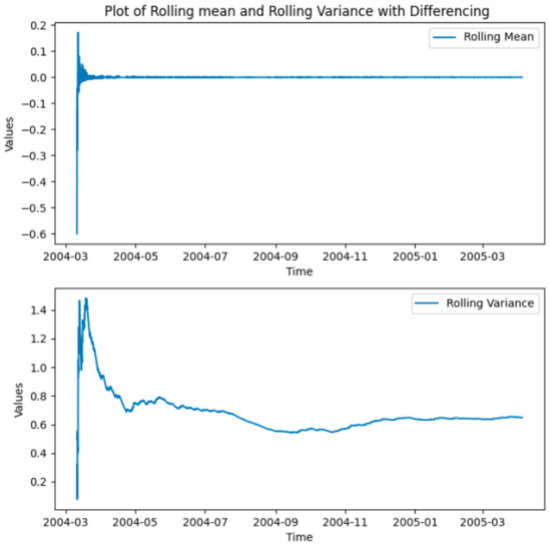
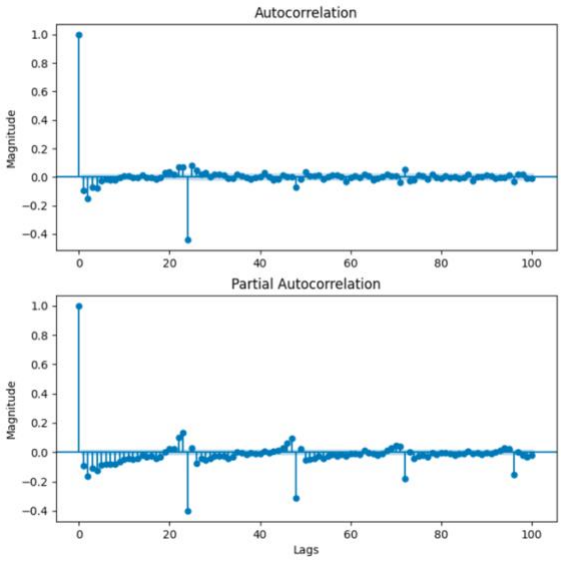


Fig 7. Differenced data -ACF/PACF




```
ADF Statistic: -27.376938
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
```

Let's look at the differenced data, the Fig 6 shows the rolling mean and variance is more like constant before. The ADF statistic value is -27, which is more negative than the original data. It means the differenced data is more stationary. There is still pattern in the ACF/PACF, which means the differenced data can be modelled as well.

Time Series Decomposition

Fig 8.

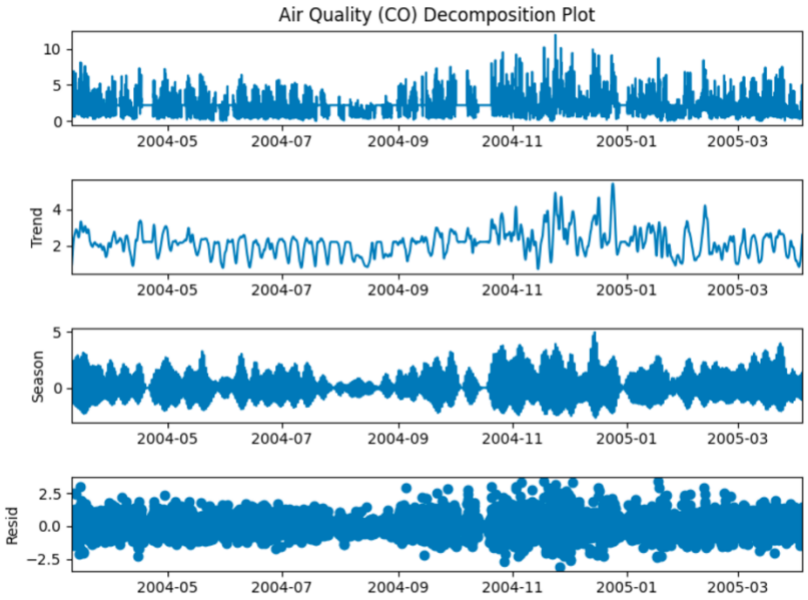


Fig 8 shows the break down of target variable CO, display trend, season and residual separately.

Fig 9.

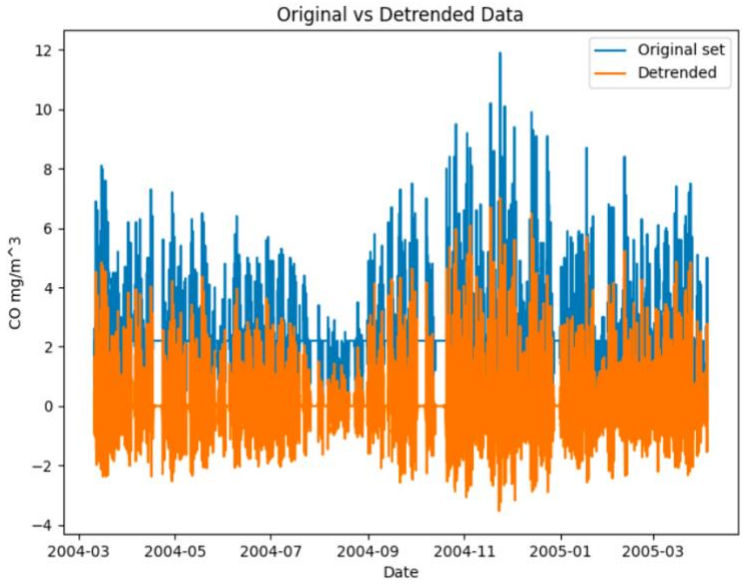


Fig 9 show the original data verses the detrended data. The detrend data has less variance than original one.

Fig 10.

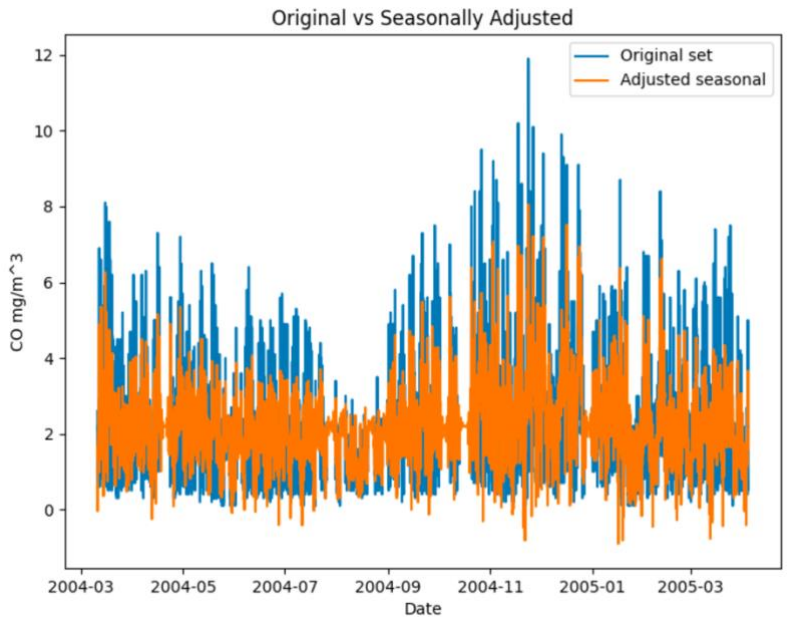


Fig 10 shows the original data verses seasonal adjusted data. The variance of seasonal adjusted data looks smaller than the original one. The seasonal pattern is not very clear.

```
The strength of trend for this dataset is = 0.6232512296157902
The strength of seasonality for this dataset is = 0.7240884422155038
```

The calculation of strength of trend is 0.62 and the strength of seasonality is 0.72, which means the trend and seasonality are both detectable and the data has more seasonality variation than trend. I should implement SARIMA model later.

Feature selection

Fig 11. Before feature selection:

OLS Regression Results

Dep. Variable:	CO(GT)	R-squared:	0.804
Model:	OLS	Adj. R-squared:	0.803
Method:	Least Squares	F-statistic:	2780.
Date:	Mon, 03 May 2021	Prob (F-statistic):	0.00
Time:	14:11:08	Log-Likelihood:	-6526.5
No. Observations:	7485	AIC:	1.308e+04
Df Residuals:	7473	BIC:	1.316e+04
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.3309	0.193	-1.715	0.086	-0.709	0.047
PT08.S1(CO)	0.0013	8.86e-05	14.335	0.000	0.001	0.001
C6H6(GT)	0.0754	0.006	12.287	0.000	0.063	0.087
PT08.S2(NMHC)	-0.0017	0.000	-8.111	0.000	-0.002	-0.001
NOx(GT)	0.0031	7.48e-05	41.303	0.000	0.003	0.003
PT08.S3(NOx)	-0.0002	6.48e-05	-2.923	0.003	-0.000	-6.24e-05
NO2(GT)	0.0051	0.000	15.334	0.000	0.004	0.006
PT08.S4(NO2)	0.0014	6.74e-05	21.254	0.000	0.001	0.002
PT08.S5(O3)	-0.0004	5.31e-05	-7.844	0.000	-0.001	-0.000
T	-0.0152	0.003	-4.718	0.000	-0.022	-0.009
RH	-0.0054	0.001	-4.287	0.000	-0.008	-0.003
AH	-0.2625	0.056	-4.658	0.000	-0.373	-0.152

Omnibus:	888.408	Durbin-Watson:	0.910
Prob(Omnibus):	0.000	Jarque-Bera (JB):	8882.833
Skew:	-0.127	Prob(JB):	0.00
Kurtosis:	8.331	Cond. No.	7.36e+04

```
The condition number for X original is = 73626.9322293961
SingularValues d original is = [4.88304520e+10 1.41902431e+09 3.22554639e+08 1.03381034e+08
 4.51651725e+07 3.77278721e+07 4.18970311e+06 1.38719897e+06
 1.75442552e+05 1.81487920e+04 1.05400936e+02 9.00777093e+00]
```

Fig 11. Shows the model with all 11 features with one constant. The model summary shows the Adjusted R-squared value is 0.803. The condition number is 73636, which is very large and indicate the collinearity exits. And the singular values listed has some relatively small, which indicates at least 4 features should be removed.

The features I removed are listed below with the order I removed: 1, Constant. 2, AH. 3, C6H6(GT). 4, PT08.S2(NMHC). 5, T. 6, PT08.S5(O3). 7, RH. 8, NO2(GT). 9, PT08.S1(CO). I remove the features have large p-value first, then remove the features have high std error. I stop remove features ehen the Adjusted R-squared value is going to drop significantly. This way I can eliminate the features and keep good Adjusted R-squared value at same time.

Fig 12. After feature selection:

OLS Regression Results

Dep. Variable:	CO(GT)	R-squared (uncentered):	0.933
Model:	OLS	Adj. R-squared (uncentered):	0.933
Method:	Least Squares	F-statistic:	3.449e+04
Date:	Mon, 03 May 2021	Prob (F-statistic):	0.00
Time:	14:11:08	Log-Likelihood:	-7577.6
No. Observations:	7485	AIC:	1.516e+04
Df Residuals:	7482	BIC:	1.518e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
NOx(GT)	0.0042	4.34e-05	97.025	0.000	0.004	0.004
PT08.S3(NOx)	-0.0008	2.1e-05	-37.989	0.000	-0.001	-0.001
PT08.S4(NO2)	0.0012	1.47e-05	84.976	0.000	0.001	0.001

Omnibus:	959.922	Durbin-Watson:	0.720
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3063.784
Skew:	0.659	Prob(JB):	0.00
Kurtosis:	5.843	Cond. No.	10.4

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
The condition number for X after feature elimination is = 10.401380397474536
SingularValues d after feature elimination is = [2.38024850e+10 8.86116849e+08 2.20008947e+08]

Fig 12 shows the final features I keep are "NOx(GT)", "PT08.S3(NOx)" and "PT08.S4(NO2)". From the model summary, I can see that all the p-values of coefficients are close to 0, which means they are significant. The condition number is 10.4, it is much smaller than original dataset will all the features, which means there is no collinearity issue. The singular values are large and in similar range, which means no more features need to be removed. The Adjusted R-squared value is 0.933, which means this model has better performance than the one before.

Model Building

Multiple Linear Regression Model:

Fig 13.

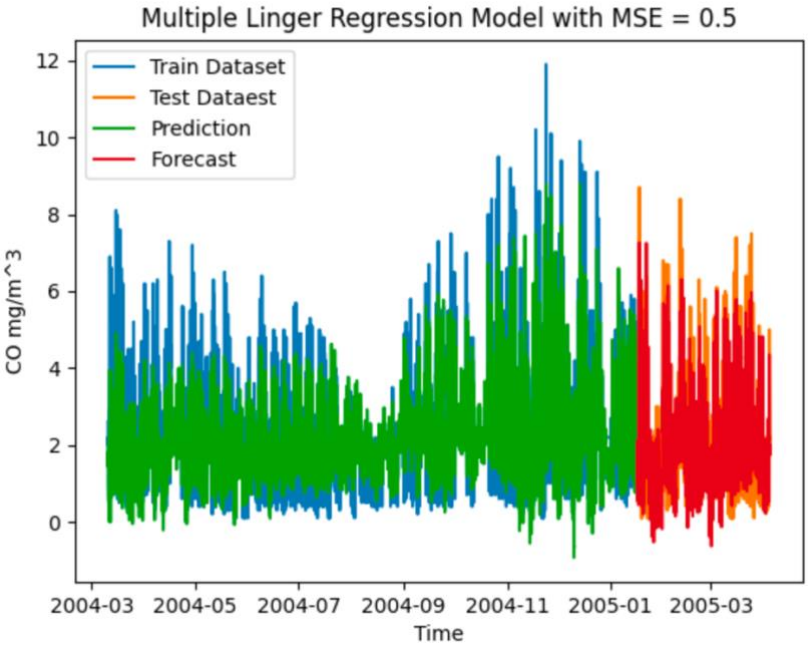


Fig 13 shows the plot of the train set, test set, prediction set and forecast. The prediction seems overlap with train set a lot and the forecast set looks like overlap with test set a lot as well.

```
The p-value of t-test is:  NOx(GT)          0.000000e+00
PT08.S3(NOx)    6.296617e-289
PT08.S4(NO2)     0.000000e+00
dtype: float64
The p-value of f-test is:  0.0
```

From the hypothesis test analysis, we can see that all the p-value of t-test are close to zero, which means they are all important coefficients. The p-value of f-test is close to zero as well, which means the model with features is better than the model that constants only.

```
The AIC value of the model is : 15161.100533661438
The BIC value of the model is : 15181.862502552
The MSE value of residual is : 0.5
The RMSE value of residual is : 0.7071067811865476
The R-squared value of the model is:  0.9325713664749158
The Adjusted R-squared value of the model is:  0.9325443301342883
The Q-value is : 3616.4884809867335
```

This model has large AIC, BIC and Q values. The MSE is 0.5, RMSE is 0.7 and the Adjusted R-squared value is 0.93 which means the model has high accuracy.

Fig 14. ACF of residuals

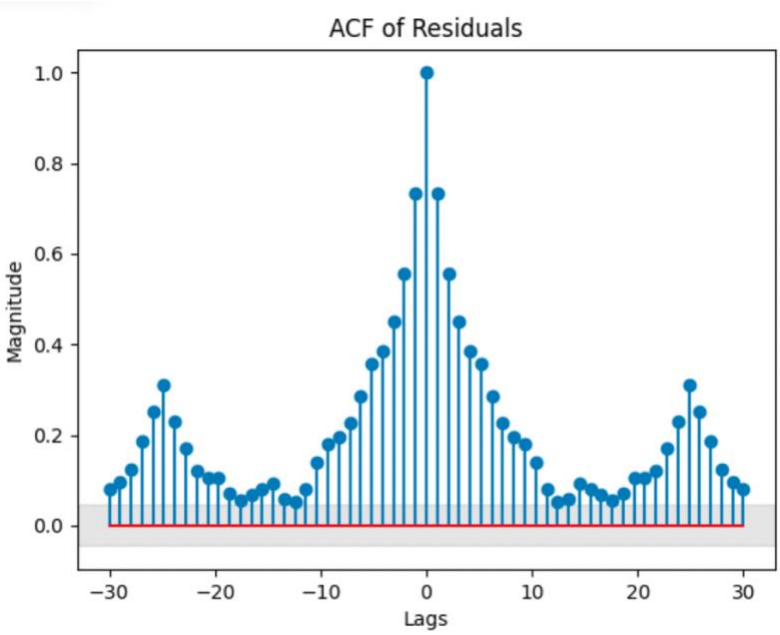
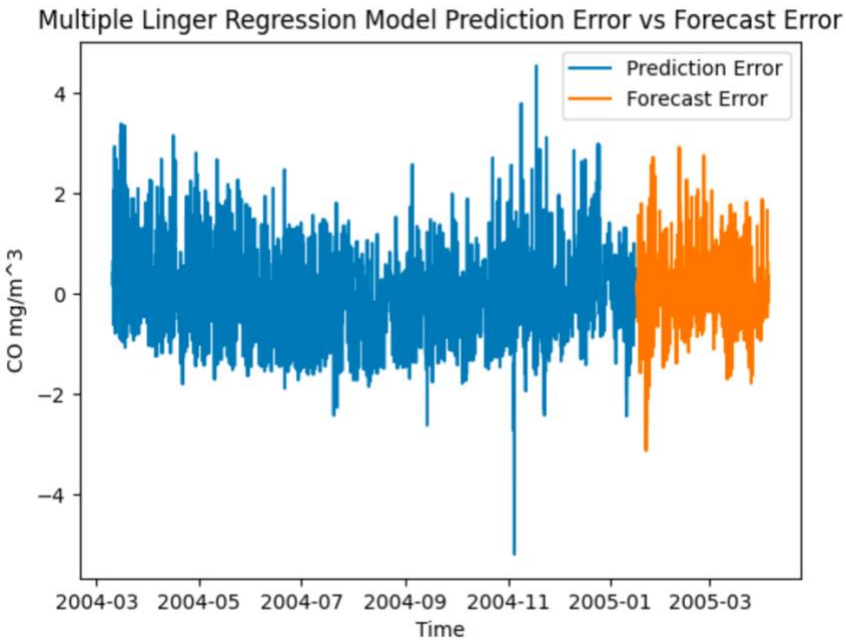


Fig 14 show the residuals are not white noise, which means there is still information left in the dataset, and model can be improve.

Fig 15.




```
The variance of the predict error is : 0.4432929510596945
The mean of the predict error is : -0.012927087426475319
The variance of the forecast error is : 0.49334712262776864
The mean of the forecast error is : -0.08281126932454426
```

Fig 15. Shows the prediction error and forecast error. Two errors look similar and after calculations, the variance and mean of the residuals are close, this indicates a good model.

Average Model

Fig 16.

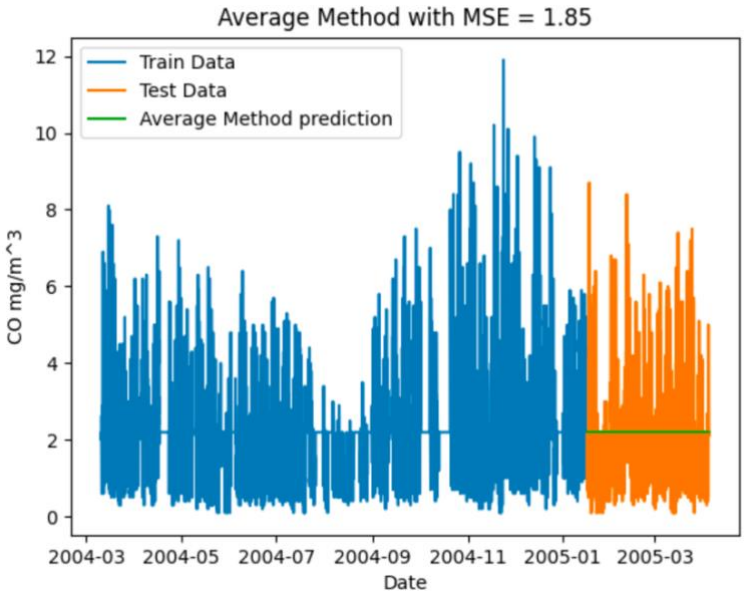
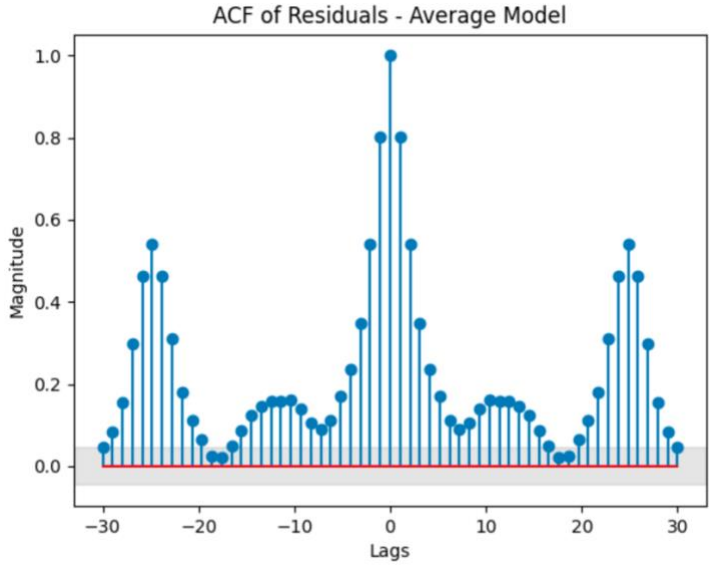


Fig 17.




```
The mean of error of Average Model is : -0.22935649075929648
The variance of Average of Naive Model is : 1.7976151471984805
The MSE of Average Model is : 1.85
The RMSE of Average Model is : 1.3601470508735443
The Q value of Average Model is: 4326.331056251148
```

Fig 16 shows the train set, test set and forecast set generated by average modelling. Fig 17 shows the residuals of average model is not white. There is more information left in the dataset. Most of the metrics are not indicating this is a good model. The MSE is small, it may because the missing value are filled with average.

Naïve Model

Fig 18.

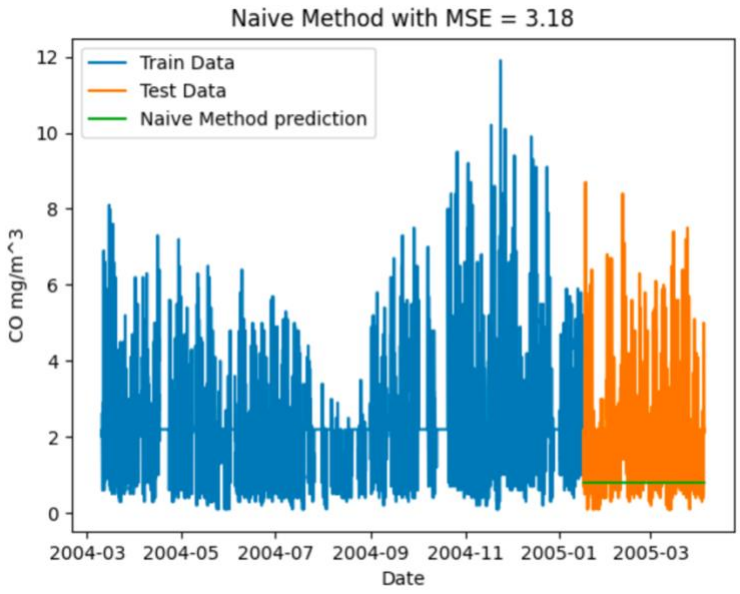
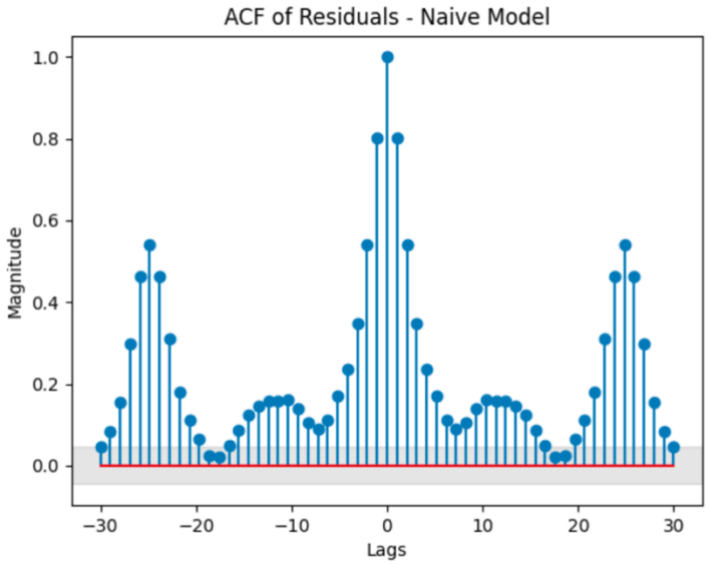


Fig 19.




```
The mean of error of Naive Model is : 1.1777777777777778
The variance of error of Naive Model is : 1.7976151471984805
The MSE of Naive Model is : 3.18
The RMSE of Naive Model is : 1.783255450012701
The Q value of Naive Model is: 4326.331056251148
```

Fig 18 shows the train set, test set, and forecast set generated by naïve model. The forecast set is a green flat line, which cannot capture the variance of the dataset. All the metrics are not good for model evaluations. Fig 19 shows the ACF of residuals are not white, a better model should be developed.

Drift Model

Fig 20.

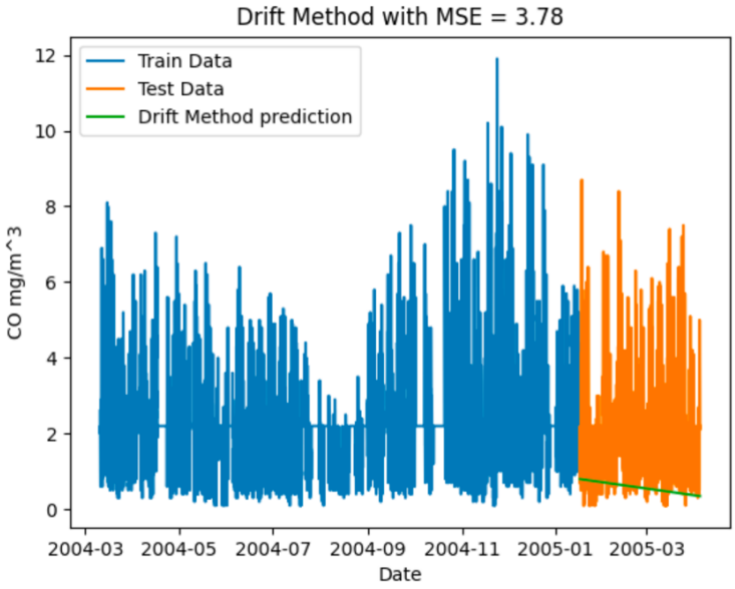
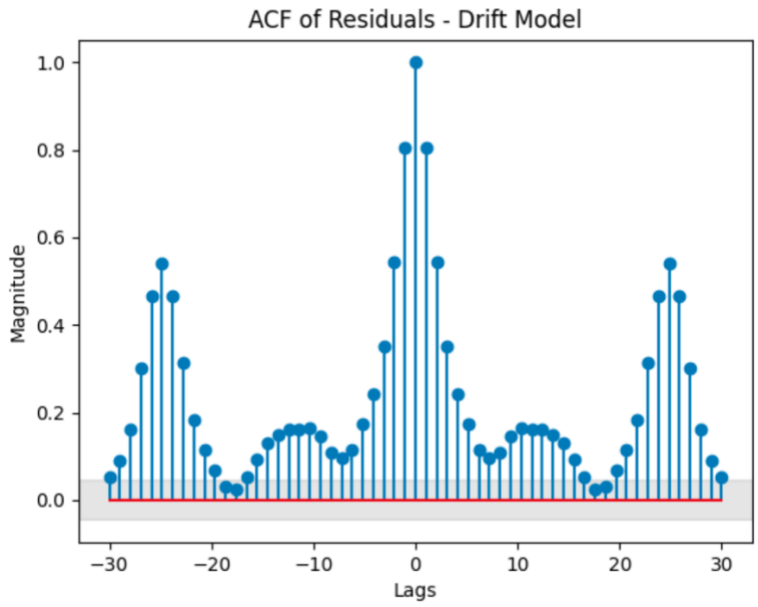


Fig 21.




```
The mean of error of Drift Model is : 1.4030182908723796
The variance of error of Drift Model is : 1.8073779488676824
The MSE of Drift Model is : 3.78
The RMSE of Drift Model is : 1.944222209522358
The Q value of Drift Model is: 4392.906747725301
```

Fig 20 shows the train set, test set, and forecast set generated by drift model. The forecast set is a green flat line that going down, which cannot capture the variance of the dataset. All the metrics are not good for model evaluations. Fig 21 shows the ACF of residuals are not white, a better model should be developed.

Simple and Exponential Smoothing Model

Fig 22.

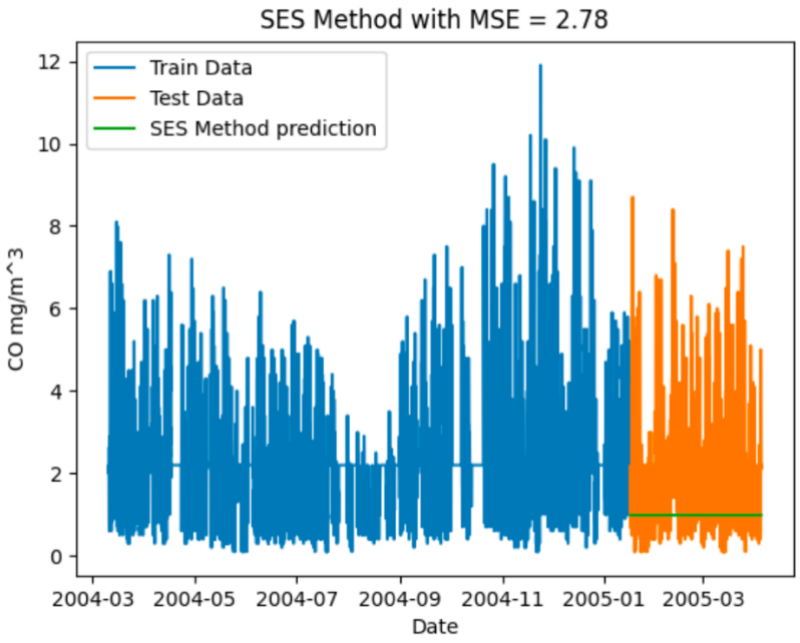
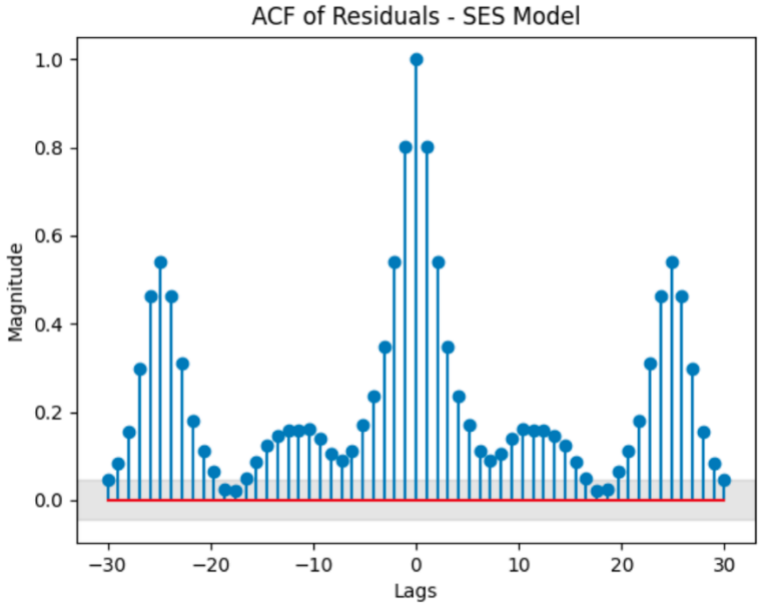


Fig 23.




```
The mean of error of SES Model is : 0.9931098313851702
The variance of error of SES Model is : 1.7976151471984805
The MSE of SES Model is : 2.78
The RMSE of SES Model is : 1.6673332000533065
The Q value of SES Model is: 4326.331056251147
```

Fig 22 shows the train set, test set, and forecast set generated by SES model with alfa = 0.5. The forecast set is a green flat line, which cannot capture the variance of the dataset. The MSE is 2.78 which is better than drift model, but other metrics are not good enough for model evaluations. Fig 23 shows the ACF of residuals are not white, a better model should be developed.

Holt-Winters Model (seasonal_periods=24)

Fig 24.

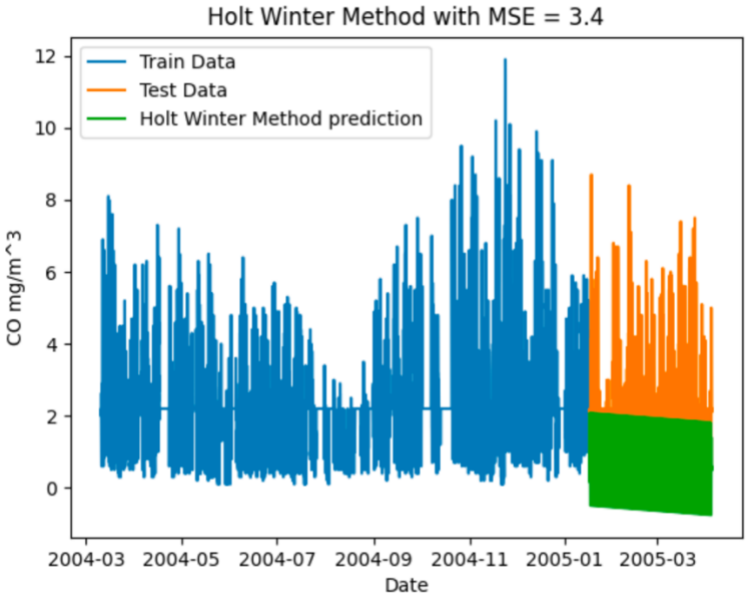
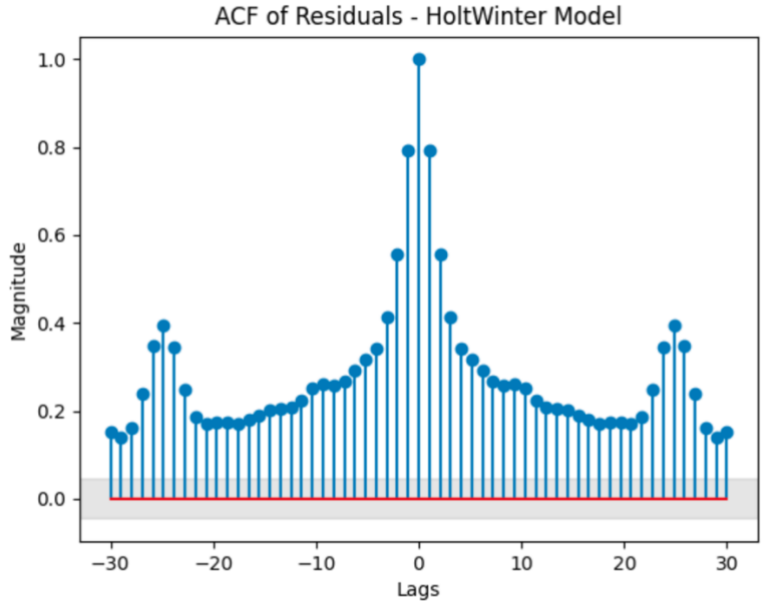


Fig 25.




```
The mean of error of HoltWinter Model is : 1.4263175076231824
The variance of error of HoltWinter Model is : 1.3608247009310142
The MSE of HoltWinter Model is : 3.4
The RMSE of HoltWinter Model is : 1.8439088914585775
The Q value of HoltWinter Model is: 4980.040103929021
```

Fig 24 shows the train set, test set, and forecast set generated by Holt-Winter model with seasonal_periods=24. I use 24 because the dataset is hourly data set with circle 24. The forecast set is the green area, which has seasonality and trend going down. The metrics are not good enough for model evaluations. Fig 25 shows the ACF of residuals are decay slowly and not white noise, a better model should be developed.

Holt-Winters Model (seasonal_periods=12):

Fig 26.

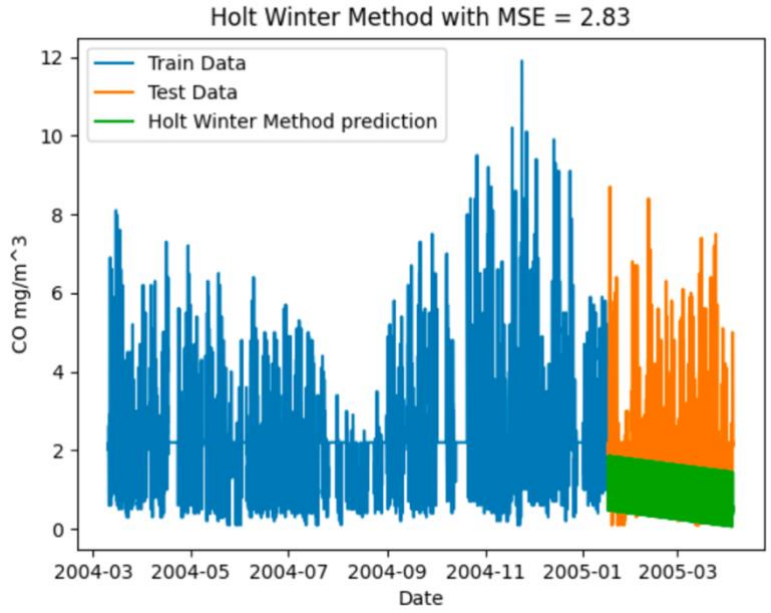
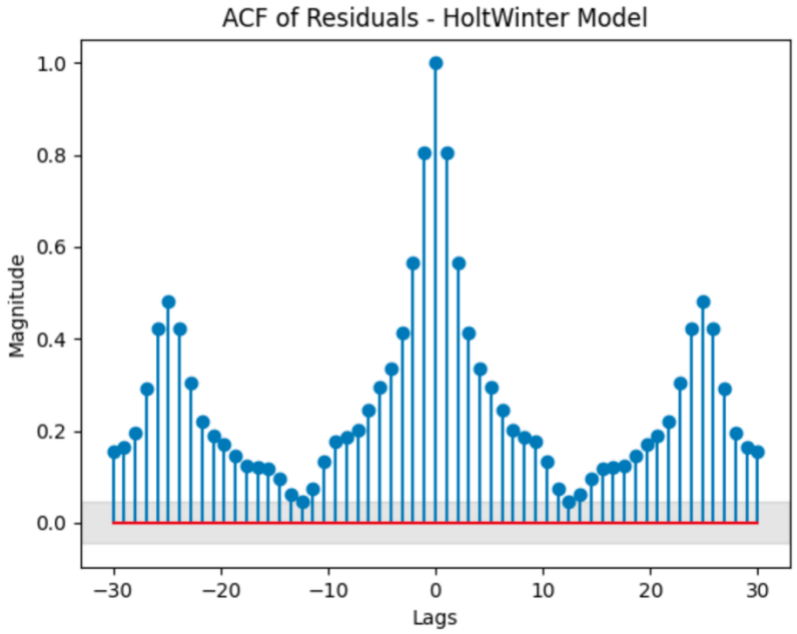


Fig 27.




```
The mean of error of HoltWinter Model is : 1.1161173399614286
The variance of error of HoltWinter Model is : 1.5864615369528228
The MSE of HoltWinter Model is : 2.83
The RMSE of HoltWinter Model is : 1.6822603841260722
The Q value of HoltWinter Model is: 4827.990859502824
```

Fig 26 shows the train set, test set, and forecast set generated by Holt-Winter model with seasonal_periods=12. After I use 24 in the last model, I also try 12 and 48. The 12 one has smaller MSE which is 2.83. The forecast set is the green area, which has seasonality and trend going down. The metrics are not good enough for model evaluations. Fig 27 shows the ACF of residuals are decay slowly and not white noise. It looks better than the model has seasonal_periods=24, but the model can still be improved.

ARMA Model

Fig 28.

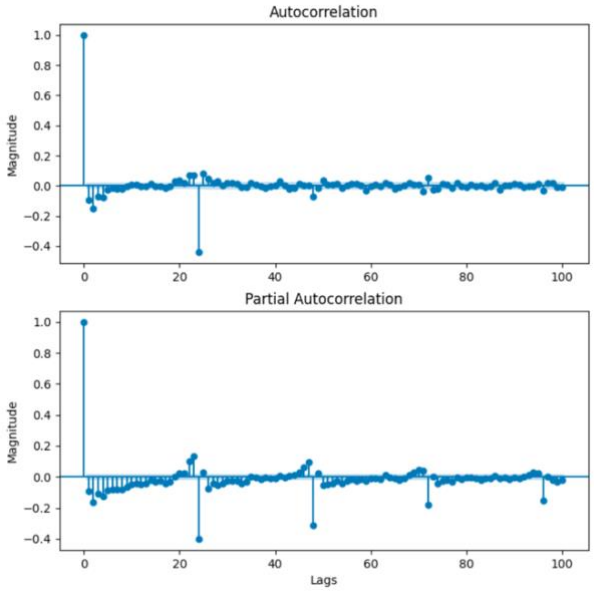
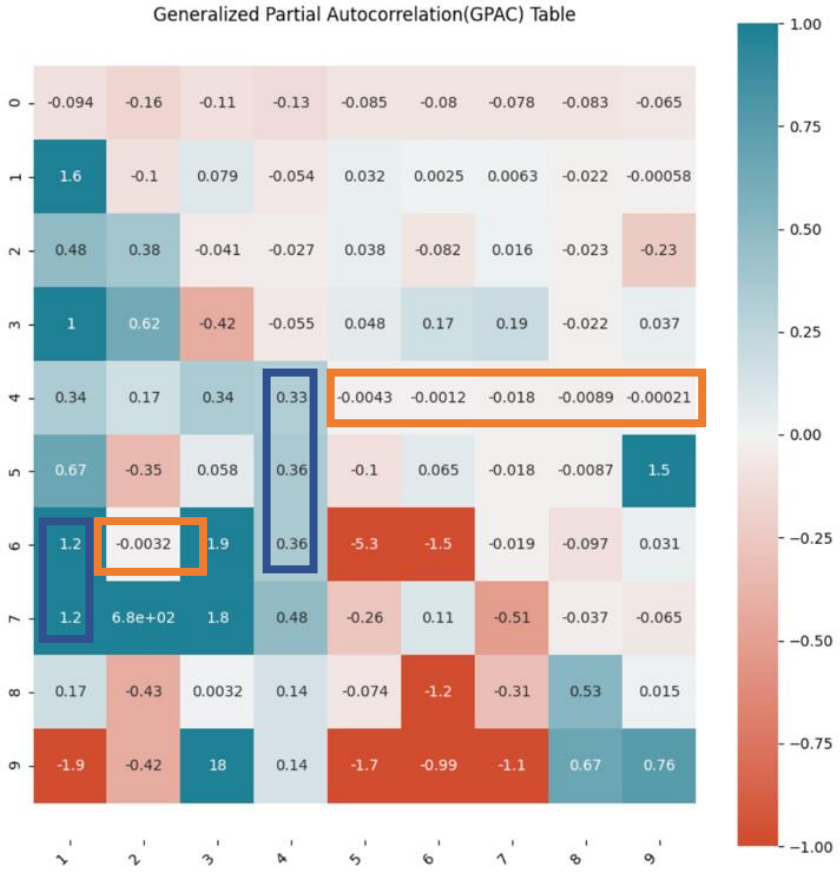


Fig 29.



To build ARMA model, I mainly use Fig 29 GPAC for order determination. I find following ARMA model order could be potentials: ARMA (4,4) and ARMA (1,6)

Fig 30. Estimate ARMA (4,4) model parameters using Levenberg Marquardt algorithm

ARMA Model Results

Dep. Variable:	y	No. Observations:	7485
Model:	ARMA(4, 4)	Log Likelihood	-8356.578
Method:	css-mle	S.D. of innovations	0.739
Date:	Tue, 04 May 2021	AIC	16731.156
Time:	22:35:39	BIC	16793.442
Sample:	0	HQIC	16752.546

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1.y	1.0037	2.44e-05	4.11e+04	0.000	1.004	1.004
ar.L2.y	-0.4016	2.55e-06	-1.57e+05	0.000	-0.402	-0.402
ar.L3.y	1.0244	9.52e-06	1.08e+05	0.000	1.024	1.024
ar.L4.y	-0.6265	2.43e-05	-2.58e+04	0.000	-0.627	-0.626
ma.L1.y	-0.0261	0.011	-2.339	0.019	-0.048	-0.004
ma.L2.y	0.1371	0.005	25.749	0.000	0.127	0.148
ma.L3.y	-0.8606	0.005	-162.583	0.000	-0.871	-0.850
ma.L4.y	-0.2379	0.011	-21.252	0.000	-0.260	-0.216

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.3930	-0.9842j	1.0598	-0.3105
AR.2	-0.3930	+0.9842j	1.0598	0.3105
AR.3	1.0000	-0.0000j	1.0000	-0.0000
AR.4	1.4211	-0.0000j	1.4211	-0.0000
MA.1	1.0038	-0.0000j	1.0038	-0.0000
MA.2	-0.3864	-0.9689j	1.0431	-0.3104
MA.3	-0.3864	+0.9689j	1.0431	0.3104
MA.4	-3.8490	-0.0000j	3.8490	-0.5000

The standard deviation of the parameter estimates is: 0.6545350247696496

Fig 30 shows estimated ARMA(4,4) parameters. The parameter estimates are in the green square, the confidence intervals are in the yellow square. The standard deviation of the parameter estimates is 0.65.

Diagnostic Analysis for ARMA(4,4)

Fig 31.

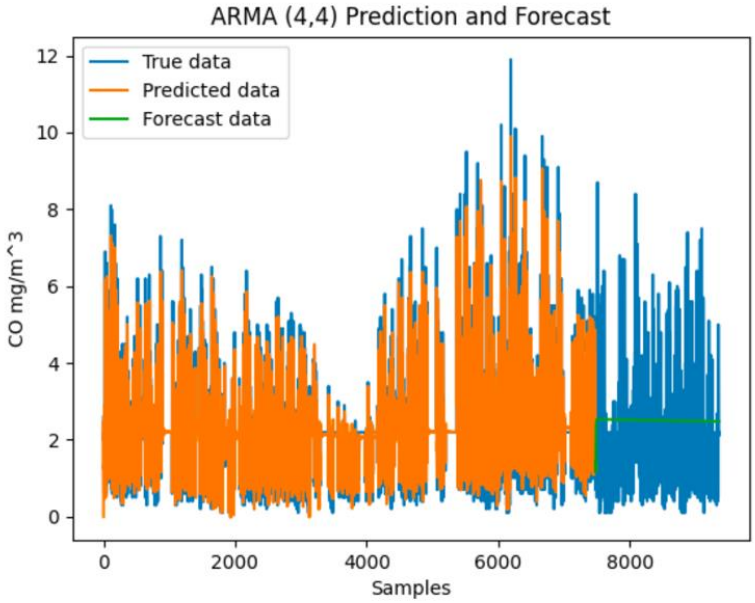


Fig 31 shows the ARMA (4,4) real data, prediction set and forecast set. The prediction set looks like capture most of the variance of train set. But the forecast set is flat line which will cause a large MSE.

Fig 32.

Roots

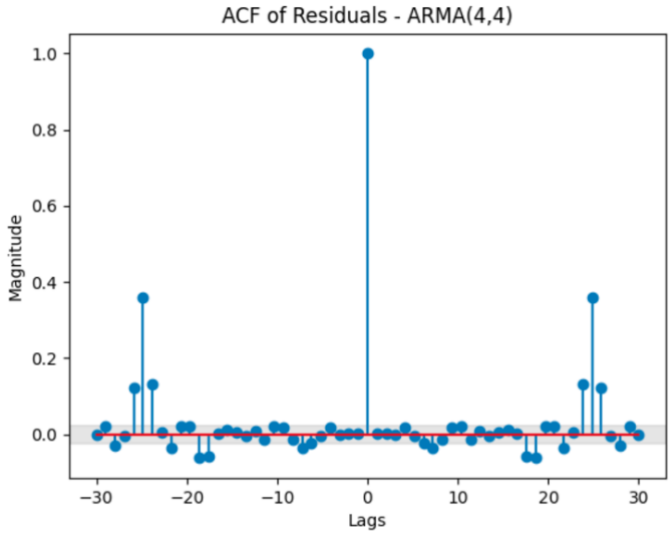
	Real	Imaginary	Modulus	Frequency
AR.1	-0.3930	-0.9842j	1.0598	-0.3105
AR.2	-0.3930	+0.9842j	1.0598	0.3105
AR.3	1.0000	-0.0000j	1.0000	-0.0000
AR.4	1.4211	-0.0000j	1.4211	-0.0000
MA.1	1.0038	-0.0000j	1.0038	-0.0000
MA.2	-0.3864	-0.9689j	1.0431	-0.3104
MA.3	-0.3864	+0.9689j	1.0431	0.3104
MA.4	-3.8490	-0.0000j	3.8490	-0.5000

Fig 32 show the coefficients of AR and MA are all different, which mean there is no zero/pole cancellation needed.

```
The confidence interval for estimated coefficient a0 is: [1.00368849 1.00378418]
The confidence interval for estimated coefficient a1 is: [-0.40161776 -0.40160775]
The confidence interval for estimated coefficient a2 is: [1.02436634 1.02440367]
The confidence interval for estimated coefficient a3 is: [-0.62656554 -0.62647026]
The confidence interval for estimated coefficient b0 is: [-0.04800631 -0.00423333]
The confidence interval for estimated coefficient b1 is: [0.12669624 0.14757349]
The confidence interval for estimated coefficient b2 is: [-0.87096508 -0.85021602]
The confidence interval for estimated coefficient b3 is: [-0.25979253 -0.21591932]
```

All the intervals does not include zero, which means the coefficients are significant.

Fig 33.




```
The MSE of ARMA (4,4) is :  2.07
The variance of prediction error is :  0.5468311822473908
The variance of forecast error is : 1.794946578668091
The variance of prediction vs forecast error is :  0.30465039391487486
The Q value of ARMA (4,4) is :  1325.590752222067
The residual is NOT white
```

```
The corvariance of estimated parameters is :  [[ 5.95990628e-10 -2.40959986e-10  2.21956351e-11 -1.72726805e-10
  -4.79725676e-09 -9.93796136e-09 -9.40185626e-09 -6.97779211e-09]
 [-2.40959986e-10  6.51189017e-12  2.31364226e-10  2.63104959e-12
   2.13720055e-10  1.09781626e-10 -9.03702075e-11 -1.46279118e-10]
 [ 2.21956351e-11  2.31364226e-10  9.06666433e-11 -3.48254682e-10
   1.13966191e-10  1.65410414e-10  1.02626558e-11  2.92952412e-10]
 [-1.72726805e-10  2.63104959e-12 -3.48254682e-10  5.90836240e-10
  -1.41728876e-09 -3.65669792e-09 -3.07084129e-09 -2.82508854e-09]
 [-4.79725676e-09  2.13720055e-10  1.13966191e-10 -1.41728876e-09
   1.24697045e-04 -3.01236387e-05  2.26014986e-05 -1.13891771e-04]
 [-9.93796136e-09  1.09781626e-10  1.65410414e-10 -3.65669792e-09
  -3.01236387e-05  2.83654958e-05 -1.42064344e-05  2.33500795e-05]
 [-9.40185626e-09 -9.03702075e-11  1.02626558e-11 -3.07084129e-09
   2.26014986e-05 -1.42064344e-05  2.80182474e-05 -2.93957907e-05]
 [-6.97779211e-09 -1.46279118e-10  2.92952412e-10 -2.82508854e-09
  -1.13891771e-04  2.33500795e-05 -2.93957907e-05  1.25268745e-04]]
```

Fig 33 shows that the ACF of ARMA(4,4) is not white noise. There is information left in the dataset. The variance of prediction error is 0.546 and the variance of forecast error is 1.79. The variances are not close, which means the model is not good enough.

Fig 34. Estimate ARMA (1,6) model parameters using Levenberg Marquardt algorithm

ARMA Model Results

Dep. Variable:	y	No. Observations:	7485
Model:	ARMA(1, 6)	Log Likelihood	-8394.406
Method:	css-mle	S.D. of innovations	0.743
Date:	Wed, 05 May 2021	AIC	16804.813
Time:	00:09:24	BIC	16860.178
Sample:	0	HQIC	16823.827

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1.y	0.9996	0.000	3198.034	0.000	0.999	1.000
ma.L1.y	-0.0302	0.012	-2.584	0.010	-0.053	-0.007
ma.L2.y	-0.2710	0.011	-23.899	0.000	-0.293	-0.249
ma.L3.y	-0.2388	0.012	-19.828	0.000	-0.262	-0.215
ma.L4.y	-0.1535	0.013	-12.270	0.000	-0.178	-0.129
ma.L5.y	-0.1098	0.012	-9.521	0.000	-0.132	-0.087
ma.L6.y	-0.1125	0.012	-9.398	0.000	-0.136	-0.089

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0004	+0.0000j	1.0004	0.0000
MA.1	1.0259	-0.0000j	1.0259	-0.0000
MA.2	0.6172	-1.4707j	1.5950	-0.1868
MA.3	0.6172	+1.4707j	1.5950	0.1868
MA.4	-1.5046	-0.0000j	1.5046	-0.5000
MA.5	-0.8658	-1.2302j	1.5043	-0.3476
MA.6	-0.8658	+1.2302j	1.5043	0.3476

```
The standard deviation of the parameter estimates is:  0.4102089100790332
```

Fig 34 shows estimated ARMA(1,6) parameters. The parameter estimates are in the green square, the confidence intervals are in the yellow square. The standard deviation of the parameter estimates is 0.41.

Diagnostic Analysis for ARMA(1,6)

Fig 35.

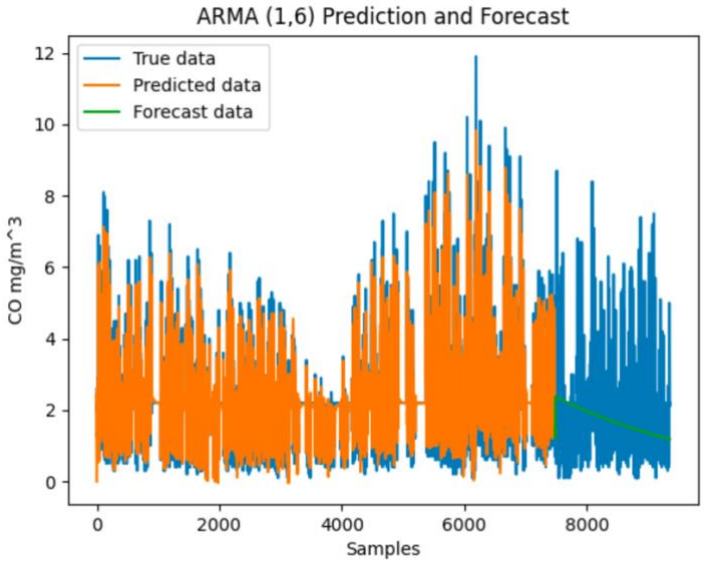


Fig 35 shows the ARMA (1,6) real data, prediction set and forecast set. The prediction set looks like capture most of the variance of train set. But the forecast set is a green line and going down, which cannot capture the variation of test set.

Fig 36.

Roots

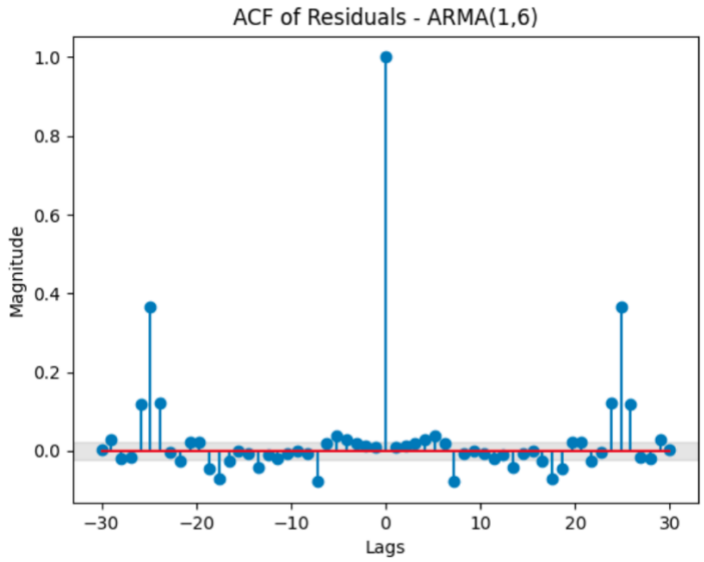
	Real	Imaginary	Modulus	Frequency
AR.1	1.0004	+0.0000j	1.0004	0.0000
MA.1	1.0259	-0.0000j	1.0259	-0.0000
MA.2	0.6172	-1.4707j	1.5950	-0.1868
MA.3	0.6172	+1.4707j	1.5950	0.1868
MA.4	-1.5046	-0.0000j	1.5046	-0.5000
MA.5	-0.8658	-1.2302j	1.5043	-0.3476
MA.6	-0.8658	+1.2302j	1.5043	0.3476

Fig 36 show the coefficients of AR and MA are all different, which mean there is no zero/pole cancellation needed.

```
The confidence interval for estimated coefficient a0 is: [0.99901652 1.00024179]
The confidence interval for estimated coefficient b0 is: [-0.05303337 -0.00728584]
The confidence interval for estimated coefficient b1 is: [-0.29317647 -0.24873491]
The confidence interval for estimated coefficient b2 is: [-0.26240877 -0.21519767]
The confidence interval for estimated coefficient b3 is: [-0.17803719 -0.12899349]
The confidence interval for estimated coefficient b4 is: [-0.13243066 -0.08721514]
The confidence interval for estimated coefficient b5 is: [-0.13600019 -0.08906483]
```

All the intervals do not include zero, which means the coefficients are significant.

Fig 37.




```
The MSE of ARMA (1,6) is :  1.97
The variance of prediction error is :  0.5521599736751408
The variance of forecast error is : 1.8980612532649817
The variance of prediction vs forecast error is :  0.29090735229188924
The Q value of ARMA (1,6) is :  1381.4985774027805
The residual is NOT white
```

```
The corvariance of estimated parameters is :  [[ 9.77038487e-08 -2.83866440e-07 -3.57226934e-07 -3.58315499e-07
  -4.14776510e-07 -5.39784360e-07 -5.29328240e-07]
 [-2.83866440e-07  1.36200622e-04 -8.08163993e-06 -4.64989215e-05
  -4.28641018e-05 -2.24784489e-05  1.46905420e-05]
 [-3.57226934e-07 -8.08163993e-06  1.28535345e-04 -1.36765722e-05
  -4.30371201e-05 -1.01498089e-05 -1.47565672e-05]
 [-3.58315499e-07 -4.64989215e-05 -1.36765722e-05  1.45054815e-04
   3.15875517e-05 -3.36831099e-05 -4.39648677e-05]
 [-4.14776510e-07 -4.28641018e-05 -4.30371201e-05  3.15875517e-05
   1.56534542e-04 -1.18328983e-05 -4.55685474e-05]
 [-5.39784360e-07 -2.24784489e-05 -1.01498089e-05 -3.36831099e-05
  -1.18328983e-05  1.33051245e-04  3.31738118e-06]
 [-5.29328240e-07  1.46905420e-05 -1.47565672e-05 -4.39648677e-05
  -4.55685474e-05  3.31738118e-06  1.43365281e-04]]
```

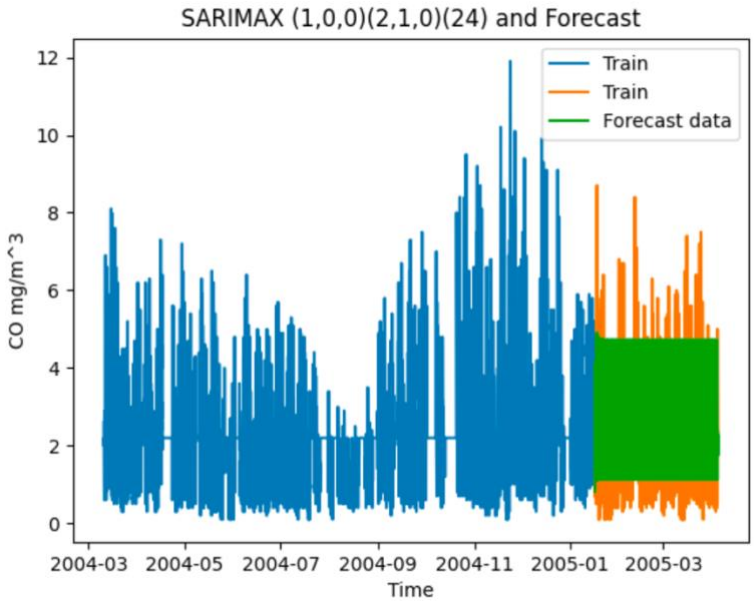
Fig 37 shows that the ACF of ARMA(1,6) is not white noise. There is information left in the dataset. The variance of prediction error is 0.55 and the variance of forecast error is 1.89. The variances are not close, which means the model is not good enough.

SARIMAX Model

```
Performing stepwise search to minimize aic
 ARIMA(2,0,1)(1,1,1)[24] intercept   : AIC=inf, Time=170.46 sec
 ARIMA(0,0,0)(0,1,0)[24] intercept   : AIC=30436.012, Time=1.11 sec
 ARIMA(1,0,0)(1,1,0)[24] intercept   : AIC=21006.733, Time=33.32 sec
 ARIMA(0,0,1)(0,1,1)[24] intercept   : AIC=inf, Time=44.09 sec
 ARIMA(0,0,0)(0,1,0)[24]             : AIC=30434.013, Time=0.73 sec
 ARIMA(1,0,0)(0,1,0)[24] intercept   : AIC=22855.450, Time=3.26 sec
 ARIMA(1,0,0)(2,1,0)[24] intercept   : AIC=19978.569, Time=161.86 sec
 ARIMA(1,0,0)(2,1,1)[24] intercept   : AIC=inf, Time=705.87 sec
 ARIMA(1,0,0)(1,1,1)[24] intercept   : AIC=inf, Time=67.12 sec
 ARIMA(0,0,0)(2,1,0)[24] intercept   : AIC=28865.842, Time=92.65 sec
```

For SARIMAX model, I pick the order ARIMA(1,0,0)(2,1,0)[24], because it has the smallest AIC value.

Fig 38.




```
The MSE of SARIMAX (1,0,0)(2,1,0)(24) is :  1.63
The variance of forecast error is :  1.5103967009786492
The mean of forecast error is :  -0.3517500199405328
The Q value of SARIMAX (1,0,0)(2,1,0)(24) is :  52399.673084391514
```

Fig 38 shows the SARIMAX (1,0,0)(2,1,0)(24) model train set, test set and forecast set. The green lines are the forecast which can capture the seasonality of the data. The MSE is 1.63, which is good, but the Q value is too high, thus this model will not ne selected.

Final Model Selection

Model	Forecast MSE
Multiple Linear Regression Model	0.5
SARIMAX (1,0,0)(2,1,0)(24)	1.63
Average Model	1.85
ARMA(1,6)	1.97
ARMA(4,4)	2.07
SES Model (alfa = 0.5)	2.78
Holt-Winters Model (seasonal_periods=12)	2.83
Naïve Model	3.18
Holt-Winters Model (seasonal_periods=24)	3.4
Drift Model	3.78

For the final model selection, I use the forecast MSE as the most important metric. Because it measures the error between test and forecast set and our goal is to have good forecast for new dataset. The from is sorted from the smallest MSE to the largest MSE. It indicates that Multiple Linear Regression Model has the lowest MSE, and it will be my final model.

Forecast Function:

Y = (0.0042) * NO(GT) + (-0.0008) * PT08.S3(NOx) + (0.0012) * PT08.S42(NO2)

The forecast function is listed above, it combines with three features and their coefficients.

H-step Ahead Prediction

Fig 39.

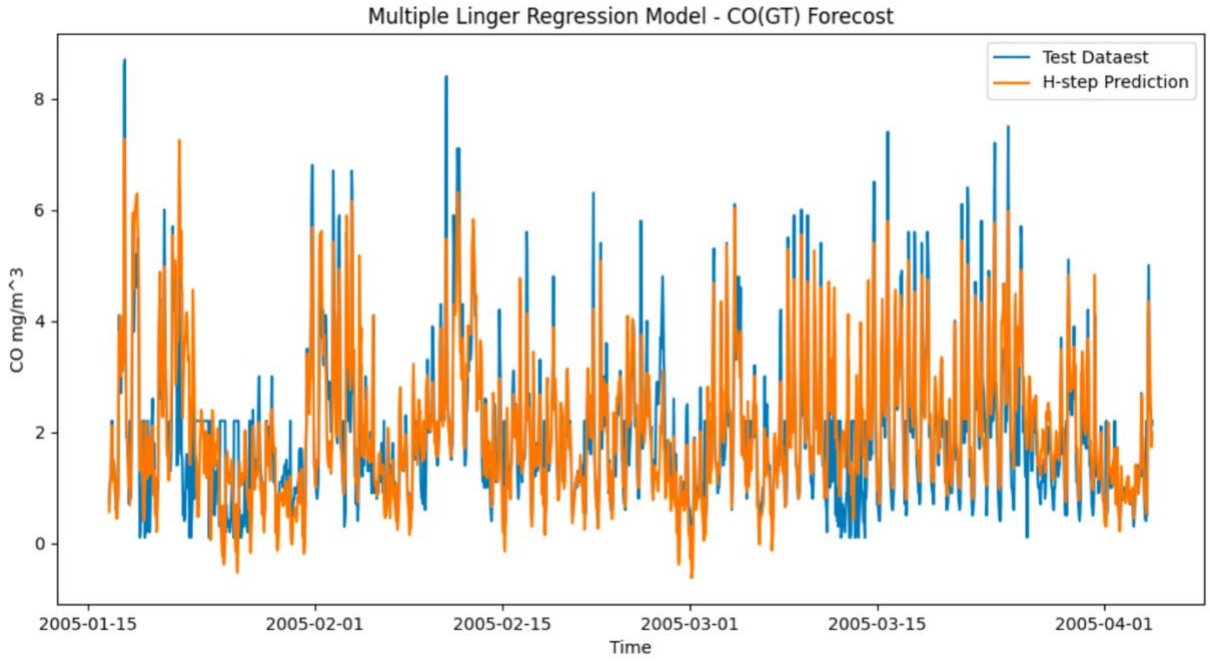


Fig 39 shows the h-step ahead prediction of Multiple Linger Regression Model. The test data is in blue color and the forecast is in orange color. As we can see that the forecast set capture most of the variation of the test set because the orange covers most of the blue lines.

Summary and Conclusion

After model performance comparison, the Multiple Linger Regression Model is the best one among 10 models. It has lowest MSE 0.5 and the variance of prediction error and forecast error are very close. Which means the model can be applied on new dataset and have good performance as well. The final model uses three features and have decent adjusted r-square value as 0.93, which indicates the model has high accuracy.

Appendix

```
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.stats import chi2
import statsmodels.api as sm
from numpy import linalg as LA
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import STL
import statsmodels.tsa.holtwinters as ets
```

```
from statsmodels.tsa.stattools import adfuller
from sklearn.model_selection import train_test_split
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
from pmdarima import auto_arima
from statsmodels.tsa.arima_model import ARIMA,ARIMAResults,ARMA,ARMAResults
from statsmodels.tsa.statespace.sarimax import SARIMAX

# ADF test function
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))
# ACF function
def auto_cor_cal(y, t=0):
    y_diff = y - np.mean(y)
    if t:
        return np.sum(y_diff[t:] * y_diff[:-t]) \
               / np.sum(np.square(y_diff))
    else:
        return 1

# GPAC function
def gpac_cal (r, row, col):
    def calc_phi(r, j, k):

        b = [[0 for _ in range(k)] for _ in range(k)]
        for m in range(k):
            for n in range(k):
                b[m][n] = r[abs(j + m - n)]

        a = [[e for e in row] for row in b]  # b.copy()
        for i in range(k):
            a[i][-1] = r[j + 1 + i]

        # to numpy
        a_np = np.array(a)
        b_np = np.array(b)
        return np.linalg.det(a_np) / np.linalg.det(b_np)

    retval = [[0 for _ in range(1, col)] for _ in range(row)] # all zero

    for j in range(row):
        for k in range(1, col):
            retval[j][k - 1] = calc_phi(r, j, k)

    result = np.array(retval)

    df = pd.DataFrame(result, index=np.arange(row), columns=np.arange(1,
col)) # table
    print(df)
    return df

# import dataset
data = pd.read_excel('AirQualityUCI.xlsx',parse_dates=[["Date",
```

```
"Time"]],index_col=[0])

# Plot of the dependent variable vs time
y = data[["CO(GT)"]]
plt.figure()#fig, ax = plt.subplots()
plt.plot(y)
plt.title("CO(GT) vs Time")
plt.ylabel("CO mg/m^3")
plt.xlabel("Time")
plt.legend(["CO(GT)"])
plt.show()

# ACF/PACF pf the dependent variables
lags = 100
acf = sm.tsa.stattools.acf(y, nlags = lags)
pacf = sm.tsa.stattools.pacf(y, nlags = lags)

plt.figure(figsize=(8,8))
plt.subplot(211)
plot_acf(y,ax=plt.gca(),lags =lags)
plt.ylabel("Magnitude")
plt.subplot(212)
plot_pacf(y,ax=plt.gca(),lags =lags)
plt.ylabel("Magnitude")
plt.xlabel("Lags")
plt.show()

# Correlation Matrix with seaborn heatmap
cor = data.corr()

fig, ax = plt.subplots(figsize=(10,10))
ax =sns.heatmap(cor, vmin = -1, vmax = 1, annot=True,
                center=0,cmap=sns.diverging_palette(20,220,n=200),
square=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_xticklabels(ax.get_xticklabels(),rotation=45,
horizontalalignment='right')
plt.title("Correlation Matrix of Air Quality Dataset")
plt.show()

# Split the dataset into train set 80% and test set 20%
y_train, y_test = train_test_split(y, shuffle= False, test_size=0.2)

# Stationarity check ------------------------------------------------
Stationarity check
ADF_Cal(y)

# Plot mean and variance versus time
CO_mean = []
CO_var = []

for i in range(1, len(data)+1):
    CO_mean.append(data.head(i)["CO(GT)"].mean())
    CO_var.append(data.head(i)["CO(GT)"].var())

data["COmean"] = CO_mean
```

```
data["COvar"] = CO_var

plt.figure(figsize=(8,8))
plt.subplot(211)
plt.plot(data.COmean)
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Rolling Mean and Rolling Variance of Orginal data')
plt.legend(["Rolling Mean"])

plt.subplot(212)
plt.plot(data.COvar)
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend(["Rolling Variance"])
plt.show()

#================================
# y with first differencing
#================================
CO_1 = y.diff(1)
CO_1 = CO_1[1:]
data['CO_1'] = CO_1

CO_1_mean = []
CO_1_var = []

for i in range(1, len(data)+1):
    CO_1_mean.append(data.head(i).CO_1.mean())
    CO_1_var.append(data.head(i).CO_1.var())

data["CO_1_mean"] = CO_1_mean
data["CO_1_var"] = CO_1_var

plt.figure(figsize=(8,8))
plt.subplot(211)
plt.plot(data.CO_1_mean)
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Plot of Rolling mean and Rolling Variance with Differencing ')
plt.legend(["Rolling Mean"])

plt.subplot(212)
plt.plot(data.CO_1_var)
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend(["Rolling Variance"])
plt.show()

ADF_Cal(CO_1)

# ACF/PACF pf the differenced y target
lags = 100

acf = sm.tsa.stattools.acf(CO_1, nlags = lags)
pacf = sm.tsa.stattools.pacf(CO_1, nlags = lags)
```

```
plt.figure(figsize=(8,8))
plt.subplot(211)
plot_acf(CO_1,ax=plt.gca(),lags =lags)
plt.ylabel("Magnitude")
plt.subplot(212)
plot_pacf(CO_1,ax=plt.gca(),lags =lags)
plt.ylabel("Magnitude")
plt.xlabel("Lags")
plt.show()

#================================
# y with second differencing
#================================
y = data[["CO_1"]]
CO_1_24 = y.diff(24)
CO_1_24 = CO_1_24[24:]
CO_1_24 = CO_1_24[1:]
data['CO_1_24'] = CO_1_24

CO_1_24_mean = []
CO_1_24_var = []

for i in range(1, len(data)+1):
    CO_1_24_mean.append(data.head(i).CO_1.mean())
    CO_1_24_var.append(data.head(i).CO_1.var())

data["CO_1_24_mean"] = CO_1_24_mean
data["CO_1_24_var"] = CO_1_24_var

plt.figure(figsize=(8,8))
plt.subplot(211)
plt.plot(data.CO_1_24_mean)
plt.xlabel('Time')
plt.ylabel('Values')
plt.title('Plot of Rolling mean and Rolling Variance with Differencing ')
plt.legend(["Rolling Mean"])

plt.subplot(212)
plt.plot(data.CO_1_24_var)
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend(["Rolling Variance"])
plt.show()

ADF_Cal(CO_1_24)

# ACF/PACF of the differenced y target

acf = sm.tsa.stattools.acf(CO_1_24, nlags = lags)
pacf = sm.tsa.stattools.pacf(CO_1_24, nlags = lags)

plt.figure(figsize=(8,8))
plt.subplot(211)
plot_acf(CO_1_24,ax=plt.gca(),lags =lags)
plt.ylabel("Magnitude")
plt.subplot(212)
plot_pacf(CO_1_24,ax=plt.gca(),lags =lags)
```

```
plt.ylabel("Magnitude")
plt.xlabel("Lags")
plt.show()

# Time Series Decomposition ----------------------------------------
Decomposition
CO = data["CO(GT)"]
CO = pd.Series(np.array(data["CO(GT)"]),
                  index = pd.date_range('2004-03-10 18:00:00', periods
=len(CO), freq='h'),
                  name='Air Quality (CO) Decomposition Plot' )
STL = STL(CO)
res = STL.fit()

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

fig = res.plot()
fig.set_size_inches(8, 6)
plt.show()

#Calculate seasonally adjusted data and plot it vs the original
adjusted_seasonal = CO - S
detrend = CO - T

plt.figure(figsize=(8,6))
plt.plot(CO, label= 'Original set')
plt.plot(detrend, label='Detrended')
plt.title("Original vs Detrended Data")
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.legend()
plt.show()

plt.figure(figsize=(8,6))
plt.plot(CO, label= 'Original set')
plt.plot(adjusted_seasonal, label='Adjusted seasonal')
plt.title("Original vs Seasonally Adjusted")
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.legend()
plt.show()

# # Find the strength of trend and strength of seasonality
R = np.array(R)
S = np.array(S)
T = np.array(T)
Ft = np.max([0,1 - np.var(R)/np.var(T+R)])
Fs = np.max([0,1 - np.var(R)/np.var(S+R)])
print("The strength of trend for this dataset is =", Ft)
print("The strength of seasonality for this dataset is =", Fs)

# Feature selection ------------------------------------------------Feature
selection
X = data[["PT08.S1(CO)","C6H6(GT)", "PT08.S2(NMHC)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)","T","RH","AH"]]
```

```
X = sm.add_constant(X)
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())
X = X_train.values
print("The condition number for X original is =", LA.cond(X))
H = np.matmul(X.T,X)
s,d,v = np.linalg.svd(H)
print("SingularValues d original is =", d)

#================
# remove constant 0.949
#================
X = data[["PT08.S1(CO)","C6H6(GT)", "PT08.S2(NMHC)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)","T","RH","AH"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#================
# remove AH 0.949
#================
X = data[["PT08.S1(CO)","C6H6(GT)", "PT08.S2(NMHC)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)","T","RH"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#================
# remove C6H6(GT) 0.946
#================
X = data[["PT08.S1(CO)","PT08.S2(NMHC)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)","T","RH"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#=====================
# remove PT08.S2(NMHC) 0.946
#=====================
X = data[["PT08.S1(CO)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)","T","RH"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
```

```
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#=====================
# remove T 0.938
#=====================
X = data[["PT08.S1(CO)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","PT08.S5(O3)","RH"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#=====================
# remove PT08.S5(O3) 0.938
#=====================
X = data[["PT08.S1(CO)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)","RH"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#=====================
# remove RH 0.936
#=====================
X = data[["PT08.S1(CO)","NOx(GT)","PT08.S3(NOx)",
          "NO2(GT)","PT08.S4(NO2)"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#=====================
# remove NO2(GT) 0.934
#=====================
X = data[["PT08.S1(CO)","NOx(GT)","PT08.S3(NOx)","PT08.S4(NO2)"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
predictions = model.predict(X_test)
print(model.summary())

#=========================
# remove PT08.S1(CO) 0.933
#=========================
X = data[[ "NOx(GT)","PT08.S3(NOx)","PT08.S4(NO2)"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
```

```
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
print(model.summary())

X = X_train.values
print("The condition number for X after feature elimination is =",
LA.cond(X))
H = np.matmul(X.T,X)
s,d,v = np.linalg.svd(H)
print("SingularValues d after feature elimination is =", d)

# Develop the multiple linear regression model------------------------------
Muti Linear
Muli_linear_FE = np.ndarray.flatten(Y_test.values) -
np.array(test_predictions) # residuals of forecast
MSE_Muli_linear = round(np.square(Muli_linear_FE).mean(),2)

plt.figure()
plt.plot(Y_train)
plt.plot(Y_test)
plt.plot(train_predictions)
plt.plot(test_predictions)
plt.title("Multiple Linger Regression Model with MSE =
{}".format(MSE_Muli_linear))
plt.xlabel("Time")
plt.ylabel("CO mg/m^3")
plt.legend(['Train Dataset','Test Dataest','Prediction','Forecast'])
plt.show()

#t-test
print("The p-value of t-test is: ",model.pvalues)
#f-test
print("The p-value of f-test is: ",model.f_pvalue)

# AIC,BIC,RMSE,R-squared and Adjusted R-squared
Muli_linear_PE = np.ndarray.flatten(Y_train.values) -
np.array(train_predictions) # residuals of prediction
Muli_linear_FE = np.ndarray.flatten(Y_test.values) -
np.array(test_predictions) # residuals of forecast
MSE_Muli_linear = round(np.square(Muli_linear_FE).mean(),2)

print("The AIC value of the model is :",model.aic)
print("The BIC value of the model is :",model.bic)
print("The MSE value of residual is :",MSE_Muli_linear)
print("The RMSE value of residual is :",np.sqrt(MSE_Muli_linear))
print("The R-squared value of the model is: ",model.rsquared)
print("The Adjusted R-squared value of the model is: ",model.rsquared_adj)

plt.figure()
plt.plot(Y_train.index,Muli_linear_PE)
plt.plot(Y_test.index,Muli_linear_FE)
plt.title("Multiple Linger Regression Model Prediction Error vs Forecast
Error")
plt.xlabel("Time")
plt.ylabel("CO mg/m^3")
```

```
plt.legend(['Prediction Error','Forecast Error'])
plt.show()

# ACF of residuals
lags = 30
ry = []
for t in range(lags):
    ry.append(auto_cor_cal(Muli_linear_FE, t))
Ry_Muli_linear_FE = ry[::-1][:-1] + ry
x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(Muli_linear_FE))
plt.stem(x, Ry_Muli_linear_FE)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - Multiple Linger Regression Model")
plt.show()

# Q value
Q_Muli_linear_FE =
len(Muli_linear_FE)*sum(np.square(Ry_Muli_linear_FE[lags:]))
print("The Q-value is :",Q_Muli_linear_FE)

# Variance and mean of the residuals
Var_Muli_linear_PE = Muli_linear_PE.var()
Mean_Muli_linear_PE = Muli_linear_PE.mean()

Var_Muli_linear_FE = Muli_linear_FE.var()
Mean_Muli_linear_FE = Muli_linear_FE.mean()

print("The variance of the predict error is :",Var_Muli_linear_PE)
print("The mean of the predict error is :",Mean_Muli_linear_PE)

print("The variance of the forecast error is :",Var_Muli_linear_FE)
print("The mean of the forecast error is :",Mean_Muli_linear_FE)

# 14 Base-models
# -------------Average Method-----------------------------------------------
----
lags = 30
ytrain = y_train.values
ytest = y_test.values
hstep = []
for i in range(len(ytest)):
    hstep.append(np.mean(ytrain))
hstep_np = np.array(hstep)                                  #forecast
Average_FE = ytest.flatten()-hstep_np                       #forecast error
MSE_Average = round(np.square(Average_FE).mean(),2)         #forecast MSE

RMSE_Average = np.sqrt(MSE_Average)
Average_FE_mean = Average_FE.mean()
Average_FE_var = Average_FE.var()

print("The mean of error of Average Model is :", Average_FE_mean)
print("The variance of Average of Naive Model is :", Average_FE_var)
print("The MSE of Average Model is :", MSE_Average)
print("The RMSE of Average Model is :",RMSE_Average)
```

```
plt.plot(y_train,label= "Train Data")
plt.plot(y_test,label= "Test Data")
plt.plot(y_test.index,hstep_np,label= "Average Method prediction")
plt.legend(loc='upper left')
plt.title("Average Method with MSE = {}".format(MSE_Average))
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.show()

ry = []
for t in range(lags):
    ry.append(auto_cor_cal(Average_FE, t))
Average_Ry = ry[::-1][:-1] + ry                          #forecast error ACF

Average_Q = len(Average_FE)*np.sum(np.square(ry[1:]))    # Q
print("The Q value of Average Model is:",Average_Q)

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(Average_FE))
plt.stem(x, Average_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - Average Model")
plt.show()

#----------Naive Method-----------------------------------------------------
-------
hstepNaive = []
for i in range(len(ytest)):
    hstepNaive.append(ytrain[-1])
Naive_hstep_np = np.array(hstepNaive)                    #forecast
Naive_FE = ytest.flatten()-Naive_hstep_np.flatten()           #forecast
error
MSE_Naive = round(np.square(Naive_FE).mean(),2)     #forecast MSE

RMSE_Naive = np.sqrt(MSE_Naive)
Naive_FE_mean = Naive_FE.mean()
Naive_FE_var = Naive_FE.var()

print("The mean of error of Naive Model is :", Naive_FE_mean)
print("The variance of error of Naive Model is :", Naive_FE_var)
print("The MSE of Naive Model is :", MSE_Naive)
print("The RMSE of Naive Model is :",RMSE_Naive)

plt.plot(y_train,label= "Train Data")
plt.plot(y_test,label= "Test Data")
plt.plot(y_test.index,Naive_hstep_np,label= "Naive Method prediction")
plt.legend(loc='upper left')
plt.title("Naive Method with MSE = {}".format(MSE_Naive))
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.show()

ry = []
for t in range(lags):
```

```
    ry.append(auto_cor_cal(Naive_FE, t))
Naive_Ry = ry[::-1][:-1] + ry                           #forecast error ACF

Naive_Q = len(Naive_FE)*np.sum(np.square(ry[1:]))    # Q
print("The Q value of Naive Model is:",Naive_Q)

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(Naive_FE))
plt.stem(x, Naive_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - Naive Model")
plt.show()

#----------Drift Method-----------------------------------------------------
-------
hstepDrift = []
for i in range(len(ytrain)+1,len(data)+1):
    slope = (ytrain[-1] - ytrain[0]) / (len(ytrain) - 1)
    hstepDrift.append(slope * i + ytrain[0] - slope )
Drift_hstep_np = np.array(hstepDrift)                    #forecast
Drift_FE = ytest.flatten()-Drift_hstep_np.flatten()             #forecast
error
MSE_Drift = round(np.square(Drift_FE).mean(),2)     #forecast MSE
RMSE_Drift = np.sqrt(MSE_Drift)
Drift_FE_mean = Drift_FE.mean()
Drift_FE_var = Drift_FE.var()

print("The mean of error of Drift Model is :", Drift_FE_mean)
print("The variance of error of Drift Model is :", Drift_FE_var)
print("The MSE of Drift Model is :", MSE_Drift)
print("The RMSE of Drift Model is :",RMSE_Drift)

plt.plot(y_train,label= "Train Data")
plt.plot(y_test,label= "Test Data")
plt.plot(y_test.index,Drift_hstep_np,label= "Drift Method prediction")
plt.legend(loc='upper left')
plt.title("Drift Method with MSE = {}".format(MSE_Drift))
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.show()

ry = []
for t in range(lags):
    ry.append(auto_cor_cal(Drift_FE, t))
Drift_Ry = ry[::-1][:-1] + ry                           #forecast error ACF

Drift_Q = len(Drift_FE)*np.sum(np.square(ry[1:]))    # Q
print("The Q value of Drift Model is:",Drift_Q)

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(Drift_FE))
plt.stem(x, Drift_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
```

```
plt.title("ACF of Residuals - Drift Model")
plt.show()

#----------SES Method-----------------------------------------------------
-----
yhat = []
for i in range(len(ytrain)-1):
    yhat.append(ytrain[i])
    yhat[i] = 0.5*ytrain[i]+0.5*yhat[i-1]

hstepSES = []
for i in range(len(ytest)):
    hstepSES.append(yhat[-1])
SES_hstep_np = np.array(hstepSES)                              #forecast
SES_FE = ytest.flatten()- SES_hstep_np.flatten()                    #forecast
error
MSE_SES = round(np.square(SES_FE).mean(),2)            #forecast MSE
RMSE_SES = np.sqrt(MSE_SES)
SES_FE_mean = SES_FE.mean()
SES_FE_var = SES_FE.var()

print("The mean of error of SES Model is :", SES_FE_mean)
print("The variance of error of SES Model is :", SES_FE_var)
print("The MSE of SES Model is :", MSE_SES)
print("The RMSE of SES Model is :",RMSE_SES)

plt.plot(y_train,label= "Train Data")
plt.plot(y_test,label= "Test Data")
plt.plot(y_test.index, SES_hstep_np,label= "SES Method prediction")
plt.legend(loc='upper left')
plt.title("SES Method with MSE = {}".format(MSE_SES))
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.show()

ry = []
for t in range(lags):
    ry.append(auto_cor_cal(SES_FE, t))
SES_Ry = ry[::-1][:-1] + ry                              #forecast error ACF

SES_Q = len(SES_FE)*np.sum(np.square(ry[1:]))     # Q
print("The Q value of SES Model is:",SES_Q)

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(SES_FE))
plt.stem(x, SES_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - SES Model")
plt.show()

# ---------------Holt-Winters method-----------------------------------------
--------
holtt =
ets.ExponentialSmoothing(y_train,trend='add',damped_trend=False,seasonal='add
',seasonal_periods=24).fit()
```

```
holtf = holtt.forecast(steps=len(y_test))
# please try seasonal_periods= 12
HoltWinter_hstep_np =np.array(holtf)
#forecast
HoltWinter_FE = np.array(y_test.values).flatten() - HoltWinter_hstep_np
#forecast error
MSE_HoltWinter = round(np.square(HoltWinter_FE).mean(),2)
#forecast MSE

RMSE_HoltWinter = np.sqrt(MSE_HoltWinter)
HoltWinter_FE_mean = HoltWinter_FE.mean()
HoltWinter_FE_var = HoltWinter_FE.var()

print("The mean of error of HoltWinter Model is :", HoltWinter_FE_mean)
print("The variance of error of HoltWinter Model is :", HoltWinter_FE_var)
print("The MSE of HoltWinter Model is :", MSE_HoltWinter)
print("The RMSE of HoltWinter Model is :",RMSE_HoltWinter)

lags = 30
ry = []
for t in range(lags):
    ry.append(auto_cor_cal(HoltWinter_FE, t))
HoltWinter_Ry = ry[::-1][:-1] + ry                          #forecast error
ACF

HoltWinter_Q = len(HoltWinter_FE)*np.sum(np.square(ry[1:]))    # Q
print("The Q value of HoltWinter Model is:",HoltWinter_Q)

plt.plot(y_train,label= "Train Data")
plt.plot(y_test,label= "Test Data")
plt.plot(y_test.index,HoltWinter_hstep_np,label= "Holt Winter Method
prediction")
plt.legend(loc='upper left')
plt.title("Holt Winter Method with MSE = {}".format(MSE_HoltWinter))
plt.xlabel("Date")
plt.ylabel("CO mg/m^3")
plt.show()

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(HoltWinter_FE))
plt.stem(x, HoltWinter_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - HoltWinter Model")
plt.show()

# 11, ARMA and ARIMA and SARIMA model order determination--------------------
------ARMA
y = CO_1_24.values # differenced data

ry = []
for t in range(lags):
    ry.append(auto_cor_cal(y, t))

# Display GPAC table for k=8, j=8, use seaborn
table = gpac_cal(ry,10,10)
```

```
fig, ax = plt.subplots(figsize=(10,10))
ax =sns.heatmap(table, vmin = -1, vmax = 1, annot=True,
                center=0,cmap=sns.diverging_palette(20,220,n=200),
square=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_xticklabels(ax.get_xticklabels(),rotation=45,
horizontalalignment='right')
plt.title("Generalized Partial Autocorrelation(GPAC) Table")
plt.show()


# 12 Estimate ARMA model parameters using Levenberg Marquardt algorithm
y = data["CO(GT)"].values # orignal data
y_train, y_test = train_test_split(y, shuffle= False, test_size=0.2)
# na = 4
# nb = 4
na = 1
nb = 6

model = sm.tsa.ARMA(y_train,(na,nb)).fit(trend='nc',disp=0)

print(model.summary())

for i in range(na):
    print("The AR estimated coefficient a{}".format(i), "is:",
model.params[i])
for i in range(nb):
    print("The MA estimated coefficient b{}".format(i), "is:",
model.params[i+na])
for i in range(na):
    print("The confidence interval for estimated coefficient a{}".format(i),
"is:", model.conf_int()[i])
for i in range(nb):
    print("The confidence interval for estimated coefficient b{}".format(i),
"is:", model.conf_int()[i+na])

print("The standard deviation of the parameter estimates is:
",model.params.std())

# 13 Diagnostic Analysis

# Chi-sqyare test
model_hat = model.predict(start=0, end=len(y_train)-1)
model_h_hat = model.predict(start=len(y_train)-1, end=len(data))

plt.figure()
plt.plot(y, label = "True data")
plt.plot(model_hat, label = "Predicted data")
plt.plot(np.arange(7484-1,9357),model_h_hat, label = "Forecast data")
plt.xlabel("Samples")
plt.ylabel("CO mg/m^3")
plt.legend()
plt.title(" ARMA (1,6) Prediction and Forecast ")
plt.show()

lags = 30
```

```
ARMA_PE = y_train-model_hat
ARMA_FE = y_test-model_h_hat[2:]    # forecast error
MSE_ARMA = round(np.square(ARMA_FE).mean(),2)
print("The MSE of ARMA (1,6) is : ",MSE_ARMA)
print("The variance of prediction error is : ",ARMA_PE.var())
print("The variance of forecast error is :",ARMA_FE.var())
print("The variance of prediction vs forecast error is : ",ARMA_PE.var()
/ARMA_FE.var())
ry = []
for t in range(lags):
    ry.append(auto_cor_cal(ARMA_PE, t))
ARMA_Ry = ry[::-1][:-1] + ry                              #prediction error ACF

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(ARMA_PE))
plt.stem(x, ARMA_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - ARMA(1,6)")
plt.show()

Q = len(y_train)*np.sum(np.square(ry[1:]))
print("The Q value of ARMA (1,6) is : ",Q)
DOF = lags - na - nb
alfa = 0.01
chi_critical = chi2.ppf(1-alfa, DOF)
if Q< chi_critical:
    print("The residual is white ")
else:
    print("The residual is NOT white ")

print("The corvariance of estimated parameters is : ",model.cov_params())

# # SARIMA model ----------------------------------------------------------
SARIMA model
# y = CO_1_24.values # differenced data
# # Finding best orders
# Sarimax_model = auto_arima(data["CO(GT)"],
#                            start_P=1,
#                            start_q=1,
#                            max_p=3,
#                            max_q=3,
#                            m=24,
#                            seasonal=True,
#                            d=None,
#                            D=1,
#                            trace=True,
#                            error_action='ignore',
#                            suppress_warnings=True,
#                            stepwise=True)
# Sarimax_model.summary()

# ARIMA(1,0,0)(2,1,0)[24]
y = data[["CO(GT)"]]
y_train, y_test = train_test_split(y, shuffle= False, test_size=0.2)
model = SARIMAX(y_train,order=(1, 0, 0),
```

```
                seasonal_order=(2, 1, 0, 24),
                enforce_stationarity=False,
                enforce_invertibility=False)
results = model.fit()

forecast = results.predict(start = len(y_train),
                           end=len(data),
                           typ='levels')

plt.figure()
plt.plot(y_train, label = "Train")
plt.plot(y_test, label = "Train")
plt.plot(y_test.index,forecast[:-1], label = "Forecast data")
plt.xlabel("Time")
plt.ylabel("CO mg/m^3")
plt.legend()
plt.title(" SARIMAX (1,0,0)(2,1,0)(24) and Forecast ")
plt.show()

SARIMAX_FE = y_test.values.flatten() - forecast[:-1]     # forecast error
MSE_SARIMAX = round(np.square(SARIMAX_FE).mean(),2)
print("The MSE of SARIMAX (1,0,0)(2,1,0)(24) is : ",MSE_SARIMAX)
print("The variance of forecast error is : ",SARIMAX_FE.var())
print("The mean of forecast error is : ",SARIMAX_FE.mean())

lags = 30
ry = []
for t in range(lags):
    ry.append(auto_cor_cal(SARIMAX_FE, t))
SARIMAX_Ry = ry[::-1][:-1] + ry                          #forecast error ACF

x = np.linspace(-lags, lags, 2*lags-1)
m = 1.96/np.sqrt(len(SARIMAX_FE))
plt.stem(x, SARIMAX_Ry)
plt.axhspan(-m,m,alpha = .1, color = 'black')
plt.xlabel("Lags")
plt.ylabel("Magnitude")
plt.title("ACF of Residuals - SARIMAX (1,0,0)(2,1,0)(24)")
plt.show()

Q = len(y_test)*np.sum(np.square(ry[1:]))
print("The Q value of SARIMAX (1,0,0)(2,1,0)(24) is : ",Q)

# H-step Ahead Prediction
X = data[[ "NOx(GT)","PT08.S3(NOx)","PT08.S4(NO2)"]]
Y = data[["CO(GT)"]]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, shuffle = False,
test_size = 0.2)
model = sm.OLS(Y_train, X_train).fit()
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)

plt.figure(figsize=(12,6))
plt.plot(Y_test)
plt.plot(test_predictions)
plt.title("Multiple Linger Regression Model - CO(GT) Forecost")
plt.xlabel("Time")
```

```
plt.ylabel("CO mg/m^3")
plt.legend(['Test Dataest','H-step Prediction'])
plt.show()
```