PREDICTION OF CARBON MONOXIDE

FINAL PROJECT REPORT

TRINH VU

TV 12/16/2020

DATS 6540 Time Series Analysis & Modeling

Professor: Reza Jafari

Table of Contents

Abstract ... 1
Introduction ... 1
Description of Dataset ... 2
Source of dataset ... 2
Data description ... 2
Attribute Information ... 2
Data Preprocessing ... 3
Data Visualization ... 4
Stationarity ... 5
Time Series Decomposition ... 5
Modeling ... 6
1. Multiple Linear Regression ... 6
1.1 Feature Selection ... 6
1.2 Multiple Linear Regression Model ... 10
2. Holt Winters Method ... 12
3. Basic models ... 13
3.1 Average method ... 13
3.2 Naïve method ... 15
3.3 Drift method ... 16
3.4 Simple Exponential Smoothing (SES) ... 17
4. ARMA model ... 18
4.1 Order Estimation ... 18
4.2 Parameters Estimation, Prediction, and Diagnostic Analysis ... 20
5. SARIMA model ... 26
Final Model Selection ... 29
Forecast Function and h-step prediction ... 29
Summary and Conclusion ... 31
References ... 31
Appendix ... 32

Abstract

Air pollution has been a major problem because it affects health by inhaling the air directly every day. One of the pollutants is Carbon Monoxide (CO) gas that is considered a “silent killer” because it can cause sudden death. Knowing the CO level for the future time will be helpful in preventing sudden deaths caused by exposing to CO. In this project, carbon monoxide level of outdoor air will be forecasted using time series models. Different time series models are developed to predict the CO level based on historical data. The models developed in this project includes Multiple Linear Regression, Holt Winters, basic methods (average, naïve, drift, and simple exponential smoothing), autoregressive moving average (ARMA), and seasonal autoregressive integrated moving average (SARIMA). The models are evaluated based on mean squared errors (MSE), Q value, mean of residuals, variance of residuals, and variance of forecast errors. They will be compared, and the best model will be selected to forecast the CO level for outdoor air.

Introduction

Carbon monoxide (CO) is an odorless, colorless gas that can cause a sudden illness and death. CO is produced by burning gasoline, wood, propane, charcoal, or other fuel. Improperly ventilated appliances and engines in an enclosed space can accumulate CO to dangerous levels. CO poisoning occurs when CO builds up in the bloodstream. When too much CO exists in the air, CO will replace oxygen in the red blood cells, which prevents oxygen going to tissues and organs in the body. This will lead to serious tissue damage and sudden death. Therefore, forecasting CO level can help prevent the exposure to CO and reduce the risk of death.

If there is data for historical CO level, then CO level for the future can be predicted using time series. Time series analysis is the process of forecasting future data using historical data. In this project, the historical CO data for an Italian city is obtained on Kaggle; the goal is to predict the CO level for future time.

The time series and modeling process usually includes understanding the dataset using plots over time and autocorrelation plot. The time series will be checked for stationarity and decomposition. Stationarity is required in ARMA, ARIMA, and SARIMA models. If the data is not stationary, transformation need to be done to make it stationary. A few transformation methods are differencing, log transformation, and square root transformation. Time series decomposition can help visualize the trend and seasonality. Usually if there are trend and seasonality, ARIMA or SARIMA model can be used. ARIMA can take care of the trend but it cannot take care of the seasonality. SARIMA model can take care of both trend and seasonality.

Different models will be performed on the time series and the best model will be selected. The models to forecast time series can be multiple linear regression, Holt Winter, ARMA (ARIMA, SARIMA), and basic methods such as average, naïve, drift, simple exponential smoothing. After all models are derive, the best model can be selected using MSE, Q value, variance of prediction, and variance of forecast. For ARMA (ARIMA, SARIMA), the order and parameters need to be estimated and diagnostic testing need to be done on the model in order to know if the model is valid. Once the best model is selected, the model is used to forecast future data.

The objective of this project is to apply different models on the time series for CO level and select the best model to predict future CO level. The process is outlined below.

- Understand the dataset (description of the dataset)
- Check for stationarity
- Time series decomposition
- Feature selection
- Modeling
    - Multiple linear regression
    - Holt Winter method
    - Basic models: average, naïve, drift, simple exponential smoothing (SES)
    - ARMA (ARIMA, SARIMA) models
        - Order estimation
        - Parameter estimation
        - Diagnostic analysis
- Model selection
- Forecast function and h-step prediction
- Summary and conclusion

Description of Dataset

Source of dataset

The dataset is Air Quality of an Italian city, obtained from Kaggle. Here is the link to the dataset https://www.kaggle.com/aayushkandpal/air-quality-time-series-data-uci

Data description

The dataset has 9357 instances of hourly averaged responses from 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significant polluted area, at road level, within an Italian city. The data was collected from 10th March 2004 to 4th April 2005. Ground truth hourly averaged concentrations for CO, Non-Metallic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) were provided by a co-located reference certified analyzer.

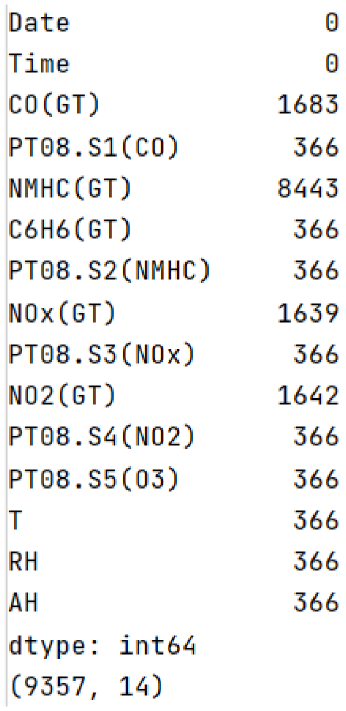
Attribute Information

0 Date (DD/MM/YYYY)
1 Time (HH.MM.SS)
2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)
3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
4 True hourly averaged overall Non-Metallic Hydrocarbons concentration in microg/m^3 (reference analyzer)

5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)
6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
7 True hourly averaged NOx concentration in ppb (reference analyzer)
8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)
10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
12 T: Temperature in °C
13 RH: Relative Humidity (%)
14 AH: Absolute Humidity

Data Preprocessing

When loading the dataset into python, there are 9471 rows. 114 rows at the end of the dataset have no date and time, so the 114 rows are dropped from the dataset. Because these data are at the end, dropping them will not affect the continuity of the timeline. The shape of the dataset after dropping them is (9357, 15). Missing values are represented as "-200", so "-200" is converted back to NaN in order to check for the number of missing values in the dataset.



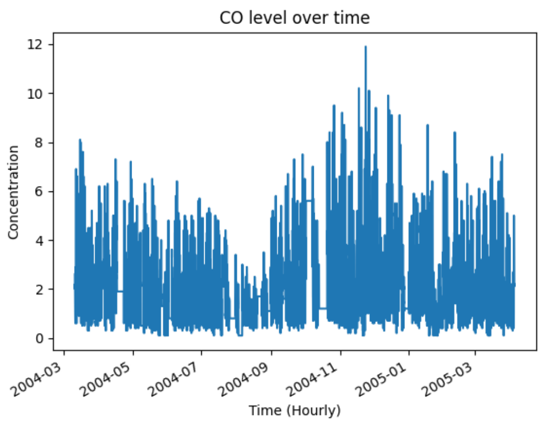
```
Date               0
Time               0
CO(GT)          1683
PT08.S1(CO)      366
NMHC(GT)        8443
C6H6(GT)         366
PT08.S2(NMHC)    366
NOx(GT)         1639
PT08.S3(NOx)     366
NO2(GT)         1642
PT08.S4(NO2)     366
PT08.S5(O3)      366
T                366
RH               366
AH               366
dtype: int64
(9357, 14)
```

Figure 1. Number of missing values in the dataset

The column NMHC(GT) is dropped because the number of missing values is too big (8443 out of 9357). The other missing values are imputed with the preceding values.

Data Visualization (plot of dependent variable, ACF, correlation matrix)

The dependent variable is CO(GT), and the rest of the features are the independent variables. To understand the time series, the plot of CO level over time and ACF plot of CO are shown below.



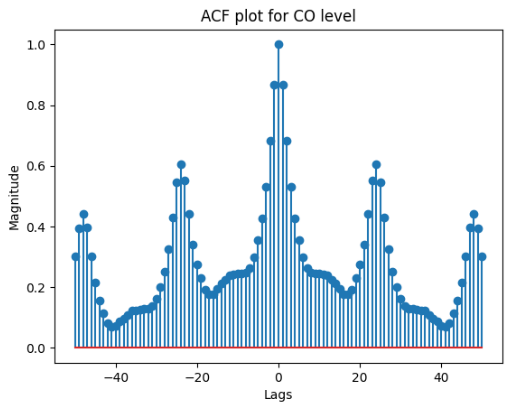


Figure 2. Plots of CO level over time and its ACF

There are peaks at lags 24 and 48, so there is seasonality in this time series which is daily (24 hours).

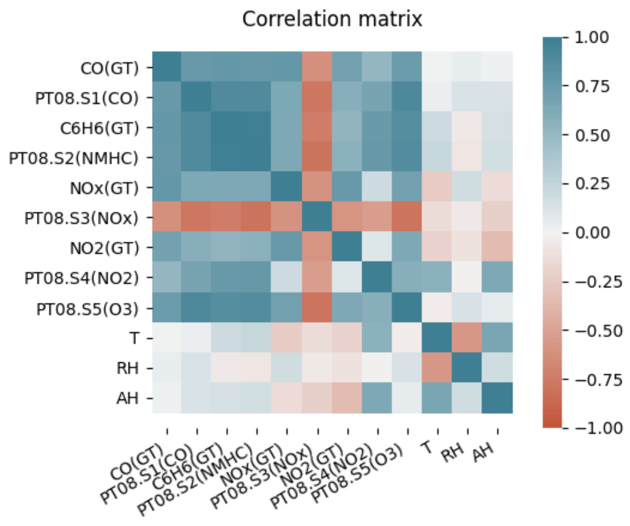


Figure 3. Correlation matrix of the dataset

Observing the correlation matrix, most features have positive correlation with CO, except PT08.S2(NMHC) which has a negative correlation with CO. Temperature, relative humidity, and absolute humidity have weak correlations with CO.

Stationarity

Stationarity is important in time series analysis, especially when building ARMA models. ARMA models require a stationary dataset. ADF test is used to check if a time series is stationary. If the time series is non-stationary, transformation is needed to make it stationary. Transformation can be differencing, log transformation, square root, or combination of different transformation methods. First, time series CO is checked for stationary using ADF test. The result is:

ADF Statistic: -9.423843

p-value: 0.000000

Critical Values:

  1%: -3.431

  5%: -2.862

  10%: -2.567

The p-value is less than 0.05 and the ADF statistic is also less than the critical values, so the CO time series is stationary.

Time Series Decomposition

Using STL, the CO time series components are decomposed as in the graph below.

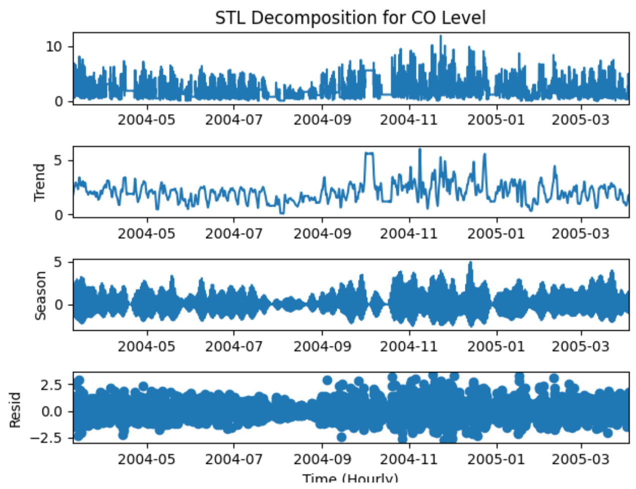
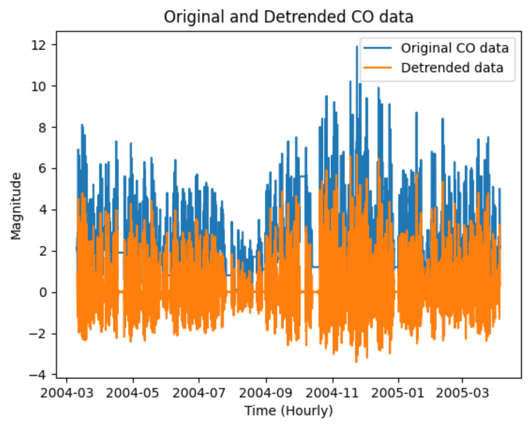


Figure 4. Time series decomposition

Figure 4 shows the decomposed components of CO time series. There are some trend and seasonality in the time series. The detrended data and seasonal adjusted data are going to be calculated.



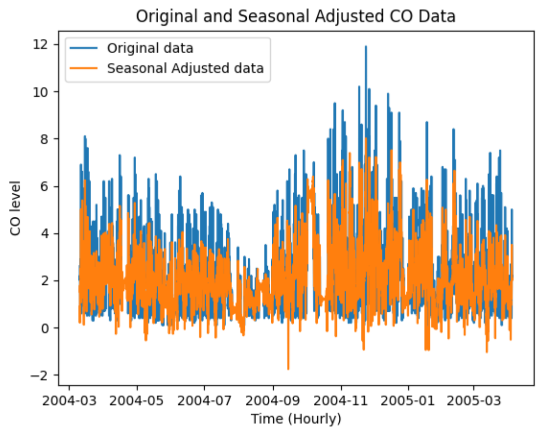


Figure 5. Detrended and Seasonal Adjusted Data

The detrended data is shown on the left of figure 5, and the seasonal-adjusted data is shown on the right. The strengths of trend and seasonality is calculated in order to know if the time series is highly trended or highly seasonal. The strength of trend for the CO time series is 0.7676. The strength of seasonality for the CO time series is 0.7562. CO time series has both trend and seasonality. The strengths for both trend and seasonality are about the same (around 0.76) and this number indicates a moderate trend and seasonality.

Modeling

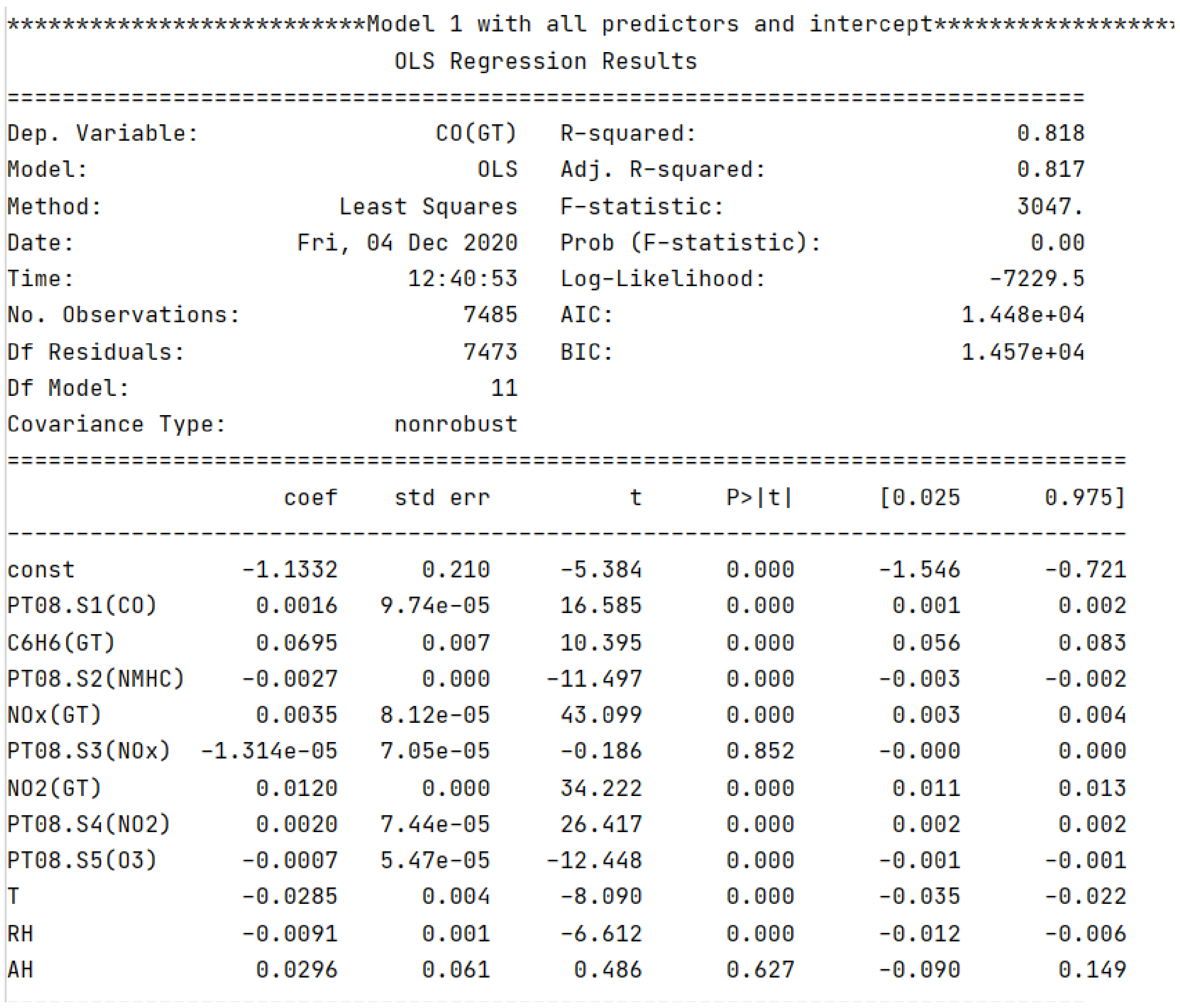
The data is split into 80% train set and 20% test set. Different models are built using the train set. After comparing different models, the best model will be selected to perform the forecast on the test set. The models used for training are Multiple Linear Regression, Holt Winters, basic methods (average, naïve, drift, and SES), ARMA, and SARIMA models.

1. Multiple Linear Regression

1.1 Feature Selection

In order to know which features are important to go into a multiple linear regression model, backward stepwise selection is performed. The first model includes all features and the intercept, then each feature is removed one by one until the adjusted R^2 is not improved anymore. In order to know which feature to eliminate, p-values of coefficients are used. Feature that has the highest p-value (insignificant coefficient) is removed from the model.

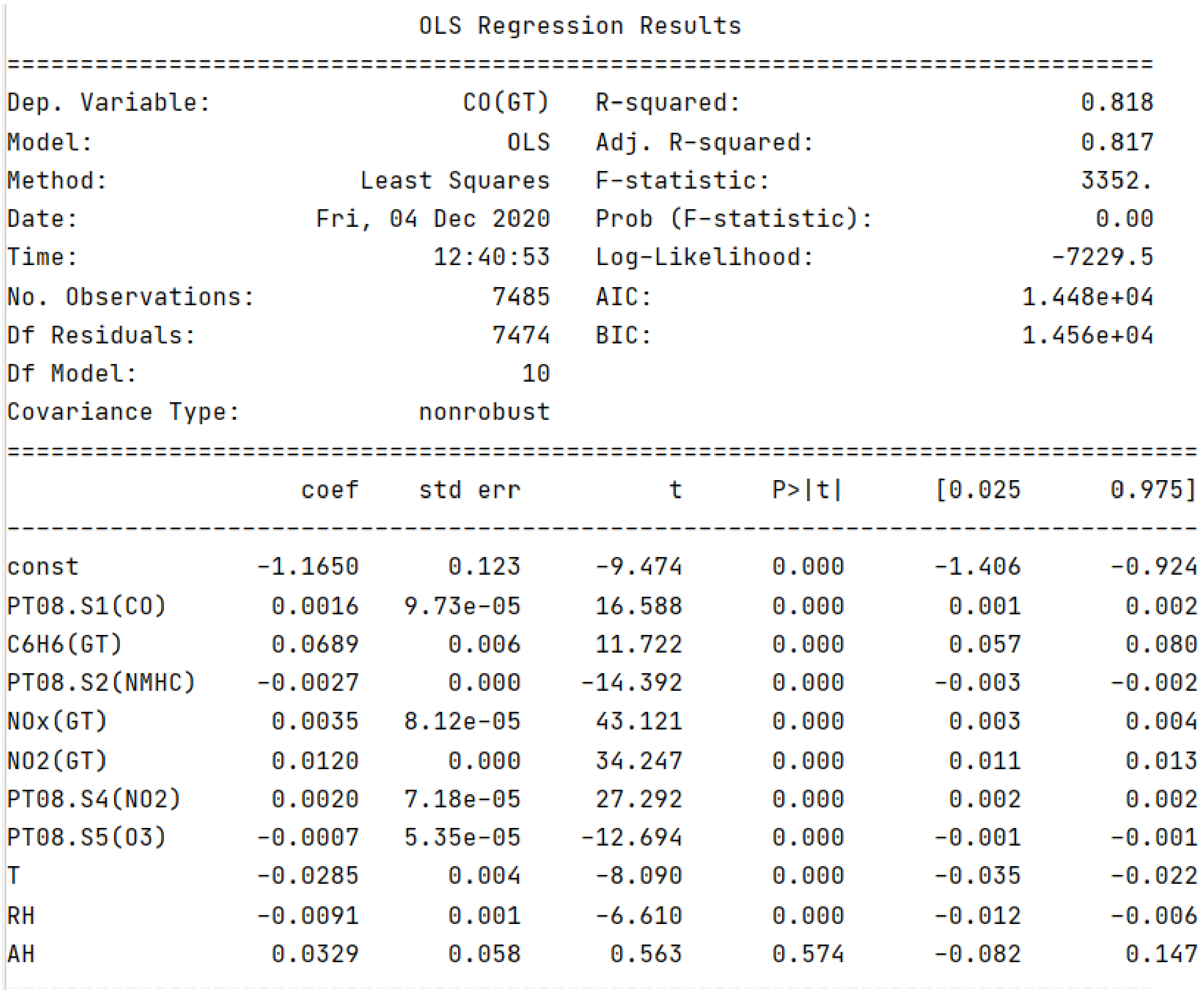
Model 1 has all the features and the intercept. The summary of model 1 is:



```
**************************Model 1 with all predictors and intercept*****************
                            OLS Regression Results
==============================================================================
Dep. Variable:                 CO(GT)   R-squared:                       0.818
Model:                            OLS   Adj. R-squared:                  0.817
Method:                 Least Squares   F-statistic:                     3047.
Date:                Fri, 04 Dec 2020   Prob (F-statistic):               0.00
Time:                        12:40:53   Log-Likelihood:                -7229.5
No. Observations:                7485   AIC:                         1.448e+04
Df Residuals:                    7473   BIC:                         1.457e+04
Df Model:                          11
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -1.1332      0.210     -5.384      0.000      -1.546      -0.721
PT08.S1(CO)       0.0016   9.74e-05     16.585      0.000       0.001       0.002
C6H6(GT)          0.0695      0.007     10.395      0.000       0.056       0.083
PT08.S2(NMHC)    -0.0027      0.000    -11.497      0.000      -0.003      -0.002
NOx(GT)           0.0035   8.12e-05     43.099      0.000       0.003       0.004
PT08.S3(NOx)  -1.314e-05   7.05e-05     -0.186      0.852      -0.000       0.000
NO2(GT)           0.0120      0.000     34.222      0.000       0.011       0.013
PT08.S4(NO2)      0.0020   7.44e-05     26.417      0.000       0.002       0.002
PT08.S5(O3)      -0.0007   5.47e-05    -12.448      0.000      -0.001      -0.001
T                -0.0285      0.004     -8.090      0.000      -0.035      -0.022
RH               -0.0091      0.001     -6.612      0.000      -0.012      -0.006
AH                0.0296      0.061      0.486      0.627      -0.090       0.149
```

All the coefficients are significant except for the variables PT08.S3(NOx) and AH. The variable PT08.S3(NOx) is removed from the model first because its p-value is higher.

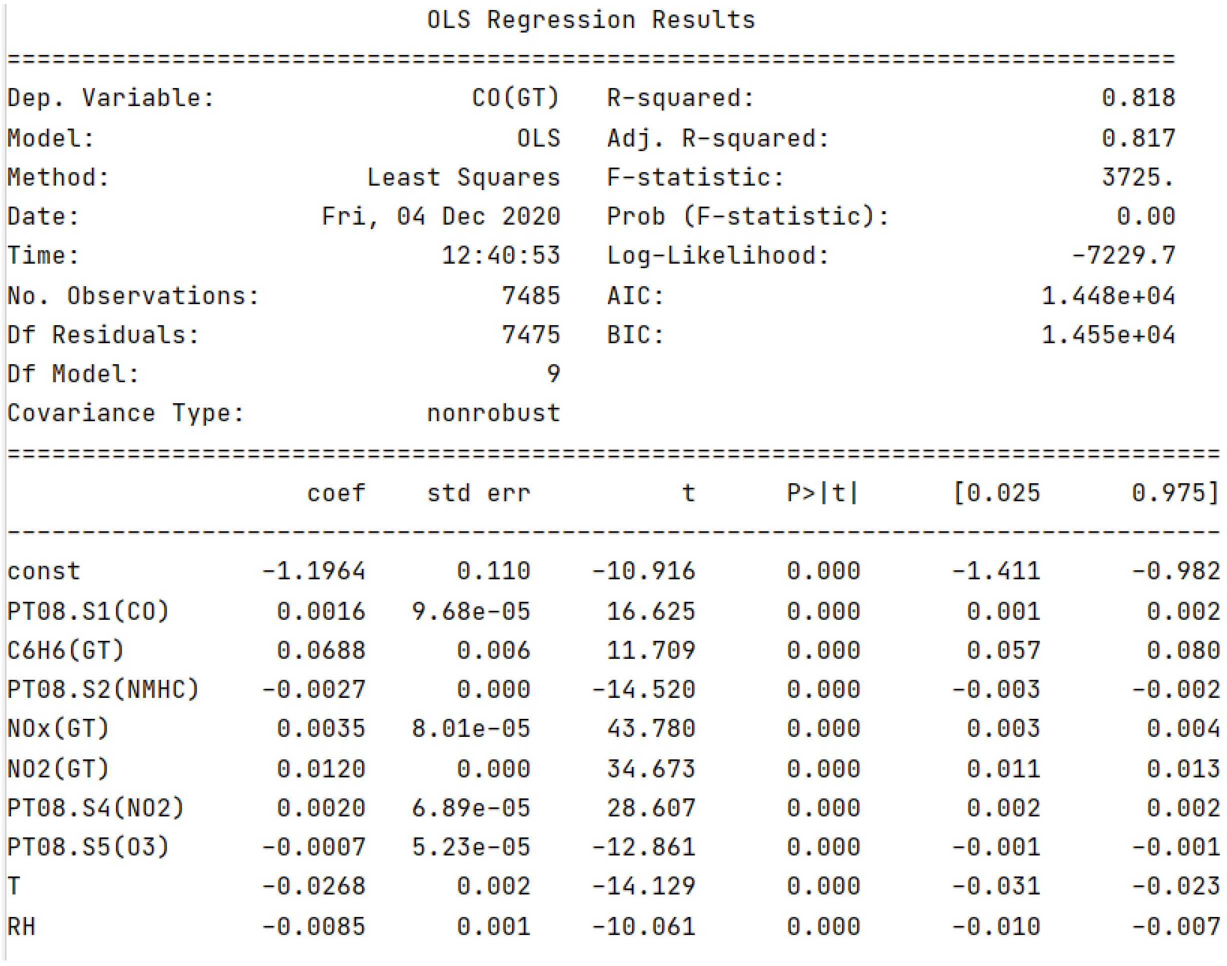
Model 2 with variable PT08.S3(NOx) removed



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  CO(GT)   R-squared:                       0.818
Model:                             OLS   Adj. R-squared:                  0.817
Method:                  Least Squares   F-statistic:                     3352.
Date:                 Fri, 04 Dec 2020   Prob (F-statistic):               0.00
Time:                         12:40:53   Log-Likelihood:                -7229.5
No. Observations:                 7485   AIC:                         1.448e+04
Df Residuals:                     7474   BIC:                         1.456e+04
Df Model:                           10
Covariance Type:             nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -1.1650      0.123     -9.474      0.000      -1.406      -0.924
PT08.S1(CO)       0.0016   9.73e-05     16.588      0.000       0.001       0.002
C6H6(GT)          0.0689      0.006     11.722      0.000       0.057       0.080
PT08.S2(NMHC)    -0.0027      0.000    -14.392      0.000      -0.003      -0.002
NOx(GT)           0.0035   8.12e-05     43.121      0.000       0.003       0.004
NO2(GT)           0.0120      0.000     34.247      0.000       0.011       0.013
PT08.S4(NO2)      0.0020   7.18e-05     27.292      0.000       0.002       0.002
PT08.S5(O3)      -0.0007   5.35e-05    -12.694      0.000      -0.001      -0.001
T                -0.0285      0.004     -8.090      0.000      -0.035      -0.022
RH               -0.0091      0.001     -6.610      0.000      -0.012      -0.006
AH                0.0329      0.058      0.563      0.574      -0.082       0.147
```

All the coefficients are significant except for AH, so AH will be removed in the next model.

Model 3 with variable AH removed



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  CO(GT)   R-squared:                       0.818
Model:                             OLS   Adj. R-squared:                  0.817
Method:                  Least Squares   F-statistic:                     3725.
Date:                 Fri, 04 Dec 2020   Prob (F-statistic):               0.00
Time:                         12:40:53   Log-Likelihood:                -7229.7
No. Observations:                 7485   AIC:                         1.448e+04
Df Residuals:                     7475   BIC:                         1.455e+04
Df Model:                            9
Covariance Type:             nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -1.1964      0.110    -10.916      0.000      -1.411      -0.982
PT08.S1(CO)       0.0016   9.68e-05     16.625      0.000       0.001       0.002
C6H6(GT)          0.0688      0.006     11.709      0.000       0.057       0.080
PT08.S2(NMHC)    -0.0027      0.000    -14.520      0.000      -0.003      -0.002
NOx(GT)           0.0035   8.01e-05     43.780      0.000       0.003       0.004
NO2(GT)           0.0120      0.000     34.673      0.000       0.011       0.013
PT08.S4(NO2)      0.0020   6.89e-05     28.607      0.000       0.002       0.002
PT08.S5(O3)      -0.0007   5.23e-05    -12.861      0.000      -0.001      -0.001
T                -0.0268      0.002    -14.129      0.000      -0.031      -0.023
RH               -0.0085      0.001    -10.061      0.000      -0.010      -0.007
```

All the coefficients now are significant. Adjusted R^2 and AIC stay the same, BIC is improved, so this model is good.

Table 1. Comparison table for the 3 regression models and select the best features

Model	**Feature removed from previous model**	**AIC**	**BIC**	**Adjusted R^2**
1 (all features and intercept)	N/A	14480	14570	0.817
2	PT08.S3(NOx)	14480	14560	0.817
3	AH	14480	14550	0.817

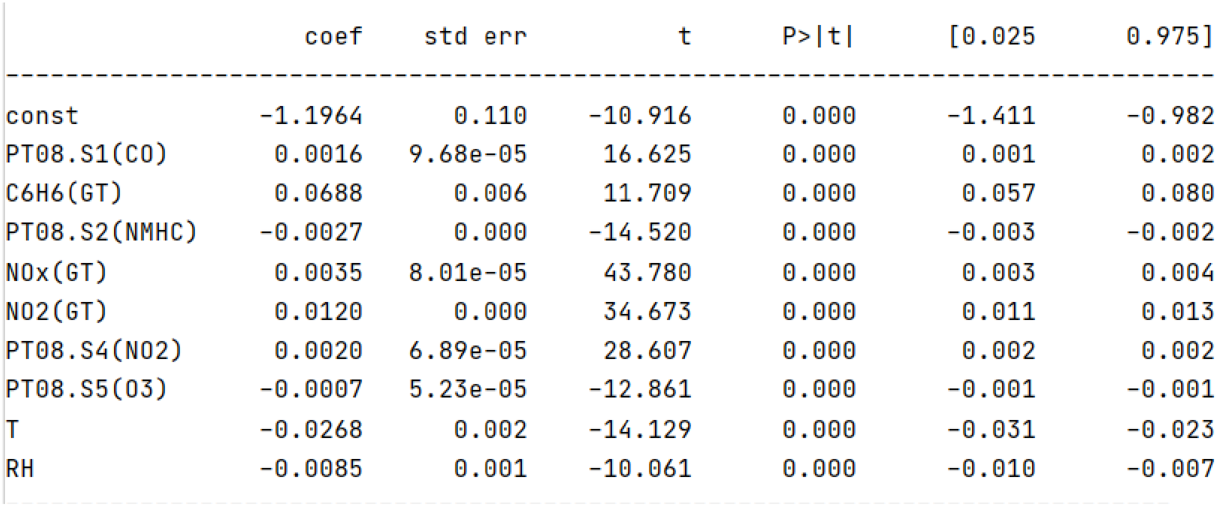
Model 3 is the best regression model because all the coefficients are significant. The model includes the intercept and features ['PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH']

1.2 Multiple Linear Regression Model

Now that the best features are selected, the multiple linear regression model can be used for training. First, the accuracy of the model is checked using F-test and t-test. The one-step prediction is performed on both train set and test set. The metrics such as MSE, AIC, BIC, adjusted R^2, ACF of residuals, variance, and mean are used to evaluate the model.

<u>t-test analysis</u>
The null hypothesis is the coefficients are 0.
The alternative hypothesis is the coefficients are different from 0.

From the summary of the best model, here are the t-statistics with p-values



```
                    coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const            -1.1964      0.110    -10.916      0.000      -1.411      -0.982
PT08.S1(CO)       0.0016   9.68e-05     16.625      0.000       0.001       0.002
C6H6(GT)          0.0688      0.006     11.709      0.000       0.057       0.080
PT08.S2(NMHC)    -0.0027      0.000    -14.520      0.000      -0.003      -0.002
NOx(GT)           0.0035   8.01e-05     43.780      0.000       0.003       0.004
NO2(GT)           0.0120      0.000     34.673      0.000       0.011       0.013
PT08.S4(NO2)      0.0020   6.89e-05     28.607      0.000       0.002       0.002
PT08.S5(O3)      -0.0007   5.23e-05    -12.861      0.000      -0.001      -0.001
T                -0.0268      0.002    -14.129      0.000      -0.031      -0.023
RH               -0.0085      0.001    -10.061      0.000      -0.010      -0.007
```

All the p-values are less than 0.05, so the null hypothesis is rejected. The coefficients are not 0. The coefficients are statistically important, and they do contribute to the model.

<u>F-test analysis</u>
The null hypothesis: The fit of the only-intercept model and the developed model are equal.
The alternative hypothesis: The fit of the only-intercept model is significantly reduced compared to the developed model.

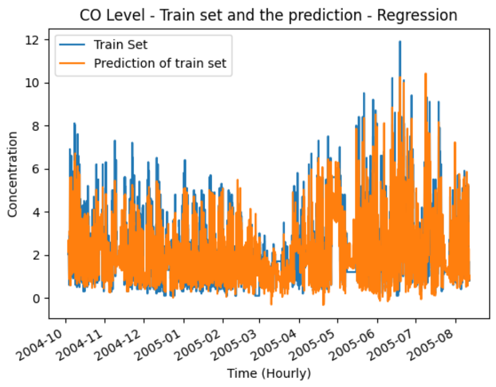
From the model summary, the F-statistic and its p-value are

```
F-statistic:                     3725.
Prob (F-statistic):               0.00
```

The p-value for F-statistic is 0.00, less than 0.05, so the null hypothesis is rejected. That means the developed model fits the data better than the only-intercept model.

The regression equation is y = -1.1964 + 0.0016*[PT08.S1(CO)] + 0.0688*[C6H6(GT)] – 0.0027*[PT08.S2(NMHC)] + 0.0035*[NOx(GT)] + 0.012*[NO2(GT)] + 0.002*[PT08.S4(NO2)] – 0.0007*[PT08.S5(O3)] – 0.0268*[T] – 0.0085*[RH]

<u>Perform the prediction on the train set and forecast the test set.</u>



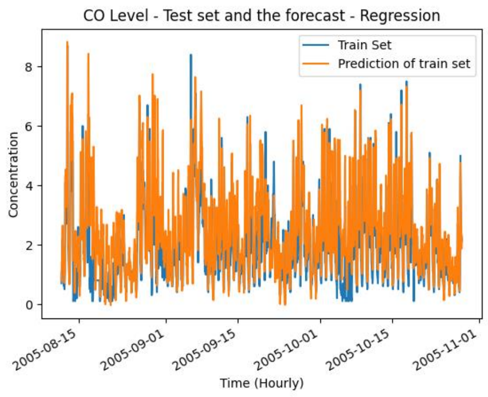


Figure 6. Plot of prediction of train set forecast of test set

Variance of prediction error is 0.4041

Variance of forecast error is 0.4584

The variance of prediction error is lower than the variance of the forecast error. The difference between them is not too much. This means the model performance on the train set is better but overall, prediction of train set and the test set are not too different.

Here are the metrics for linear regression method:

R-squared: 0.818
Adjusted R-squared: 0.817
AIC: 14480
BIC: 14450
MSE of residuals is 0.4041
MSE of forecast error is 0.7187
The MSE, AIC and BIC are very small, which is good because these are measure of errors, so the smaller they are, the better. Adjusted R-squared is high, that means 95.6% of variance in pH can be explained by independent variables.

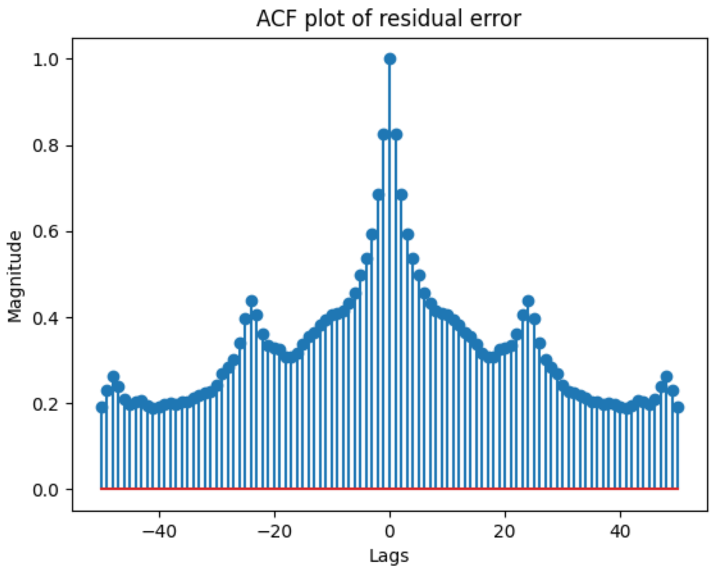


Figure 7. ACF of residual errors

Variance of prediction error is 0.4041

Mean of prediction error is 1.3952648136542355e-13 = 0

Q value of prediction error is 46288.

Even though the residual errors have 0 mean, but Q-value is very high. The ACF plot also shows that the residuals are not white noise, so the regression model does not capture all the information.

2. Holt Winters Method

Holt Winters is another method used in modeling that could capture seasonality. Figure 8 is the plot of forecast of CO level using Holt Winters method. The forecast looks different from the actual test data.

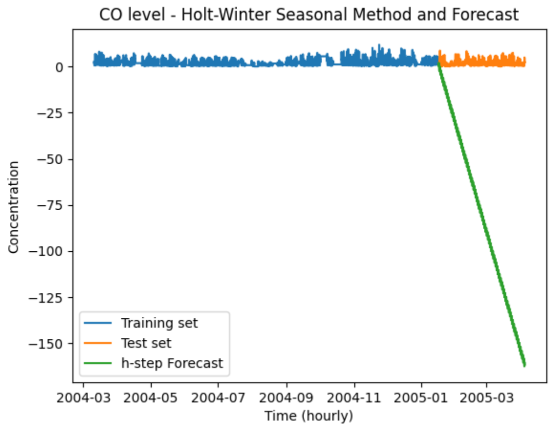
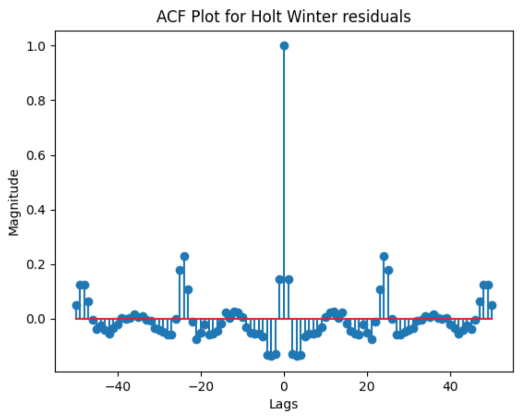


Figure 8. Plot of train set, test set, and forecast



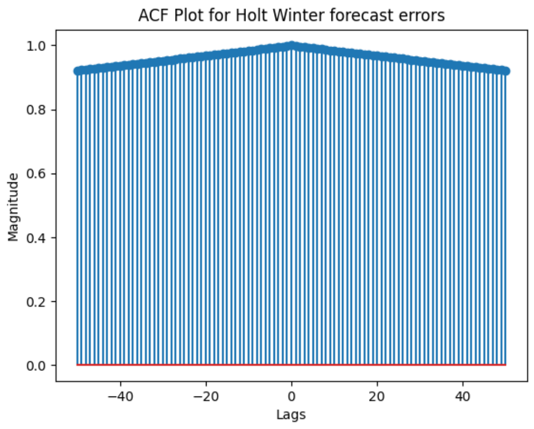


Figure 9. ACF plots of residual errors and forecast errors from Holt Winters method

Both ACF plots in figure 9 show that the residual errors and forecast errors are not white. The forecast errors in fact contains more information that has not been captured.

Variance of prediction error - Holt Winter method is 0.4615.

MSE of prediction error - Holt Winter method is 0.4615.

Mean of prediction error is -0.00035 = 0.

Q value of residual is 1965.

Variance of forecast error - Holt Winter method is 2201.

MSE of forecast error - Holt Winter method is 9017.

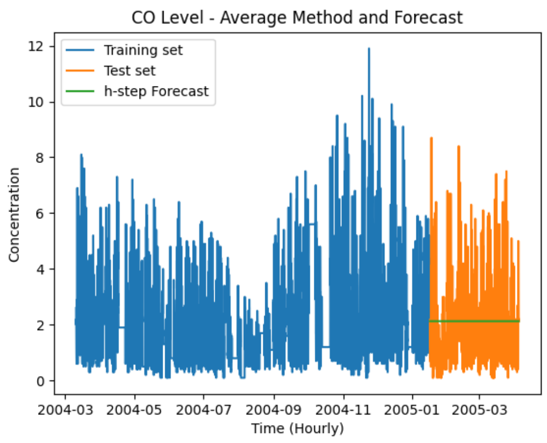
Q of forecast error is 86112.

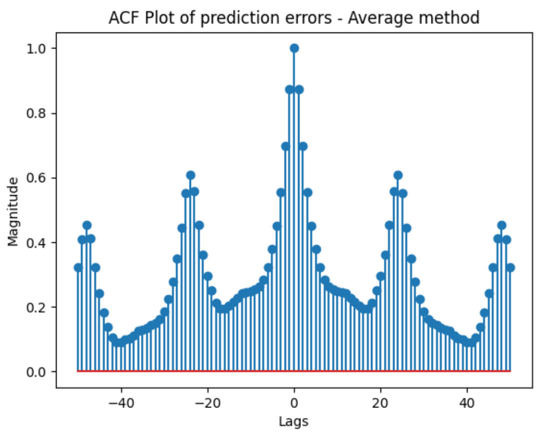
Even though the variance, MSE, and Q of prediction error are low, but the variance, MSE, and Q of forecast error are extremely high. This means Holt Winters method is not a good forecast method.

3. Basic models

3.1 Average method

In average method, the forecasts of all future values are equal to the average of the historical data.





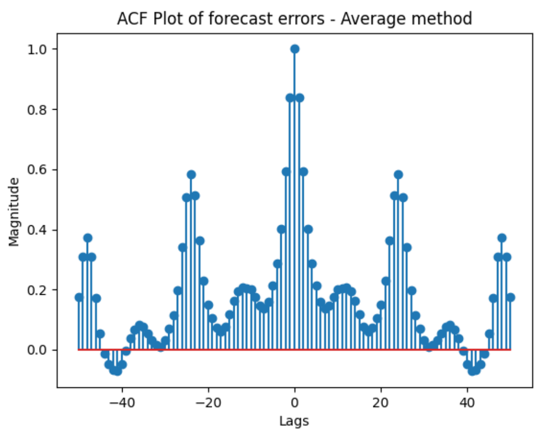


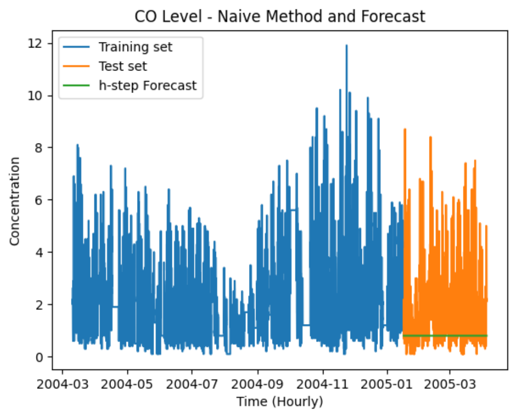
Figure10. Top plot: Forecast of average method. Bottom left: ACF of prediction errors. Bottom right: ACF of forecast errors

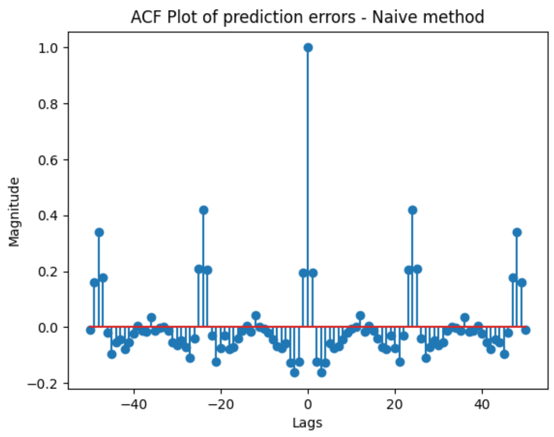
MSE of prediction errors: 2.2182	MSE of forecast errors: 1.9453
Variance of prediction error: 2.2109	Variance of forecast error: 1.8935
Mean of prediction errors is 0.0859	Mean of forecast errors is -0.2277
Q value of average method: 42390	Q of forecast error is 6245

The MSE, variance, and Q of prediction errors are higher than those from forecast errors, which means average method does not perform well on the train set.

3.2 Naïve method

In Naïve method, all forecasts are the value of the last observation.





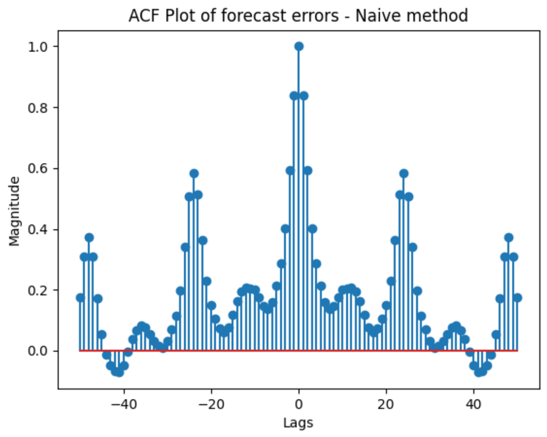


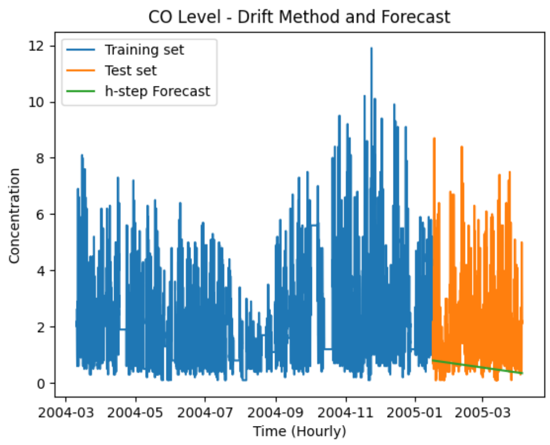
Figure 11. Top plot: Forecast of naïve method. Bottom left: ACF of prediction errors. Bottom right: ACF of forecast errors

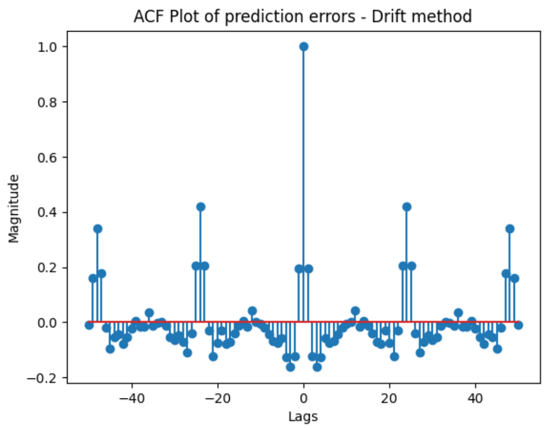
MSE of prediction errors: 0.5563	MSE of forecast errors: 3.1037
Variance of prediction error: 0.5563	Variance of forecast error: 1.8936
Mean of prediction errors is -0.00024	Mean of forecast errors is 1.1000
Q value of naive method: 4782	Q of forecast error is 6245

MSE, variance, and Q of prediction errors are smaller than those from forecast errors. This means naïve method performs well on the train set, but not forecast so well on the test set.

3.3 Drift method

Drift method is a variation of the naïve method where it allows the forecasts to increase or decrease over time, where the amount of change over time (the drift) is the average change seen in the historical data.





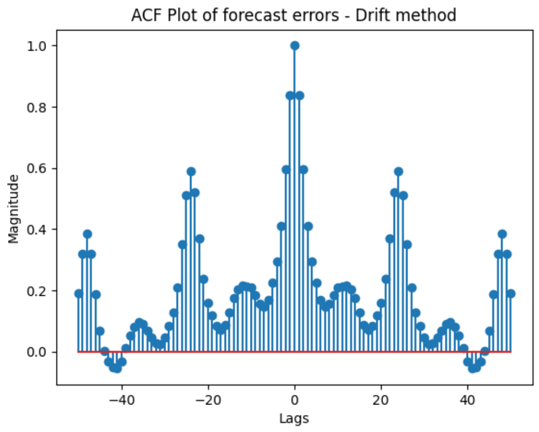


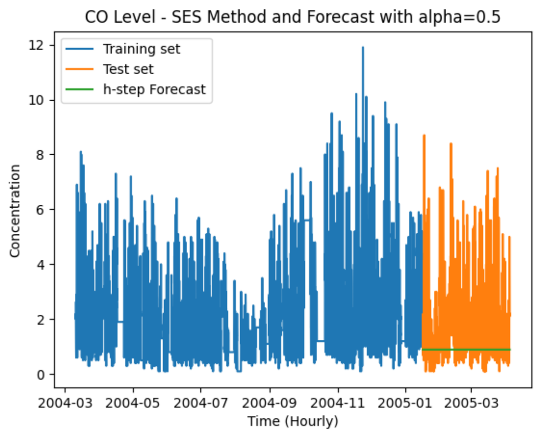
Figure 12. Top plot: Forecast of drift method. Bottom left: ACF of prediction errors. Bottom right: ACF of forecast errors

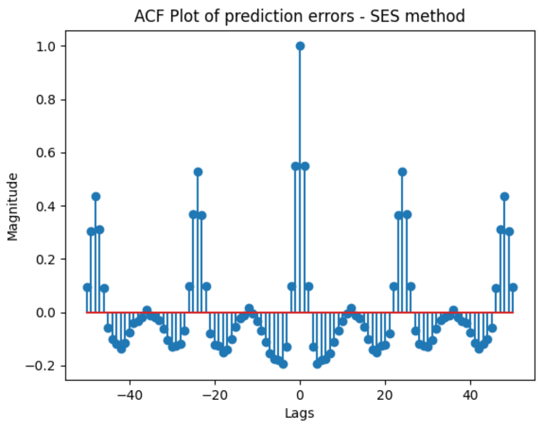
MSE of prediction errors: 0.5571	MSE of forecast errors: 3.6757
Variance of prediction error: 0.5571	Variance of forecast error: 1.9193
Mean of prediction errors is 0.0004	Mean of forecast errors is 1.3253
Q value of drift method: 4776	Q of forecast error is 6544

MSE, variance, and Q of prediction errors are smaller than those from forecast errors. This means drift method performs well on the train set, but not forecast so well on the test set.

3.4 Simple Exponential Smoothing (SES)

SES weights the averages where the weights decreases exponentially as observations come from further in the past. The smallest weights are associated with the oldest observations.





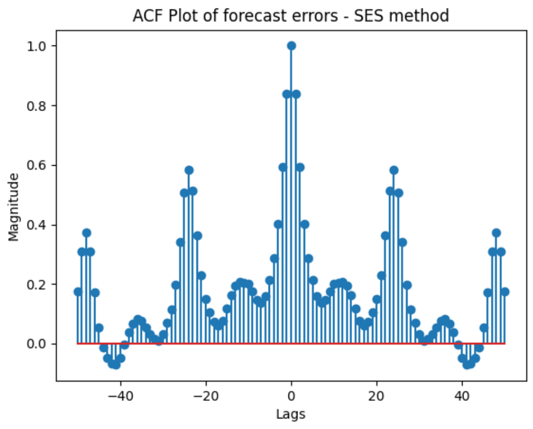


Figure 13. Top plot: Forecast of SES method. Bottom left: ACF of prediction errors. Bottom right: ACF of forecast errors

MSE of prediction errors: 0.7941	MSE of forecast errors: 2.9091
Variance of prediction error: 0.7941	Variance of forecast error: 1.8934
Mean of prediction errors is -0.0004	Mean of forecast errors is 1.0077
Q value of SES method: 12474	Q of forecast error is 6246

MSE and variance of prediction errors are smaller than those from forecast errors. This means SES method performs well on the train set, but not forecast so well on the test set.

4. ARMA model

4.1 Order Estimation

In order to determine the ARMA model's orders of na and nb, the concepts of partial correlation and partial autocorrelation need to be understood. Partial correlation is the correlation between two variables while excluding the effect of one or more independent variables. Similarly, for a stationary process, the partial autocorrelation (PACF) at lag k is the direct correlation between y_t and y_{t-k} with the removed linear dependence between the intermediate variables y_s with $t - k < s < t$. In other words, the partial autocorrelation between y_t and y_{t-h} is the conditional correlation between y_t and y_{t-h} excluding the effect of set of observations that come between the time point t and t – h.

PACF can be used to estimate the order of ARMA(n_a, n_b) model using the estimated autocorrelations. Then, the partial autocorrelation function will be derived and finally, from PACF, the Generalized partial autocorrelation function (GPAC) can be generated. The order of ARMA model, na and nb will be estimated.

When na = 0, then $R_y(\tau) = 0$ for all $\tau >$ nb. When nb = 0,

$$\left(\underbrace{\begin{matrix} \hat{R}_y(0) & \hat{R}_y(1) & \dots & \hat{R}_y(n_a-1) \\ \hat{R}_y(1) & \hat{R}_y(0) & \dots & \hat{R}_y(na-2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(n_a-1) & \hat{R}_y(n_a-2) & \dots & \hat{R}_y(0) \end{matrix}}_{\hat{\boldsymbol{X}}}\right)\left(\underbrace{\begin{matrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_{n_a} \end{matrix}}_{\hat{\boldsymbol{a}}}\right) = \left(\underbrace{\begin{matrix} \hat{R}_y(1) \\ \hat{R}_y(2) \\ \vdots \\ \hat{R}_y(n_a) \end{matrix}}_{\hat{\boldsymbol{Y}}}\right)$$

This approach is called Partial Autocorrelation (PAC). The above system of equations is called Yule-Walker Equations.

$$\boxed{\hat{\boldsymbol{a}} = \hat{\boldsymbol{X}}^{-1}\hat{\boldsymbol{Y}}}$$

When na = k, and $\hat{a}_{n_a}$ is replaced by ϕ_{kk}, we have

$$\phi_{kk} = \frac{\begin{vmatrix} \hat{R}_y(0) & \hat{R}_y(1) & \dots & \hat{R}_y(1) \\ \hat{R}_y(1) & \hat{R}_y(0) & \dots & \hat{R}_y(2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(k-1) & \hat{R}_y(k-2) & \dots & \hat{R}_y(k) \end{vmatrix}}{\begin{vmatrix} \hat{R}_y(0) & \hat{R}_y(1) & \dots & \hat{R}_y(k-1) \\ \hat{R}_y(1) & \hat{R}_y(0) & \dots & \hat{R}_y(k-2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(k-1) & \hat{R}_y(k-2) & \dots & \hat{R}_y(0) \end{vmatrix}}$$

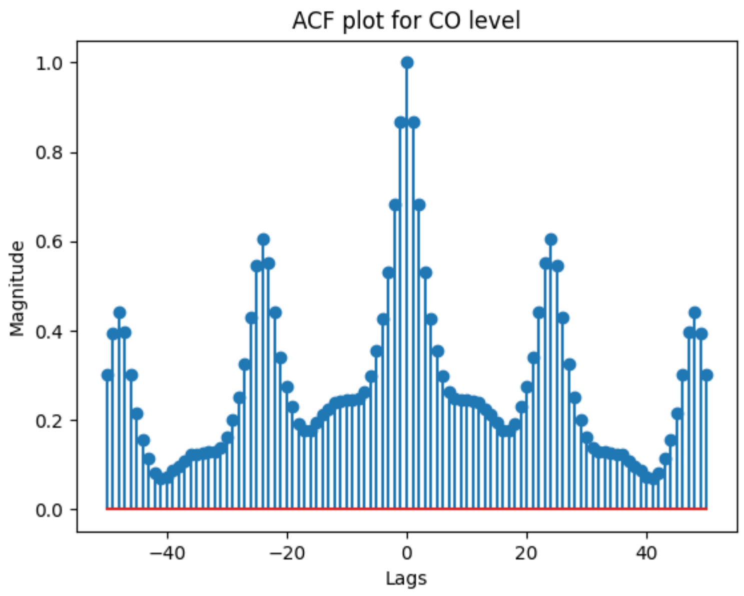
This is the partial autocorrelation function is 0 for all k > na if nb = 0.

The PAC only considers the case when nb = 0, so when $n_a \neq 0$ and $n_b \neq 0$, we need to consider the GPAC which is used to estimate the order of ARMA model when $n_a \neq 0$ and $n_b \neq 0$.

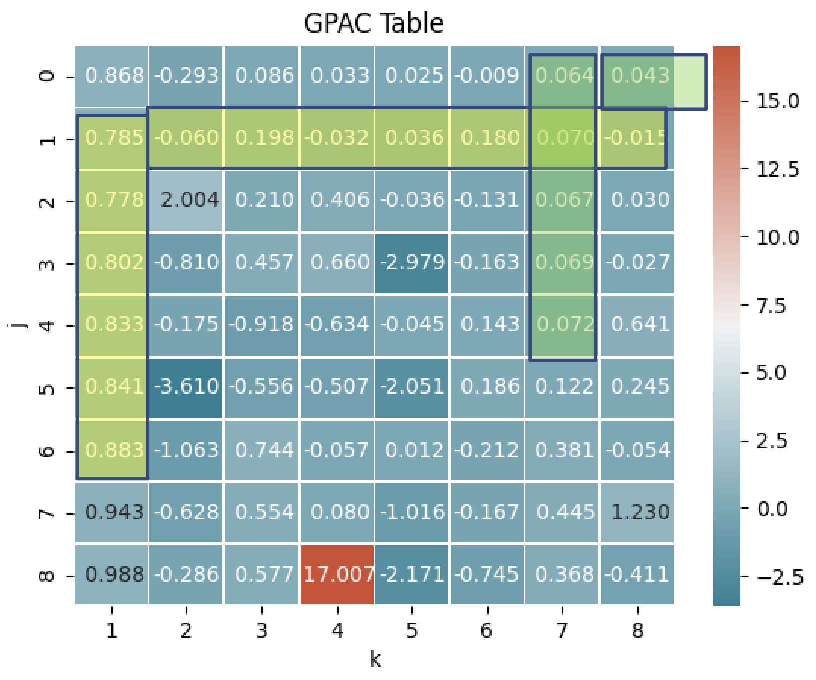
$$\phi_{kk}^{j} = \frac{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j+k) \end{vmatrix}}{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j-k+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j-k+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j) \end{vmatrix}}$$

If the autocorrelation is calculated, the autocorrelation values can be put into the formula above and phi can be calculated. After the phi value is computed, it will be put into GPAC table. The order of na is the column of constants, the order of na is the row of zeros.

After calculating the autocorrelation, here is the plot of ACF.



GPAC table for CO time series is shown below.



j \ k	1	2	3	4	5	6	7	8
0	0.868	-0.293	0.086	0.033	0.025	-0.009	0.064	0.043
1	0.785	-0.060	0.198	-0.032	0.036	0.180	0.070	-0.015
2	0.778	2.004	0.210	0.406	-0.036	-0.131	0.067	0.030
3	0.802	-0.810	0.457	0.660	-2.979	-0.163	0.069	-0.027
4	0.833	-0.175	-0.918	-0.634	-0.045	0.143	0.072	0.641
5	0.841	-3.610	-0.556	-0.507	-2.051	0.186	0.122	0.245
6	0.883	-1.063	0.744	-0.057	0.012	-0.212	0.381	-0.054
7	0.943	-0.628	0.554	0.080	-1.016	-0.167	0.445	1.230
8	0.988	-0.286	0.577	17.007	-2.171	-0.745	0.368	-0.411

Figure 14. GPAC table for ARMA order estimation

Possible orders are highlighted in GPAC table. There are two possible orders: ARMA(1,1) and ARMA(7,0).

4.2 Parameters Estimation, Prediction, and Diagnostic Analysis

ARMA(1,1)

The parameters of ARMA model are estimated using Levenberg Marquardt algorithm.

Estimated parameters: [-0.9379534303691117, 0.26989784301526953]

Variance: [[0.51367202]]

Standard deviation: 0.7167

```
Covariance of estimated parameters: [[   1.72951347e-05     1.27989992e-05]
                                     [1.27989992e-05       1.33394522e-04]]
```

<u>Check confidence intervals</u>

Confidence interval: -0.9462709143029704 < parameter 1 < -0.929635946435253

Confidence interval: 0.24679853373445435 < parameter 2 < 0.29299715229608475

Both coefficients are within the range of their confidence intervals. They are significant because there is no 0 in the confidence intervals.

Check zero/pole cancellation

Roots of numerator: [-0.26989784]

Roots of denominator: [0.93795343]

There is no zero/pole cancellation because the roots of numerator and denominator are different.

Prediction

After having the estimated parameters for ARMA(1,1), the derived equation is

y(t) - 0.938*y(t-1) = e(t) + 0.270*e(t-1)

1-step prediction is performed on the train set. The 1-step prediction function is

yhat(1) = E[0.938*y(t) + e(t+1) + 0.27*e(t)]

yhat(1) = 0.938*y(t) + 0.27*y(t) – 0.27*yhat(t|t-1)

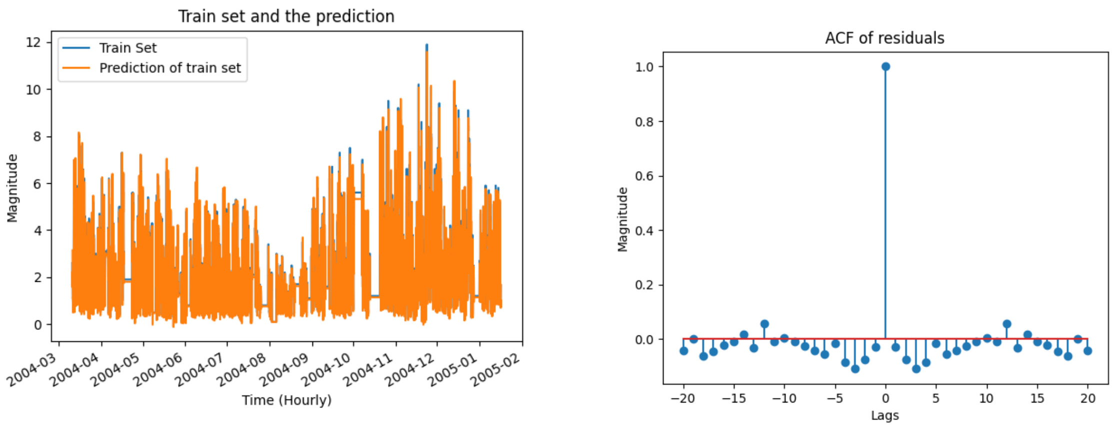


Figure 15. Prediction of train set and ACF of residual errors of ARMA(1,1)

MSE of residuals is 0.5127

Variance of prediction error is 0.5019

Mean of prediction error is 0.1037

Q value of residual is 325, which is larger than the chi critical value (34.8), so the residual is NOT white.

Because the mean of prediction errors is close to 0, so the derived model is unbiased. However, the residual error is still white noise, there are still information left.

H-step forecast is performed on the test set. Forecast function is

yhat(1) = 0.938*y(t) + 0.27*y(t) – 0.27*yhat(t|t-1)

yhat(2) = E[0.938*y(t+1) + e(t+2) + 0.27*e(t+1)] = 0.938*yhat(1)

And so on for the rest of the length of test set.

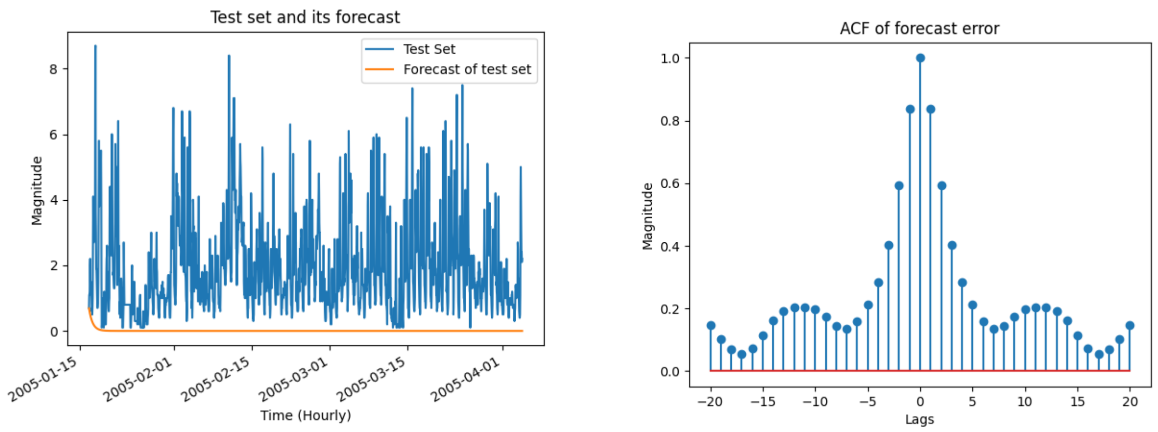


Figure 16. Forecast of test set and ACF of forecast errors of ARMA(1,1)

MSE of forecast error is 5.4835

Variance of forecast error is 1.8959

Mean of forecast error is 1.8941

When comparing the metrics from residuals and forecast errors of ARMA(1,1), the MSE and variance of prediction errors are much lower than the MSE and variance of forecast error, so this ARMA(1,1) model can perform well on the train set but not so well on the test set.

ARMA(7,0)

Estimated parameters: [-1.1749, 0.3564, -0.0593, -0.00194, -0.06689, 0.07505, -0.09679]

Variance: [[0.49691097]]

Standard deviation: 0.7049

Covariance:

```
[[ 1.32475699e-04     -1.56147464e-04      4.67944120e-05       -7.95664200e-06
  -1.01913575e-06     -4.34154549e-06      -5.16874579e-06]
[-1.56147464e-04    3.16325284e-04    -2.11477356e-04    5.61387636e-05
 -7.07572861e-06    5.93521499e-06     -4.34532096e-06]
[ 4.67944120e-05    -2.11477356e-04    3.32721097e-04    -2.14331598e-04
  5.55360083e-05    -7.09527459e-06    -1.00561811e-06]
[-7.95664200e-06    5.61387636e-05    -2.14331598e-04    3.33204005e-04
 -2.14353180e-04    5.61775910e-05    -7.97849578e-06]
[-1.01913575e-06    -7.07572861e-06    5.55360083e-05    -2.14353180e-04
  3.32770538e-04    -2.11544949e-04    4.68271530e-05]
[-4.34154549e-06    5.93521499e-06    -7.09527459e-06    5.61775910e-05
 -2.11544949e-04    3.16415543e-04    -1.56194121e-04]
[-5.16874579e-06    -4.34532096e-06    -1.00561811e-06    -7.97849578e-06
  4.68271530e-05    -1.56194121e-04    1.32506896e-04]]
```

<u>Check confidence intervals</u>

```
Confidence interval: -1.197913  < parameter 1 <  -1.151874
Confidence interval: 0.320796  < parameter 2 <  0.391938
Confidence interval: -0.095764  < parameter 3 <  -0.0228015
Confidence interval: -0.0384495  < parameter 4 <  0.0345659
Confidence interval: -0.103374  < parameter 5 <  -0.030406
Confidence interval: 0.039475  < parameter 6 <  0.110627
Confidence interval: -0.119815  < parameter 7 <  -0.073771
```

Parameter 4 is not significant because its confidence interval has 0, so we will omit this parameter from the model. The other parameters are within their confidence intervals and they are significant because there is no 0 in their confidence intervals.

<u>Check zero/pole cancellation</u>

```
Roots of numerator: [0. 0. 0. 0. 0. 0. 0.]
Roots of denominator: [ 0.97527205    0.58346295    0.58346295
 -0.53516437   -0.53516437   0.05151241    0.05151241]
```

There is no zero/pole cancellation because the roots of numerator and denominator are different.

Prediction

After having the estimated parameters for ARMA(7,0), the derived equation is

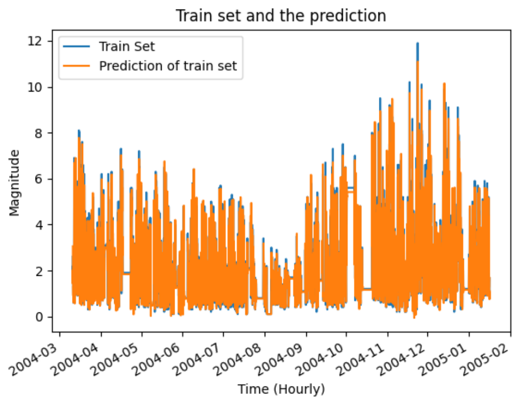
> y(t) - 1.17*y(t-1) + 0.356*y(t-2) - 0.0593*y(t-3) - 0.0669*y(t-5) + 0.0751*y(t-6) - 0.0968*y(t-7) = e(t)

Note: the term y(t-4) is omitted because the coefficient is not significant.

1-step prediction is performed on the train set. The 1-step prediction function is

yhat(1) = E[1.17*y(t) - 0.356*y(t-1) + 0.0593*y(t-2) + 0.0669*y(t-4) - 0.0751*y(t-5) + 0.0968*y(t-6) + e(t+1)]

yhat(1) = 1.17*y(t) - 0.356*y(t-1) + 0.0593*y(t-2) + 0.0669*y(t-4) - 0.0751*y(t-5) + 0.0968*y(t-6)



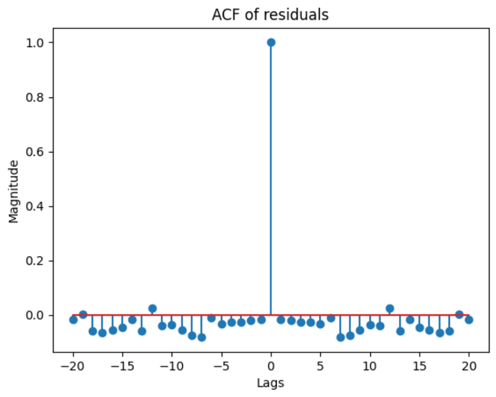


Figure 17. Prediction of train set and ACF of residual errors of ARMA(7,0)

MSE of residuals is 0.4956

Variance of prediction error is 0.4906

Mean of prediction error is 0.0712

Q value of residual is 286, greater than chi critical value (27.7), so the residual is NOT white.

Because the mean of prediction errors is close to 0, so the derived model is unbiased. The residuals errors are not white noise, so there is still information left in the dataset.

H-step forecast is performed on the test set. Forecast function is yhat(1) = 1.17*y(t) - 0.356*y(t-1) + 0.0593*y(t-2) + 0.0669*y(t-4) - 0.0751*y(t-5) + 0.0968*y(t-6)

yhat(2) = 1.17*yhat(1) - 0.356*y(t) + 0.0593*y(t-1) + 0.0669*y(t-3) - 0.0751*y(t-4) + 0.0968*y(t-5)

yhat(3) = 1.17*yhat(2) - 0.356*yhat(1) + 0.0593*y(t) + 0.0669*y(t-2) - 0.0751*y(t-3) + 0.0968*y(t-4)

yhat(4) = 1.17*yhat(3) - 0.356*yhat(2) + 0.0593*yhat(1) + 0.0669*y(t-1) - 0.0751*y(t-2) + 0.0968*y(t-3)

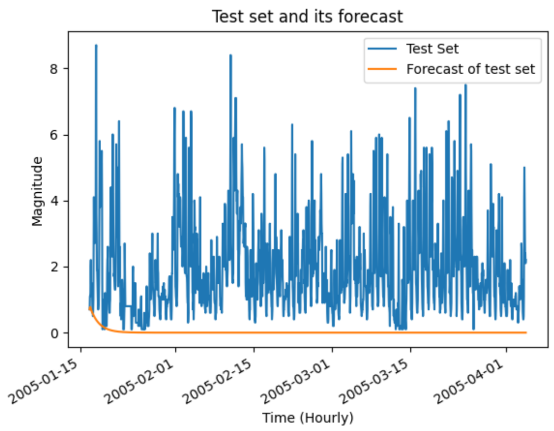
yhat(5) = 1.17*yhat(4) - 0.356*yhat(3) + 0.0593*yhat(2) + 0.0669*y(t) - 0.0751*y(t-1) + 0.0968*y(t-2)

yhat(6) = 1.17*yhat(5) - 0.356*yhat(4) + 0.0593*yhat(3) + 0.0669*yhat(1) - 0.0751*y(t) + 0.0968*y(t-1)

yhat(7) = 1.17*yhat(6) - 0.356*yhat(5) + 0.0593*yhat(4) + 0.0669*yhat(2) - 0.0751*yhat(1) + 0.0968*y(t)

yhat(8) = 1.17*yhat(7) - 0.356*yhat(6) + 0.0593*yhat(5) + 0.0669*yhat(3) - 0.0751*yhat(2) + 0.0968*yhat(1)

And so on for the rest of the length of test set.



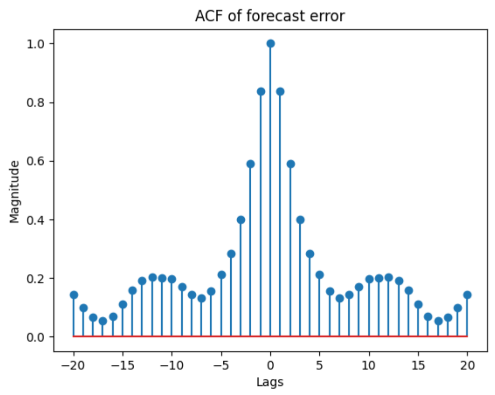


Figure 18. Forecast of test set and ACF of forecast errors of ARMA(7,0)

MSE of forecast error is 5.4398

Variance of forecast error is 1.8918

Mean of forecast error is 1.8836

When comparing the metrics from residuals and forecast errors of ARMA(7,0), the MSE and variance of prediction errors are much lower than the MSE and variance of forecast error, so this ARMA(7,0) model can perform well on the train set but not so well on the test set.

5. SARIMA model

ARMA model cannot take care of the trend or seasonality, but SARIMA model can. SARIMA(na,d,nb)(Na,D,Nb)m model needs to have its elements determined. The elements include:

- na: non-seasonal autoregressive order
- d: non-seasonal differencing order
- nb: non-seasonal moving average order
- m: seasonality of time series
- Na: seasonal autoregressive order
- D: seasonal differencing order
- Nb: seasonal moving average order

The time series is stationary, so d is 0 and D is also 0. The order of non-seasonal AR, MA and seasonal AR, MA can be determined using ACF and PACF.

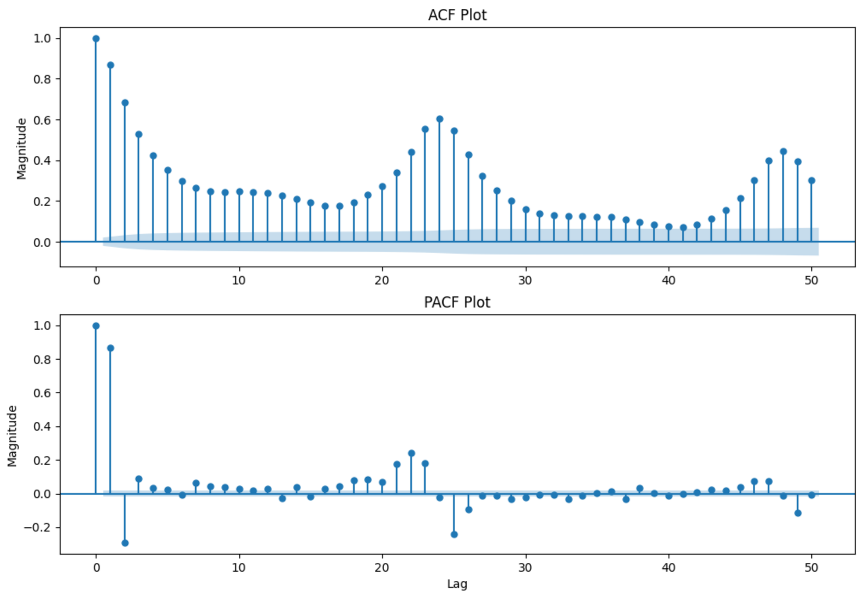
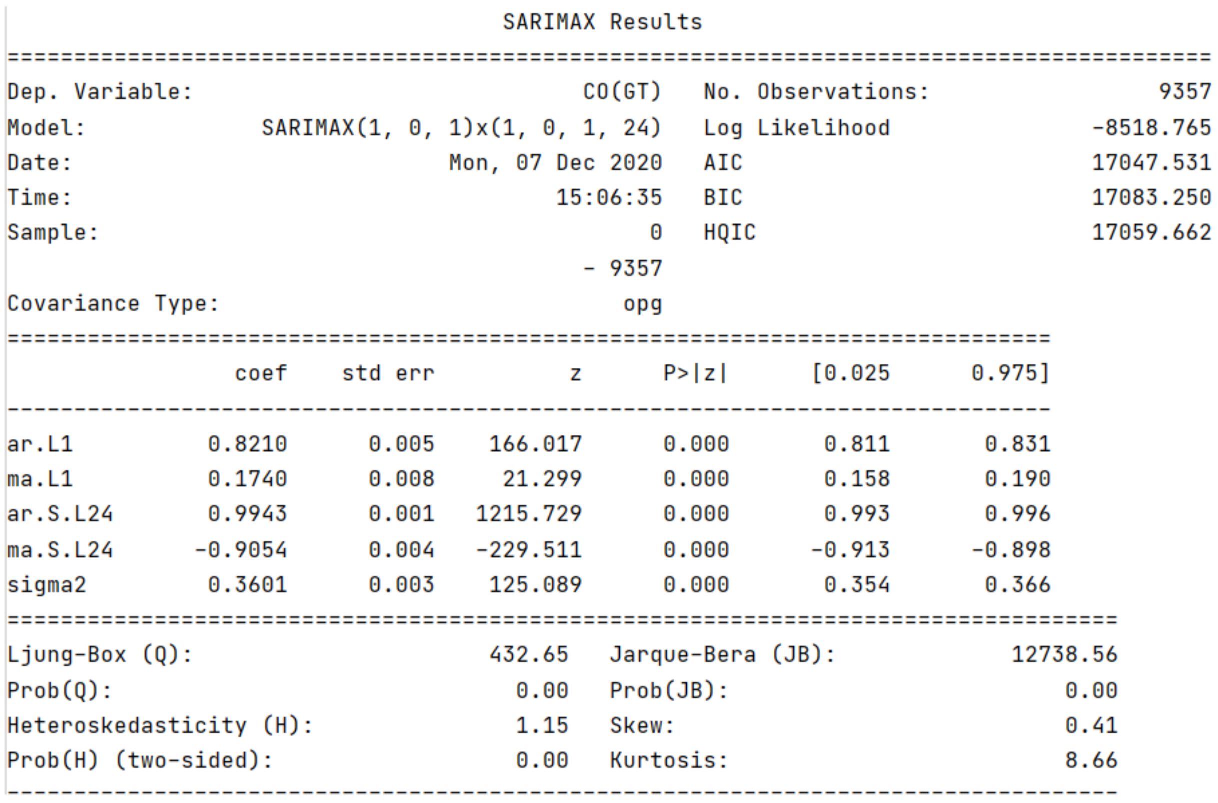


Figure 19. ACF and PACF of CO time series

The ACF plot tails of at lags 24, 48, … while the PACF is cut off after lags 12, 48. This indicate a seasonal AR(1)$_{24}$. The PACF also tails off at lags 12, 24, so seasonal MA is 1. The first significant peak is at lag 1 in ACF so the non-seasonal MA is 1. The first significant peak is at lag 1 of PACF and cut off afterward, so the order of non-seasonal AR is 1. The SARIMA model therefore could be SARIMA(1,0,1)(1,0,1)24. This model is fed into python package and below is the summary of the model.



```
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                              CO(GT)   No. Observations:                9357
Model:             SARIMAX(1, 0, 1)x(1, 0, 1, 24)   Log Likelihood               -8518.765
Date:                             Mon, 07 Dec 2020   AIC                          17047.531
Time:                                     15:06:35   BIC                          17083.250
Sample:                                          0   HQIC                         17059.662
                                            - 9357
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.8210      0.005    166.017      0.000       0.811       0.831
ma.L1          0.1740      0.008     21.299      0.000       0.158       0.190
ar.S.L24       0.9943      0.001   1215.729      0.000       0.993       0.996
ma.S.L24      -0.9054      0.004   -229.511      0.000      -0.913      -0.898
sigma2         0.3601      0.003    125.089      0.000       0.354       0.366
===================================================================================
Ljung-Box (Q):                      432.65   Jarque-Bera (JB):             12738.56
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.15   Skew:                             0.41
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.66
-----------------------------------------------------------------------------------
```

All coefficients are significant because their p-values are less than 0.05, the coefficients are within their confidence intervals and there is no 0 in the confidence intervals.

Prediction of train set

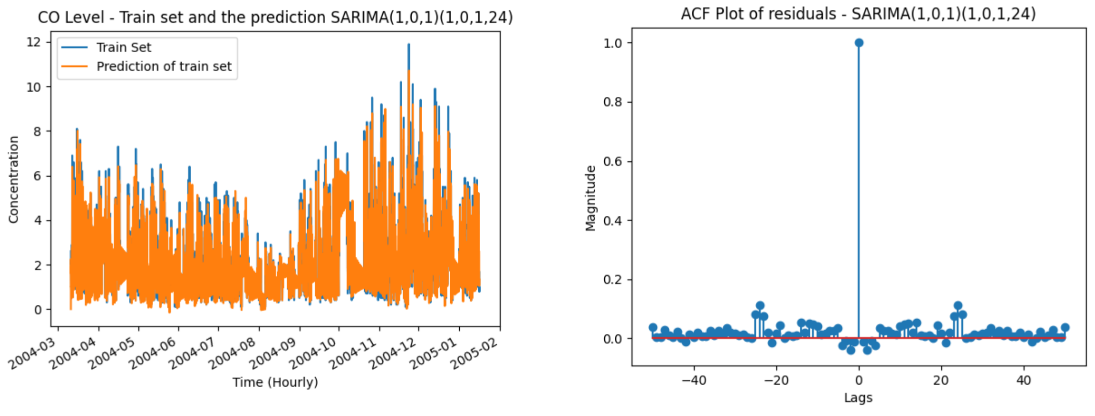


Figure 20. Prediction of train set and ACF of prediction errors

MSE of residuals is 0.3650

Variance of prediction error is 0.3645

Mean of prediction error is 0.0234

Q of prediction error is 374

Mean of prediction error is close to 0, so this model is unbiased. Q value is greater than chi_critical value (73.7) so the residual is not white noise.

Forecast of test set

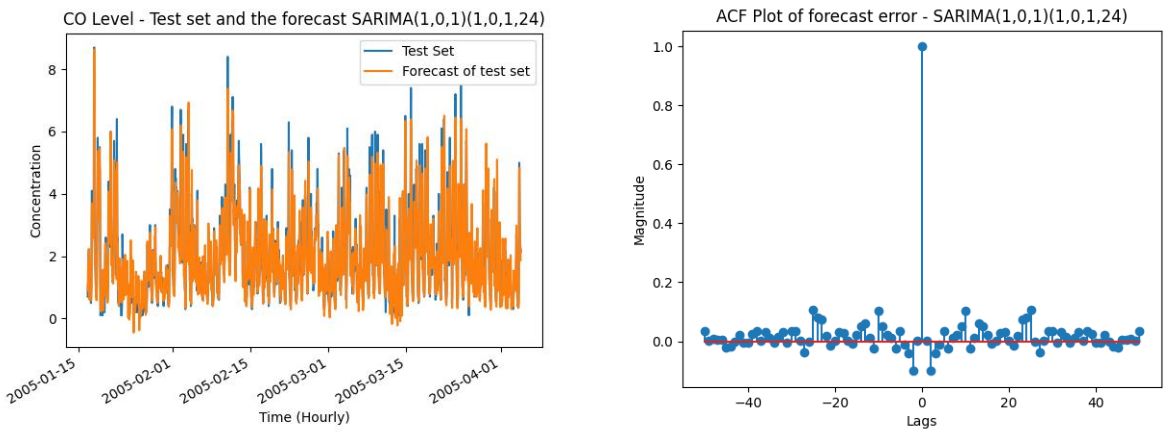


Figure 21. Forecast of test set and ACF of forecast errors

MSE of forecast is 0.3555

Variance of forecast error is 0.3555

Mean of forecast error is 0.0047

Q of forecast error is 135

The variance of prediction error and forecast error are very close to each other, so SARIMA (1,0,1)(1,0,1)24 model is performing well on both train set and test set.

Final Model Selection

Table 2 summarizes the metrics from all models for comparison.

Table 2. Summary of all models

	Q of residuals	MSE of residuals	Mean of residuals	Variance of residuals	Variance of forecast errors
Average method	42390	2.2182	0.0859	2.2109	1.8936
Naïve method	4782	0.5563	0	0.5563	1.8936
Drift method	4776	0.5571	0	0.5571	1.9193
SES method (alpha = 0.5)	12474	0.7941	0	0.7941	1.8936
Holt Winter	1965	0.4615	0	0.4615	2201
Regression	46288	0.4041	0	0.4041	0.4584
ARMA(1,1)	325	0.5127	0.1037	0.5019	1.8959
ARMA(7,0)	285	0.4956	0.0712	0.4906	1.8918
SARIMA(1,0,1)(1,0,1,24)	374	0.3650	0.0234	0.3645	0.3556

Out of 9 models, SARIMA(1,0,1)(1,0,1,24) is the best model because it has the lowest MSE of residual. Its mean is close to 0, the variance of residual error and forecast error are also the lowest. The difference between the variance of residual error and forecast error is very small, indicating that this model is performing well on both the train set and the test set. Therefore, the final model is SARIMA(1,0,1)(1,0,1,24).

Forecast Function and h-step prediction

Now the best model SARIMA has been selected, the forecast function can be developed.

The equation for SARIMA(1,0,1)(1,0,1,24) is

$(1 – aq^{-1})(1 – Aq^{-24})*y(t) = (1 – bq^{-1})(1 – Bq^{-24})*e(t)$

The coefficients are obtained from the SARIMA model summary in SARIMA section. When the coefficients for AR, MA, AR.S, and MA.S are substituted, the equation becomes

$(1 – 0.821q^{-1})(1 – 0.9943q^{-24})*y(t) = (1 – 0.174q^{-1})(1 + 0.9054q^{-24})*e(t)$

After calculations, the derived forecast function is:

$y(t) – 0.9943*y(t-24) – 0.821*y(t-1) + 0.8163*y(t-25) = e(t) + 0.9054*e(t-24) – 0.174*e(t-1) – 0.1575*e(t-25)$

h-step prediction

1-step

yhat(1) = E[0.9943*y(t-23) + 0.821*y(t) – 0.8163*y(t-24) + e(t+1) + 0.9054*e(t-23) – 0.174*e(t) – 0.1575*e(t-24)]

yhat(1) = 0.9943*y(t-23) + 0.821*y(t) – 0.8163*y(t-24) +0+ 0.9054*y(t-23) – 0.9054*yhat(t-23) – 0.174*y(t) + 0.174*yhat(t) – 0.1575*y(t-24) + 0.1575*yhat(t-24)

yhat(1) = 0.647*y(t) + 1.8997*y(t-23) – 0.9738*y(t-24) – 0.9054*yhat(t-23) + 0.174*yhat(t) + 0.1575y(t-24)

Perform the calculation in the same manner for multistep prediction until 25th step.

yhat(2) = 1.8997*y(t-22) – 0.9738*y(t-23) + 0.821*yhat(1) – 0.9054*yhat(t-22) + 0.1575*yhat(t-23)

yhat(3) = 1.8997*y(t-21) – 0.9738*y(t-22) + 0.821*yhat(2) – 0.9054*yhat(t-21) + 0.1575*yhat(t-22)

And this is kept going on until step 25

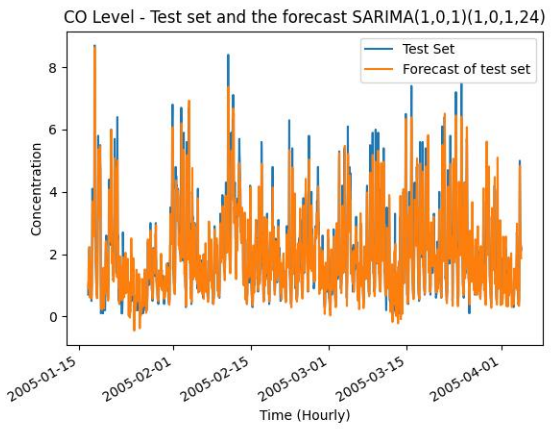
yhat(25) = 0.9943*yhat(1) + 0.821*yhat(24) – 0.9738*y(t) + 0.1575*yhat(t)

yhat(26) = 0.9943*yhat(2) + 0.821*yhat(25) – 0.8163*y(1)

And so on for the rest of the length of test set.

Multistep prediction needs to be performed on the test set.

Forecast of test set



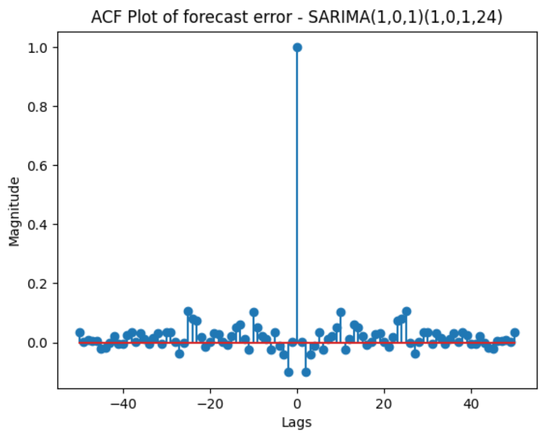


Figure 21. Forecast of test set and ACF of forecast errors

MSE of forecast is 0.3555

Variance of forecast error is 0.3555

Mean of forecast error is 0.0047

Q of forecast error is 135

The model is doing well on the test set. The forecast is similar to the test set when plotted together. Even though the forecast error is not completely white noise, but the seasonality in ACF plot has been reduced significantly. This is a big improve compared to other models. The MSE of forecast is very small. The variance of forecast error is small, and the mean is 0.

Summary and Conclusion

Different models were developed to forecast the CO level of an Italian city. The models are multiple linear regression, Holt Winters, basic methods (average, naïve, drift, SES), ARMA(1,1), ARMA(7,0), and SARIMA(1,0,1)(1,0,1,24). The best model is selected based on MSE, variance of residuals, mean of residuals, and variance of forecast errors. The final model is SARIMA(1,0,1)(1,0,1,24). The worst model is Holt Winter because the difference between variance of residuals and forecast errors is extremely large. Meanwhile, the SARIMA model can do very well on the train set and test set. That means it could be used for unseen data. The ACF plot shows that the residuals from SARIMA model is not completely white, but the seasonality has been reduced significantly compared to other models. Nonetheless, SARIMA is the best model among the models developed in this project. It still does not capture all the information, so the future work could be developing better models in order to have white noise residuals and better forecast. For example, grid search could be implemented to find the orders of ARMA and SARIMA models. It might suggest different orders and different SARIMA models could be developed. Additionally, this project focuses on linear models, so non-linear models could be developed and compared with SARIMA to select the model that performs better. In the scope of this project, the SARIMA(1,0,1)(1,0,1,24) is the best model to forecast CO level for outdoor air. This model is useful to give a forecast value for everyone to know and take precautions to limit their exposure to CO gas.

References

1. DATS 6540 Time Series Analysis & Modeling lecture notes.

2. (2019, October 16). *Carbon monoxide poisoning.* Mayoclinic. https://www.mayoclinic.org/diseases-conditions/carbon-monoxide/symptoms-causes/syc-20370642

3. Kandpal, A. (2020, September). Carbon monoxide poisoning. *Air Quality Time Series data UCI.* Kaggle. https://www.kaggle.com/aayushkandpal/air-quality-time-series-data-uci

4. Centers for Disease Control and Prevention. (2020, December 10). *Carbon Monoxide Poisoning.* CDC. https://www.cdc.gov/co/default.htm

5. United States Environmental Protection Agency. (2019, August 1). *Indoor Air Quality (IAQ)*. EPA. https://www.epa.gov/indoor-air-quality-iaq/what-average-level-carbon-monoxide-homes

6. Hatalis, K. (2018, April 12). *Multistep Forecasting with Seasonal ARIMA*. Data Science Central. https://www.datasciencecentral.com/profiles/blogs/tutorial-forecasting-with-seasonal-arima

Appendix

The python code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import STL
from Help import GPAC
from Autocorrelation import Cal_autocorr_fn, Cal_autocorr_with_plot_fn,
Cal_Q_fn
import statsmodels.api as sm
import matplotlib.dates as mdates
from statsmodels.tsa.stattools import acf
from sklearn.model_selection import train_test_split
import copy
from scipy import signal
import warnings
warnings.filterwarnings('ignore')


df = pd.read_csv("C:\\Users\\Trinh\\Desktop\\GWU\\Fall 2020\\Time
Series\\Final Project\\AirQualityUCI.csv",
                  header=0, parse_dates=['Date']) #parse_dates=[['Date',
'Time']], index_col=['Date_Time'])
print(df.head())
df = df.iloc[:, :-2]
print(df.shape)
print(df.describe())
print(df.info())
print(df.isna().sum())
# There are 114 rows at the end without date and time, let's drop them
df = df[:-114]
print(df.shape)
# NA values are written as -200. Let's change it back to NaN
df = df.replace(-200, np.nan)
#print(df.head(15))

print(df.isna().sum())
# 8443 values in NMHC(GT) is NaN, so let's drop the column
df = df.drop(columns = ['NMHC(GT)'])
print(df.shape)

# Fill in nan with forward fill (replace nan with preceding value)
df = df.fillna(method='ffill')
```

```
time = pd.date_range('2004-03-10 18:00:00', periods=len(df.Time), freq='h')
# Plot CO level over time
plt.figure()
plt.plot(time, df['CO(GT)'])
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO level over time')
plt.show()


lags = 50
# ACF plot
Cal_autocorr_with_plot_fn(df['CO(GT)'], lags, 'ACF plot for CO level')


# Correlation matrix
corr = df.corr()
ax = sns.heatmap(corr, vmin=-1, vmax=1, 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.gcf().autofmt_xdate()
plt.title('Correlation matrix')
plt.show()

# Split data into 80% train and 20% test
y = df['CO(GT)']
y_train, y_test = train_test_split(y, shuffle=False, test_size=0.2)
x_train, x_test = train_test_split(time, shuffle=False, test_size=0.2)


# Check for stationary
from Help import ADF_Cal
print(ADF_Cal(df['CO(GT)']))


# ******************TIME SERIES DECOMPOSITION************************
# STL decomposition
data = pd.Series(np.array(df['CO(GT)']), index=pd.date_range('2004-3-10',
periods=len(df['CO(GT)']), freq='h'), name='STL Decomposition for CO Level')
STL = STL(data)
res = STL.fit()
STL_plot = res.plot()
plt.xlabel('Time (Hourly)')
plt.show()

R = res.resid
T = res.trend
S = res.seasonal
# Plot trend, seasonal, and remainder in one plot
plt.figure()
plt.plot(time, T, label='Trend')
plt.plot(time, S, label='Seasonality')
plt.plot(time, R, label = 'Remainder')
plt.legend()
plt.xlabel('Time (Hourly)')
plt.ylabel('Magnitude')
plt.title('Components from STL decomposition')
plt.show()

# Calculate detrended data and plot it with original data
y_detrended = S + R
plt.figure()
plt.plot(time, data, label='Original CO data')
plt.plot(time, y_detrended, label='Detrended data')
plt.legend()
plt.xlabel('Time (Hourly)')
plt.ylabel('Magnitude')
plt.title('Original and Detrended CO data')
plt.show()

# Find seasonally adjusted data
seasonal_adj = T + R

# Plot seasonally adjusted data with original data
plt.figure()
plt.plot(time, data, label = 'Original data')
plt.plot(time, seasonal_adj, label='Seasonal Adjusted data')
plt.legend()
plt.title('Original and Seasonal Adjusted CO Data')
plt.xlabel('Time (Hourly)')
plt.ylabel('CO level')
plt.show()

# Calculate strength of trend
Ft = 1 - np.var(R)/np.var(T+R)
print('The strength of trend for the CO time series is', Ft)

# Calculate strength of seasonality
Fs = 1 - np.var(R)/np.var(S+R)
print('The strength of seasonality for the CO time series is', Fs)


##### Regression#####
X = df[['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 = df['CO(GT)']
# Split the dataset into 80% train and 20% test
X_train, X_test, y_train, y_test = train_test_split(X, Y, shuffle=False,
test_size=0.2)
time_train, time_test = train_test_split(time, shuffle=False, test_size=0.2)

X_train = sm.add_constant(X_train)
X_test = sm.add_constant((X_test))
model = sm.OLS(y_train, X_train).fit()

print('*************************Model 1 with all predictors and
intercept**********************')
print(model.summary())
print('PT08.S3(NOx) has the highest p-value (0.852), so we are going to
remove it from the model')
print('*****************************Model 2*****************************')

X_train = X_train[['PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)',
'NO2(GT)', 'PT08.S4(NO2)',
        'PT08.S5(O3)', 'T', 'RH', 'AH']]
X_test = X_test[['PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)',
'NO2(GT)', 'PT08.S4(NO2)',
        'PT08.S5(O3)', 'T', 'RH', 'AH']]
X_train = sm.add_constant(X_train)
X_test = sm.add_constant((X_test))
model2 = sm.OLS(y_train, X_train).fit()
forecast2 = model2.predict(X_test)

MSE_f_2 = np.square(np.subtract(y_test, forecast2)).mean()
print('MSE is', MSE_f_2)
print(model2.summary())

# Model 3 with 'AH' removed
print('***********************************Model
3***************************')
X_train = X_train[['PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)',
'NO2(GT)', 'PT08.S4(NO2)',
        'PT08.S5(O3)', 'T', 'RH']]
X_test = X_test[['PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)',
'NO2(GT)', 'PT08.S4(NO2)',
        'PT08.S5(O3)', 'T', 'RH']]
X_train = sm.add_constant(X_train)
X_test = sm.add_constant((X_test))
model3 = sm.OLS(y_train, X_train).fit()
forecast3 = model3.predict(X_test)

f_e = y_test - forecast3
MSE_f_3 = np.square(np.subtract(y_test, forecast3)).mean()
print('MSE of forecast error is', MSE_f_3)
print('Variance of forecast error is', np.var(f_e))
print(model3.summary())

# Residual error
prediction = model3.predict(X_train)
# Plot original data and prediction
plt.figure()
plt.plot(time_train, y_train, label='Train Set')
plt.plot(time_train, prediction, label='Prediction of train set')
plt.legend()
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Train set and the prediction - Regression')
plt.show()

e = y_train - prediction
print('MSE of residuals is', np.mean(np.square(e)))
print('Variance of prediction error is', np.var(e))
print('Mean of prediction error is', np.mean(e))
print('Q value of prediction error is', Cal_Q_fn(e, lags, len(y_train)))
# ACF Plot
Cal_autocorr_with_plot_fn(e, lags, 'ACF plot of residual error')

# Plot forecast with test set
plt.figure()
plt.plot(time_test, y_test, label='Train Set')
plt.plot(time_test, forecast3, label='Prediction of train set')
plt.legend()
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Test set and the forecast - Regression')
plt.show()

# ACF Plot of forecast errors
Cal_autocorr_with_plot_fn(f_e, lags, 'ACF Plot of forecast errors')


##### Holt Winter ######
lags = 50
# Split the data into 80% train and 20% test
y = df['CO(GT)']
y_train, y_test = train_test_split(y, shuffle=False, test_size=0.2)
x_train, x_test = train_test_split(time, shuffle=False, test_size=0.2)

# ******************************Holt-Winter
method*****************************

Holt_Winter = ets.ExponentialSmoothing(y_train, trend='add', seasonal='add',
seasonal_periods=24).fit()
Holt_Winter_pr = Holt_Winter.fittedvalues
Holt_Winter_forecast = Holt_Winter.forecast(len(y_test))

# Plot the Holt-Winter Seasonal Method and Forecast
plt.figure()
plt.plot(x_train, y_train, label = 'Training set')
plt.plot(x_test, y_test, label = 'Test set')
plt.plot(x_test, Holt_Winter_forecast, label = 'h-step Forecast')
plt.legend()
plt.xlabel('Time (hourly)')
plt.ylabel('Concentration')
plt.title('CO level - Holt-Winter Seasonal Method and Forecast')
plt.show()

# Residual errors
Holt_Winter_pr_e = y_train - Holt_Winter_pr
print('Variance of prediction error - Holt Winter method is',
np.var(Holt_Winter_pr_e))
Holt_Winter_MSE = np.mean(Holt_Winter_pr_e**2)
print('MSE of prediction error - Holt Winter method is', Holt_Winter_MSE)
print('Mean of prediction error is', np.mean(Holt_Winter_pr_e))
print('Q value of residual is', Cal_Q_fn(Holt_Winter_pr_e, lags,
len(y_train)))
```

```
# ACF plot of residual errors
Cal_autocorr_with_plot_fn(Holt_Winter_pr_e, lags, 'ACF Plot for Holt Winter
residuals')

# Forecast errors
Holt_Winter_f_e = y_test - Holt_Winter_forecast
print('Variance of forecast error - Holt Winter method is',
np.var(Holt_Winter_f_e))
# MSE of the forecast error
Holt_Winter_f_MSE = np.mean(Holt_Winter_f_e**2)
print('MSE of forecast error - Holt Winter method is', Holt_Winter_f_MSE)
print('Q of forecast error is', Cal_Q_fn(Holt_Winter_f_e, lags, len(y_test)))
Cal_autocorr_with_plot_fn(Holt_Winter_f_e, lags, 'ACF Plot for Holt Winter
forecast errors')



# *******************************AVERAGE METHOD******************************
print('*********************AVERAGE METHOD*****************************')
from Help import average_1step_prediction, average_hstep_forecast
prediction, pr_e, pr_e_sq, pr_MSE = average_1step_prediction(y_train)
forecast, forecast_e, forecast_e_sq, forecast_MSE =
average_hstep_forecast(y_train, y_test)

print('MSE of prediction errors:', pr_MSE)
print('Variance of prediction error:', np.var(pr_e))
print('Mean of prediction errors is', np.mean(pr_e))

# Plot the Average Method and Forecast
plt.figure()
plt.plot(x_train, y_train, label = 'Training set')
plt.plot(x_test, y_test, label = 'Test set')
plt.plot(x_test, forecast, label = 'h-step Forecast')
plt.legend()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Average Method and Forecast')
plt.show()

# ACF of prediction errors
Cal_autocorr_with_plot_fn(pr_e, lags, 'ACF Plot of prediction errors -
Average method')
# ACF of forecast errors
Cal_autocorr_with_plot_fn(forecast_e, lags, 'ACF Plot of forecast errors -
Average method')
# Q-value
print('Q value of residuals - average method:', Cal_Q_fn(pr_e, lags,
len(y_train)))

print('MSE of forecast errors:', forecast_MSE)
print('Variance of forecast error:', np.var(forecast_e))
print('Mean of forecast errors is', np.mean(forecast_e))
print('Q of forecast error is', Cal_Q_fn(forecast_e, lags, len(y_test)))


# *******************************NAIVE METHOD******************************
print('*********************NAIVE METHOD*****************************')
```

```
from Help import naive_1step_prediction, naive_hstep_forecast
naive_pr, naive_pr_e, naive_pr_e_sq, naive_pr_MSE =
naive_1step_prediction(y_train)
naive_forecast, naive_f_e, naive_f_e_sq_, naive_f_MSE =
naive_hstep_forecast(y_train, y_test)

print('MSE of prediction errors:', naive_pr_MSE)
print('Variance of prediction error:', np.var(naive_pr_e))
print('Mean of prediction errors is', np.mean(naive_pr_e))
print('Q value of naive method:', Cal_Q_fn(naive_pr_e, lags, len(y_train)))

# Plot the Naive Method and Forecast
plt.figure()
plt.plot(x_train, y_train, label = 'Training set')
plt.plot(x_test, y_test, label = 'Test set')
plt.plot(x_test, naive_forecast, label = 'h-step Forecast')
plt.legend()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Naive Method and Forecast')
plt.show()

# ACF of prediction errors
Cal_autocorr_with_plot_fn(naive_pr_e, lags, 'ACF Plot of prediction errors -
Naive method')
# ACF of forecast errors
Cal_autocorr_with_plot_fn(naive_f_e, lags, 'ACF Plot of forecast errors -
Naive method')

print('MSE of forecast errors:', naive_f_MSE)
print('Variance of forecast error:', np.var(naive_f_e))
print('Mean of forecast errors is', np.mean(naive_f_e))
print('Q of forecast error is', Cal_Q_fn(naive_f_e, lags, len(y_test)))


# *******************************DRIFT METHOD******************************
print('**********************DRIFT METHOD******************************')
from Help import drift_1step_prediction, drift_hstep_forecast
drift_pr, drift_pr_e, drift_pr_e_sq, drift_pr_MSE =
drift_1step_prediction(y_train)
drift_forecast, drift_f_e, drift_f_e_sq_, drift_f_MSE =
drift_hstep_forecast(y_train, y_test)

print('MSE of prediction errors:', drift_pr_MSE)
print('Variance of prediction error:', np.var(drift_pr_e))
print('Mean of prediction errors is', np.mean(drift_pr_e))
print('Q value of drift method:', Cal_Q_fn(drift_pr_e, lags, len(y_train)))

# Plot the Drift Method and Forecast
plt.figure()
plt.plot(x_train, y_train, label = 'Training set')
plt.plot(x_test, y_test, label = 'Test set')
plt.plot(x_test, drift_forecast, label = 'h-step Forecast')
plt.legend()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Drift Method and Forecast')
plt.show()

# ACF of prediction errors
Cal_autocorr_with_plot_fn(drift_pr_e, lags, 'ACF Plot of prediction errors -
Drift method')
# ACF of forecast errors
Cal_autocorr_with_plot_fn(drift_f_e, lags, 'ACF Plot of forecast errors -
Drift method')

print('MSE of forecast errors:', drift_f_MSE)
print('Variance of forecast error:', np.var(drift_f_e))
print('Mean of forecast errors is', np.mean(drift_f_e))
print('Q of forecast error is', Cal_Q_fn(drift_f_e, lags, len(y_test)))


# *******************************SES METHOD******************************
print('**********************SES METHOD******************************')
from Help import SES_1step_prediction, SES_hstep_forecast
SES_pr, SES_pr_e, SES_pr_e_sq, SES_pr_MSE = SES_1step_prediction(y_train,
alpha=0.5)
SES_forecast, SES_f_e, SES_f_e_sq_, SES_f_MSE = SES_hstep_forecast(y_train,
y_test, alpha=0.5)

print('MSE of prediction errors:', SES_pr_MSE)
print('Variance of prediction error:', np.var(SES_pr_e))
print('Mean of prediction errors is', np.mean(SES_pr_e))
print('Q value of SES method:', Cal_Q_fn(SES_pr_e, lags, len(y_train)))

# Plot the SES Method and Forecast
plt.figure()
plt.plot(x_train, y_train, label = 'Training set')
plt.plot(x_test, y_test, label = 'Test set')
plt.plot(x_test, SES_forecast, label = 'h-step Forecast')
plt.legend()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - SES Method and Forecast with alpha=0.5')
plt.show()

# ACF of prediction errors
Cal_autocorr_with_plot_fn(SES_pr_e, lags, 'ACF Plot of prediction errors -
SES method')
# ACF of forecast errors
Cal_autocorr_with_plot_fn(SES_f_e, lags, 'ACF Plot of forecast errors - SES
method')

print('MSE of forecast errors:', SES_f_MSE)
print('Variance of forecast error:', np.var(SES_f_e))
print('Mean of forecast errors is', np.mean(SES_f_e))
print('Q of forecast error is', Cal_Q_fn(SES_f_e, lags, len(y_test)))



#### ARMA(1,1) #####
# Estimate the order of ARMA
ry = Cal_autocorr_fn(df['CO(GT)'], lags)
acf_list = list(ry[::-1]) + list(ry[1:])
```

```
GPAC(acf_list, lags, 8, 8)
# Possible order set: (na, nb) = (1,1), (7,0)

mu_factor = 10
lags = 20
delta = 1e-6
epsilon = 0.001
dt = 1
mu = 0.01
mu_max = 1e10

na = 1
nb = 1
n = na + nb

theta = [0]*(n)

# Levenberg Marquardt Algorithm
# Step 1
def step1(data, theta, na, nb):
    max_order = max(na, nb)
    num = [0]*(max_order+1)
    den = [0]*(max_order+1)
    for i in range(na+1):
        if i==0:
            den[i] = 1
        else:
            den[i] = theta[i-1]

    for i in range(nb+1):
        if i==0:
            num[i] = 1
        else:
            num[i] = theta[na+i-1]

    system1 = (den, num, 1)
    _, e = signal.dlsim(system1, data)
    SSE = np.transpose(e).dot(e)

    X = []
    for i in range(n):
        theta_update = copy.copy(theta)
        theta_update[i] = theta_update[i] + delta
        for i in range(na + 1):
            if i == 0:
                den[i] = 1
            else:
                den[i] = theta_update[i - 1]

        for i in range(nb + 1):
            if i == 0:
                num[i] = 1
            else:
                num[i] = theta_update[na + i - 1]

        system = (den, num, 1)
```

```
            _, e_new = signal.dlsim(system, data)

            xi = (e - e_new) / delta
            X.append(xi)

        X = np.array(X)
        X = np.array([X[i].flatten() for i in range(len(X))]).T
        A = np.transpose(X).dot(X)
        g = np.transpose(X).dot(e)

        return SSE, X, A, g

    # Step 2
    def step2(data, A, g, theta, mu):
        I = np.identity(n)
        delta_theta = np.linalg.inv(A + mu*I).dot(g)
        delta_theta = np.reshape(delta_theta, (n,))
        theta_new = theta + delta_theta
        theta_new = theta_new.tolist()
        SSE_new, X_new, A_new, g_new = step1(data, theta_new, na, nb)

        if np.isinf(SSE_new):
            SSE_new = SSE
        if np.isnan(SSE_new):
            SSE_new = SSE*(10**8)

        return delta_theta, theta_new, SSE_new

    # Compute steps 1 and 2
    SSE, X, A, g = step1(y_train, theta, na, nb)
    delta_theta, theta_new, SSE_new = step2(y_train, A, g, theta, mu)

    # Step 3

    max_iter = 50
    i = 0
    iteration_list = []
    SSE_list = []

    iteration_list.append(i) # initial iteration 0
    SSE_list.append(SSE_new) # SSE at initial iteration 0

    while i < max_iter:
        if SSE_new < SSE:
            if np.linalg.norm(delta_theta) < epsilon:
                theta = theta_new
                variance = SSE_new/(len(y_train)-n)
                covariance = variance*(np.linalg.inv(A))
                print('Estimated parameters:', theta)
                print('Variance:', variance)
                print('Covariance:', covariance)
                break
            else:
                theta = theta_new

        while SSE_new >= SSE:
            mu = mu*10
            delta_theta, theta_new, SSE_new = step2(y_train, A, g, theta, mu)
            if mu > mu_max:
                print(SSE_new)
                print('mu > mu_max. Check your code.')
                break


        i += 1
        iteration_list.append(i)
        if i > max_iter:
            print('i > 50. Check your code.')
            break

        theta = theta_new
        mu = mu / 10
        SSE, X, A, g = step1(y_train, theta, na, nb)
        delta_theta, theta_new, SSE_new = step2(y_train, A, g, theta, mu)
        SSE_list.extend(SSE_new)

    # Plot SSE with iterations
    plt.figure()
    plt.plot(iteration_list, SSE_list)
    plt.xlabel('Number of Iterations')
    plt.ylabel('SSE')
    plt.title('SSE vs number of iterations')
    plt.show()

    # Confidence interval
    for i in range(n):
        low_limit = theta[i] - 2*np.sqrt(covariance[i][i])
        upper_limit = theta[i] + 2*np.sqrt(covariance[i][i])
        print('Confidence interval:', low_limit, ' < parameter {} <
    '.format(i+1), upper_limit)

    # 1-step prediction
    # y_hat(t+1|t) = 0.938y(t) +0.27y(t) - 0.27*yhat(t|t-1)
    # so if t = 0, yhat(1|0) = 0.938*y(0) +0.27*y(0) - 0.27*yhat(0|-1) =
    0.938*y(0) +0.27*y(0)
    # if t=1, yhat(2|1) = 0.938*y(1) +0.27*y(1) - 0.27*yhat(1|0)
    # and so on for the rest

    prediction = []
    yhat1 = -theta[0]*y_train[0] + theta[1] * y_train[0]
    prediction.append(yhat1)
    for i in range(1,len(y_train)-1):
        yhat = -theta[0]*y_train[i] + theta[1]*y_train[i] -
    theta[1]*prediction[i-1]
        prediction.append(yhat)

    plt.figure()
    plt.plot(x_train, y_train, label='Train Set')
    plt.plot(x_train[1:], prediction, label='Prediction of train set')
    plt.legend()
    plt.xticks(rotation=45)
    plt.gcf().autofmt_xdate()
    plt.xlabel('Time (Hourly)')
    plt.ylabel('Magnitude')
    plt.title('Train set and the prediction')
    plt.show()

    e = y_train[1:] - prediction
    Cal_autocorr_with_plot_fn(e, lags, 'ACF of residuals')
    print('MSE of residuals is', np.mean(e**2))
    print('Variance of prediction error is', np.var(e))
    print('Mean of prediction error is', np.mean(e))
    # Q-value
    Q = Cal_Q_fn(e,lags,len(y_train))
    print('Q value of residual is', Q)
    # Chi-square test
    DOF = lags - na - nb
    alpha = 0.01
    chi_critical = chi2.ppf(1-alpha, DOF)
    if Q < chi_critical:
        print('The residual is white.')
    else:
        print('The residual is NOT white.')

    # Zero/Pole cancellation
    roots_zeros = np.roots([1,theta[1]])
    roots_poles = np.roots([1,theta[0]])
    print('Roots of numerator:', roots_zeros)
    print('Roots of denominator:', roots_poles)

    # h-step forecast
    forecast = []
    yhat1 = -theta[0]*y_train.iloc[-1] + theta[1]*y_train.iloc[-1] -
    theta[1]*prediction[-1]
    forecast.append(yhat1)
    for i in range(len(y_test)-1):
        yhat = -theta[0]*forecast[i]
        forecast.append(yhat)

    forecast_e = y_test - forecast
    print('MSE of forecast error is', np.mean(forecast_e**2))
    print('Variance of forecast error is', np.var(forecast_e))
    print('Mean of forecast error is', np.mean(forecast_e))

    plt.figure()
    plt.plot(x_test, y_test, label='Test Set')
    plt.plot(x_test, forecast, label='Forecast of test set')
    plt.legend()
    plt.xticks(rotation=45)
    plt.gcf().autofmt_xdate()
    plt.xlabel('Time (Hourly)')
    plt.ylabel('Magnitude')
    plt.title('Test set and its forecast')
    plt.show()

    Cal_autocorr_with_plot_fn(forecast_e, lags, 'ACF of forecast error')


    ##### ARMA(7,0)######
    na = 7
    nb = 0
```

```
n = na + nb

theta = [0]*(n)

# Levenberg Marquardt Algorithm
# Step 1
def step1(data, theta, na, nb):
    max_order = max(na, nb)
    num = [0]*(max_order+1)
    den = [0]*(max_order+1)
    for i in range(na+1):
        if i==0:
            den[i] = 1
        else:
            den[i] = theta[i-1]

    for i in range(nb+1):
        if i==0:
            num[i] = 1
        else:
            num[i] = theta[na+i-1]

    system1 = (den, num, 1)
    _, e = signal.dlsim(system1, data)
    SSE = np.transpose(e).dot(e)

    X = []
    for i in range(n):
        theta_update = copy.copy(theta)
        theta_update[i] = theta_update[i] + delta
        for i in range(na + 1):
            if i == 0:
                den[i] = 1
            else:
                den[i] = theta_update[i - 1]

        for i in range(nb + 1):
            if i == 0:
                num[i] = 1
            else:
                num[i] = theta_update[na + i - 1]

        system = (den, num, 1)
        _, e_new = signal.dlsim(system, data)

        xi = (e - e_new) / delta
        X.append(xi)

    X = np.array(X)
    X = np.array([X[i].flatten() for i in range(len(X))]).T
    A = np.transpose(X).dot(X)
    g = np.transpose(X).dot(e)

    return SSE, X, A, g

# Step 2
def step2(data, A, g, theta, mu):
    I = np.identity(n)
    delta_theta = np.linalg.inv(A + mu*I).dot(g)
    delta_theta = np.reshape(delta_theta, (n,))
    theta_new = theta + delta_theta
    theta_new = theta_new.tolist()
    SSE_new, X_new, A_new, g_new = step1(data, theta_new, na, nb)

    if np.isinf(SSE_new):
        SSE_new = SSE
    if np.isnan(SSE_new):
        SSE_new = SSE*(10**8)

    return delta_theta, theta_new, SSE_new

# Compute steps 1 and 2
SSE, X, A, g = step1(y_train, theta, na, nb)
delta_theta, theta_new, SSE_new = step2(y_train, A, g, theta, mu)

# Step 3

max_iter = 50
i = 0
iteration_list = []
SSE_list = []

iteration_list.append(i) # initial iteration 0
SSE_list.append(SSE_new) # SSE at initial iteration 0

while i < max_iter:
    if SSE_new < SSE:
        if np.linalg.norm(delta_theta) < epsilon:
            theta = theta_new
            variance = SSE_new/(len(y_train)-n)
            covariance = variance*(np.linalg.inv(A))
            print('Estimated parameters:', theta)
            print('Variance:', variance)
            print('Covariance:', covariance)
            break
        else:
            theta = theta_new

    while SSE_new >= SSE:
        mu = mu*10
        delta_theta, theta_new, SSE_new = step2(y_train, A, g, theta, mu)
        if mu > mu_max:
            print(SSE_new)
            print('mu > mu_max. Check your code.')
            break


    i += 1
    iteration_list.append(i)
    if i > max_iter:
        print('i > 50. Check your code.')
        break

    theta = theta_new
    mu = mu / 10
    SSE, X, A, g = step1(y_train, theta, na, nb)
    delta_theta, theta_new, SSE_new = step2(y_train, A, g, theta, mu)
    SSE_list.extend(SSE_new)

# Plot SSE with iterations
plt.figure()
plt.plot(iteration_list, SSE_list)
plt.xlabel('Number of Iterations')
plt.ylabel('SSE')
plt.title('SSE vs number of iterations')
plt.show()

# Confidence interval
for i in range(n):
    low_limit = theta[i] - 2*np.sqrt(covariance[i][i])
    upper_limit = theta[i] + 2*np.sqrt(covariance[i][i])
    print('Confidence interval:', low_limit, ' < parameter {} <
'.format(i+1), upper_limit)

# 1-step prediction
prediction = []
yhat1 = -theta[0]*y_train[0]
prediction.append(yhat1)
yhat2 = -theta[0]*y_train[1] - theta[1] * y_train[0]
prediction.append((yhat2))
yhat3 = -theta[0]*y_train[2] - theta[1] * y_train[1] - theta[2]*y_train[0]
prediction.append((yhat3))
yhat4 = -theta[0]*y_train[3] - theta[1] * y_train[2] - theta[2]*y_train[1] -
0*y_train[0]
prediction.append((yhat4))
yhat5 = -theta[0]*y_train[4] - theta[1] * y_train[3] - theta[2]*y_train[2] -
0*y_train[1] - theta[4]*y_train[0]
prediction.append((yhat5))
yhat6 = -theta[0]*y_train[5] - theta[1] * y_train[4] - theta[2]*y_train[3] -
0*y_train[2] - theta[4]*y_train[1] - theta[5]*y_train[0]
prediction.append((yhat6))

for i in range(6,len(y_train)-1):
    yhat = -theta[0]*y_train[i] - theta[1] * y_train[i-1] -
theta[2]*y_train[i-2] - 0*y_train[i-3] - theta[4]*y_train[i-4] -
theta[5]*y_train[i-5] - theta[6]*y_train[i-6]
    prediction.append(yhat)

plt.figure()
plt.plot(x_train, y_train, label='Train Set')
plt.plot(x_train[1:], prediction, label='Prediction of train set')
plt.legend()
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Magnitude')
plt.title('Train set and the prediction')
plt.show()

e = y_train[1:] - prediction
Cal_autocorr_with_plot_fn(e, lags, 'ACF of residuals')
```

```
print('MSE of residuals is', np.mean(e**2))
print('Variance of prediction error is', np.var(e))
print('Mean of prediction error is', np.mean(e))
# Q-value
Q = Cal_Q_fn(e,lags,len(y_train))
print('Q value of residual is', Q)
# Chi-square test
DOF = lags - na - nb
alpha = 0.01
chi_critical = chi2.ppf(1-alpha, DOF)
if Q < chi_critical:
    print('The residual is white.')
else:
    print('The residual is NOT white.')

# Zero/Pole cancellation
roots_zeros = np.roots([1,0,0,0,0,0,0,0])
roots_poles =
np.roots([1,theta[0],theta[1],theta[2],theta[3],theta[4],theta[5],theta[6]])
print('Roots of numerator:', roots_zeros)
print('Roots of denominator:', roots_poles)

# h-step forecast
forecast = []
yhat1 = -theta[0]*y_train.iloc[-1] - theta[1]*y_train.iloc[-2] -
theta[2]*y_train.iloc[-3] -0*y_train.iloc[-4] - theta[4]*y_train.iloc[-5] -
theta[5]*y_train.iloc[-6] - theta[6]*y_train.iloc[-7]
forecast.append(yhat1)
yhat2 = -theta[0]*forecast[0] - theta[1]*y_train.iloc[-1] -
theta[2]*y_train.iloc[-2] -0*y_train.iloc[-3] - theta[4]*y_train.iloc[-4] -
theta[5]*y_train.iloc[-5] - theta[6]*y_train.iloc[-6]
forecast.append(yhat2)
yhat3 = -theta[0]*forecast[1] - theta[1]*forecast[0] -
theta[2]*y_train.iloc[-1] -0*y_train.iloc[-2] - theta[4]*y_train.iloc[-3] -
theta[5]*y_train.iloc[-4] - theta[6]*y_train.iloc[-5]
forecast.append(yhat3)
yhat4 = -theta[0]*forecast[2] - theta[1]*forecast[1] - theta[2]*forecast[0] -
0*y_train.iloc[-1] - theta[4]*y_train.iloc[-2] - theta[5]*y_train.iloc[-3] -
theta[6]*y_train.iloc[-4]
forecast.append(yhat4)
yhat5 = -theta[0]*forecast[3] - theta[1]*forecast[2] - theta[2]*forecast[1] -
0*forecast[0] - theta[4]*y_train.iloc[-1] - theta[5]*y_train.iloc[-2] -
theta[6]*y_train.iloc[-3]
forecast.append(yhat5)
yhat6 = -theta[0]*forecast[4] - theta[1]*forecast[3] - theta[2]*forecast[2] -
0*forecast[1] - theta[4]*forecast[0] - theta[5]*y_train.iloc[-1] -
theta[6]*y_train.iloc[-2]
forecast.append(yhat6)
yhat7 = -theta[0]*forecast[5] - theta[1]*forecast[4] - theta[2]*forecast[3] -
0*forecast[2] - theta[4]*forecast[1] - theta[5]*forecast[0] -
theta[6]*y_train.iloc[-1]
forecast.append(yhat7)
for i in range(6,len(y_test)-1):
    yhat = -theta[0]*forecast[i] - theta[1]*forecast[i-1] -
theta[2]*forecast[i-2] -0*forecast[i-3] - theta[4]*forecast[i-4] -
theta[5]*forecast[i-5] - theta[6]*forecast[i-6]
    forecast.append(yhat)
```

```
forecast_e = y_test - forecast
print('MSE of forecast error is', np.mean(forecast_e**2))
print('Variance of forecast error is', np.var(forecast_e))
print('Mean of forecast error is', np.mean(forecast_e))

plt.figure()
plt.plot(x_test, y_test, label='Test Set')
plt.plot(x_test, forecast, label='Forecast of test set')
plt.legend()
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Magnitude')
plt.title('Test set and its forecast')
plt.show()

Cal_autocorr_with_plot_fn(forecast_e, lags, 'ACF of forecast error')



##### SARIMA #######
# Check ACF and PACF
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(y, lags=50, ax=ax1)
plt.title('ACF Plot')
plt.ylabel('Magnitude')

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(y, lags=50, ax=ax2)
plt.title('PACF Plot')
plt.ylabel('Magnitude')
plt.xlabel('Lag')
plt.show()

#
********************************SARIMA(1,0,1)(1,0,1,24)********************
************
import statsmodels.api as sm
model = sm.tsa.statespace.SARIMAX(df['CO(GT)'], order=(1,0,1),
seasonal_order=(1,0,1,24)).fit()

print(model.summary())

prediction = model.predict(start=0, end=len(y_train)-1)
# Plot prediction with original data
plt.figure()
plt.plot(x_train, y_train, label='Train Set')
plt.plot(x_train, prediction, label='Prediction of train set')
plt.legend()
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Train set and the prediction SARIMA(1,0,1)(1,0,1,24)')
plt.show()
```

```
res_e = y_train - prediction
Cal_autocorr_with_plot_fn(res_e, lags, 'ACF Plot of residuals - 
SARIMA(1,0,1)(1,0,1,24)')
print('MSE of residuals is', np.mean(res_e**2))
print('Variance of prediction error is', np.var(res_e))
print('Mean of prediction error is', np.mean(res_e))
print('Q of prediction error is', Cal_Q_fn(res_e, lags, len(y_train)))

# Q-value
Q = Cal_Q_fn(res_e,lags,len(y_train))

na=1
nb=1
# Chi-square test
from scipy.stats import chi2
DOF = lags - na - nb
alpha = 0.01
chi_critical = chi2.ppf(1-alpha, DOF)
if Q < chi_critical:
    print('The residual is white.')
else:
    print('The residual is NOT white.')

## Forecast
forecast = model.predict(start=len(y_train), end=len(y_train)+len(y_test)-1)
# Plot forecast
plt.figure()
plt.plot(x_test, y_test, label='Test Set')
plt.plot(x_test, forecast, label='Forecast of test set')
plt.legend()
plt.xticks(rotation=45)
plt.gcf().autofmt_xdate()
plt.xlabel('Time (Hourly)')
plt.ylabel('Concentration')
plt.title('CO Level - Test set and the forecast SARIMA(1,0,1)(1,0,1,24)')
plt.show()

f_e_arima = y_test - forecast
Cal_autocorr_with_plot_fn(f_e_arima, lags, 'ACF Plot of forecast error - 
SARIMA(1,0,1)(1,0,1,24)')
print('MSE of forecast is', np.mean(f_e_arima**2))
print('Variance of forecast error is', np.var(f_e_arima))
print('Mean of forecast error is', np.mean(f_e_arima))
print('Q of forecast error is', Cal_Q_fn(f_e_arima, lags, len(y_test)))
print('Mean of forecast', np.mean(forecast))
```