Time Series Analysis – Final Term Project

US Air Pollution Prediction and Forecast

Abstract

The aim of this project is to demonstrate the techniques learnt from Time Series Analysis course by applying them on a real dataset. For this project, I have used daily dataset US Air Pollution whose records are between 2000 and 2016, and was downloaded from data.world. Some of the notable models used in this project would be – ARIMA, and base models such as – Average, Drift, Naïve, and Simple Exponential Smoothing.

Introduction

Air pollution is a contamination of natural oxygen by environmental factors such as vehicle emissions, harmful gas emitted by industries, and even a fire accident in a forest. Such pollution raises serious health concerns for humans and may cause respiratory problems that includes – Asthma, Bronchitis, etc. Air quality index is a measure that informs the quality of oxygen, for instance – with carbon monoxide air quality index, the lesser the index the lesser the natural oxygen is contaminated with CO.

Daily AQI Color	Levels of Concern	Values of Index	Description of Air Quality
Green	Good	0 to 50	Air quality is satisfactory, and air pollution poses little or no risk.
Yellow	Moderate	51 to 100	Air quality is acceptable. However, there may be a risk for some people, particularly those who are unusually sensitive to air pollution.
Orange	Unhealthy for Sensitive Groups	101 to 150	Members of sensitive groups may experience health effects. The general public is less likely to be affected.
Red	Unhealthy	151 to 200	Some members of the general public may experience health effects; members of sensitive groups may experience more serious health effects.
Purple	Very Unhealthy	201 to 300	Health alert: The risk of health effects is increased for everyone.
Maroon	Hazardous	301 and higher	Health warning of emergency conditions: everyone is more likely to be affected.

Source: AQI Basics | AirNow.gov

Dataset

The dataset contains about 1.7 million records, and 16 features. There are four major pollutants that was found in the dataset – Carbon Monoxide (CO), Nitrogen Dioxide (NO_2), Sulfur Dioxide (SO_2), and Ground-level ozone (O^3). Due to the nature of the project whose requirement is to demonstrate the skills learnt from the course, I have decided to use only the Carbon Monoxide Air Quality Index as the dependent variable.

Notable Variables	Units
Air Temperature	Degrees Centigrade
Relative Humidity	Percent
Solar Radiation	Megajoules per Square Meter
Precipitation	Millimeters
Soil Temperature	Degrees Centigrade
Wind Speed	Meters per second
Wind Vector Magnitude	Meters per second
Wind Vector Direction	Degrees
Wind Direction Standard Deviation	Degrees
Reference Evapotranspiration	Millimeters
Heat Units	Centigrade
Carbon Monoxide Air Quality Index	

All the variables from Air Pollution dataset were Air Quality Indices for 47 states in the US and several places within them. Not all the records for each state were equally sized, some of them only had about 3000 records, while some of them had missing records for only 3 or 4 days. These details will be explained in the pre-processing section.

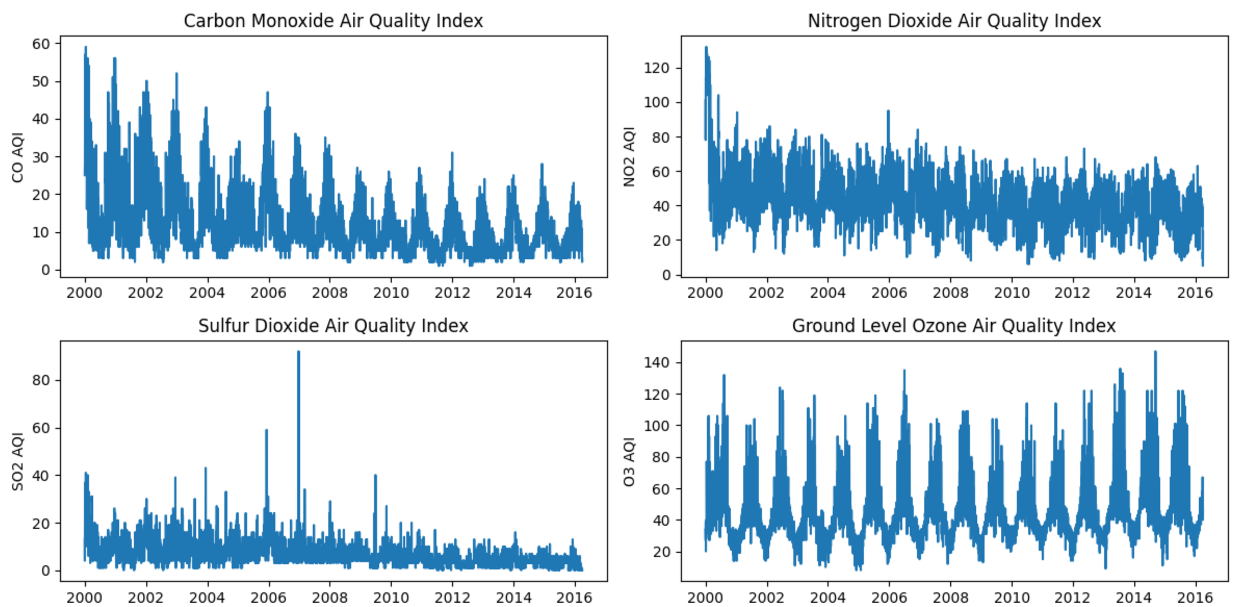
Since all of them are dependent variables, I had decided to use climatical information such as temperature, precipitation, etc., to help in regression analysis. The weather dataset was downloaded from https://ag.arizona.edu/azmet/.htm for

Pre-processing

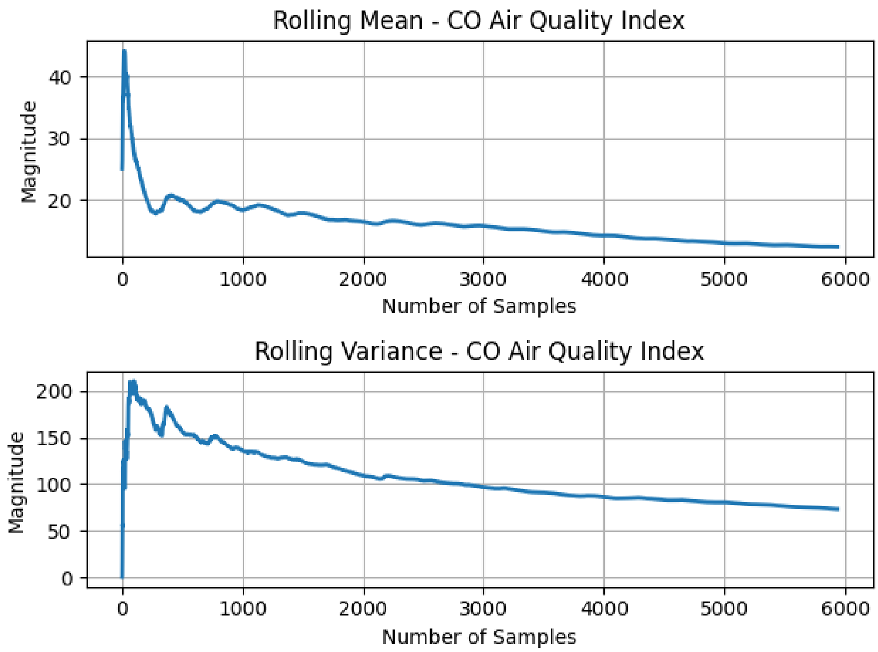
As mentioned in the previous section, records for each state were not equally sized. In order to handle this, I visualized the start and the end date for each state, and picked the state with lengthy duration. The chosen state was Arizona, and it has about 5937 records after aggregation. Naturally, the data had two records for each date, and after thorough research, it appears the scientists have recorded the air quality index twice a day at random times, so I have used average aggregation to make it a valid daily dataset where a day only has one record.

The Arizona subset still had a few missing records whose NA values were replaced using Seasonal Naïve method ensuring the seasonal information is not compromised and was joined with weather information downloaded specifically for Arizona state to form a complete dataset.

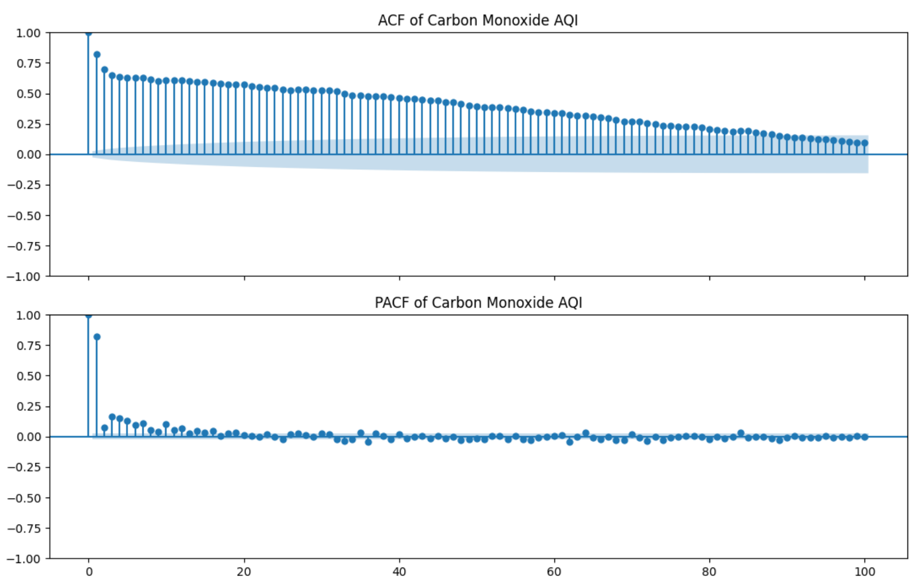
Plot of the dataset



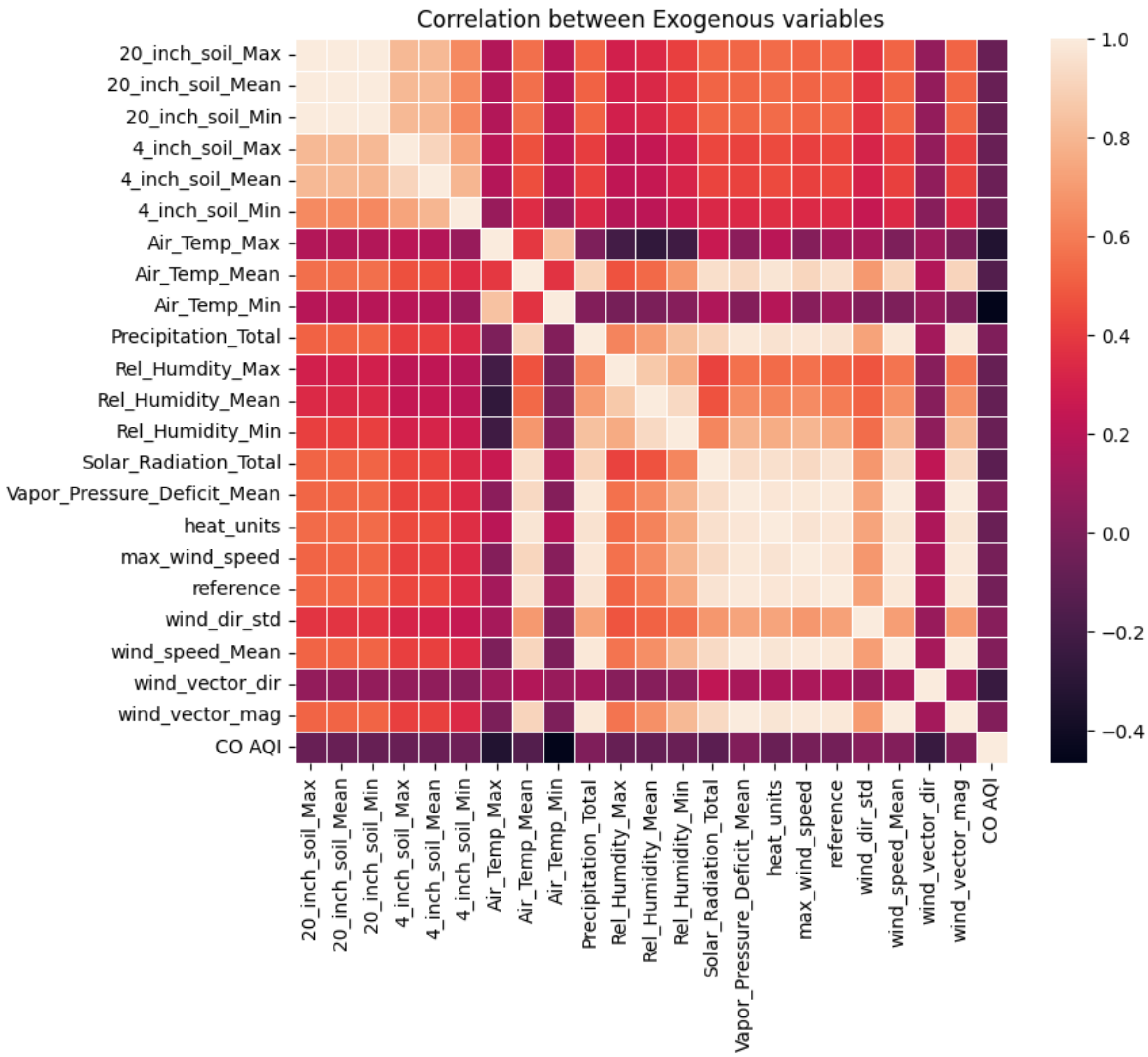
Rolling Mean and the Variance



ACF and the PACF



Correlation plot



Strength of trend, and seasonality

Strength of trend in CO AQI is: 80.51915950547837%
Strength of seasonality in CO AQI is 37.355263265149816%

ADF of raw CO AQI data

ADF Statistic: -5.314366

p-value: 0.000005

Critical Values:

1%: -3.431

5%: -2.862

10%: -2.567

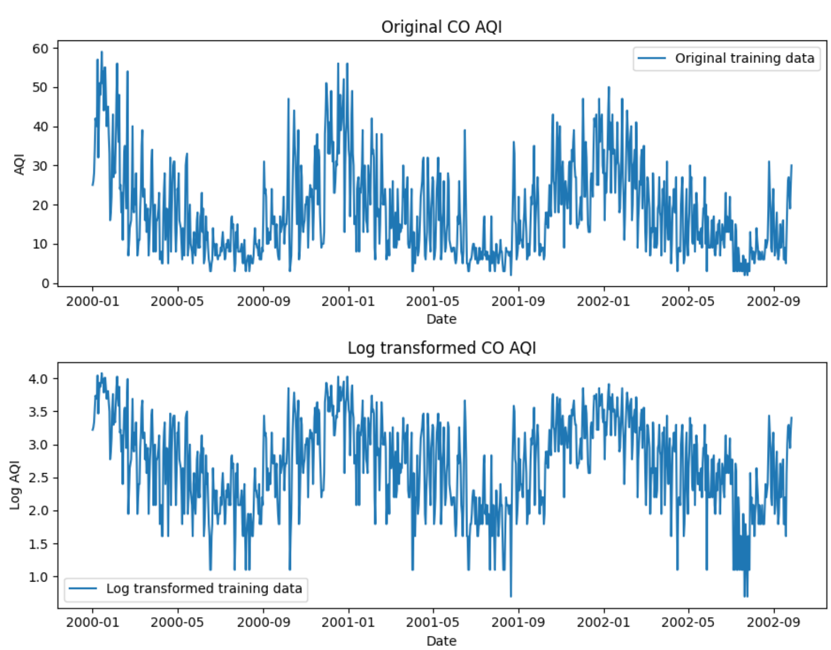
Results of KPSS Test:

Test Statistic	2.234447
p-value	0.010000
Lags Used	37.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000
Critical Value (2.5%)	0.574000
Critical Value (1%)	0.739000

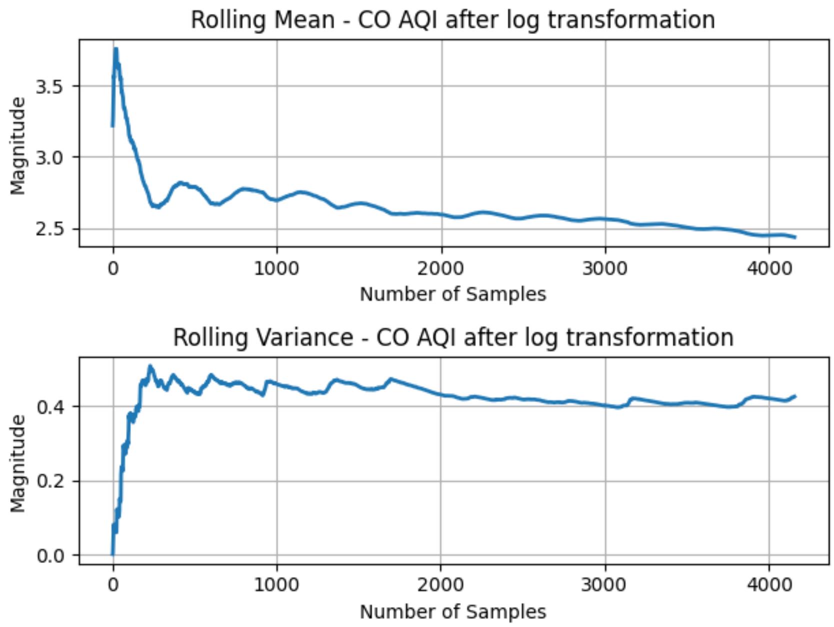
Making it stationary

The CO AQI was very seasonal, and somewhat resistant to seasonal differencing. Hence, in order to make it stationary, I performed log transformation followed by a first order non-seasonal differencing. This paved way for further analysis such as order derivation from the GPAC, etc.

Plot of the data after log transformation



I have limited the X-axis window to better understand the seasonality pattern, and to visualize the effects of logarithmic transformation to the seasonal data.



ADF test for CO AQI after log transformation:

ADF Statistic: -5.014841
p-value: 0.000021
Critical Values:
1%: -3.432
5%: -2.862
10%: -2.567

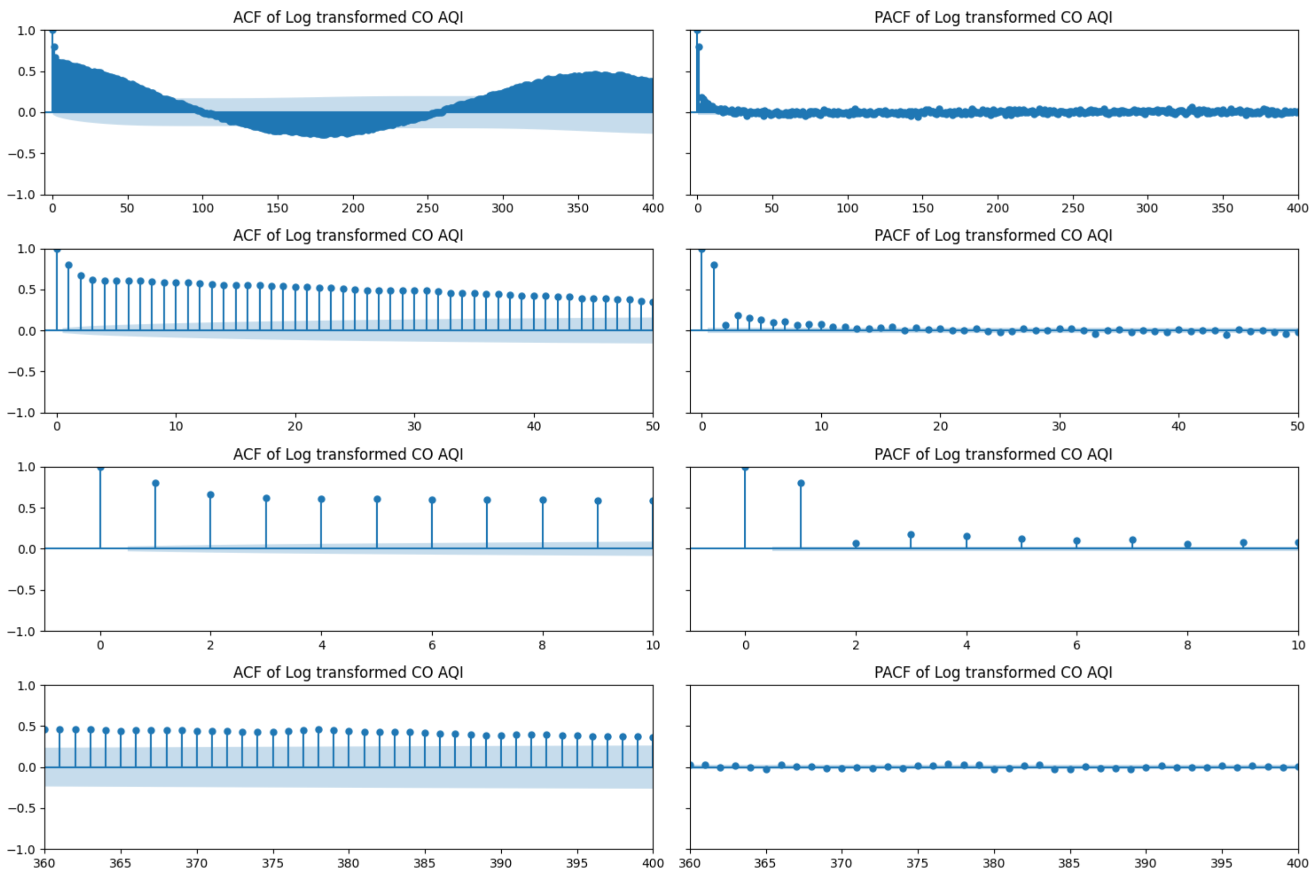
KPSS test for CO AQI after log transformation:

Test Statistic	1.915565
p-value	0.010000
Lags Used	38.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000
Critical Value (2.5%)	0.574000
Critical Value (1%)	0.739000

Strength of trend in CO AQI Original data is: 78.57788169398243%
Strength of trend in CO AQI after log transformation is: 78.29234750495951%

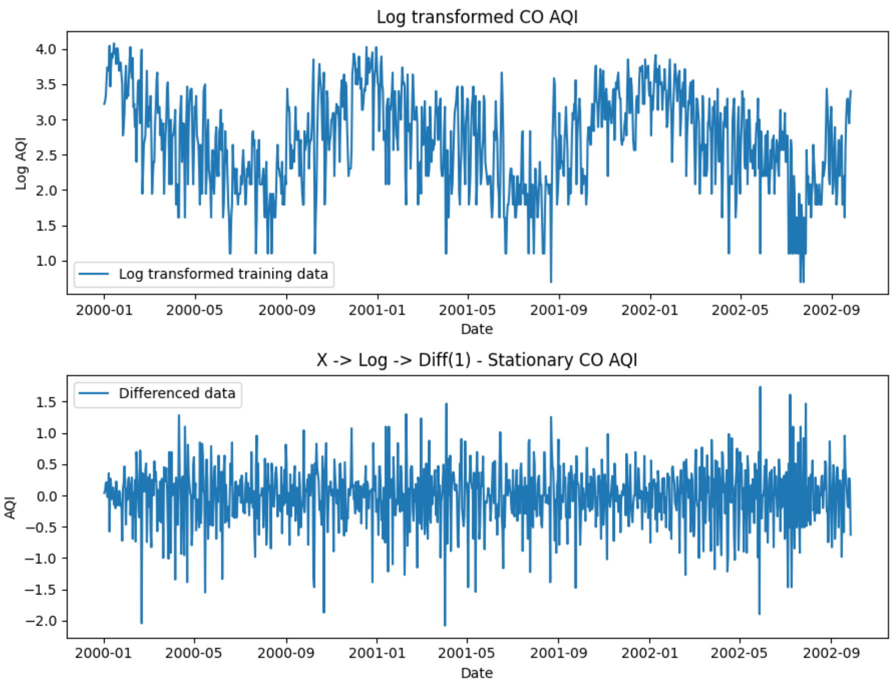
Strength of seasonality in CO AQI Original data is 37.454057571565826%
Strength of seasonality in CO AQI after log transformation is 35.7691123968119%

ACF, and PACF of the log transformed data

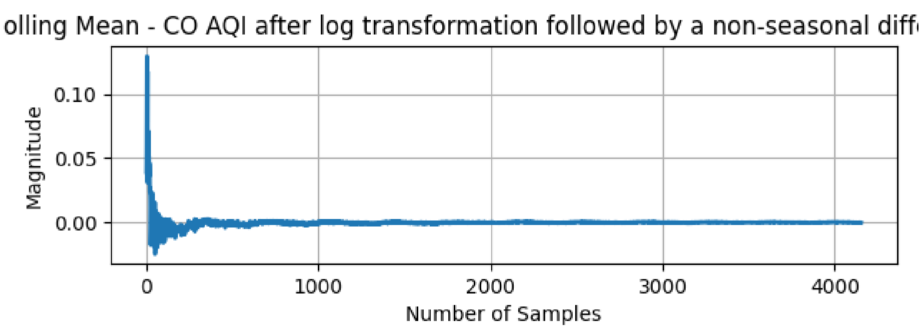


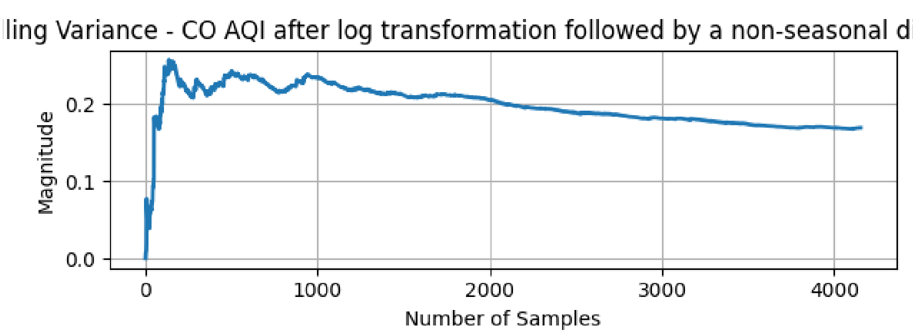
Following the log transformation with a non-seasonal differencing

Plot of the data



Rolling mean and the variance





ADF-test

ADF Statistic: -20.099069
p-value: 0.000000
Critical Values:
  1%: -3.432
  5%: -2.862
  10%: -2.567

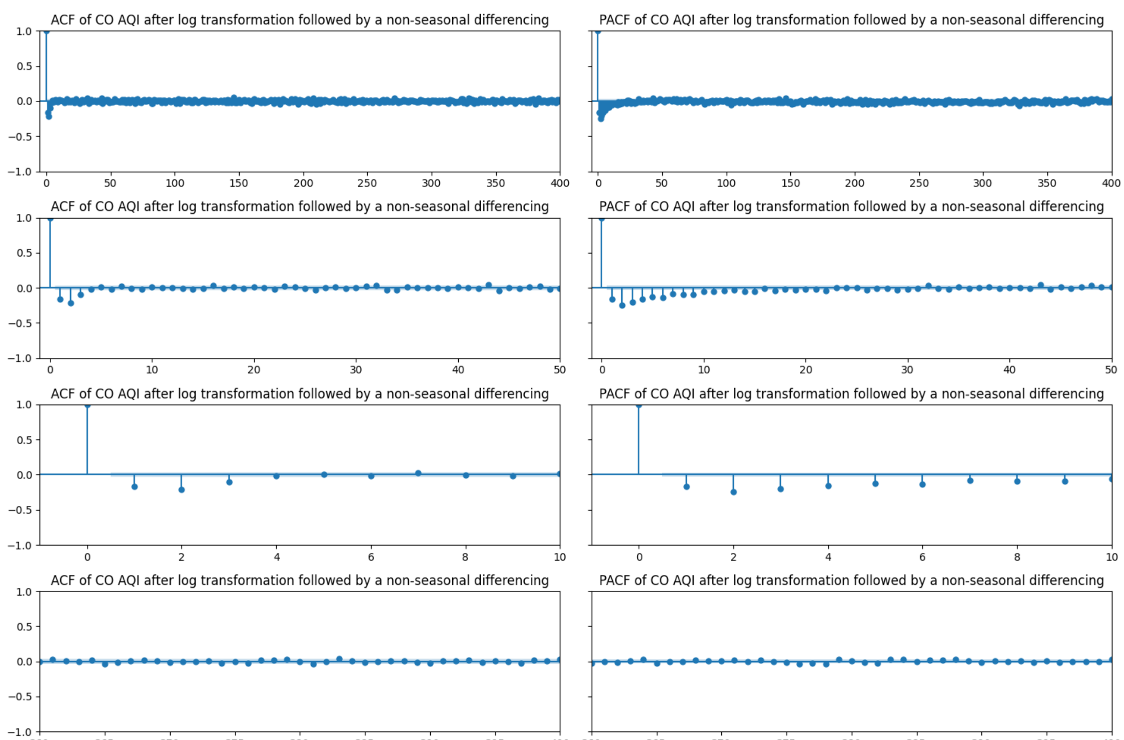
KPSS-test

Test Statistic	0.077684
p-value	0.100000
Lags Used	591.000000
Critical Value (10%)	0.347000
Critical Value (5%)	0.463000
Critical Value (2.5%)	0.574000
Critical Value (1%)	0.739000

dtype: float64

Both the ADF-test, and the KPSS-test show the two transformations has made the data stationary.

ACF, and PACF



The pattern in the ACF of log transformed, and non-seasonally differenced data between the 0th and the 4th lag appears to be cut-off while that in the PACF is tail-off. Hence, it is presumed based on this information, the non-seasonal MA would be 4 given no second occurrence is found.

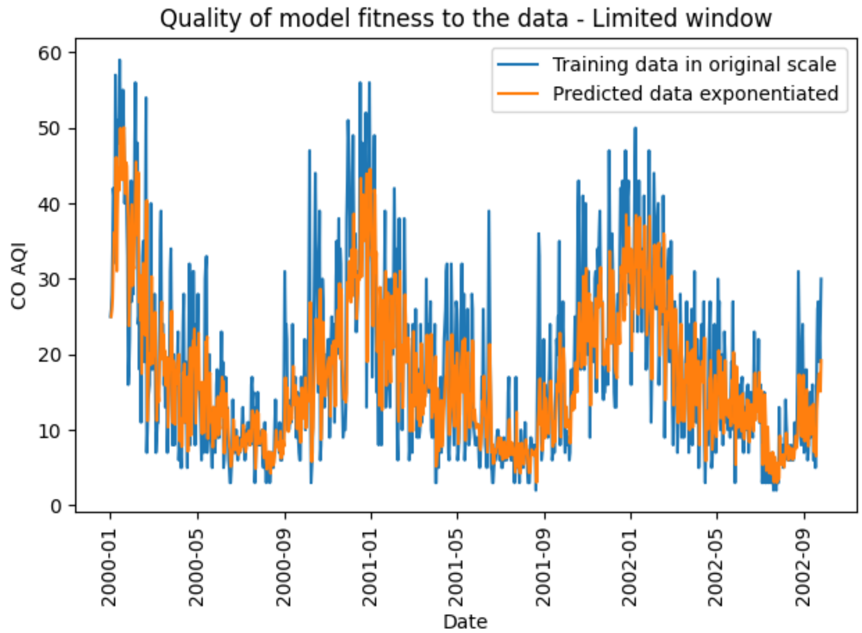
GPAC

The second row does shows values near zero as well, however, the chosen order was found to be the best fit to the data based on the results of the residual analysis. The final order of the AR is 4, MA is 3, and since we did one non-seasonal differencing, the order of integration would be 1. Hence ARIMA(4,1,3).

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                 CO AQI   No. Observations:                 4155
Model:                 ARIMA(4, 1, 3)   Log Likelihood               -1719.136
Date:                Fri, 23 Dec 2022   AIC                           3454.273
Time:                        17:24:35   BIC                           3504.927
Sample:                             0   HQIC                          3472.194
                               - 4155
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7758      0.035    -21.966      0.000      -0.845      -0.707
ar.L2         -0.3869      0.034    -11.387      0.000      -0.453      -0.320
ar.L3          0.3138      0.029     10.676      0.000       0.256       0.371
ar.L4         -0.1282      0.018     -7.200      0.000      -0.163      -0.093
ma.L1          0.4002      0.034     11.651      0.000       0.333       0.468
ma.L2         -0.2245      0.028     -8.036      0.000      -0.279      -0.170
ma.L3         -0.8534      0.030    -28.304      0.000      -0.912      -0.794
sigma2         0.1334      0.002     56.692      0.000       0.129       0.138
===================================================================================
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):               680.53
Prob(Q):                              0.75   Prob(JB):                         0.00
Heteroskedasticity (H):               0.66   Skew:                            -0.74
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.31
===================================================================================
```

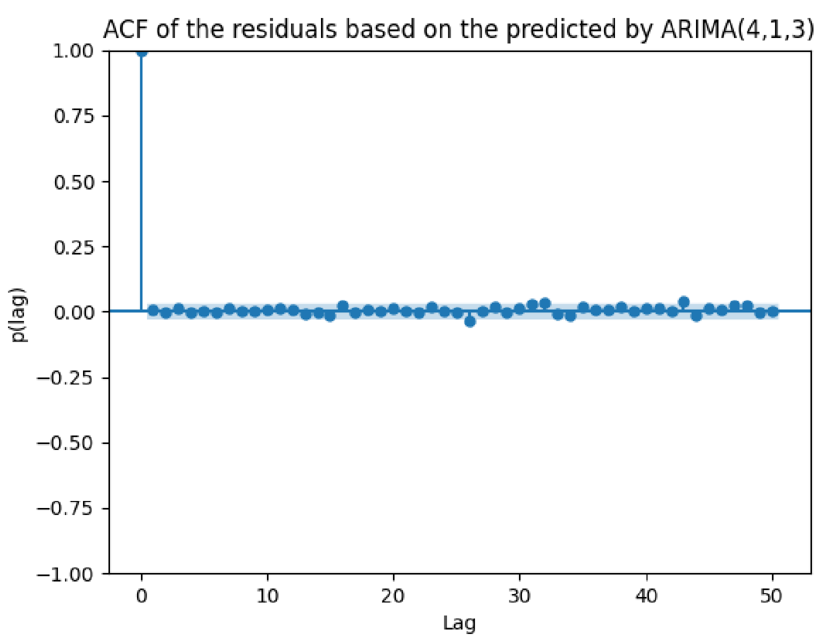
As you can see the Ljung-Box Q value is very close to 0, and the probability Ljung-box test produced a p-value 0.75 which says the residuals does not show a lack of fit to the white nosie. Also, the summary shows no variables are statistically insignificant.

Plot of the training data with prediction



Some of the plots are limited in window size to visualize the goodness of fit.

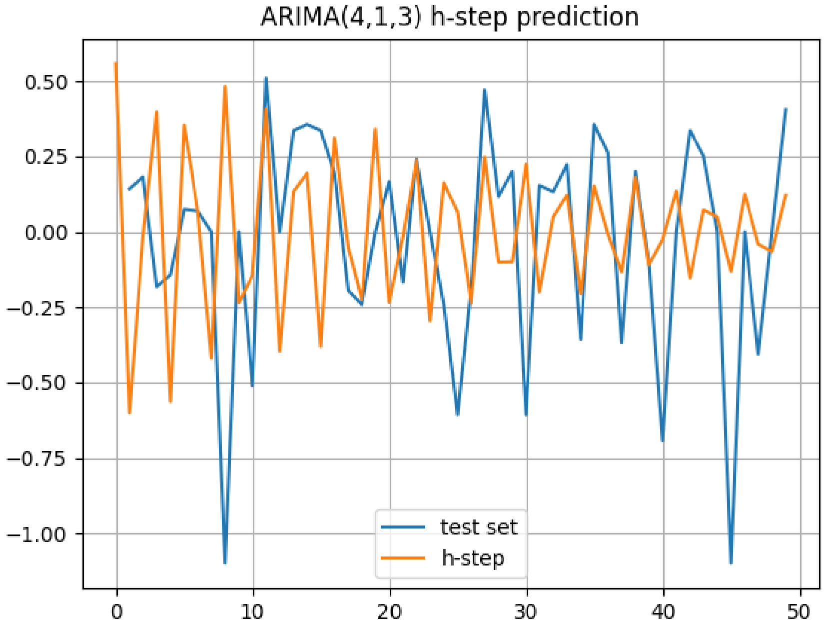
ACF of the residuals



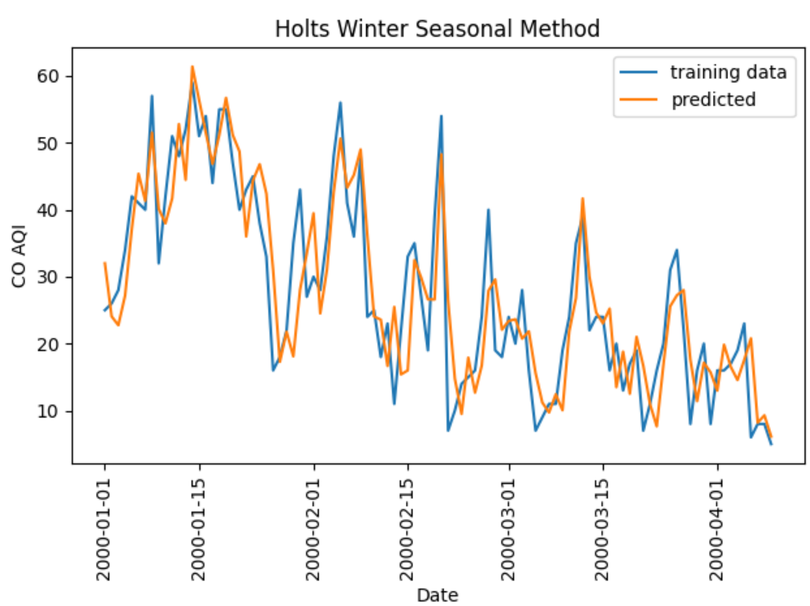
This is an unbiased estimator.

The estimated variance of the error is 0.1334.

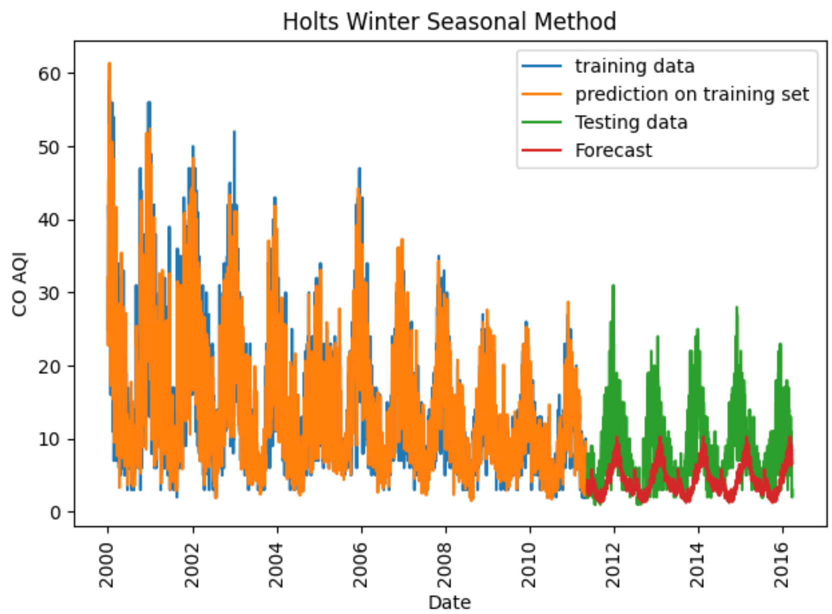
h-step prediction



Holt-Winters Seasonal Method



70% of the data was used for training, while the rest was reserved for testing purposes.



Feature Selection

Since there are several features in the dataset, I had used Principal Component Analysis to reduce the dimensionality of the dataset. Based on the eigen-analysis, the basis vectors with variance less than 5% was eliminated ensuring at the same time AIC, and BIC values from the OLS do not deteriorate. It would not be possible to list the names of the variables since the variables in the low dimensional subspace are not interpretable. However, the results of the analysis were fruitful.

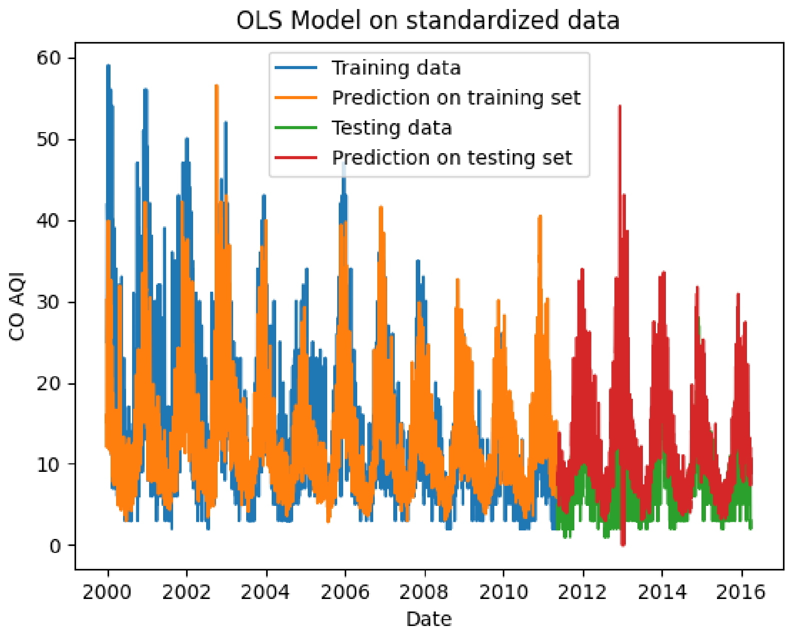
Following PCA, three different transformations were attempted to find the best fit for OLS to the data.

1. Limiting to simple standardization
2. ZCA – Decorrelating data using rotation neutralized transformation
3. PCA – Reduced dimension

OLS on Standardized data

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 CO AQI   R-squared:                       0.550
Model:                            OLS   Adj. R-squared:                  0.548
Method:                 Least Squares   F-statistic:                     229.9
Date:                Fri, 23 Dec 2022   Prob (F-statistic):               0.00
Time:                        17:24:41   Log-Likelihood:                -2467.6
No. Observations:                4156   AIC:                             4981.
Df Residuals:                    4133   BIC:                             5127.
Df Model:                          22
Covariance Type:            nonrobust
=============================================================================================
                                coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------------
20_inch_soil_Max             16.7704      1.476     11.360      0.000      13.876      19.665
20_inch_soil_Mean           -18.8219      1.759    -10.698      0.000     -22.271     -15.372
20_inch_soil_Min              2.0286      0.506      4.009      0.000       1.037       3.021
4_inch_soil_Max              -0.0151      0.023     -0.669      0.504      -0.059       0.029
4_inch_soil_Mean              1.6676      0.410      4.063      0.000       0.863       2.472
4_inch_soil_Min              -1.6246      0.404     -4.022      0.000      -2.417      -0.833
Air_Temp_Max                  0.6916      0.061     11.296      0.000       0.572       0.812
Air_Temp_Mean                -1.2774      0.198     -6.456      0.000      -1.665      -0.889
Air_Temp_Min                  0.0157      0.052      0.304      0.761      -0.086       0.117
Precipitation_Total          -0.1046      0.045     -2.327      0.020      -0.193      -0.016
Rel_Humdity_Max               0.0242      0.014      1.763      0.078      -0.003       0.051
Rel_Humidity_Mean            -0.2408      0.028     -8.535      0.000      -0.296      -0.185
Rel_Humidity_Min              0.0282      0.031      0.894      0.371      -0.034       0.090
Solar_Radiation_Total        -0.6679      0.047    -14.285      0.000      -0.760      -0.576
Vapor_Pressure_Deficit_Mean   4.2153      0.317     13.293      0.000       3.594       4.837
heat_units                   -1.1779      0.122     -9.625      0.000      -1.418      -0.938
max_wind_speed               -0.4715      0.059     -7.948      0.000      -0.588      -0.355
reference                    -1.1287      0.130     -8.710      0.000      -1.383      -0.875
wind_dir_std                 -0.0427      0.019     -2.214      0.027      -0.080      -0.005
wind_speed_Mean              -0.0389      0.440     -0.088      0.930      -0.902       0.824
wind_vector_dir              -0.0636      0.007     -8.520      0.000      -0.078      -0.049
wind_vector_mag               0.5315      0.429      1.239      0.215      -0.310       1.373
intercept                     2.4368      0.007    357.543      0.000       2.423       2.450
==============================================================================
Omnibus:                       40.980   Durbin-Watson:                   0.906
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.502
```

Plot

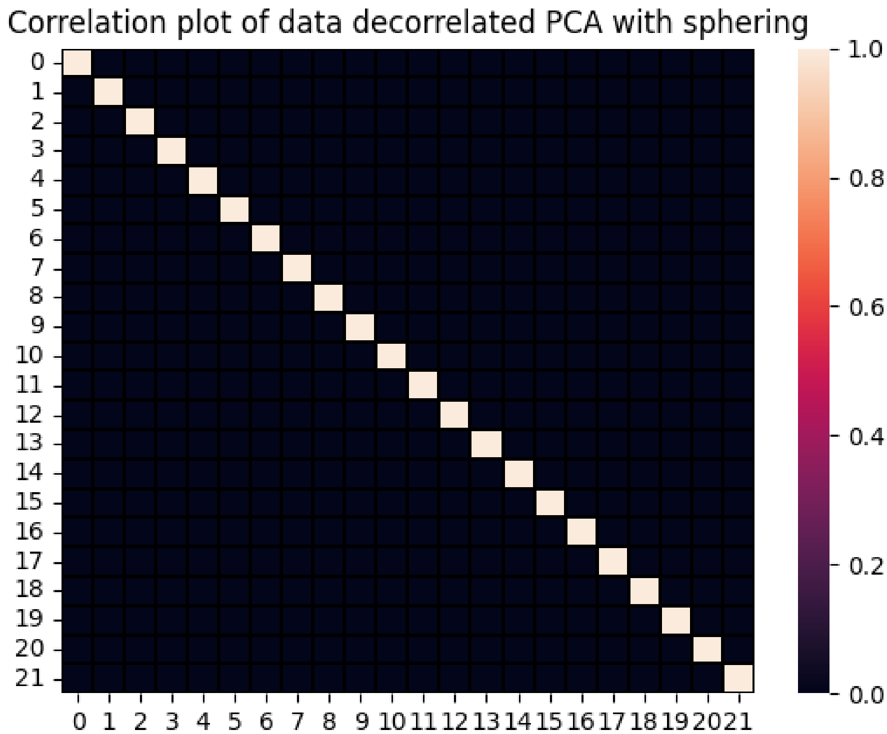


The ACF of the residuals does not shows as the residuals being white. Hence, I wish not to disclose it.

Q value: 56457.51928906493

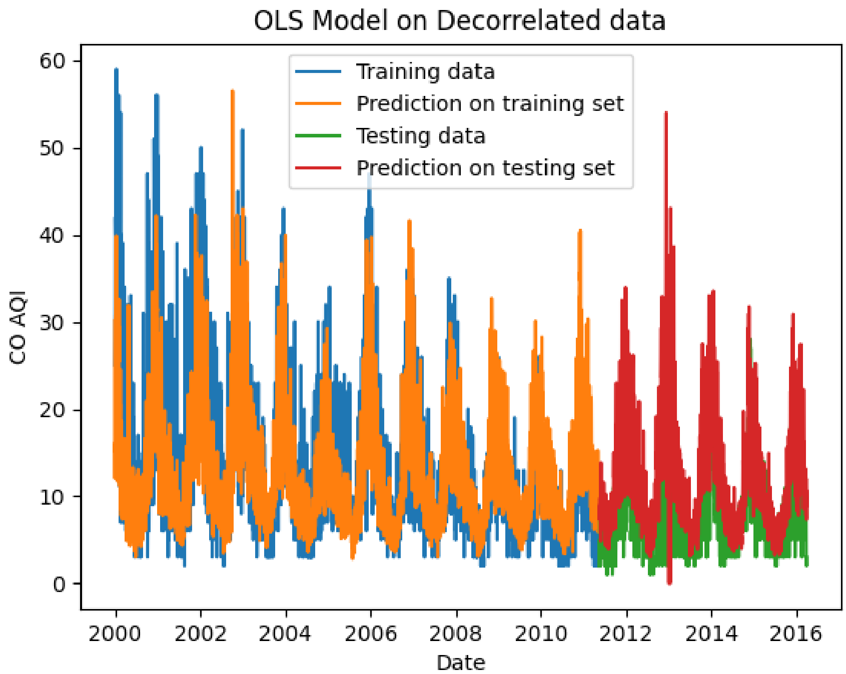
ZCA

ZCA is nothing but the process of whitening the data as it is widely believed that decorrelating the data at times help in improving the performance of the regression analysis. The correlation plot of the decorrelated data is as follows:


Correlation plot of data decorrelated PCA with sphering
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
1.0
0.8
0.6
0.4
0.2
0.0

OLS on ZCA transformed data

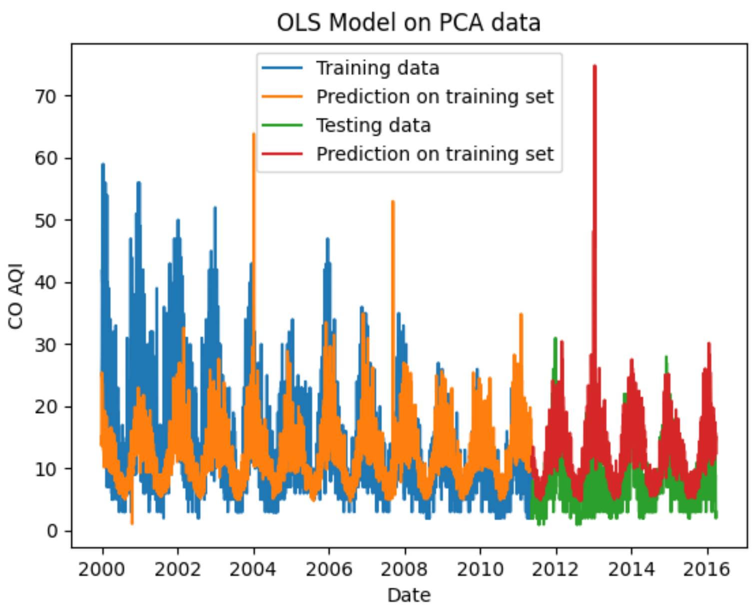
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 CO AQI   R-squared:                       0.550
Model:                            OLS   Adj. R-squared:                  0.548
Method:                 Least Squares   F-statistic:                     229.9
Date:                Fri, 23 Dec 2022   Prob (F-statistic):               0.00
Time:                        17:24:41   Log-Likelihood:                -2467.6
No. Observations:                4156   AIC:                             4981.
Df Residuals:                    4133   BIC:                             5127.
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
0              0.0457      0.007      6.708      0.000       0.032       0.059
1             -0.0656      0.007     -9.619      0.000      -0.079      -0.052
2             -0.0255      0.007     -3.746      0.000      -0.039      -0.012
3             -0.0079      0.007     -1.162      0.245      -0.021       0.005
4              0.0247      0.007      3.618      0.000       0.011       0.038
5             -0.0266      0.007     -3.908      0.000      -0.040      -0.013
6             -0.0864      0.007    -12.673      0.000      -0.100      -0.073
7             -0.0485      0.007     -7.111      0.000      -0.062      -0.035
8             -0.3197      0.007    -46.898      0.000      -0.333      -0.306
9              0.0362      0.007      5.310      0.000       0.023       0.050
10            -0.0426      0.007     -6.249      0.000      -0.056      -0.029
11            -0.0877      0.007    -12.862      0.000      -0.101      -0.074
12            -0.0496      0.007     -7.273      0.000      -0.063      -0.036
13            -0.1943      0.007    -28.511      0.000      -0.208      -0.181
14             0.1599      0.007     23.464      0.000       0.147       0.173
15            -0.0235      0.007     -3.442      0.001      -0.037      -0.010
16            -0.0816      0.007    -11.976      0.000      -0.095      -0.068
17             0.0016      0.007      0.236      0.814      -0.012       0.015
18             0.0540      0.007      7.916      0.000       0.041       0.067
19             0.0595      0.007      8.732      0.000       0.046       0.073
20            -0.1319      0.007    -19.353      0.000      -0.145      -0.119
21             0.0829      0.007     12.155      0.000       0.069       0.096
intercept      2.4368      0.007    357.543      0.000       2.423       2.450
==============================================================================
Omnibus:                       40.980   Durbin-Watson:                   0.906
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.502
Skew:                          -0.199   Prob(JB):                     1.32e-10
Kurtosis:                       3.324   Cond. No.                         1.00
==============================================================================
```



OLS on PCA data

```
OLS Model on Latent space
                            OLS Regression Results
==============================================================================
Dep. Variable:                 CO AQI   R-squared:                       0.342
Model:                            OLS   Adj. R-squared:                  0.341
Method:                 Least Squares   F-statistic:                     359.1
Date:                Fri, 23 Dec 2022   Prob (F-statistic):               0.00
Time:                        19:58:35   Log-Likelihood:                -3259.1
No. Observations:                4156   AIC:                             6532.
Df Residuals:                    4149   BIC:                             6577.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
0             -0.0166      0.002     -6.871      0.000      -0.021      -0.012
1              0.0313      0.004      7.691      0.000       0.023       0.039
2              0.1372      0.005     25.381      0.000       0.127       0.148
3             -0.2283      0.007    -30.442      0.000      -0.243      -0.214
4             -0.1637      0.008    -19.479      0.000      -0.180      -0.147
5             -0.1051      0.011     -9.889      0.000      -0.126      -0.084
intercept      2.4368      0.008    296.108      0.000       2.421       2.453
==============================================================================
Omnibus:                       36.226   Durbin-Watson:                   0.683
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               51.181
Skew:                          -0.098   Prob(JB):                     7.69e-12
Kurtosis:                       3.507   Cond. No.                         4.41
==============================================================================
```

Plot of the fit



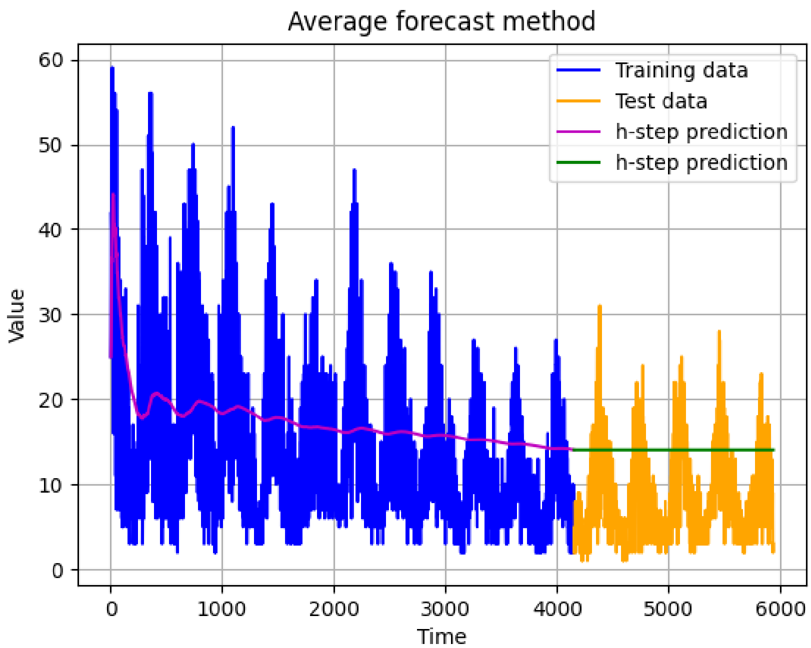
VIF

On the standardized training set, the VIF values are very large, and hence, I believe I took the appropriate step of decorrelating the data before fitting it onto the OLS.

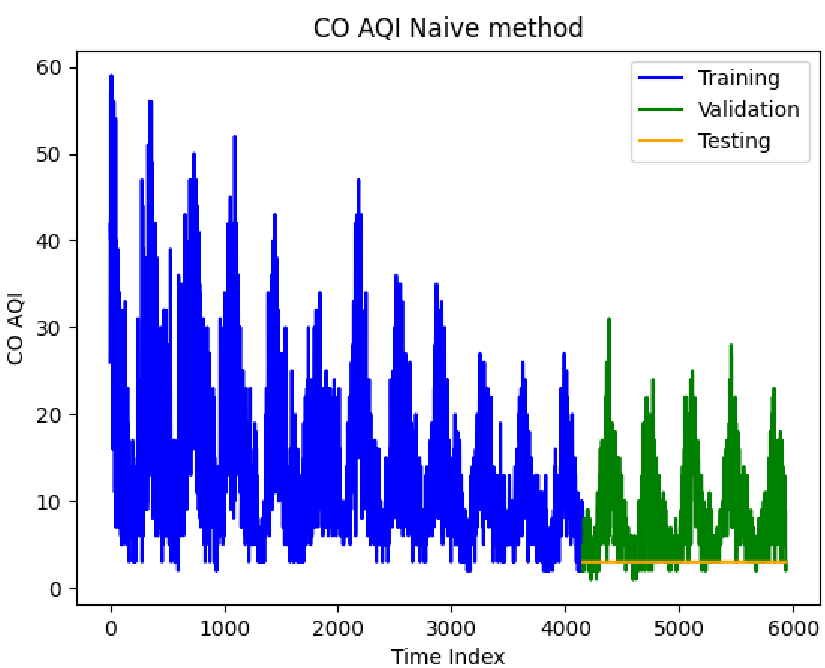
	Variable	VIF
0	20_inch_soil_Max	46911.964907
1	20_inch_soil_Mean	66629.840099
2	20_inch_soil_Min	5510.416467
3	4_inch_soil_Max	10.986406
4	4_inch_soil_Mean	3625.553987
5	4_inch_soil_Min	3511.640967
6	Air_Temp_Max	80.675439
7	Air_Temp_Mean	842.544934
8	Air_Temp_Min	57.448027
9	Precipitation_Total	43.514759
10	Rel_Humdity_Max	4.048655
11	Rel_Humidity_Mean	17.125868
12	Rel_Humidity_Min	21.356718
13	Solar_Radiation_Total	47.044728
14	Vapor_Pressure_Deficit_Mean	2164.268151
15	heat_units	322.382505
16	max_wind_speed	75.734511
17	reference	361.396298
18	wind_dir_std	7.989991
19	wind_speed_Mean	4170.717737
20	wind_vector_mag	3961.510308

Base Models

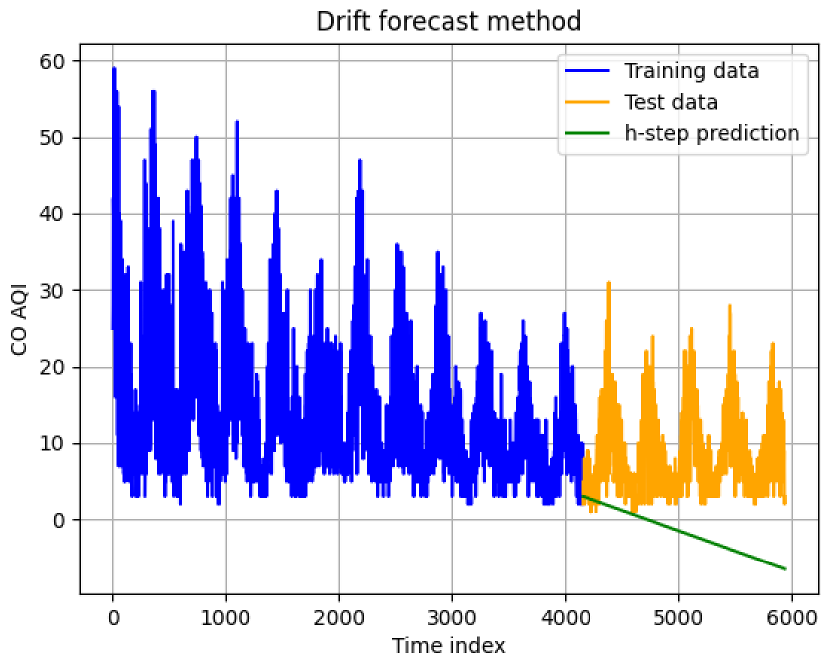
Average forecast method



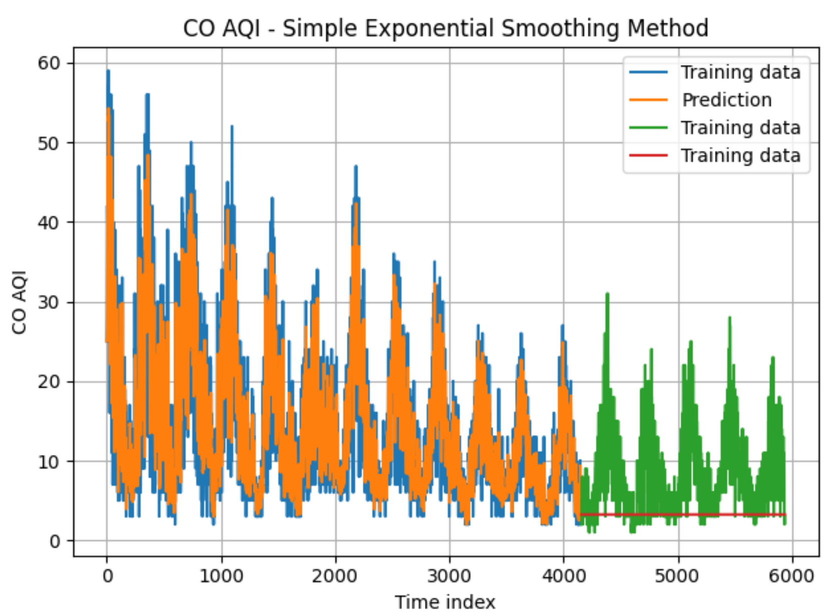
Naïve forecast method



Drift Forecast Method



Simple Exponential Smoothing Method



Conclusion

Comparing all the models, it appears that ARIMA and Holts-Winter Seasonal forecast was found to have the best fit compared to all the models. In particular, Holt-Winters Seasonal forecast method was able to capture the seasonality with lower RMSE compared to the ARIMA. The multi-linear regression was good in terms of predicting pattern in the training set. However, it was unable to generalize the multiplicative component of the seasonality to the unseen time steps as was evident from the forecast plot.

Due to limited time constraints, I wish to wrap up the analysis at this point, but if I had more time, I would be interested in exploring how well Deep Learning model is able to generalize the multiplicative component of seasonality present in the data.

References

1. AirNow.gov, U.S. EPA. (n.d.). Aqi Basics. AQI Basics | AirNow.gov. Retrieved December 24, 2022, from https://www.airnow.gov/aqi/aqi-basics/#:~:text=The%20higher%20the%20AQI%20value,300%20represents%20hazardous%20air%20quality.

2. Kumar, K., & Pande, B. P. (2022, May 15). *Air Pollution Prediction with Machine Learning: A case study of Indian cities*. International journal of environmental science and technology : IJEST. Retrieved December 24, 2022, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9107909/

3. University of Arizona. AZMET : The Arizona Meteorological Network - The University of Arizona. (n.d.). Retrieved December 24, 2022, from https://cals.arizona.edu/azmet/

Appendix – Python code

```
#%% Load the libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.holtwinters import ExponentialSmoothing as ETS
from Utilities.WhitenessTest import WhitenessTest as WT
from Utilities.Correlation import Correlation as Corr
from statsmodels.tsa.seasonal import STL
import seaborn as sns
from numpy.testing import assert_equal
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from Utilities.GPAC import gpac_table
from scipy.stats import chi2

#%%
def whiteness_test(x, name):
    wt = WT(x)
    print(f"ADF test for {name}:\n")
    wt.ADF_Cal()
    print(f"\nKPSS test for {name}:\n")
    wt.kpss_test()
    wt.Plot_Rolling_Mean_Var(name=name)

#%%
def plot_acf_pacf(x, lags, name, xlims=None):
    r_idx = 0
    if xlims:
        fig, axes = plt.subplots(len(xlims), 2, sharex=False,
sharey=True, figsize=(15, 10))
        for xlim in xlims:
            plot_acf(x, lags=lags, ax=axes[r_idx, 0])
            axes[r_idx, 0].set_xlim(xlim)
            axes[r_idx, 0].set_title(f'ACF of {name}')

            plot_pacf(x, lags=lags, ax=axes[r_idx, 1])
            axes[r_idx, 1].set_xlim(xlim)
            axes[r_idx, 1].set_title(f'PACF of {name}')

            r_idx += 1
    else:
        fig, axes = plt.subplots(2, 1, sharex=True, figsize=(11,
7))
        plot_acf(x, lags=lags, ax=axes[0])
        axes[0].set_title(f'ACF of {name}')
        plot_pacf(x, lags=lags, ax=axes[1])
```

```
        axes[1].set_title(f'PACF of {name}')
    plt.tight_layout()
    plt.show()

#%% Load the data
uspoll = pd.read_csv('pollution_us_2000_2016.csv', header=0,
index_col=0)
uspoll['Date Local'] = pd.to_datetime(uspoll['Date Local'])

#%% Dtypes
print(uspoll.dtypes)

#%% First and the last date
first_date_by_state = uspoll[['State', 'Date
Local']].groupby(['State']).agg('first')
last_date_by_state = uspoll[['State', 'Date
Local']].groupby(['State']).agg('last')

#%%
fig, axes = plt.subplots(1, 2, figsize=(16, 8))
axes[0].stem(first_date_by_state.index.values,
first_date_by_state['Date Local'])
axes[0].axhline(y=first_date_by_state.min(), color='r')
axes[0].tick_params(labelrotation=90)
axes[0].set_ylabel('Start Date')
axes[0].set_title('Start date in each state')

axes[1].stem(last_date_by_state.index.values,
last_date_by_state['Date Local'])
axes[1].axhline(y=last_date_by_state.mode(), color='r')
axes[1].tick_params(labelrotation=90)
axes[1].set_ylabel('Last Date')
axes[1].set_title('Last date in each state')
fig.tight_layout()
plt.show()

#%% Drop columns
"""
We will aggregate the values from all address within a State as the
pollutant values
are not expected to vary by much.
"""
uspoll = uspoll.drop(columns=['State Code', 'County Code', 'Site
Num',
                              'Address', 'County', 'City'])

#%% Replace NaN's in AQI variables

# Replace NaN's in CO AQI
```

```
co_aqi = uspoll[['State', 'Date Local', 'CO AQI']]\
                    .groupby(['State', 'Date Local'])\
                    .agg('max').reset_index()

# Replace NaN's in SO2 AQI
so2_aqi = uspoll[['State', 'Date Local', 'SO2 AQI']]\
                .groupby(['State', 'Date Local'])\
                .agg('max').reset_index()

# Replace NaN's in O3 AQI
o3_aqi = uspoll[['State', 'Date Local', 'O3 AQI']]\
                .groupby(['State', 'Date Local'])\
                .agg('max').reset_index()

# Replace NaN's in NO2 AQI
no2_aqi = uspoll[['State', 'Date Local', 'NO2 AQI']]\
                .groupby(['State', 'Date Local'])\
                .agg('max').reset_index()

#%%
"""
We are aggregating values other than AQI since columns other than
AQI have multiple records
for the same day, and the differences between the instances are
subtle. Hence, an average of those
values would yield an equally sized daily data.
"""
non_cat_vars = uspoll.dtypes[uspoll.dtypes != 'object']
aqi_vars = ['CO AQI', 'NO2 AQI', 'SO2 AQI', 'O3 AQI']
uspoll_agg = uspoll[list(np.setdiff1d(non_cat_vars.index.values,
aqi_vars)) + ['State']]\
                    .groupby(['State', 'Date Local'])\
                    .agg('mean').reset_index()

#%%
"""
Adding back all the AQI columns that we ignored in the aggregation
step (last step) since
under the multiple records each day, only one of them has the value
while the rest were filled
by NaN. This is why removed NaN prior to the aggregation, and now
we join all AQIs with the
averaged data frame.
"""
uspoll_agg = pd.merge(uspoll_agg, co_aqi, on=['State', 'Date
Local'],
                      how='inner')
uspoll_agg = pd.merge(uspoll_agg, so2_aqi, on=['State', 'Date
Local'],
```

```
                      how='inner')
uspoll_agg = pd.merge(uspoll_agg, no2_aqi, on=['State', 'Date
Local'],
                      how='inner')
uspoll_agg = pd.merge(uspoll_agg, o3_aqi, on=['State', 'Date
Local'],
                      how='inner')

#%% List of States that are maximally sized
fdbs = first_date_by_state
ldbs = last_date_by_state
states_with_min_sy = (fdbs.loc[((fdbs ==
fdbs.min()).reset_index(drop=True)['Date
Local']).values].reset_index())['State']
states_with_max_ey = (ldbs.loc[((ldbs ==
ldbs.mode().values[0]).reset_index(drop=True)['Date
Local']).values].reset_index())['State']
states_equally_sized = np.intersect1d(states_with_min_sy,
states_with_max_ey)

#%%
# It is my understanding that the using one of the states would
suffice the requirement
# since the requirement is to have at least 5000 instances.
chosen_state = states_equally_sized[0]
uspoll_state = uspoll_agg.loc[uspoll_agg.State == chosen_state]
print(uspoll_state.head())

#%%
"""
We do an left outer join between the filtered uspoll_state data
frame (data corresponding to Arizona)
and the
data frame that contain only the dates between the start and last
date found in uspoll_state.
Just in case there are dates that have missing values under any
column. This would allow us
to identify the dates on which no information was registered.
Perhaps we could use forecasting
techniques to replace the missing values.
"""
original_date_range = pd.date_range(start='2000-01-01', end='2016-
03-31',
                                    freq='D', inclusive='both')
original_date_range = pd.DataFrame({'Date
Local':original_date_range})

uspoll_state = pd.merge(left=original_date_range,
right=uspoll_state, how='left',
```

```
                        on='Date Local', sort=False, copy=True)

#%% Extract the dates under which the data is missing
missing_dates = uspoll_state.loc[uspoll_state["CO AQI"].isnull(),
"Date Local"]
print(f'List of dates for which values are missing:\n'
      f'{missing_dates}')

#%% Let's plot, and visualize the data until the start of the
missing values
subset_nmissing = uspoll_state.loc[uspoll_state['Date Local'] <
missing_dates.min(),
                                          ['Date Local'] +
aqi_vars]\
                              .set_index('Date Local')\
                              .asfreq('D')

fig, axes = plt.subplots(2, 2, figsize=(12, 6))
axes[0, 0].plot(subset_nmissing['CO AQI'])
axes[0, 0].set_ylabel('CO AQI')
axes[0, 0].set_title('Carbon Monoxide Air Quality Index')
axes[0, 1].plot(subset_nmissing['NO2 AQI'])
axes[0, 1].set_ylabel('NO2 AQI')
axes[0, 1].set_title('Nitrogen Dioxide Air Quality Index')
axes[1, 0].plot(subset_nmissing['SO2 AQI'])
axes[1, 0].set_ylabel('SO2 AQI')
axes[1, 0].set_title('Sulfur Dioxide Air Quality Index')
axes[1, 1].plot(subset_nmissing['O3 AQI'])
axes[1, 1].set_ylabel('O3 AQI')
axes[1, 1].set_title('Ground Level Ozone Air Quality Index')
fig.tight_layout()
plt.show()

#%%
# acronym nmissing correspond to not missing
co_aqi = subset_nmissing['CO AQI']
so2_aqi = subset_nmissing['SO2 AQI']
o3_aqi = subset_nmissing['O3 AQI']
no2_aqi = subset_nmissing['NO2 AQI']

#%%
def seasonal_naive_forecast(df, vars, m):
    def forecast(X, last_tr_index, test_length):
        # Keep everything as indices
        T = X.loc[:last_tr_index].shape[0] - 1
        for h in range(1, test_length + 1):
            k = int((h - 1) / m)
            index = T + h - (m * (k + 1))
            X.iloc[T + h] = X.iloc[index]
```

```
        return X

    indices = df.index
    for var in vars:
        missing_indices = np.where(df.loc[:, var].isna())[0]
        if len(missing_indices) > 0:
            prev_time_index = indices[missing_indices[0] - 1]
            # train_length = len(indices[:(missing_indices[0]-1)])
            df.loc[:, var] = forecast(df[var],
last_tr_index=prev_time_index,

test_length=len(missing_indices))
    return df

uspoll_state = uspoll_state.set_index('Date Local')
uspoll_state.loc[:, aqi_vars] =
seasonal_naive_forecast(df=uspoll_state, vars=aqi_vars, m=375)
# print('debug...')

#%% Visualize the complete data after replacing the missing values
using Seasonal Naive method
fig, axes = plt.subplots(2, 2, figsize=(12, 6))
axes[0, 0].plot(uspoll_state['CO AQI'])
axes[0, 0].set_ylabel('CO AQI')
axes[0, 0].set_title('Carbon Monoxide Air Quality Index')
axes[0, 1].plot(uspoll_state['NO2 AQI'])
axes[0, 1].set_ylabel('NO2 AQI')
axes[0, 1].set_title('Nitrogen Dioxide Air Quality Index')
axes[1, 0].plot(uspoll_state['SO2 AQI'])
axes[1, 0].set_ylabel('SO2 AQI')
axes[1, 0].set_title('Sulfur Dioxide Air Quality Index')
axes[1, 1].plot(uspoll_state['O3 AQI'])
axes[1, 1].set_ylabel('O3 AQI')
axes[1, 1].set_title('Ground Level Ozone Air Quality Index')
fig.tight_layout()
plt.show()


# HWS removed and stored in backup.txt
#%%
"""
Weather Dataset is now included to aid in the process of
regression.
"""

#%% Merge two dataframes
weather_df = pd.read_csv('arizona_weather.csv', header=0)
weather_df['Date Local'] = pd.to_datetime(weather_df['Date Local'])
joined_df = pd.merge(left=weather_df,
```

```
right=uspoll_state.reset_index(), how='inner',
                     on='Date Local', sort=False)
joined_df = joined_df.set_index('Date Local')

#%% ADF-test of raw data
wt_raw_no = WT(x=joined_df['NO2 AQI'])
print("ADF of raw NO2 AQI data\n")
wt_raw_no.ADF_Cal()

wt_raw_co = WT(x=joined_df['CO AQI'])
print("\nADF of raw CO AQI data\n")
wt_raw_co.ADF_Cal()

wt_raw_so = WT(x=joined_df['SO2 AQI'])
print("\nADF of raw SO2 AQI data\n")
wt_raw_so.ADF_Cal()

wt_raw_o3 = WT(x=joined_df['O3 AQI'])
print("\nADF of raw O3 AQI data\n")
wt_raw_o3.ADF_Cal()


#%% Rolling mean and the variance of raw dataset

wt = WT(joined_df['NO2 AQI'])
wt.Plot_Rolling_Mean_Var(name='NO2 Air Quality Index')

wt = WT(joined_df['CO AQI'])
wt.Plot_Rolling_Mean_Var(name='CO Air Quality Index')

wt = WT(joined_df['SO2 AQI'])
wt.Plot_Rolling_Mean_Var(name='SO2 Air Quality Index')

wt = WT(joined_df['O3 AQI'])
wt.Plot_Rolling_Mean_Var(name='O3 Air Quality Index')

#%%
"""-------------------------------------------------------------------
----------
Step 7 - Print the strength of seasonality and trend
-------------------------------------------------------------------
-------"""

def print_strength_seas_tren(x, name):
    stl = STL(x, period=12).fit()
    residual = stl.resid
    trend = stl.trend
    seasonal = stl.seasonal
    f_t = np.max([0, 1-(np.var(residual)/np.var(residual +
```

```
trend))]) # denominator = seasonally adjusted data
    f_s = np.max([0, 1-(np.var(residual)/np.var(residual +
seasonal))]) # denominator = detrended data
    print(f"Strength of trend in {name} is: {f_t * 100}%")
    print(f"Strength of seasonality in {name} is {f_s * 100}%")

print_strength_seas_tren(joined_df['CO AQI'], 'CO AQI')
print("")
print_strength_seas_tren(joined_df['SO2 AQI'], 'SO2 AQI')
print("")
print_strength_seas_tren(joined_df['NO2 AQI'], 'NO2 AQI')
print("")
print_strength_seas_tren(joined_df['O3 AQI'], 'O3 AQI')


#%%


#%%
"""---------------------------------------------
Make the CO AQI data stationary
---------------------------------------------"""

def seasonal_differencing(y, seasonal_period):
    m = seasonal_period
    s_diff = []
    for t in range(m, len(y)):
        s_diff.append(y[t] - y[t-m])
    return s_diff

# Split the data into train, and test sets
# N = joined_df.shape[0]
# train_test_split = int(0.7 * N)
# test_len = N - train_test_split
# co = joined_df['CO AQI'][:train_test_split]
# co_test = joined_df['CO AQI'][train_test_split:]

N = joined_df.shape[0]
train_test_split = int(0.7 * N)
test_len = N - train_test_split
co = joined_df['CO AQI'][:train_test_split]
co_test = joined_df['CO AQI'][train_test_split:]

# Plot the data constrained to a narrow window so it is easier to
visualize the
# seasonality.
plt.figure()
co.reset_index(drop=True).head(1000).plot()
plt.xlabel('Observation index')
```

```
plt.title('CO AQI of Arizona (Original Data) - limited window')
plt.ylabel('AQI')
plt.tight_layout()
plt.show()

# Strength of Seasonality and trend in the raw dataset
print_strength_seas_tren(co, name='CO AQI Original Data')

# ADF, and KPSS test on original data
whiteness_test(co, name='CO AQI Original Data')

"""
ACF of the dependent variable
"""
plot_acf_pacf(joined_df['CO AQI'], name='Carbon Monoxide AQI',
lags=100)
# plot_acf_pacf(joined_df['NO2 AQI'], name='Nitrogen Dioxide AQI',
lags=100)
# plot_acf_pacf(joined_df['SO2 AQI'], name='Sulfur Dioxide AQI',
lags=100)
# plot_acf_pacf(joined_df['O3 AQI'], name='Ground Level Ozone AQI',
lags=100)

"""
The Carbon Monoxide AQI data was very stubborn in not lending
itself
to not being able to transform a stationary data despite several
attempts
in making seasonal and non-seasonal differencing. Hence Log
transformation is
preferred
"""
co_log = np.log(co)
co_test_log = np.log(co_test)

# Plot of Raw and log transformed data
fig, axes = plt.subplots(2, 1, figsize=(9, 7))
axes[0].plot(co.head(1000), label='Original training data')
axes[0].set_title('Original CO AQI')
axes[0].set_xlabel('Date')
axes[0].set_ylabel('AQI')
axes[0].legend()
axes[1].plot(co_log.head(1000), label='Log transformed training
data')
axes[1].set_title('Log transformed CO AQI')
axes[1].set_xlabel('Date')
axes[1].set_ylabel('Log AQI')
axes[1].legend()
fig.tight_layout()
```

```python
plt.show()

# Strength of Seasonality and trend after log transformation
print_strength_seas_tren(co, name='CO AQI Original data')
print('')
print_strength_seas_tren(co_log, name='CO AQI after log
transformation')

# ADF, and KPSS test
whiteness_test(co_log, name='CO AQI after log transformation')

# ACF, and PACF of Log transformed CO AQI
plot_acf_pacf(co_log, lags=400, name='Log transformed CO AQI',
              xlims=[[-5, 400], [-1, 50], [-1, 10], [360, 400]])


"""
There is still some level of seasonality present in the data. Going
for a non-seasonal differencing.
"""
co_diff1 = co_log.diff()[1:]

# Plot of Raw and log transformed data
fig, axes = plt.subplots(2, 1, figsize=(9, 7))
axes[0].plot(co_log.head(1000), label='Log transformed training
data')
axes[0].set_title('Log transformed CO AQI')
axes[0].set_xlabel('Date')
axes[0].set_ylabel('Log AQI')
axes[0].legend()
axes[1].plot(co_diff1[:1000], label='Differenced data')
axes[1].set_title('X -> Log -> Diff(1) - Stationary CO AQI')
axes[1].set_xlabel('Date')
axes[1].set_ylabel('AQI')
axes[1].legend()
fig.tight_layout()
plt.show()

# Strength of Seasonality and trend after log transformation
print_strength_seas_tren(co, name='CO AQI Original data')
print('')
print_strength_seas_tren(co_log, name='CO AQI after log
transformation')
print('')
print_strength_seas_tren(co_diff1, name='CO AQI after log
transformation followed by a non-seasonal differencing')

# ADF, and KPSS test
whiteness_test(co_diff1, name='CO AQI after log transformation
```

```
followed by a non-seasonal differencing')

# ACF, and PACF of Log transformed CO AQI
plot_acf_pacf(co_diff1, lags=400, name='CO AQI after log
transformation followed by a non-seasonal differencing',
              xlims=[[-5, 400], [-1, 50], [-1, 10], [360, 400]])

# Only at this stage, GPAC allows the orders to be derived

#%% GPAC
"""--------------------------------
Generalized Partial AutoCorrelation
--------------------------------"""
corr = Corr()
lags = int(co.shape[0]/50)
acf_vals, _ = corr.acf(co_diff1.reset_index(drop=True),
max_lag=lags, plot=False, return_acf=True)
gpac_vals = gpac_table(acf_vals, na=13, nb=13, plot=False)
plt.figure(figsize=(13, 10))
sns.heatmap(gpac_vals, annot=True)
plt.xticks(ticks=np.array(list(range(13))) + .5,
labels=list(range(1, 14)))
plt.title('Generalized Partial Autocorrelation (GPAC) Table')
plt.xlabel('AR Order')
plt.ylabel('MA Order')
plt.tight_layout()
plt.show()

#%% ARIMA
"""
Main model - ARIMA
"""
na = 4
d = 1
nb = 3
arima_fit = sm.tsa.ARIMA(endog=co_log, order=(4,1,3),
trend='n').fit()
print(arima_fit.summary())

# Residual Analysis
y_hat = arima_fit.predict()
residuals = co_log.values - y_hat.values

# Plot ACF of the residuals
plt.figure()
plot_acf(residuals, lags=50)
plt.title('ACF of the residuals based on the predicted by
ARIMA(4,1,3)')
plt.xlabel('Lag')
```

```
plt.ylabel(r'p(lag)')
plt.show()

# Compute the Q value
arima_Q = co.shape[0] * (acf_vals[1:].T @ acf_vals[1:])
chi2_from_table = chi2.ppf(q=0.95, df=co.shape[0]-na-nb-1)

print(f"Chi-square critical value Q {arima_Q} is less than the
value from the"
      f"table {chi2_from_table}. Since the Q value did not enter
the critical region"
      f"in the distribution, we refuse to reject the null
hypothesis which states"
      f"the data exhibits no statistical significance. In simple
words, the residuals"
      f"are white.")

print(f"More accurate diagnosis - Ljung-Box test:")
print(sm.stats.acorr_ljungbox(x=residuals, lags=[50]))

#%%
"""
Plot that shows the fit between the original data and the predicted
data
"""
y_hat_exp = np.exp(y_hat)

fig, axes = plt.subplots(2, 1, figsize=(15, 15))
axes[0].plot(co, label='Training data in original scale')
axes[0].plot(y_hat_exp, label='Predicted data exponentiated')
axes[0].set_title('Quality of model fitness to the data')
axes[0].set_xlabel('Date')
axes[0].set_ylabel('CO AQI')
axes[0].legend()

axes[1].plot(co.head(1000), label='Training data in original
scale')
axes[1].plot(y_hat_exp.head(1000), label='Predicted data
exponentiated')
axes[1].set_title('Quality of model fitness to the data - Limited
window')
axes[1].set_xlabel('Date')
axes[1].set_ylabel('CO AQI')
axes[1].legend()
fig.tight_layout()
plt.show()

plt.figure()
plt.plot(co.head(1000), label='Training data in original scale')
```

```
plt.plot(co.head(1000).index.values[1:], y_hat_exp.head(1000)[1:],
label='Predicted data exponentiated')
plt.title('Quality of model fitness to the data - Limited window')
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()

#%%
"""
ARIMA(4,1,3) - MANUAL H-STEP PREDICTION - FORECAST FUNCTION WITH
BACK TRANSFORMATION
"""

joined_df = pd.read_csv('joined_df.csv', header=0, index_col='Date
Local')
co_x = joined_df['CO
AQI'][:train_test_split+50].reset_index(drop=True)
co_train_arima = np.log(co_x[:-50]).diff()
co_test_arima = np.log(co_x[-50:]).diff()
e = co_train_arima.diff()

co_train_arima = co_train_arima.values
co_test_arima = co_test_arima.values

test_set = joined_df['CO AQI'][-50:].reset_index(drop=True).values

def h_step_pred(h, y, yhat, e, T):

    a1 = -0.7758
    a2 = -0.3869
    a3 = 0.3138
    a4 = -0.1282

    b1 = 0.4002
    b2 = -0.2245
    b3 = -0.8534

    lag_1 = T+h-1
    lag_2 = T+h-2
    lag_3 = T+h-3
    lag_4 = T+h-4

    if lag_1 > T:
        lag1_val = yhat[lag_1]
        lag1_eval = 0
    else:
```

```
        lag1_val = y[lag_1]
        lag1_eval = e[lag_1]

    if lag_2 > T:
        lag2_val = yhat[lag_2]
        lag2_eval = 0
    else:
        lag2_val = y[lag_2]
        lag2_eval = e[lag_2]

    if lag_3 > T:
        lag3_val = yhat[lag_3]
        lag3_eval = 0
    else:
        lag3_val = y[lag_3]
        lag3_eval = e[lag_3]

    if lag_4 > T:
        lag4_val = yhat[lag_4]
    else:
        lag4_val = y[lag_4]

    y_val = a1 * lag1_val + a2 * lag2_val + a3 * lag3_val + a4 *
lag4_val +\
            b1 * lag1_eval + b2 * lag2_eval + b3 * lag3_eval

    return y_val

y_h_step_pred = co_train_arima
for h in range(1, 52):
    tmp_val = h_step_pred(h=h, y=co_train_arima,
yhat=y_h_step_pred,
                          e=e, T=co_train_arima.shape[0]-1)
    y_h_step_pred = np.r_[y_h_step_pred, tmp_val]

def back_transformation(y, p):
    reversed = []
    for i in range(len(y)-1):
        reversed.append(y[i+1] - p[i+1])
    return reversed

last_50_hstep = y_h_step_pred[-51:][:50]

p = [last_50_hstep[0] - co_log[-1]]
for i in range(1, 50):
    p.append(last_50_hstep[i] - last_50_hstep[i-1])

reversed_h_step = back_transformation(last_50_hstep, p)
h_step_complete_bt = np.exp(reversed_h_step)
```

```python
plt.figure()
plt.plot(co.head(1000), label='Training data in original scale')
plt.plot(co.head(1000).index.values[1:], y_hat_exp.head(1000)[1:],
label='Predicted data exponentiated')
plt.title('Quality of model fitness to the data - Limited window')
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()


plt.figure()
plt.plot(co_test_arima, label='test set')
plt.plot(last_50_hstep, label='h-step')
plt.legend()
plt.title('ARIMA(4,1,3) h-step prediction')
plt.grid(True)
plt.show()

#%%
"""
--------------------------------------------------------------------
--------------------
Holt-Winters Seasonal Method
--------------------------------------------------------------------
---------------------
As the professor mentioned in the class for my question on how to
estimate
the seasonality period is a very difficult question to answer, I
thought it would
be reasonable to do a grid-search. Perhaps the parameter with the
minimal mse
could be considered as the optimal seasonality period. I believe
this is a more
systematic way to approach the estimate the parameter rather than
to making assumption
that may not necessarily hold or rationale at all circumstances.
"""
# Iterate over different seasonality period and try fitting Holts
Winter Seasonal method. Pick the
# seasonality period that yields the minimal MSE value.
#

# mse_list_co = [np.Inf, np.Inf]
#
```

```
# for i in range(2, 400):
#     ets_co = ETS(co, trend=None, damped_trend=False,
#                                        seasonal="mul",
seasonal_periods=i).fit()
#     y_true = co_test.reset_index(drop=True).values.reshape([-1])
#     y_pred = ets_co.forecast(steps=co_test.shape[0])
#     diff = y_true - y_pred
#     mse = 1/len(diff) * (diff.T @ diff)
#     mse_list_co.append(mse)
#
# # we add an extra 2 because the range starts from 2
# seasonal_period = np.argmin(mse_list_co) + 2
seasonal_period = 375  # for convenient debugging...
print(f"The estimated optimal seasonality period would be
{seasonal_period}")

# Fitting HW Seasonal based on estimated seasonality period
ets = ETS(endog=co, seasonal='mul',
seasonal_periods=seasonal_period,
          trend=None).fit()

ets_yhat = ets.predict(start=0, end=co.shape[0]-1)

# Plotting y vs the yhat to illustrate the goodness of fit of the
model to the data
plt.figure()
plt.plot(co.head(100), label='training data')
plt.plot(ets_yhat.head(100), label='predicted')
plt.title('Holts Winter Seasonal Method')
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()

ets_forecast = ets.forecast(steps=co_test.shape[0])
plt.figure()
plt.plot(co, label='training data')
plt.plot(ets_yhat, label='predicted')
plt.plot(co_test, label='test data')
plt.plot(co_test.index.values, ets_forecast.reset_index(drop=True),
label='forecast data')
plt.title('Holts Winter Seasonal Method')
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
```

```
plt.show()

tmp_diff = co_test.reset_index(drop=True).values -
ets_forecast.reset_index(drop=True).values
ets_forecast_rmse = np.sqrt(np.mean(tmp_diff.T @ tmp_diff))

#%% Replace missing values in the exogenous variables using naive
method

# def naive_iterpolate(df, vars):
#     df = df.set_index('Date Local')
#     indices = df.index
#     for var in vars:
#         missing_indices = np.where(df.loc[:, var].isna())[0]
#         # -1 here because .min() refers to the start of missing
indices
#         # and we want values from the last entry of the non-
missing dataset
#         # Assuming the missing_indices are consecutive after
visual verification.
#         for m_index in missing_indices:
#             prev_time_index = indices[m_index - 1]
#             time_index = indices[m_index]
#             df.loc[time_index, var] = df.loc[prev_time_index,
var]
#     return df

def snaive_interpolate(df, vars):
    df = df.set_index('Date Local')
    indices = df.index
    for var in vars:
        missing_indices = np.where(df.loc[:, var].isna())[0]
        # -1 here because .min() refers to the start of missing
indices
        # and we want values from the last entry of the non-missing
dataset
        # Assuming the missing_indices are consecutive after visual
verification.
        for m_index in missing_indices:
            prev_time_index = indices[m_index - seasonal_period]
            time_index = indices[m_index]
            df.loc[time_index, var] = df.loc[prev_time_index, var]
    return df

#%%
"""
Impute one or very few missing values in feature columns
"""
joined_df = joined_df.drop(columns=['Year', 'Day_of_Year',
```

```
'Station_Number'])
data_types = joined_df.dtypes
float_vars = data_types.index.values[data_types == "float64"]
joined_df = joined_df.reset_index()
joined_df.loc[:, float_vars] = snaive_interpolate(joined_df,
float_vars).reset_index()
joined_df = joined_df.set_index('Date Local')

#%%
"""---------------------------------------
Beginning of Regression Analysis
---------------------------------------"""

#%% Remove unncessary variables - At this point, I only wish to
limit the analysis to CO AQI
# and leave out the other AQIs.
import re
data_types = joined_df.dtypes
# float_indices = np.where(data_types == "float64")[0]
# float_vars = joined_df.dtypes.reset_index().iloc[float_indices,
0]
discarded_aqi = ['SO2', 'NO2', 'O3', 'CO']
cnames = joined_df.columns
# p_formula -> pollutant formula
p_cnames = []
for p_formula in discarded_aqi:
    pattern = f"{p_formula}+[\s\w]*"
    p_cnames = np.unique(np.r_[p_cnames, re.findall(pattern, ",
".join(cnames))])
useful_vars = np.setdiff1d(float_vars, p_cnames)
useful_vars = np.r_[useful_vars, ['CO AQI']]

#%% Correlation plot between exogenous variables
plt.figure(figsize=(9, 8))
sns.heatmap(joined_df[useful_vars].corr(), linewidths=.5)
plt.title('Correlation between Exogenous variables')
plt.tight_layout()
plt.show()

#%%
co = joined_df['CO AQI']
X = joined_df[np.setdiff1d(useful_vars, ['CO AQI'])]
X_train = X[:round(0.7 * X.shape[0])]
X_test = X[round(0.7 * X.shape[0]):]
y = co
y_train = y[:round(0.7 * co.shape[0])]
y_test = y[round(0.7 * co.shape[0]):]

print("Printing sample of independent variables Linear Regression -
```

```
not preprocessed yet")
print(X_train.head())

# Standardization
mu = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train_z = (X_train - mu)/std
X_test_z = (X_test - mu)/std

# Eigen analysis
cov = (1/(X_train_z.shape[0]-1)) * (X_train_z.T @ X_train_z)
print(f"Covariance matrix:\n{cov}")
vals, vecs = np.linalg.eig(cov)

# Include intercept to the X feature set
X_train_z['intercept'] = np.ones([X_train_z.shape[0], 1])
X_test_z['intercept'] = np.ones([X_test_z.shape[0], 1])

print("Printing sample of independent variables Linear Regression -
standardized")
print(X_train_z.head())

"""------------------------
OLS on standardized data
-----------------------"""
ols_fit2 = sm.regression.linear_model.OLS(endog=np.log(y_train),
exog=X_train_z).fit()
print("OLS summary on standardized data")
print(ols_fit2.summary())

# Make prediction using the OLS fitted
ols2_train_pred = ols_fit2.predict(exog=X_train_z)
ols2_test_pred = ols_fit2.predict(exog=X_test_z)
diff2 = y_test - np.exp(ols2_test_pred)
ols2_mse = (1/diff2.shape[0]) * (diff2.T @ diff2)

spaced_xlabels = []
for i, index_val in zip(range(X_train.shape[0]),
X_train.index.values):
    if i%375 == 0:
        spaced_xlabels.append(index_val)
    else:
        spaced_xlabels.append(None)

# Plot the quality of fit
plt.figure()
plt.plot(y_train, label='Training data')
plt.plot(np.exp(ols_fit2.predict(X_train_z)), label='Prediction on
training set')
```

```
plt.plot(y_test, label='Testing data')
plt.plot(np.exp(ols_fit2.predict(X_test_z)), label='Prediction on
testing set')
plt.legend()
plt.title('OLS Model on standardized data')
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.show()

# Print the variance/information present in each axis
indices_desc = np.argsort(vals)[::-1]
vals_normalized = vals/np.max(vals)
vals_normalized = vals_normalized[indices_desc]
print(f"Proportion of variance/information in each basis vector
that span the vector space:\n"
      f"{vals_normalized}")

# Decorrelate the data
whiten_W = vecs @ np.diag((1/vals)**.5) @ vecs.T
X_train_w = X_train_z[np.setdiff1d(X_train_z.columns,
['intercept'])] @ whiten_W
X_test_w = X_test_z[np.setdiff1d(X_test_z.columns, ['intercept'])]
@ whiten_W

plt.figure()
sns.heatmap(X_train_w.corr(), linewidths=0.05, linecolor='black')
plt.title(f"Correlation plot of data decorrelated PCA with
sphering")
plt.show()

# Include intercept for whitened data
X_train_w['intercept'] = np.ones([X_train_w.shape[0], 1])
X_test_w['intercept'] = np.ones([X_test_w.shape[0], 1])

"""---------------------------------------------------------------
OLS on orthgonally projected data that is rotation neutralized
The following transformation whitens the data to faciliate/boost
the
linear regression
---------------------------------------------------------------"""
ols_fit = sm.regression.linear_model.OLS(endog=np.log(y_train),
exog=X_train_w).fit()
print("Printing summary of OLS fitted on decorrelated data")
print(ols_fit.summary())

# Plot the quality of fit
plt.figure()
plt.plot(y_train, label='Training data')
plt.plot(np.exp(ols_fit.predict(X_train_w)), label='Prediction on
```

```
training set')
plt.plot(y_test, label='Testing data')
plt.plot(np.exp(ols_fit.predict(X_test_w)), label='Prediction on
testing set')
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.legend()
plt.title('OLS Model on Decorrelated data')
plt.show()

# Making prediction using the model fitted on whitened data
ols1_train_pred = ols_fit.predict(exog=X_train_w)
ols1_test_pred = ols_fit.predict(exog=X_test_w)

# MSE
diff1 = y_test - np.exp(ols1_test_pred)
ols1_mse = (1/diff1.shape[0]) * (diff1.T @ diff1)

# PCA - dimensionality reduction
vals = vals[indices_desc]
vecs = vecs[:, indices_desc]
informative_cols = vals_normalized >= 0.05
retained_evecs = vecs[:, informative_cols]

# Orthogonal projection following including a intercept variable
X_train_l = X_train_z[np.setdiff1d(X_train_z.columns,
['intercept'])]
X_train_l = X_train_l @ retained_evecs
X_train_l['intercept'] = np.ones([X_train_l.shape[0], 1])

X_test_l = X_test_z[np.setdiff1d(X_test_z.columns, ['intercept'])]
X_test_l = X_test_l @ retained_evecs
X_test_l['intercept'] = np.ones([X_test_l.shape[0], 1])

"""-----------------------------------------------------------------
OLS on data transformed onto low dimensional subspace
using orthogonal projection - PCA.
--------------------------------------------------------------"""
ols3_fit = sm.regression.linear_model.OLS(endog=np.log(y_train),
exog=X_train_l).fit()
print("OLS Model on Latent space")
print(ols3_fit.summary())

# Making prediction using the model fitted on data that is reduced
in dimension
ols3_train_pred = ols3_fit.predict(exog=X_train_l)
ols3_test_pred = ols3_fit.predict(exog=X_test_l)

# Plot the quality of fit
```

```
plt.figure()
plt.plot(y_train, label='Training data')
plt.plot(np.exp(ols3_fit.predict(X_train_l)), label='Prediction on
training set')
plt.plot(y_test, label='Testing data')
plt.plot(np.exp(ols3_fit.predict(X_test_l)), label='Prediction on
training set')
plt.legend()
plt.xlabel('Date')
plt.ylabel('CO AQI')
plt.title('OLS Model on PCA data')
plt.show()

diff3 = y_test - np.exp(ols3_test_pred)
ols3_mse = (1/diff3.shape[0]) * (diff3.T @ diff3)

print("Performance on test set")
print(f"RMSE of OLS - Whitened data is {ols1_mse**.5}")
print(f"RMSE of OLS - Standardized data is {ols2_mse**.5}")
print(f"RMSE of OLS - PCA is {ols3_mse**.5}")



#%%
from Utilities.Forecasts import *

#%%
co = joined_df['CO AQI'][:train_test_split]
train_size = co.shape[0]
test_size = co_test.shape[0]

#%%
co_avg_pred_forecast = avg_forecast(x=joined_df['CO AQI'],
T=train_size,
                                    one=True, h=True,
h_length=test_size,
                                    plot=True)
plt.show()
indices = joined_df.index.values

train_indices = indices[:train_size]
test_indices = indices[train_size:]


#%%

co_naive_pred_forecast = naive_method(x=joined_df['CO AQI'],
T=train_size,
                                    one=True, h=True,
```

```
h_length=test_size)
indices = joined_df.index
plt.figure()
plt.plot(joined_df.iloc[1:train_size]['CO
AQI'].reset_index(drop=True), '-b', label='Training')
plt.plot(list(range(train_size, train_size+test_size)),
joined_df.iloc[train_size:]['CO AQI'].reset_index(drop=True), '-g',
label='Validation' )
plt.plot(list(range(train_size, train_size+test_size)),
co_naive_pred_forecast[1], '-', color='orange', label='Testing')
plt.xlabel("Time Index")
plt.ylabel('CO AQI')
plt.title('CO AQI Naive method')
plt.legend()
plt.show()

#%%
co_drift_pred_forecast = drift_forecast(x=joined_df['CO AQI'],
T=train_size,
                                        one=True, h=True,
h_length=test_size,
                                        plot=True)

#%%
co_ses_pred_forecast = ses(x=joined_df['CO AQI'], T=train_size,
h_length=test_size)

plt.figure()
plt.plot(co.reset_index(drop=True), label='Training data')
plt.plot(co_ses_pred_forecast[0], label='Prediction')
plt.plot(range(train_size, train_size+test_size),
co_test.reset_index(drop=True), label='Training data')
plt.plot(range(train_size, train_size+test_size),
co_ses_pred_forecast[1], label='Training data')
plt.legend()
plt.xlabel('Time index')
plt.ylabel('CO AQI')
plt.title('CO AQI - Simple Exponential Smoothing Method')
plt.grid(True)
plt.tight_layout()
plt.show()

#%%

from statsmodels.stats.outliers_influence import
variance_inflation_factor as VIF

cnames = X_train_z.columns
def feature_select_vif(df_train, y_train, target):
```

```
    df_train = pd.concat([pd.DataFrame({'bias_c':
np.ones([df_train.shape[0]])}), df_train.reset_index(drop=True)],
                         axis=1)
    curr_value = 11
    filtered_cnames = np.setdiff1d(cnames, ['bias_c']+target)
    ignored_cols = []
    while curr_value > 10:
        ignore_col = None
        X_tr_subset = df_train[['bias_c'] + list(filtered_cnames)]
        vif_vals =
pd.DataFrame([{'Variable':X_tr_subset.columns[i],
'VIF':VIF(X_tr_subset, i)} for i in range((X_tr_subset.shape[1]))
if VIF(X_tr_subset, i) > 3])
        if X_tr_subset.shape[1] > 1:
            curr_value = vif_vals.VIF.iloc[1:].max()
            max_vif_idx = vif_vals.VIF.iloc[1:].argmax()
            ignore_col =
vif_vals.Variable.iloc[1:].iloc[max_vif_idx]
            ols = sm.regression.linear_model.OLS(y_train.reshape([-
1]), X_tr_subset).fit()
            aic = ols.aic
            bic = ols.bic
            adj_r2 = ols.rsquared_adj

        else:
            final_ols =
sm.regression.linear_model.OLS(y_train.reshape([-1]),
df_train[['bias_c'] + list(filtered_cnames)]).fit()
            return filtered_cnames, ignored_cols, final_ols

        if ignore_col:
            ##--------- Check for improvement
            tmp_filtered_cnames = list(filter(lambda x: x !=
ignore_col, filtered_cnames))
            tmp_X = df_train[['bias_c'] +
list(tmp_filtered_cnames)]
            tmp_ols =
sm.regression.linear_model.OLS(y_train.reshape([-1]), tmp_X).fit()
            new_adj_r2 = tmp_ols.rsquared_adj
            new_bic = tmp_ols.bic
            new_aic = tmp_ols.aic
            if new_adj_r2 < adj_r2 and np.abs(new_adj_r2 - adj_r2)
> 2e-2:
                final_ols =
sm.regression.linear_model.OLS(y_train.reshape([-1]),
df_train[['bias_c'] + list(filtered_cnames)]).fit()
                return ignored_cols, filtered_cnames, final_ols
            ##--------
            ignored_cols.append(ignore_col)
```

```
            filtered_cnames = np.setdiff1d(cnames, ignore_col)
    final_ols = sm.regression.linear_model.OLS(y_train.reshape([-
1]), df_train[['bias_c'] + list(filtered_cnames)]).fit()
    return filtered_cnames, ignored_cols, final_ols

tmp_train = X_train_z[np.setdiff1d(X_train_z.columns,
['intercept'])]
tmp_train['intercept'] = np.ones([tmp_train.shape[0], 1])
filtered_cnames_vif, ignored_cnames_vif, final_ols_vif =
feature_select_vif(tmp_train, y_train.values, target=['CO AQI'])








# cols = np.setdiff1d(X_train_z.columns, ['intercept'])
# vif_vals = pd.DataFrame({"Feature": cols,
"VIF":np.zeros([len(cols), 1])})
# vif_vals = vif_vals.set_index('Feature')
# for col_idx in range(len(cols)):
#     cname = cols[col_idx]
#     vif_vals.loc[cname, 'VIF'] = vif(X_train, col_idx)














# Preprocess weather dataset - separate file.

import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt

main_df = None

ind_cnames = ['Year', 'Day_of_Year', 'Station_Number',
'Air_Temp_Max', 'Air_Temp_Min', 'Air_Temp_Mean',
              'Rel_Humdity_Max', 'Rel_Humidity_Min',
'Rel_Humidity_Mean', 'Vapor_Pressure_Deficit_Mean',
```

```
                'Solar_Radiation_Total', 'Precipitation_Total',
'4_inch_soil_Max', '4_inch_soil_Min',
                '4_inch_soil_Mean', '20_inch_soil_Max',
'20_inch_soil_Min', '20_inch_soil_Mean',
                'wind_speed_Mean',
                'wind_vector_mag', 'wind_vector_dir', 'wind_dir_std',
'max_wind_speed', 'heat_units',
                'reference']
for i in range(2000, 2016+1):
    # if i == 2003:
    #     print('breakpoint...')
    tmp = pd.read_csv(f'Arizona\\{i}.txt', header=None,
                        skip_blank_lines=True)
    tmp = tmp.iloc[:, :len(ind_cnames)].set_axis(ind_cnames,
axis=1)

    dates_df = pd.date_range(start=f'{i}-01-01', end=f'{i}-12-31',
freq='D')
    dates_df = pd.DataFrame({'Date Local':dates_df,
'Day_of_Year':dates_df.dayofyear})
    tmp = tmp.iloc[:dates_df.shape[0]] # upto max_wind_speed
    tmp['Day_of_Year'] = tmp['Day_of_Year'].astype(int)
    # tmp = tmp.sort_values(by='Day_of_Year', ascending=True,
ignore_index=True)
    tmp = pd.merge(left=dates_df, right=tmp, how='left',
on='Day_of_Year', sort=False)

    if main_df is None:
        main_df = tmp
    else:
        main_df = pd.concat([main_df, tmp], axis=0)
```