Air Pollution Forecasting

Name: Soh Hong Yu

Admin Number: P2100775

Class: DAAA/FT/2A/01

Module Code: ST1511 AI and Machine Learning

References (In Harvard format):

1. Archive.ics.uci.edu. 2022. UCI Machine Learning Repository: Air Quality Data Set. [online] Available at: https://archive.ics.uci.edu/ml/datasets/Air+Quality [Accessed 11 August 2022].
2. Wmo.asu.edu. 2022. World Meteorological Organization's World Weather & Climate Extremes Archive. [online] Available at: https://wmo.asu.edu/content/world-lowest-temperature [Accessed 11 August 2022].
3. 2022. [online] Available at: https://pubs.acs.org/doi/10.1021/j100907a003# [Accessed 11 August 2022].
4. WallStreetMojo. 2022. Cointegration. [online] Available at: https://www.wallstreetmojo.com/cointegration/ [Accessed 11 August 2022].
5. Blog.quantinsti.com. 2022. Johansen Cointegration Test: Learn How to Implement it in Python. [online] Available at: https://blog.quantinsti.com/johansen-test-cointegration-building-stationary-portfolio/ [Accessed 11 August 2022].
6. InfluxData. 2022. Autocorrelation in Time Series Data | What is Autocorrelation?. [online] Available at: https://www.influxdata.com/blog/autocorrelation-in-time-series-data/ [Accessed 11 August 2022].
7. Medium. 2022. Interpreting ACF or Auto-correlation plot. [online] Available at: https://medium.com/analytics-vidhya/interpreting-acf-or-auto-correlation-plot-d12e9051cd14 [Accessed 11 August 2022].
8. Support.minitab.com. 2022. Interpret the partial autocorrelation function (PACF) - Minitab. [online] Available at: https://support.minitab.com/en-us/minitab/18/help-and-how-

to/modeling-statistics/time-series/how-to/partial-autocorrelation/interpret-the-results/partial-autocorrelation-function-pacf/ [Accessed 11 August 2022].
9. Medium. 2022. Identifying AR and MA terms using ACF and PACF Plots in Time Series Forecasting. [online]
Available at: https://towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8 [Accessed 11 August 2022].
10. Chicago Tribune. 2011. Ask Tom why: Has the relative humidity ever dropped to zero percent? [online]
Available at: https://www.chicagotribune.com/news/ct-xpm-2011-12-16-ct-wea-1216-asktom-20111216-story.html [Accessed 11 August 2022].
11. Zvornicanin, E., 2022. Choosing the best q and p from ACF and PACF plots in ARMA-type modeling | Baeldung on Computer Science. [online] Baeldung on Computer Science.
Available at: https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling [Accessed 11 August 2022].
12. Analyzing Alpha. 2022. How to Interpret ARIMA Results - Analyzing Alpha. [online]
Available at: https://analyzingalpha.com/interpret-arima-results [Accessed 11 August 2022].
13. Otexts.com. 2022. 3.3 Residual diagnostics | Forecasting: Principles and Practice (2nd ed). [online]
Available at: https://otexts.com/fpp2/residuals.html [Accessed 11 August 2022].
14. Brownlee, J., 2022. Probabilistic Model Selection with AIC, BIC, and MDL. [online] Machine Learning Mastery.
Available at: https://machinelearningmastery.com/probabilistic-model-selection-measures/ [Accessed 11 August 2022].
15. Uber Blog. 2022. Omphalos, Uber's Parallel and Language-Extensible Time Series Backtesting Tool. [online]
Available at: https://eng.uber.com/omphalos/ [Accessed 11 August 2022].

Project Objective

Use a time series model to predict and forecast the amount of air pollutants in open air

Background Information

The dataset seems to be derived from a Air Quality dataset found in UCI Machine Learning Repository, with some of the values in this dataset to match with the Air Quality dataset. Based on the dataset information, here we can assume that the data is curated using 4 metal

oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city.

Initialising Libraries and Variables

In [ ]:
```
# Basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas_profiling import ProfileReport
import seaborn as sns
import warnings
from itertools import product
from pathlib import Path

# Metrics
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error

# Models
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.statespace.varmax import VARMAX

# EDA + Model Selection
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.vector_ar.vecm import coint_johansen
from sklearn.model_selection import TimeSeriesSplit
from statsmodels.tsa.stattools import adfuller

# Pickle Library = Saving Models
import pickle
```

Loading Datasets

In [ ]:
```
df = pd.read_csv("./train.csv", sep=",")
df.head()
```

Out[ ]:

	Date	T	RH	Gas	Value	Unnamed: 5	Unnamed: 6
0	15/3/2016	12.020833	54.883334	CO	1053.200000	NaN	NaN
1	16/3/2016	9.833333	64.069791	CO	995.250000	NaN	NaN
2	17/3/2016	11.292708	51.107292	CO	1025.250000	NaN	NaN
3	18/3/2016	12.866319	51.530903	CO	1064.444444	NaN	NaN
4	19/3/2016	16.016667	48.843750	CO	1088.741667	NaN	NaN

Exploratory Data Analysis

We will begin by conducting an exploratory data analysis of the data, to gain a better understanding of the characteristics of the dataset.

This is a dataset collected from a Air Quality dataset found in UCI Machine Learning Repository, it contains 1312 data points with 7 columns.

Date - date
T - temperature
RH - relative humidity
Gas - type of pollution gas
Value - amount of pollution

Data Exploration

To prevent mutation of our original data, we will make a copy of our data to perform eda on it.

In [ ]:
```
df_eda = df.copy()
```

Descriptive Statistics

In [ ]:
```
df_eda.shape
```

Out[ ]:
```
(1312, 7)
```

There are 1312 rows and 7 columns in the entire data set.

In [ ]:
```
df_eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1312 entries, 0 to 1311
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Date        1312 non-null   object 
 1   T           1312 non-null   float64
 2   RH          1312 non-null   float64
 3   Gas         1312 non-null   object 
 4   Value       1312 non-null   float64
 5   Unnamed: 5  0 non-null      float64
 6   Unnamed: 6  0 non-null      float64
dtypes: float64(5), object(2)
memory usage: 71.9+ KB
```

Observations

- The shape of dataset is (1312, 7) whereby there is 1312 observations and 7 columns. (4 Features + 1 Target Variable: "Value" + 2 Useless Variable)
- Datatype of all columns are numeric except Date and Gas
- There is no missing values in the entire dataset.

Data Information

In [ ]:
```
descriptive_stats = df_eda.describe(include="all").T
descriptive_stats["Proportion of Most Frequent Value"] = (
    descriptive_stats["freq"] / len(df_eda) * 100
)
descriptive_stats.sort_values("Proportion of Most Frequent Value", ascending=False)
```

Out[ ]:

	count	unique	top	freq	mean	std	min	25%	50
Gas	1312	4	CO	328	NaN	NaN	NaN	NaN	Na
Date	1312	328	15/3/2016	4	NaN	NaN	NaN	NaN	Na
T	1312.0	NaN	NaN	NaN	11.634917	37.041779	-200.0	12.38776	18.83229
RH	1312.0	NaN	NaN	NaN	39.873729	43.217836	-200.0	36.348177	46.07899
Value	1312.0	NaN	NaN	NaN	750.218839	225.520132	-160.0	648.370833	769.37083
Unnamed: 5	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Unnamed: 6	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Observations

- All 4 Gases have equal amount of data points as proportion of most frequent value is 25%
- T and RH are likely to be negatively skewed with many outliers as the min value of T and RH is much lower than the lower quartile

Editing columns date and dropping useless columns

In [ ]:
```
# converting date to a datetime object, then set as index
df_eda["Date"] = pd.to_datetime(df_eda["Date"], format="%d/%m/%Y")
df["Date"] = pd.to_datetime(df["Date"], format="%d/%m/%Y")
df_eda.set_index("Date", inplace=True)
df.set_index("Date", inplace=True)
```

```
# dropping useless columns
df_eda = df_eda.loc[:, ~df_eda.columns.str.contains("^Unnamed")]
df = df.loc[:, ~df.columns.str.contains("^Unnamed")]
display(df)
```

Date	T	RH	Gas	Value
2016-03-15	12.020833	54.883334	CO	1053.200000
2016-03-16	9.833333	64.069791	CO	995.250000
2016-03-17	11.292708	51.107292	CO	1025.250000
2016-03-18	12.866319	51.530903	CO	1064.444444
2016-03-19	16.016667	48.843750	CO	1088.741667
...	...	...	...	...
2017-02-01	5.267708	39.614930	O3	553.180556
2017-02-02	-55.515972	-24.010417	O3	343.500000
2017-02-03	-14.272917	28.563542	O3	334.458333
2017-02-04	4.848611	37.832986	O3	379.513889
2017-02-05	7.273958	31.809375	O3	947.333333

1312 rows × 4 columns

Pandas-Profiling

Pandas-Profiling is a convenient tool to quickly explore the datasets along with some alerts/warnings about the dataset

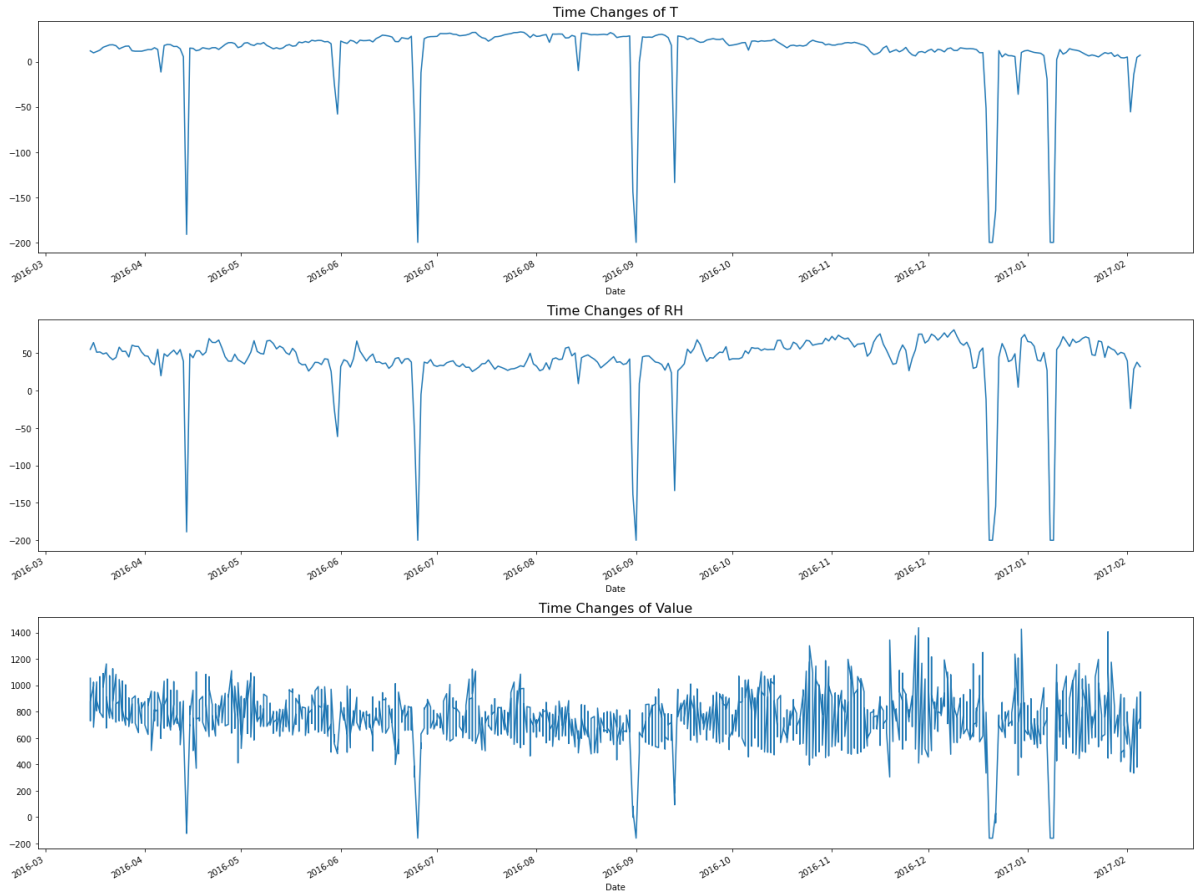
In [ ]:
```
prof = ProfileReport(df_eda, explorative=True)
# prof.to_notebook_iframe()
# prof.to_file(output_file='AirPollution.html')
```

In [ ]:
```
# Summary 1
print("Summary")
print(f"1. T is strong correlated with RH = {df_eda.corr()['T']['RH']}")
print(f"2. T is correlated with Value = {df_eda.corr()['T']['Value']}")
print(f"3. RH is correlated with Value = {df_eda.corr()['RH']['Value']}")
print(
    f"4. The different gases are uniformly distributed: \n{df_eda['Gas'].value_cour
)
```

```
Summary
1. T is strong correlated with RH = 0.9033185659007723
2. T is correlated with Value = 0.663510936586675
3. RH is correlated with Value = 0.6929274128393887
4. The different gases are uniformly distributed:
CO     328
HC     328
NO2    328
O3     328
Name: Gas, dtype: int64
```

Time Series Visualisation

In [ ]:
```
continuous = ["T", "RH", "Value"]
fig = plt.figure(tight_layout=True, figsize=(20, 15))
for i, column in enumerate(continuous):
    ax = fig.add_subplot(len(continuous), 1, i + 1)
    df_eda[column].plot(ax=ax)
    ax.set_title("Time Changes of " + column, fontsize=16)
plt.show()
```

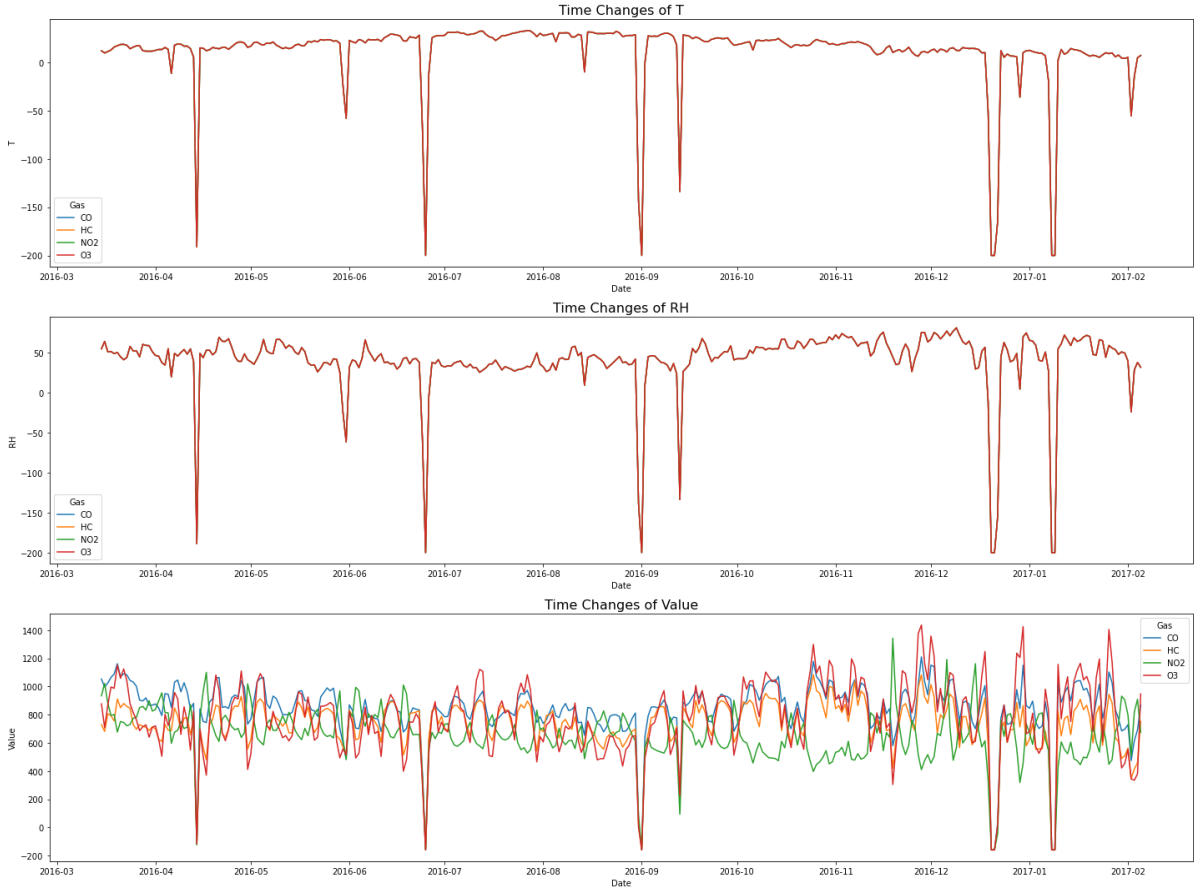


Summary / Observations

- The Value Series looks very clustered and have many spikes which means that there is a high variance in quantity of the different gases.
- There seems to have a reoccurring drop to the negative values in the series.

- As it is likely unlikely for temperature and relative humidity to be in the negative value, as the lowest recorded value in the world is -89.2°C in Antarctica at an elevation of 3420m.
- Therefore, it is likely that a fault might have affected the temperature sensor and humidity sensor.
- Note that because of the dips the different value would also decrease

In [ ]:
```
fig = plt.figure(tight_layout=True, figsize=(20, 15))
for i, c in enumerate(continuous):
    ax = fig.add_subplot(len(continuous), 1, i + 1)
    sns.lineplot(data=df_eda, x=df_eda.index, y=c, hue="Gas", ax=ax)
    ax.set_title("Time Changes of " + c, fontsize=16)
plt.show()
```



Observations

- The data of RH and T are the same for all the different gases.
- The spikes down of temperature and relative humidity is the same point where the value of the different gases down as well.

Univariate Analysis

We will begin with a univariate analysis, analysing the distribution of each variable.

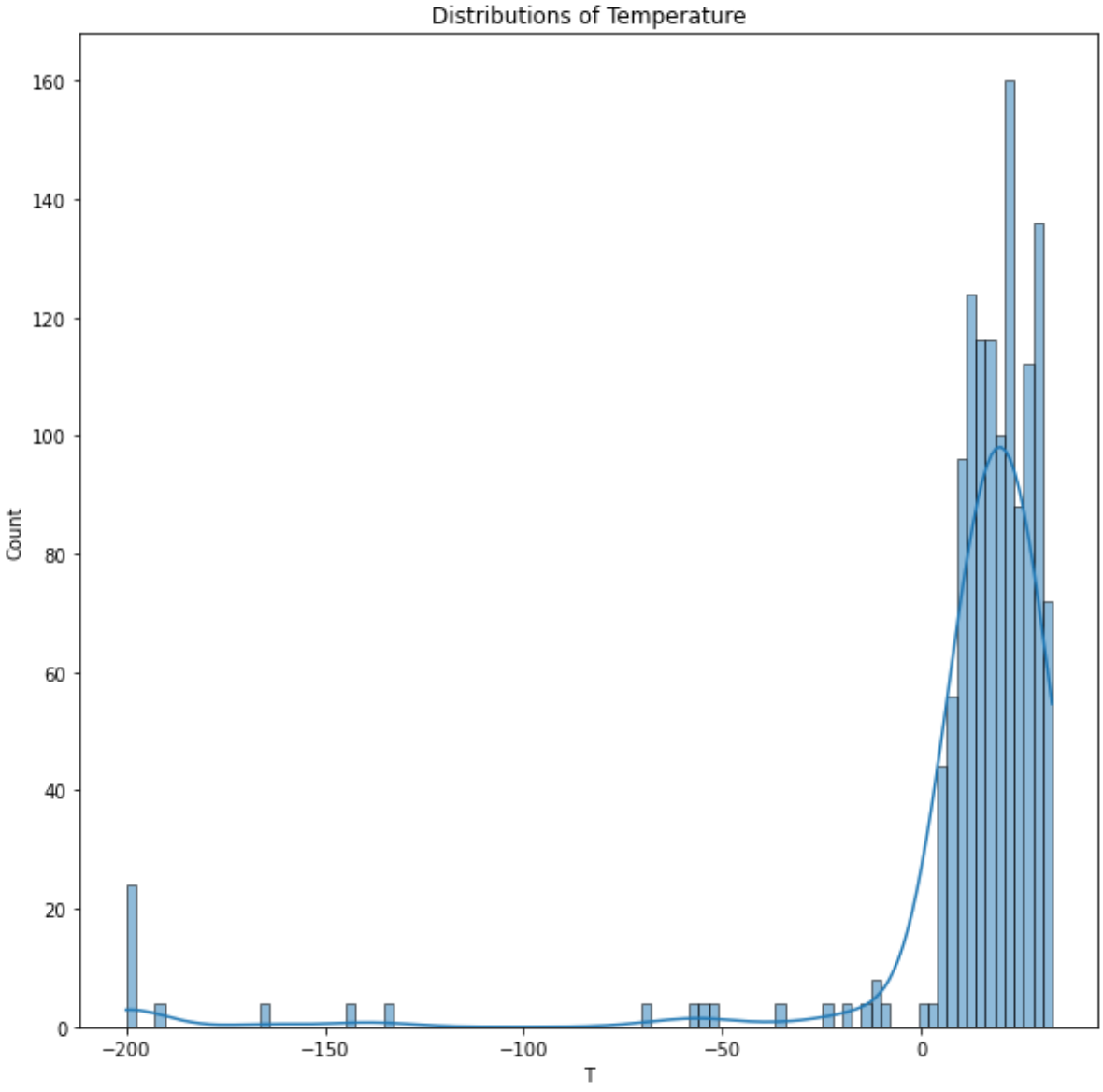
Distributions

We will plot and see the distribution of the different variables and how it relates to the different Gases

Temperature

As mentioned, Temperatures are all the same for the different gases and therefore, we can just compare temperature's Distribution

In [ ]:
```
plt.figure(figsize=(10, 10))
sns.histplot(data=df_eda, x="T", kde=True)
plt.title("Distributions of Temperature")
plt.show()
```



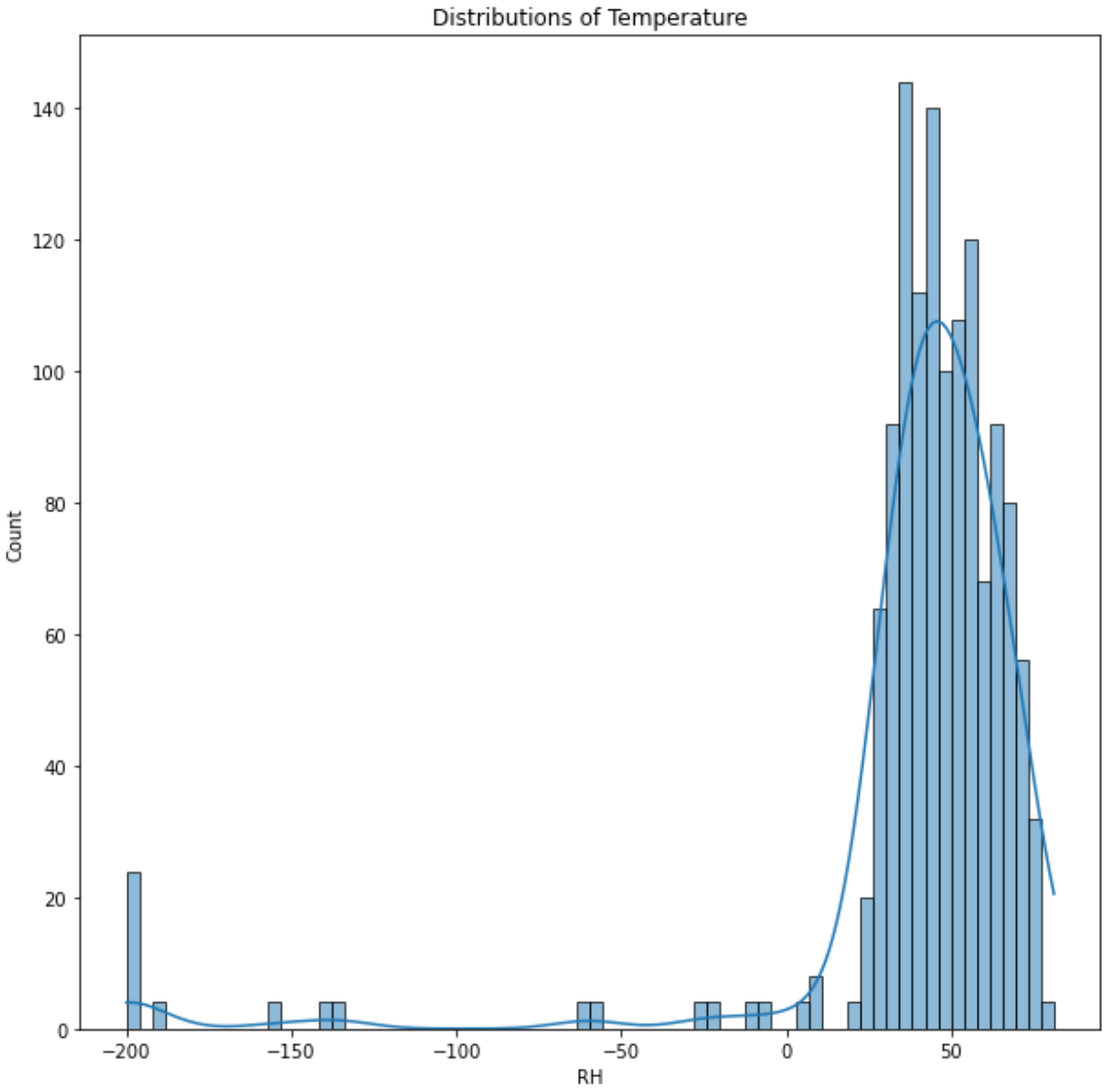
Observations

- As mentioned previously, temperature is very negatively skewed and have several points of outliers likely due to failure of sensors.

Relative Humidity

As mentioned, Relative Humidity are all the same for the different gases and therefore, we can just compare Relative Humidity's Distribution

In [ ]:
```
plt.figure(figsize=(10, 10))
sns.histplot(data=df_eda, x="RH", kde=True)
plt.title("Distributions of Temperature")
plt.show()
```



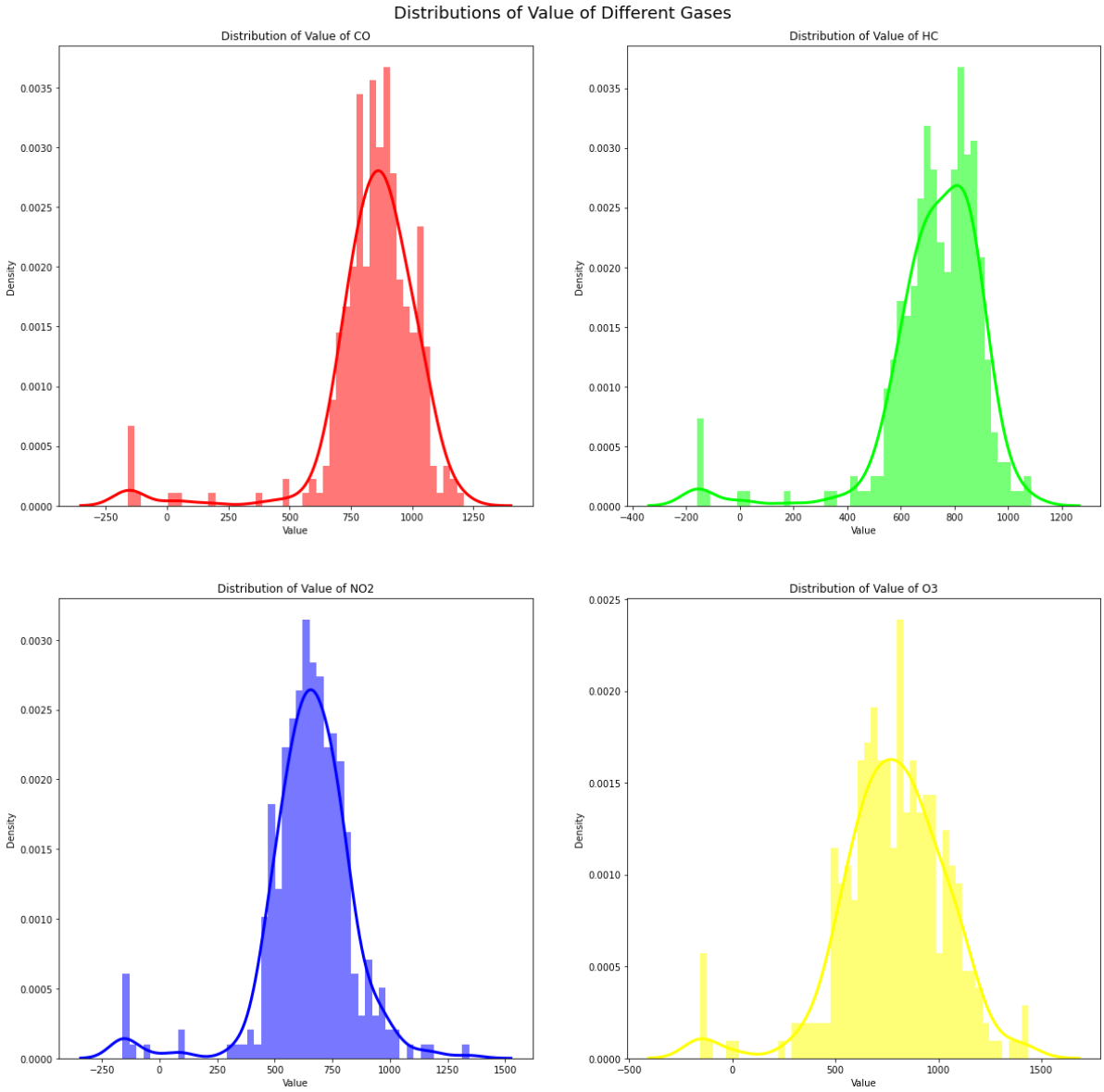
Observations

- As mentioned previously, relative humidity is very negatively skewed and have several points of outliers likely due to failure of sensors.

Value

As the values of the gases are varied, we will look at the distribution of each gases individually.

In [ ]:
```
# Value
fig, ax = plt.subplots(2, 2, figsize=(20, 20))
gasColor = ["#ff0000", "#00ff00", "#0000ff", "#ffff00"]
for gasIndex in range(df_eda["Gas"].nunique()):
    ax[gasIndex // 2][gasIndex % 2].hist(
        df_eda[df_eda["Gas"] == df_eda["Gas"].unique()[gasIndex]]["Value"],
        density=True,
        bins=50,
        color=gasColor[gasIndex] + "88",
    )
    sns.kdeplot(
        df_eda[df_eda["Gas"] == df_eda["Gas"].unique()[gasIndex]]["Value"],
        lw=3,
        color=gasColor[gasIndex],
        ax=ax[gasIndex // 2][gasIndex % 2],
    )
    ax[gasIndex // 2][gasIndex % 2].set_title(
        f"Distribution of Value of {df_eda['Gas'].unique()[gasIndex]}"
    )

plt.suptitle("Distributions of Value of Different Gases", fontsize=18, y=0.91)
plt.show()
```



Observations

- Apart from the outliers, generally the distribution of value of the different gases is very normally distributed.

Time Series Decomposition

Before we can fix some of the outlier issue caused by the failure sensor, we will have to perform some tests to decompose the time series problem.

These tests are:

1. Seasonality Decomposition
2. Test for Causation

3. Test for Cointegration
4. Test for Stationarity
5. Outlier Analysis
6. Autocorrelation Analysis

Before we start any test, we will have to manipulate the data a bit

In [ ]:
```
df_pivot = df_eda.copy(deep=True)
df_pivot = df_pivot.reset_index()
df_pivot = df_pivot.pivot(index="Date", columns="Gas", values="Value").join(
    df_eda[["T", "RH"]]
)
df_pivot = df_pivot[~df_pivot.index.duplicated(keep="first")]
df_pivot
```

Out[ ]:

Date	CO	HC	NO2	O3	T	RH
2016-03-15	1053.200000	729.800000	933.800000	876.833333	12.020833	54.883334
2016-03-16	995.250000	681.441667	1021.750000	708.025000	9.833333	64.069791
2016-03-17	1025.250000	806.583333	881.375000	867.375000	11.292708	51.107292
2016-03-18	1064.444444	794.258333	794.527778	996.625000	12.866319	51.530903
2016-03-19	1088.741667	755.083333	800.883333	987.341667	16.016667	48.843750
...	...	...	...	...	...	...
2017-02-01	729.422222	562.650000	797.647222	553.180556	5.267708	39.614930
2017-02-02	474.291667	347.480556	508.180556	343.500000	-55.515972	-24.010417
2017-02-03	615.700000	414.475000	819.733333	334.458333	-14.272917	28.563542
2017-02-04	691.713889	458.947222	909.375000	379.513889	4.848611	37.832986
2017-02-05	867.600000	751.833333	673.741667	947.333333	7.273958	31.809375

328 rows × 6 columns

Seasonal Decomposition

The seasonal decomposition is a method used in time series analysis to represent a time series as a sum (or, sometimes, a product) of three components - the linear trend, the periodic (seasonal) component, and random residuals.

In a gist, the following plots will be able to split the data into its seasonal components, trend components and remainders. And if you were to sum up the values, you will have the actual data back.

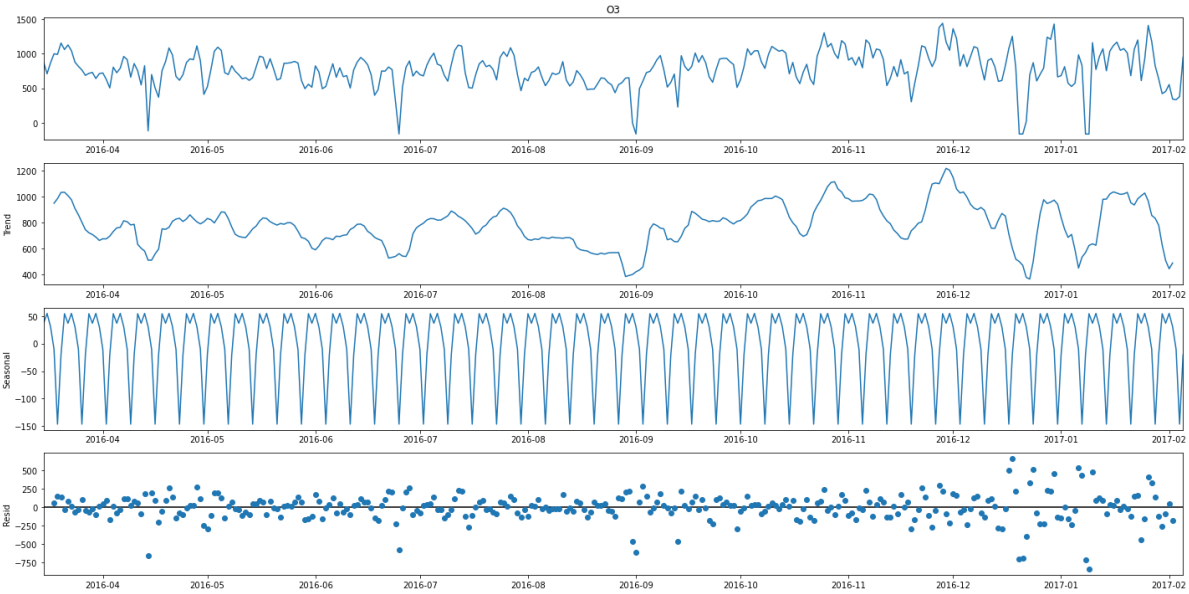
In [ ]:
```
for i in df_pivot.columns:
    plt.rc("figure", figsize=(20, 10))
    print("Seasonal Decomposition\nColumn:", i)
    decomposition = seasonal_decompose(df_pivot[i])
    print("Seasonal", len(decomposition.seasonal.drop_duplicates()))
    decomposition.plot()
    plt.show()
```

Seasonal Decomposition
Column: CO
Seasonal 7

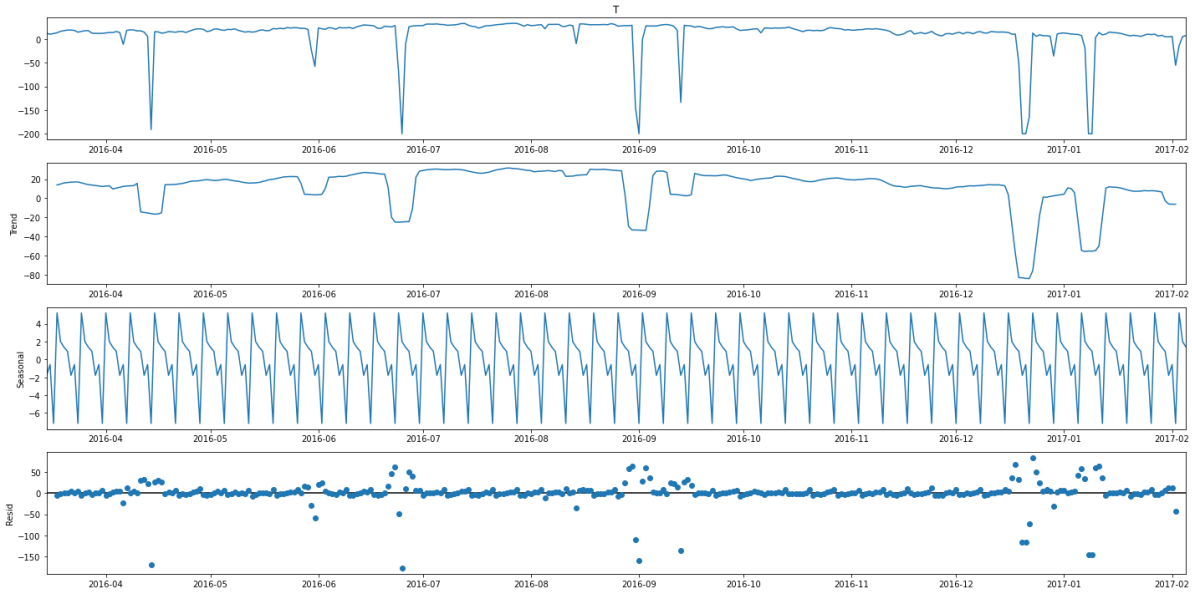
Seasonal Decomposition
Column: HC
Seasonal 7

Seasonal Decomposition
Column: NO2
Seasonal 7

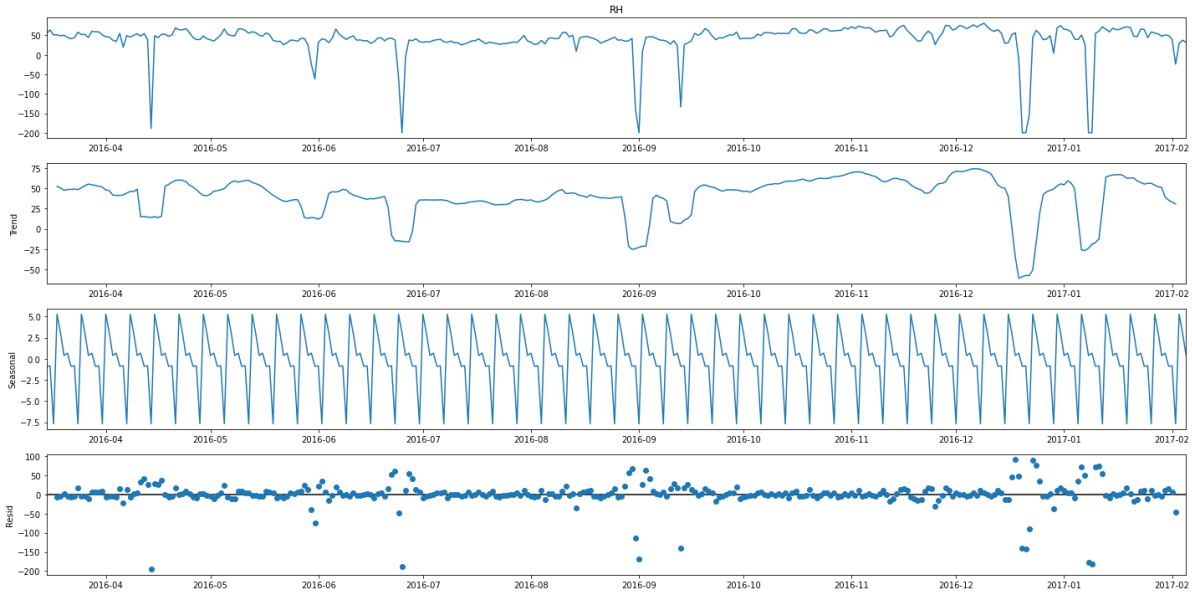
Seasonal Decomposition
Column: O3
Seasonal 7



Seasonal Decomposition
Column: T
Seasonal 7




```
Seasonal Decomposition
Column: RH
Seasonal 7
```



Observations

- There is no linear trend in the data
- Seasonality is the same throughout all the variables. This repeat occurs every 5 days/steps (each tick is around 30 days as a month usually has 30 days. The repeat occurs 4 times a month which means that the repeat occurs like every 30/4 = 5 days)
- The number of unique seasonal count is 7 which we will be using for the models later on

Test for Causation

We will be using the Granger causality test. It is a statistical hypothesis test for determining whether one time series is a factor and offer useful information in forecasting another time series. This will help us in concluding if a Multivariate Modelling can be used for this time series.

Granger causality test works like this.

H_0: There is no causality between time series

H_1: There is some causality between time series

In [ ]:
```
maxlag = 12
test = "ssr_chi2test"

g_matrix = pd.DataFrame(
    np.zeros((len(df_pivot.columns), len(df_pivot.columns))),
    columns=df_pivot.columns,
    index=df_pivot.columns,
)
for c in g_matrix.columns:
    for r in g_matrix.index:
        test_result = grangercausalitytests(
            df_pivot[[r, c]], maxlag=maxlag, verbose=False
        )
        p_values = [round(test_result[i + 1][0][test][1], 4) for i in range(maxlag)
        min_p_value = np.min(p_values)
        g_matrix.loc[r, c] = min_p_value
g_matrix.columns = [var + "_x" for var in df_pivot.columns]
g_matrix.index = [var + "_y" for var in df_pivot.columns]

g_matrix
```

Out[ ]:

	CO_x	HC_x	NO2_x	O3_x	T_x	RH_x
CO_y	1.0000	0.1086	0.2394	0.2212	0.0350	0.0250
HC_y	0.0002	1.0000	0.0111	0.3177	0.0068	0.0020
NO2_y	0.4698	0.1523	1.0000	0.0833	0.1702	0.3143
O3_y	0.0049	0.3991	0.0221	1.0000	0.0140	0.0285
T_y	0.6076	0.7240	0.2505	0.5610	1.0000	0.1257
RH_y	0.2407	0.3673	0.0305	0.1439	0.0380	1.0000

In [ ]:
```
plt.figure(figsize=(10, 10))
sns.heatmap(g_matrix, annot=True, cmap="Blues", vmin=0, vmax=1)
plt.title("Granger's Test for Causality")
plt.show()
```

Observations

- If value is less than the significant level (0.05), then we can reject the null hypothesis and conclude that the x value will granger cause y value.
- Based on this, we can concluded that
    1. Temperature granger cause CO emissions
    2. Relative Humidity granger cause CO emissions
    3. CO granger cause HC
    4. NO granger cause HC
    5. Temperature granger cause HC emissions
    6. Relative Humidity granger cause HC emissions
    7. O3 granger cause NO2
    8. CO granger cause O3
    9. NO2 granger cause O3

    - NO2 causes O3 depletion as NO2 + O3 -> NO3 + O2
10. Temperature granger cause O3 emissions
11. Relative Humidity granger cause O3 emissions
12. NO2 granger cause Relative Humidity
13. Temperature granger cause NO2

Test for Cointegration

Cointegration is a statistical method used to test the correlation between two or more non-stationary time series in the long-run or for a specified time period. The method helps in identifying long-run parameters or equilibrium for two or more sets of variables.

We will be use Johansen's cointegration test is used to established if there is a correlation between several time series in the long term, to determine if there is any significant relationship to use for Multivariate Modelling.

H_0: There is no cointegration

H_1: There is some cotintegram

In [ ]:
```
out = coint_johansen(df_pivot, -1, 5)
d = {"0.90": 0, "0.95": 1, "0.99": 2}
traces = out.lr1
cvts = out.cvt[:, d[str(1 - 0.05)]]


def adjust(val, length=6):
    return str(val).ljust(length)


# Summary
print("Name   ::  Test Stat > C(95%)    =>   Significant  \n", "--" * 20)
for col, trace, cvt in zip(df_pivot.columns, traces, cvts):
    print(
        adjust(col),
        ":: ",
        adjust(round(trace, 2), 9),
        ">",
        adjust(cvt, 8),
        " =>  ",
        trace > cvt,
    )
```

```
Name   ::  Test Stat > C(95%)    =>   Significant
 ----------------------------------------
CO     ::  151.09    > 83.9383   =>   True
HC     ::  90.83     > 60.0627   =>   True
NO2    ::  45.03     > 40.1749   =>   True
O3     ::  19.66     > 24.2761   =>   False
T      ::  4.85      > 12.3212   =>   False
RH     ::  0.64      > 4.1296    =>   False
```

Observations

- Ozone, Temperature and Relative Humidity does not have a significant cointegration
- CO, HC and NO2 have a significant cointegration which suggest that there might be a correlation between the 3 values in the long run.

Test for Stationarity

Stationarity can be defined in precise mathematical terms, but for our purpose we mean a flat looking series, without trend, constant variance over time, a constant autocorrelation structure over time and no periodic fluctuations (seasonality).

This can be tested using the Augmented Dickey-Fuller test also known as the AD Fuller Test to test for stationarity.

H_0: Data is not stationary

H_1: Data is stationary

In [ ]:
```
def adfuller_test(series, significant=0.05, name="", verbose=False):
    r = adfuller(series, autolag="AIC")
    output = {
        "test_statistic": round(r[0], 4),
        "p_value": round(r[1], 4),
        "n_lags": round(r[2], 4),
        "n_obs": r[3],
    }
    p_value = output["p_value"]

    def adjust(val, length=6):
        return str(val).ljust(length)

    # Print Summary
    print(f'Augmented Dickey-Fuller Test on "{name}"', "\n", "-" * 47)
    print(f"Null Hypothesis: Data has unit root. Non-Stationary.")
    print(f"Significance Level    = {significant}")
    print(f'Test Statistic        = {output["test_statistic"]}')
    print(f'No. Lags Chosen       = {output["n_lags"]}')

    for key, val in r[4].items():
        print(f" Critical value {adjust(key)} = {round(val, 3)}")

    if p_value <= significant:
        print(f"=> P-Value = {p_value}. Rejecting Null Hypothesis.")
        print(f"=> Series is Stationary.")
    else:
        print(f"=> P-Value = {p_value}. Weak evidence to reject the Null Hypothesis
        print(f"=> Series is Non-Stationary.")


for name, column in df_pivot.iteritems():
```

```
    adfuller_test(column, name=column.name)
    print("\n")
```

```
Augmented Dickey-Fuller Test on "CO"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -9.5435
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "HC"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -9.6683
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "NO2"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -10.0552
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "O3"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -9.5691
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "T"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -10.251
```

```
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "RH"
 ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -10.2458
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.
```

Observations

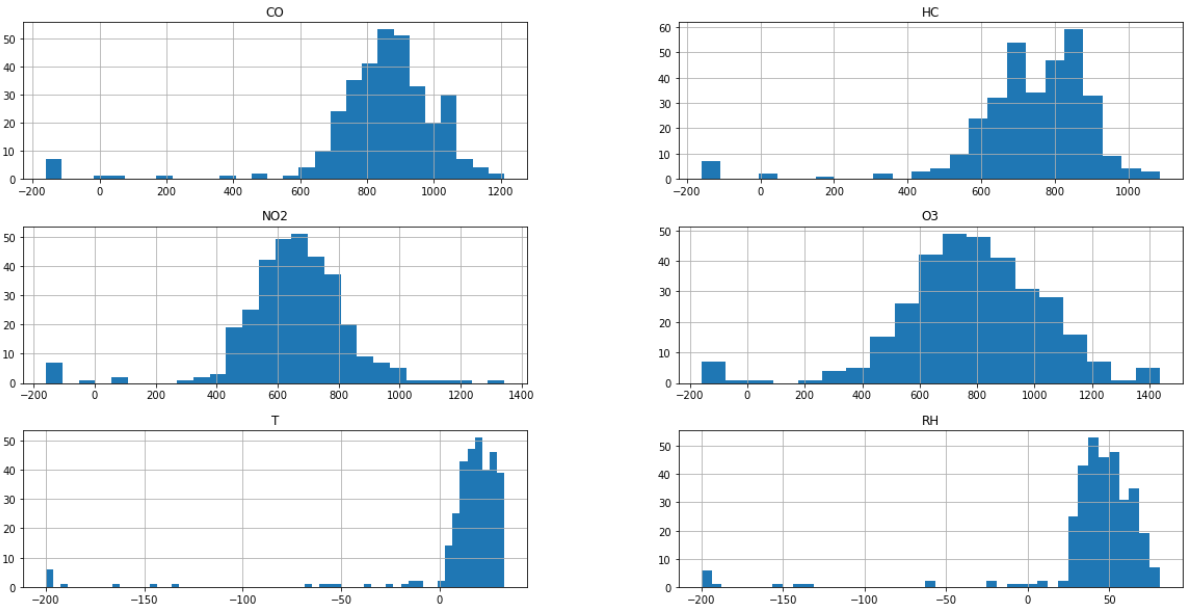
- All the time series appears to be stationary
- As the P-Value of all values are 0.0, this means that which indicates very strong stationary for every variable in this time series

Outlier Analysis

As stated in the distribution part, the dataset has many different outliers and the data is skewed. This data should be voided, deskew or interpolated. We will analysis and see what is the suitable method and we will check if in the end is there any stationary columns.

In [ ]:
```
plt.figure(figsize=(10, 10))
df_pivot.hist(bins="auto")
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



Observations

- Many values are negatively skewed which might have affected the stationarity due to the large variance between the points.
- Based on the lowest temperature recorded on earth, is -89.2°C which means the values are abnormal. To fix the temperature, we will be using the interpolation process

Interpolation of data

As the lowest temperature recorded on earth is -89.2°C, the lowest relative humidity recorded is 1 percent therefore, any values that are less than -89.2°C and 1 percent temperature and relative humidity respectively should be interpolated.

Interpolation is the process of estimating unknown values that falls between 2 known points. We will be using the linear method to interpolate the data using the interpolate method in pandas. [Draws a straight line between the 2 points and estimate using the midway data of the 2 points]

In [ ]:
```
df_inter = df_pivot.copy(deep=True)
df_inter.loc[df_inter["T"] < -89.2, "T"] = np.nan
df_inter.loc[df_inter["RH"] < 0, "RH"] = np.nan
df_inter = df_inter.interpolate(method="time", limit_direction="forward")
df_inter
```

Out[ ]:

Date	CO	HC	NO2	O3	T	RH
2016-03-15	1053.200000	729.800000	933.800000	876.833333	12.020833	54.883334
2016-03-16	995.250000	681.441667	1021.750000	708.025000	9.833333	64.069791
2016-03-17	1025.250000	806.583333	881.375000	867.375000	11.292708	51.107292
2016-03-18	1064.444444	794.258333	794.527778	996.625000	12.866319	51.530903
2016-03-19	1088.741667	755.083333	800.883333	987.341667	16.016667	48.843750
...	...	...	...	...	...	...
2017-02-01	729.422222	562.650000	797.647222	553.180556	5.267708	39.614930
2017-02-02	474.291667	347.480556	508.180556	343.500000	-55.515972	34.089236
2017-02-03	615.700000	414.475000	819.733333	334.458333	-14.272917	28.563542
2017-02-04	691.713889	458.947222	909.375000	379.513889	4.848611	37.832986
2017-02-05	867.600000	751.833333	673.741667	947.333333	7.273958	31.809375

328 rows × 6 columns

In [ ]:

```
plt.title("Time Changes for both T and RH")
sns.lineplot(data=df_inter, x=df_inter.index, y="T")
sns.lineplot(data=df_inter, x=df_inter.index, y="RH")
plt.legend(
    ["T", "RH"],
    loc="upper center",
    bbox_to_anchor=(0.5, -0.1),
    fancybox=True,
    shadow=True,
    ncol=3,
)
plt.show()
```

Observations

After interpolating the data, there is a slight decrease and the dips in the data are more reasonable. However, as we have manipulated the data, the data might not be stationary any more therefore, we will have to do another AD Fuller Test

In [ ]:
```
for name, column in df_inter.iteritems():
    adfuller_test(column, name=column.name)
    print("\n")
```

```
Augmented Dickey-Fuller Test on "CO"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -9.5435
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "HC"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -9.6683
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "NO2"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -10.0552
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "O3"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -9.5691
No. Lags Chosen       = 1
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "T"
 ---------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -8.5363
```

```
No. Lags Chosen       = 0
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


Augmented Dickey-Fuller Test on "RH"
 ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -4.8428
No. Lags Chosen       = 2
 Critical value 1%     = -3.451
 Critical value 5%     = -2.87
 Critical value 10%    = -2.572
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.
```

Observations

- We note that after manipulation of the feature through interpolation, there is no change in the stationarity of the 'T' and 'RH' features

Autocorrelation Analysis

Autocorrelation is intended to measure the relationship between a variable's present value and any past values that you may have access to. In a sense, autocorrelations can help us to see how the previous can be useful for predicting the current value of gases. We will be using ACF and PACF to do comparison.

Autocorrelation: the degree of similarity between the time series and it's lagged version.

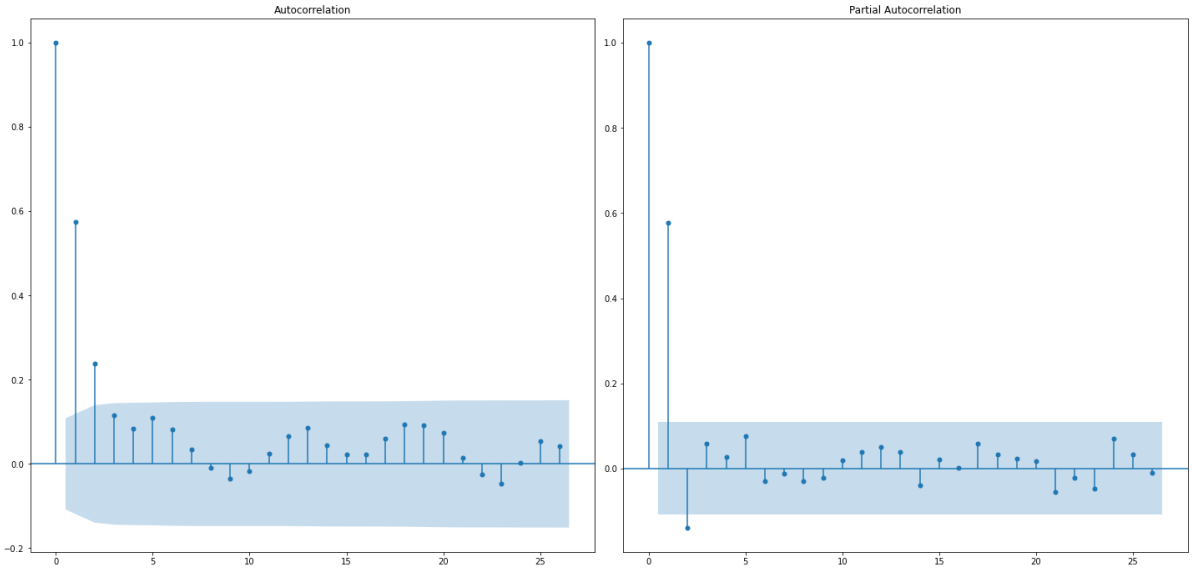
Partial Autocorrelation: the degree similarity between the time series and it's lagged version, without account the mutual relationship with other variables.

The plots will help us in the interpretation of the p and q values for our models below later on.

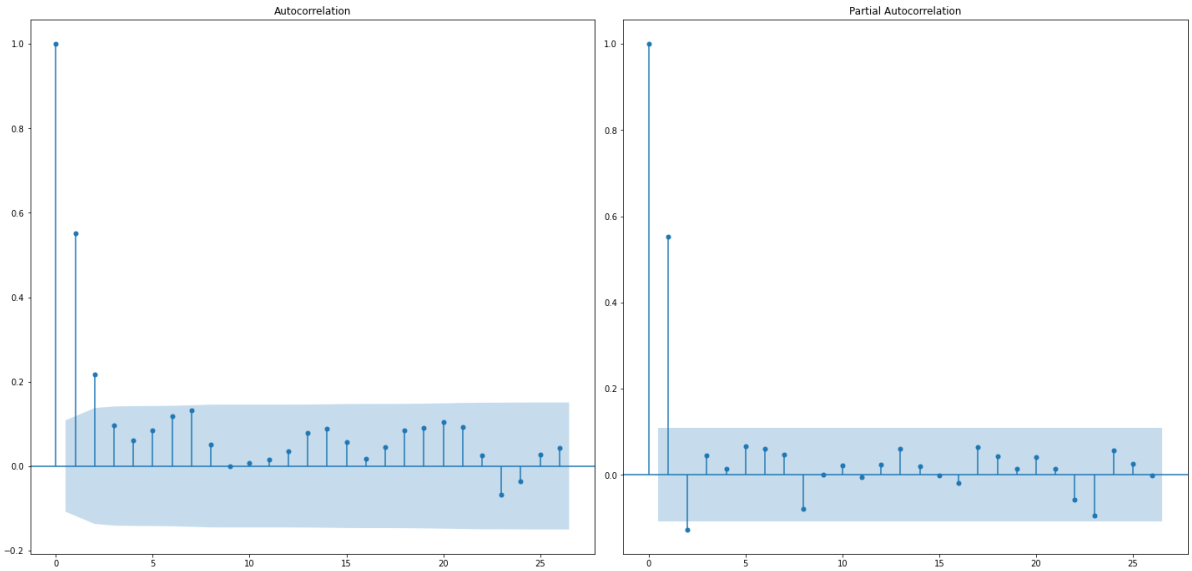
We will be also doing a comparison with the interpolated data as well as the non interpolated data

In [ ]:
```
for i in df_inter.columns:
    fig, ax = plt.subplots(1, 2, tight_layout=True)
    plot_acf(df_inter[i], ax=ax[0])
    plot_pacf(df_inter[i], ax=ax[1])
    fig.suptitle(i, fontsize=16)
    plt.show()
```

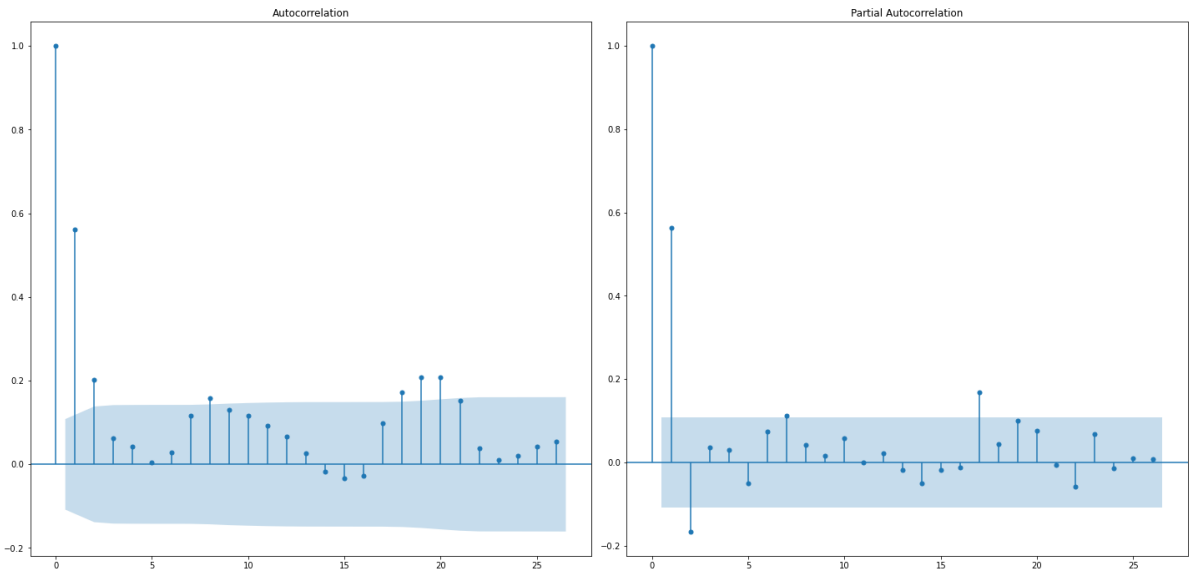
CO


Autocorrelation
Partial Autocorrelation

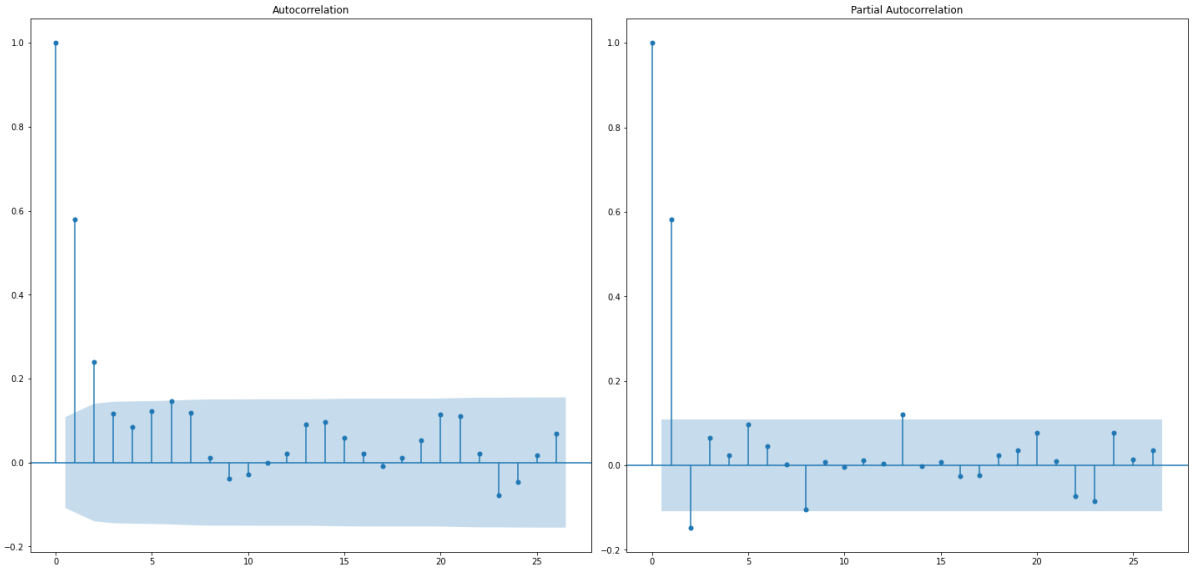
HC


Autocorrelation
Partial Autocorrelation

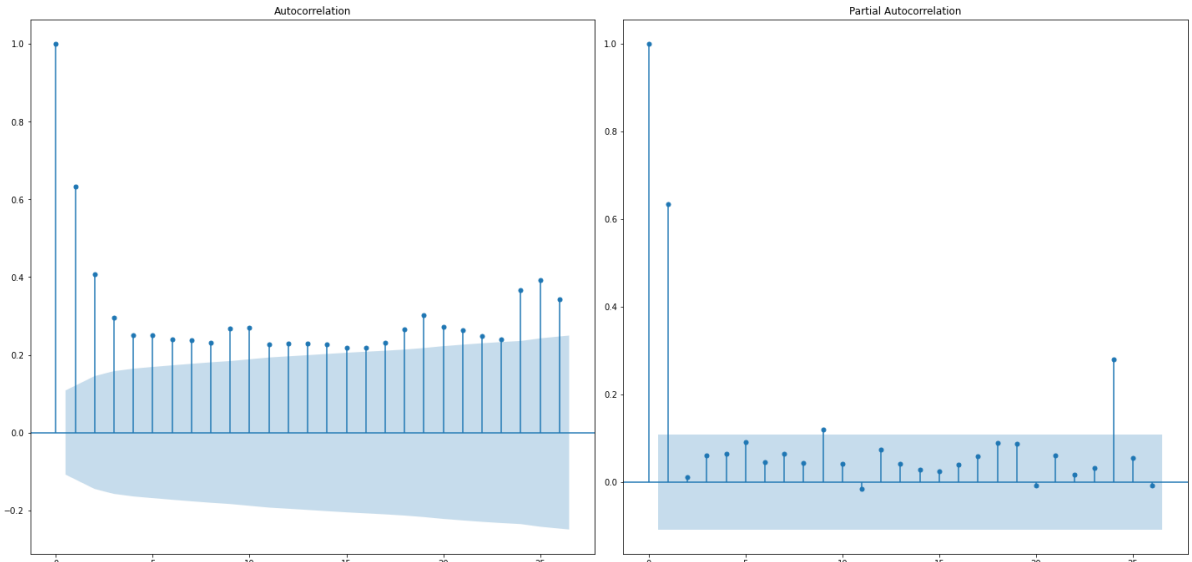
NO2


Autocorrelation
Partial Autocorrelation

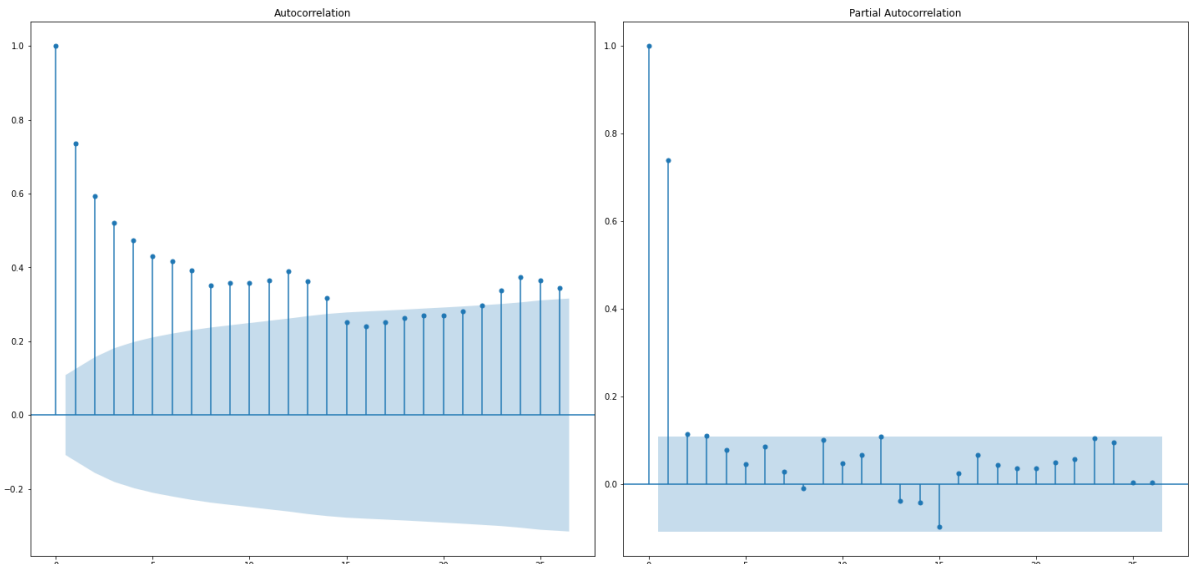
O3


Autocorrelation
Partial Autocorrelation

T


Autocorrelation
Partial Autocorrelation

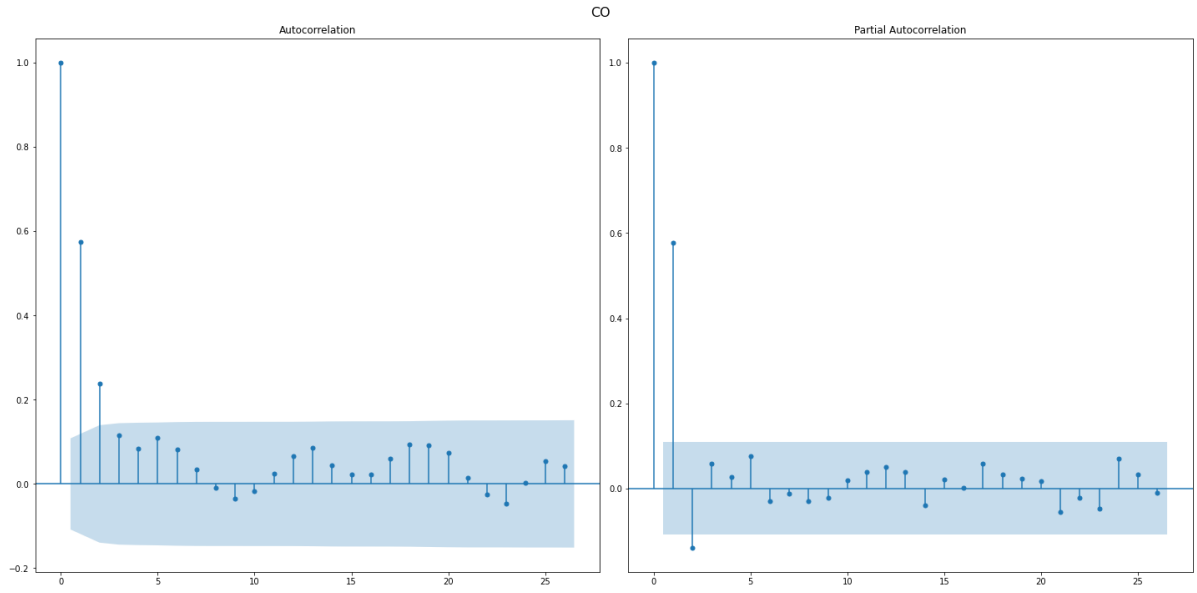
RH


Autocorrelation
Partial Autocorrelation

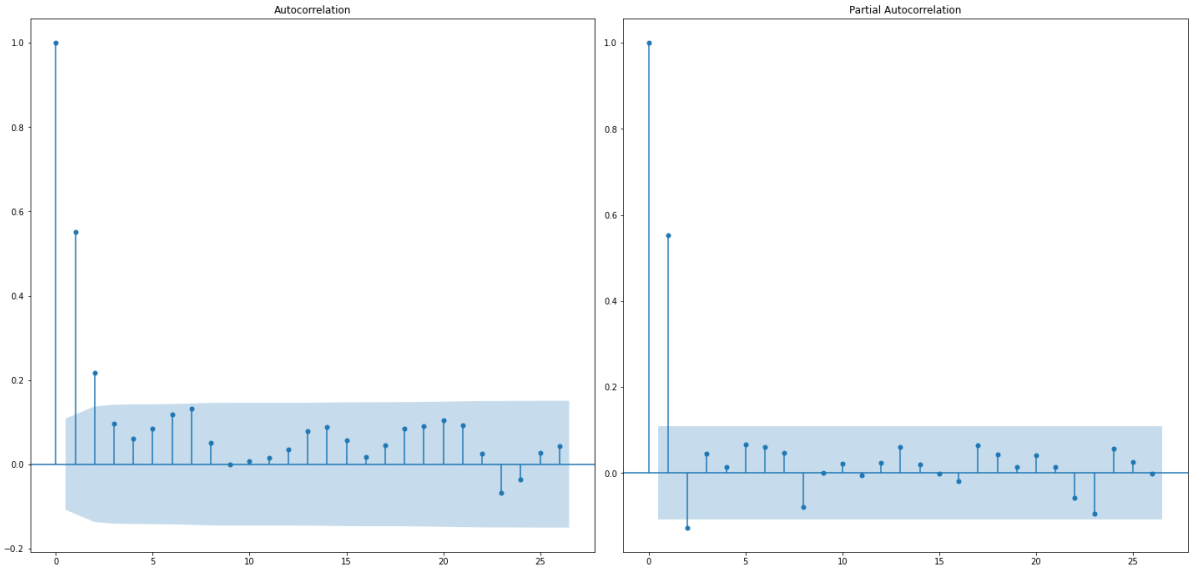
Observations

- We notice that at zero lag, the ACF is always at zero as that variable is 100% correlated with itself. The values inside of the blue bands are statistically insignificant.
- To choose the p value, we will first look at the partial autocorrelation plots. There is a significant lag spike at lag 1 for all the autocorrelation plot which makes p = 1 a prime candidate.
- To choose the q value, we will look at the autocorrelation plot. Any points that are outside of the blue shaded region are potential candidates which means q = 1 - 14 is a good number. q = 1 and q = 2 are better candidates as all correlation plots have the two lags outside of the blue region.

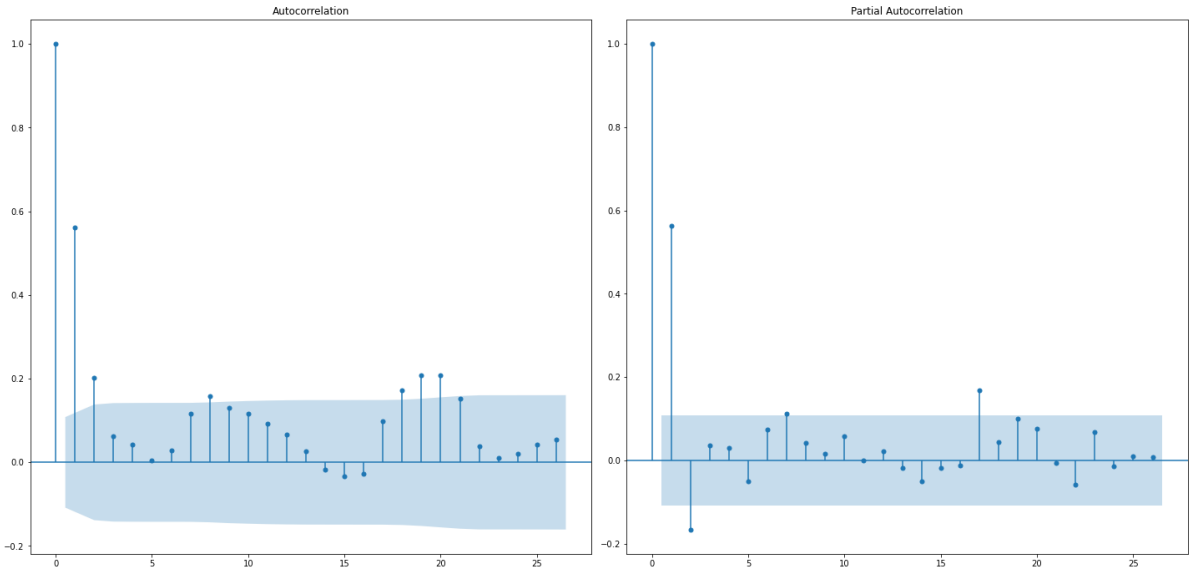
In [ ]:
```
for i in df_pivot.columns:
    fig, ax = plt.subplots(1, 2, tight_layout=True)
    plot_acf(df_pivot[i], ax=ax[0])
    plot_pacf(df_pivot[i], ax=ax[1])
    fig.suptitle(i, fontsize=16)
    plt.show()
```



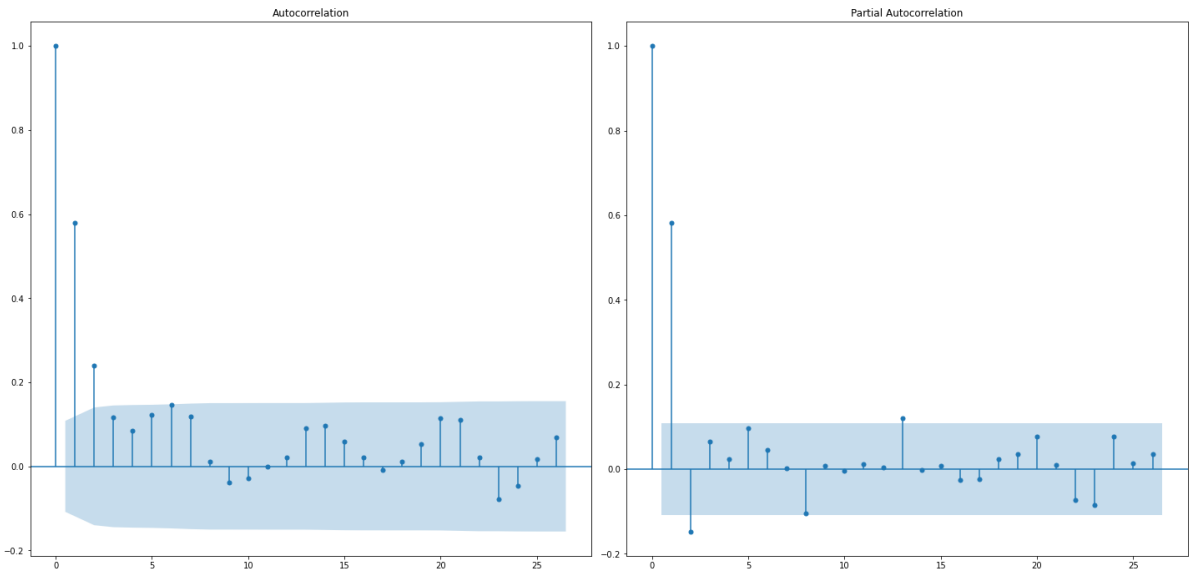
HC

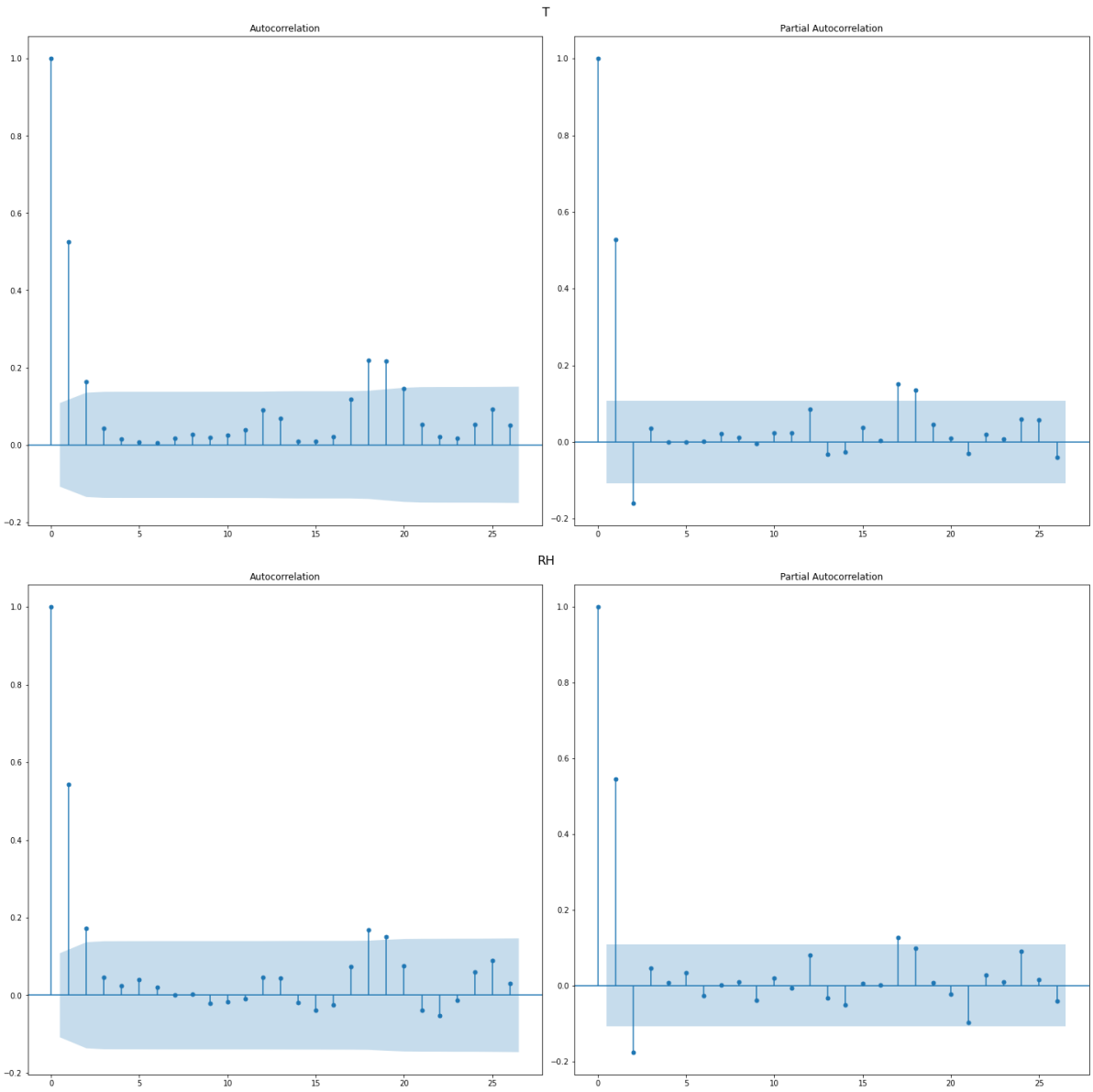

Autocorrelation
Partial Autocorrelation

NO2


Autocorrelation
Partial Autocorrelation

O3


Autocorrelation
Partial Autocorrelation



Observations

- To choose the p value, we will first look at the partial autocorrelation plots. There is a significant lag spike at lag [0,1,2] for all the autocorrelation plot which makes p = [0,1,2] prime candidates.
- To choose the q value, we will look at the autocorrelation plot. Any points that are outside of the blue shaded region are potential candidates which means q = [0,1,2] is a good number.

This values will be used to help us to get the perfect number of AR and MA to train our models

CO (p,q) = (1,2)

HC (p,q) = (1,2)

NO2 (p,q) = (1,2)

O3 (p,q) = (1,2)

Model Selection

There are many models for time series. We will be using 3 models and doing a comparison.

1. ARIMA (Baseline)
2. SARIMA
3. VARMAX

ARIMA vs SARIMA: Autoregressive Integrated Moving Average(ARIMA) is an forecasting algorithm which takes intro account the past values and predict the future value based on the autoregressive and moving average. Seasonal Autoregressive Integrated Moving Average(SARIMA) is very similar to ARIMA but SARIMA takes into account the seasonality patterns in the time series.

ARIMA & SARIMA vs VARMAX Both ARIMA & SARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. While the VARMAX procedure enables you to model the dynamic relationship both between the dependent variables and also between the dependent and independent variables. VARMAX models are defined in terms of the orders of the autoregressive or moving-average process (or both).

The gist is that ARIMA & SARIMA is used for univariate data while VARMAX is used for multivariate data. SARIMA takes into account seasonality and ARIMA does not.

Before we start making the models, we need to make sure the dates in our data has been formatted to individual days and remove all na values

In [ ]:
```
df_model = df_pivot.copy().dropna().to_period("d")
df_model
```

Out[ ]:

Date	CO	HC	NO2	O3	T	RH
2016-03-15	1053.200000	729.800000	933.800000	876.833333	12.020833	54.883334
2016-03-16	995.250000	681.441667	1021.750000	708.025000	9.833333	64.069791
2016-03-17	1025.250000	806.583333	881.375000	867.375000	11.292708	51.107292
2016-03-18	1064.444444	794.258333	794.527778	996.625000	12.866319	51.530903
2016-03-19	1088.741667	755.083333	800.883333	987.341667	16.016667	48.843750
...	...	...	...	...	...	...
2017-02-01	729.422222	562.650000	797.647222	553.180556	5.267708	39.614930
2017-02-02	474.291667	347.480556	508.180556	343.500000	-55.515972	-24.010417
2017-02-03	615.700000	414.475000	819.733333	334.458333	-14.272917	28.563542
2017-02-04	691.713889	458.947222	909.375000	379.513889	4.848611	37.832986
2017-02-05	867.600000	751.833333	673.741667	947.333333	7.273958	31.809375

328 rows × 6 columns

In [ ]:
```
df_inter_model = df_inter.copy().dropna().to_period("d")
df_inter_model
```

Out[ ]:

Date	CO	HC	NO2	O3	T	RH
2016-03-15	1053.200000	729.800000	933.800000	876.833333	12.020833	54.883334
2016-03-16	995.250000	681.441667	1021.750000	708.025000	9.833333	64.069791
2016-03-17	1025.250000	806.583333	881.375000	867.375000	11.292708	51.107292
2016-03-18	1064.444444	794.258333	794.527778	996.625000	12.866319	51.530903
2016-03-19	1088.741667	755.083333	800.883333	987.341667	16.016667	48.843750
...	...	...	...	...	...	...
2017-02-01	729.422222	562.650000	797.647222	553.180556	5.267708	39.614930
2017-02-02	474.291667	347.480556	508.180556	343.500000	-55.515972	34.089236
2017-02-03	615.700000	414.475000	819.733333	334.458333	-14.272917	28.563542
2017-02-04	691.713889	458.947222	909.375000	379.513889	4.848611	37.832986
2017-02-05	867.600000	751.833333	673.741667	947.333333	7.273958	31.809375

328 rows × 6 columns

Splitting data points

In [ ]:
```
x, y = df_model[["T", "RH"]], df_model.drop(["T", "RH"], axis=1)
print(x.shape, y.shape)
```

```
(328, 2) (328, 4)
```

In [ ]:
```
train_size = int(len(df_model) * 0.8)
x_train, x_test = x[:train_size], x[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
print(f"x_train.shape = {x_train.shape}, y_train.shape = {y_train.shape}")
print(f"x_test.shape = {x_test.shape}, y_test.shape = {y_test.shape}")
```

```
x_train.shape = (262, 2), y_train.shape = (262, 4)
x_test.shape = (66, 2), y_test.shape = (66, 4)
```

In [ ]:
```
x_inter, y_inter = df_inter_model[["T", "RH"]], df_inter_model.drop(["T", "RH"], ax
print(x_inter.shape, y_inter.shape)
```

```
(328, 2) (328, 4)
```

In [ ]:
```
train_size = int(len(df_inter_model) * 0.8)
x_inter_train, x_inter_test = x_inter[:train_size], x_inter[train_size:]
y_inter_train, y_inter_test = y_inter[:train_size], y_inter[train_size:]
print(f"x_train.shape = {x_inter_train.shape}, y_train.shape = {y_inter_train.shape
print(f"x_test.shape = {x_inter_test.shape}, y_test.shape = {y_inter_test.shape}")
```

```
x_train.shape = (262, 2), y_train.shape = (262, 4)
x_test.shape = (66, 2), y_test.shape = (66, 4)
```

ARIMA & SARIMA

We will need to make more changes to the dataset as ARIMA & SARIMA is only able to train on time series at a time which means we will have to break the data into 4 different parts for the 4 different Gases.

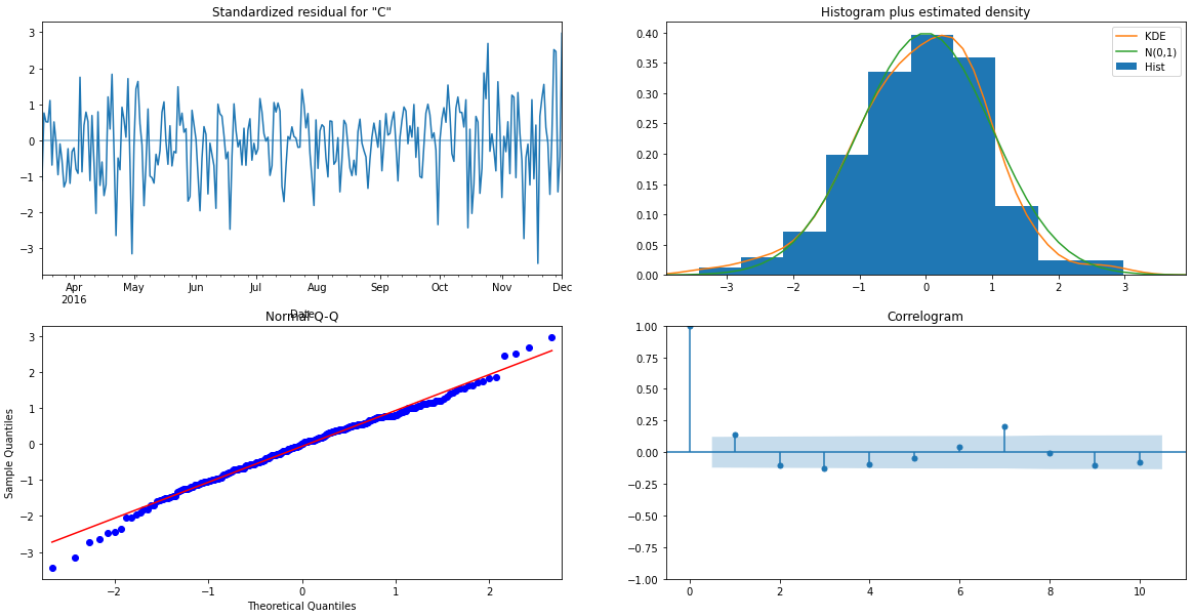
In [ ]:
```
GAS_List = ["CO", "HC", "NO2", "O3"]
CO_train, CO_test = y_train["CO"], y_test["CO"]
HC_train, HC_test = y_train["HC"], y_test["HC"]
NO2_train, NO2_test = y_train["NO2"], y_test["NO2"]
O3_train, O3_test = y_train["O3"], y_test["O3"]

GAS_train = [CO_train, HC_train, NO2_train, O3_train]
GAS_test = [CO_test, HC_test, NO2_test, O3_test]
```

In [ ]:
```
CO_inter_train, CO_inter_test = y_inter_train["CO"], y_inter_test["CO"]
HC_inter_train, HC_inter_test = y_inter_train["HC"], y_inter_test["HC"]
NO2_inter_train, NO2_inter_test = y_inter_train["NO2"], y_inter_test["NO2"]
O3_inter_train, O3_inter_test = y_inter_train["O3"], y_inter_test["O3"]

GAS_inter_train = [CO_inter_train, HC_inter_train, NO2_inter_train, O3_inter_train]
GAS_inter_test = [CO_inter_test, HC_inter_test, NO2_inter_test, O3_inter_test]
```

ARIMA - Baseline

In [ ]:
```
model_Arr = []
for i in range(len(GAS_train)):
    model_Arr.append(ARIMA(endog=GAS_train[i], exog=x_train, order=(1, 1, 1)).fit()
    print(model_Arr[i].summary())
    model_Arr[i].plot_diagnostics()
    plt.show()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                     CO   No. Observations:                  262
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1476.857
Date:                Thu, 11 Aug 2022   AIC                           2963.714
Time:                        19:11:54   BIC                           2981.536
Sample:                    03-15-2016   HQIC                          2970.878
                         - 12-01-2016
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
T              1.1795      0.584      2.021      0.043       0.036       2.323
RH             3.0268      0.511      5.922      0.000       2.025       4.029
ar.L1          0.6957      0.060     11.607      0.000       0.578       0.813
ma.L1         -0.9727      0.025    -39.233      0.000      -1.021      -0.924
sigma2      4789.3585    371.055     12.907      0.000    4062.104    5516.613
===================================================================================
Ljung-Box (L1) (Q):                   4.89   Jarque-Bera (JB):                 7.6
3
Prob(Q):                              0.03   Prob(JB):                         0.0
2
Heteroskedasticity (H):               1.32   Skew:                            -0.2
5
Prob(H) (two-sided):                  0.20   Kurtosis:                         3.6
7
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```



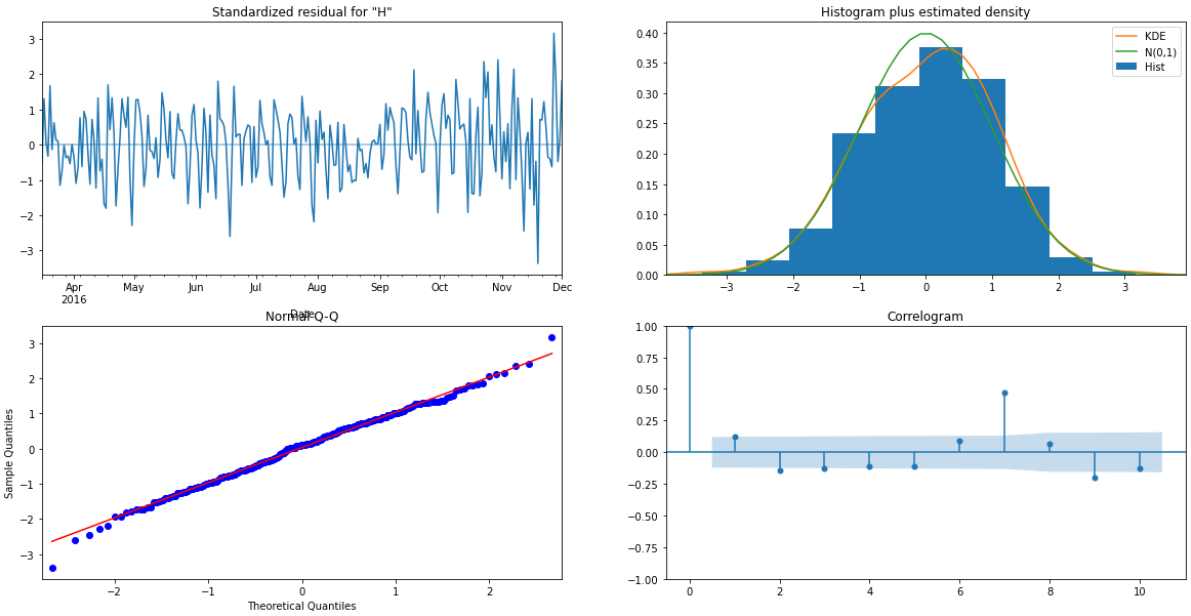
SARIMAX Results

Dep. Variable:	HC	No. Observations:	262
Model:	ARIMA(1, 1, 1)	Log Likelihood	-1534.437
Date:	Thu, 11 Aug 2022	AIC	3078.874
Time:	19:11:57	BIC	3096.696
Sample:	03-15-2016 - 12-01-2016	HQIC	3086.038
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
T	3.0700	0.762	4.028	0.000	1.576	4.564
RH	0.8775	0.670	1.309	0.191	-0.437	2.192
ar.L1	0.5595	0.060	9.388	0.000	0.443	0.676
ma.L1	-0.9634	0.022	-43.311	0.000	-1.007	-0.920
sigma2	7440.1368	653.773	11.380	0.000	6158.766	8721.508

Ljung-Box (L1) (Q):	4.19	Jarque-Bera (JB):	1.12
Prob(Q):	0.04	Prob(JB):	0.57
Heteroskedasticity (H):	1.73	Skew:	-0.14
Prob(H) (two-sided):	0.01	Kurtosis:	3.15

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



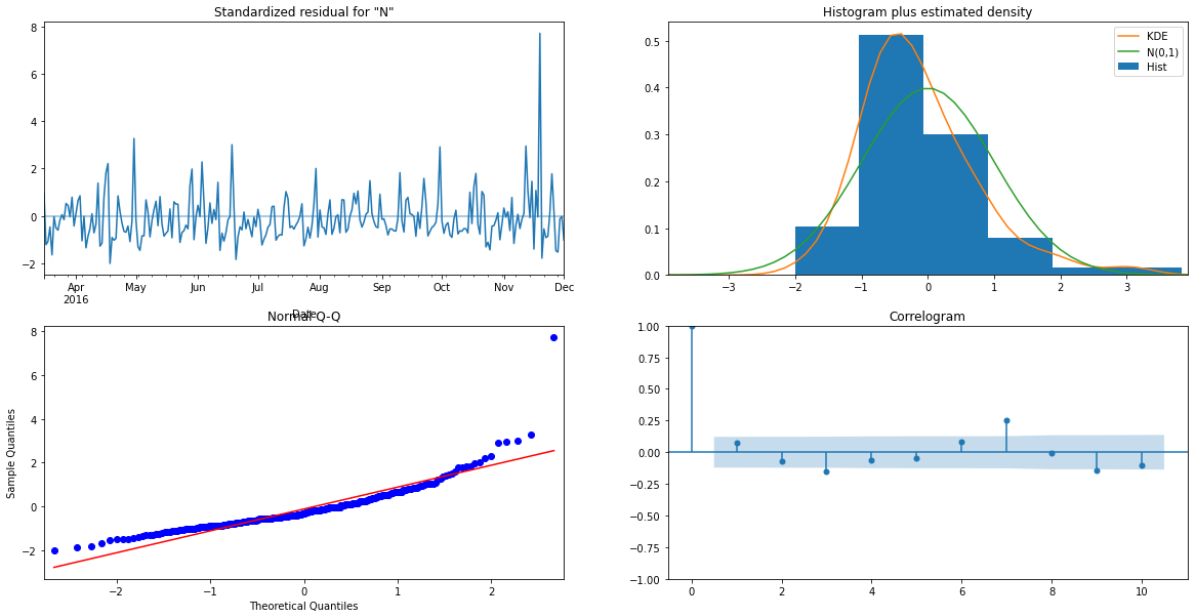
SARIMAX Results

Dep. Variable:	NO2	No. Observations:	262
Model:	ARIMA(1, 1, 1)	Log Likelihood	-1565.483
Date:	Thu, 11 Aug 2022	AIC	3140.965
Time:	19:12:00	BIC	3158.788
Sample:	03-15-2016 - 12-01-2016	HQIC	3148.130
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
T	3.9984	0.873	4.578	0.000	2.287	5.710
RH	-0.2482	0.756	-0.328	0.743	-1.729	1.233
ar.L1	0.5797	0.055	10.451	0.000	0.471	0.688
ma.L1	-0.9589	0.025	-38.017	0.000	-1.008	-0.909
sigma2	9448.0317	314.927	30.001	0.000	8830.787	1.01e+04

Ljung-Box (L1) (Q):	1.61	Jarque-Bera (JB):	2511.97
Prob(Q):	0.20	Prob(JB):	0.00
Heteroskedasticity (H):	1.71	Skew:	2.52
Prob(H) (two-sided):	0.01	Kurtosis:	17.34

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



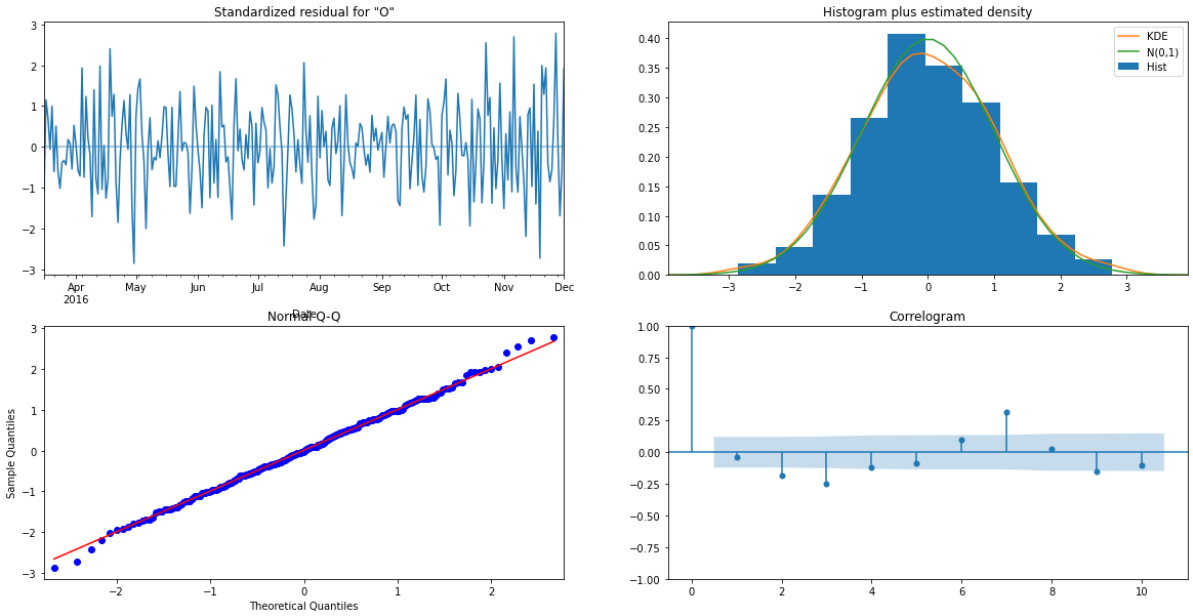
SARIMAX Results

Dep. Variable:	O3	No. Observations:	262
Model:	ARIMA(1, 1, 1)	Log Likelihood	-1687.731
Date:	Thu, 11 Aug 2022	AIC	3385.461
Time:	19:12:02	BIC	3403.284
Sample:	03-15-2016 - 12-01-2016	HQIC	3392.625
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
T	2.6711	1.264	2.113	0.035	0.194	5.149
RH	1.1397	1.104	1.032	0.302	-1.024	3.304
ar.L1	-0.7332	0.365	-2.008	0.045	-1.449	-0.017
ma.L1	0.7826	0.330	2.373	0.018	0.136	1.429
sigma2	2.422e+04	2153.107	11.248	0.000	2e+04	2.84e+04

Ljung-Box (L1) (Q):	0.40	Jarque-Bera (JB):	0.01
Prob(Q):	0.53	Prob(JB):	0.99
Heteroskedasticity (H):	1.32	Skew:	0.01
Prob(H) (two-sided):	0.19	Kurtosis:	2.99

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



Observations

In the P>|z| column, it represents the term significance using a hypothesis testing with the following hypothesis

H_0: Each coefficient is NOT statistically significant

H_1: Each coefficient is statistically significant

The Ljung-Box (L1) (Q) is the LBQ test statistic at lag 1. It is used as a hypothesis testing for the following hypothesis

H_0: The errors are white noise

H_1: The errors are not white noise

Heteroskedasticity (H) tests that the error residuals are homoscedastic or have the same variance.

H_0: The residuals show variances

H_1: The residuals show no variances

Jarque-Bera tests for the normality of errors.

H_0: Data is normally distributed against an alternative of another distribution

H_1: Data is not normally distributed against an alternative of another distribution

The log-likelihood function identifies a distribution that fits best with the sampled data.

Akaike's Information Criterion (AIC) helps determine the strength of the linear regression model. The AIC penalizes a model for adding parameters since adding more parameters will always increase the maximum likelihood value.

Bayesian Information Criterion (BIC), like the AIC, also punishes a model for complexity, but it also incorporates the number of rows in the data.

Hannan-Quinn Information Criterion (HQIC), like AIC and BIC, is another criterion for model selection; however, it's not used as often in practice.

Residuals

The difference between the observations and the corresponding fitted values.

$$e_t = y_t - \hat{y}_t$$

A good forecast will yield the following:

1. The residuals are uncorrelated. If there are correlations between residuals, then there is information left in the residuals which should be used in computing forecasts.
2. The residuals have zero mean. If the residuals have a mean other than zero, then the forecasts are biased.

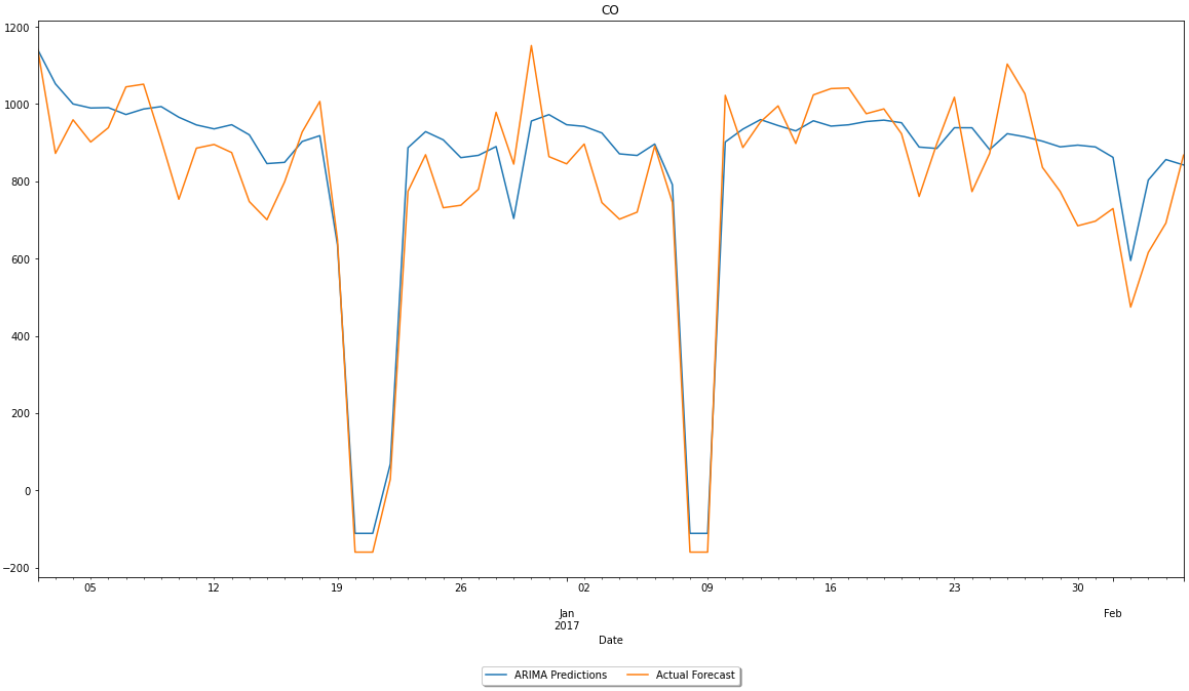
Individual Gases Observations

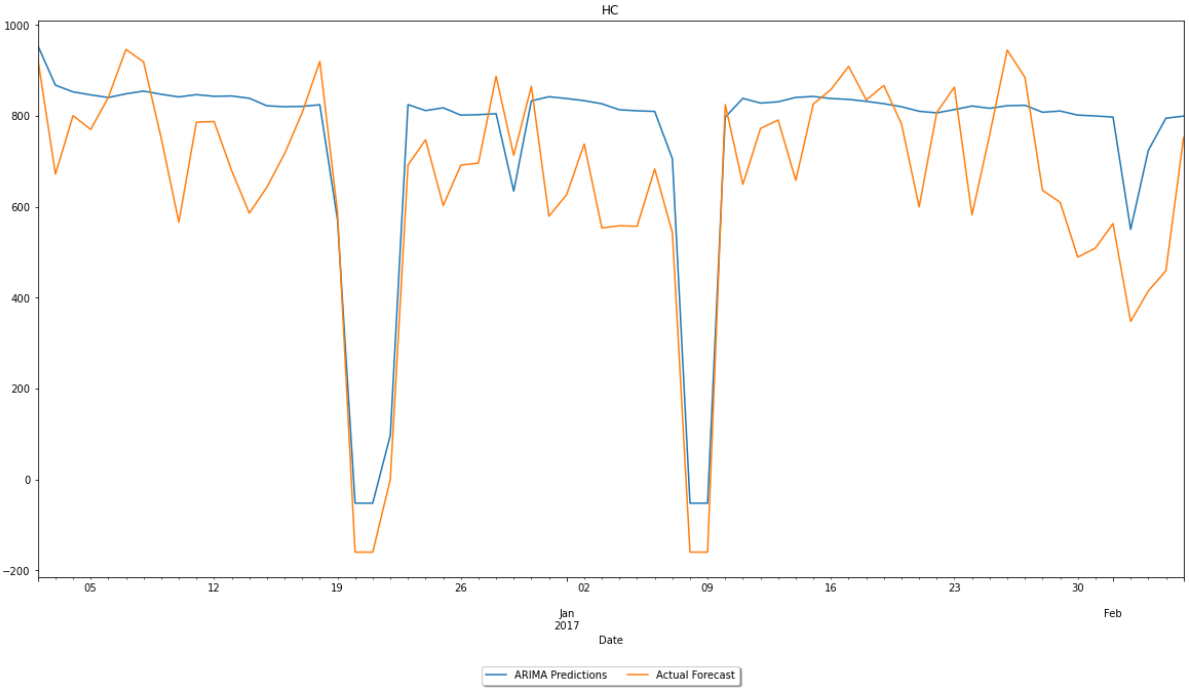
- CO
  - All coefficients are statistically significant
  - The errors caused are white noise
  - The residuals show no variances
  - Data is normally distributed against an alternative of another distribution
- HC
  - All coefficients are statistically significant
  - The errors caused are white noise
  - The residuals show variances
  - Data is not normally distributed against an alternative of another distribution
- NO2
  - RH is not statistically significant as its p-value is 0.302 which is > 0.05
  - The errors caused are not white noise
  - The residuals show variances
  - Data is normally distributed against an alternative of another distribution
- O3
  - RH is not statistically significant as its p-value is 0.743 which is > 0.05
  - The errors caused are not white noise
  - The residuals show no variances
  - Data is not normally distributed against an alternative of another distribution

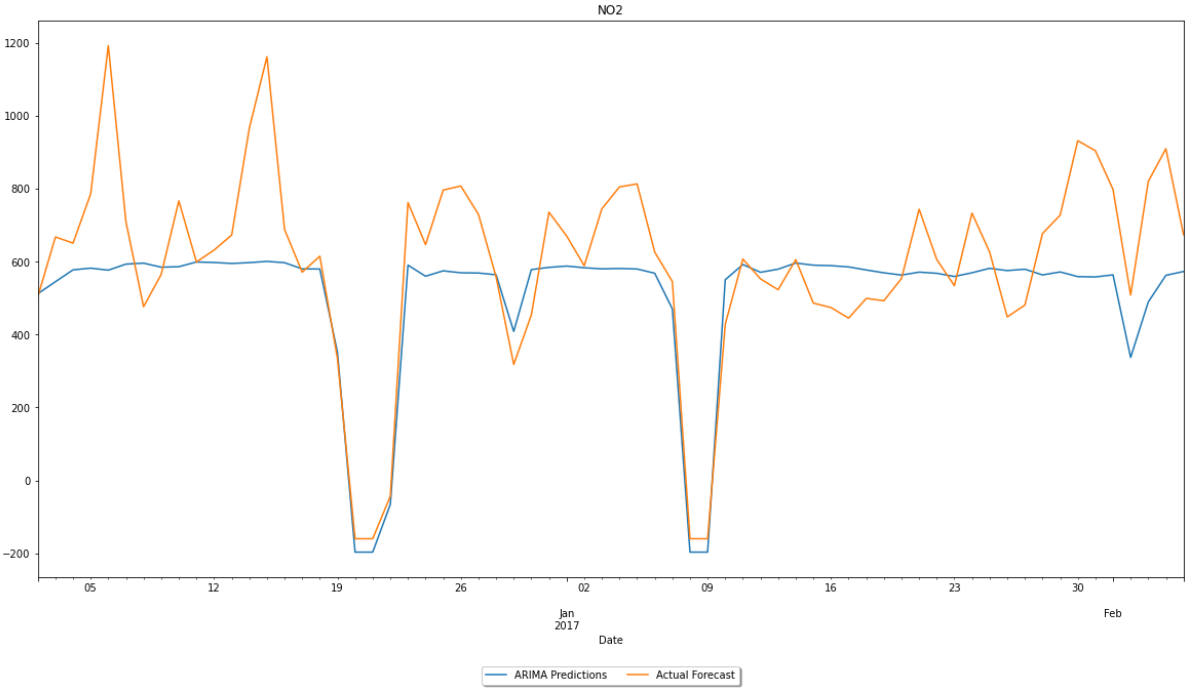
In conclusion, each model will need to be tuned further using their respective orders collected through the autocorrelation and partial autocorrelation plots. Based on the residuals from all the models, it is generally centered to 0 which means it is generally correct however there are a lot of spikes which means the residuals are very staggered. The histogram suggest that the residual may not be normal as they are left skewed, left skewed, right skewed and right skewed respectively for CO,HC,NO2,O3. There is also a spike in correlogram value of 7. This is likely due to the fact that there is a seasonality of 7 in this Time Series which makes lag 7 an important values. This will be remedied using SARIMA as it takes into account for seasonality.

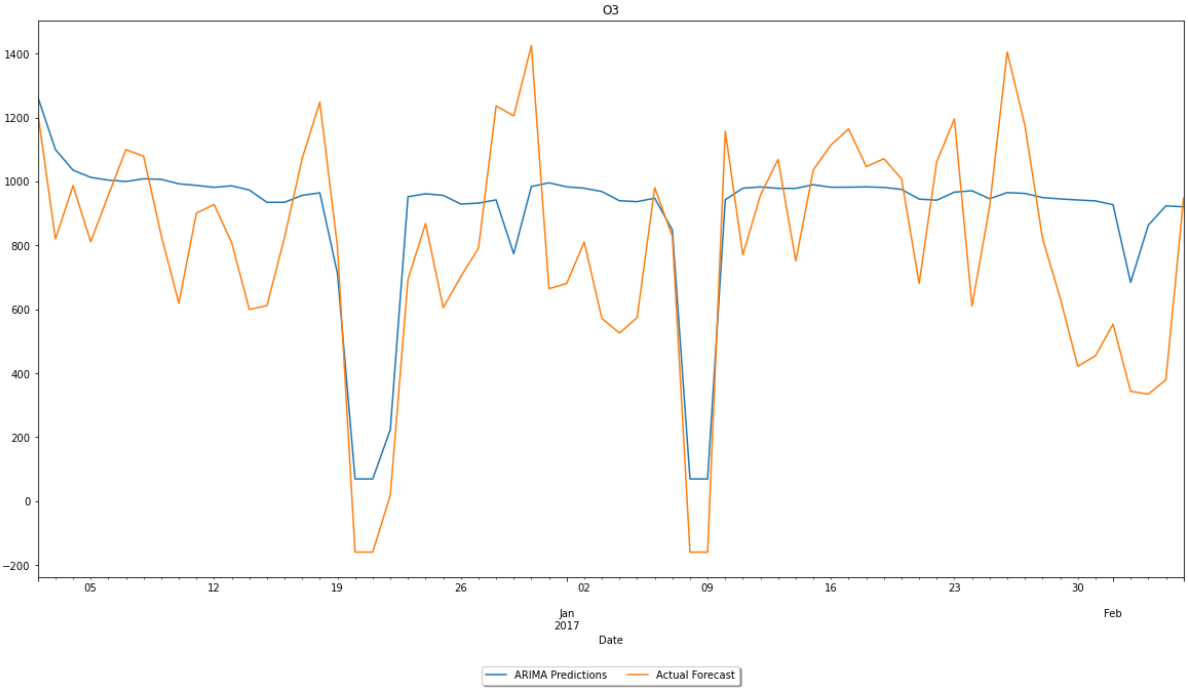
In [ ]:
```
CO_model = ARIMA(endog=CO_train, exog=x_train, order=(1, 1, 2)).fit()
HC_model = ARIMA(endog=HC_train, exog=x_train, order=(1, 1, 2)).fit()
NO2_model = ARIMA(endog=NO2_train, exog=x_train, order=(1, 1, 2)).fit()
O3_model = ARIMA(endog=O3_train, exog=x_train, order=(1, 1, 2)).fit()
CO_pred = CO_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test
HC_pred = HC_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test
NO2_pred = NO2_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_te
O3_pred = O3_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test

GAS_pred = [CO_pred, HC_pred, NO2_pred, O3_pred]
```

In [ ]:
```
for i in range(len(GAS_pred)):
    fig, ax = plt.subplots(1, 1, figsize=(20, 10))
    GAS_pred[i].plot(ax=ax)
    GAS_test[i].plot(ax=ax)
    plt.title(GAS_List[i])
    plt.legend(
        ["ARIMA Predictions", "Actual Forecast"],
        loc="upper center",
        bbox_to_anchor=(0.5, -0.15),
        fancybox=True,
        shadow=True,
        ncol=3,
    )
    plt.show()
```


CO
1200
1000
800
600
400
200
0
−200
05
12
19
26
02
09
16
23
30
Jan
2017
Feb
Date
ARIMA Predictions
Actual Forecast


HC
1000
800
600
400
200
0
−200
05
12
19
26
02
09
16
23
30
Jan
2017
Feb
Date
ARIMA Predictions
Actual Forecast





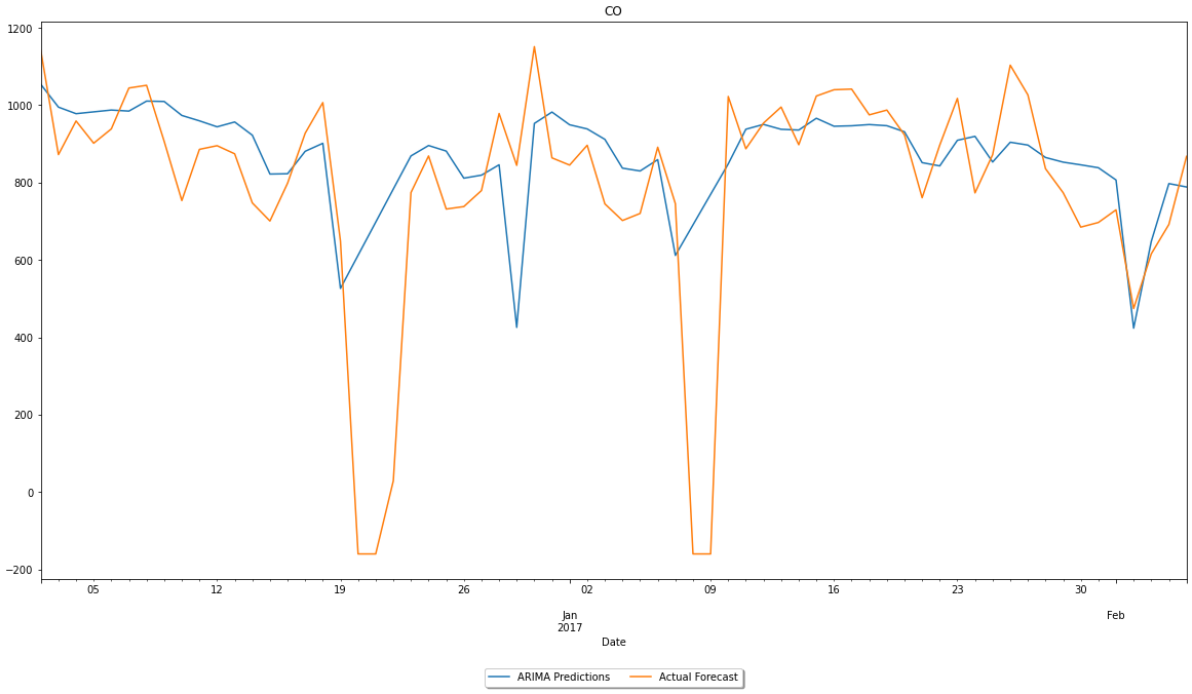
In [ ]:

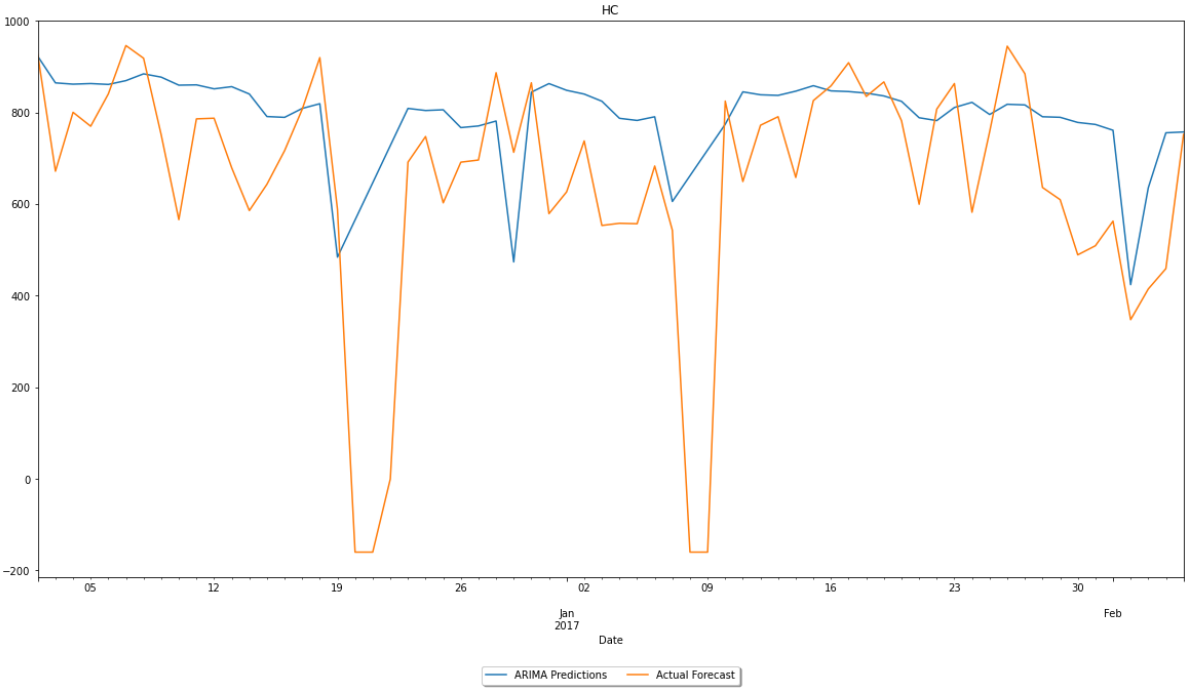
```
CO_inter_model = ARIMA(endog=CO_inter_train, exog=x_inter_train, order=(1, 1, 2)).f
HC_inter_model = ARIMA(endog=HC_inter_train, exog=x_inter_train, order=(1, 1, 2)).f
NO2_inter_model = ARIMA(
    endog=NO2_inter_train, exog=x_inter_train, order=(1, 1, 2)
).fit()
O3_inter_model = ARIMA(endog=O3_inter_train, exog=x_inter_train, order=(1, 1, 2)).f
CO_inter_pred = CO_inter_model.predict(
    start=x_inter_test.index[0], end=x_inter_test.index[-1], exog=x_inter_test
)
HC_inter_pred = HC_inter_model.predict(
    start=x_inter_test.index[0], end=x_inter_test.index[-1], exog=x_inter_test
)
```

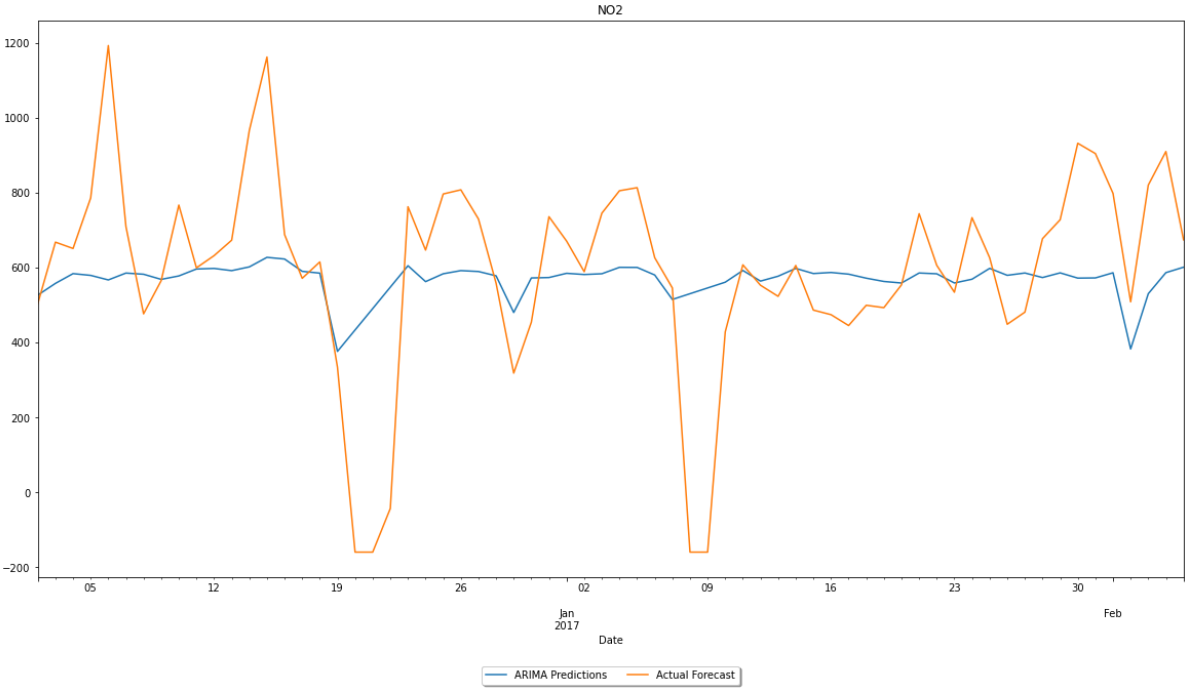
```
NO2_inter_pred = NO2_inter_model.predict(
    start=x_inter_test.index[0], end=x_inter_test.index[-1], exog=x_inter_test
)
O3_inter_pred = O3_inter_model.predict(
    start=x_inter_test.index[0], end=x_inter_test.index[-1], exog=x_inter_test
)

GAS_inter_pred = [CO_inter_pred, HC_inter_pred, NO2_inter_pred, O3_inter_pred]
```

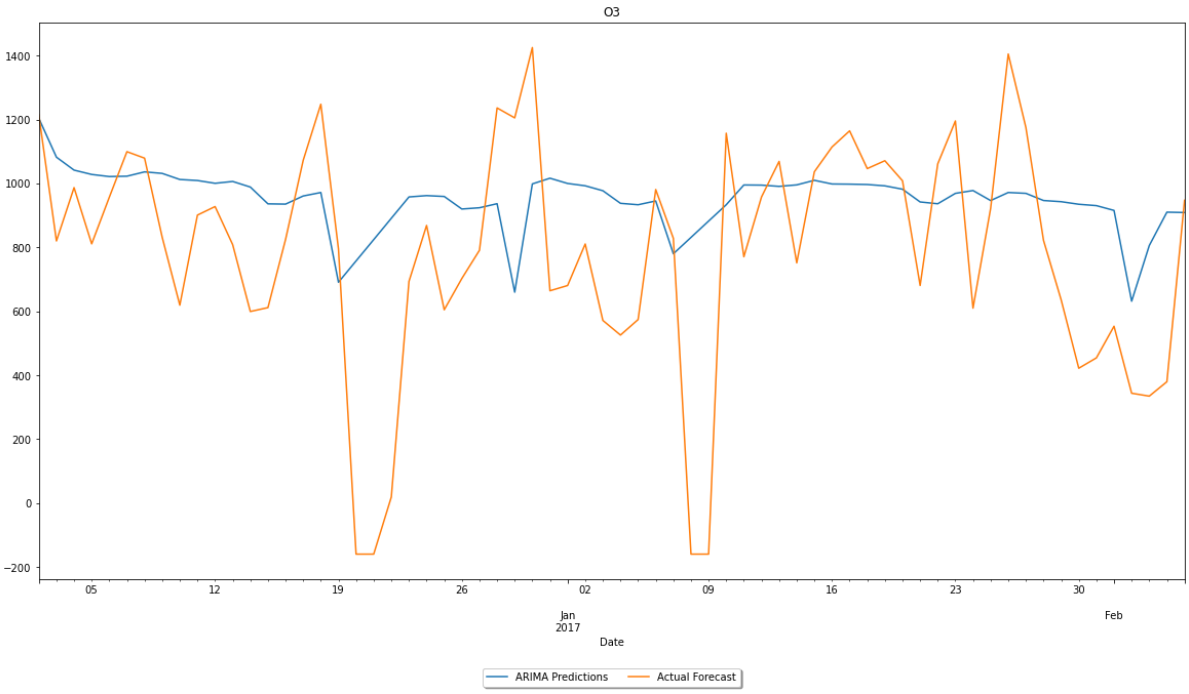
In [ ]:

```
for i in range(len(GAS_inter_pred)):
    fig, ax = plt.subplots(1, 1, figsize=(20, 10))
    GAS_inter_pred[i].plot(ax=ax)
    GAS_inter_test[i].plot(ax=ax)
    plt.title(GAS_List[i])
    plt.legend(
        ["ARIMA Predictions", "Actual Forecast"],
        loc="upper center",
        bbox_to_anchor=(0.5, -0.15),
        fancybox=True,
        shadow=True,
        ncol=3,
    )
    plt.show()
```




HC
1000
800
600
400
200
0
−200
05
12
19
26
02
09
16
23
30
Jan
2017
Feb
Date
ARIMA Predictions
Actual Forecast


NO2
1200
1000
800
600
400
200
0
−200
05
12
19
26
02
09
16
23
30
Jan
2017
Feb
Date
ARIMA Predictions
Actual Forecast



Interpolated vs Original

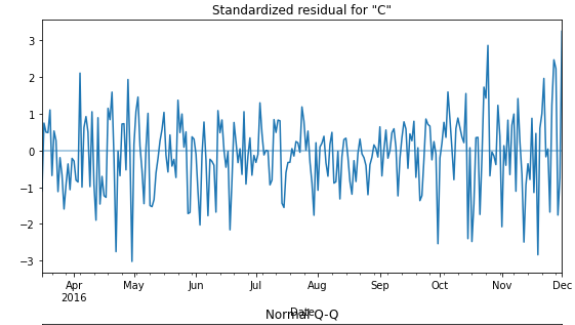
As we can see from the predicted values and graphs, the interpolated dataset did not perform as well as the original dataset. Furthermore, future readings may also fail which we might also need to predict so that we know when the sensors could potentially fail at and fix it in case of emergencies. Because of this, we will be using the no interpolated data(original data) for the rest of the time series predictions.

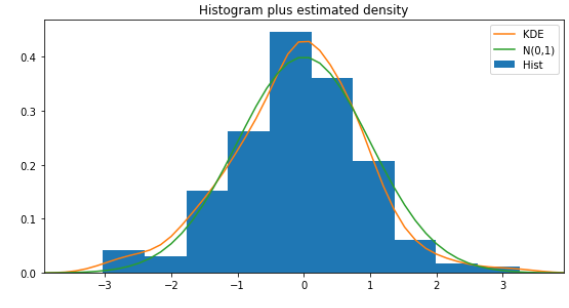
SARIMAX

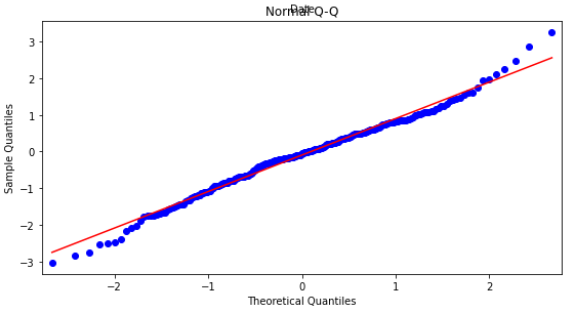
SARIMA will be used to compare with the baseline ARIMA in terms of performance. It should perform better due to SARIMA using more computation power to predict inclusive with seasons. Based on seasonality decomposition, we know that all variables have a seasonality of 7.

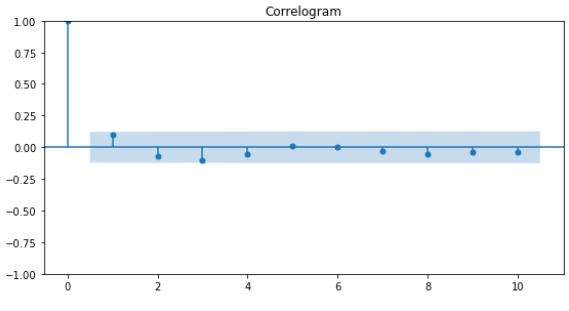
In [ ]:
```
model_Arr = []
for i in range(len(GAS_train)):
    model_Arr.append(
        SARIMAX(
            endog=GAS_train[i],
            exog=x_train,
            order=(1, 1, 1),
            seasonal_order=(2, 0, 0, 7),
        ).fit()
    )
    print(model_Arr[i].summary())
    model_Arr[i].plot_diagnostics()
    plt.show()
```

```
                                SARIMAX Results
==========================================================================================
========
Dep. Variable:                                  CO   No. Observations:
262
Model:             SARIMAX(1, 1, 1)x(2, 0, [], 7)   Log Likelihood               -
1468.197
Date:                            Thu, 11 Aug 2022   AIC
2950.394
Time:                                    19:12:24   BIC
2975.346
Sample:                                03-15-2016   HQIC
2960.424
                                     - 12-01-2016
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
T              1.2716      0.544      2.336      0.019       0.205       2.338
RH             2.9454      0.481      6.124      0.000       2.003       3.888
ar.L1          0.7330      0.047     15.505      0.000       0.640       0.826
ma.L1         -0.9991      0.101     -9.870      0.000      -1.198      -0.801
ar.S.L7        0.2172      0.060      3.639      0.000       0.100       0.334
ar.S.L14       0.1335      0.068      1.976      0.048       0.001       0.266
sigma2      4446.0492    514.360      8.644      0.000    3437.922    5454.176
==========================================================================================
=
Ljung-Box (L1) (Q):                   2.69   Jarque-Bera (JB):                 5.3
0
Prob(Q):                              0.10   Prob(JB):                         0.0
7
Heteroskedasticity (H):               1.29   Skew:                            -0.1
2
Prob(H) (two-sided):                  0.23   Kurtosis:                         3.6
6
==========================================================================================
=

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

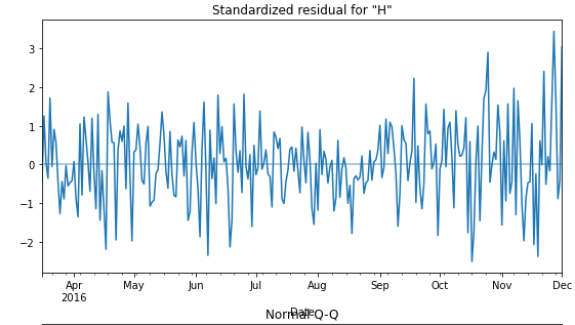

Standardized residual for "C"
Apr
2016
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date

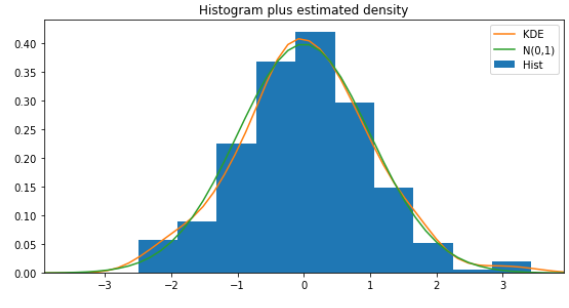

Histogram plus estimated density
KDE
N(0,1)
Hist

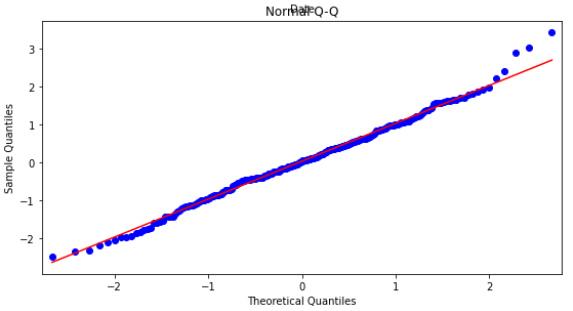

Normal Q-Q
Sample Quantiles
Theoretical Quantiles

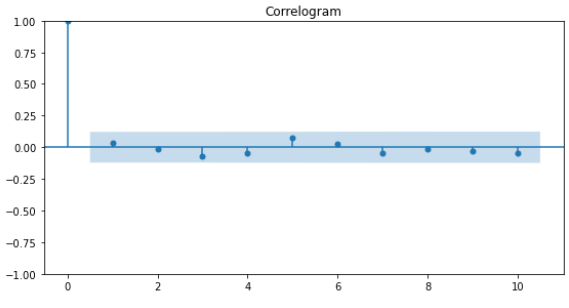

Correlogram

```
                                     SARIMAX Results
==========================================================================================
========
Dep. Variable:                                  HC   No. Observations:
262
Model:             SARIMAX(1, 1, 1)x(2, 0, [], 7)   Log Likelihood               -
1489.902
Date:                            Thu, 11 Aug 2022   AIC
2993.804
Time:                                    19:12:28   BIC
3018.756
Sample:                                03-15-2016   HQIC
3003.834
                                     - 12-01-2016
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
T              3.5025      0.609      5.752      0.000       2.309       4.696
RH             0.4632      0.538      0.861      0.389      -0.591       1.518
ar.L1          0.6368      0.052     12.220      0.000       0.535       0.739
ma.L1         -0.9996      0.291     -3.438      0.001      -1.569      -0.430
ar.S.L7        0.3968      0.058      6.790      0.000       0.282       0.511
ar.S.L14       0.2517      0.063      3.971      0.000       0.127       0.376
sigma2      5212.3591   1465.851      3.556      0.000    2339.344    8085.374
==========================================================================================
=
Ljung-Box (L1) (Q):                   0.30   Jarque-Bera (JB):                 2.2
8
Prob(Q):                              0.59   Prob(JB):                         0.3
2
Heteroskedasticity (H):               1.70   Skew:                             0.1
4
Prob(H) (two-sided):                  0.01   Kurtosis:                         3.3
7
==========================================================================================
=

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

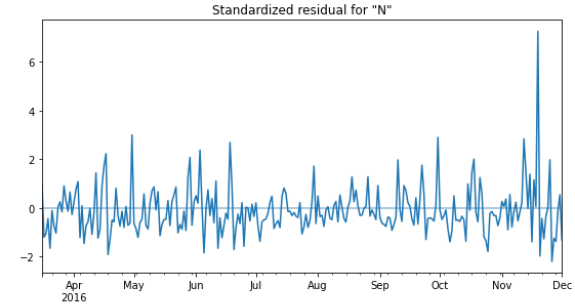

Standardized residual for "H"
Apr
2016
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date

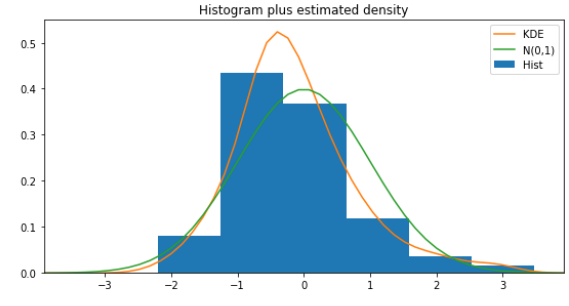

Histogram plus estimated density
KDE
N(0,1)
Hist

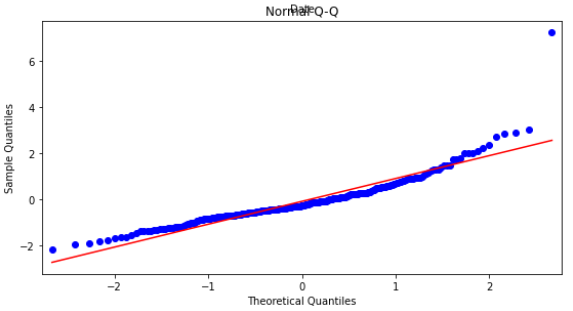

Normal Q-Q
Sample Quantiles
Theoretical Quantiles

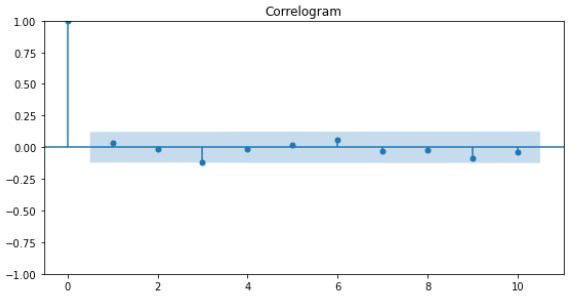

Correlogram

```
                                SARIMAX Results
==========================================================================================
========
Dep. Variable:                                NO2   No. Observations:
262
Model:             SARIMAX(1, 1, 1)x(2, 0, [], 7)   Log Likelihood               -
1552.128
Date:                            Thu, 11 Aug 2022   AIC
3118.256
Time:                                    19:12:31   BIC
3143.207
Sample:                                03-15-2016   HQIC
3128.285
                                     - 12-01-2016
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
T              3.9018      0.823      4.740      0.000       2.288       5.515
RH            -0.1408      0.722     -0.195      0.845      -1.556       1.274
ar.L1          0.6137      0.049     12.652      0.000       0.519       0.709
ma.L1         -0.9771      0.021    -46.043      0.000      -1.019      -0.935
ar.S.L7        0.2300      0.048      4.756      0.000       0.135       0.325
ar.S.L14       0.1949      0.074      2.632      0.008       0.050       0.340
sigma2      8503.6958    417.942     20.347      0.000    7684.545    9322.846
==========================================================================================
=
Ljung-Box (L1) (Q):                   0.32   Jarque-Bera (JB):              1549.6
5
Prob(Q):                              0.57   Prob(JB):                         0.0
0
Heteroskedasticity (H):               1.75   Skew:                             2.1
0
Prob(H) (two-sided):                  0.01   Kurtosis:                        14.1
8
==========================================================================================
=

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

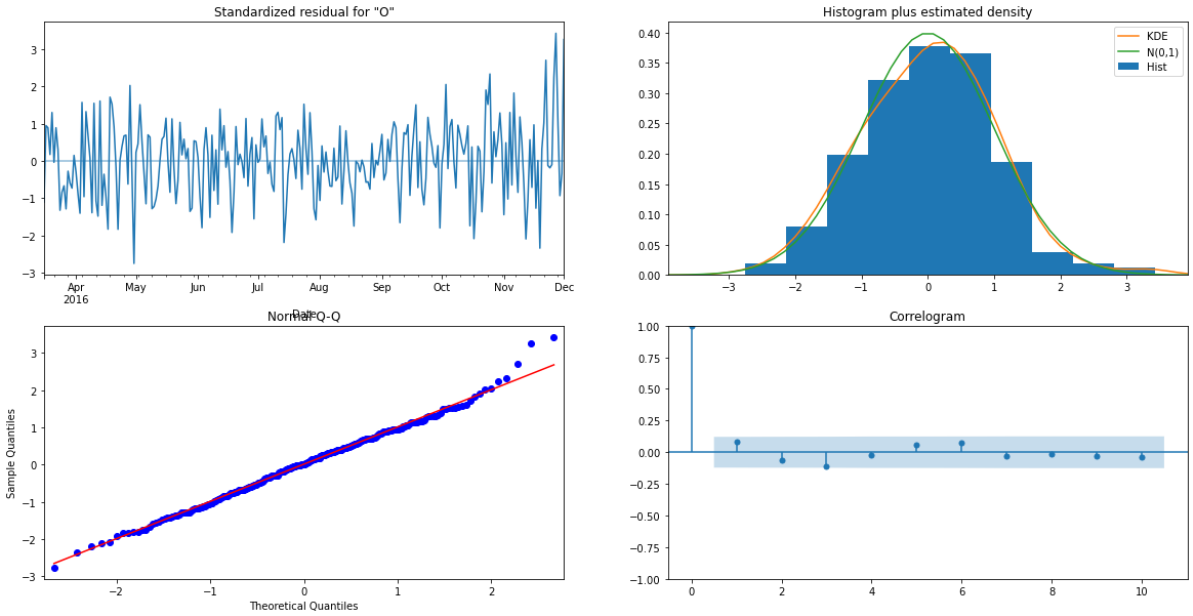

Standardized residual for "N"
Apr
2016
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date


Histogram plus estimated density
KDE
N(0,1)
Hist


Normal Q-Q
Sample Quantiles
Theoretical Quantiles


Correlogram

```
                                SARIMAX Results
==========================================================================================
========
Dep. Variable:                                  O3   No. Observations:
262
Model:             SARIMAX(1, 1, 1)x(2, 0, [], 7)   Log Likelihood               -
1646.827
Date:                            Thu, 11 Aug 2022   AIC
3307.653
Time:                                    19:12:36   BIC
3332.605
Sample:                                03-15-2016   HQIC
3317.683
                                     - 12-01-2016
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
T              2.3490      1.145      2.052      0.040       0.106       4.592
RH             1.4634      1.029      1.423      0.155      -0.553       3.479
ar.L1          0.6719      0.055     12.317      0.000       0.565       0.779
ma.L1         -0.9998      0.566     -1.766      0.077      -2.109       0.110
ar.S.L7        0.2878      0.064      4.528      0.000       0.163       0.412
ar.S.L14       0.1999      0.070      2.853      0.004       0.063       0.337
sigma2      1.742e+04   9288.476      1.876      0.061    -782.397    3.56e+04
==========================================================================================
=
Ljung-Box (L1) (Q):                   1.67   Jarque-Bera (JB):                 1.7
9
Prob(Q):                              0.20   Prob(JB):                         0.4
1
Heteroskedasticity (H):               1.35   Skew:                             0.1
4
Prob(H) (two-sided):                  0.16   Kurtosis:                         3.2
9
==========================================================================================
=

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```



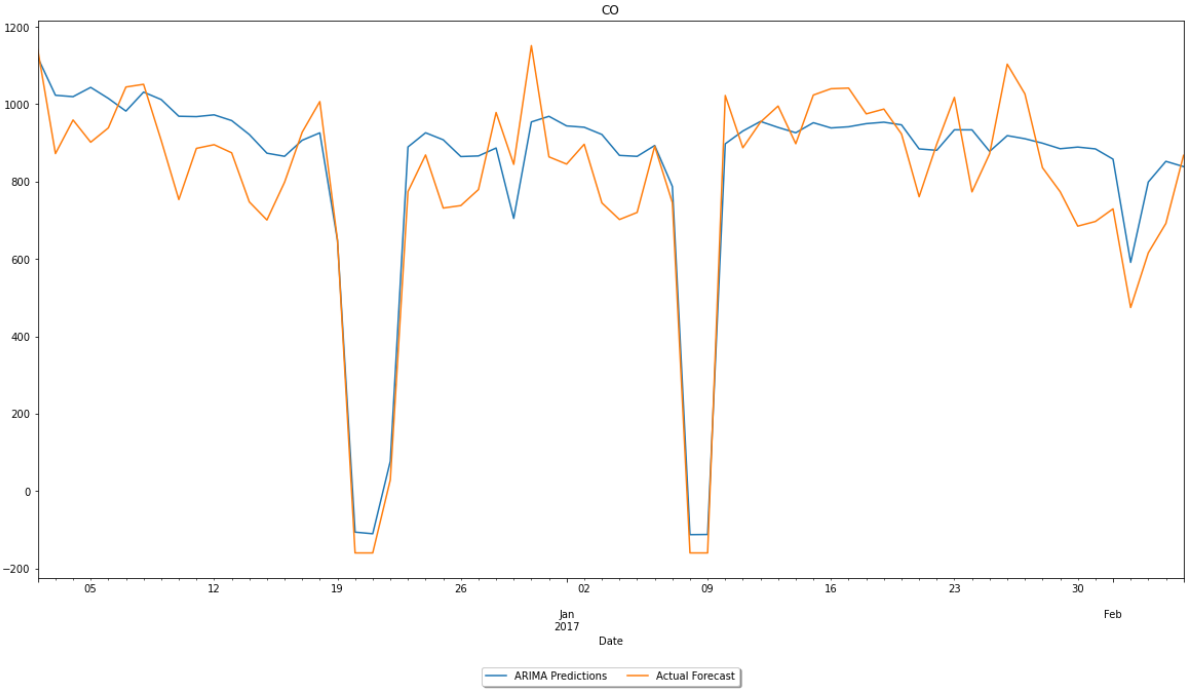
Observations

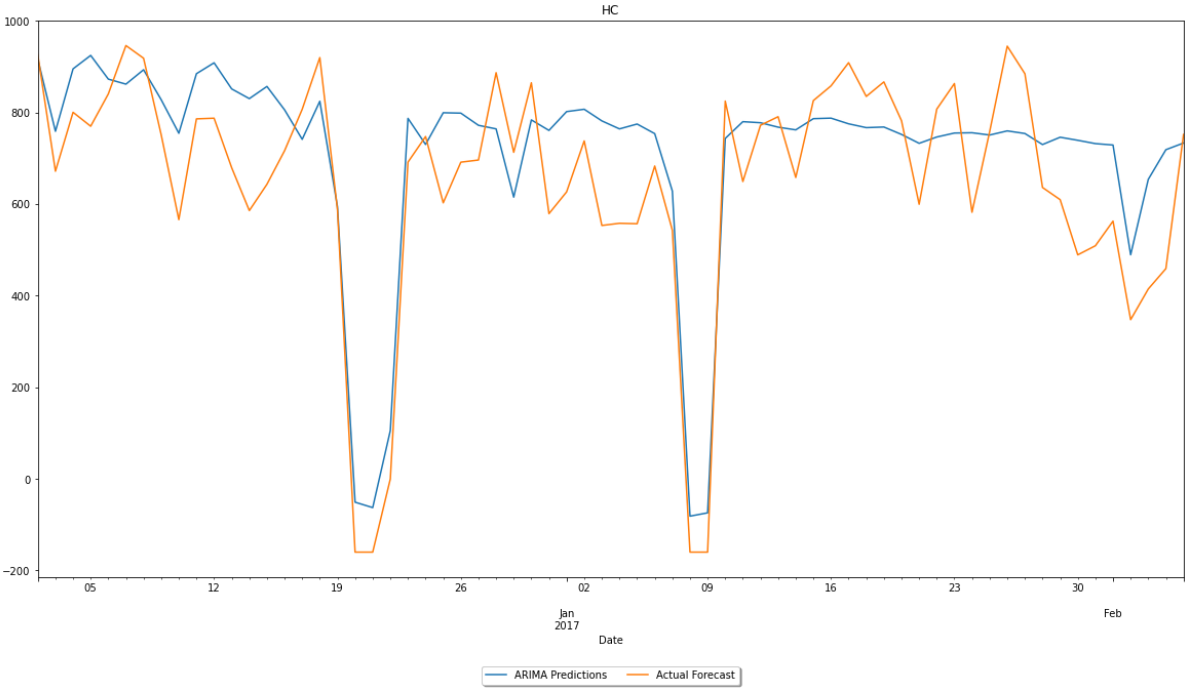
- CO
  - All coefficients are statistically significant
  - The errors caused are not white noise
  - The residuals show no variances
  - Data is not normally distributed against an alternative of another distribution
- HC
  - RH is not statistically significant as its p-value is 0.389 which is > 0.05
  - The errors caused are not white noise
  - The residuals show variances
  - Data is not normally distributed against an alternative of another distribution
- NO2
  - RH is not statistically significant as its p-value is 0.845 which is > 0.05
  - The errors caused are not white noise
  - The residuals show variances
  - Data is normally distributed against an alternative of another distribution
- O3
  - RH is not statistically significant as its p-value is 0.155 which is > 0.05
  - The errors caused are not white noise
  - The residuals show no variances
  - Data is not normally distributed against an alternative of another distribution

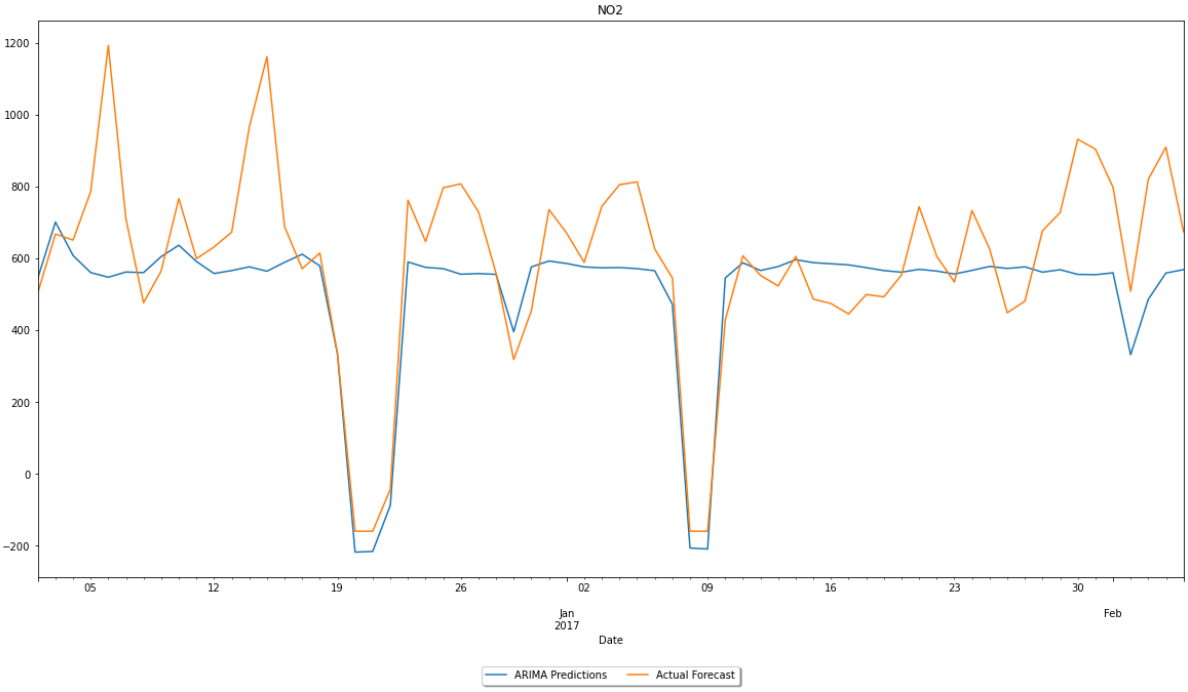
In conclusion, each model will need to be tuned further using their respective orders collected through the autocorrelation and partial autocorrelation plots. Compared to the ARIMA model, SARIMA performed better than it as the residual plot is more close to 0 with

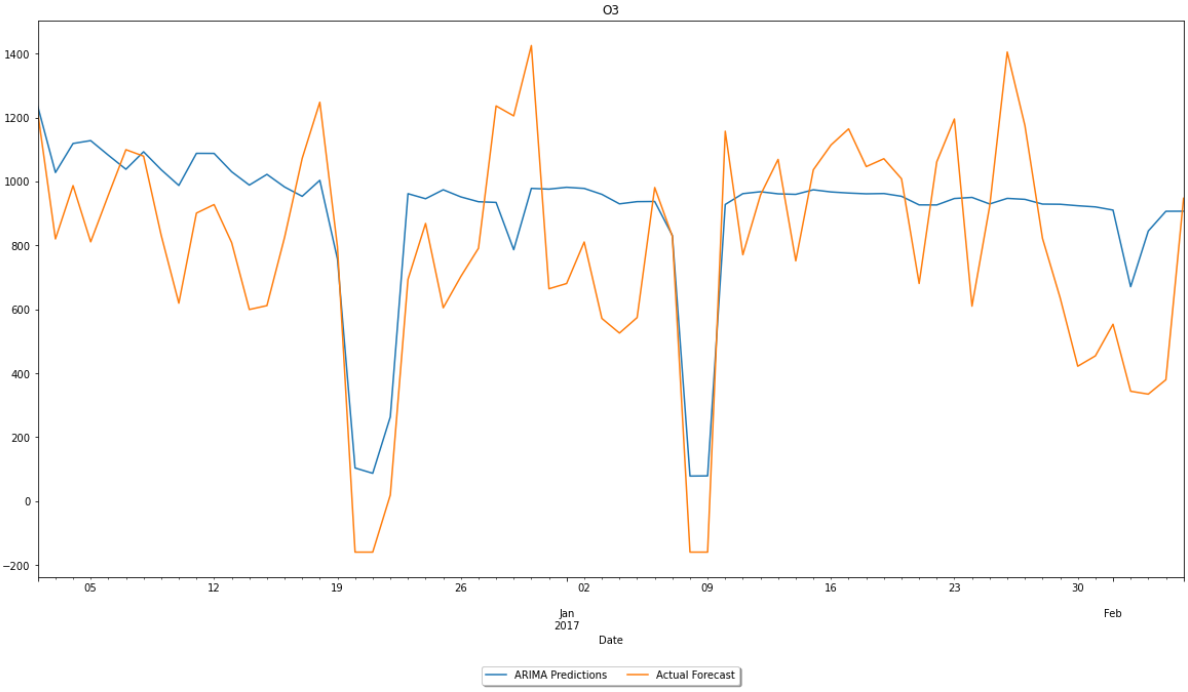
slightly less variation. In the correlogram plot, there is no lag that is outside of the 5% range which is good.

In [ ]:
```
CO_model = SARIMAX(
    endog=CO_train, exog=x_train, order=(1, 1, 2), seasonal_order=(2, 0, 0, 7)
).fit()
HC_model = SARIMAX(
    endog=HC_train, exog=x_train, order=(1, 1, 2), seasonal_order=(2, 0, 0, 7)
).fit()
NO2_model = SARIMAX(
    endog=NO2_train, exog=x_train, order=(1, 1, 2), seasonal_order=(2, 0, 0, 7)
).fit()
O3_model = SARIMAX(
    endog=O3_train, exog=x_train, order=(1, 1, 2), seasonal_order=(2, 0, 0, 7)
).fit()
CO_pred = CO_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test
HC_pred = HC_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test
NO2_pred = NO2_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_te
O3_pred = O3_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test

GAS_pred = [CO_pred, HC_pred, NO2_pred, O3_pred]


for i in range(len(GAS_pred)):
    fig, ax = plt.subplots(1, 1, figsize=(20, 10))
    GAS_pred[i].plot(ax=ax)
    GAS_test[i].plot(ax=ax)
    plt.title(GAS_List[i])
    plt.legend(
        ["ARIMA Predictions", "Actual Forecast"],
        loc="upper center",
        bbox_to_anchor=(0.5, -0.15),
        fancybox=True,
        shadow=True,
        ncol=3,
    )
    plt.show()
```


CO
1200
1000
800
600
400
200
0
−200
05
12
19
26
02
09
16
23
30
Jan
2017
Feb
Date
ARIMA Predictions
Actual Forecast


HC
1000
800
600
400
200
0
−200
05
12
19
26
02
09
16
23
30
Jan
2017
Feb
Date
ARIMA Predictions
Actual Forecast





Comparing SARIMA with ARIMA(Baseline)

When comparing Time Series models, we look at the AIC and BIC. The lower the AIC and BIC, the better the models performance. AIC as mentioned helps determine the strength of the linear regression model and would emphasis more on model performance (select more complex models) and BIC as mentioned helps to select the true model of the dataset which follows more strictly to the original training dataset this causes the it to select a more simple model which focuses more on getting the data to work. This means AIC and BIC needs to be balanced.

Based on the SARIMA and ARIMA summary, the SARIMA models have a decrease in AIC and BIC score by around 50 - 300 points. This means that with the inclusion of the seasonality, the model performs better.

Furthermore, based on the side by side comparision of the model prediction on the validation set it seems SARIMA follows the trend of the actual forecast more closely compared to ARIMA.

Diagnosing Baseline SARIMA Model

As Time Series is similar to a regression problem, we will be using the following scoring metrics to help us in the time series

1. Mean Absolute Percentage Error
    - A measure of prediction accuracy of a forecasting method in statistics
2. Root Mean Squared Error
    - RMSE is the square root of the mean of the square of all of the error. We use RMSE more often as MSE values can be too big to compare easily. MSE shows how close the data points are closely fitted to a line.

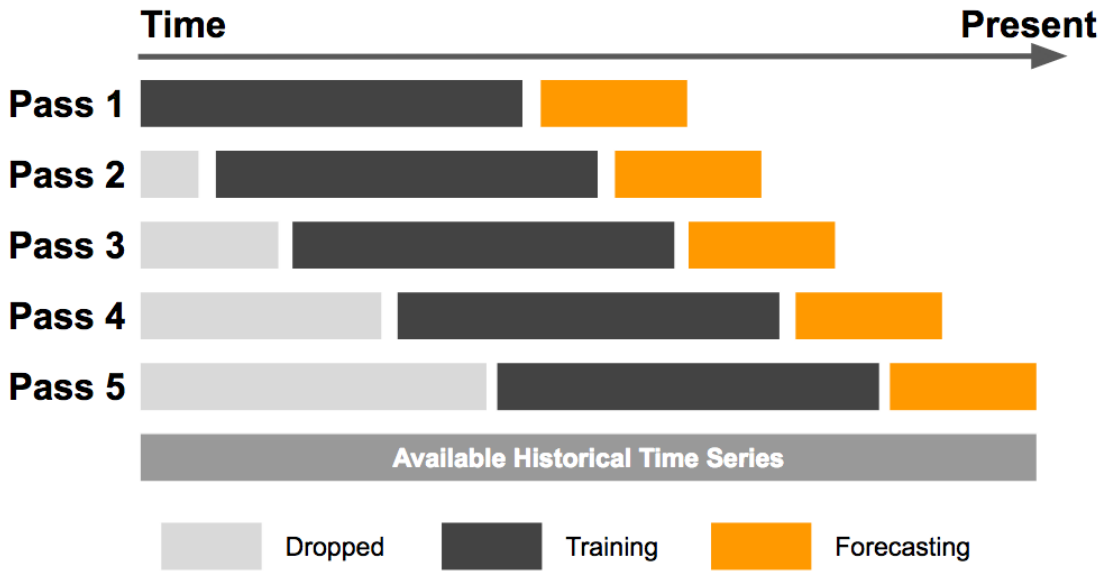
In [ ]:
```
CO_train_pred = CO_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
HC_train_pred = HC_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
NO2_train_pred = NO2_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
O3_train_pred = O3_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)

GAS_train_pred = [CO_train_pred, HC_train_pred, NO2_train_pred, O3_train_pred]

for i in range(len(GAS_List)):
    print(
        f"{GAS_List[i]} Train MAPE:\t\t{mean_absolute_percentage_error(y_train[GAS_
    )
    print(
        f"{GAS_List[i]} Train RMSE:\t\t{mean_squared_error(y_train[GAS_List[i]],GAS
    )
    print(
        f"{GAS_List[i]} Validation MAPE:\t{mean_absolute_percentage_error(y_test[GA
    )
    print(
        f"{GAS_List[i]} Validation RMSE:\t{mean_squared_error(y_test[GAS_List[i]],C
    )
    print()
```

```
CO Train MAPE:          6.928%
CO Train RMSE:          85.607
CO Validation MAPE:     15.369%
CO Validation RMSE:     109.315

HC Train MAPE:          9.431%
HC Train RMSE:          83.326
HC Validation MAPE:     981.828%
HC Validation RMSE:     131.328

NO2 Train MAPE:         11.050%
NO2 Train RMSE:         107.408
NO2 Validation MAPE:    21.158%
NO2 Validation RMSE:    184.255

O3 Train MAPE:          38.968%
O3 Train RMSE:          140.103
O3 Validation MAPE:     60.418%
O3 Validation RMSE:     267.255
```

Observations

- CO has the lowest difference and the forecasting ability is quite good as the RMSE of the validation and RMSE of the training set is relatively close.
- HC has a good training forecast but once it has reached the validation set, it did not perform as well as intended. Parameters can be adjusted and orders can be used to improve on the models performance.
- NO2 has a good training and validation RMSE and is relatively close which means that the order is quite good for prediction of NO2 gas. Graphically, the data points do not spike up when need.
- O3 has a good training forecast but not a strong validation forecast. The dips and spikes are not very clear which needs to be adjusted.

Cross Validation - Expanding Window



To validate and make sure the model is evaluated general potential. One way to achieve this is to train the data using the passage of time in different windows.

In [ ]:
```
def model_cv(
    model, endog, exog, order=(1, 1, 1), splits=3, seasonal_order=None, max_iter=10
):
    valid_rmse = []
    valid_mape = []
    train_rmse = []
    train_mape = []
    model_aic = []
    model_bic = []
    time_series = TimeSeriesSplit(n_splits=splits)
    for train_index, test_index in time_series.split(exog):
        X_train, X_test = exog.iloc[train_index], exog.iloc[test_index]
        y_train, y_test = endog.iloc[train_index], endog.iloc[test_index]
        if seasonal_order is None:
            model_fit = model(endog=y_train, exog=X_train, order=order).fit(
                maxiter=max_iter, disp=False
            )
        else:
            model_fit = model(
                endog=y_train, exog=X_train, seasonal_order=seasonal_order
            ).fit(maxiter=max_iter, disp=False)
        y_pred = model_fit.predict(
            start=X_test.index[0], end=X_test.index[-1], exog=X_test
        )
        valid_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
        valid_mape.append(mean_absolute_percentage_error(y_test, y_pred))
        train_pred = model_fit.predict(
            start=X_train.index[0], end=X_train.index[-1], exog=X_train
        )
        train_rmse.append(mean_squared_error(y_train, train_pred, squared=False))
        train_mape.append(mean_absolute_percentage_error(y_train, train_pred))
        model_aic.append(model_fit.aic)
        model_bic.append(model_fit.bic)
    return (
```

```
        pd.Series(
            {
                "train_rmse": np.mean(train_rmse),
                "train_rmse_std": np.std(train_rmse),
                "valid_rmse": np.mean(valid_rmse),
                "valid_rmse_std": np.std(valid_rmse),
                "train_mape": np.mean(train_mape),
                "valid_mape": np.mean(valid_mape),
                "AIC": np.mean(model_aic),
                "BIC": np.mean(model_bic),
            },
            name=order,
        ),
        pd.DataFrame(
            {
                "train_rmse": train_rmse,
                "train_rmse_std": train_rmse,
                "valid_rmse": valid_rmse,
                "valid_rmse_std": valid_rmse,
                "train_mape": train_mape,
                "valid_mape": valid_mape,
                "AIC": model_aic,
                "BIC": model_bic,
            }
        ),
    )
```

In [ ]:
```
avg_scores = []
cv_table = []
for i in GAS_List:
    score, cv = model_cv(
        SARIMAX, exog=x, endog=y[i], splits=5, seasonal_order=(1, 1, 2, 7)
    )
    avg_scores.append(score)
    cv_table.append(cv)
```

In [ ]:
```
for i in range(len(cv_table)):
    print(GAS_List[i])
    display(
        cv_table[i]
        .style.apply(
            lambda x: [
                "background-color: green; color: white" if v else ""
                for v in x == x.min()
            ]
        )
        .apply(
            lambda x: [
                "background-color: red; color: white" if v else "" for v in x == x.
            ]
        )
    )
```

CO

	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AIC	
0	357.446249	357.446249	131.904478	131.904478	0.173741	0.162438	600.783578	
1	241.114709	241.114709	95.210348	95.210348	0.123807	0.105678	1196.549122	1
2	200.777546	200.777546	104.940262	104.940262	0.095747	0.101699	1758.902594	1
3	172.891407	172.891407	114.542178	114.542178	0.088486	0.096843	2351.712728	2
4	155.016154	155.016154	107.527807	107.527807	0.085085	0.162321	2996.071669	3

HC

	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AIC	
0	289.558361	289.558361	107.721196	107.721196	0.190471	0.144765	584.730847	
1	198.726073	198.726073	108.945717	108.945717	0.139233	0.129224	1186.405427	1
2	168.380172	168.380172	132.108710	132.108710	0.113728	0.174578	1764.535567	1
3	144.252210	144.252210	149.995831	149.995831	0.106327	0.144264	2370.064258	2
4	130.621383	130.621383	134.445258	134.445258	0.101664	9.756214	3025.633116	3

NO2

	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AIC	
0	242.228626	242.228626	130.614890	130.614890	0.173871	0.157243	610.495193	
1	192.952993	192.952993	105.891130	105.891130	0.142404	0.148053	1250.576662	1
2	161.828087	161.828087	128.328467	128.328467	0.115961	0.231518	1840.769978	1
3	153.345921	153.345921	182.235857	182.235857	0.124040	0.211636	2470.771801	2
4	156.092875	156.092875	174.958813	174.958813	0.128878	0.233500	3210.808485	3

O3

	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AIC	
0	384.880150	384.880150	222.632837	222.632837	0.286188	0.309980	663.157199	
1	257.954444	257.954444	177.178685	177.178685	0.218261	0.238075	1324.891025	1
2	223.504782	223.504782	208.895546	208.895546	0.188406	1.351125	1979.042694	2
3	193.703227	193.703227	283.748969	283.748969	0.270236	0.233471	2641.800826	2
4	179.775801	179.775801	268.166430	268.166430	0.243837	0.558143	3353.105245	3

Hyperparameter Tuning

To enhance the model, we will need to find the best order. Although we can find the value of p and q using the autocorrelation and partial autocorrelation plots. We can use loops to

look through the best values. We are making us of a function that works quite similar to the GridSearchCV from sklearn.

In [ ]:
```
warnings.filterwarnings("ignore")
scores_df = pd.DataFrame()
p = np.arange(0, 5)
q = np.arange(0, 5)
for i in list(product(p, q)):
    gas_df = pd.DataFrame()
    i = (i[0], 1, i[1], 7)
    print("Computing (p, 1, q, 7):", i)
    try:
        for j in range(len(GAS_List)):
            scores_df[i, GAS_List[j]], _ = model_cv(
                SARIMAX, seasonal_order=i, splits=5, exog=x, endog=y[GAS_List[j]]
            )
    except Exception as e:
        print("Error computing (p, 1, q, 7):", i)
        print(e)
        pass
```

```
Computing (p, 1, q, 7): (0, 1, 0, 7)
Computing (p, 1, q, 7): (0, 1, 1, 7)
Computing (p, 1, q, 7): (0, 1, 2, 7)
Computing (p, 1, q, 7): (0, 1, 3, 7)
Computing (p, 1, q, 7): (0, 1, 4, 7)
Computing (p, 1, q, 7): (1, 1, 0, 7)
Computing (p, 1, q, 7): (1, 1, 1, 7)
Computing (p, 1, q, 7): (1, 1, 2, 7)
Computing (p, 1, q, 7): (1, 1, 3, 7)
Computing (p, 1, q, 7): (1, 1, 4, 7)
Computing (p, 1, q, 7): (2, 1, 0, 7)
Computing (p, 1, q, 7): (2, 1, 1, 7)
Computing (p, 1, q, 7): (2, 1, 2, 7)
Computing (p, 1, q, 7): (2, 1, 3, 7)
Computing (p, 1, q, 7): (2, 1, 4, 7)
Computing (p, 1, q, 7): (3, 1, 0, 7)
Computing (p, 1, q, 7): (3, 1, 1, 7)
Computing (p, 1, q, 7): (3, 1, 2, 7)
Computing (p, 1, q, 7): (3, 1, 3, 7)
Computing (p, 1, q, 7): (3, 1, 4, 7)
Computing (p, 1, q, 7): (4, 1, 0, 7)
Computing (p, 1, q, 7): (4, 1, 1, 7)
Computing (p, 1, q, 7): (4, 1, 2, 7)
Computing (p, 1, q, 7): (4, 1, 3, 7)
Computing (p, 1, q, 7): (4, 1, 4, 7)
```

In [ ]:
```
cv_df = scores_df.T.reset_index()
CO_cv = []
HC_cv = []
NO2_cv = []
O3_cv = []
for i in range(len(cv_df)):
    gas = str(cv_df.loc[:, "index"][i]).split("'")[1]
    if gas == "CO":
        CO_cv.append(cv_df.iloc[i])
```

```
    elif gas == "HC":
        HC_cv.append(cv_df.iloc[i])
    elif gas == "NO2":
        NO2_cv.append(cv_df.iloc[i])
    elif gas == "O3":
        O3_cv.append(cv_df.iloc[i])

CO_cv = pd.DataFrame(CO_cv).set_index("index")
HC_cv = pd.DataFrame(HC_cv).set_index("index")
NO2_cv = pd.DataFrame(NO2_cv).set_index("index")
O3_cv = pd.DataFrame(O3_cv).set_index("index")
```

In [ ]:

```
CO_cv.style.apply(
    lambda x: [
        "background-color: green; color: white" if v else "" for v in x == x.min()
    ]
).apply(
    lambda x: ["background-color: red; color: white" if v else "" for v in x == x.m
)
```

Out[ ]:

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((0, 1, 0, 7), 'CO')	228.131572	63.315999	128.421792	38.812400	0.132707	0.147949	1857.44492
((0, 1, 1, 7), 'CO')	223.671195	69.073010	109.522914	10.523281	0.113708	0.124360	1778.12962
((0, 1, 2, 7), 'CO')	225.428048	72.181050	110.864402	11.206556	0.113412	0.128177	1779.21010
((0, 1, 3, 7), 'CO')	224.907143	72.157094	109.443563	15.430673	0.113135	0.127078	1779.97828
((0, 1, 4, 7), 'CO')	224.496172	72.036565	107.983794	22.782132	0.113777	0.123609	1780.86254
((1, 1, 0, 7), 'CO')	231.160574	68.655002	123.811704	27.363136	0.123708	0.152981	1817.42114
((1, 1, 1, 7), 'CO')	225.105872	71.584493	111.002662	11.712933	0.113366	0.127823	1779.28920
((1, 1, 2, 7), 'CO')	225.449213	72.111657	110.825014	12.226862	0.113373	0.125796	1780.80393
((1, 1, 3, 7), 'CO')	224.636243	71.819237	118.665734	19.108347	0.113640	0.136296	1781.19726
((1, 1, 4, 7), 'CO')	223.230865	71.996755	113.465984	28.141337	0.113373	0.127991	1780.58050
((2, 1, 0, 7), 'CO')	228.962574	70.566803	117.282038	26.481222	0.118642	0.151812	1802.05555

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((2, 1, 1, 7), 'CO')	224.780722	72.098853	109.191438	15.071043	0.113067	0.126536	1779.99380
((2, 1, 2, 7), 'CO')	224.914515	72.371431	112.017902	11.704601	0.113713	0.128589	1781.21185
((2, 1, 3, 7), 'CO')	223.151910	71.030932	121.883034	18.578023	0.111996	0.142105	1779.02122
((2, 1, 4, 7), 'CO')	221.910468	72.464594	115.711611	21.917703	0.110207	0.121326	1776.32250
((3, 1, 0, 7), 'CO')	227.906999	70.936460	112.515962	27.009884	0.116036	0.136435	1786.94442
((3, 1, 1, 7), 'CO')	225.171323	72.421817	109.212215	17.884477	0.113389	0.124179	1781.29359
((3, 1, 2, 7), 'CO')	223.483579	70.314740	104.487037	16.154343	0.113270	0.117586	1781.32310
((3, 1, 3, 7), 'CO')	221.639246	71.873940	103.973338	13.892593	0.110058	0.107357	1775.88503
((3, 1, 4, 7), 'CO')	220.606223	70.421718	115.806231	31.390047	0.111397	0.120749	1781.94052
((4, 1, 0, 7), 'CO')	227.898600	70.617940	112.832028	26.522744	0.116146	0.133622	1788.04686
((4, 1, 1, 7), 'CO')	225.726937	71.960333	105.128289	21.917314	0.114462	0.118740	1782.93485

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((4, 1, 2, 7), 'CO')	223.739077	71.370409	108.552221	8.846064	0.112847	0.123601	1782.69236
((4, 1, 3, 7), 'CO')	221.196933	70.949024	108.350520	7.054516	0.110991	0.121021	1779.43715
((4, 1, 4, 7), 'CO')	221.669877	71.624652	103.685466	13.898846	0.110258	0.112606	1781.79051

In [ ]:

```
HC_cv.style.apply(
    lambda x: [
        "background-color: green; color: white" if v else "" for v in x == x.min()
    ]
).apply(
    lambda x: ["background-color: red; color: white" if v else "" for v in x == x.m
)
```

Out[ ]:

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((0, 1, 0, 7), 'HC')	187.888210	46.713044	138.813859	49.711782	0.153325	2.709477	1852.09543
((0, 1, 1, 7), 'HC')	185.324718	54.741757	123.698528	18.395170	0.130318	2.235863	1783.79995
((0, 1, 2, 7), 'HC')	186.056248	56.659403	125.877306	15.959083	0.130901	2.060544	1784.98022
((0, 1, 3, 7), 'HC')	185.183160	54.979182	124.728737	17.981489	0.129856	2.089617	1785.93434
((0, 1, 4, 7), 'HC')	185.287201	54.940296	126.090411	18.556894	0.129722	2.062415	1787.85056
((1, 1, 0, 7), 'HC')	190.455731	52.711802	139.370612	40.324136	0.143275	3.918523	1815.08576
((1, 1, 1, 7), 'HC')	186.197279	56.739316	126.284514	15.727005	0.130708	2.069653	1784.56184
((1, 1, 2, 7), 'HC')	186.307640	56.579045	126.643342	16.169925	0.130285	2.069809	1786.27384
((1, 1, 3, 7), 'HC')	185.340985	54.953232	124.884819	18.084280	0.129739	2.059509	1787.55995
((1, 1, 4, 7), 'HC')	185.220512	55.138635	125.561637	18.848237	0.129670	2.176157	1789.23629
((2, 1, 0, 7), 'HC')	189.636744	54.767212	134.290362	34.995309	0.138566	4.319168	1803.57433

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((2, 1, 1, 7), 'HC')	185.510988	55.494084	125.284543	17.352300	0.129907	2.086320	1786.19003
((2, 1, 2, 7), 'HC')	185.706144	55.524760	125.248931	17.247742	0.129896	2.028909	1787.81160
((2, 1, 3, 7), 'HC')	184.846129	55.060836	128.948892	23.857246	0.128604	1.525290	1786.16281
((2, 1, 4, 7), 'HC')	185.340003	55.510409	133.100166	26.413087	0.129216	1.599057	1789.25787
((3, 1, 0, 7), 'HC')	189.123614	55.303619	131.605433	32.769075	0.135731	3.125555	1797.64455
((3, 1, 1, 7), 'HC')	186.281531	56.799573	126.372665	16.003861	0.130759	2.083197	1788.45516
((3, 1, 2, 7), 'HC')	186.069699	56.407474	125.910128	17.210104	0.129969	2.046916	1789.63428
((3, 1, 3, 7), 'HC')	184.001937	53.249239	132.603292	28.699781	0.128587	2.412150	1787.68583
((3, 1, 4, 7), 'HC')	184.356274	55.206366	129.499646	24.328935	0.128007	1.687056	1788.73659
((4, 1, 0, 7), 'HC')	188.941519	55.122143	130.381511	31.840348	0.133918	2.660089	1798.12808
((4, 1, 1, 7), 'HC')	184.871144	54.132914	122.806123	18.285330	0.129472	1.925421	1789.46194

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((4, 1, 2, 7), 'HC')	184.423009	54.078515	129.228202	24.362280	0.128899	2.282933	1789.09691
((4, 1, 3, 7), 'HC')	184.767112	55.024619	130.273414	25.539292	0.128888	2.166164	1788.77881
((4, 1, 4, 7), 'HC')	184.563713	54.016950	125.686472	20.494043	0.128869	1.634376	1793.29695

In [ ]:

```
NO2_cv.style.apply(
    lambda x: [
        "background-color: green; color: white" if v else "" for v in x == x.min()
    ]
).apply(
    lambda x: ["background-color: red; color: white" if v else "" for v in x == x.m
)
```

Out[ ]:

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	A
((0, 1, 0, 7), 'NO2')	205.612829	42.955073	154.028498	73.042990	0.165474	0.236603	1955.5999
((0, 1, 1, 7), 'NO2')	185.120992	39.187714	136.994253	32.810299	0.140655	0.193583	1877.6184
((0, 1, 2, 7), 'NO2')	182.261484	34.906788	142.641262	28.385614	0.137306	0.199030	1876.6715
((0, 1, 3, 7), 'NO2')	181.429633	33.438616	140.233218	30.012088	0.136167	0.192675	1877.2153
((0, 1, 4, 7), 'NO2')	181.615254	33.419512	140.101536	31.699111	0.136222	0.191649	1878.9012
((1, 1, 0, 7), 'NO2')	188.623185	33.378022	141.819786	50.812222	0.147337	0.203625	1908.1691
((1, 1, 1, 7), 'NO2')	181.830952	34.222318	140.180030	29.416779	0.137239	0.196094	1876.3205
((1, 1, 2, 7), 'NO2')	181.289700	33.597840	144.405832	29.314329	0.137031	0.196390	1876.6844
((1, 1, 3, 7), 'NO2')	181.991359	34.656093	139.033517	34.081733	0.137123	0.190073	1878.1022
((1, 1, 4, 7), 'NO2')	181.645464	33.784329	141.556942	30.253418	0.136392	0.193653	1880.5254
((2, 1, 0, 7), 'NO2')	185.576447	32.062612	135.371655	44.470979	0.142648	0.189829	1893.6307
((2, 1, 1, 7), 'NO2')	182.691318	35.861532	139.227217	30.467083	0.137497	0.193126	1877.9591
((2, 1, 2, 7), 'NO2')	181.337611	33.472576	143.886281	30.389878	0.136868	0.196378	1878.0122
((2, 1, 3, 7), 'NO2')	182.206026	34.090125	134.539893	34.686731	0.136997	0.185557	1879.3837

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	A
((2, 1, 4, 7), 'NO2')	181.430601	33.485862	139.333331	32.665622	0.136507	0.188924	1882.0023
((3, 1, 0, 7), 'NO2')	184.027092	32.689725	134.706566	39.963405	0.139553	0.182798	1886.1892
((3, 1, 1, 7), 'NO2')	181.299734	33.222922	142.603462	29.313602	0.136504	0.195919	1879.1923
((3, 1, 2, 7), 'NO2')	181.332374	33.511148	142.422472	29.137486	0.136838	0.195597	1881.0447
((3, 1, 3, 7), 'NO2')	181.775335	33.649490	142.982974	31.388092	0.137447	0.198200	1882.5072
((3, 1, 4, 7), 'NO2')	182.482631	35.005413	138.845505	36.965086	0.136008	0.185298	1881.6598
((4, 1, 0, 7), 'NO2')	183.472962	32.701517	137.640105	40.296029	0.138995	0.187673	1886.2522
((4, 1, 1, 7), 'NO2')	181.277959	33.207240	142.393518	25.719016	0.136818	0.196457	1881.7530
((4, 1, 2, 7), 'NO2')	181.519381	34.014348	141.837902	29.261124	0.136382	0.194368	1882.4752
((4, 1, 3, 7), 'NO2')	181.954211	33.784323	148.569741	27.653710	0.135951	0.204590	1881.8618
((4, 1, 4, 7), 'NO2')	182.199968	34.246421	140.950970	32.326849	0.135327	0.189123	1882.5359

In [ ]:
```
O3_cv.style.apply(
    lambda x: [
        "background-color: green; color: white" if v else "" for v in x == x.min()
    ]
).apply(
    lambda x: ["background-color: red; color: white" if v else "" for v in x == x.m
)
```

Out[ ]:

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((0, 1, 0, 7), 'O3')	262.473954	59.098881	252.466755	77.922163	0.428828	0.816818	2066.39450
((0, 1, 1, 7), 'O3')	246.447939	70.922455	226.599269	40.820186	0.239829	0.712715	1990.23131
((0, 1, 2, 7), 'O3')	248.205218	74.417094	231.451621	39.934104	0.237166	0.716739	1991.39769
((0, 1, 3, 7), 'O3')	246.911583	71.720709	229.323506	40.210739	0.240089	0.663327	1992.75426
((0, 1, 4, 7), 'O3')	246.542373	71.182489	230.423171	40.102290	0.239072	0.668178	1994.59493
((1, 1, 0, 7), 'O3')	258.837034	64.922677	250.817675	59.579531	0.392319	0.877001	2024.27337
((1, 1, 1, 7), 'O3')	247.693668	73.394681	230.869158	40.034501	0.235595	0.716089	1991.17156
((1, 1, 2, 7), 'O3')	247.963681	73.536326	232.124494	39.020059	0.241386	0.538159	1992.39939
((1, 1, 3, 7), 'O3')	246.619219	71.258891	231.417522	41.620296	0.232134	0.521243	1993.72337
((1, 1, 4, 7), 'O3')	246.049041	72.679535	222.657105	48.893653	0.225439	0.692109	1995.54485
((2, 1, 0, 7), 'O3')	257.064991	69.899385	243.105622	53.182756	0.304542	0.639548	2012.28447

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((2, 1, 1, 7), 'O3')	248.149096	74.019116	231.034459	39.677734	0.240888	0.669583	1992.90427
((2, 1, 2, 7), 'O3')	248.405378	74.707722	232.135597	40.963044	0.234942	0.516172	1994.25973
((2, 1, 3, 7), 'O3')	246.262801	71.503521	237.847864	34.517209	0.220500	0.544798	1992.57229
((2, 1, 4, 7), 'O3')	246.250953	72.318951	229.641347	40.743348	0.225985	0.500451	1995.52713
((3, 1, 0, 7), 'O3')	255.168222	71.063623	238.900574	53.338804	0.281317	0.547307	2005.20732
((3, 1, 1, 7), 'O3')	249.548091	75.072890	218.129719	54.580898	0.242425	0.762376	1995.93935
((3, 1, 2, 7), 'O3')	247.420064	73.325646	232.969138	40.358083	0.226735	0.529388	1995.54457
((3, 1, 3, 7), 'O3')	247.600504	73.722550	233.761843	41.249593	0.267794	0.530910	1996.21520
((3, 1, 4, 7), 'O3')	245.881184	71.215971	231.756476	40.105820	0.216103	0.491553	1997.21355
((4, 1, 0, 7), 'O3')	255.172942	71.508551	238.847559	51.879745	0.236840	0.425928	2005.79528
((4, 1, 1, 7), 'O3')	248.652822	72.961042	218.039516	55.308579	0.240964	0.790645	1997.87139

index	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AI
((4, 1, 2, 7), 'O3')	247.675924	73.821840	231.192691	40.433339	0.223905	0.507086	1997.14463
((4, 1, 3, 7), 'O3')	243.155875	71.382697	227.039828	41.031395	0.234690	0.430303	1994.32934
((4, 1, 4, 7), 'O3')	244.681029	71.130034	226.715002	47.652697	0.229659	0.301560	1997.47460

Observations

Overall, we can use the following order to go our predictions

- CO
  - (3, 1, 3, 7)
  - It has the lowest MPE and the valid RMSE is close to the lowest valid RMSE
- HC
  - (0, 1, 1, 7)
  - The model has the best AIC and BIC which means the models is not complex but performs well
  - The valid RMSE is also very close to the lowest valid RMSE
- NO2
  - (3, 1, 0, 7)
  - The model has the lowest valid MAPE and the valid RMSE is close to the lowest valid RMSE
- O3
  - (4, 1, 1, 7)
  - The model has the lowest valid RMSE and the AIC and BIC is close to the lowest AIC and BIC

In [ ]:
```
SARIMA_GAS_train_pred = [CO_train_pred, HC_train_pred, NO2_train_pred, O3_train_pre
```

In [ ]:
```
CO_model = SARIMAX(endog=y_train["CO"], exog=x_train, seasonal_order=(3, 1, 3, 7)).
HC_model = SARIMAX(endog=y_train["HC"], exog=x_train, seasonal_order=(0, 1, 1, 7)).
NO2_model = SARIMAX(
    endog=y_train["NO2"], exog=x_train, seasonal_order=(3, 1, 0, 7)
).fit()
O3_model = SARIMAX(endog=y_train["O3"], exog=x_train, seasonal_order=(4, 1, 1, 7)).
CO_train_pred = CO_model.predict(
```

```
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
HC_train_pred = HC_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
NO2_train_pred = NO2_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
O3_train_pred = O3_model.predict(
    start=x_train.index[0], end=x_train.index[-1], exog=x_train
)
CO_pred = CO_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test
HC_pred = HC_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test
NO2_pred = NO2_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_te
O3_pred = O3_model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test

SARIMA_GAS_train_pred = [CO_train_pred, HC_train_pred, NO2_train_pred, O3_train_pre
SARIMA_GAS_pred = [CO_pred, HC_pred, NO2_pred, O3_pred]
```

In [ ]:
```
for i in range(len(GAS_List)):
    print(
        f"{GAS_List[i]} Train MAPE:\t\t{mean_absolute_percentage_error(y_train[GAS_
    )
    print(
        f"{GAS_List[i]} Train RMSE:\t\t{mean_squared_error(y_train[GAS_List[i]],SAR
    )
    print(
        f"{GAS_List[i]} Validation MAPE:\t{mean_absolute_percentage_error(y_test[GA
    )
    print(
        f"{GAS_List[i]} Validation RMSE:\t{mean_squared_error(y_test[GAS_List[i]],S
    )
    print()
```

```
CO Train MAPE:          8.637%
CO Train RMSE:          156.107
CO Validation MAPE:     15.741%
CO Validation RMSE:     108.774

HC Train MAPE:          10.301%
HC Train RMSE:          132.532
HC Validation MAPE:     925.182%
HC Validation RMSE:     135.916

NO2 Train MAPE:         12.811%
NO2 Train RMSE:         158.264
NO2 Validation MAPE:    24.552%
NO2 Validation RMSE:    198.951

O3 Train MAPE:          25.746%
O3 Train RMSE:          181.409
O3 Validation MAPE:     48.880%
O3 Validation RMSE:     251.127
```

In [ ]:
```
# CO_model = SARIMAX(endog=y["CO"], exog=x, seasonal_order=(3, 1, 3, 7)).fit()
# HC_model = SARIMAX(endog=y["HC"], exog=x, seasonal_order=(0, 1, 1, 7)).fit()
```

```
# NO2_model = SARIMAX(endog=y["NO2"], exog=x, seasonal_order=(3, 1, 0, 7)).fit()
# O3_model = SARIMAX(endog=y["O3"], exog=x, seasonal_order=(4, 1, 1, 7)).fit()
# CO_pred = CO_model.predict(start=X_test.index[0], end=X_test.index[-1], exog=X_te
# HC_pred = HC_model.predict(start=X_test.index[0], end=X_test.index[-1], exog=X_te
# NO2_pred = NO2_model.predict(start=X_test.index[0], end=X_test.index[-1], exog=X_
# O3_pred = O3_model.predict(start=X_test.index[0], end=X_test.index[-1], exog=X_te

# GAS_pred = [CO_pred, HC_pred, NO2_pred, O3_pred]
```

In [ ]:
```
# pd.DataFrame(np.array(GAS_pred).flatten()).reset_index().rename({'index':"id",0:
#     "./kaggle/sarima2.csv",index=False
# )
# print("Exporting to csv")
```

VARMAX

As mentioned previously, VARMAX is able to handle multiple data and time series together. Furthermore, we have shown in the Causality and Cointegration Test at least 2 time series influence each other. Therefore we will use a multivariate regressor like VARMAX

In [ ]:
```
model = VARMAX(endog=y_train, exog=x_train, order=(1, 1)).fit(maxiter=1000, disp=Fa
model.summary()
```

Out[ ]:

Statespace Model Results

Dep. Variable:	['CO', 'HC', 'NO2', 'O3']	**No. Observations:**	262
Model:	VARMAX(1,1)	**Log Likelihood**	-5721.890
	+ intercept	**AIC**	11551.779
Date:	Thu, 11 Aug 2022	**BIC**	11744.470
Time:	19:42:26	**HQIC**	11629.226
Sample:	03-15-2016		
	- 12-01-2016		
Covariance Type:	opg		

Ljung-Box (L1) (Q):	5.22, 11.87, 10.35, 21.32	**Jarque-Bera (JB):**	1.80, 0.43, 363.43, 66.88
Prob(Q):	0.02, 0.00, 0.00, 0.00	**Prob(JB):**	0.41, 0.81, 0.00, 0.00
Heteroskedasticity (H):	1.44, 0.96, 1.45, 2.04	**Skew:**	-0.05, 0.09, 0.85, 0.73
Prob(H) (two-sided):	0.09, 0.83, 0.09, 0.00	**Kurtosis:**	3.39, 3.09, 8.51, 4.99

Results for equation CO

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	115.1368	528.098	0.218	0.827	-919.915	1150.189
L1.CO	-5.4921	5.617	-0.978	0.328	-16.501	5.517
L1.HC	-0.5681	0.629	-0.903	0.367	-1.802	0.665
L1.NO2	3.0909	2.949	1.048	0.295	-2.690	8.872
L1.O3	4.6699	4.042	1.155	0.248	-3.252	12.591
L1.e(CO)	5.7633	5.610	1.027	0.304	-5.231	16.758
L1.e(HC)	0.3010	0.601	0.501	0.616	-0.876	1.478
L1.e(NO2)	-2.8151	2.973	-0.947	0.344	-8.642	3.011
L1.e(O3)	-4.1302	4.056	-1.018	0.309	-12.079	3.819
beta.T	1.6007	0.583	2.747	0.006	0.458	2.743
beta.RH	2.8840	0.475	6.075	0.000	1.954	3.815

Results for equation HC

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	303.4175	343.502	0.883	0.377	-369.834	976.669
L1.CO	-3.2589	3.690	-0.883	0.377	-10.491	3.973
L1.HC	0.0374	0.426	0.088	0.930	-0.797	0.872
L1.NO2	1.6220	1.930	0.841	0.401	-2.160	5.404
L1.O3	2.5675	2.629	0.977	0.329	-2.585	7.720

L1.e(CO)	2.8220	3.702	0.762	0.446	-4.433	10.077
L1.e(HC)	0.1063	0.421	0.253	0.801	-0.719	0.931
L1.e(NO2)	-1.5010	1.972	-0.761	0.447	-5.367	2.365
L1.e(O3)	-1.9887	2.666	-0.746	0.456	-7.215	3.237
beta.T	2.0944	0.696	3.010	0.003	0.731	3.458
beta.RH	1.9269	0.572	3.370	0.001	0.806	3.048

Results for equation NO2

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	48.1959	602.731	0.080	0.936	-1133.135	1229.526
L1.CO	-5.3333	6.317	-0.844	0.399	-17.714	7.047
L1.HC	-0.7126	0.750	-0.950	0.342	-2.183	0.758
L1.NO2	3.3665	3.243	1.038	0.299	-2.990	9.723
L1.O3	4.3382	4.404	0.985	0.325	-4.294	12.970
L1.e(CO)	6.6596	6.356	1.048	0.295	-5.798	19.117
L1.e(HC)	0.7619	0.831	0.917	0.359	-0.867	2.391
L1.e(NO2)	-3.0224	3.268	-0.925	0.355	-9.427	3.382
L1.e(O3)	-5.1563	4.423	-1.166	0.244	-13.824	3.512
beta.T	2.2185	0.769	2.886	0.004	0.712	3.725
beta.RH	0.8024	0.656	1.224	0.221	-0.483	2.087

Results for equation O3

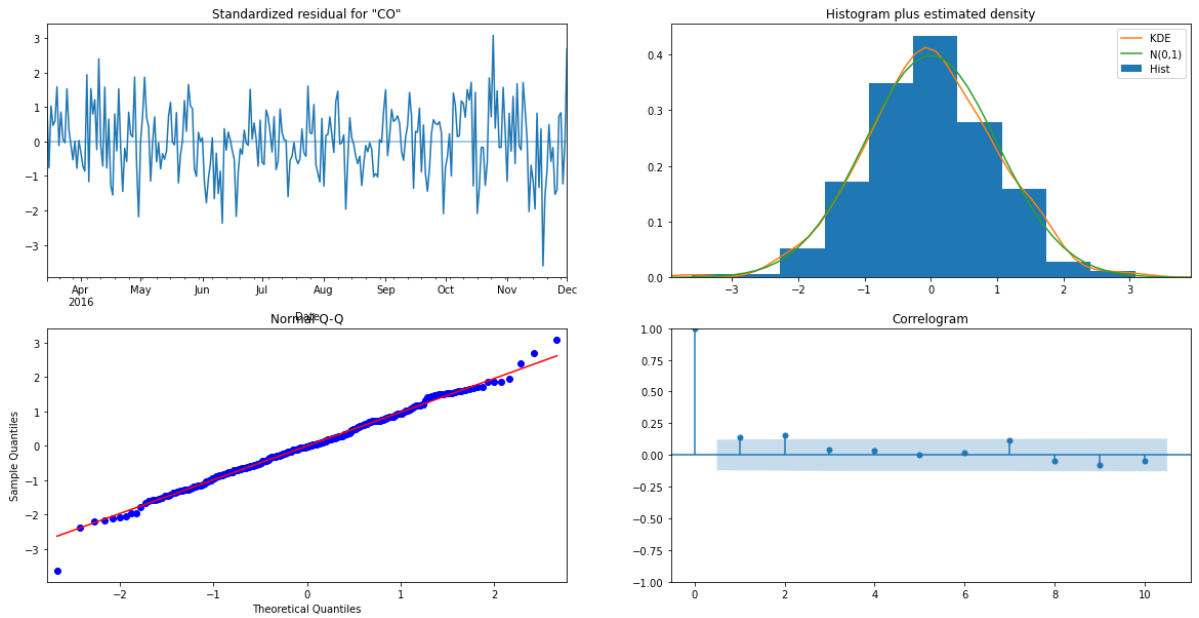
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	217.3890	502.515	0.433	0.665	-767.521	1202.299
L1.CO	-3.6744	5.378	-0.683	0.494	-14.216	6.867
L1.HC	-0.3599	0.670	-0.537	0.591	-1.672	0.953
L1.NO2	1.8498	2.793	0.662	0.508	-3.624	7.324
L1.O3	3.3313	3.785	0.880	0.379	-4.087	10.750
L1.e(CO)	3.1047	5.432	0.572	0.568	-7.543	13.752
L1.e(HC)	-0.0185	0.707	-0.026	0.979	-1.404	1.367
L1.e(NO2)	-1.6876	2.886	-0.585	0.559	-7.343	3.968
L1.e(O3)	-2.0109	3.860	-0.521	0.602	-9.577	5.555
beta.T	0.7086	1.207	0.587	0.557	-1.657	3.075
beta.RH	3.0684	0.968	3.170	0.002	1.171	4.966

Error covariance matrix

	coef	std err	z	P>|z|	[0.025	0.975]
sqrt.var.CO	68.2907	4.427	15.424	0.000	59.613	76.968
sqrt.cov.CO.HC	73.8049	6.529	11.304	0.000	61.008	86.602
sqrt.var.HC	40.6955	2.409	16.892	0.000	35.974	45.418
sqrt.cov.CO.NO2	-75.8759	7.149	-10.613	0.000	-89.888	-61.864
sqrt.cov.HC.NO2	-16.7579	5.636	-2.974	0.003	-27.804	-5.712
sqrt.var.NO2	60.5867	3.234	18.732	0.000	54.248	66.926
sqrt.cov.CO.O3	127.5987	10.243	12.457	0.000	107.522	147.675
sqrt.cov.HC.O3	39.2484	4.155	9.447	0.000	31.105	47.392
sqrt.cov.NO2.O3	-2.8691	4.534	-0.633	0.527	-11.756	6.018
sqrt.var.O3	44.4336	1.815	24.487	0.000	40.877	47.990

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [ ]:
```
model.plot_diagnostics()
plt.show()
```



Observations

Residual plot is not very varied, the Histogram of residual states that the residuals are quite normally distributed. In the correlogram plot, Lag 1 and 2 appears to be outside the range of 5% which means that the model might require some tuning

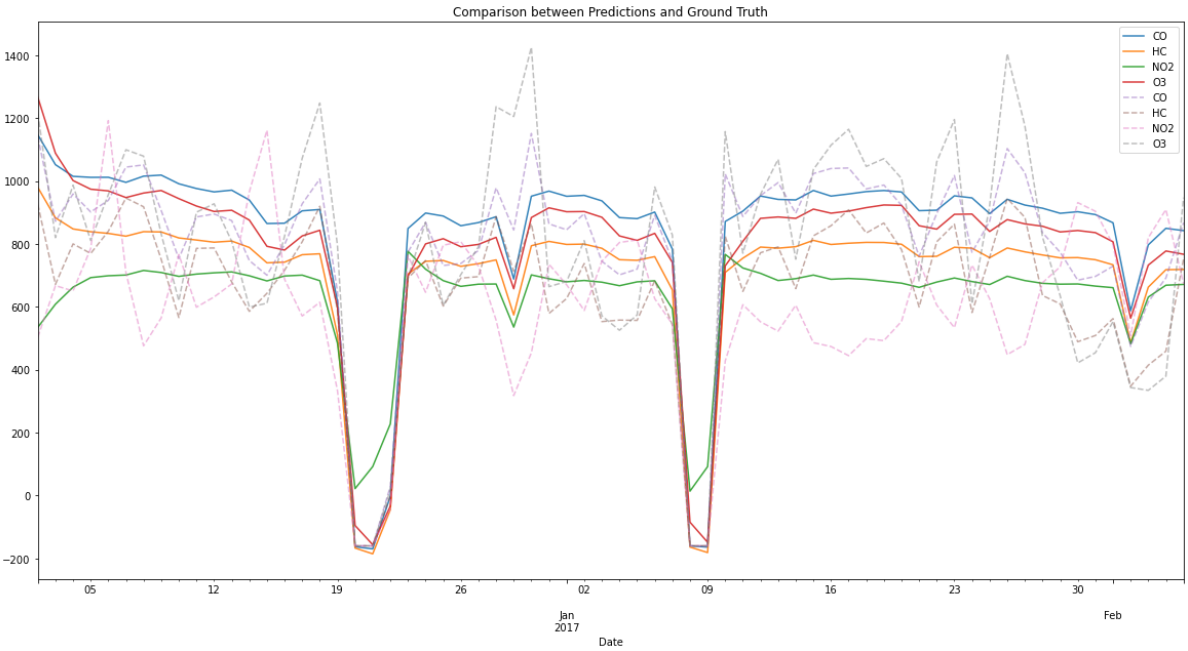
In [ ]:
```
train_pred = model.predict(start=x_train.index[0], end=x_train.index[-1], exog=x_tr
valid_pred = model.predict(start=x_test.index[0], end=x_test.index[-1], exog=x_test

print(
    "Train MAPE:\t\t{:.3f}%".format(
        mean_absolute_percentage_error(y_train, train_pred) * 100
    )
)
print(
    "Train RMSE:\t\t{:.3f}".format(
        mean_squared_error(y_train, train_pred, squared=False)
    )
)
print(
    "Validation MAPE:\t{:.3f}%".format(
        mean_absolute_percentage_error(y_test, valid_pred) * 100
    )
)
print(
    "Validation RMSE:\t{:.3f}".format(
        mean_squared_error(y_test, valid_pred, squared=False)
    )
)
```

```
Train MAPE:             12.823%
Train RMSE:             98.433
Validation MAPE:        126.684%
Validation RMSE:        162.732
```

Observations

1. The model performed very well at the training data and might have a bit of a hiccup in the validation data due to MAPE being more than 100%
2. Both RMSE are relatively close to 0 which means VARMAX model is strong

In [ ]:
```
fig, ax = plt.subplots()
valid_pred.plot(ax=ax, alpha=0.9)
y_test.plot(alpha=0.5, ax=ax, linestyle="--")
ax.set_title("Comparison between Predictions and Ground Truth")
plt.show()
```



Observations The predict values do not have the splits and dips that the time series have which can means that the order selected was not the best. More needs to be done to make the model better

In [ ]:
```
avg, _ = model_cv(VARMAX, exog=x, endog=y, splits=5, order=(1, 1))
avg
```

Out[ ]:
```
train_rmse         115.899651
train_rmse_std      22.994974
valid_rmse         176.237147
valid_rmse_std      34.414391
train_mape           0.237945
valid_mape           0.864523
AIC               7400.438812
BIC               7561.333814
Name: (1, 1), dtype: float64
```

Observations

The order (1,1) has a good valid rmse and train rmse as it is close to 0. We note that VARMAX models take a lot of penalties for AIC and BIC due to it being a more complex model and takes more computational power compared to SARIMA and ARIMA.

Hyperparameter Tuning

In [ ]:
```
scores_df = pd.DataFrame()
p = np.arange(0, 2)
q = np.arange(0, 9)
for i in list(product(p, q)):
    print("Computing (p, q):", i)
    try:
```

```
        scores_df[i], _ = model_cv(
            VARMAX, order=i, splits=5, exog=x, endog=y, max_iter=100
        )
    except Exception as e:
        print("Error computing (p, q):", i)
        print(e)
        pass
```

```
Computing (p, q): (0, 0)
Error computing (p, q): (0, 0)
Invalid VARMAX(p,q) specification; at least one p,q must be greater than zero.
Computing (p, q): (0, 1)
Computing (p, q): (0, 2)
Computing (p, q): (0, 3)
Computing (p, q): (0, 4)
Computing (p, q): (0, 5)
Computing (p, q): (0, 6)
Computing (p, q): (0, 7)
Computing (p, q): (0, 8)
Computing (p, q): (1, 0)
Computing (p, q): (1, 1)
Computing (p, q): (1, 2)
Computing (p, q): (1, 3)
Computing (p, q): (1, 4)
Computing (p, q): (1, 5)
Computing (p, q): (1, 6)
Computing (p, q): (1, 7)
Computing (p, q): (1, 8)
```

In [ ]:

```
scores_df.T.style.apply(
    lambda x: [
        "background-color: green; color: white" if v else "" for v in x == x.min()
    ]
).apply(
    lambda x: ["background-color: red; color: white" if v else "" for v in x == x.m
)
```

Out[ ]:

	train_rmse	train_rmse_std	valid_rmse	valid_rmse_std	train_mape	valid_mape	AIC
(0, 1)	93.002998	5.984985	149.322418	20.420019	0.119277	0.341591	7265.277936
(0, 2)	89.894723	5.060442	148.138933	21.556525	0.108975	0.268482	7210.076954
(0, 3)	87.454531	5.161977	147.877021	23.014975	0.124809	0.366505	7198.910173
(0, 4)	85.775759	6.370535	147.983182	23.274048	0.115429	0.407572	7203.596825
(0, 5)	86.101466	5.380451	149.164965	23.486088	0.122183	0.524566	7211.172464
(0, 6)	84.415567	5.695552	148.354794	21.899687	0.109404	0.410920	7224.695291
(0, 7)	81.483808	5.960166	147.263737	22.531687	0.110160	0.419208	7219.963276
(0, 8)	80.425377	6.208739	149.015601	23.068919	0.093411	0.376010	7246.019678
(1, 0)	158.196426	11.585930	412.602575	191.908196	0.347873	7.344966	8309.787134
(1, 1)	121.019752	26.809005	191.197882	23.561448	0.270154	1.271650	7544.308337
(1, 2)	114.368379	18.039646	195.553870	23.238892	0.286826	1.729729	7382.320308
(1, 3)	116.987539	24.907146	213.782802	75.596954	0.318983	2.305007	7413.075070
(1, 4)	111.762610	14.711369	204.025143	48.382068	0.328935	2.840975	7458.830950
(1, 5)	138.268160	40.031101	381.715221	200.632227	0.336641	2.836224	7574.868847
(1, 6)	121.886049	27.000724	213.582887	39.151251	0.324774	2.986266	7559.088203
(1, 7)	129.453995	31.606046	402.582310	176.558449	0.326138	2.749244	7669.510859
(1, 8)	126.038263	42.463538	454.537465	432.270181	0.334832	2.353588	7651.635889

Observations

The order (0,7) performed the best as it has the lowest valid rmse. We also note that by increasing the p value (Auto Regressive) makes the score worst as the valid rmse increase

drastically when it is added. Order (0,8) is also relatively good as it has the lowest train rmse and train_mape.

Model Selection

Based on the valid rmse, generally VARMAX performs more effectively compared to SARIMA. This is likely due to the correlation between the different variables that VARMAX is able to use in assistance to the model which means that we will be using VARMAX as the main model.

Kaggle Test Set Predictions

After we are satisfied with the model performance on the train and validation set, it is about time for us to generate the final prediction for the test set and submit it on the Kaggle competition page.

In [ ]:
```
df_test = pd.read_csv("./test.csv").drop("id", axis=1)
df_test["Date"] = pd.to_datetime(df_test["Date"], format="%d/%m/%Y")
df_test
```

Out[ ]:

	Date	T	RH	Gas	Unnamed: 5	Unnamed: 6
0	2017-02-06	6.616667	51.734375	CO	NaN	NaN
1	2017-02-07	7.613194	43.930903	CO	NaN	NaN
2	2017-02-08	7.252083	50.966667	CO	NaN	NaN
3	2017-02-09	7.473611	50.166319	CO	NaN	NaN
4	2017-02-10	5.571875	46.604167	CO	NaN	NaN
...	...	...	...	...	...	...
247	2017-04-05	17.554167	50.092708	O3	NaN	NaN
248	2017-04-06	15.919792	35.959722	O3	NaN	NaN
249	2017-04-07	15.489583	32.213542	O3	NaN	NaN
250	2017-04-08	18.381250	33.686458	O3	NaN	NaN
251	2017-04-09	16.966667	42.791667	O3	NaN	NaN

252 rows × 6 columns

In [ ]:
```
# prepare exogenous testing set
X_test = df_test.set_index("Date")[["T", "RH"]].resample("d").mean().to_period("D")
X_test
```

Out[ ]:

Date	T	RH
2017-02-06	6.616667	51.734375
2017-02-07	7.613194	43.930903
2017-02-08	7.252083	50.966667
2017-02-09	7.473611	50.166319
2017-02-10	5.571875	46.604167
...	...	...
2017-04-05	17.554167	50.092708
2017-04-06	15.919792	35.959722
2017-04-07	15.489583	32.213542
2017-04-08	18.381250	33.686458
2017-04-09	16.966667	42.791667

63 rows × 2 columns

In [ ]:

```
model = VARMAX(endog=y, exog=x, order=(0, 7), trend="c").fit(maxiter=1000, disp=Fal

# making a directory if it does not exist
Path("./models").mkdir(parents=True, exist_ok=True)

# saving model to pickle
with open("./models/varmax07.pkl", "wb") as f:
    pickle.dump(model, f)
    print("Model saved!")

model.summary()
```

```
Model saved!
```

Out[ ]:

Statespace Model Results

Dep. Variable:	['CO', 'HC', 'NO2', 'O3']	**No. Observations:**	328
Model:	VMAX(7)	**Log Likelihood**	-7098.015
	+ intercept	**AIC**	14464.031
Date:	Thu, 11 Aug 2022	**BIC**	14972.295
Time:	22:37:35	**HQIC**	14666.814
Sample:	03-15-2016		
	- 02-05-2017		
Covariance Type:	opg		

Ljung-Box (L1) (Q):	4.93, 19.73, 4.18, 7.84	**Jarque-Bera (JB):**	11.03, 0.17, 1057.23, 32.33
Prob(Q):	0.03, 0.00, 0.04, 0.01	**Prob(JB):**	0.00, 0.92, 0.00, 0.00
Heteroskedasticity (H):	1.79, 0.87, 1.88, 2.39	**Skew:**	-0.21, -0.06, 1.54, 0.41
Prob(H) (two-sided):	0.00, 0.46, 0.00, 0.00	**Kurtosis:**	3.80, 2.99, 11.24, 4.30

Results for equation CO

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	685.8639	21.761	31.519	0.000	643.214	728.514
L1.e(CO)	0.4907	0.250	1.965	0.049	0.001	0.980
L1.e(HC)	-0.1527	0.213	-0.718	0.473	-0.569	0.264
L1.e(NO2)	0.1581	0.120	1.317	0.188	-0.077	0.393
L1.e(O3)	0.2525	0.130	1.938	0.053	-0.003	0.508
L2.e(CO)	0.2124	0.263	0.807	0.420	-0.304	0.729
L2.e(HC)	-0.0736	0.218	-0.338	0.735	-0.500	0.353
L2.e(NO2)	0.1755	0.128	1.366	0.172	-0.076	0.427
L2.e(O3)	0.1370	0.145	0.942	0.346	-0.148	0.422
L3.e(CO)	0.3549	0.299	1.186	0.236	-0.232	0.941
L3.e(HC)	-0.0589	0.245	-0.240	0.810	-0.540	0.422
L3.e(NO2)	0.2617	0.136	1.924	0.054	-0.005	0.528
L3.e(O3)	0.0195	0.151	0.129	0.897	-0.276	0.315
L4.e(CO)	0.2412	0.301	0.800	0.424	-0.349	0.832
L4.e(HC)	-0.0145	0.264	-0.055	0.956	-0.532	0.503
L4.e(NO2)	0.1825	0.121	1.508	0.132	-0.055	0.420
L4.e(O3)	-0.0463	0.160	-0.289	0.772	-0.360	0.267
L5.e(CO)	0.1156	0.329	0.351	0.725	-0.529	0.761

L5.e(HC)	-0.1880	0.265	-0.710	0.478	-0.707	0.331
L5.e(NO2)	0.1219	0.154	0.794	0.427	-0.179	0.423
L5.e(O3)	0.0512	0.154	0.332	0.740	-0.251	0.353
L6.e(CO)	0.0791	0.290	0.273	0.785	-0.489	0.647
L6.e(HC)	-0.1545	0.261	-0.592	0.554	-0.666	0.357
L6.e(NO2)	0.0733	0.121	0.604	0.546	-0.165	0.311
L6.e(O3)	0.0434	0.155	0.280	0.780	-0.261	0.347
L7.e(CO)	0.0751	0.226	0.332	0.740	-0.368	0.519
L7.e(HC)	0.2463	0.214	1.150	0.250	-0.173	0.666
L7.e(NO2)	0.0678	0.082	0.829	0.407	-0.093	0.228
L7.e(O3)	-0.0809	0.125	-0.649	0.516	-0.325	0.163
beta.T	0.6651	0.664	1.002	0.316	-0.635	1.966
beta.RH	3.6170	0.552	6.550	0.000	2.535	4.699

Results for equation HC

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	629.5415	23.681	26.584	0.000	583.127	675.956
L1.e(CO)	-0.2401	0.292	-0.822	0.411	-0.812	0.332
L1.e(HC)	0.3034	0.252	1.204	0.229	-0.191	0.797
L1.e(NO2)	0.0687	0.140	0.490	0.624	-0.206	0.344
L1.e(O3)	0.2730	0.160	1.709	0.087	-0.040	0.586
L2.e(CO)	-0.0876	0.306	-0.286	0.775	-0.688	0.513
L2.e(HC)	0.1475	0.262	0.563	0.573	-0.366	0.661
L2.e(NO2)	0.0753	0.151	0.499	0.618	-0.220	0.371
L2.e(O3)	0.0644	0.162	0.399	0.690	-0.252	0.381
L3.e(CO)	0.2388	0.333	0.717	0.473	-0.414	0.891
L3.e(HC)	0.0711	0.291	0.245	0.807	-0.499	0.641
L3.e(NO2)	0.1286	0.152	0.848	0.397	-0.169	0.426
L3.e(O3)	-0.0803	0.177	-0.455	0.649	-0.426	0.266
L4.e(CO)	0.2270	0.329	0.690	0.490	-0.418	0.872
L4.e(HC)	0.0768	0.277	0.278	0.781	-0.465	0.619
L4.e(NO2)	0.0795	0.136	0.585	0.559	-0.187	0.346
L4.e(O3)	-0.1597	0.174	-0.917	0.359	-0.501	0.182
L5.e(CO)	0.2639	0.339	0.778	0.436	-0.401	0.929

L5.e(HC)	-0.2728	0.268	-1.019	0.308	-0.797	0.252
L5.e(NO2)	0.0582	0.171	0.340	0.734	-0.278	0.394
L5.e(O3)	-0.0041	0.170	-0.024	0.981	-0.337	0.328
L6.e(CO)	-0.0502	0.314	-0.160	0.873	-0.666	0.566
L6.e(HC)	-0.1255	0.287	-0.437	0.662	-0.689	0.438
L6.e(NO2)	0.0449	0.134	0.335	0.738	-0.218	0.308
L6.e(O3)	0.1059	0.160	0.662	0.508	-0.208	0.419
L7.e(CO)	-0.2439	0.279	-0.875	0.381	-0.790	0.302
L7.e(HC)	0.6594	0.251	2.626	0.009	0.167	1.151
L7.e(NO2)	-0.0250	0.099	-0.253	0.800	-0.219	0.169
L7.e(O3)	-0.1332	0.143	-0.933	0.351	-0.413	0.147
beta.T	2.1204	0.798	2.658	0.008	0.557	3.684
beta.RH	1.8283	0.667	2.739	0.006	0.520	3.136

Results for equation NO2

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	602.8459	34.839	17.304	0.000	534.564	671.128
L1.e(CO)	-0.0759	0.361	-0.210	0.833	-0.783	0.631
L1.e(HC)	0.1022	0.311	0.329	0.743	-0.507	0.712
L1.e(NO2)	0.5609	0.160	3.504	0.000	0.247	0.875
L1.e(O3)	-0.1250	0.201	-0.621	0.534	-0.519	0.269
L2.e(CO)	-0.0290	0.465	-0.062	0.950	-0.941	0.882
L2.e(HC)	-0.1752	0.387	-0.453	0.651	-0.934	0.584
L2.e(NO2)	0.2260	0.189	1.193	0.233	-0.145	0.597
L2.e(O3)	0.0531	0.258	0.206	0.837	-0.452	0.558
L3.e(CO)	-0.4527	0.444	-1.019	0.308	-1.323	0.418
L3.e(HC)	-0.0269	0.386	-0.070	0.944	-0.784	0.730
L3.e(NO2)	-0.0292	0.179	-0.163	0.871	-0.380	0.322
L3.e(O3)	0.1461	0.258	0.567	0.571	-0.359	0.651
L4.e(CO)	-0.0529	0.396	-0.134	0.894	-0.829	0.723
L4.e(HC)	-0.1525	0.412	-0.370	0.711	-0.960	0.655
L4.e(NO2)	-0.0622	0.166	-0.374	0.708	-0.388	0.264
L4.e(O3)	0.0851	0.228	0.373	0.709	-0.362	0.532
L5.e(CO)	0.4828	0.426	1.132	0.257	-0.353	1.319

L5.e(HC)	0.1963	0.379	0.518	0.605	-0.547	0.940
L5.e(NO2)	0.0869	0.195	0.445	0.656	-0.296	0.470
L5.e(O3)	-0.2144	0.220	-0.976	0.329	-0.645	0.216
L6.e(CO)	0.2111	0.432	0.489	0.625	-0.636	1.058
L6.e(HC)	0.3580	0.361	0.991	0.321	-0.350	1.066
L6.e(NO2)	0.1772	0.187	0.950	0.342	-0.188	0.543
L6.e(O3)	-0.1587	0.243	-0.652	0.514	-0.636	0.318
L7.e(CO)	-0.1639	0.386	-0.425	0.671	-0.920	0.592
L7.e(HC)	-0.3229	0.364	-0.888	0.375	-1.036	0.390
L7.e(NO2)	0.0584	0.117	0.501	0.616	-0.170	0.287
L7.e(O3)	0.2199	0.203	1.085	0.278	-0.177	0.617
beta.T	3.7148	0.990	3.753	0.000	1.775	5.655
beta.RH	0.0155	0.836	0.019	0.985	-1.622	1.653

Results for equation O3

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	601.6083	39.005	15.424	0.000	525.160	678.056
L1.e(CO)	-0.7652	0.510	-1.500	0.134	-1.765	0.235
L1.e(HC)	-0.2130	0.470	-0.453	0.651	-1.135	0.709
L1.e(NO2)	0.0920	0.254	0.362	0.717	-0.406	0.590
L1.e(O3)	1.0920	0.285	3.832	0.000	0.533	1.651
L2.e(CO)	-0.2924	0.552	-0.529	0.597	-1.375	0.790
L2.e(HC)	-0.0724	0.508	-0.143	0.887	-1.068	0.923
L2.e(NO2)	0.0352	0.273	0.129	0.897	-0.500	0.570
L2.e(O3)	0.3085	0.290	1.064	0.287	-0.260	0.877
L3.e(CO)	0.5408	0.612	0.883	0.377	-0.659	1.741
L3.e(HC)	-0.0505	0.526	-0.096	0.923	-1.081	0.980
L3.e(NO2)	0.2484	0.279	0.891	0.373	-0.298	0.795
L3.e(O3)	-0.1243	0.318	-0.391	0.696	-0.748	0.499
L4.e(CO)	0.4881	0.594	0.822	0.411	-0.676	1.653
L4.e(HC)	0.1713	0.504	0.340	0.734	-0.817	1.160
L4.e(NO2)	0.1908	0.260	0.733	0.464	-0.320	0.701
L4.e(O3)	-0.2991	0.314	-0.954	0.340	-0.914	0.316
L5.e(CO)	0.3745	0.650	0.576	0.565	-0.900	1.649

L5.e(HC)	-0.4065	0.536	-0.759	0.448	-1.457	0.644
L5.e(NO2)	0.0286	0.326	0.088	0.930	-0.611	0.668
L5.e(O3)	-0.0536	0.322	-0.167	0.868	-0.684	0.577
L6.e(CO)	-0.1473	0.580	-0.254	0.800	-1.285	0.990
L6.e(HC)	-0.3591	0.561	-0.640	0.522	-1.459	0.741
L6.e(NO2)	0.0091	0.255	0.036	0.971	-0.491	0.509
L6.e(O3)	0.2376	0.319	0.746	0.456	-0.387	0.862
L7.e(CO)	-0.3120	0.509	-0.613	0.540	-1.309	0.685
L7.e(HC)	0.7920	0.477	1.660	0.097	-0.143	1.727
L7.e(NO2)	-0.0090	0.183	-0.049	0.961	-0.367	0.349
L7.e(O3)	-0.1347	0.270	-0.500	0.617	-0.663	0.394
beta.T	-0.8975	1.253	-0.716	0.474	-3.354	1.559
beta.RH	4.7617	1.046	4.552	0.000	2.711	6.812

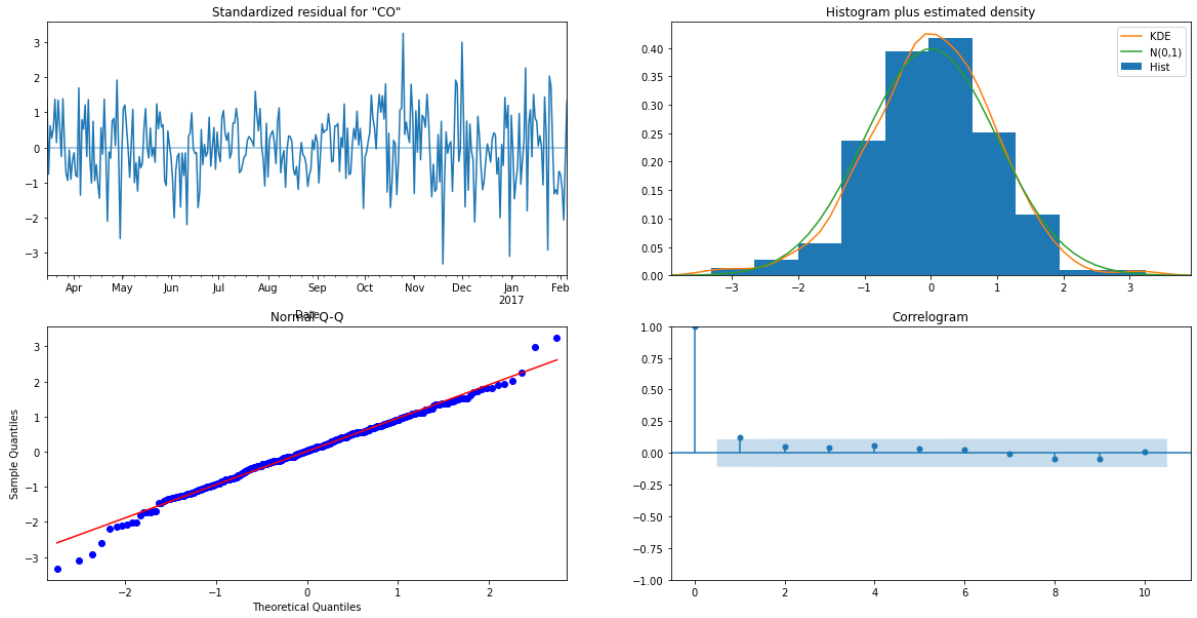
Error covariance matrix

	coef	std err	z	P>|z|	[0.025	0.975]
sqrt.var.CO	71.4015	5.318	13.426	0.000	60.978	81.825
sqrt.cov.CO.HC	79.5154	7.654	10.389	0.000	64.514	94.517
sqrt.var.HC	39.6509	2.895	13.696	0.000	33.977	45.325
sqrt.cov.CO.NO2	-80.2762	9.532	-8.421	0.000	-98.959	-61.593
sqrt.cov.HC.NO2	-14.4625	7.517	-1.924	0.054	-29.196	0.271
sqrt.var.NO2	67.8179	3.991	16.994	0.000	59.996	75.640
sqrt.cov.CO.O3	143.9678	13.106	10.985	0.000	118.280	169.656
sqrt.cov.HC.O3	33.8801	5.028	6.738	0.000	24.025	43.735
sqrt.cov.NO2.O3	-1.0858	6.225	-0.174	0.862	-13.287	11.115
sqrt.var.O3	48.0760	2.981	16.129	0.000	42.234	53.918

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [ ]:
```
model = VARMAX(endog=y, exog=x, order=(0, 8), trend="c").fit(maxiter=1000, disp=Fal

# making a directory if it does not exist
Path("./models").mkdir(parents=True, exist_ok=True)

# saving model to pickle
```

```
with open("./models/varmax08.pkl", "wb") as f:
    pickle.dump(model, f)
    print("Model saved!")

model.summary()
```

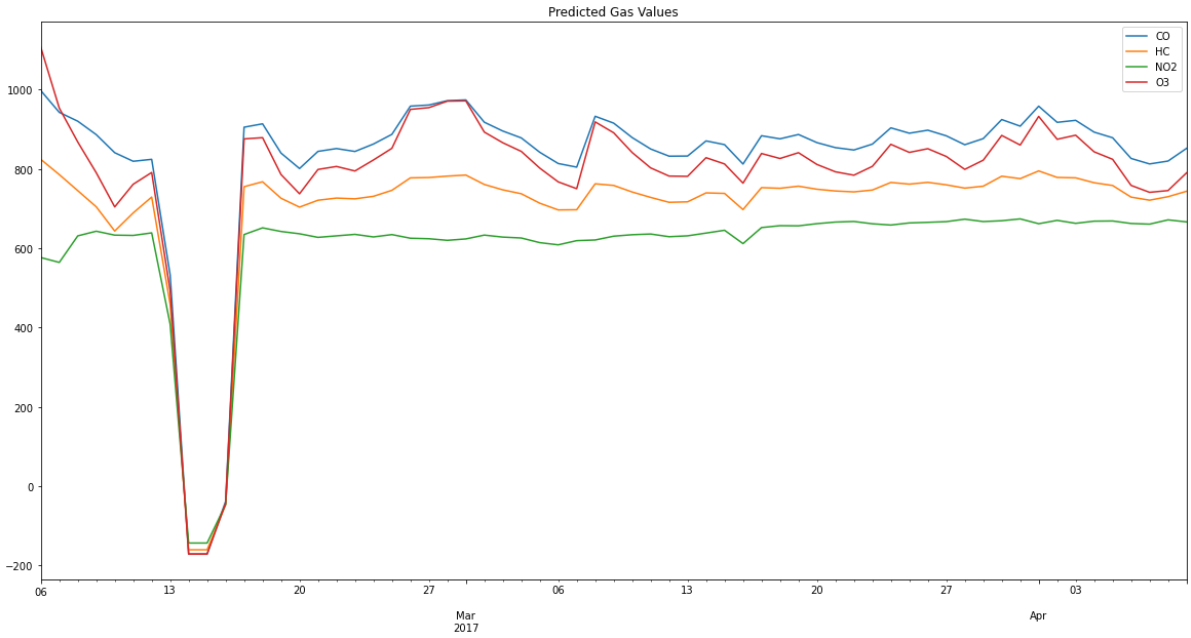


In [ ]:
```
model.plot_diagnostics()
plt.show()
```

Out[ ]:

id	variable	Value
0	CO	997.628033
1	CO	942.750789
2	CO	920.430462
3	CO	886.595934
4	CO	841.008239
...	...	...
247	O3	824.380572
248	O3	758.550143
249	O3	741.098003
250	O3	745.516315
251	O3	790.142356

252 rows × 2 columns



In [ ]:
```
# generate predictions
predictions = model.predict(start=X_test.index[0], end=X_test.index[-1], exog=X_tes
predictions.plot()
plt.title("Predicted Gas Values")
predictions = pd.melt(
    predictions, value_vars=["CO", "HC", "NO2", "O3"], value_name="Value"
)
predictions.index.rename("id", inplace=True)
predictions
```

```
Exported predictions to .csv
```

Kaggle Scores

RMSE: 152.09558

We will be using (0,8) as the second value as it has the 2 best parameter and comparing kaggle scores

In [ ]:
```
# making a directory if it does not exist
Path("./kaggle").mkdir(parents=True, exist_ok=True)
predictions["Value"].to_csv("./kaggle/varmax08.csv")
print("Exported predictions to .csv")
```

```
Model saved!
```

Out[ ]:

Statespace Model Results

Dep. Variable:	['CO', 'HC', 'NO2', 'O3']	**No. Observations:**	328
Model:	VMAX(8)	**Log Likelihood**	-7091.633
	+ intercept	**AIC**	14483.266
Date:	Thu, 11 Aug 2022	**BIC**	15052.218
Time:	22:41:27	**HQIC**	14710.261
Sample:	03-15-2016		
	- 02-05-2017		
Covariance Type:	opg		

Ljung-Box (L1) (Q):	4.51, 11.74, 7.89, 12.78	**Jarque-Bera (JB):**	3.18, 0.01, 677.46, 46.73
Prob(Q):	0.03, 0.00, 0.00, 0.00	**Prob(JB):**	0.20, 1.00, 0.00, 0.00
Heteroskedasticity (H):	1.71, 1.04, 1.82, 2.32	**Skew:**	-0.09, 0.01, 1.40, 0.59
Prob(H) (two-sided):	0.01, 0.85, 0.00, 0.00	**Kurtosis:**	3.45, 3.00, 9.46, 4.43

Results for equation CO

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	685.8356	24.523	27.967	0.000	637.771	733.900
L1.e(CO)	0.1183	0.540	0.219	0.827	-0.941	1.177
L1.e(HC)	-0.1375	0.340	-0.404	0.686	-0.804	0.529
L1.e(NO2)	0.1514	0.236	0.641	0.521	-0.311	0.614
L1.e(O3)	0.4098	0.214	1.917	0.055	-0.009	0.829
L2.e(CO)	0.2265	0.643	0.352	0.725	-1.034	1.487
L2.e(HC)	0.1381	0.429	0.322	0.747	-0.702	0.978
L2.e(NO2)	0.1691	0.241	0.701	0.484	-0.304	0.642
L2.e(O3)	0.0094	0.283	0.033	0.974	-0.545	0.563
L3.e(CO)	0.5891	0.639	0.921	0.357	-0.664	1.842
L3.e(HC)	0.1262	0.433	0.291	0.771	-0.723	0.975
L3.e(NO2)	0.3714	0.212	1.750	0.080	-0.045	0.787
L3.e(O3)	-0.1358	0.338	-0.402	0.687	-0.798	0.526
L4.e(CO)	0.4675	0.501	0.934	0.350	-0.514	1.449
L4.e(HC)	0.0445	0.356	0.125	0.901	-0.654	0.743
L4.e(NO2)	0.2668	0.166	1.609	0.108	-0.058	0.592
L4.e(O3)	-0.1475	0.251	-0.588	0.557	-0.639	0.344
L5.e(CO)	0.2988	0.449	0.666	0.506	-0.581	1.179

L5.e(HC)	-0.1417	0.312	-0.454	0.650	-0.754	0.470
L5.e(NO2)	0.0798	0.167	0.477	0.633	-0.248	0.408
L5.e(O3)	-0.0910	0.192	-0.473	0.636	-0.468	0.286
L6.e(CO)	0.4309	0.506	0.852	0.394	-0.560	1.422
L6.e(HC)	-0.2918	0.337	-0.867	0.386	-0.952	0.368
L6.e(NO2)	0.1512	0.197	0.769	0.442	-0.234	0.537
L6.e(O3)	-0.0328	0.199	-0.165	0.869	-0.422	0.357
L7.e(CO)	0.2016	0.531	0.380	0.704	-0.839	1.242
L7.e(HC)	0.1191	0.312	0.382	0.703	-0.492	0.731
L7.e(NO2)	0.0889	0.192	0.462	0.644	-0.288	0.465
L7.e(O3)	-0.0792	0.191	-0.414	0.679	-0.454	0.295
L8.e(CO)	0.1436	0.420	0.342	0.733	-0.680	0.967
L8.e(HC)	-0.0345	0.245	-0.141	0.888	-0.514	0.445
L8.e(NO2)	-0.0427	0.138	-0.308	0.758	-0.314	0.228
L8.e(O3)	-0.0520	0.162	-0.322	0.748	-0.369	0.265
beta.T	0.5701	0.743	0.767	0.443	-0.887	2.027
beta.RH	3.7081	0.624	5.939	0.000	2.484	4.932

Results for equation HC

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	629.5331	26.446	23.804	0.000	577.699	681.367
L1.e(CO)	-0.4435	0.572	-0.776	0.438	-1.564	0.677
L1.e(HC)	0.3235	0.288	1.122	0.262	-0.241	0.888
L1.e(NO2)	0.0879	0.234	0.375	0.707	-0.371	0.547
L1.e(O3)	0.4018	0.220	1.828	0.068	-0.029	0.832
L2.e(CO)	0.1854	0.673	0.275	0.783	-1.134	1.505
L2.e(HC)	0.2063	0.365	0.565	0.572	-0.509	0.921
L2.e(NO2)	0.0294	0.238	0.124	0.901	-0.436	0.495
L2.e(O3)	-0.0907	0.255	-0.356	0.722	-0.590	0.409
L3.e(CO)	0.2131	0.720	0.296	0.767	-1.198	1.624
L3.e(HC)	0.1861	0.346	0.538	0.591	-0.492	0.864
L3.e(NO2)	0.1195	0.225	0.531	0.595	-0.321	0.560
L3.e(O3)	-0.1332	0.277	-0.481	0.630	-0.676	0.409
L4.e(CO)	0.2037	0.584	0.349	0.727	-0.942	1.349

L4.e(HC)	0.0986	0.317	0.311	0.756	-0.524	0.721
L4.e(NO2)	0.0471	0.182	0.259	0.796	-0.309	0.403
L4.e(O3)	-0.1797	0.224	-0.801	0.423	-0.620	0.260
L5.e(CO)	0.1157	0.461	0.251	0.802	-0.787	1.019
L5.e(HC)	-0.2627	0.311	-0.845	0.398	-0.872	0.347
L5.e(NO2)	-0.1032	0.171	-0.605	0.545	-0.438	0.231
L5.e(O3)	-0.0431	0.204	-0.211	0.833	-0.443	0.357
L6.e(CO)	-0.1173	0.554	-0.212	0.832	-1.203	0.968
L6.e(HC)	-0.2897	0.338	-0.856	0.392	-0.953	0.374
L6.e(NO2)	0.0122	0.206	0.059	0.953	-0.391	0.415
L6.e(O3)	0.1614	0.210	0.767	0.443	-0.251	0.574
L7.e(CO)	-0.3488	0.547	-0.637	0.524	-1.422	0.724
L7.e(HC)	0.5661	0.330	1.714	0.087	-0.081	1.213
L7.e(NO2)	-0.0342	0.212	-0.161	0.872	-0.450	0.382
L7.e(O3)	-0.0430	0.188	-0.229	0.819	-0.412	0.326
L8.e(CO)	-0.0426	0.442	-0.096	0.923	-0.909	0.824
L8.e(HC)	0.2034	0.272	0.748	0.454	-0.329	0.736
L8.e(NO2)	0.0090	0.163	0.055	0.956	-0.311	0.329
L8.e(O3)	-0.0173	0.183	-0.095	0.924	-0.376	0.341
beta.T	2.0947	0.888	2.359	0.018	0.355	3.835
beta.RH	1.8466	0.744	2.481	0.013	0.388	3.305

Results for equation NO2

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	602.8451	45.475	13.257	0.000	513.716	691.974
L1.e(CO)	-0.3639	0.747	-0.487	0.626	-1.827	1.099
L1.e(HC)	0.1831	0.561	0.326	0.744	-0.917	1.283
L1.e(NO2)	0.2600	0.355	0.732	0.464	-0.436	0.956
L1.e(O3)	-0.2641	0.396	-0.666	0.505	-1.041	0.513
L2.e(CO)	-0.6775	0.869	-0.780	0.436	-2.381	1.026
L2.e(HC)	-0.4270	0.706	-0.605	0.546	-1.812	0.958
L2.e(NO2)	0.0151	0.362	0.042	0.967	-0.695	0.726
L2.e(O3)	0.3092	0.508	0.609	0.543	-0.686	1.305
L3.e(CO)	-1.1778	1.022	-1.153	0.249	-3.181	0.825

L3.e(HC)	0.0837	0.661	0.127	0.899	-1.211	1.379
L3.e(NO2)	-0.0619	0.306	-0.202	0.840	-0.661	0.537
L3.e(O3)	0.3483	0.497	0.701	0.483	-0.626	1.322
L4.e(CO)	-0.1544	1.061	-0.146	0.884	-2.233	1.924
L4.e(HC)	-0.0861	0.651	-0.132	0.895	-1.362	1.189
L4.e(NO2)	0.1572	0.258	0.610	0.542	-0.348	0.662
L4.e(O3)	0.1964	0.420	0.467	0.640	-0.628	1.020
L5.e(CO)	0.3822	1.030	0.371	0.711	-1.637	2.402
L5.e(HC)	0.0757	0.680	0.111	0.911	-1.257	1.408
L5.e(NO2)	0.2457	0.270	0.910	0.363	-0.284	0.775
L5.e(O3)	-0.0523	0.422	-0.124	0.901	-0.880	0.776
L6.e(CO)	0.0161	0.927	0.017	0.986	-1.802	1.834
L6.e(HC)	0.5908	0.680	0.868	0.385	-0.743	1.924
L6.e(NO2)	0.2821	0.294	0.961	0.337	-0.293	0.858
L6.e(O3)	-0.1233	0.438	-0.281	0.778	-0.982	0.736
L7.e(CO)	0.5599	0.786	0.713	0.476	-0.980	2.100
L7.e(HC)	-0.0624	0.609	-0.102	0.918	-1.257	1.132
L7.e(NO2)	0.2839	0.283	1.004	0.315	-0.270	0.838
L7.e(O3)	-0.1221	0.367	-0.332	0.740	-0.842	0.598
L8.e(CO)	1.5249	0.568	2.685	0.007	0.412	2.638
L8.e(HC)	-0.2664	0.505	-0.527	0.598	-1.257	0.724
L8.e(NO2)	0.4698	0.225	2.092	0.036	0.030	0.910
L8.e(O3)	-0.3388	0.290	-1.170	0.242	-0.906	0.229
beta.T	3.4976	1.242	2.817	0.005	1.064	5.931
beta.RH	0.1338	1.078	0.124	0.901	-1.978	2.246

Results for equation O3

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	601.6244	44.607	13.487	0.000	514.196	689.053
L1.e(CO)	-0.7557	1.092	-0.692	0.489	-2.896	1.384
L1.e(HC)	-0.2431	0.570	-0.426	0.670	-1.360	0.874
L1.e(NO2)	0.2285	0.466	0.490	0.624	-0.685	1.142
L1.e(O3)	1.2278	0.427	2.878	0.004	0.392	2.064
L2.e(CO)	0.4584	1.344	0.341	0.733	-2.176	3.093

L2.e(HC)	0.1462	0.671	0.218	0.828	-1.169	1.462
L2.e(NO2)	0.0565	0.468	0.121	0.904	-0.860	0.973
L2.e(O3)	-0.0657	0.514	-0.128	0.898	-1.074	0.942
L3.e(CO)	0.7424	1.404	0.529	0.597	-2.009	3.494
L3.e(HC)	0.1441	0.633	0.228	0.820	-1.096	1.384
L3.e(NO2)	0.3306	0.429	0.771	0.441	-0.510	1.171
L3.e(O3)	-0.2404	0.536	-0.449	0.654	-1.291	0.810
L4.e(CO)	0.1604	1.105	0.145	0.885	-2.006	2.326
L4.e(HC)	0.1482	0.618	0.240	0.810	-1.063	1.359
L4.e(NO2)	0.0744	0.368	0.202	0.840	-0.646	0.795
L4.e(O3)	-0.2051	0.401	-0.511	0.609	-0.991	0.581
L5.e(CO)	0.2558	0.905	0.283	0.777	-1.517	2.029
L5.e(HC)	-0.4827	0.623	-0.775	0.438	-1.703	0.738
L5.e(NO2)	-0.2575	0.351	-0.735	0.463	-0.945	0.429
L5.e(O3)	-0.1401	0.394	-0.355	0.722	-0.913	0.633
L6.e(CO)	-0.1564	1.054	-0.148	0.882	-2.223	1.910
L6.e(HC)	-0.6117	0.659	-0.928	0.353	-1.904	0.680
L6.e(NO2)	-0.0288	0.406	-0.071	0.943	-0.824	0.767
L6.e(O3)	0.2725	0.421	0.647	0.517	-0.552	1.097
L7.e(CO)	-0.4347	1.055	-0.412	0.680	-2.503	1.634
L7.e(HC)	0.6924	0.633	1.093	0.274	-0.549	1.934
L7.e(NO2)	0.0107	0.394	0.027	0.978	-0.761	0.783
L7.e(O3)	-0.0402	0.385	-0.105	0.917	-0.794	0.714
L8.e(CO)	-0.1603	0.899	-0.178	0.859	-1.922	1.602
L8.e(HC)	0.1520	0.522	0.291	0.771	-0.872	1.176
L8.e(NO2)	-0.0145	0.301	-0.048	0.961	-0.604	0.575
L8.e(O3)	0.0701	0.335	0.209	0.834	-0.587	0.727
beta.T	-0.8484	1.443	-0.588	0.557	-3.677	1.981
beta.RH	4.7549	1.202	3.957	0.000	2.400	7.110

Error covariance matrix

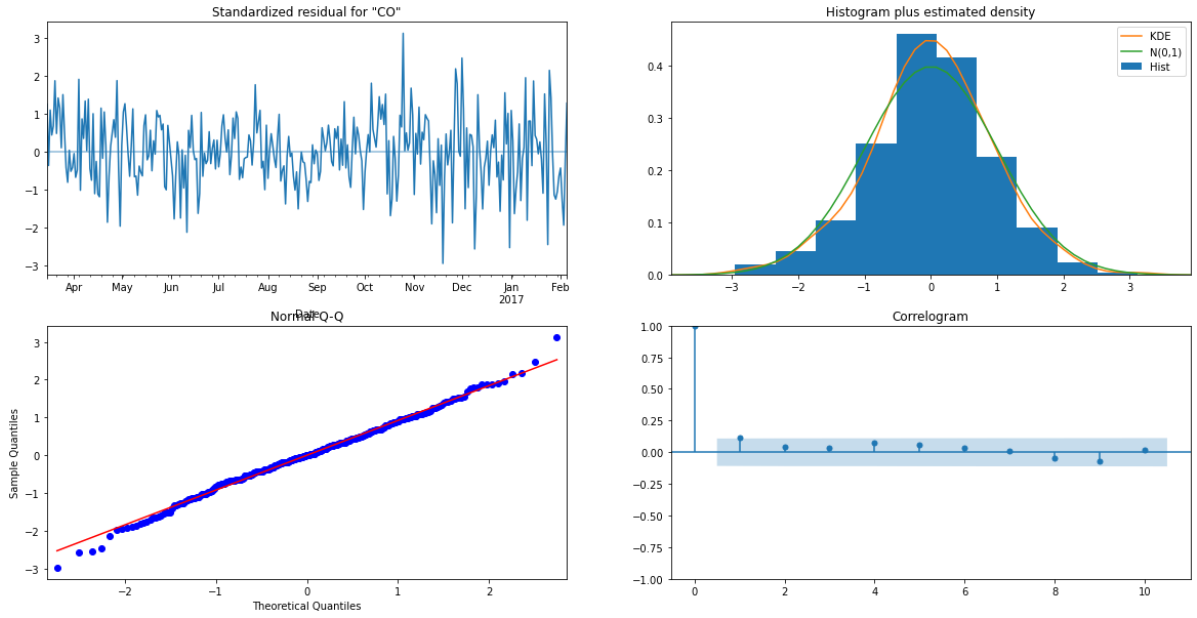
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
sqrt.var.CO	71.2068	7.225	9.856	0.000	57.046	85.367
sqrt.cov.CO.HC	79.6116	9.198	8.655	0.000	61.583	97.640

sqrt.var.HC	39.6937	5.347	7.424	0.000	29.214	50.173
sqrt.cov.CO.NO2	-80.2902	15.805	-5.080	0.000	-111.268	-49.312
sqrt.cov.HC.NO2	-14.3333	15.876	-0.903	0.367	-45.449	16.782
sqrt.var.NO2	67.5224	7.847	8.605	0.000	52.142	82.903
sqrt.cov.CO.O3	144.0027	15.541	9.266	0.000	113.543	174.463
sqrt.cov.HC.O3	33.7115	10.067	3.349	0.001	13.981	53.442
sqrt.cov.NO2.O3	-1.0173	11.066	-0.092	0.927	-22.707	20.672
sqrt.var.O3	47.9515	5.222	9.182	0.000	37.716	58.187

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [ ]:
```
model.plot_diagnostics()
plt.show()
```

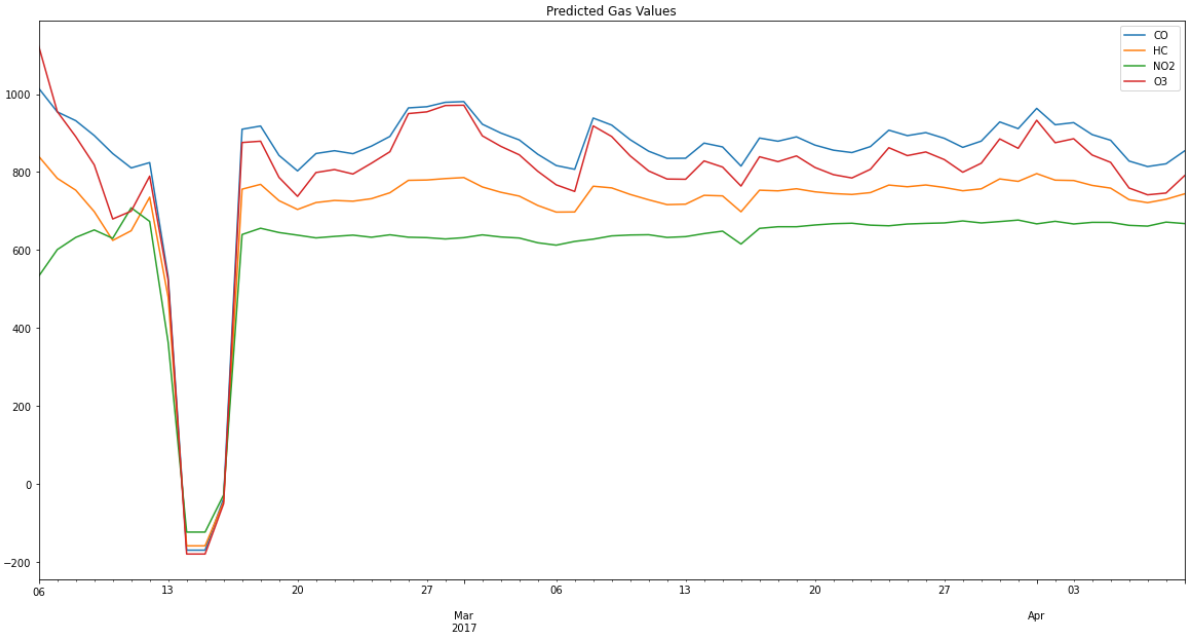


In [ ]:
```
# generate predictions
predictions = model.predict(start=X_test.index[0], end=X_test.index[-1], exog=X_tes
predictions.plot()
plt.title("Predicted Gas Values")
predictions = pd.melt(
    predictions, value_vars=["CO", "HC", "NO2", "O3"], value_name="Value"
)
predictions.index.rename("id", inplace=True)
predictions
```

Out[ ]:

id	variable	Value
0	CO	1014.665093
1	CO	954.341356
2	CO	931.782188
3	CO	893.754513
4	CO	847.458719
...	...	...
247	O3	824.917547
248	O3	759.103056
249	O3	741.655289
250	O3	746.205530
251	O3	790.700171

252 rows × 2 columns



In [ ]:
```
# making a directory if it does not exist
Path("./kaggle").mkdir(parents=True, exist_ok=True)
predictions["Value"].to_csv("./kaggle/varmax07.csv")
print("Exported predictions to .csv")
```

```
Exported predictions to .csv
```

Kaggle Scores

RMSE: 151.68211

Summary

We are able to create an forecasting model to predict air pollution in the air. With climate change being a constant threat to mankind's survival, we should start keeping the environment clean. However, the model still faces some limitations. It has only been trained and tested on an older set of data point from 2016 - 2018. As technology advances and human life advances, more pollution will appear, the trend and season of air pollution will change. For the model to be fully deployed, more data needs to be collected and use for prediction to make the model better.

Personal Reflection

The difficulty I found doing the forecasting task was being interpret the ACF and PACF plot. I feel that the points are not as accurate as the order collected using the cross validation techniques like expanding windows.