Forecasting Air Quality in New York City [...]

Author: Angela Kim

Contents [...]

Overview [...]

Business Problem [...]

Library Imports & Functions

First things first, we must import all relevant libraries.

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import itertools
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from math import sqrt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae

from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima_model import ARMA, ARIMA, ARIMAResults
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

executed in 1.88s, finished 01:04:26 2022-01-28

> In the following few blocks, I lay out functions that will be used throughout the notebook for EDA, metrics, and modeling.

In [2]:
```
# Function to check stationarity

def stationarity_check(df, pollutant):
    """
    Checks stationarity of time series with rolling statistics and Dickey-F

    Parameters:
    ----------
    df: time series DataFrame
    pollutant: str
    """
    rmean = df.rolling(window=12, center=False).mean()
    rstd = df.rolling(window=12, center=False).std()
    dftest = adfuller(df)

    # Plot rolling statistics against original
    fig = plt.figure(figsize=(20,6), dpi=300)
    plt.plot(df, color='slategrey', label='Original', alpha=0.8)
    plt.plot(rmean, color='mediumturquoise', label='Rolling Mean')
    plt.plot(rstd, color='magenta', label='Rolling Std')
    plt.title(f'Rolling Mean & Standard Deviation ({pollutant})')
    plt.xlabel('Year')
    plt.ylabel('AQI')
    plt.legend(loc='best')

    # Print Dickey-Fuller test results
    print(f'Results of Dickey-Fuller Test ({pollutant}): \n ---------------
    dfoutput = pd.Series(  dftest[0:4],
                           index=['Test Statistic',
                                  'p-value',
                                  '# of Lags Used',
                                  '# of Observations Used']  )
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

    return None


"""""""""""""


# Function to decompose time series

def decomposition_plot(df, pollutant):
    """
    Takes time series dataframe and decomposes it in order to observe trend

    Parameters:
    ----------
    df: time series DataFrame
    pollutant: str
    """
    decomposition = seasonal_decompose(df)
    trend = decomposition.trend
    seasonal = decomposition.seasonal
```

```
    residual = decomposition.resid
    plt.figure(figsize=(20,10))
    plt.subplot(511)
    plt.plot(df, label=f'Observed {pollutant}', color='darkorchid')
    plt.legend(loc='best')
    plt.subplot(512)
    plt.plot(trend, label=f'Trend {pollutant}', color='mediumvioletred')
    plt.legend(loc='best')
    plt.subplot(513)
    plt.plot(seasonal, label=f'Seasonality {pollutant}', color='gold')
    plt.legend(loc='best')
    plt.subplot(514)
    plt.plot(residual, label=f'Residuals {pollutant}', color='tomato')
    plt.legend(loc='best')
    plt.tight_layout()

    return None


"""""""""""""

# Function to perform a train-test-split

def train_test_split(df, ratio=0.85):
    train_size = int(len(df) * ratio)
    train, test = df[0:train_size], df[train_size:]
    return train, test


"""""""""""""


# Function for RMSE
# Can't be bothered to specify "squared=False" each time

def rmse(y_true, y_pred):
    """
    Computes Root Mean Squared Error (RMSE)
    """
    return mse(y_true, y_pred, squared=False)
```

executed in 55ms, finished 01:04:27 2022-01-28

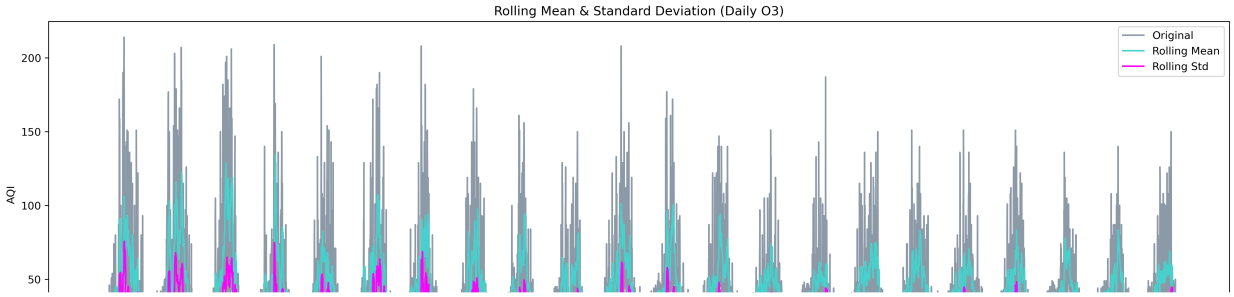
Data Preparation & Analysis [...]

Ozone

In [10]:
```
stationarity_check(dailyO3, 'Daily O3')
```

executed in 1.40s, finished 01:04:34 2022-01-28

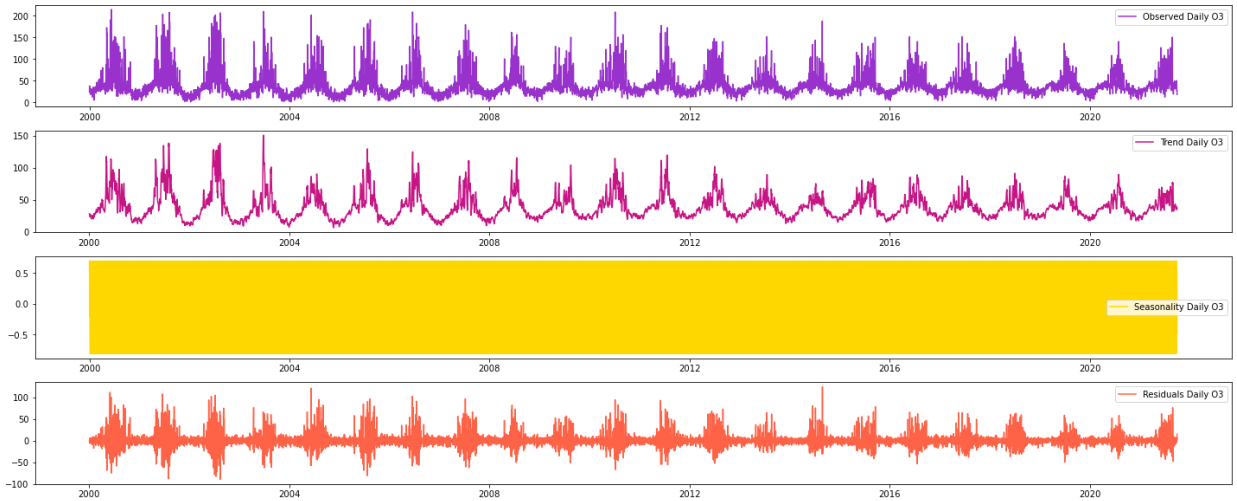
```
Results of Dickey-Fuller Test (Daily O3):
 ----------------------------
Test Statistic           -6.420012e+00
p-value                   1.801269e-08
# of Lags Used            2.900000e+01
# of Observations Used    7.914000e+03
Critical Value (1%)      -3.431177e+00
Critical Value (5%)      -2.861905e+00
Critical Value (10%)     -2.566964e+00
dtype: float64
```



The ADF test shows a p-value much smaller than 0.05, so we can reject the null hypothesis and say that `dailyO3` is stationary.

In [11]:
```
decomposition_plot(dailyO3, 'Daily O3')
```

executed in 1.34s, finished 01:04:35 2022-01-28

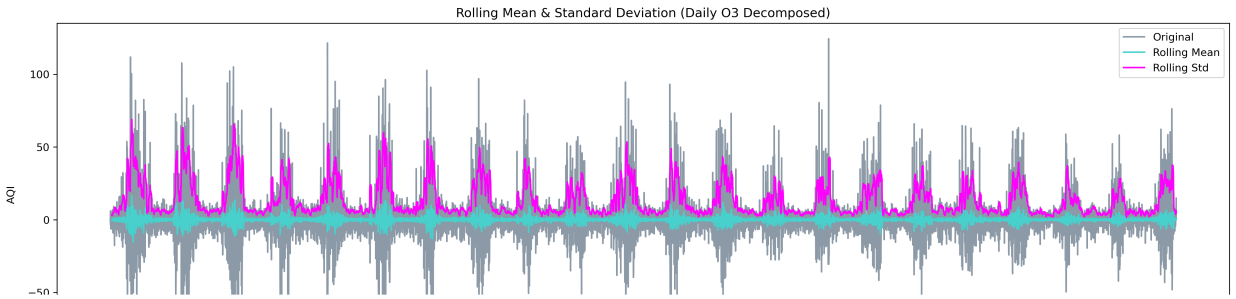


Residuals of `dailyO3` show seasonality.

In [12]:
```
decomposition = seasonal_decompose(dailyO3)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

dailyO3decomp = residual
dailyO3decomp.dropna(inplace=True)

stationarity_check(dailyO3decomp, 'Daily O3 Decomposed')
```

executed in 1.52s, finished 01:04:37 2022-01-28

```
Results of Dickey-Fuller Test (Daily O3 Decomposed):
 --------------------------
Test Statistic             -29.619233
p-value                      0.000000
# of Lags Used              36.000000
# of Observations Used    7901.000000
Critical Value (1%)         -3.431178
Critical Value (5%)         -2.861906
Critical Value (10%)        -2.566965
dtype: float64
```



In [13]:
```
mod_arma = ARMA(dailyO3decomp, order=(1,0), freq='D')
res_arma = mod_arma.fit()
res_arma.summary()
```

executed in 148ms, finished 01:04:37 2022-01-28

Out[13]:

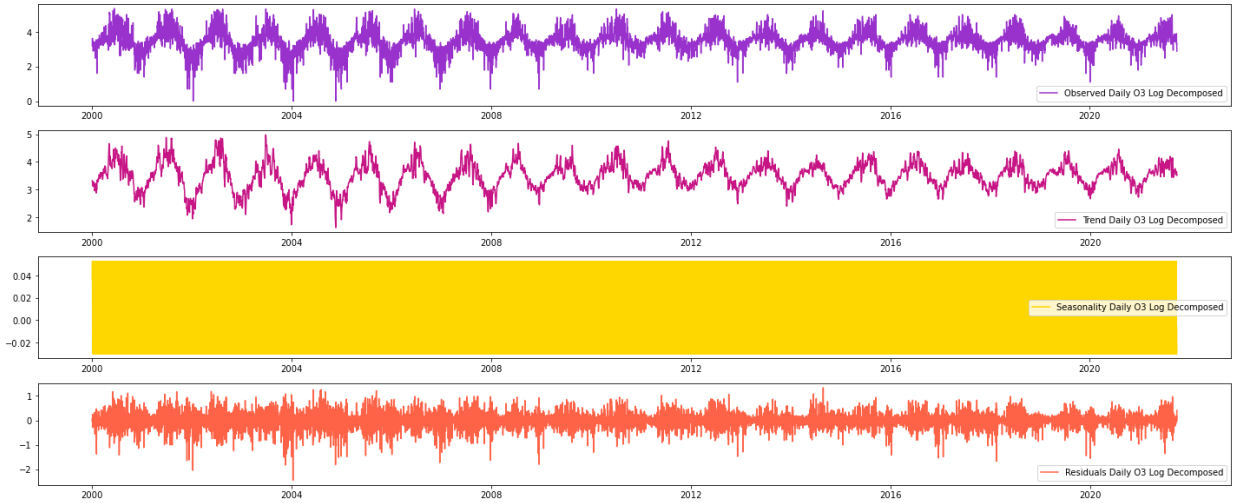
ARMA Model Results

Dep. Variable:	resid	**No. Observations:**	7938
Model:	ARMA(1, 0)	**Log Likelihood**	-33678.984
Method:	css-mle	**S.D. of innovations**	16.841
Date:	Fri, 28 Jan 2022	**AIC**	67363.967
Time:	01:04:37	**BIC**	67384.905
Sample:	01-04-2000	**HQIC**	67371.137
	- 09-27-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0019	0.216	0.009	0.993	-0.421	0.425
ar.L1.resid	0.1236	0.011	11.096	0.000	0.102	0.145

In [14]:
```
decomposition_plot(np.log(dailyO3), 'Daily O3 Log Decomposed')
```

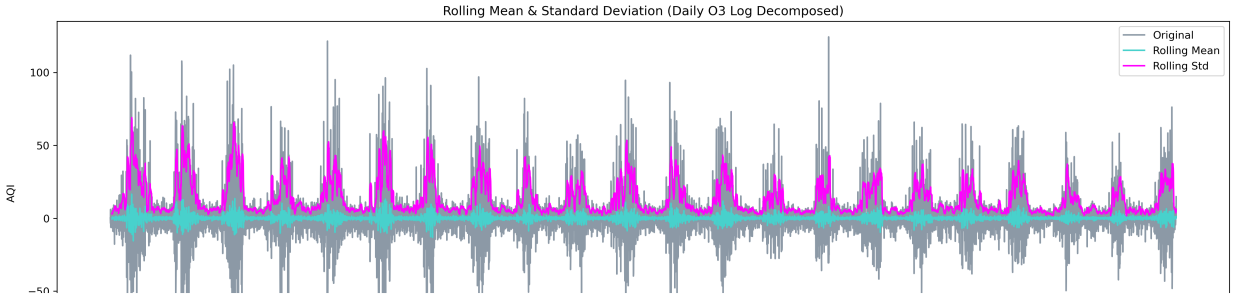
executed in 923ms, finished 01:04:38 2022-01-28



In [15]:
```
dailyO3logdecomp = residual
dailyO3logdecomp.dropna(inplace=True)

stationarity_check(dailyO3logdecomp, 'Daily O3 Log Decomposed')
```

executed in 1.91s, finished 01:04:40 2022-01-28

```
Results of Dickey-Fuller Test (Daily O3 Log Decomposed):
 ----------------------------
Test Statistic              -29.619233
p-value                       0.000000
# of Lags Used              36.000000
# of Observations Used    7901.000000
Critical Value (1%)          -3.431178
Critical Value (5%)          -2.861906
Critical Value (10%)         -2.566965
dtype: float64
```



In [16]:
```
mod_arma = ARMA(daily03logdecomp, order=(1,0), freq='D')
res_arma = mod_arma.fit()
res_arma.summary()
```

executed in 222ms, finished 01:04:40 2022-01-28

Out[16]:

ARMA Model Results

Dep. Variable:	resid	**No. Observations:**	7938
Model:	ARMA(1, 0)	**Log Likelihood**	-33678.984
Method:	css-mle	**S.D. of innovations**	16.841
Date:	Fri, 28 Jan 2022	**AIC**	67363.967
Time:	01:04:40	**BIC**	67384.905
Sample:	01-04-2000	**HQIC**	67371.137
	- 09-27-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0019	0.216	0.009	0.993	-0.421	0.425
ar.L1.resid	0.1236	0.011	11.096	0.000	0.102	0.145

In [17]:
```
mod_arima = ARIMA(daily03logdecomp, order=(1,1,0), freq='D')
res_arima = mod_arima.fit()
res_arima.summary()
```

executed in 463ms, finished 01:04:40 2022-01-28

Out[17]:

ARIMA Model Results

Dep. Variable:	D.resid	**No. Observations:**	7937
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-35717.985
Method:	css-mle	**S.D. of innovations**	21.786
Date:	Fri, 28 Jan 2022	**AIC**	71441.969
Time:	01:04:40	**BIC**	71462.907
Sample:	01-05-2000	**HQIC**	71449.139
	- 09-27-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0016	0.196	0.008	0.994	-0.383	0.387
ar.L1.D.resid	-0.2448	0.011	-22.499	0.000	-0.266	-0.224

In [18]:
```
mod_sarimax = SARIMAX(dailyO3decomp, freq='D')
res_sarimax = mod_sarimax.fit()
res_sarimax.summary()
```

executed in 356ms, finished 01:04:41 2022-01-28

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:524: ValueWarning: No frequency information was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```

Out[18]:

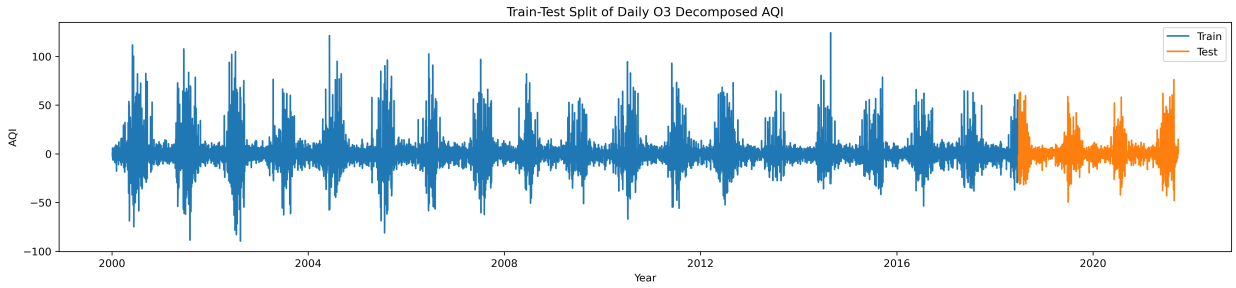
SARIMAX Results

Dep. Variable:	resid	**No. Observations:**	7938
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-33678.984
Date:	Fri, 28 Jan 2022	**AIC**	67361.967
Time:	01:04:41	**BIC**	67375.926
Sample:	01-04-2000	**HQIC**	67366.747
	- 09-27-2021		
Covariance Type:	opg		

coef	std err	z	P>|z|	[0.025	0.975]

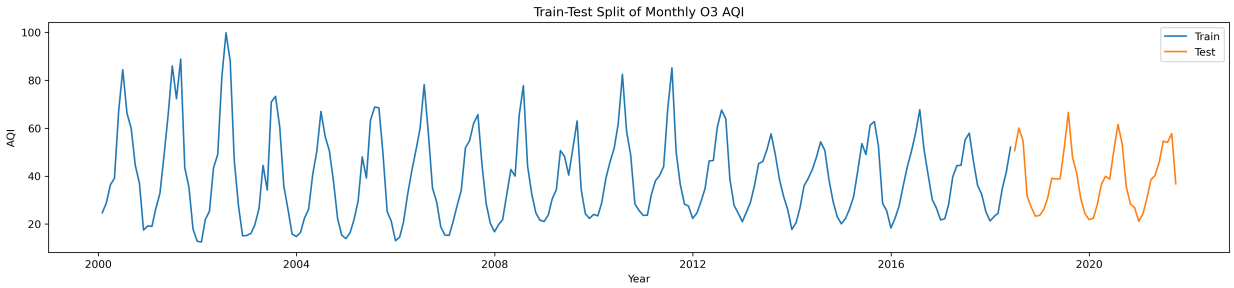
In [19]:
```
dailyO3_train, dailyO3_test = train_test_split(dailyO3decomp)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(dailyO3_train, label='Train')
ax.plot(dailyO3_test, label='Test')
ax.set_title('Train-Test Split of Daily O3 Decomposed AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 860ms, finished 01:04:42 2022-01-28



In [20]:
```
dailyO3_train, dailyO3_test = train_test_split(monthlyO3)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(dailyO3_train, label='Train')
ax.plot(dailyO3_test, label='Test')
ax.set_title('Train-Test Split of Monthly O3 AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 716ms, finished 01:04:42 2022-01-28



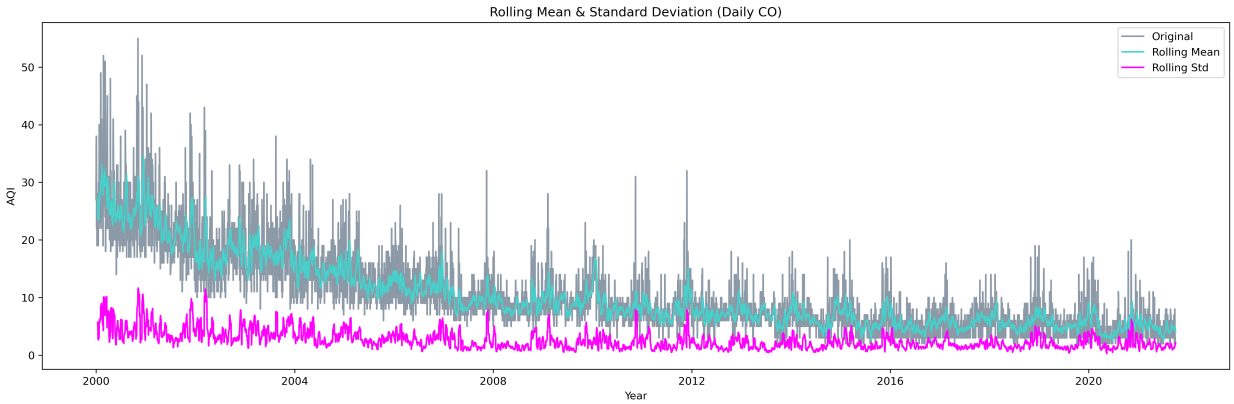
In [ ]:

Carbon Monoxide

In [21]:
```
stationarity_check(dailyCO, 'Daily CO')
```

executed in 1.56s, finished 01:04:44 2022-01-28

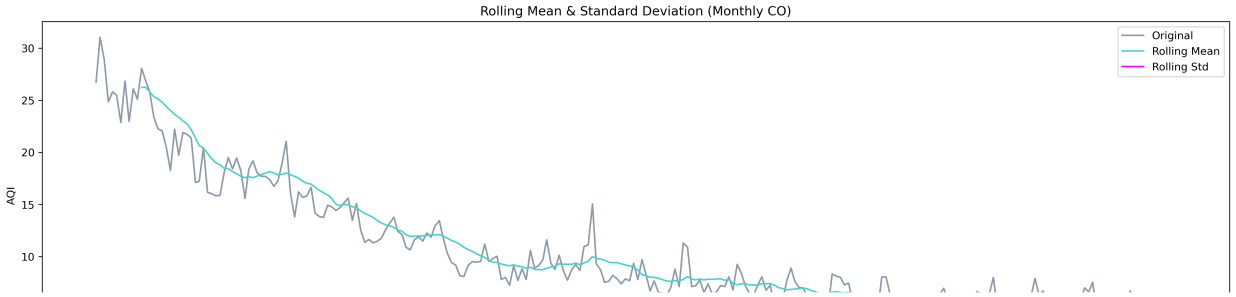
```
Results of Dickey-Fuller Test (Daily CO):
 ----------------------------
Test Statistic              -3.077750
p-value                      0.028244
# of Lags Used              36.000000
# of Observations Used    7908.000000
Critical Value (1%)         -3.431177
Critical Value (5%)         -2.861906
Critical Value (10%)        -2.566965
dtype: float64
```



In [22]:
```
stationarity_check(monthlyCO, 'Monthly CO')
```

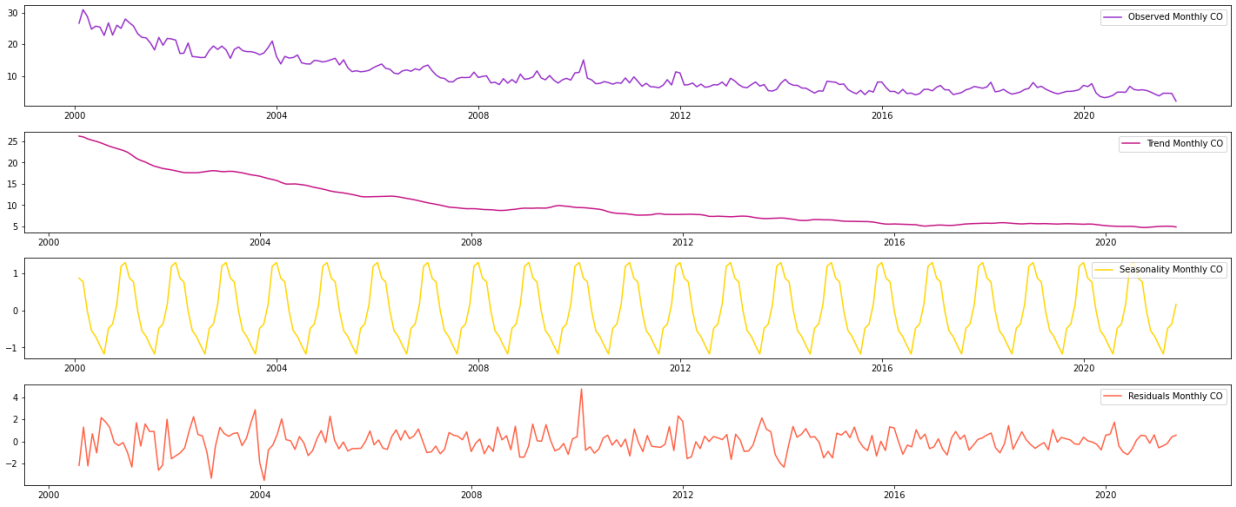
executed in 1.20s, finished 01:04:45 2022-01-28

```
Results of Dickey-Fuller Test (Monthly CO):
 ----------------------------
Test Statistic              -3.485152
p-value                      0.008377
# of Lags Used              15.000000
# of Observations Used     246.000000
Critical Value (1%)         -3.457215
Critical Value (5%)         -2.873362
Critical Value (10%)        -2.573070
dtype: float64
```



In [23]:
```
decomposition_plot(monthlyCO, 'Monthly CO')
```

executed in 1.17s, finished 01:04:46 2022-01-28



In [27]:
```
decomposition = seasonal_decompose(dailyCO)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

monthlyCOdecomp = residual

mod_arma = ARMA(monthlyCO, order=(1,0), freq='M')
res_arma = mod_arma.fit()
res_arma.summary()
```

executed in 90ms, finished 01:05:44 2022-01-28

Out[27]:

ARMA Model Results

Dep. Variable:	CO AQI	**No. Observations:**	262
Model:	ARMA(1, 0)	**Log Likelihood**	-478.561
Method:	css-mle	**S.D. of innovations**	1.494
Date:	Fri, 28 Jan 2022	**AIC**	963.121
Time:	01:05:44	**BIC**	973.826
Sample:	01-31-2000	**HQIC**	967.424
	- 10-31-2021		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	11.4595	4.017	2.853	0.004	3.586	19.333
ar.L1.CO AQI	0.9802	0.014	72.162	0.000	0.954	1.007

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	1.0202	+0.0000j	1.0202	0.0000

In [29]:
```
mod_arima = ARIMA(monthlyCO, order=(1,1,0), freq='M')
res_arima = mod_arima.fit()
res_arima.summary()
```

executed in 53ms, finished 01:05:54 2022-01-28

Out[29]:

ARIMA Model Results

Dep. Variable:	D.CO AQI	**No. Observations:**	261
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-464.065
Method:	css-mle	**S.D. of innovations**	1.432
Date:	Fri, 28 Jan 2022	**AIC**	934.130
Time:	01:05:54	**BIC**	944.823
Sample:	02-29-2000	**HQIC**	938.428
	- 10-31-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0966	0.069	-1.409	0.159	-0.231	0.038
ar.L1.D.CO AQI	-0.2945	0.060	-4.875	0.000	-0.413	-0.176

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-3.3952	+0.0000j	3.3952	0.5000

In [31]:
```
mod_sarimax = SARIMAX(monthlyCO, freq='M')
res_sarimax = mod_sarimax.fit()
res_sarimax.summary()
```

executed in 94ms, finished 01:06:09 2022-01-28

Out[31]:

SARIMAX Results

Dep. Variable:	CO AQI	**No. Observations:**	262
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-479.931
Date:	Fri, 28 Jan 2022	**AIC**	963.861
Time:	01:06:09	**BIC**	970.998
Sample:	01-31-2000	**HQIC**	966.730
	- 10-31-2021		
Covariance Type:	opg		

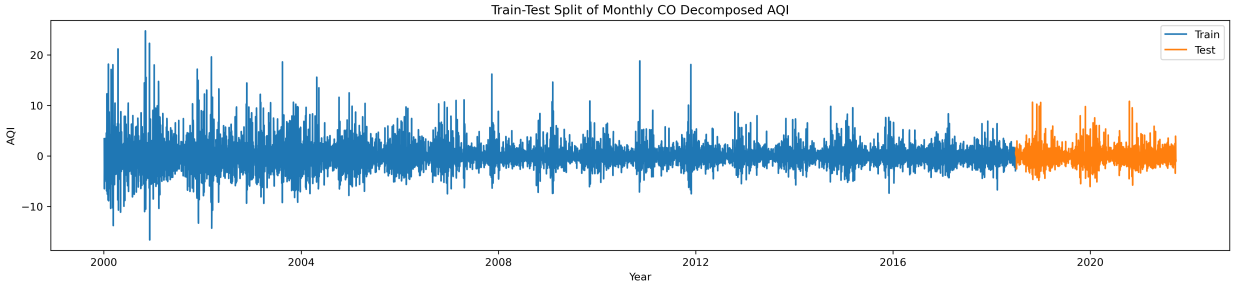
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9953	0.004	274.039	0.000	0.988	1.002
sigma2	2.2431	0.146	15.331	0.000	1.956	2.530

Ljung-Box (L1) (Q):	18.22	**Jarque-Bera (JB):**	27.85
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.32	**Skew:**	-0.09
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.59

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

In [32]:
```
CO_train, CO_test = train_test_split(monthlyCOdecomp)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(CO_train, label='Train')
ax.plot(CO_test, label='Test')
ax.set_title('Train-Test Split of Monthly CO Decomposed AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 607ms, finished 01:06:14 2022-01-28

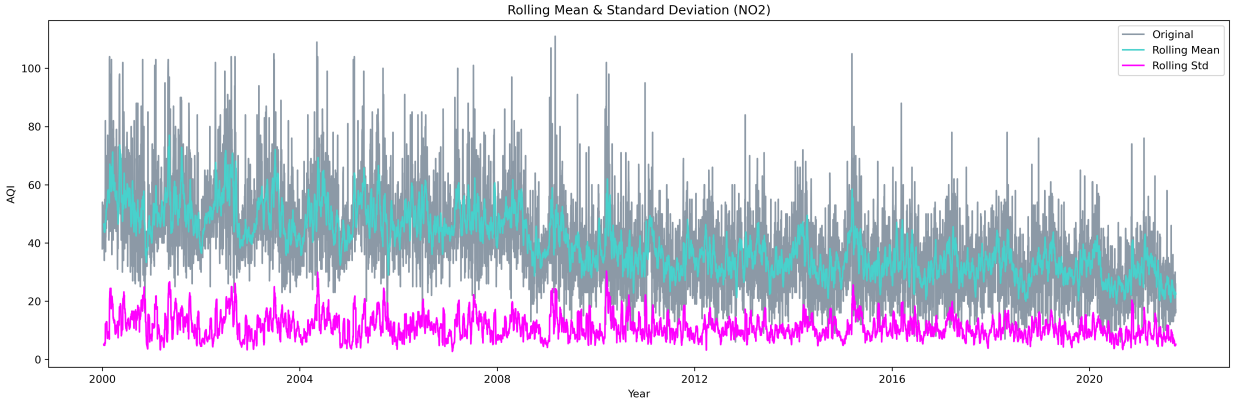


Nitrogen Dioxide

In [33]:
```
stationarity_check(dailyNO2, 'NO2')
```

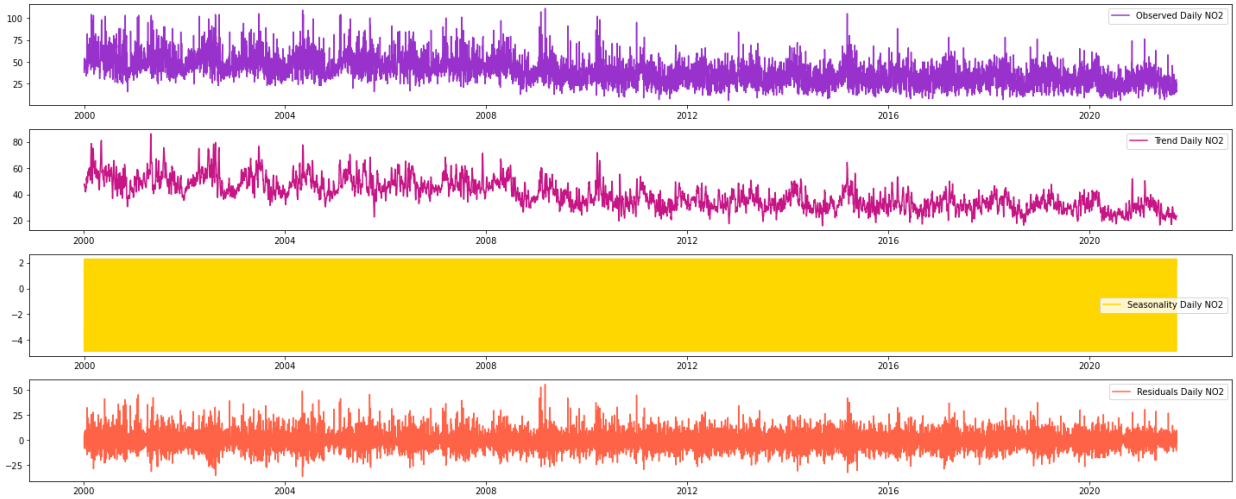
executed in 1.85s, finished 01:06:17 2022-01-28

```
Results of Dickey-Fuller Test (NO2):
 ----------------------------
Test Statistic              -4.521453
p-value                      0.000180
# of Lags Used              34.000000
# of Observations Used    7909.000000
Critical Value (1%)         -3.431177
Critical Value (5%)         -2.861906
Critical Value (10%)        -2.566965
dtype: float64
```



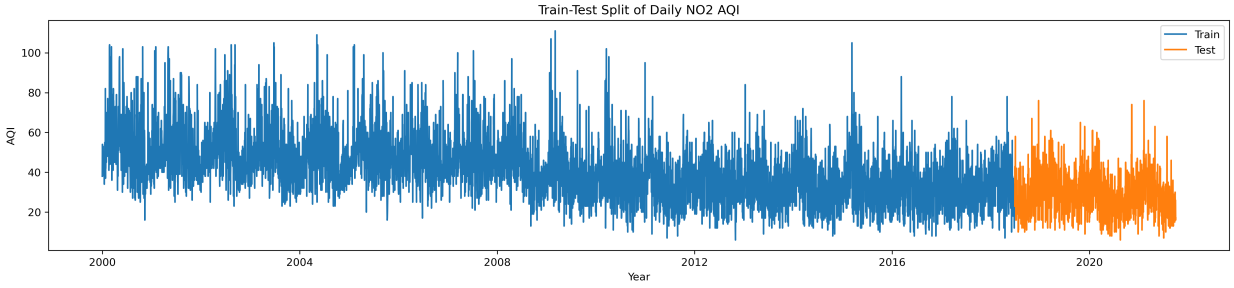
In [34]:
```
decomposition_plot(dailyNO2, 'Daily NO2')
```

executed in 1.07s, finished 01:06:18 2022-01-28



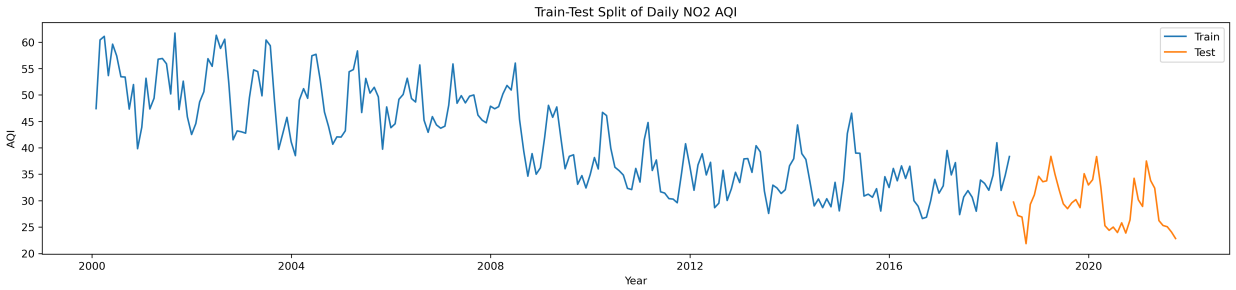
In [35]:
```
NO2_train, NO2_test = train_test_split(dailyNO2)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(NO2_train, label='Train')
ax.plot(NO2_test, label='Test')
ax.set_title('Train-Test Split of Daily NO2 AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 867ms, finished 01:06:19 2022-01-28



In [36]:
```
NO2_train, NO2_test = train_test_split(monthlyNO2)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(NO2_train, label='Train')
ax.plot(NO2_test, label='Test')
ax.set_title('Train-Test Split of Daily NO2 AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 866ms, finished 01:06:20 2022-01-28

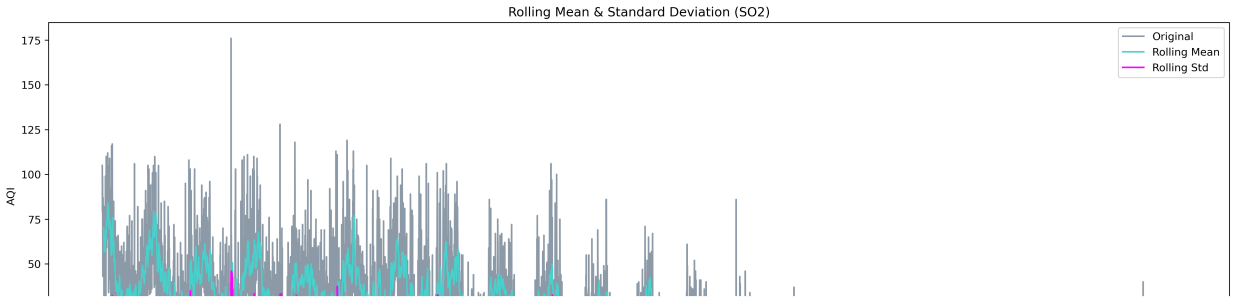


Sulfur Dioxide

In [37]:
```
stationarity_check(dailySO2, 'SO2')
```

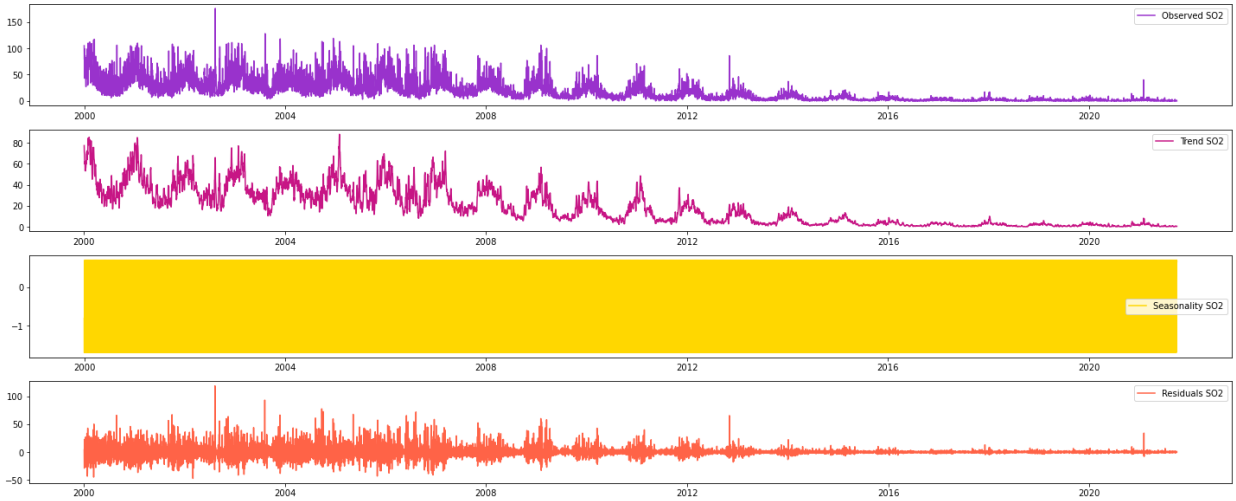
executed in 1.48s, finished 01:06:22 2022-01-28

```
Results of Dickey-Fuller Test (SO2):
 ----------------------------
Test Statistic              -3.791089
p-value                      0.003000
# of Lags Used              33.000000
# of Observations Used    7910.000000
Critical Value (1%)         -3.431177
Critical Value (5%)         -2.861905
Critical Value (10%)        -2.566965
dtype: float64
```



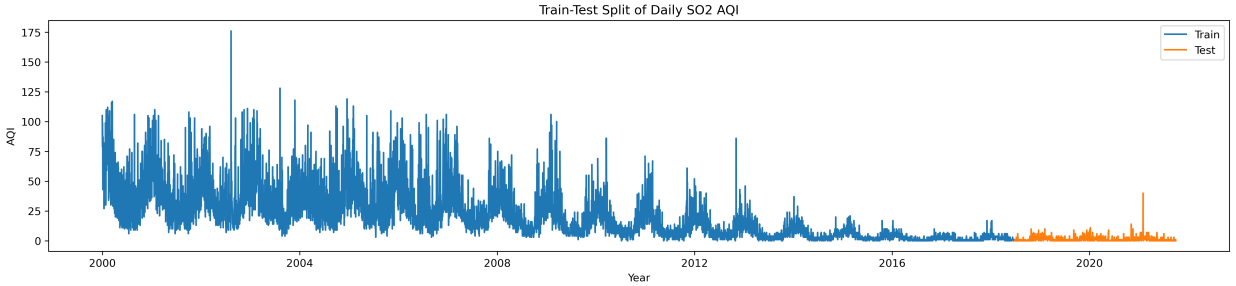
In [38]:
```
decomposition_plot(dailySO2, 'SO2')
```

executed in 928ms, finished 01:06:23 2022-01-28



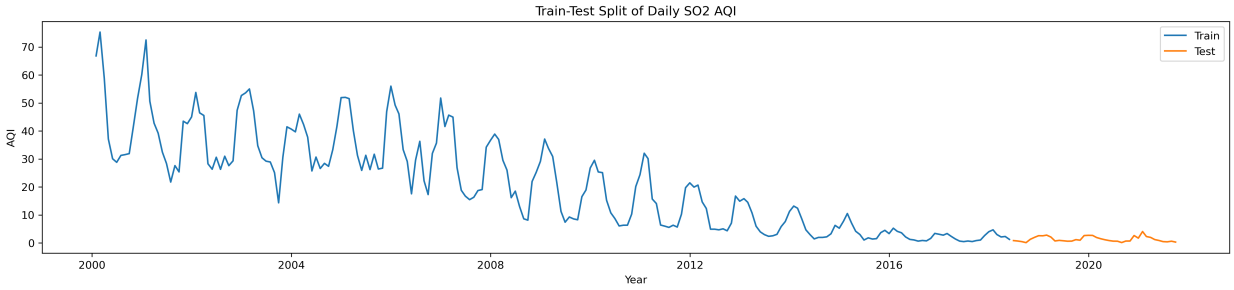
In [39]:
```
SO2_train, SO2_test = train_test_split(dailySO2)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(SO2_train, label='Train')
ax.plot(SO2_test, label='Test')
ax.set_title('Train-Test Split of Daily SO2 AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 701ms, finished 01:06:23 2022-01-28



In [40]:
```
SO2_train, SO2_test = train_test_split(monthlySO2)

fig, ax = plt.subplots(figsize=(20,4), dpi=300)
ax.plot(SO2_train, label='Train')
ax.plot(SO2_test, label='Test')
ax.set_title('Train-Test Split of Daily SO2 AQI');
ax.set_xlabel('Year')
ax.set_ylabel('AQI')
plt.legend();
```

executed in 576ms, finished 01:06:24 2022-01-28



Modeling

In [42]:
```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
```

executed in 6ms, finished 01:07:54 2022-01-28

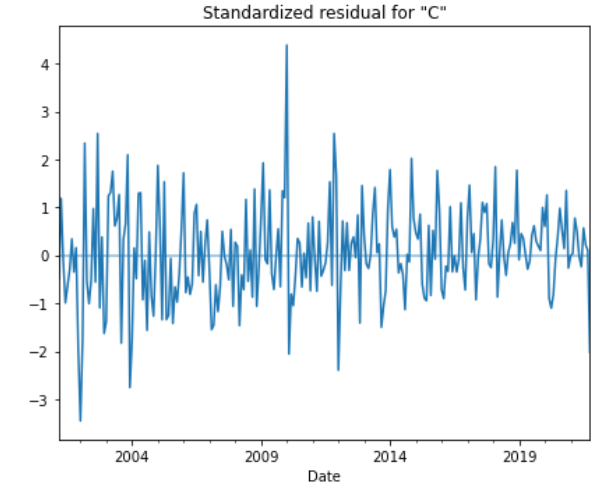
In [43]:
```
# Generate all combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2],12) for x in list(itertools.product(p, d,
```

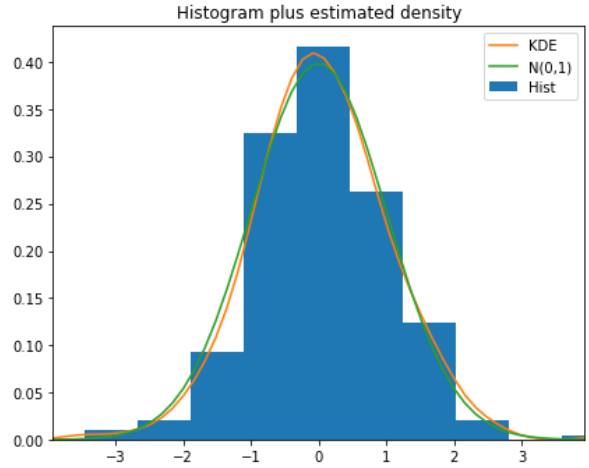
executed in 19ms, finished 01:07:54 2022-01-28

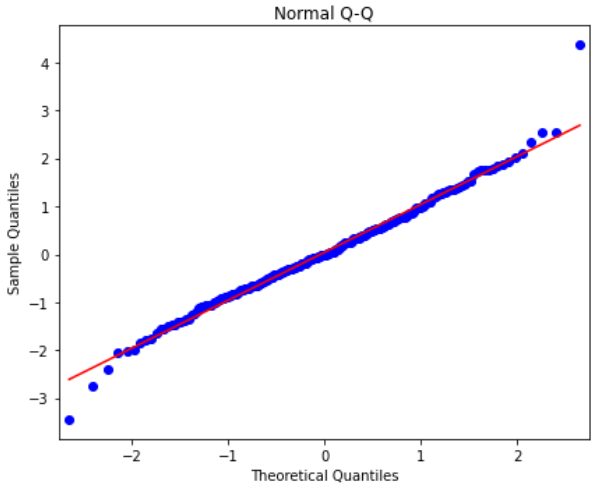
In [44]:
```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = SARIMAX(monthlyCO,
                          order=param,
                          seasonal_order=param_seasonal,
                          enforce_stationarity=False,
                          enforce_invertibility=False)

            results = mod.fit()

            # print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, r
        except:
            continue

mod = SARIMAX(monthlyCO,
              order=(1,0,1),
              seasonal_order=(1, 0, 1, 12),
              enforce_stationarity=False,
              enforce_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])

results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

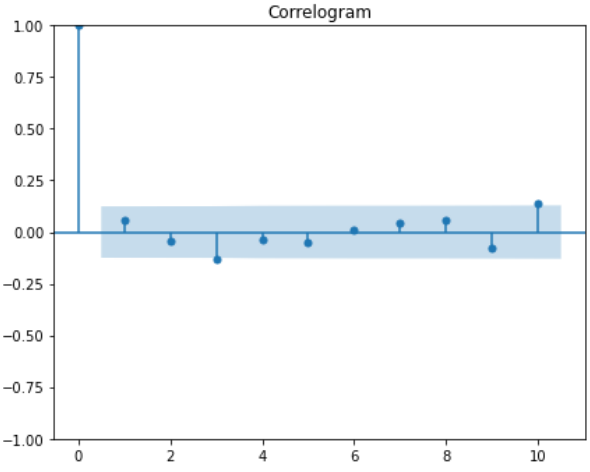
executed in 11.9s, finished 01:08:06 2022-01-28

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/bas
e/model.py:566: ConvergenceWarning: Maximum Likelihood optimization faile
d to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "

==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
ar.L1          0.8759      0.042     20.955      0.000       0.794
0.958
ma.L1         -0.5234      0.073     -7.209      0.000      -0.666      -
0.381
ar.S.L12       0.9061      0.010     90.598      0.000       0.886
0.926
ma.S.L12      -0.7427      0.055    -13.441      0.000      -0.851      -
0.634
sigma2         1.4828      0.105     14.113      0.000       1.277
1.689
==============================================================================
```









In [46]:
```
monthlyCO_log = np.log(monthlyCO)
```

executed in 5ms, finished 01:08:30 2022-01-28

In [47]:
```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = SARIMAX(monthlyCO_log,
                          order=param,
                          seasonal_order=param_seasonal,
                          enforce_stationarity=False,
                          enforce_invertibility=False)

            results = mod.fit()

            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, res
        except:
            continue

mod = SARIMAX(monthlyCO_log,
              order=(1,0,1),
              seasonal_order=(1, 0, 1, 12),
              enforce_stationarity=False,
              enforce_invertibility=False)
```

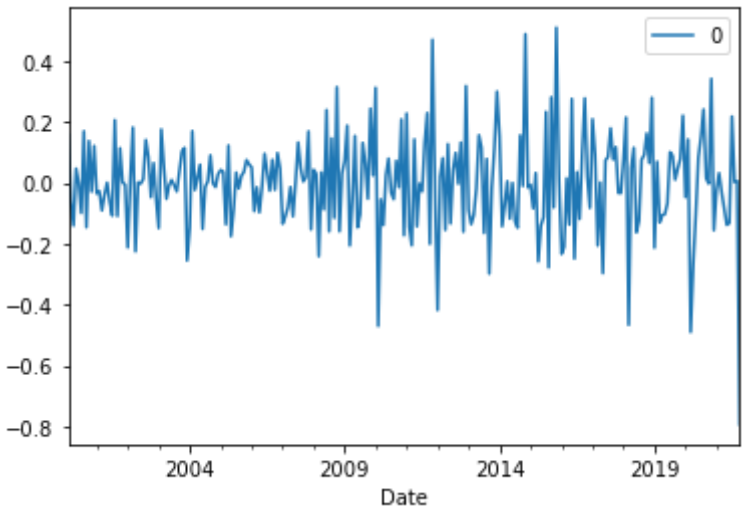
executed in 15.2s, finished 01:08:46 2022-01-28

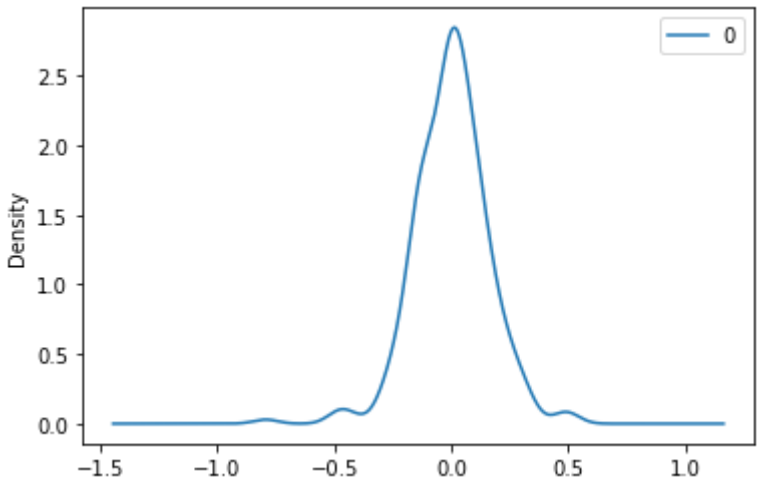
```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1165.7600595496024
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:817.4177527431933
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:-86.64817594588106
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:-84.62244571807231
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:-149.50231659299766
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:-207.73117861563975
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:-81.14731134557157
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:-111.64651254482528
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:835.8363642495663
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:545.9558014087152
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:-131.51176550747624
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:-131.35977230637826

/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/bas
e/model.py:566: ConvergenceWarning: Maximum Likelihood optimization faile
d to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [48]:
```
model = ARIMA(monthlyCO_log, order=(0,1,0))
model_fit = model.fit()
# summary of fit model
print(model_fit.summary())
# line plot of residuals
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
# density plot of residuals
residuals.plot(kind='kde')
plt.show()
# summary stats of residuals
print(residuals.describe())
```

executed in 811ms, finished 01:08:47 2022-01-28

```
                             ARIMA Model Results
==============================================================================
=====
Dep. Variable:                D.CO AQI   No. Observations:
261
Model:                 ARIMA(0, 1, 0)   Log Likelihood                  11
0.669
Method:                           css   S.D. of innovations
0.158
Date:                Fri, 28 Jan 2022   AIC                            -21
7.338
Time:                        01:08:46   BIC                            -21
0.209
Sample:                    02-29-2000   HQIC                           -21
4.473
                         - 10-31-2021
==============================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const         -0.0099      0.010     -1.014      0.311      -0.029
0.009
==============================================================================
=====
```



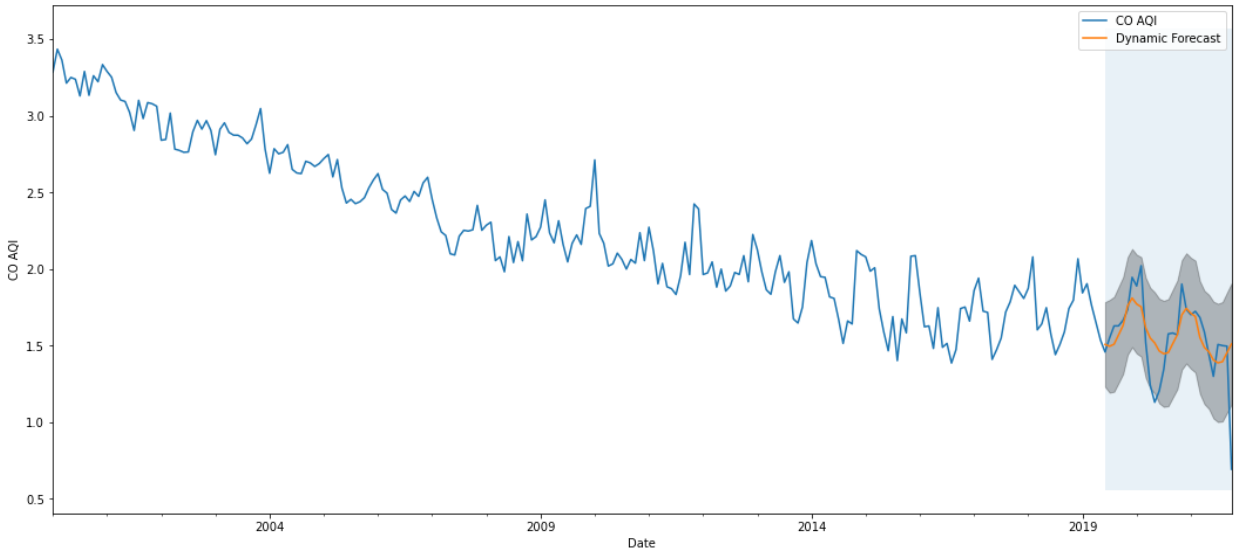


	0
count	2.610000e+02
mean	2.276379e-09
std	1.586525e-01
min	-7.935600e-01
25%	-1.028973e-01
50%	1.663533e-03
75%	8.892361e-02
max	5.102421e-01

In [49]:
```
pred_dynamic = results.get_prediction(start=pd.to_datetime('2019-06-30 00:0
pred_dynamic_ci = pred_dynamic.conf_int()

plt.figure(dpi=300)
ax = monthlyCO_log['2000':].plot(label='Observed', figsize=(18, 8))
pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_ci.index,
                pred_dynamic_ci.iloc[:, 0],
                pred_dynamic_ci.iloc[:, 1], color='k', alpha=.25)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2019-06-30 00:00:00'), mont
                 alpha=.1, zorder=-1)

ax.set_xlabel('Date')
ax.set_ylabel('CO AQI')

plt.legend()
plt.show();
```

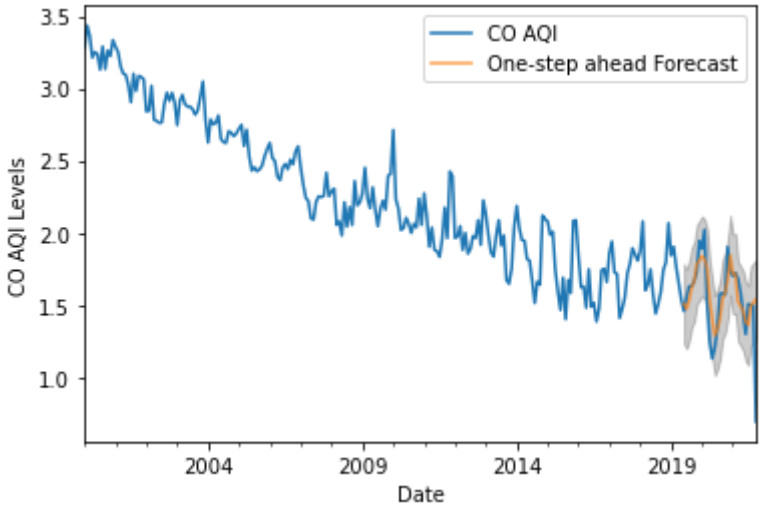
executed in 349ms, finished 01:08:47 2022-01-28

```
<Figure size 1800x1200 with 0 Axes>
```



In [50]:
```
pred = results.get_prediction(start=pd.to_datetime('2019-06-30 00:00:00'),
pred_ci = pred.conf_int()
ax = monthlyCO_log['2000':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('CO AQI Levels')
plt.legend()
plt.show()
```

executed in 364ms, finished 01:08:47 2022-01-28



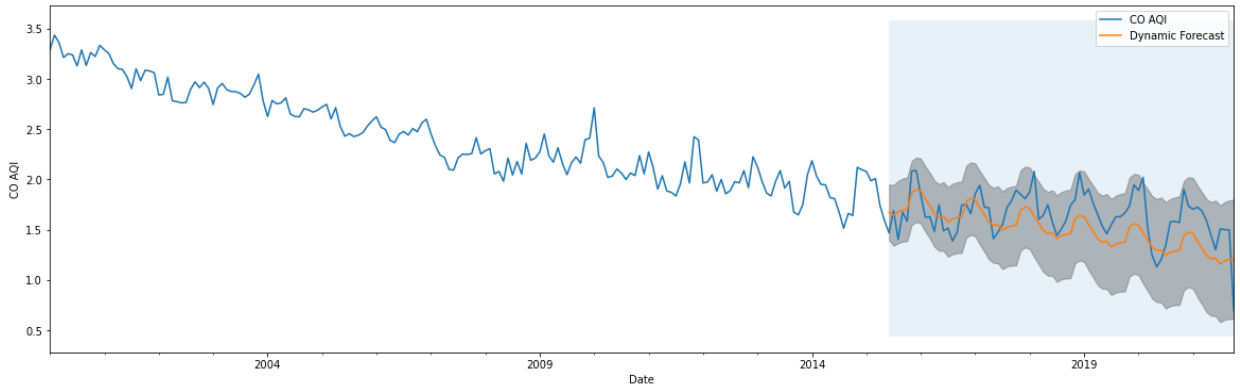
In [51]:
```
monthlyCO_forecasted = np.exp(pred.predicted_mean)
monthlyCO_real = np.exp(monthlyCO_log['2019-06-30 00:00:00':])


# Compute the mse and rmse
rmse(monthlyCO_real, monthlyCO_forecasted)
```

executed in 15ms, finished 01:08:47 2022-01-28

Out[51]: 0.7975383620703952

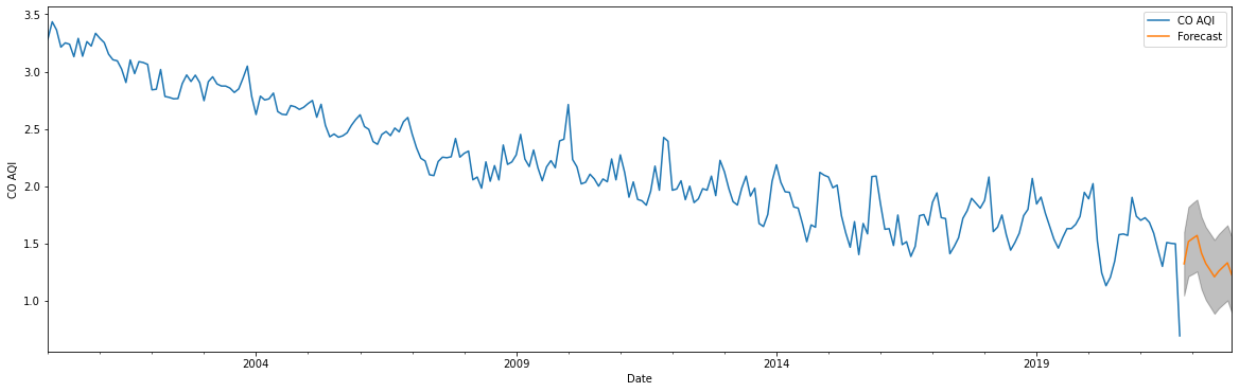
In [52]:
```
pred_dynamic = results.get_prediction(start=pd.to_datetime('2015-06-30 00:0
pred_dynamic_ci = pred_dynamic.conf_int()

ax = monthlyCO_log['2000':].plot(label='Observed', figsize=(20, 6))
pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_ci.index,
                pred_dynamic_ci.iloc[:, 0],
                pred_dynamic_ci.iloc[:, 1], color='k', alpha=.25)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2015-06-30 00:00:00'), mont
                 alpha=.1, zorder=-1)

ax.set_xlabel('Date')
ax.set_ylabel('CO AQI')

plt.legend()
plt.show()
```

executed in 369ms, finished 01:08:48 2022-01-28



In [53]:
```
# Get forecast 500 steps ahead in future
pred_uc = results.get_forecast(steps=12)

# Get confidence intervals of forecasts
pred_ci = pred_uc.conf_int()

ax = monthlyCO_log.plot(label='Observed', figsize=(20, 6))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('CO AQI')

plt.legend()
plt.show()


np.exp(pred_uc.predicted_mean)
```

executed in 464ms, finished 01:08:48 2022-01-28



Out[53]:
```
2021-11-30    3.750554
2021-12-31    4.553930
2022-01-31    4.680725
2022-02-28    4.802104
2022-03-31    4.122492
2022-04-30    3.763280
2022-05-31    3.551853
2022-06-30    3.347967
2022-07-31    3.518229
2022-08-31    3.649638
2022-09-30    3.782711
2022-10-31    3.408476
Freq: M, Name: predicted_mean, dtype: float64
```

Conclusion

In conclusion, my SARIMA model forecasted air quality in New York City quite well and could even be used in shaping government policy on public health. I would recommend implementing measures to decrease the presence of air pollutants,

especially ozone and nitrogen dioxide, as there hasn't been much decrease from 2000.

Next Steps

Given more time and resources, I would like to explore beyond New York City, modeling for other cities and even seeing how cities compare to suburban or rural areas. Another pollutant I'd like to consider is particulate matter. In terms of modeling, it would be interesting to see how well a recurrent neural network would perform.

Credits

https://www.kaggle.com/victoraqiao/time-series-forecasting-arima-co-aqi (https://www.kaggle.com/victoraqiao/time-series-forecasting-arima-co-aqi)