Untitled40

December 19, 2017

In [52]:
```
import warnings
import itertools
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

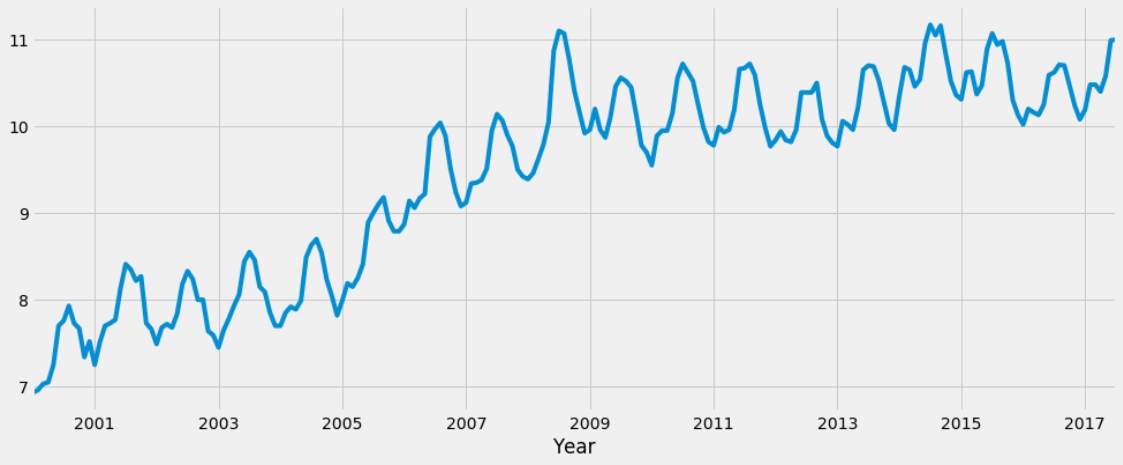
In [84]:
```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot

def parser(x):
        return datetime.strptime( x, '%Y-%m')
y = read_csv('loadnew4.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date
print(y)
```

```
Year
2000-01-01     6.93
2000-02-01     6.96
2000-03-01     7.03
2000-04-01     7.05
2000-05-01     7.25
2000-06-01     7.70
2000-07-01     7.76
2000-08-01     7.93
2000-09-01     7.73
2000-10-01     7.67
2000-11-01     7.34
2000-12-01     7.52
2001-01-01     7.25
2001-02-01     7.51
2001-03-01     7.70
2001-04-01     7.73
2001-05-01     7.77
2001-06-01     8.13
2001-07-01     8.41
2001-08-01     8.35
```

```
2001-09-01     8.22
2001-10-01     8.27
2001-11-01     7.73
2001-12-01     7.66
2002-01-01     7.49
2002-02-01     7.68
2002-03-01     7.72
2002-04-01     7.68
2002-05-01     7.84
2002-06-01     8.18
               ...
2015-02-01    10.62
2015-03-01    10.63
2015-04-01    10.37
2015-05-01    10.47
2015-06-01    10.89
2015-07-01    11.07
2015-08-01    10.94
2015-09-01    10.98
2015-10-01    10.73
2015-11-01    10.30
2015-12-01    10.13
2016-01-01    10.02
2016-02-01    10.20
2016-03-01    10.16
2016-04-01    10.13
2016-05-01    10.25
2016-06-01    10.59
2016-07-01    10.62
2016-08-01    10.71
2016-09-01    10.70
2016-10-01    10.47
2016-11-01    10.24
2016-12-01    10.08
2017-01-01    10.19
2017-02-01    10.48
2017-03-01    10.48
2017-04-01    10.40
2017-05-01    10.58
2017-06-01    10.99
2017-07-01    11.00
Name: Price, Length: 211, dtype: float64
```

```
In [85]: y.plot(figsize=(15, 6))
         plt.show()
```



In [86]:
```
# Define the p, d and q parameters to take any value between 0 and 2
d = q = range(0, 2)
p = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different 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, q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [87]:
```
warnings.filterwarnings("ignore") # specify to ignore warning messages

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
```

```
                                            enforce_invertibility=False)

                results = mod.fit()

                print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
            except:
                continue
```

```
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1234.0716220991794
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:167.50153855941147
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:148.26412403234156
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:149.6841134721401
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:166.25554461269155
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:142.9701440571548
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1261.5490973620517
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:967.5720629696922
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:-11.13725787546796
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:-14.433073071614587
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:-39.55257975401456
ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:-38.14043074789042
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:-16.350338106627092
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:-13.5661461127333
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:-128.2275582567525
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:-287.49645748961996
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:-247.25572085838192
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:-297.5890703274779
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:-268.775994154271
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:-281.5321610941657
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:-64.40469650183216
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:-142.34515062425808
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:-226.3419327241349
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:-284.37036246625394
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:-245.8592368795497
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:-294.8410756573557
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:-266.8407359285701
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:-278.9275882931724
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:-36.138276820993156
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:-126.64097284226901
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:-232.92560784354004
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:-289.6932201247929
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:-245.2564679317999
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:-295.83389305555755
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:-270.2803793404081
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:-283.3655529750232
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:-64.05544163691684
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:-141.1752944367883
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:-232.43731460480205
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:-286.12462052668116
```

```
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:-243.86012308537312
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:-295.60345102857616
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:-268.28156677206846
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:-279.57336705438433
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:-67.92221134405483
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:-144.65485185916225
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:-227.70358464817525
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:-286.4384718863247
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:-246.99801647018631
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:-296.22132555689427
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:-264.4963016321057
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:-279.96438929585514
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:-65.8235112979799
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:-140.80540154225758
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:-224.32602120804762
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:-281.7843324031063
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:-245.6499398725956
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:-293.66680083794273
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:-262.42113141975744
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:-279.81884290751526
```

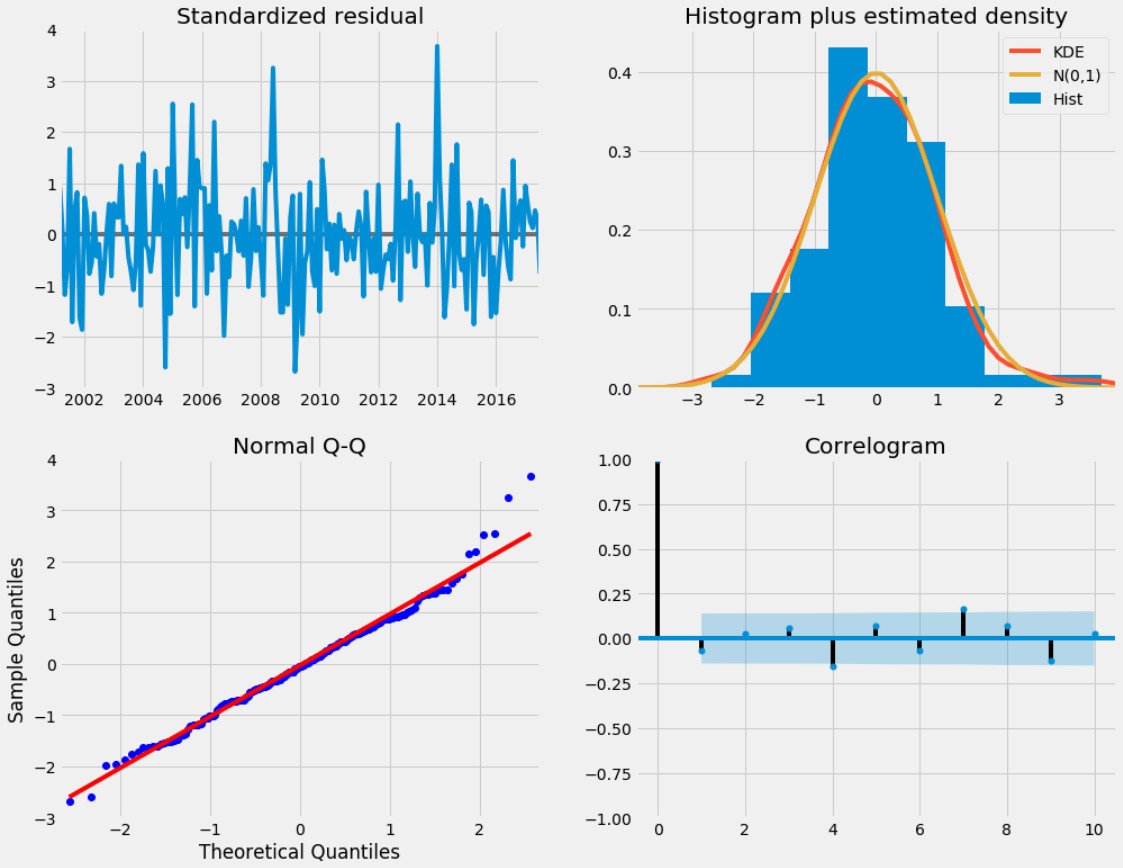
In [132]:
```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(0, 1, 0),
                                seasonal_order=(1, 0, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()

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

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12       0.9810      0.017     58.239      0.000       0.948       1.014
ma.S.L12      -0.6712      0.068     -9.890      0.000      -0.804      -0.538
sigma2         0.0121      0.001     11.641      0.000       0.010       0.014
==============================================================================
```

In [133]:
```
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```



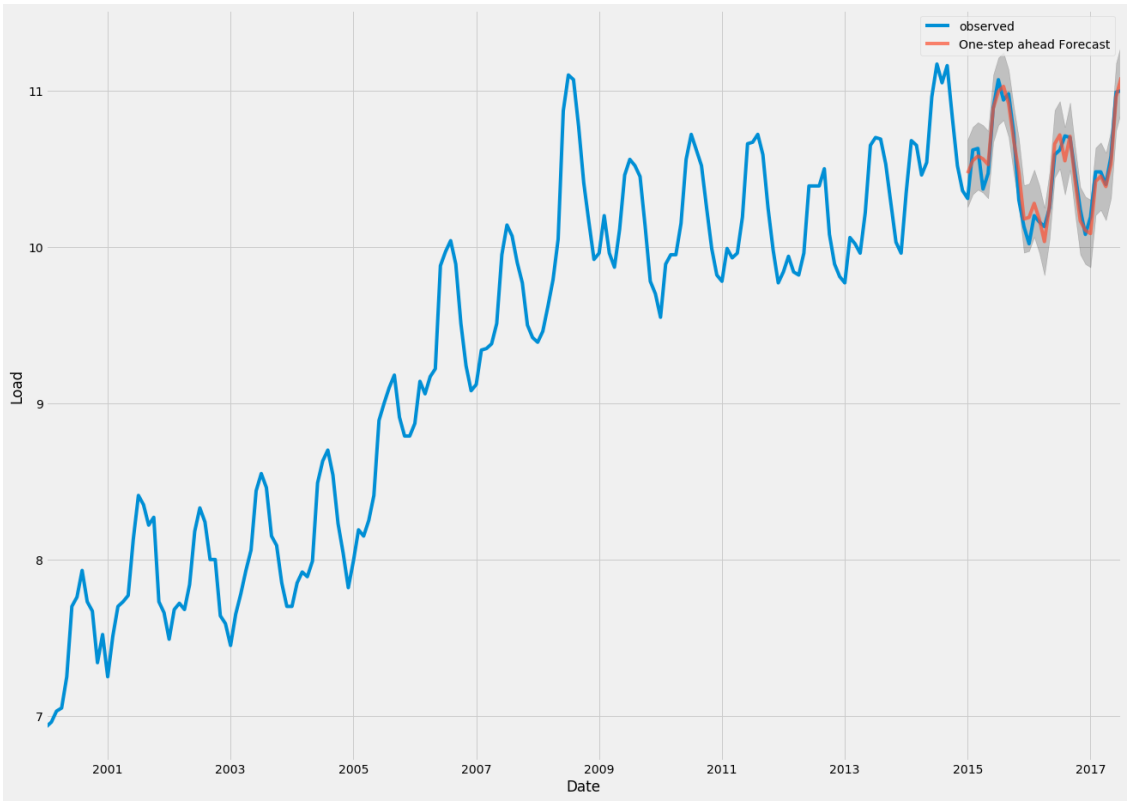

```
In [157]: pred = results.get_prediction(start=pd.to_datetime('2015-01-01'), dynamic=False)
          pred_ci = pred.conf_int()

In [158]: ax = y['2000':].plot(label='observed', figsize=(20, 15))
          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('Load')
          plt.legend()

          plt.show()
```




```
In [136]: y_forecasted = pred.predicted_mean
          y_truth = y['2016-01-01':]

          # Compute the mean square error
          mse = ((y_forecasted - y_truth) ** 2).mean()
          print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 0.01
```

```
In [137]: pred_dynamic = results.get_prediction(start=pd.to_datetime('2016-01-01'), dynamic=True
          pred_dynamic_ci = pred_dynamic.conf_int()
```

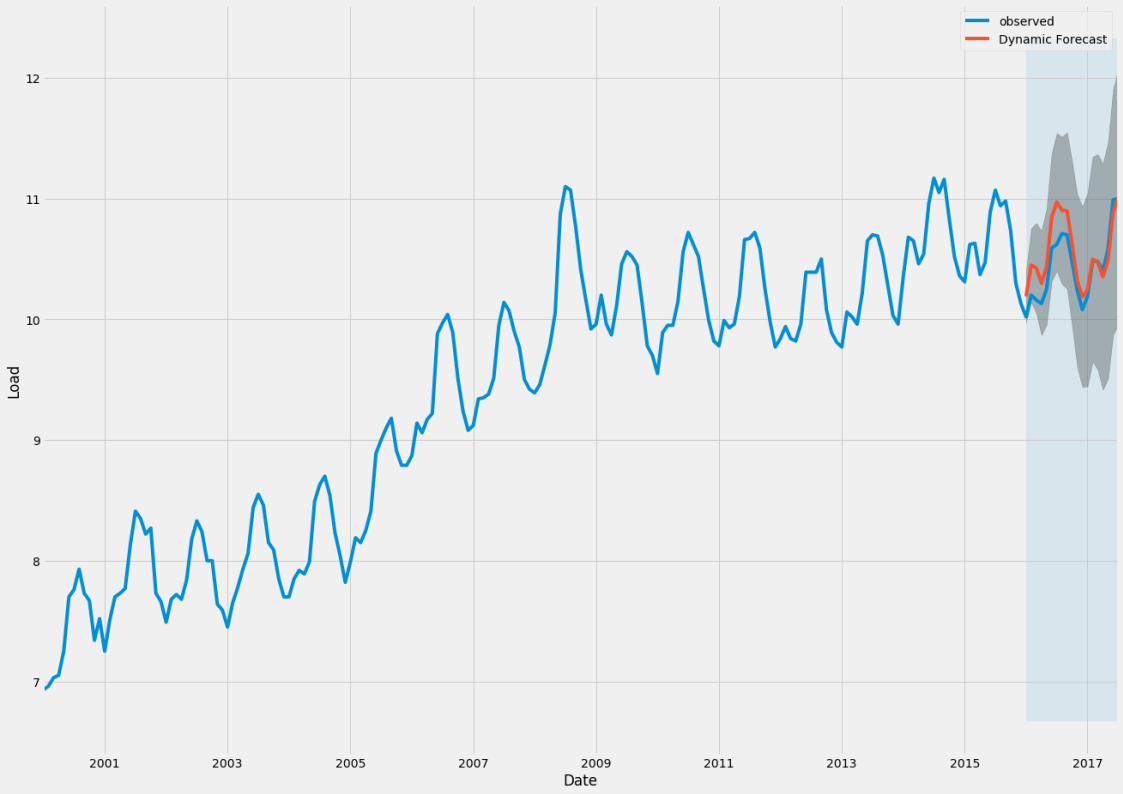
```
In [138]: ax = y['2000':].plot(label='observed', figsize=(20, 15))
          pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

          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('2016-01-01'), y.index[-1],
                           alpha=.1, zorder=-1)
```

```
ax.set_xlabel('Date')
ax.set_ylabel('Load')

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



In [139]:
```
# Extract the predicted and true values of our time series
y_forecasted = pred_dynamic.predicted_mean
y_truth = y['2016-01-01':]

# Compute the mean square error
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 0.03
```

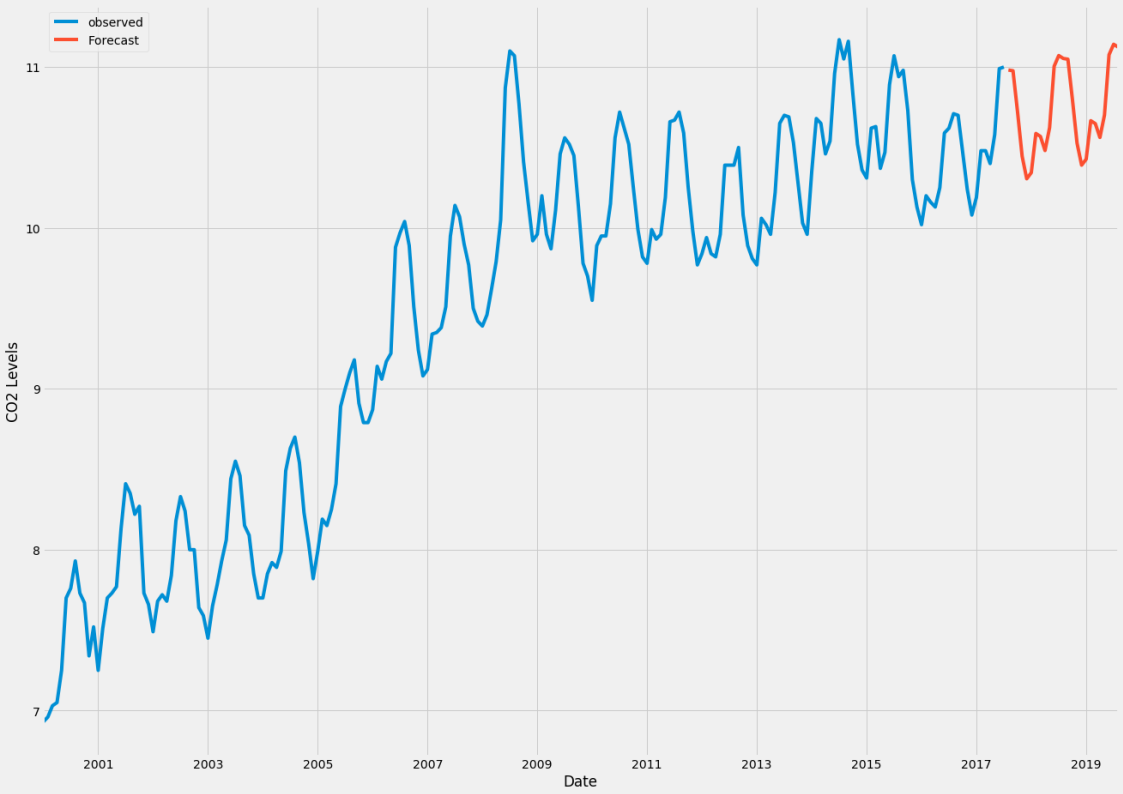
In [155]:
```
# Get forecast 500 steps ahead in future
pred_uc = results.get_forecast(steps=25)

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

In [156]:
```
ax = y.plot(label='observed', figsize=(20, 15))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')

ax.set_xlabel('Date')
ax.set_ylabel('CO2 Levels')

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



In [ ]: