Modeling and Predicting Monthly CO_2 Levels – Manua Loa Observatory

Objective

The purpose of this report, and overall study, is to attempt to generate a model, based on a time series of monthly CO_2 measurements from the Manua Loa Observatory in Hawaii, to helpfully predict the time-based behavior of atmospheric CO_2 concentrations. We will set forth to model the measurements and provide predictions, using multiple different time series techniques, in an attempt to help scientists, or enthusiastic amateurs project atmospheric concentrations into the future.

Models that will be used:

- Exponential time series smoothing
- Seasonal ARIMA (SARIMA)
- Facebook Prophet

Overall benefits will include:

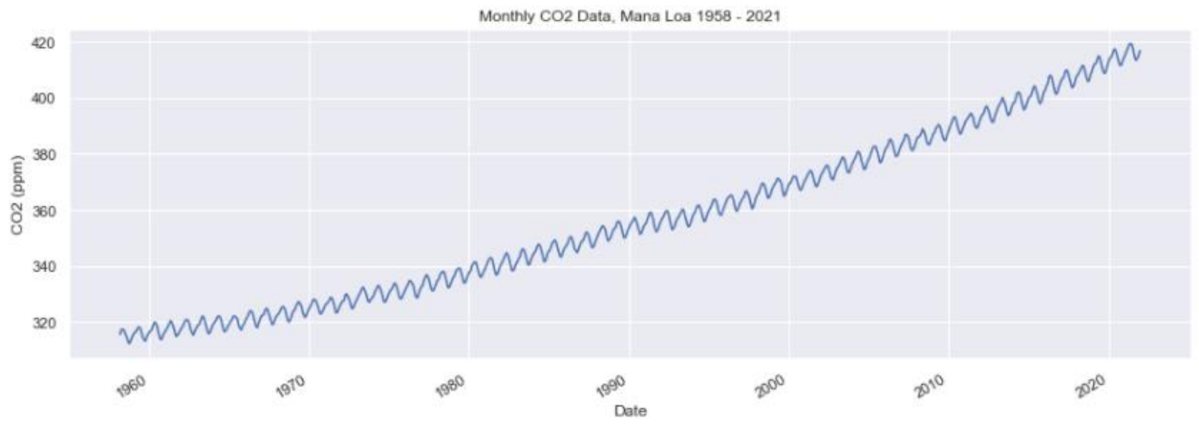
- Defining a trend of measurements based on historical data, and
- Offering a tool to accurately predict measurements into the future, based on past trends

The Data

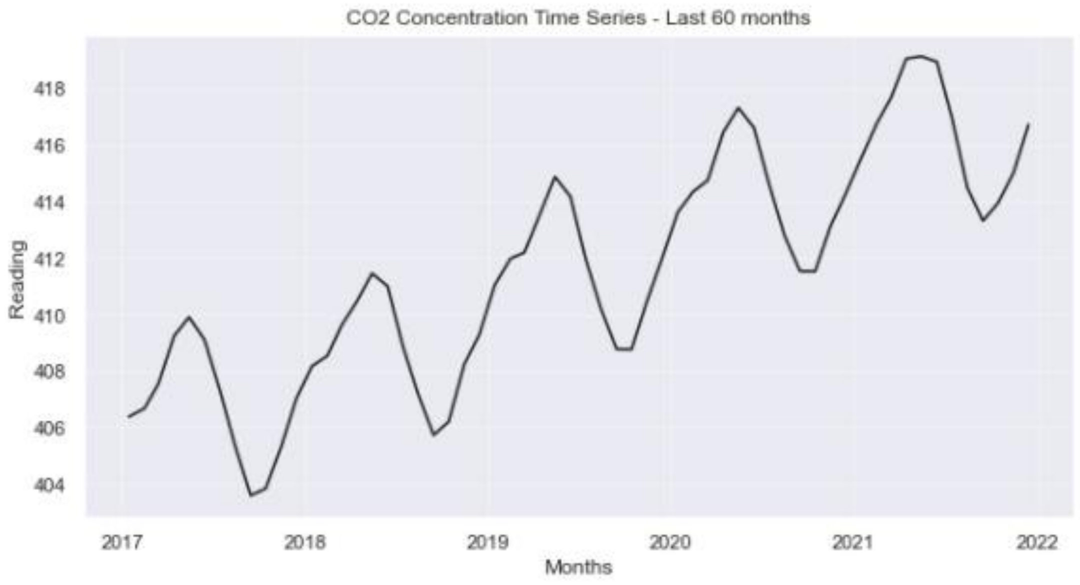
The data used for the analysis is taken from the NOAA Global monitoring Laboratory Earth System Research Lab at the Manua Loa observatory in Hawaii. It is a singe csv file that summarizes monthly CO_2 data, and the following other parameters:

- Year
- Month
- Decimal days: a decimal representation of the date (ie: 1958-03 is 1958.203)
- Average: this is the monthly mean CO_2 data, constructed from daily mean measurements. Monthly values are corrected to center of month based on average seasonal cycles

The data can be represented visually in a Time Series run sequence plot – it displays as an upwards trending, seasonal plot, with a sawtooth-like shape to the seasonal trends. Note that data exists from March 1958 to December 2021. There is no missing readings in this data set – each month has been filled with what appears to be proper readings (numerical data, date data in sequence and in proper format):



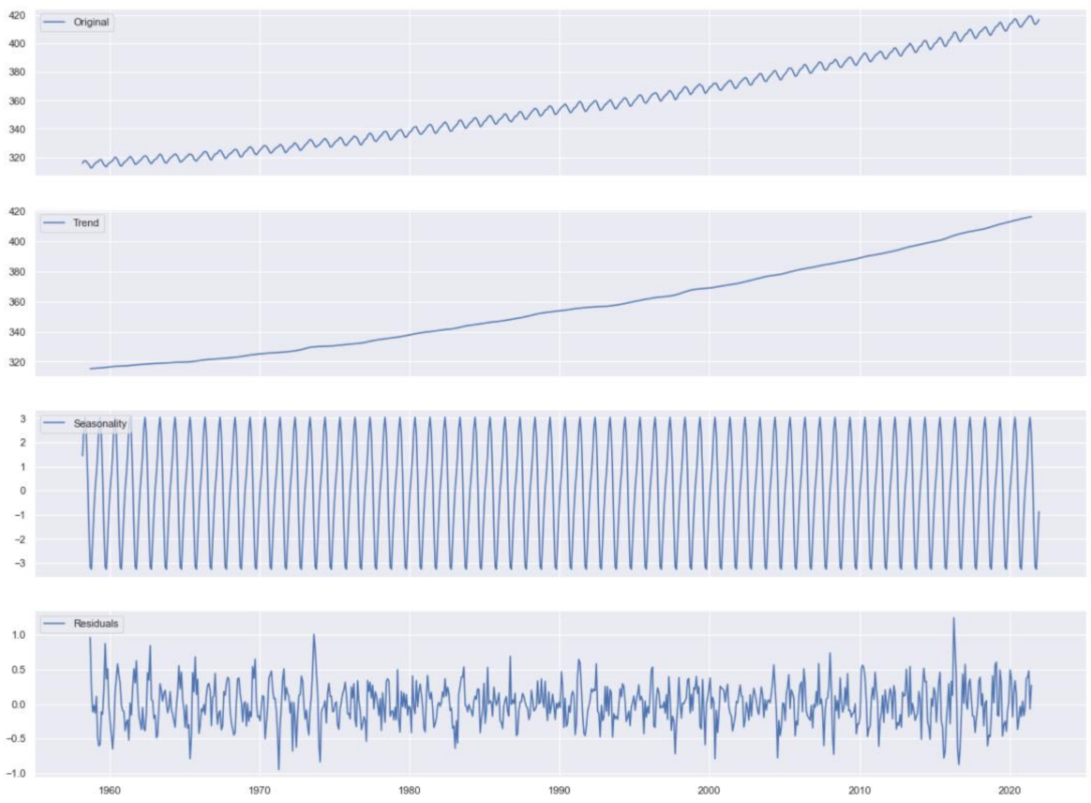
Zooming in to the last 5 years/60 months of data, we can see the same trend/seasonality holds:



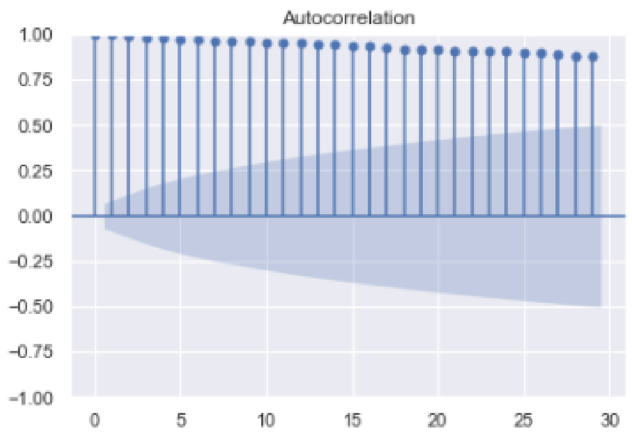
In the next section, as part of exploring the data, we will break the time series down into its components and check it for stationarity/non-stationarity.

<u>**Exploratory/Time Series Analysis**</u>

First, we decomposed the time series into its main components: trend, seasonality, and residuals. This was done using the statsmodels api seasonal decompose package. Breaking the plot into its components, it is clear that the time series decomposes pretty well, with a clear trend and seasonal component. The residual component, it could be argued, shows a little autocorrelation within. However, it is difficult to distinguish from random noise, so other techniques will be required to investigate further:

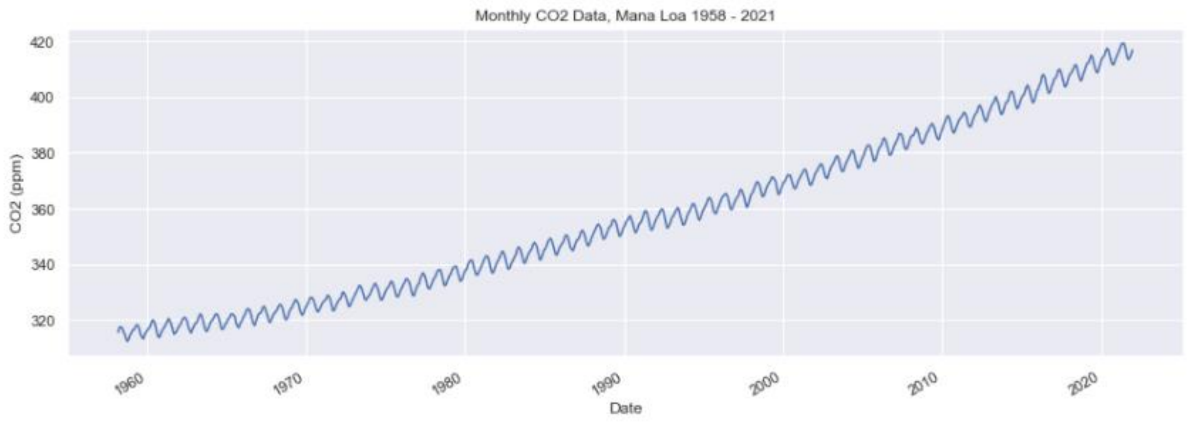


As an additional measure, we decided to look at the autocorrelation function for the time series – it shows that our first instincts were not incorrect and that there is significant autocorrelation within the time series, even at greater than 30 lags:



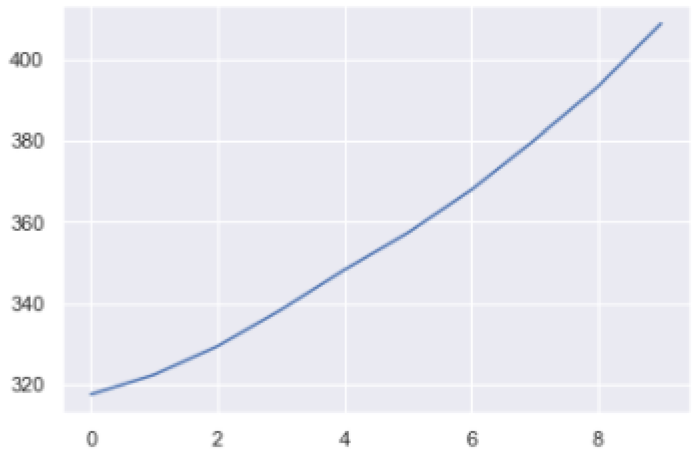
Now it is time to check the series for stationarity. Given the time series clearly shows trend, seasonality and autocorrelation, we can say for certain that it isn't a stationary series. However, we will confirm this via various testing procedures:

- Run Sequence Plot: the run sequence plot clearly shows trend & seasonality. Obviously, this means the time series is not stationary



- Chunking: splitting the time series, 760 records into evenly sized chunks of 76, we again found the series to be non-stationary.
  - Chunk mean: for each chunk there is a continued increase in the mean. This is not the mark of a stationary series (changing mean – trend)

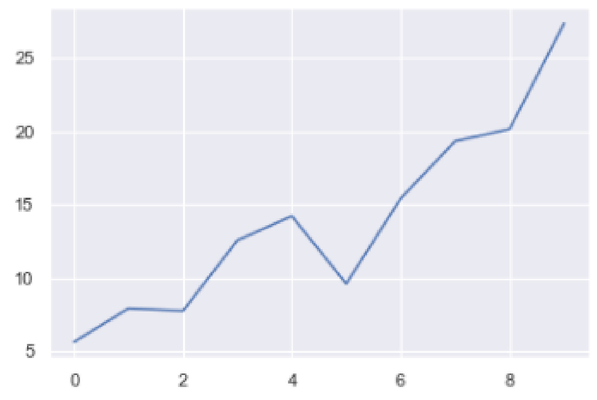
```
[15]: # check mean of chunks - mean rising indicates non-stationarity
      np.mean(chunks, axis=1)
      plt.plot(np.mean(chunks, axis=1))

[15]: [<matplotlib.lines.Line2D at 0x23669feacc8>]
```



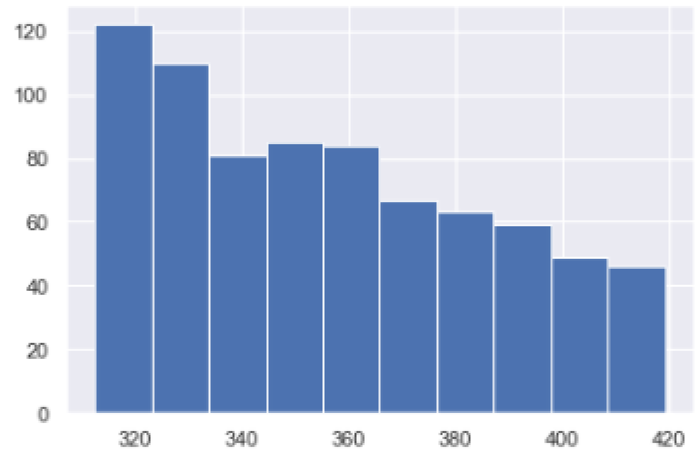
  - Chunk variance: analysis shows the chunks do not have a constant variance (heteroscedastic) and thus is not stationary

```
[16]: # check variance of chunks - changes in variance (heteroscedasticity) indicate non-stationarity
      np.var(chunks, axis = 1)
      plt.plot(np.var(chunks, axis=1))
```

```
[16]: [<matplotlib.lines.Line2D at 0x2366a368588>]
```



- Histogram: plotting a histogram to check for normal or near normal distribution of terms can indicated stationarity. Given the histogram shows a distribution of terms that are nowhere near normal, this is indicative of a non-stationary series:



- Statistical Testing: The Augmented dickey-fuller test will check for stationarity. The test tests against a null hypothesis (non-stationary), with the alternative being a stationary series. The augmented statistic is a negative number – the more negative, the stronger the rejection of the null (non-stationary) hypothesis. Given the test works better with larger datasets, and is sensitive to data with changing variance, it is used in concert with other methods to determine stationarity. For the given case, the dataset was evaluated for stationarity using the adfuller module from the statsmodels api. The results, as shown below (high adf statistic, well above any critical value threshold needed to reject the null) shows the series is definitely non-stationary:

```
ADF:  5.380000880062172
p-value:  1.0
Observations:  745
Critical Values:  {'1%': -3.4391580196774494, '5%': -2.8654273226340554, '10%': -2.5688400274762397}
```

As an example of making the series stationary via transformations, we will look at the same adf test on just the residuals for the time series decomposition. As you can now see, with trend and seasonality removed, the test rejects the null rather strongly and concludes the series is stationary:

```
ADF:  -9.193691444159366
p-value:  2.09015182428935e-15
Observations:  739
Critical Values:  {'1%': -3.439229783394421, '5%': -2.86545894814762, '10%': -2.5688568756191392}
```

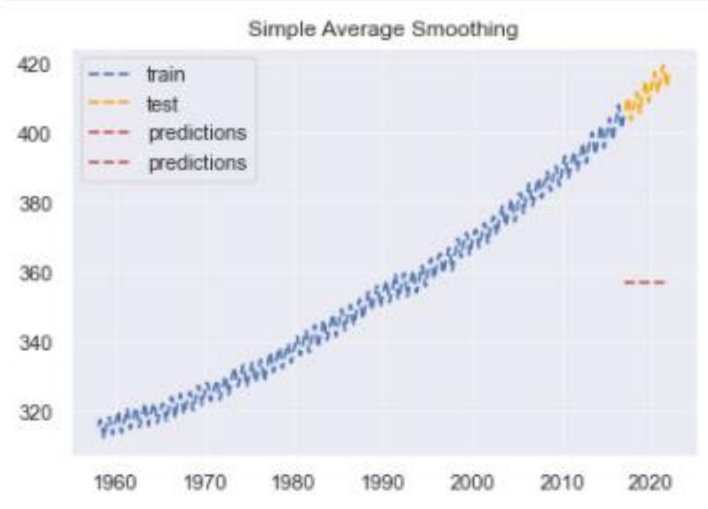
Next, we will move into modeling the time series. Even though there is trend, seasonality and autocorrelation, the models we will move forward with will deal with these components, even if some data massaging is required (in the form of differencing for autocorrelation etc).

<u>Time Series Modeling</u>

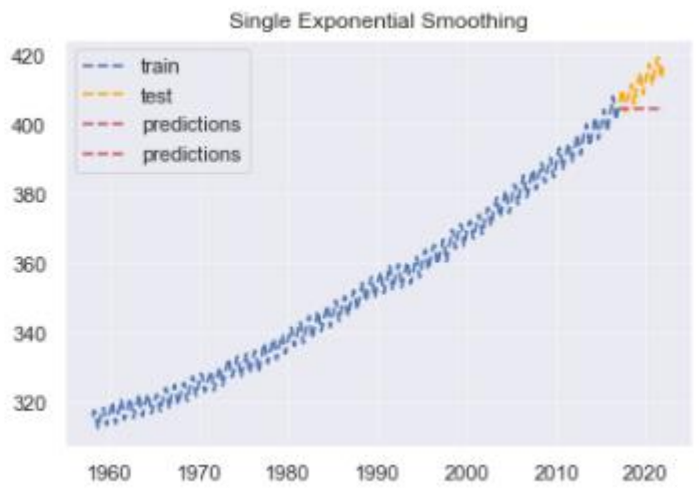
<u>Model 1 – Time Series Smoothing</u>

Exponential smoothing of a time series, when paired with a holdout data set (usually a small piece of the time series held back to compare to predictions) can prove effective, depending on the type of smoothing, in predicting the output of a time series. We modeled each of the time series smoothing algorithms to see which worked best.

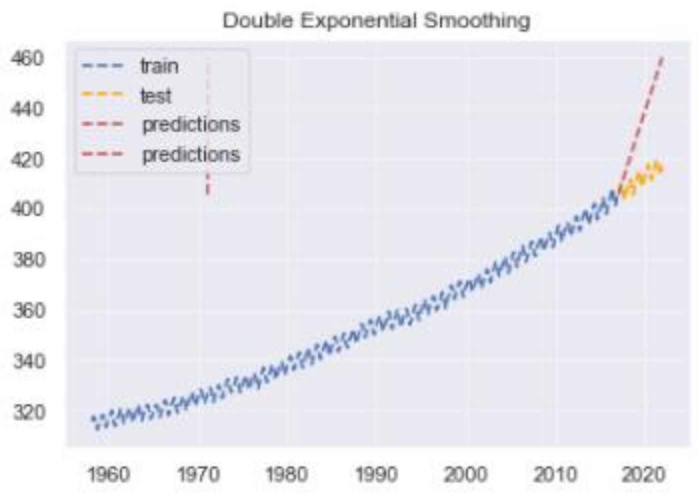
- Simple Average Smoothing: we ran a simple average smoothing algorithm, which basically just output the mean of the time series measurement. Of course, for a series with trend and seasonal components, this proved to model/predict pretty poorly, as it just output a prediction of the time series average. Graphically, we can see that this does not capture the shape of the time series and does a poor job of predicting vs the holdout set (MSE of over 18000)



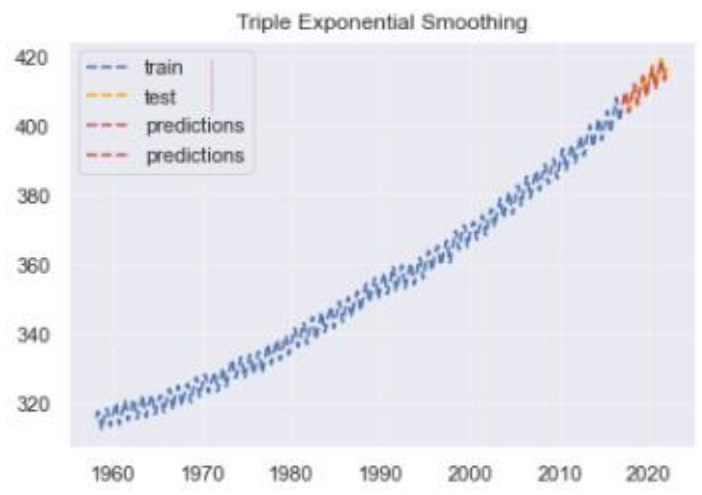
- Single Exponential Smoothing: essentially exponential weighted average smoothing, it does a slightly, although not much better job of capturing time series behavior. Instead of forecasting the time series mean, it essentially forecasts ahead the last reading of the "training set", picking up neither trend or seasonality. Graphically, we can see this – closer than simple average but not there yet (MSE ~ 3800, about 1/6th of simple average):



- Double Exponential Smoothing: has the ability to pick up trend, but not seasonality. It does this by adding a second component to the formula/algorithm to smooth out trend. It picks up trend, by drawing a tangent line to the curve at the last point of the "training set". Again, closer than single exponential smoothing, but not quite there yet (MSE is actually much higher at ~ 36000). Graphically:



- Triple Exponential Smoothing: using triple exponential smoothing, we finally hit the sweet spot, being able to predict/forecast a time series against a holdout set. It does this because triple exponential smoothing builds on double exponential smoothing by adding a 3rd term to account for smoothing seasonality. When running this against the holdout set (last 5 yrs/60 months of data). The MSE was quite low (~ 18!) indicating that the shape/behavior of the time series was being picked up. Graphically, it shows as the predictions and "test" data line up quite well on the run sequence plot:



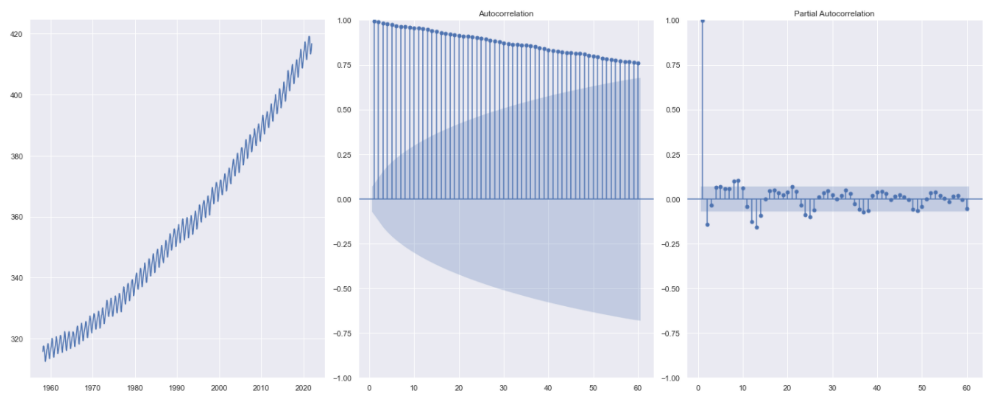
Overall, higher order time series smoothing (Triple Exponential Smoothing) worked quite well, capturing the behavior of the time series well enough to predict behavior 5 years into the future against a holdout set, with very low error. For somebody constrained by:

- Time
- Knowledge
- Computing Resources

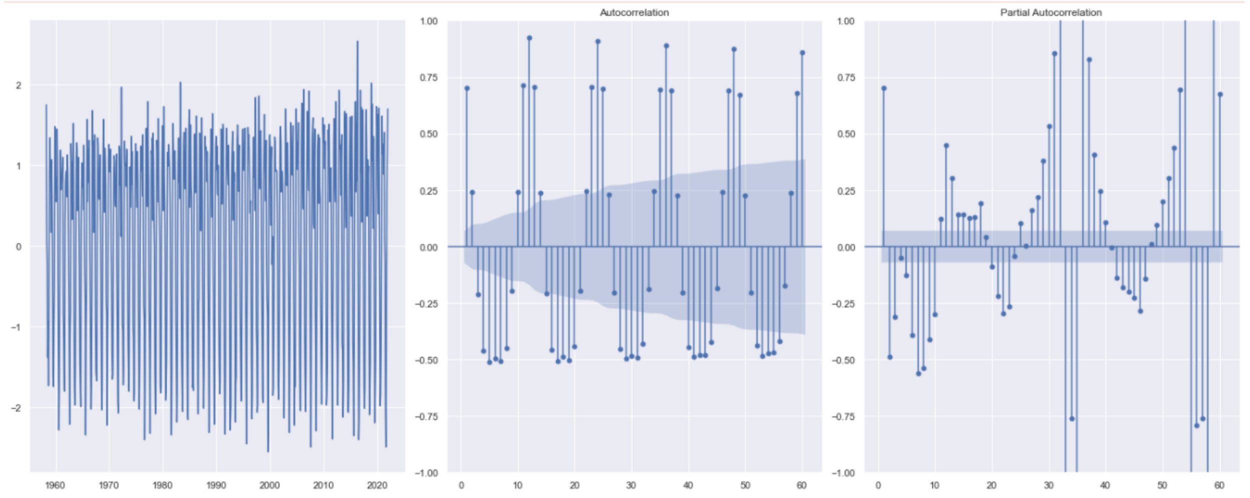
Who is just looking to gain some personal understanding, triple exponential smoothing would be a more than acceptable model to use to predict future atmospheric CO_2 concentration.

<u>Model 2 – SARIMA Modeling</u>

Knowing that our time series has both seasonality and a pretty high level of autocorrelation, it is best to use a Seasonal Autoregressive Integrated model (SARIMA). One of the challenges here, given the significance of the autocorrelation found in the data, is figuring out the best/most appropriate lag structure and the p, d, q parameters for modeling. The first plot (below) shows the auto and partial autocorrelation structure of the time series:



As we can see, autocorrelation is high, up to 60 lags. Based on Box-Jenkins methodologies, this type of structure (no, or very slow decay in the autocorrelation structure) indicates a non-stationary time series. Taking a single difference, the autocorrelation plot takes on a much different state (suspension bridge like), and now appears stationary and ready for analysis:



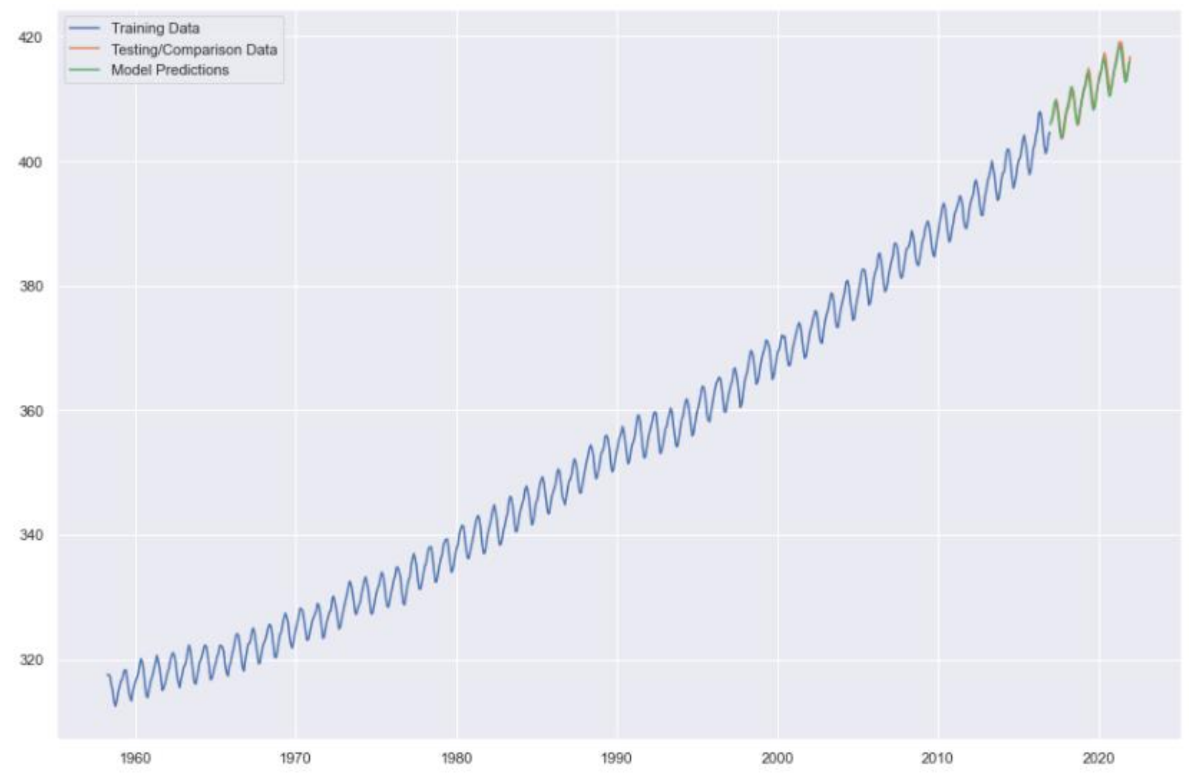
It should be noted, going forward, that due to the general messiness of the auto/partial auto structures, that a stepwise model will be used, via auto arima, to select specific p, d, q parameters. Also note that modeling will again be done on a training set, which will include all data other than the last 60 months, which will be held out as a test/comparison set. Running the stepwise algorithm, with the following parameters:

```
stepwise_model = pm.auto_arima(co2_data['average'][:-60], start_p=1, start_q=1,
                           max_p=3, max_q=3, m=12,
                           start_P=0, seasonal=True,
                           d=0, D=1, trace=True,
                           error_action='ignore',
                           suppress_warnings=True,
                           stepwise=True)
print(stepwise_model.aic())
```

We find the following, by evaluating the AIC, to be the best set of parameters for the subject time series:

```
 Best model:  ARIMA(1,0,3)(0,1,1)[12]
 Total fit time: 147.986 seconds
 368.5350643495243


auto-fit order: : (1, 0, 3)
auto-fit seasonal_order: : (0, 1, 1, 12)
```

Using this model to predict future values, based on the 700+ monthly measurements (also, holding out the 5 yrs/60 months of data for comparison), we again get a model that does a very good job of predicting future data, as compared to the holdout set – graphically, there is very little delta between the model prediction (green) and the testing data (red):



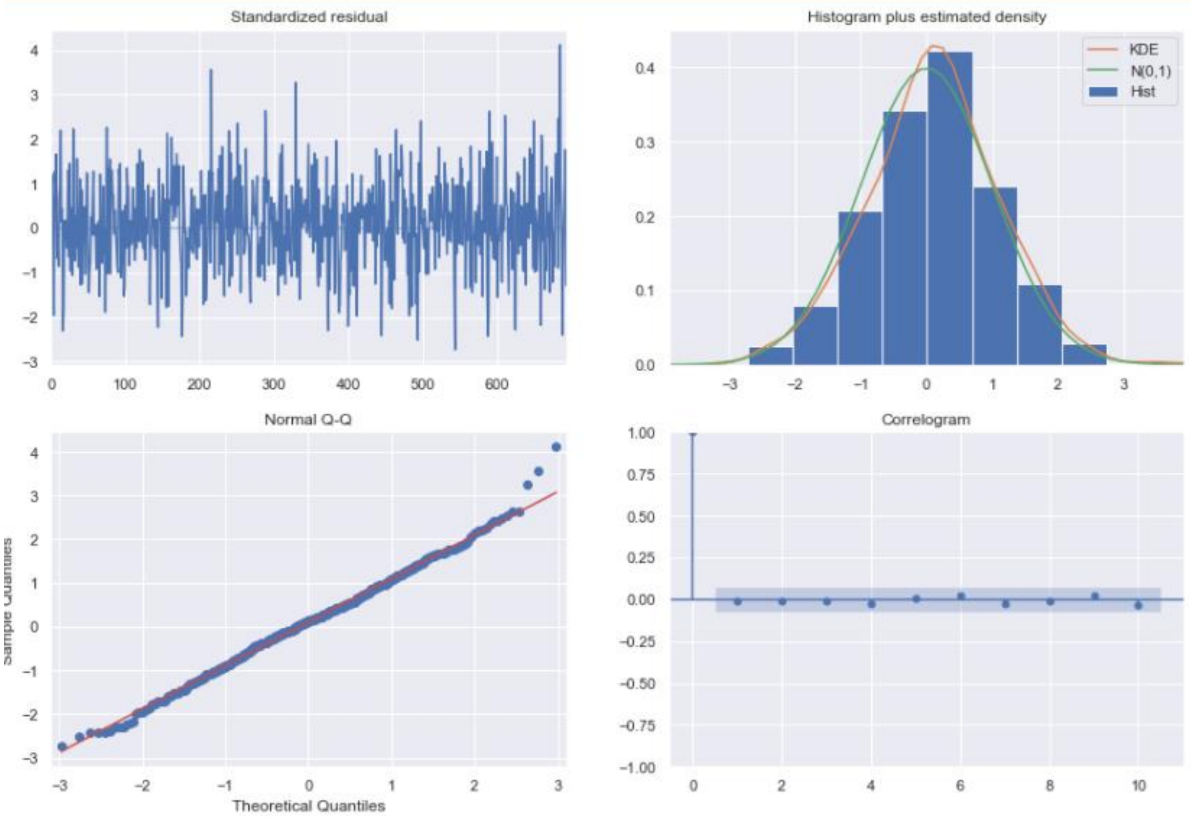
This can be further corroborated, looking at a numerical comparison between the predicted and test data - note the very small differences between predictions and holdout data (less than 1/8th of a percentage point):

[46]:

	Preds	holdout data	absolute error	squared error
2017-01-01	405.847168	406.36	0.512832	0.262997
2017-02-01	406.641681	406.66	0.018319	0.000336
2017-03-01	407.591728	407.53	0.061728	0.003810
2017-04-01	409.084749	409.22	0.135251	0.018293
2017-05-01	409.710973	409.89	0.179027	0.032051
2017-06-01	408.925079	409.08	0.154921	0.024000
2017-07-01	407.248395	407.33	0.081605	0.006659
2017-08-01	405.123919	405.32	0.196081	0.038448
2017-09-01	403.720782	403.57	0.150782	0.022735
2017-10-01	403.991404	403.82	0.171404	0.029379
2017-11-01	405.523616	405.31	0.213616	0.045632
2017-12-01	406.963221	407.00	0.036779	0.001353
2018-01-01	408.087165	408.15	0.062835	0.003948
2018-02-01	408.910753	408.52	0.390753	0.152688
2018-03-01	409.831758	409.59	0.241758	0.058447
2018-04-01	411.324590	410.45	0.874590	0.764908
2018-05-01	411.950624	411.44	0.510624	0.260737
2018-06-01	411.164540	410.99	0.174540	0.030464
2018-07-01	409.487667	408.90	0.587667	0.345352
2018-08-01	407.363001	407.16	0.203001	0.041209
2018-09-01	405.959674	405.71	0.249674	0.062337
2018-10-01	406.230106	406.19	0.040106	0.001608
2018-11-01	407.762128	408.21	0.447872	0.200589
2018-12-01	409.201544	409.27	0.068456	0.004686
2019-01-01	410.325297	411.03	0.704703	0.496606
2019-02-01	411.148696	411.96	0.811304	0.658214
2019-03-01	412.069512	412.18	0.110488	0.012208
2019-04-01	413.562154	413.54	0.022154	0.000491
2019-05-01	414.187998	414.86	0.672002	0.451586

Looking at the model performance/fit, it shows that this is a well-built model:

- Residuals are distributed like random noise
- Correlations are 1 order/lag
- Theoretical and sample quartiles fit very well/on the same place
- The histogram and estimated density show near normal behavior



Overall, although much more complex than an exponential smoothing technique, the SARIMA model does a very good job of predicting on a holdout set, with very little error. The only drawback of this model is with high autoregression, the model becomes more complex – especially when trying to figure out the orders (p, d, q). However, as a model, it carries a lot of predictive power, although probably too complex for an enthusiastic amateur.

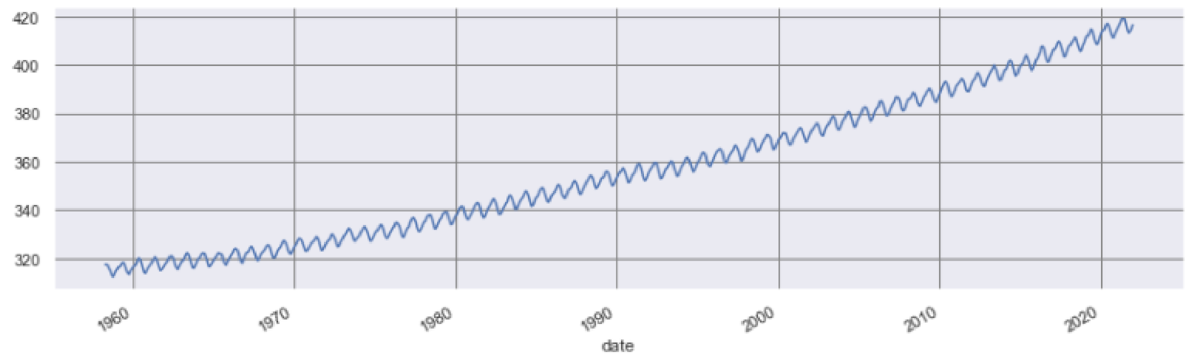
<u>Model 3 – Facebook Prophet</u>

The third model run was a Facebook prophet model. Facebook built the prophet library to aid in business forecasting, as it is a known area of weak ability. The Prophet model offers some advantages above/beyond standard ARIMA/SARIMA modeling:

- The ability to forecast at different intervals than the training data
- The ability to inject events/non-standard seasonal components
- The model itself is generally additive and deals with the trend, seasonal, and autocorrelation components separately, making it simpler to set up

To begin the Prophet modeling phase, a run sequence plot was again the start point:

```
# plot data for average CO2
co2_data['average'].plot()
plt.grid(color = 'grey', linestyle = '-')
```



Differently with Prophet though, the data needed to be massaged/engineered, configured into a data frame with 2 columns only

- ds column: containing date data
- y column: containing whatever data we want to model a time series on

The monthly CO_2 data was set up in a data frame (co2_data_p) for this specific task:

```
[55]: co2_data_p
```

```
[55]:
           ds          y
  0    1958-04-15  317.45
  1    1958-05-15  317.51
  2    1958-06-15  317.24
  3    1958-07-15  315.86
  4    1958-08-15  314.93
 ...       ...        ...
760    2021-08-15  414.47
761    2021-09-15  413.30
762    2021-10-15  413.93
763    2021-11-15  415.01
764    2021-12-15  416.71

765 rows × 2 columns
```

Like we did in previous modeling exercises, the data was split into a train/test set, so that we can compare the Prophet forecasts to some real-world data. Again, to be consistent, holding out 60 months (5 yrs) was determined to be a good set.

After fitting the model, using the .fit() method on the train set, it was time to forecast. Prophet requires you to set up a future data frame to make and store predictions. Note the model was set to forecast monthly:

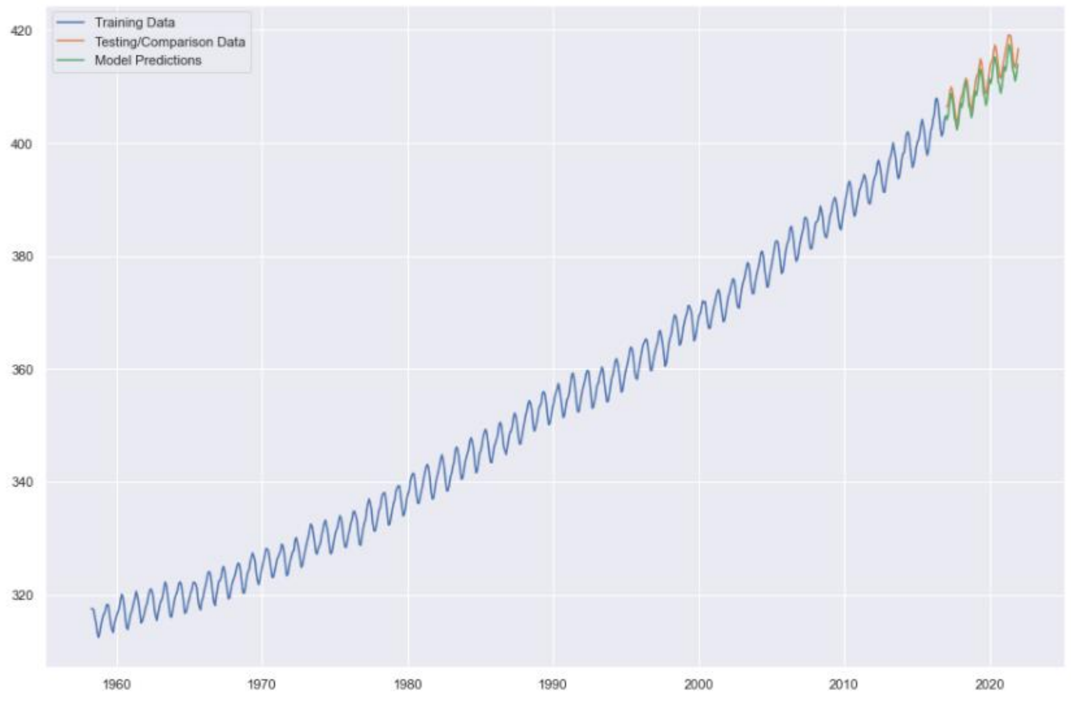
```
[58]: # forecast 5 yrs/60 months into future to match other models
      # prophet requires a blank dataframe to input predictions
      # will also provide blank set for dates within dataset to allow for fit
      future = model.make_future_dataframe(periods=60, freq='M', include_history=True)
      future.tail()
```

[58]:

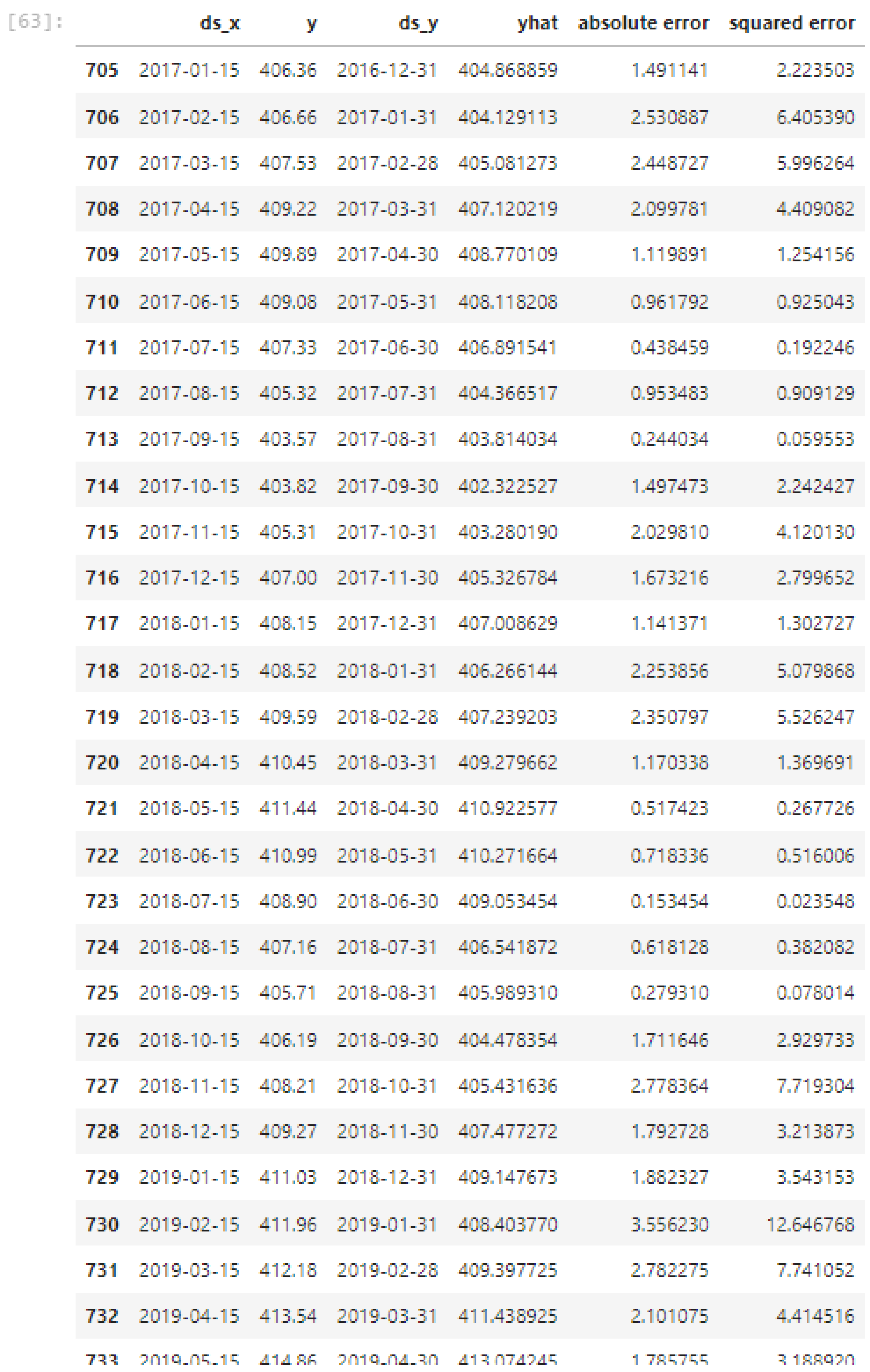
	ds
760	2021-07-31
761	2021-08-31
762	2021-09-30
763	2021-10-31
764	2021-11-30

A forecast was then created using the predict method, with the future data frame as a parameter. This generates a data frame full of predictions, out 60 months from the end of the training data set, which matches up with our holdout set. Taking only relevant columns from the data frame, we ended up with dates, predictions (yhat) and upper/lower bounds. Plotting only the yhat prediction, and comparing it again to the holdout data, we see the following graphically:

- the model once again does an excellent job of predicting future values, with very little error when compared to the holdout set (~ ¼ of a percentage point)



Numerically, tis is captured in the following data frame – note the low values for absolute and squared error on each record. It is the opinion of this author, that something that predicts out to the future (5 yrs) with no error greater than 3 ppm (on 400+ ppm) is an excellent model:



[63]:

	ds_x	y	ds_y	yhat	absolute error	squared error
705	2017-01-15	406.36	2016-12-31	404.868859	1.491141	2.223503
706	2017-02-15	406.66	2017-01-31	404.129113	2.530887	6.405390
707	2017-03-15	407.53	2017-02-28	405.081273	2.448727	5.996264
708	2017-04-15	409.22	2017-03-31	407.120219	2.099781	4.409082
709	2017-05-15	409.89	2017-04-30	408.770109	1.119891	1.254156
710	2017-06-15	409.08	2017-05-31	408.118208	0.961792	0.925043
711	2017-07-15	407.33	2017-06-30	406.891541	0.438459	0.192246
712	2017-08-15	405.32	2017-07-31	404.366517	0.953483	0.909129
713	2017-09-15	403.57	2017-08-31	403.814034	0.244034	0.059553
714	2017-10-15	403.82	2017-09-30	402.322527	1.497473	2.242427
715	2017-11-15	405.31	2017-10-31	403.280190	2.029810	4.120130
716	2017-12-15	407.00	2017-11-30	405.326784	1.673216	2.799652
717	2018-01-15	408.15	2017-12-31	407.008629	1.141371	1.302727
718	2018-02-15	408.52	2018-01-31	406.266144	2.253856	5.079868
719	2018-03-15	409.59	2018-02-28	407.239203	2.350797	5.526247
720	2018-04-15	410.45	2018-03-31	409.279662	1.170338	1.369691
721	2018-05-15	411.44	2018-04-30	410.922577	0.517423	0.267726
722	2018-06-15	410.99	2018-05-31	410.271664	0.718336	0.516006
723	2018-07-15	408.90	2018-06-30	409.053454	0.153454	0.023548
724	2018-08-15	407.16	2018-07-31	406.541872	0.618128	0.382082
725	2018-09-15	405.71	2018-08-31	405.989310	0.279310	0.078014
726	2018-10-15	406.19	2018-09-30	404.478354	1.711646	2.929733
727	2018-11-15	408.21	2018-10-31	405.431636	2.778364	7.719304
728	2018-12-15	409.27	2018-11-30	407.477272	1.792728	3.213873
729	2019-01-15	411.03	2018-12-31	409.147673	1.882327	3.543153
730	2019-02-15	411.96	2019-01-31	408.403770	3.556230	12.646768
731	2019-03-15	412.18	2019-02-28	409.397725	2.782275	7.741052
732	2019-04-15	413.54	2019-03-31	411.438925	2.101075	4.414516
733	2019-05-15	414.86	2019-04-30	413.074245	1.785755	3.188920

One, again, it can be said that the Prophet model did an excellent job capturing the underlying structure of the time series, to the point where medium term (5 year) predictions had very little error when compared to actual values from a holdout set. The Facebook Prophet model is definitely an excellent model.

Recommended Model

All models that were tested (Exponential Smoothing, SARIMA, Prophet) did an excellent job of forecasting against a holdout data set, with very small errors. Each model would be a fine choice to use as a forecaster/predictor of atmospheric CO_2 concentration. Based on the use case, and the skillset/understanding of the person, the following models are recommended, with corresponding reasoning:

- Enthusiastic Amateurs/Hobbyists: Exponential Smoothing, as it requires the least amount of mathematical knowledge and understanding. One can simply run the model from statsmodels, with only a seasonality period required for the Triple Exponential Smooth.
- Professionals: Facebook Prophet. Overall, even though it is slightly less accurate (although we're talking fractions of a percentage point of error in both cases), the model offers more in the way of flexibility and utility. Specifically:
    - The ability to model at differing time intervals
    - The ability to split apart all components, without having to go through the often-messy process of figuring out coefficients for autocorrelation, moving average etc
    - The ability to dig deeper and add differing components (one off vents etc) to get a deeper understanding of cause-effect relationships
    - The ability to analyze confidence intervals/ranges of potential outcomes

However, it should be noted once again that all models to an excellent job of prediction/forecasting, it is just a matter of intended use case that differentiates.

Key Findings / Take Aways

Key findings and takeaways of this study/report include:

- Exponential Smoothing, for certain series, can be used to predict/forecast against a holdout set with great accuracy. It is an excellent tool for hobbyists, as it is simpler to run and understand
- Although much more complex, at least with this dataset, ARIMA/SARIMA models can forecast with extraordinary accuracy against a holdout set (to tenths of a percentage point accuracy). However, they are much more complex to set up, especially when tuning the autoregressive and moving average orders.
- Facebook Prophet forecasts to almost the same level as an ARIMA/SARIMA model, although its structure (generally additive, breaking down each component and treating separately, before recombining) lends it to be more easily understood and implemented (no tuning p, d, q parameters etc). Additionally, the flexibility allows for more easily implemented extended analysis (one off events, sparse seasonality components, data driven events etc).

Overall, one key takeaway is that, with the correct data set, each of the models can be quite accurate in implementing a prediction/forecast. To say the accuracy was surprising would be an understatement. However, with decades of good, clean, accurate data I guess it should have been expected.

Further Analysis

As previously mentioned, it would be instructive, especially when using the Facebook Prophet model, for a professional to dig into specific events and markers, to better gain a cause and effect understanding of the atmospheric concentration. Data for discrete events, such as known volcanic events could be isolated using prophet to gain a better understanding of how they affect concentration, versus thins that are continuously at work (everyday emissions, deforestation etc). With the Prophet module, the flexibility is there to model many different scenarios, to give data either in:

- Increased granularity, or
- To gain a better understanding of how certain events affect the time series