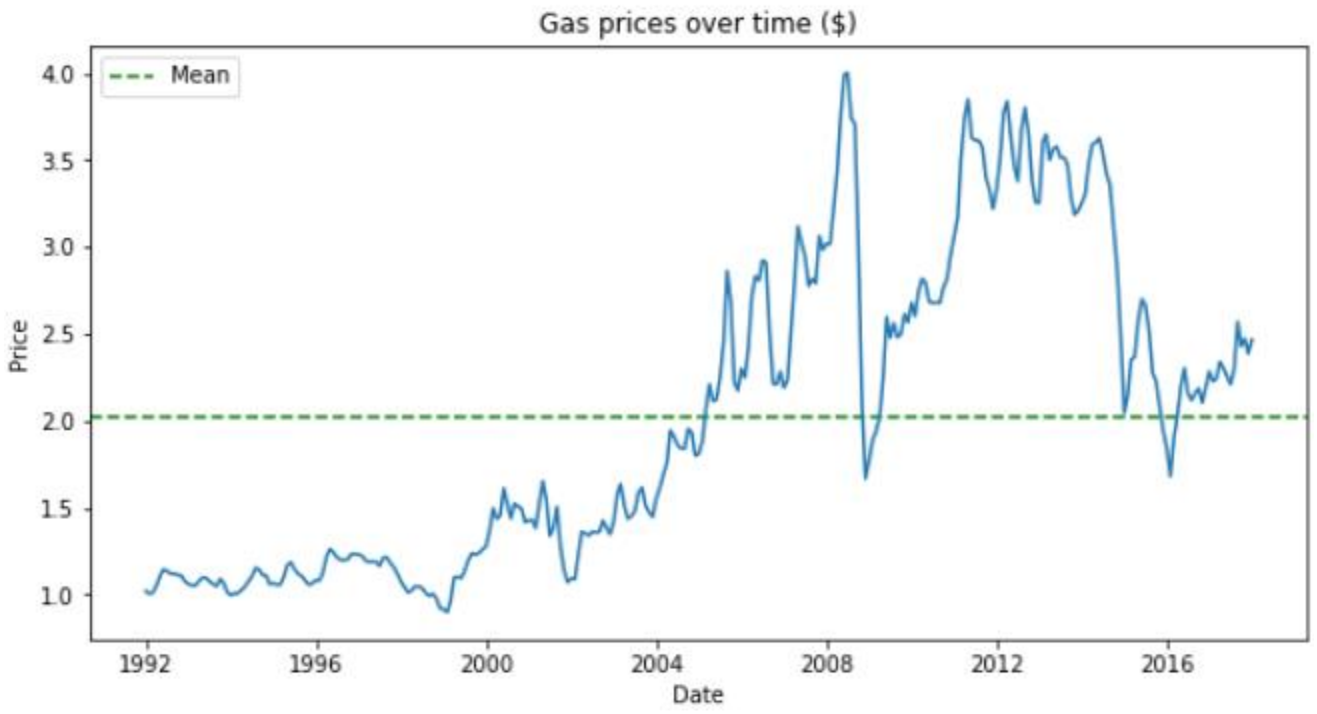
Forecasting Take Home Assignment

Forecast of Retail Motor Gasoline and On-Highway Diesel Fuel Prices (**1992-01-01** to **2018-02-01**)

First a quick look at the data.



What stands out just from visual inspection:

1. Sudden periods of extreme volatility are common
2. Weak evidence of yearly seasonality
3. Spikes climb/drop often more than at least 1 month at a time

Modeling

How to split into train/test sets to best evaluate model performance? CV?

We can select our interest in predicting the price of gasoline accurately over the next **12 months**, with equal weight being assigned to each of these 12 measurements.

12 months is sufficiently long of a horizon here to be interesting due to the high volatility.

- **Training set:** All data except last 12 months
- **Test set:** Just last 12 months

We will use *cross validation* on our training set to validate and select our best model according to past performance and then test this model on our test/holdout set.

Forecasting Take Home Assignment

Our CV will be done using the *expanding window* method using a small initial training period, just enough to learn some yearly seasonality:

- **Initial period**: 365 * 3 days
- **Horizon**: 365 days (also period = 365 days for simplicity)
- **Total predictions** : (25-3 years) * 12 months/year = 264, which we average over evenly
- **Metric**: **RMSE –** For interpretability we use RMSE as we are working with a unit of dollars but other metrics such as MAPE could also have been used here (no zeros)
    - Additionally, RMSE penalizes large errors more than a metric like MAE which in our case is desirable as we want to be more conservative with this data [1]

We can use the built-in to Prophet *cross_validation()* function to this end.

Key Prophet parameters?

- **changepoint_range** – Default value of first 80% of data to consider for last trend changepoint will be too low in this volatile case [1].
    - Set to high end of ***0.95*** from visual inspection.
- **changepoint_prior_scale** – We want to be careful not to overfit, but we also do not want to get stuck in the past and not react to changes quickly enough with default 0.05 [1 & 3].
    - Will run CV with values [0.05, 0.1, 0.5, 1, 10]
- **seasonality_prior_scale –** In contrast to business product forecasting such as Christmas trees we do not have strong evidence of seasonality so will likely lower the default 10.
    - Will run CV with values [0.01, 1, 10]
- **growth='linear' –** We do not have reason to believe in a near future saturation point for the price of gasoline, so we use a linear instead of a logistic trend

Outliers?

- Even though Prophet can handle outliers without any issues (as we can set them to None which Prophet can model around) we have so much volatility and sudden spikes that they are rather the rule to the data, rather than the exception, so we want to keep all our data.
- Removing outliers such as the spikes around 2008 would decrease the uncertainty for our trend forecast for the future, which we do not want to overconfidently say is improbable.

To incorporate holidays or not?

While theoretically there could be a case for holidays increasing gasoline demand and therefore price as more people travel, especially because gasoline is relatively inelastic, the data is not at a weekly/daily level so it would require some clever orchestration, and also we can hope that holidays are generally reflected in our monthly seasonalities.

Analysis

We run the CV grid mentioned above for **changepoint_prior_scale** and **seasonality_prior_scale** and obtain the best combination to be: (Full results can be found in the Appendix)

- **changepoint_prior_scale: 0.50**
- **seasonality_prior_scale : 0.01**

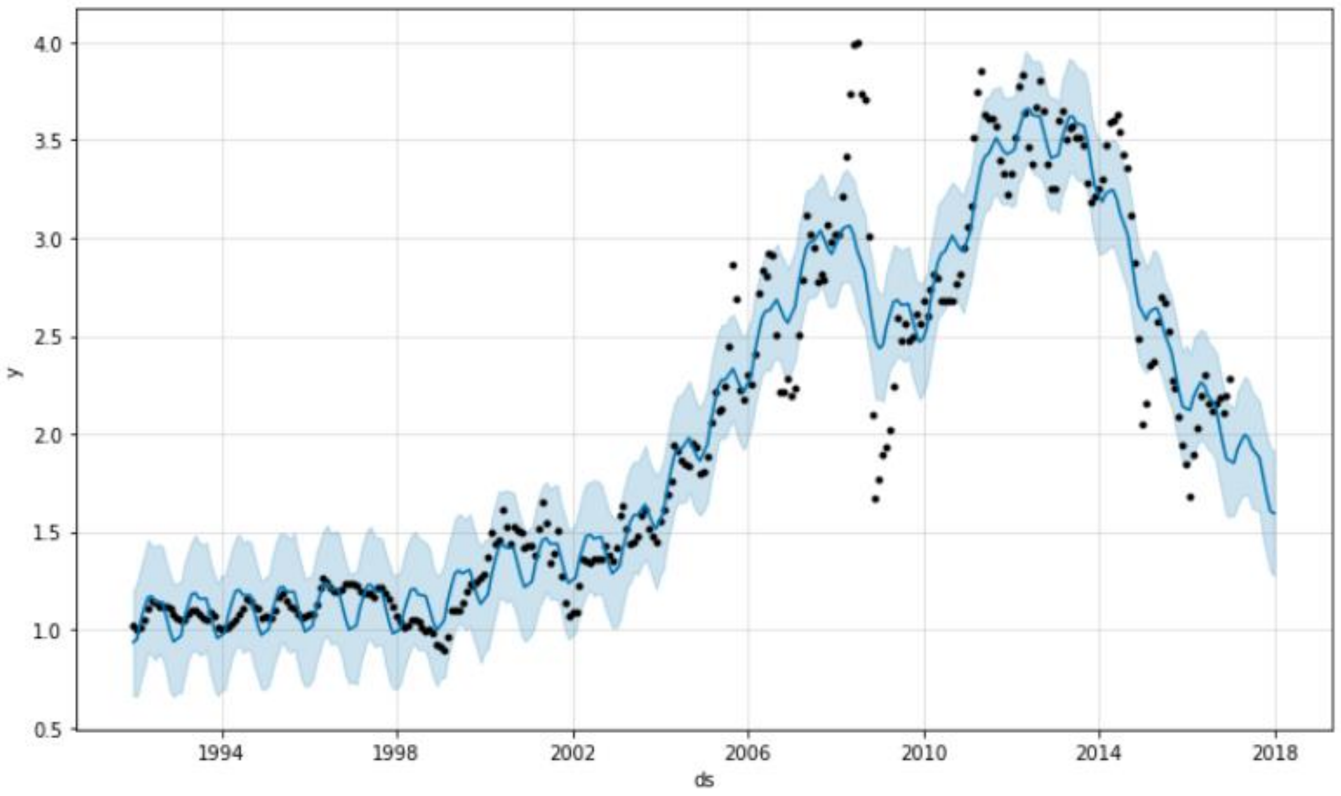
With an RMSE of **0.4075** averaged over all 22 (1995-present) of the 12 month horizons (each consisting of 12 predictions) which we can interpret directly in dollars.

While at first glance this value does not seem too good, we must remember that we have some essentially impossible to predict rapid 1 month drops (such as in 2008) that are always going to add a lot of error to an RMSE metric.

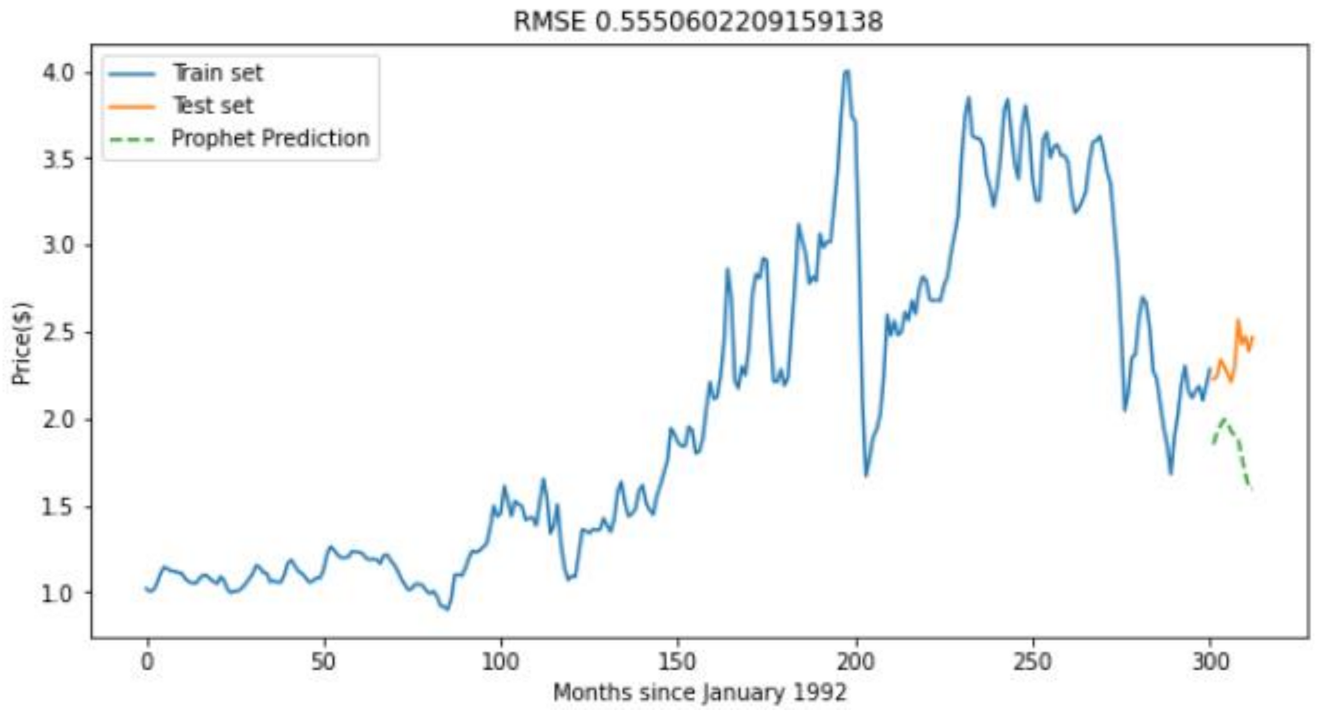
Training model using best CV parameters

Below are the model predictions represented by the blue line with light blue uncertainty bounds and a 12 month forward prediction added on at the end.

- Slow to react to some spikes (2013-2014 range) but models them better than a more conservative model would
- Trend is relatively strong



We next take a look at how this predicted 12 month period compares with our real test data.

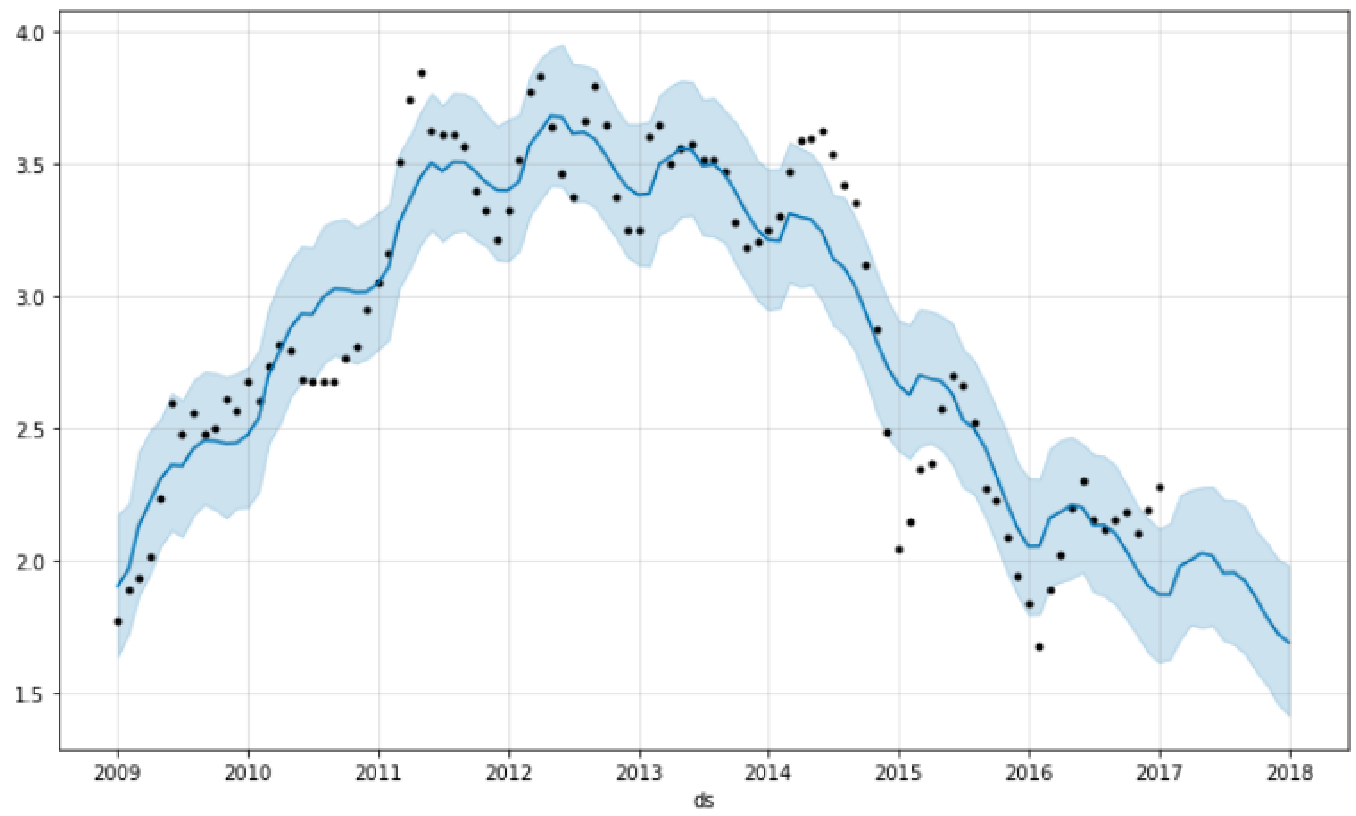


Unfortunately, we notice that these predictions are not quite good. Our **RMSE is 0.555** but the direction of change is wrong.

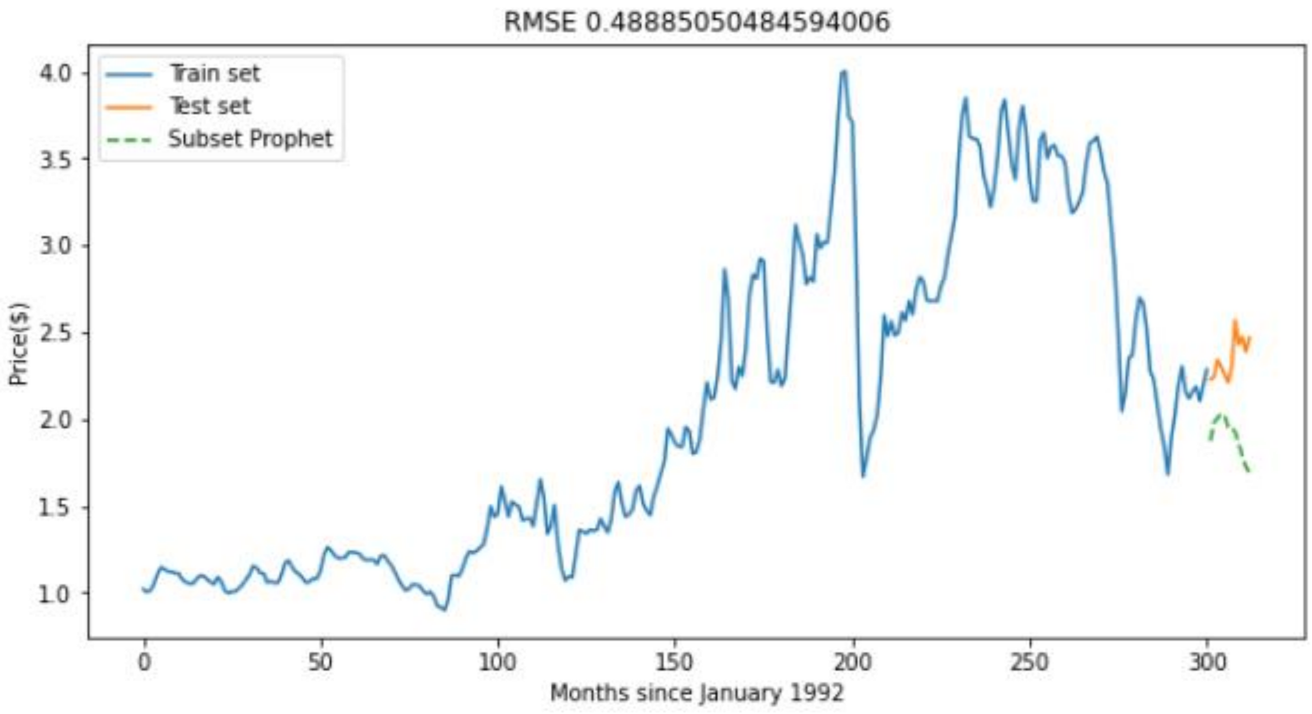
It predicts that the general strong downward trend over the past few years will continue while the true prices increased during that time.

What if we change the length of time we look at?

- Perhaps it would be effective to not try to backtest the model all the way from 1992 but rather focus on recent years (under the assumption that recent data is more likely informative of the future) to allow the model more flexibility.
  - We try to subset our training set to only dates **>= '2009-01-01'.**
- To do an apples-to-apples comparison of the full data model and the subset, we must perform CV on the same folds so we can directly compare RMSE values.
  - Our CV run with the same *param_grid* as before looks like:
    - **Initial:** 365 * 3 days
    - **Horizon**: 365 days
    - **Total predictions:** (8-3 years) * 12 months/year = 60 months predicted
  - We compare these 60 months with the last 60 months predicted by our full model
    - Subset Model CV (see Appendix) best RMSE: **0.467239**
    - Full Model with earlier CV parameters RMSE: **0.479282**
- We can see that our subset 8 year model is doing only marginally better
  - The graph of its predictions is below



And the corresponding prediction performance:



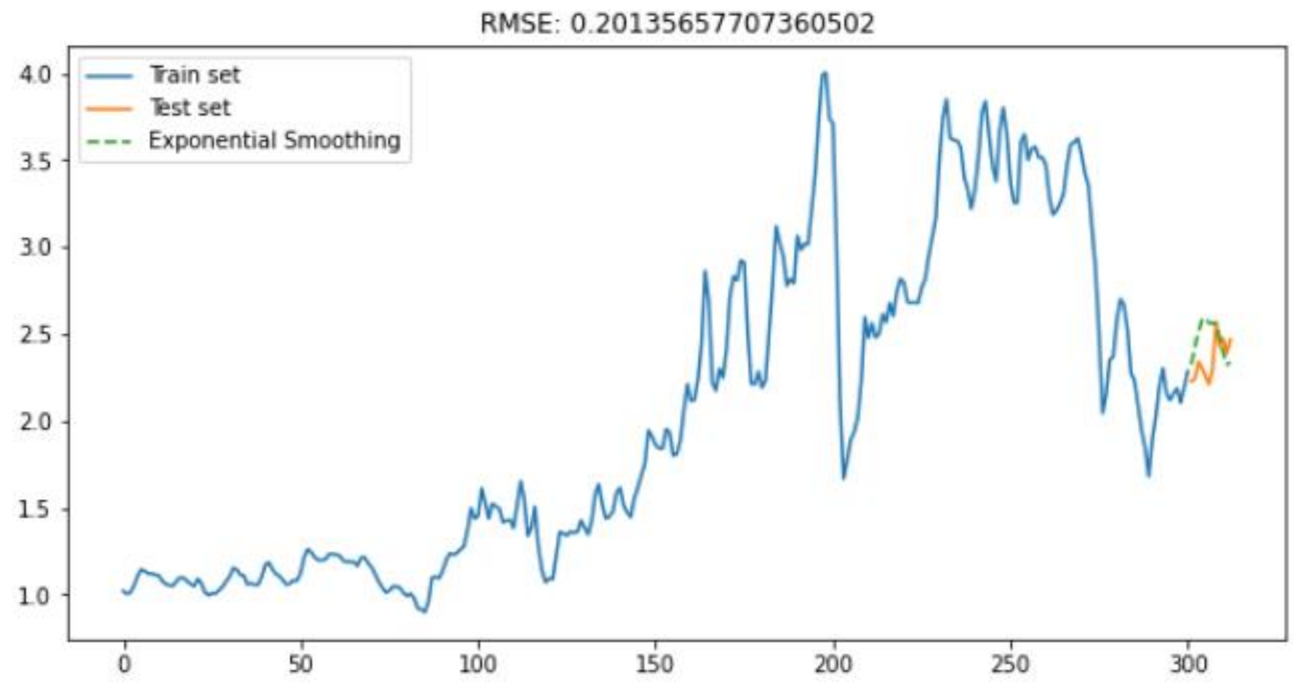
We note an improvement in the RMSE of **0.489 < 0.555** though the trend is still wrong and the graphs are almost indiscernible, the above just has slightly lower predictions.

How does Prophet compare to other methods?

For a quick comparison we look at an older well-known method known as Exponential Smoothing, used in this case from the *statsmodels* package.

We run it with rather basic/default parameters over all but 12 months of our data, and test it.

- trend = ‘add’, seasonal = ‘add’, seasonal_periods = ‘12’



This is better, with an accurate direction and a much better RMSE **(0.201 < 0.555)**, so let us take a closer look, as perhaps we just got lucky.

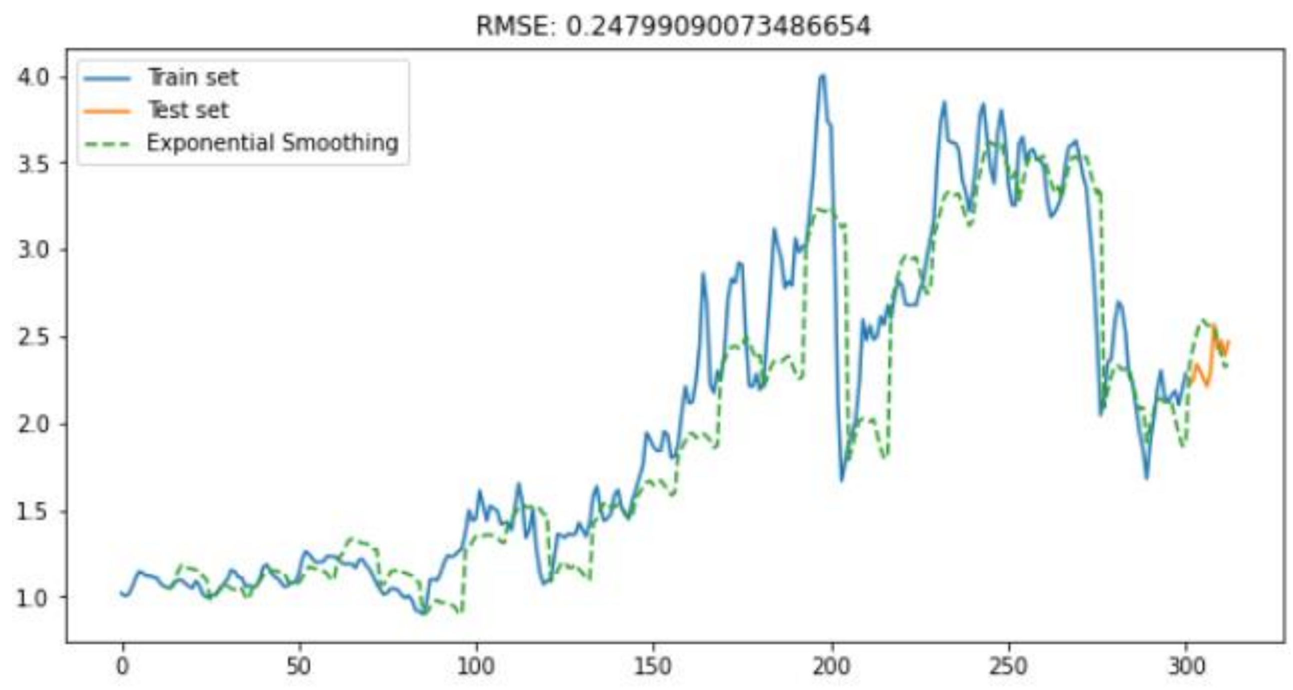
Backtesting Exponential Smoothing

We build a TimeSeriesSplit from sklearn, emulating what our Prophet model was doing above.

- Expanding windows
- 12 month horizons

```
TRAIN: [ 0  1  2  3  4  5  6  7  8  9 10 11 12] TEST: [13 14 15 16 17 18 19 20 21 22 23 24]
TRAIN: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24] TEST: [25 26 27 28 29 30 31 32 33 34 35 36]
TRAIN: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36] TEST: [37 38 39 40 41 42 43 44 45 46 47 48]
```

Results of this can be seen in the below graph.



Here we see all the predictions that were done on a 12 month rolling horizon (used 24 splits due to the amount of data we had so that our test sets were comparable to Prophet model).

Impressively, we see our all of our data our RMSE has risen just a bit to **0.248** but this is still significantly better than we were able to achieve using Prophet (0.555).

This is additionally without any actual grid search for best parameters for exponential smoothing, as it was just run using other default ones, so there is potentially more room for improvement.

Takeaways

Prophet is proven to be quite effective at forecasting more typical trends such as for seasonal business sales and is quite good even with just using default parameters.

However, the case above has shown that the parameter tuning can be tricky, as slight changes in the priors lead to large differences in the predictions when dealing with such volatile data over such a long period of time.

Exponential smoothing is a tried and true method and here may be the safer pick, as it is easier to interpret and less computationally demanding than Prophet.

Instructions

Instructions follow those provided in the Markdown file

1. Build the Docker image
    a. $ bash driver.sh build
2. Start the Jupyter server
    a. $ bash driver.sh jupyter
3. Navigate to http://localhost:8888/ in a browser
    a. Open notebooks folder
        i. forecasting_take_home_data.xlsx must be in the same directory as forecasting.ipynb
    b. Open forecasting.ipynb
4. Run the notebook
5. Stop the container
    a. $ bash driver.sh stop

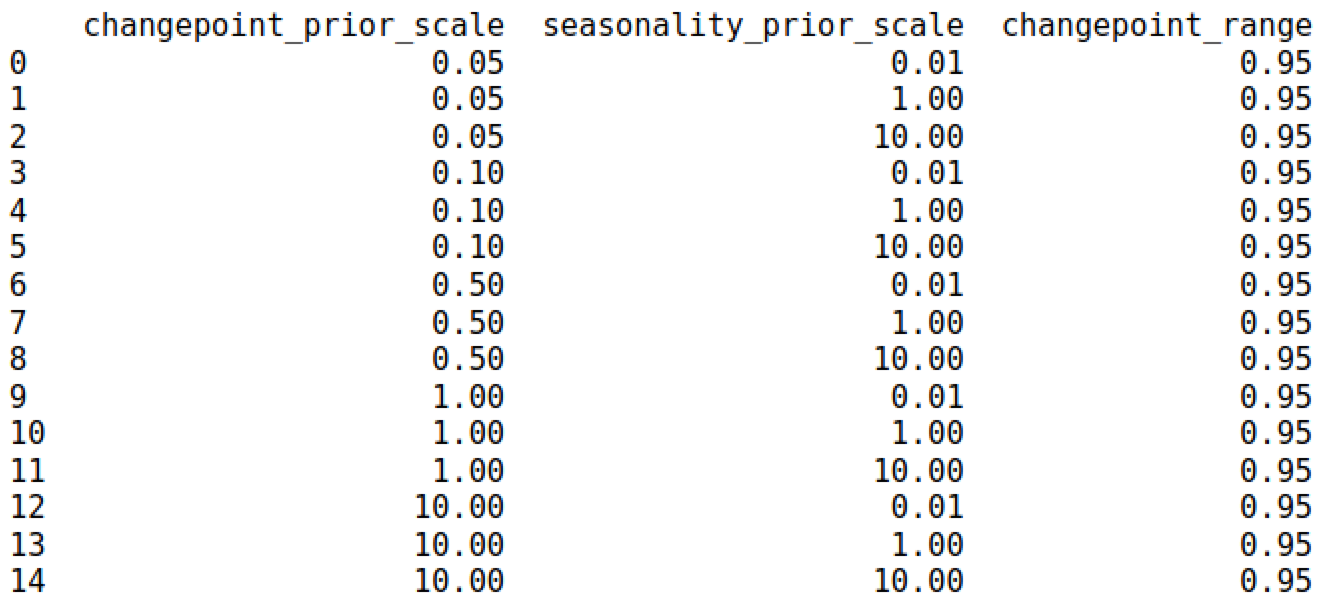
Appendix

Full Model CV Results

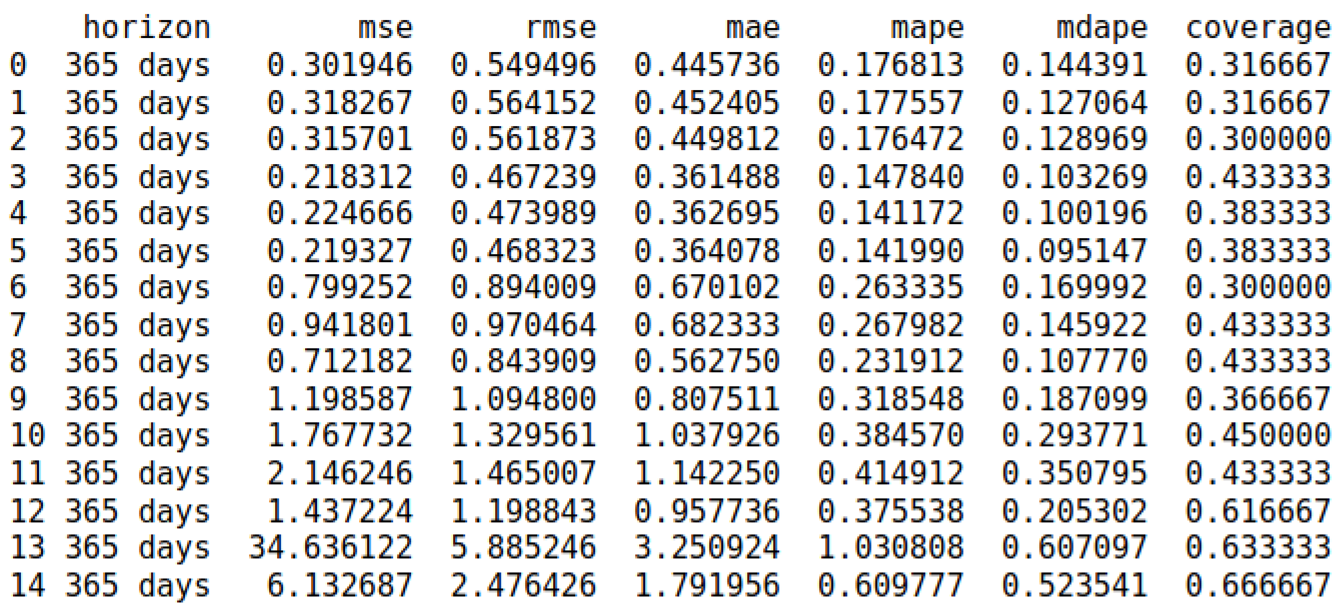
```
    changepoint_prior_scale  seasonality_prior_scale  changepoint_range
0                      0.05                     0.01               0.95
1                      0.05                     1.00               0.95
2                      0.05                    10.00               0.95
3                      0.10                     0.01               0.95
4                      0.10                     1.00               0.95
5                      0.10                    10.00               0.95
6                      0.50                     0.01               0.95
7                      0.50                     1.00               0.95
8                      0.50                    10.00               0.95
9                      1.00                     0.01               0.95
10                     1.00                     1.00               0.95
11                     1.00                    10.00               0.95
12                    10.00                     0.01               0.95
13                    10.00                     1.00               0.95
14                    10.00                    10.00               0.95
```

```
    horizon       mse      rmse       mae      mape     mdape  coverage
0  365 days  0.283580  0.532522  0.379674  0.183388  0.131322  0.314394
1  365 days  0.285157  0.534001  0.383570  0.185942  0.129376  0.291667
2  365 days  0.287158  0.535871  0.383956  0.186082  0.127675  0.291667
3  365 days  0.247637  0.497632  0.351387  0.166681  0.113532  0.295455
4  365 days  0.245876  0.495859  0.353280  0.170578  0.117401  0.246212
5  365 days  0.247126  0.497118  0.354077  0.170583  0.116346  0.257576
6  365 days  0.166050  0.407493  0.290845  0.143035  0.095796  0.367424
7  365 days  0.176157  0.419710  0.308028  0.157441  0.106777  0.344697
8  365 days  0.175153  0.418513  0.306178  0.156959  0.107266  0.344697
9  365 days  0.202474  0.449971  0.322214  0.156651  0.113054  0.382576
10 365 days  0.205187  0.452975  0.331586  0.165009  0.116120  0.359848
11 365 days  0.207350  0.455357  0.333062  0.165243  0.119518  0.359848
12 365 days  0.243273  0.493227  0.348419  0.170911  0.135627  0.428030
13 365 days  0.405623  0.636886  0.426220  0.240196  0.142146  0.431818
14 365 days  0.256539  0.506497  0.368462  0.188031  0.139652  0.450758
```

Subset (8 year) Model CV Results



```
    changepoint_prior_scale  seasonality_prior_scale  changepoint_range
0                      0.05                     0.01               0.95
1                      0.05                     1.00               0.95
2                      0.05                    10.00               0.95
3                      0.10                     0.01               0.95
4                      0.10                     1.00               0.95
5                      0.10                    10.00               0.95
6                      0.50                     0.01               0.95
7                      0.50                     1.00               0.95
8                      0.50                    10.00               0.95
9                      1.00                     0.01               0.95
10                     1.00                     1.00               0.95
11                     1.00                    10.00               0.95
12                    10.00                     0.01               0.95
13                    10.00                     1.00               0.95
14                    10.00                    10.00               0.95
```



```
    horizon        mse      rmse       mae      mape     mdape  coverage
0  365 days   0.301946  0.549496  0.445736  0.176813  0.144391  0.316667
1  365 days   0.318267  0.564152  0.452405  0.177557  0.127064  0.316667
2  365 days   0.315701  0.561873  0.449812  0.176472  0.128969  0.300000
3  365 days   0.218312  0.467239  0.361488  0.147840  0.103269  0.433333
4  365 days   0.224666  0.473989  0.362695  0.141172  0.100196  0.383333
5  365 days   0.219327  0.468323  0.364078  0.141990  0.095147  0.383333
6  365 days   0.799252  0.894009  0.670102  0.263335  0.169992  0.300000
7  365 days   0.941801  0.970464  0.682333  0.267982  0.145922  0.433333
8  365 days   0.712182  0.843909  0.562750  0.231912  0.107770  0.433333
9  365 days   1.198587  1.094800  0.807511  0.318548  0.187099  0.366667
10 365 days   1.767732  1.329561  1.037926  0.384570  0.293771  0.450000
11 365 days   2.146246  1.465007  1.142250  0.414912  0.350795  0.433333
12 365 days   1.437224  1.198843  0.957736  0.375538  0.205302  0.616667
13 365 days  34.636122  5.885246  3.250924  1.030808  0.607097  0.633333
14 365 days   6.132687  2.476426  1.791956  0.609777  0.523541  0.666667
```