Forecasting Trend in the CAISO Duck Curve using Seasonal Decomposition, ARIMA and SARIMAX

Artificial Intelligence & Machine Learning
Capstone Project, UC Berkeley
Mike Jones
12/26/2024

INTRODUCTION

Electrical Energy markets are a pervasive and a constant part of everybody's daily lives. We live in an advanced capitalist and industrial society that is interwoven with electrical energy and it's applications. The point has been reached where electricity is present everywhere, ubiquitous, and so ordinary that people hardly notice or talk about it. Electricity is constantly at work in the many devices, products, buildings, and vehicles all around us - in and among and throughout the daily activities of people going about their lives.

For most people in society - electricity just seems to work.

Yet there is a broad and deep story to be told about how electricity is created and arrives to us. There is an equilibrium between electrical energy production and consumption. The people who work in the industry (energy, utilities, and related logistics, etc) perform a continual balancing and rebalancing act in order to meet the changing energy needs of electricity customers throughout society.

Over the past decade, the innovation and scale of new ways to produce Solar and Wind energy, in combination with the advent of Smart Grids have created new challenges for those who operate and manage electrical power systems. Being able to accurately forecast the big picture trends in electrical power demand (especially with respect to clean energy and renewable power sources) is a vitally important capability for planners and operators in the world's largest energy markets. This project shows ways to produce short term 7-day forecasts for The Total Load in an electrical energy market as well as The Duck Curve (a concept that is introduced in greater detail below).

OUTLINE

This paper is written to accompany my Capstone Project for the UC Berkeley Artificial Intelligence and Machine Learning program (UCB AIML). It ties together and describes the central organizing theme of the many ideas that are represented in the project. These ideas are used to build code that meets the objective of forecasting trend in electrical energy markets.

This paper serves to summarize the motivation and context for why these ideas and techniques are important, as well as provide descriptions pointing out details that may not be immediately obvious when looking at either the code or it's output. The paper has a simple structure as described in the following outline:

[1] Introduction
[2] Outline
[3] CRISP-DM
[4] Business Understanding
[5] Data Understanding and Preparation
[6] Modeling and Evaluation
[7] Deployment
[8] Conclusion

See the Readme.md file for an additional guide to the collection of associated files in the project's GitHub directory.

CRISP-DM

The analysis was conducted according to the CRISP-DM framework (where CRISP-DM stands for Cross Industry Standard Process for Data Mining). At a high level, CRISP-DM is a cyclical set of activities which starts with establishing an initial analytic objective and forming a basic understanding. Then one works toward achieving that objective by continually moving through and refining the steps in the process to eventually produce a valuable result. The steps in the CRISP-DM Cycle are:

[1] Business Understanding
[2] Data Understanding
[3] Data Preparation
[4] Modeling
[5] Evaluation
[6] Deployment

In this particular assignment the steps [1] and [6] Business Understanding and Deployment (i.e. turning in the project) were established at the beginning and end of the project. The main concentration of effort was put toward multiple passes through the steps [2] - [5] of the cycle (Data Understanding, Preparation, Modeling, and Evaluation) in order to achieve the result.

BUSINESS UNDERSTANDING

The goal of this paper is to present techniques for determining the underlying trend of both The Total Load and The Duck Curve in an electrical energy markets. Once the trend is identified, it can then be used as a basis for making short term 7-day forecasts in order to project where the trend is likely to be in the near future. Accurately predicting near term load gives planners and managers the ability to operate systems more efficiently and effectively.

This work represents an initial foray, and early steps toward making these 7-day forecasts. Much was discovered and learned in the process of producing these preliminary results (both in the sense of machine learning as well as human learning). The work forms a foundation for ongoing research into this data set. It represents a basis for further development as well as potential business and industrial applications of the methods and findings.

There are nine ISOs (Independent System Operators) in North America, of which CAISO is one. CAISO stands for the California Independent System Operator, and is the organization which provides electrical power to most of California and parts of Southern Nevada. It is comprised of the four utility companies serving the region. They are:

PGE	Pacific Gas and Electric
SCE	Southern California Edison
SDGE	San Diego Gas and Electric
VAE	Valley Electric Association

This study explores the timeseries methods from Module 10 of the UCB AIML course in order to understand and make forecasts using two data sets related to CAISO's operational characteristics.

CAISO data is obtained from a company called GridStatus.io via API calls. Specifically, the two data sets under consideration are “Total Load” and “Fuel Mix”. Where the total load represents the system equilibrium (i.e the amount of electricity that is both produced and consumed), and is referred to as “The CAISO Load”. It is the total amount of energy that is both produced by electrical power generators and simultaneously consumed at a given time by CAISO’s customers. It is measured in megawatt hours (MWh) and the records in the data set are given on an hourly basis (i.e. at the top of the hour) for nearly six years of collected data spanning 2019 – 2024. Additionally, The Fuel Mix data set gives the amount of electricity generated by the various fuel sources which contribute to the overall production of electricity which satisfies and fulfills the total load at any given time. The data set contains energy production measurements for a given time and a given source. All of the Fuel Mix production adds up to the Total Load at a given time. Fuel Mix is also given in megawatt hours, with hourly data points over the same six year time period.

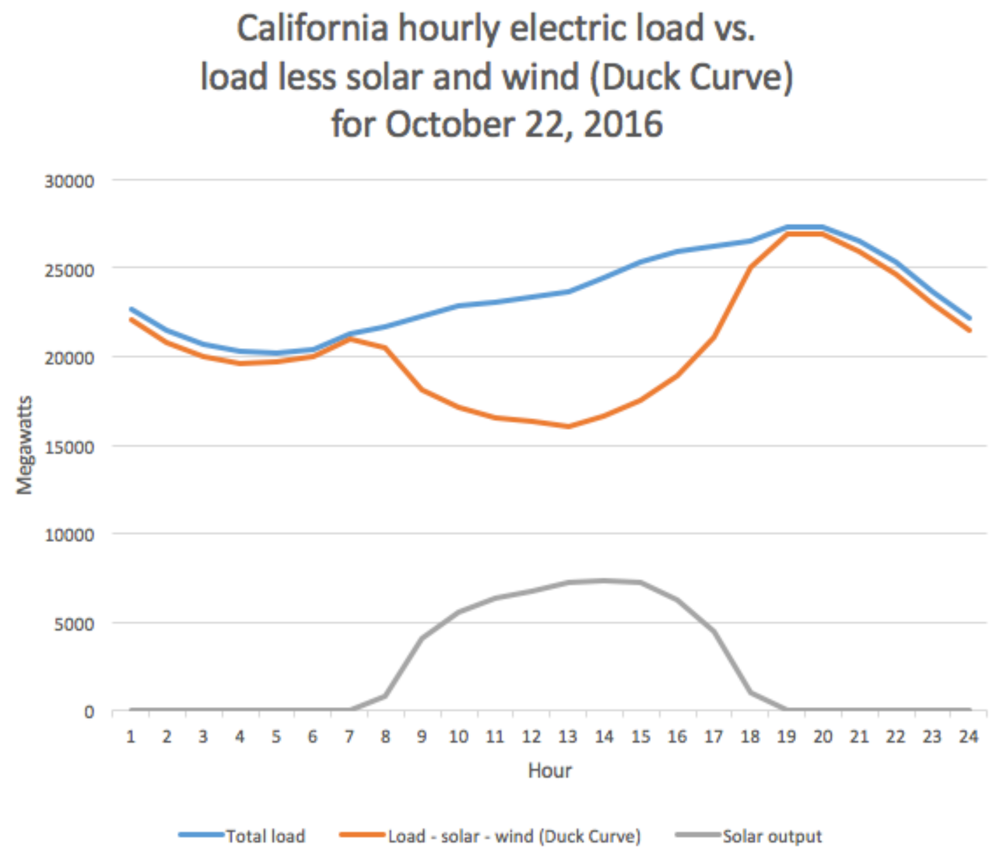
The Fuel Mix contains both traditional and renewable energy sources, where examples of traditional include: Hydro, Coal, Natural Gas, and Nuclear power. The most important renewable sources are Solar and Wind, which must be immediately consumed or stored, otherwise that power is permanently lost. Contrast that with traditional power sources, which can be moderated or regulated to meet changing demand. For example, by putting less coal into a furnace, that plant will produce less electricity. Conversely, by putting more coal into the furnace it will produce more electricity. Since there is no real penalty for feeding more or less coal into a furnace, adding more or subtracting less is a way to meet the changing total demand of the system when holding solar and wind power fixed. The same is similar for the other traditional energy sources. As such, the traditional sources are increased or decreased to meet the target for Total Load demand.

The same is not true of Solar or Wind in the sense that the amount available is what happens to be occurring at the moment due to the weather, time of day, and/or the seasons of the year. For example, If it is 12:00p noon on a hot and windy summer day with clear skies, that means there is a lot of Solar and Wind energy to convert into electricity. However if the weather conditions are overcast and dead calm with no wind – then there won’t be much electricity produced. Same is true at night – no Solar.

Solar and Wind have come to prominence in the 21st Century with innovation leading to increased reliability and scalability for renewable energy production. They are known as “Intermittent Renewable Energy Sources” (IRES), which means that Solar and Wind energy needs to be either consumed immediately or stored somehow, otherwise it is lost. For that reason, Solar and Wind can be subtracted from The Total Load to form what is called The Duck Curve which is the amount of energy required to be produced by all Non-Solar and Non-Wind fuel sources. It is called The Duck Curve, because when you look at the 24 hour graph, it somewhat resembles the silhouette of a duck. The equation for the Duck Curve is:

$$\text{Duck} = \text{Load} - (\text{Solar} + \text{Wind})$$

An example graph of The Duck Curve over a 24 hour period is given in the following image.



The Duck Curve is relevant because it represents the amount of energy required by non-renewable sources in order to satisfy customer demand and keep the load balanced. These sources undergo much larger and significant fluctuations during a shorter time period, and this puts greater strain on the plants, equipment and people to ramp energy production up and down through these large swings. Regardless, this is what must be done in order to accommodate bringing more Solar and Wind online. It is for this reason (and many others) that The Duck Curve is closely watched and observed by the people who make California's Electrical Power Market function.

The Duck Curve is calculated, and then its trend is determined using Seasonal Decomposition. ARIMA and SARIMAX are then applied to the trend in order to produce 7-day forecasts for both The Total CAISO Load as well as The Duck Curve. It attempts to predict the likely values across a future seven day window, and then compares the prediction against actual values across that same seven day window. Formally, the comparison is "scoring" the prediction against the test set, where the standard statistical procedures MSE, RMSE, MAE, and MAPE are used to compute the scores (see ARIMA section below for more details).

Additional background information on The Duck Curve can be found in the following articles:

https://aurorasolar.com/blog/the-duck-curve-a-review-of-californias-daily-load-predictions/
Understanding the California Duck Curve for Daily Load Projections

https://www.greentechmedia.com/articles/read/eia-charts-californias-real-and-growing-duck-curve
EIA Data Reveals California's Real and Growing Duck Curve

Additionally, Wikipedia has excellent introductory articles on Smart Grids and The Duck Curve and Electrical Systems in general:

https://en.wikipedia.org/wiki/Smart_grid
https://en.wikipedia.org/wiki/Duck_curve
https://en.wikipedia.org/wiki/Variable_renewable_energy
https://en.wikipedia.org/wiki/Electric_power_transmission
https://en.wikipedia.org/wiki/California_Independent_System_Operator

DATA UNDERSTANDING AND PREPARATION

The data for the capstone was obtained from the website / firm GridStatus.io in October 2024. It consists of two data sets: [1] Total CAISO Load and [2] Fuel Mix, the documentation for which can be found here:

https://www.gridstatus.io/datasets/caiso_load
https://www.gridstatus.io/datasets/caiso_fuel_mix

It is important to point out that the documentation appears to be an evolving aspect of the entire product offering from The Grid Status Company. Which is to say, that this is very much software under development and is subject to change or disappear at any time (which is exactly what happened to the availability of CAISO data via the API during the course of this project).

The documentation found on the site, continues to evolve and become more detailed and comprehensive in scope. Its appearance is becoming more corporate and I suspect that the company is rapidly growing and professionalizing their services. Whereas CAISO Load and Fuel Mix were previously freely available, they are now apparently behind a paywall requiring API keys in order to access.

Some aspects of the previously accessible data still remain freely available at the time of this writing. For example, for data is available for the NYISO (New York Independent System Operator). At the time of this writing, the following python code:

```
    import gridstatus
    nyiso = gridstatus.NYISO()

    gls0 = gridstatus.list_isos()
    print(f' gls0 { gls0 }')
    print(f' gridstatus.__version__ { gridstatus.__version__ }')

    load0 = nyiso.get_load("today")
    print(load0)

    fuel0 = caiso.get_fuel_mix("today")
    print(fuel0)
```

produces results, returning the following for list_isos():

```
    gls0                                     Name     Id  Class
    0                       Midcontinent ISO   miso   MISO
    1                         California ISO  caiso  CAISO
    2                                    PJM    pjm    PJM
    3  Electric Reliability Council of Texas  ercot  Ercot
    4                    Southwest Power Pool    spp    SPP
    5                           New York ISO  nyiso  NYISO
    6                        ISO New England  isone  ISONE
```

And, for NYISO specifically, the API call to get_load('today') returns:

```
    Name         Timestamp        Load     CAPITL          N.Y.C.     NORTH
    0    2024-12-26 00:00  17240.7136  1444.5430  ...  5250.8840  736.2094
    1    2024-12-26 00:05  17215.0717  1434.8036  ...  5236.6850  764.6216
    2    2024-12-26 00:10  17103.0779  1430.3510  ...  5190.7910  671.5379
    3    2024-12-26 00:15  16968.4203  1443.9225  ...  5189.7393  606.7305
    4    2024-12-26 00:20  16956.4011  1425.3980  ...  5190.6504  631.0508
    5    2024-12-26 00:25  16908.4444  1428.0290  ...  5178.6255  605.7811
    6    2024-12-26 00:30  16903.0264  1425.3310  ...  5173.7646  613.5656

    .                                                .
    .                                                .
    .                                                .

    265  2024-12-26 21:55  19748.2583  1708.4192  ...  5928.8710  614.9772
    266  2024-12-26 22:00  19637.9511  1709.2439  ...  5874.6700  612.7719
    267  2024-12-26 22:05  19526.3248  1677.2137  ...  5825.2130  616.1946
    268  2024-12-26 22:10  19309.8087  1655.7572  ...  5825.6910  599.9683
    269  2024-12-26 22:15  19252.6951  1672.3241  ...  5785.9175  609.4955
```

This result shows an abbreviated form the five minute incremental data for today's total load along with it's component utilities companies. In total there are 11 component utilities all contributing to the total NYISO load, where CAPITL, N.Y.C., and NORTH are shown in the table above. Although The Total Load for NYISO is currently available, similar API calls to NYISO fuel_mix('today') are not available at this time. When executing the fuel_mix() API call, the following error is returned:

```
urllib.error.HTTPError: HTTP Error 404: Not Found
```

This is the very same error received during the past month when making API calls to CAISO Load and Fuel Mix. The error seems unlikely to change anytime soon, without signing up for a paid subscription.

Fortunately in my initial Exploratory Data Analysis looking into the data set, I was able to collect and save nearly six years of data for both CAISO Load and Fuel Mix. The Python code I ran back in October which I used to create the files is:

```
caiso = gridstatus.CAISO()
start = pd.Timestamp("Jan 1, 2019").normalize()
end = pd.Timestamp.now().normalize()
fm1 = caiso.get_fuel_mix(start, end=end, verbose=True)
fm1.to_csv('fuel_mix_20190101_20241003.csv',index=False)

lod0  = caiso.get_load('2019-10-01',end='2024-10-03')
lod0.to_csv('load_20190101_20241003.csv',index=False)
```

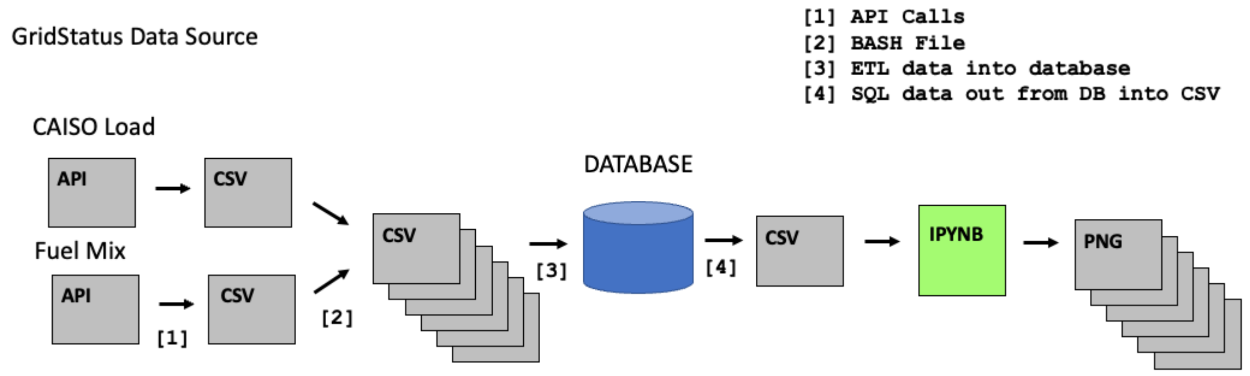
I saved the six years of data (at five minute sample intervals) aside in two CSV files, just to have it handy. The files are:

fuel_mix_20190101_20241003.csv
load_20190101_20241003.csv

The files are somewhat large. The fuel_mix file is 95.6 MB and has 605,091 records ranging from between 2019-01-01 and 2024-10-02. The load file is 45.3 MB has 526,486 records ranging from between 2019-10-01 and 2024-10-02. Working with files of this magnitude in combination was too much for the Jupyter Notebook and overwhelmed the Notebook to the point where it was too slow to get anything done. As such, I ended up using a Postgres Database to do preliminary data preparation work on the API raw data before bringing it into the Jupyter Notebook environment.

For this project I used my laptop and did all of the work locally. When I ran into the "Big Data like" issues, I decided to do some preprocessing outside of the notebook in order to reduce the data size to something more manageable. The following diagram is a drawing of the data flow topology showing the stages of data preparation prior to loading the data set into the Jupyter Notebook labeled IPYNB in green.

Data Flow Diagram - preparing the CAISO Data Set



The Data Preparations included:

[1] running the API Calls in a Python script,
[2] removing most incremental data using Bash script,
[3] loading data into the database using a Python script, and
[4] creating a csv file via the SQL script.

See the files in the directory "_other" for the Bash, ETL and SQL scripts used to reduce the total data size to something more manageable. Aspects of the Feature Engineering and Data Preparation were done in the database, including: filling in missing timestamps, joining the Load and Fuel_Mix tables, adding together Solar and Wind and subtracting both from the total load to calculate The Duck Curve:

Duck = Load - (Solar + Wind)

The results were saved to a CSV so that it could be loaded up to GitHub along with the Jupyter Notebook so as to form a consistent whole. As such, the code and the data could be distributed in such a way where nothing other than the Notebook and the CSV data was needed in order to duplicate and perform the calculations. That file is:

ucb_aiml_capstone_caiso.csv

and holds data in the following format, where the last full day of data is the following:

```
ts0                dt0       hr0   solar   wind   caiso_load   sol_wind   duck
2024-10-02 00:00   10/2/24    0     -56     917   26333          861      25472
2024-10-02 01:00   10/2/24    1     -59    1059   25552         1000      24552
2024-10-02 02:00   10/2/24    2     -60    1152   24259         1092      23167
2024-10-02 03:00   10/2/24    3     -59    1101   23123         1042      22081
2024-10-02 04:00   10/2/24    4     -59    1085   22757         1026      21731
2024-10-02 05:00   10/2/24    5     -61    1140   23152         1079      22073
2024-10-02 06:00   10/2/24    6     -60    1189   24497         1129      23368
```

```
2024-10-02 07:00   10/2/24    7      269   1245   25889           1514     24375
2024-10-02 08:00   10/2/24    8     7712   1044   26449           8756     17693
2024-10-02 09:00   10/2/24    9    14825    969   26858          15794     11064
2024-10-02 10:00   10/2/24   10    16580    906   26753          17486      9267
2024-10-02 11:00   10/2/24   11    16444    844   27269          17288      9981
2024-10-02 12:00   10/2/24   12    16695    801   28470          17496     10974
2024-10-02 13:00   10/2/24   13    16675    801   31104          17476     13628
2024-10-02 14:00   10/2/24   14    16467    824   33908          17291     16617
2024-10-02 15:00   10/2/24   15    16224   1033   37133          17257     19876
2024-10-02 16:00   10/2/24   16    15032   1619   39782          16651     23131
2024-10-02 17:00   10/2/24   17    10912   1740   40997          12652     28345
2024-10-02 18:00   10/2/24   18     1927   1877   41185           3804     37381
2024-10-02 19:00   10/2/24   19      -48   2295   39724           2247     37477
2024-10-02 20:00   10/2/24   20      -37   2407   37673           2370     35303
2024-10-02 21:00   10/2/24   21      -29   2451   35253           2422     32831
2024-10-02 22:00   10/2/24   22      -31   2727   33028           2696     30332
2024-10-02 23:00   10/2/24   23      -32   2772   30162           2740     27422
```

MODELING AND EVALUATION

The plan for modeling is to apply the following timeseries methods in the following order:

[1] Seasonal Decomposition,
[2] ARIMA, and
[3] SARIMAX.

Initially [1] Seasonal Decomposition is used for the purpose of establishing the component trend, seasonality, and residue of the original timeseries, which together comprise that given timeseries. Next, the identified trend will be used as input into the [2] ARIMA and [3] SARIMAX functions in order to produce 7-day forecasts on the trend. In the case of ARIMA, methods analogous to grid search and cross validation are performed on multiple forecasts that are made with varying inputs (arima_order (p,d,q) and arima period), and applied across several different date ranges. These are scored and evaluated so as to determine which parameters are most useful in forecasting the trend timeseries. The demonstration of SARIMAX however, is simply applied as-is to the trend in order to produce its 7-day forecasts. There was no further grid search or cross validation being performed in the SARIMAX context.

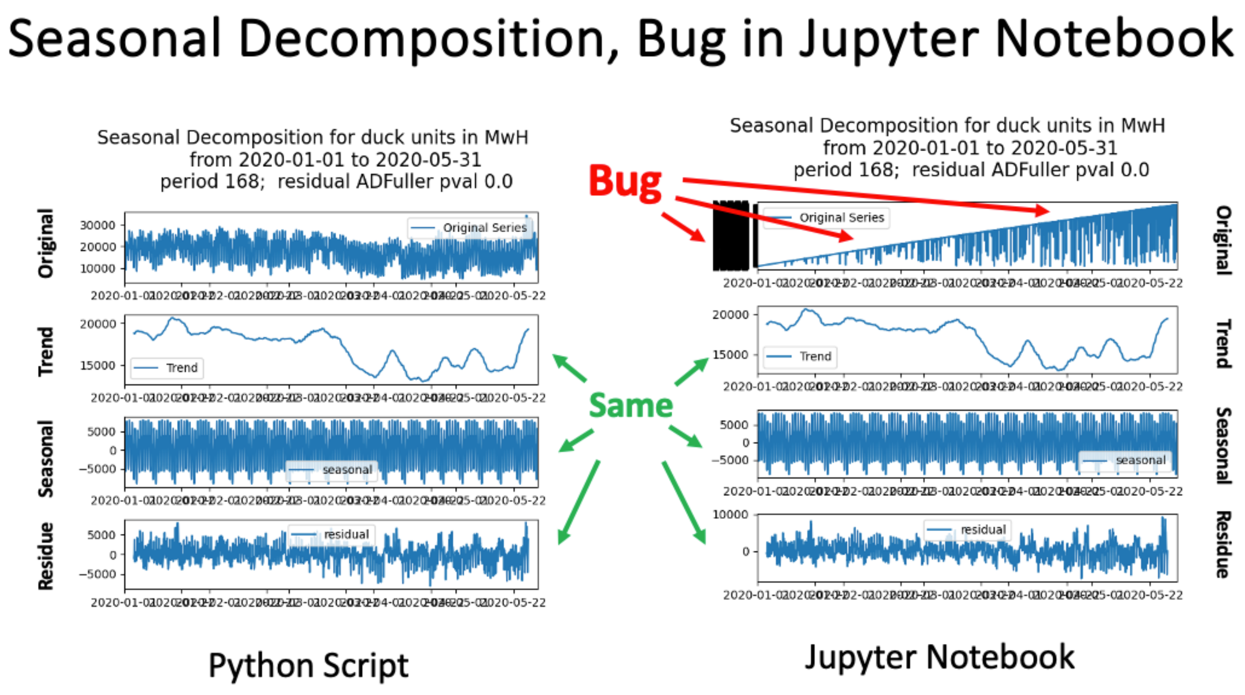
[1] Seasonal Decomposition is a way of separating out the components of a timeseries. The components are trend, seasonality and residue. Algebraically, they are represented as the following equations:

$y = t * s * r$ (multiplicative) , or
$y = t + s + r$ (additive)

where:

y is the original timeseries,
t is the trend,
s is the seasonal component, and
r is the residue.

Note that Seasonal Decomposition can be determined by using either *additive* or *multiplicative* combinations of the t, s and r factors. For the current project, *additive* was chosen for use as the initial results seemed most promising. Thus in the Jupyter IPYNB file, *additive* is hardcoded into the call to Seasonal Decomposition. It is worth mentioning that in future studies a more expansive search through the tunable hyperparameters could include testing both *additive* and *multiplicative* decomposition as well as varying the decomposition period in order to find the most valuable trends. It is also worth mentioning that initial work with Seasonal Decomposition was performed in a python script that was with data directly read in from the database. It turns out there is a bug in the Jupyter Notebook during the display of the original timeseries when using a Jupyter Notebook on data loaded in from the csv. Fortunately the bug only affects the image for the original timeseries in the topmost pane of the graph. The original timeseries itself is intact, as are the component trend, seasonal and residue timeseries. So the trend is ok to use for the purpose of forecasting. The bug appears to only affect the display of the original timeseries as can be seen in the graph below by comparing the Python vs Jupyter images. The effects of the bug can safely be ignored.



The Seasonal Decomposition Trend was determined for both CAISO Load and Duck Curve. Once the trend was isolated, it was then used for further analysis and forecasting by ARIMA and SARIMAX.

[2] ARIMA uses as input, the trend timeseries that was output from performing Seasonal Decomposition. The trend input for ARIMA is the basis for producing the 7-day forecasts. It should be noted that the 7-day forecasts themselves predict the trend and not the original / full CASIO Load or Duck Curve timeseries. Forecasting trend is a good first achievement, and relevant step toward understanding the data to the point where forecasts of the full original timeseries can be made.

Arima was performed using the StatsModels STLForecast method which takes an input timeseries, ARIMA parameters (p,d,q) and a period, where:

timeseries is the trend discovered in Seasonal Decomposition,
p is the autoregressive model order (num time lags for AR model),
d is for differencing (finite difference similar to a derivative in calculus),
q is the moving average model order (num coefficients in MA model), and
period represents seasonality removed by the STLForecast before applying ARIMA.

After a brief scan of the internet, it was found that StatsModels does not have a built in facility for conducting Grid Search on ARIMA as indicated by the following google search result:

> While scikit-learn's GridSearchCV doesn't directly support ARIMA models, you can still use it with a bit of customization.

As such, that methodology was "built by hand" in the cells of the notebook below. In the absence of a built in ARIMA Grid Search method, I decided to create my own model scoring mechanism for the purpose of optimizing hyperparameters. I performed my DIY GridSearch on ARIMA over three dimensions of hyperparameters:

[d1] arima order (p,d,q) as described above,
[d2] arima period for STL Forecast function, and
[d3] three dates marking training / test date ranges (sliding window).

A DIY (Do It Yourself) Exhaustive Search was used in lieu of the official SKLearn GridSearch, as nothing appears to have been explicitly built for ARIMA. Note that there is a built in GridSearch function available via SKLearn associated with SARIMAX (see below), yet after multiple attempts this functionality could not be made to work in my computing environment.

Also note that the sliding date window in dimension [d3] acts in a way that is similar to cross validation by training the model against different subsets of the timeseries, then making a prediction so that the prediction can be scored against known historical data for that same time period. While it is not strictly k-fold cross validation, it serves a similar purpose. For convenience, I will use the terms “grid search” and “exhaustive search” interchangeably in the rest of the paper, and I will just say “cross validation” when referring to the training/test sliding window.

The main calculation in the Jupyter Notebook relies on the machine learning concepts and principles of generating and testing each point in a hyperparameter search state space. Doing an exhaustive search through that space reveals hyperparameter which along with the trend timeseries become inputs into to the model. Next, one finds the most optimal hyperparameters that will produce the best resulting predictions by scoring the prediction against the actual history of the data set. By running the model across many hyperparameter input permutations and testing the forecast against the actual timeseries historical result, one can evaluate which predictions are most effective and which are not.

At a very high level, the full computation can be described as carrying out the following steps:

[1] Take a timeseries and find the trend using Seasonal Decomposition,
[2] Use that trend as input to STL Forecast ARIMA with (p,d,q) and period as hyper parameters,
[3] Foreach set of hyperparameters, generate several Training / Test split sub-timeseries,
[4] Train the model,
[5] Use the model for prediction,
[6] Score the prediction against the test (i.e. compare prediction to actual values),
[7] Tabulate the result for later comparison.

In the current project something analogous to cross validation is performed by creating a "Sliding Date Windows" that is implemented by taking subsets of the timeseries for the training / test split and moving those window segments down the timeline. The basic concept is to train and test the model by partitioning a timeseries into two subsets. These are known as training and test sequences. In the code these are defined by starting with the following dates:

d10 = 2020-01-10,
d11 = 2020-02-21, and
d12 = 2020-02-28.

Where d10 – d11 is the training set used to train the model which is then used to make the prediction, and d11 – d12 is the test set used to score the prediction. These dates are then advanced by five days each, thereby creating the "sliding window" effect across the timeseries in order to exercise the hyperparameters through a variety of the timeseries characteristics.

To summarize - I created a triply nested for loop to generate and scan through a three dimensional search space applying the hyperparameter combinations to the STLForecast ARIMA function. Each set of hyperparameters was applied to several training/test date segments and a prediction was found and scored.

For any combination of arima_order and STL_period, several 7-day forecasts are run and the accuracy of the forecasts are scored with:

Mean Square Error (MSE),

Root Mean Square Error (RMSE),
Mean Absolute Error (MAE), and
Mean Absolute Percentage Error (MAPE).

This triply nested for loop was used to produce 100 different model runs for both Caiso Load and Duck Curve. They are based on the first two dimensions [d1] & [d2] which form a 10x10 cross product / Cartesian grid denoted as X_Y where X is the arima_order, and Y is the STL_period. I give the column containing these values the name "IDX" for "index" in the result files as well as the excel file mentioned below. Then, once all of the Grid Search runs with the different hyperparameters are completed their result is collected are written out to the files:

result_caiso_load.txt , and
result_duck.txt

and then manually put into an excel file for further analysis:

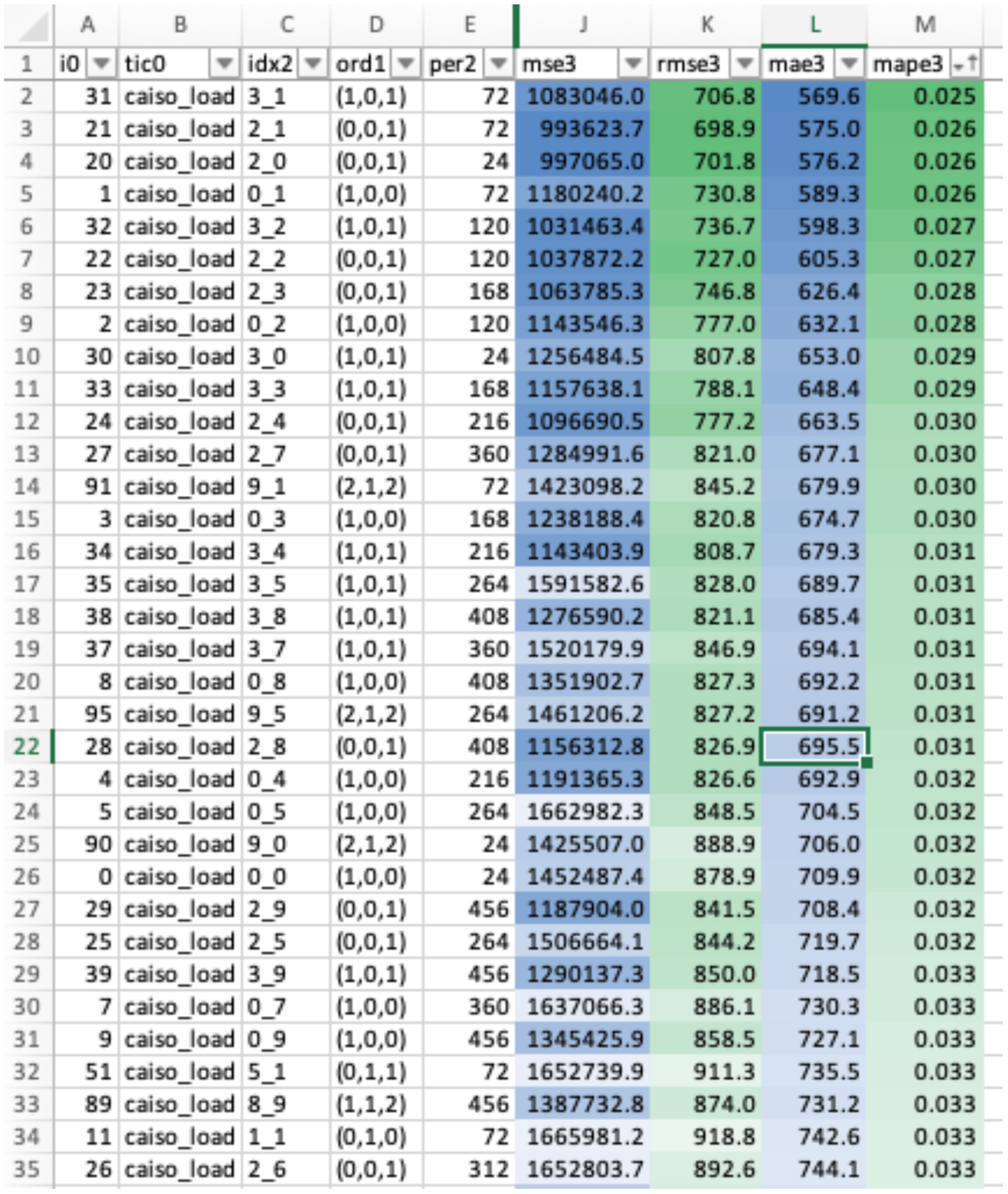
rpt_0_caiso_load_duck_matrix.xlsx

The file has three sheets:

caiso_scores
duck_scores
matrix

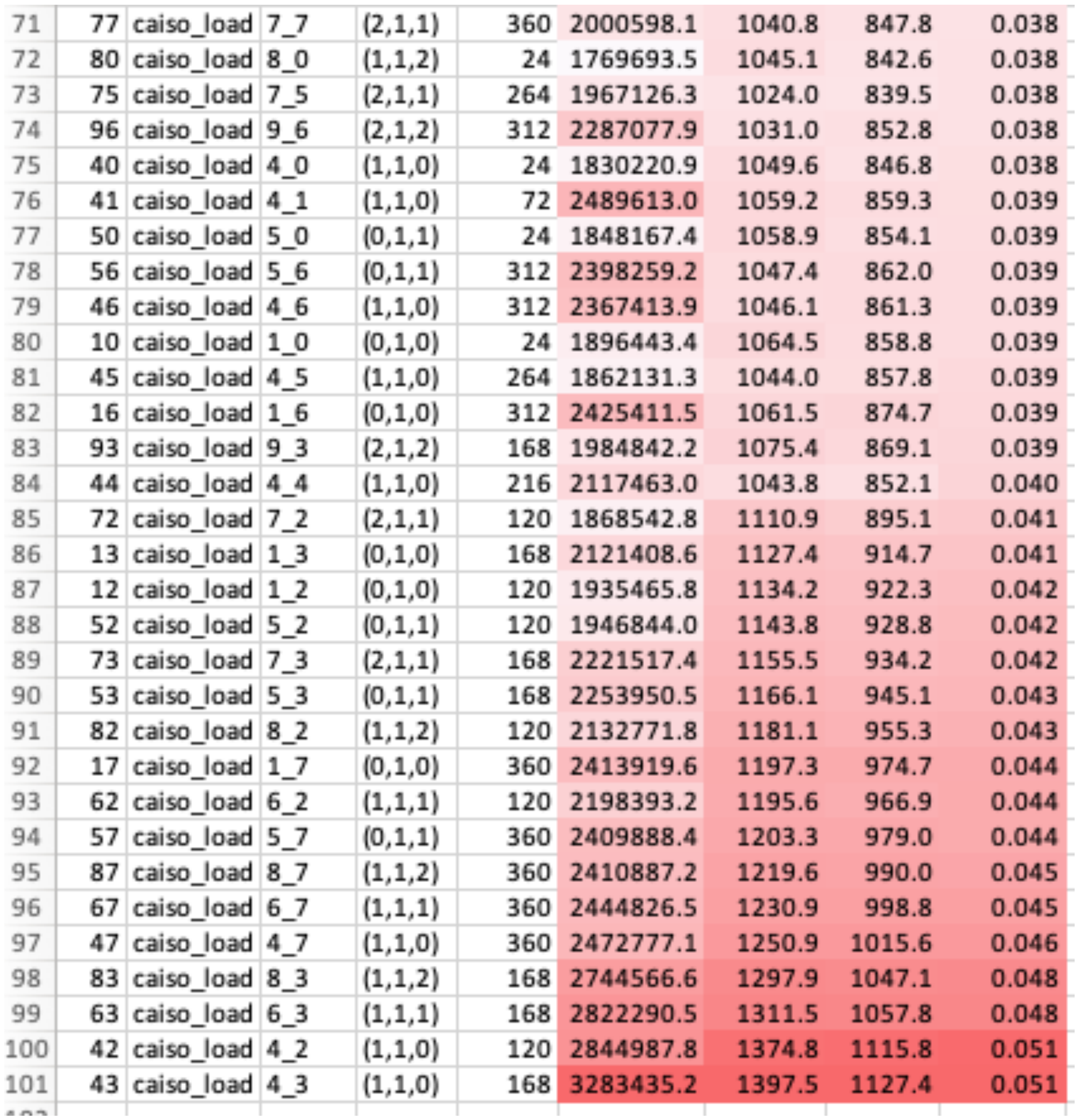
where the caiso_scores and duck_scores sheets correspond to the result_caiso_load.txt and result_duck.txt files.

The results can then be cross compared and evaluated by looking at which runs have lower or higher values for MSE, RMSE, MAE, and MAPE. The IDX's with lower scores indicate the most accurate forecasts, and the IDX's with higher scores indicate the least accurate forecasts. Scores for MSE, RMSE, MAE and MAPE are colored in from blue to red or green to red to show how that forecast performed. Here is what The CAISO LOAD results of the full run of 100 hyperparameters looks like in the excel file. They are sorted by the MAPE3 column (excel column M). By sorting on MAPE3, one can see the full spectrum of scores across all of the hyperparameter tests. Note that the scores for other measures such as MAE, RMSE and MSE are also correlated and similar but don’t produce the exact same ordering from best to worst.



	A	B	C	D	E	J	K	L	M
1	i0	tic0	idx2	ord1	per2	mse3	rmse3	mae3	mape3
2	31	caiso_load	3_1	(1,0,1)	72	1083046.0	706.8	569.6	0.025
3	21	caiso_load	2_1	(0,0,1)	72	993623.7	698.9	575.0	0.026
4	20	caiso_load	2_0	(0,0,1)	24	997065.0	701.8	576.2	0.026
5	1	caiso_load	0_1	(1,0,0)	72	1180240.2	730.8	589.3	0.026
6	32	caiso_load	3_2	(1,0,1)	120	1031463.4	736.7	598.3	0.027
7	22	caiso_load	2_2	(0,0,1)	120	1037872.2	727.0	605.3	0.027
8	23	caiso_load	2_3	(0,0,1)	168	1063785.3	746.8	626.4	0.028
9	2	caiso_load	0_2	(1,0,0)	120	1143546.3	777.0	632.1	0.028
10	30	caiso_load	3_0	(1,0,1)	24	1256484.5	807.8	653.0	0.029
11	33	caiso_load	3_3	(1,0,1)	168	1157638.1	788.1	648.4	0.029
12	24	caiso_load	2_4	(0,0,1)	216	1096690.5	777.2	663.5	0.030
13	27	caiso_load	2_7	(0,0,1)	360	1284991.6	821.0	677.1	0.030
14	91	caiso_load	9_1	(2,1,2)	72	1423098.2	845.2	679.9	0.030
15	3	caiso_load	0_3	(1,0,0)	168	1238188.4	820.8	674.7	0.030
16	34	caiso_load	3_4	(1,0,1)	216	1143403.9	808.7	679.3	0.031
17	35	caiso_load	3_5	(1,0,1)	264	1591582.6	828.0	689.7	0.031
18	38	caiso_load	3_8	(1,0,1)	408	1276590.2	821.1	685.4	0.031
19	37	caiso_load	3_7	(1,0,1)	360	1520179.9	846.9	694.1	0.031
20	8	caiso_load	0_8	(1,0,0)	408	1351902.7	827.3	692.2	0.031
21	95	caiso_load	9_5	(2,1,2)	264	1461206.2	827.2	691.2	0.031
22	28	caiso_load	2_8	(0,0,1)	408	1156312.8	826.9	695.5	0.031
23	4	caiso_load	0_4	(1,0,0)	216	1191365.3	826.6	692.9	0.032
24	5	caiso_load	0_5	(1,0,0)	264	1662982.3	848.5	704.5	0.032
25	90	caiso_load	9_0	(2,1,2)	24	1425507.0	888.9	706.0	0.032
26	0	caiso_load	0_0	(1,0,0)	24	1452487.4	878.9	709.9	0.032
27	29	caiso_load	2_9	(0,0,1)	456	1187904.0	841.5	708.4	0.032
28	25	caiso_load	2_5	(0,0,1)	264	1506664.1	844.2	719.7	0.032
29	39	caiso_load	3_9	(1,0,1)	456	1290137.3	850.0	718.5	0.033
30	7	caiso_load	0_7	(1,0,0)	360	1637066.3	886.1	730.3	0.033
31	9	caiso_load	0_9	(1,0,0)	456	1345425.9	858.5	727.1	0.033
32	51	caiso_load	5_1	(0,1,1)	72	1652739.9	911.3	735.5	0.033
33	89	caiso_load	8_9	(1,1,2)	456	1387732.8	874.0	731.2	0.033
34	11	caiso_load	1_1	(0,1,0)	72	1665981.2	918.8	742.6	0.033
35	26	caiso_load	2_6	(0,0,1)	312	1652803.7	892.6	744.1	0.033

36	79	caiso_load	7_9	(2,1,1)	456	1427458.1	885.3	737.0	0.033
37	99	caiso_load	9_9	(2,1,2)	456	1351802.9	881.2	738.0	0.033
38	71	caiso_load	7_1	(2,1,1)	72	1772500.4	931.5	751.3	0.034
39	88	caiso_load	8_8	(1,1,2)	408	1657443.1	919.2	752.1	0.034
40	78	caiso_load	7_8	(2,1,1)	408	1569806.5	915.4	754.9	0.034
41	98	caiso_load	9_8	(2,1,2)	408	1527894.9	909.2	753.7	0.034
42	81	caiso_load	8_1	(1,1,2)	72	1848457.9	941.4	762.2	0.034
43	48	caiso_load	4_8	(1,1,0)	408	1667630.5	935.9	763.4	0.034
44	92	caiso_load	9_2	(2,1,2)	120	1507888.6	951.0	767.4	0.034
45	49	caiso_load	4_9	(1,1,0)	456	1428919.9	906.9	764.3	0.034
46	74	caiso_load	7_4	(2,1,1)	216	1626072.9	912.6	756.9	0.034
47	61	caiso_load	6_1	(1,1,1)	72	1893618.2	950.9	770.8	0.035
48	36	caiso_load	3_6	(1,0,1)	312	1971568.1	936.1	777.2	0.035
49	94	caiso_load	9_4	(2,1,2)	216	1389383.6	914.6	757.3	0.035
50	68	caiso_load	6_8	(1,1,1)	408	1682411.0	942.1	770.7	0.035
51	69	caiso_load	6_9	(1,1,1)	456	1461518.8	915.6	771.8	0.035
52	6	caiso_load	0_6	(1,0,0)	312	2109208.6	957.3	794.7	0.036
53	59	caiso_load	5_9	(0,1,1)	456	1562771.3	936.5	789.0	0.036
54	19	caiso_load	1_9	(0,1,0)	456	1575994.9	941.3	793.5	0.036
55	14	caiso_load	1_4	(0,1,0)	216	1764863.8	960.4	797.2	0.037
56	58	caiso_load	5_8	(0,1,1)	408	1714253.4	978.6	811.4	0.037
57	97	caiso_load	9_7	(2,1,2)	360	1890435.7	1000.5	817.6	0.037
58	15	caiso_load	1_5	(0,1,0)	264	1945235.7	983.3	808.7	0.037
59	18	caiso_load	1_8	(0,1,0)	408	1718884.1	987.5	819.8	0.037
60	84	caiso_load	8_4	(1,1,2)	216	1852516.2	973.3	801.0	0.037
61	54	caiso_load	5_4	(0,1,1)	216	1798587.2	971.0	805.3	0.037
62	55	caiso_load	5_5	(0,1,1)	264	1931372.4	992.9	817.0	0.037
63	64	caiso_load	6_4	(1,1,1)	216	1875438.0	977.8	803.9	0.037
64	86	caiso_load	8_6	(1,1,2)	312	2276948.3	1015.3	835.3	0.038
65	66	caiso_load	6_6	(1,1,1)	312	2291053.4	1016.7	836.8	0.038
66	70	caiso_load	7_0	(2,1,1)	24	1760883.1	1034.0	833.2	0.038
67	85	caiso_load	8_5	(1,1,2)	264	1873611.4	1016.6	832.3	0.038
68	76	caiso_load	7_6	(2,1,1)	312	2222705.1	1021.9	845.2	0.038
69	65	caiso_load	6_5	(1,1,1)	264	1881959.9	1020.2	835.0	0.038
70	60	caiso_load	6_0	(1,1,1)	24	1757095.3	1041.2	839.1	0.038

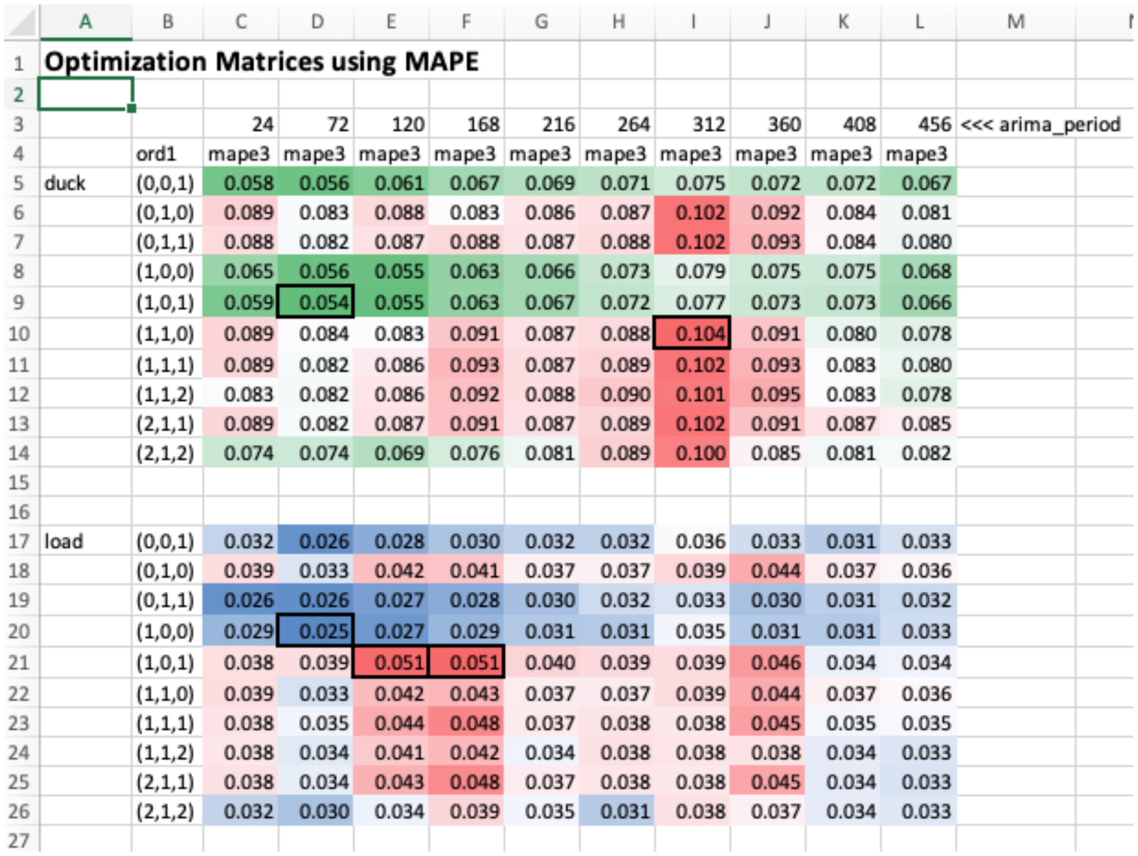


71	77	caiso_load	7_7	(2,1,1)	360	2000598.1	1040.8	847.8	0.038
72	80	caiso_load	8_0	(1,1,2)	24	1769693.5	1045.1	842.6	0.038
73	75	caiso_load	7_5	(2,1,1)	264	1967126.3	1024.0	839.5	0.038
74	96	caiso_load	9_6	(2,1,2)	312	2287077.9	1031.0	852.8	0.038
75	40	caiso_load	4_0	(1,1,0)	24	1830220.9	1049.6	846.8	0.038
76	41	caiso_load	4_1	(1,1,0)	72	2489613.0	1059.2	859.3	0.039
77	50	caiso_load	5_0	(0,1,1)	24	1848167.4	1058.9	854.1	0.039
78	56	caiso_load	5_6	(0,1,1)	312	2398259.2	1047.4	862.0	0.039
79	46	caiso_load	4_6	(1,1,0)	312	2367413.9	1046.1	861.3	0.039
80	10	caiso_load	1_0	(0,1,0)	24	1896443.4	1064.5	858.8	0.039
81	45	caiso_load	4_5	(1,1,0)	264	1862131.3	1044.0	857.8	0.039
82	16	caiso_load	1_6	(0,1,0)	312	2425411.5	1061.5	874.7	0.039
83	93	caiso_load	9_3	(2,1,2)	168	1984842.2	1075.4	869.1	0.039
84	44	caiso_load	4_4	(1,1,0)	216	2117463.0	1043.8	852.1	0.040
85	72	caiso_load	7_2	(2,1,1)	120	1868542.8	1110.9	895.1	0.041
86	13	caiso_load	1_3	(0,1,0)	168	2121408.6	1127.4	914.7	0.041
87	12	caiso_load	1_2	(0,1,0)	120	1935465.8	1134.2	922.3	0.042
88	52	caiso_load	5_2	(0,1,1)	120	1946844.0	1143.8	928.8	0.042
89	73	caiso_load	7_3	(2,1,1)	168	2221517.4	1155.5	934.2	0.042
90	53	caiso_load	5_3	(0,1,1)	168	2253950.5	1166.1	945.1	0.043
91	82	caiso_load	8_2	(1,1,2)	120	2132771.8	1181.1	955.3	0.043
92	17	caiso_load	1_7	(0,1,0)	360	2413919.6	1197.3	974.7	0.044
93	62	caiso_load	6_2	(1,1,1)	120	2198393.2	1195.6	966.9	0.044
94	57	caiso_load	5_7	(0,1,1)	360	2409888.4	1203.3	979.0	0.044
95	87	caiso_load	8_7	(1,1,2)	360	2410887.2	1219.6	990.0	0.045
96	67	caiso_load	6_7	(1,1,1)	360	2444826.5	1230.9	998.8	0.045
97	47	caiso_load	4_7	(1,1,0)	360	2472777.1	1250.9	1015.6	0.046
98	83	caiso_load	8_3	(1,1,2)	168	2744566.6	1297.9	1047.1	0.048
99	63	caiso_load	6_3	(1,1,1)	168	2822290.5	1311.5	1057.8	0.048
100	42	caiso_load	4_2	(1,1,0)	120	2844987.8	1374.8	1115.8	0.051
101	43	caiso_load	4_3	(1,1,0)	168	3283435.2	1397.5	1127.4	0.051

Color was added to the cells in order to accentuate the scores. Strongly blue or green cells indicate the best relative scores in the column and strongly red cells indicates the least accurate hyper parameters.

The matrices below show a condensed picture of the MAPE Scores in a tabular format with rows indicating arima_order and the colums indicate the STL_period. The best and worst MAPE scores are identified in the matrices with thick dark squares on those cells. They are:

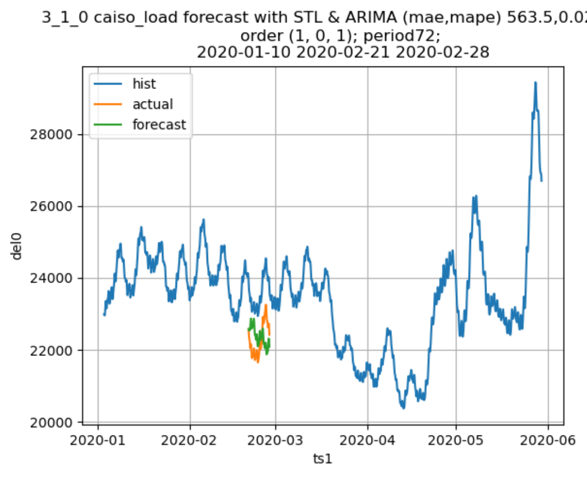
	Best	Worst
Duck	0.054	0.104
Load	0.025	0.051

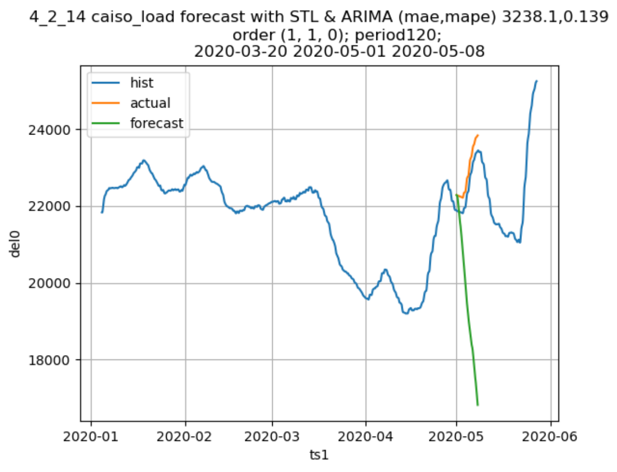


Optimization Matrices using MAPE

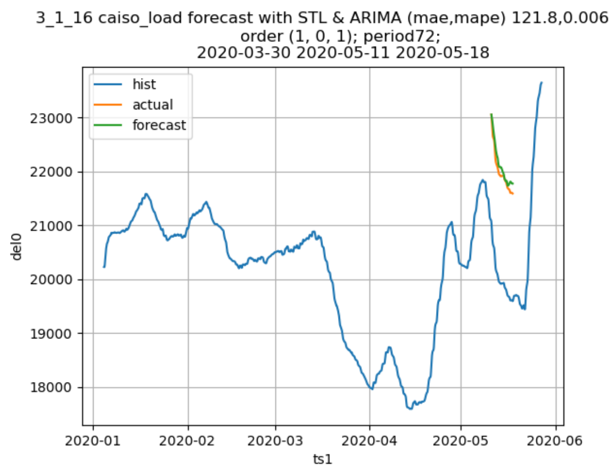
		24	72	120	168	216	264	312	360	408	456	<<< arima_period
	ord1	mape3	mape3	mape3	mape3	mape3	mape3	mape3	mape3	mape3	mape3	
duck	(0,0,1)	0.058	0.056	0.061	0.067	0.069	0.071	0.075	0.072	0.072	0.067	
	(0,1,0)	0.089	0.083	0.088	0.083	0.086	0.087	0.102	0.092	0.084	0.081	
	(0,1,1)	0.088	0.082	0.087	0.088	0.087	0.088	0.102	0.093	0.084	0.080	
	(1,0,0)	0.065	0.056	0.055	0.063	0.066	0.073	0.079	0.075	0.075	0.068	
	(1,0,1)	0.059	0.054	0.055	0.063	0.067	0.072	0.077	0.073	0.073	0.066	
	(1,1,0)	0.089	0.084	0.083	0.091	0.087	0.088	0.104	0.091	0.080	0.078	
	(1,1,1)	0.089	0.082	0.086	0.093	0.087	0.089	0.102	0.093	0.083	0.080	
	(1,1,2)	0.083	0.082	0.086	0.092	0.088	0.090	0.101	0.095	0.083	0.078	
	(2,1,1)	0.089	0.082	0.087	0.091	0.087	0.089	0.102	0.091	0.087	0.085	
	(2,1,2)	0.074	0.074	0.069	0.076	0.081	0.089	0.100	0.085	0.081	0.082	
load	(0,0,1)	0.032	0.026	0.028	0.030	0.032	0.032	0.036	0.033	0.031	0.033	
	(0,1,0)	0.039	0.033	0.042	0.041	0.037	0.037	0.039	0.044	0.037	0.036	
	(0,1,1)	0.026	0.026	0.027	0.028	0.030	0.032	0.033	0.030	0.031	0.032	
	(1,0,0)	0.029	0.025	0.027	0.029	0.031	0.031	0.035	0.031	0.031	0.033	
	(1,0,1)	0.038	0.039	0.051	0.051	0.040	0.039	0.039	0.046	0.034	0.034	
	(1,1,0)	0.039	0.033	0.042	0.043	0.037	0.037	0.039	0.044	0.037	0.036	
	(1,1,1)	0.038	0.035	0.044	0.048	0.037	0.038	0.038	0.045	0.035	0.035	
	(1,1,2)	0.038	0.034	0.041	0.042	0.034	0.038	0.038	0.038	0.034	0.033	
	(2,1,1)	0.038	0.034	0.043	0.048	0.037	0.038	0.038	0.045	0.034	0.033	
	(2,1,2)	0.032	0.030	0.034	0.039	0.035	0.031	0.038	0.037	0.034	0.033	

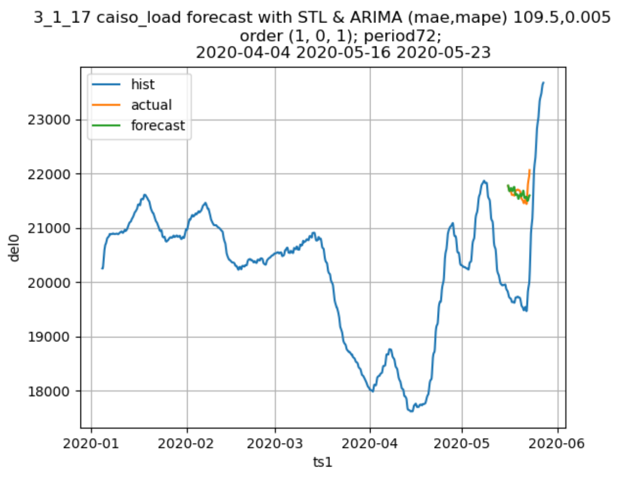
Both good and bad examples can be drawn from the best and worst performing forecasts as determined by the exhaustive search hyperparameter optimization. The following two images present examples illustrating both good (Left) and bad (Right) forecasts. The image on the Left Hand Side (LHS) shows a good / accurate forecast having low MAE and MAPE scores printed at the top. The 7-day forecast line segment is drawn in green and the actual historical values for that same date range are given in orange. These segments are nearly overlapping and their close proximity to one and other indicates a good forecast. Note that the green and orange line segments are slightly vertically offset from the original blue historical timeseries. This is due to a tricky aspect of plotting all three lines simultaneously. Compare and contrast the good forecast on the LHS to the bad forecast in the image on the Right Hand Side (RHS). Here the green forecast and orange actual history lines immediately and significantly diverge. Green is seen to shoot straight down sharply while orange moves in the opposite directly, climbing upward quickly. The MAE and MAPE scores are both very high in this case, indicating how bad this forecast turned out to be.



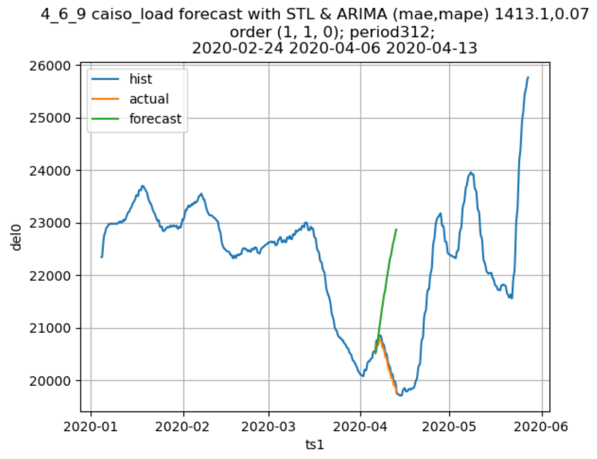


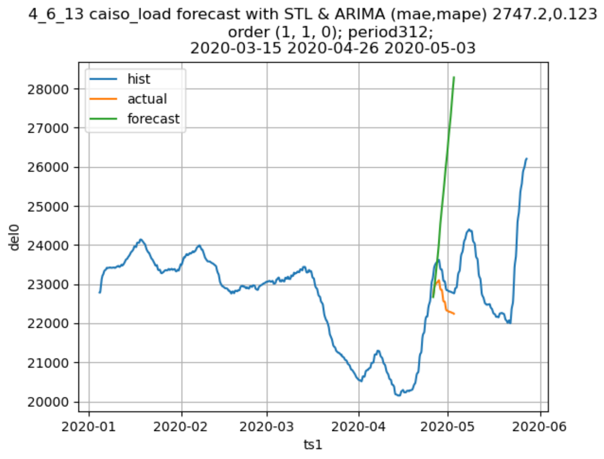
Further examples of the best forecasts have relatively low MAE and MAPE scores, and the forecast / actual segments appear to converge or overlap each other, one on top of another, as in the following two images.





The following two images are examples of some of the worst performing forecasts. They have relatively high MAE and MAPE scores indicating a very low and poor forecast accuracy. The green and orange lines clearly deviate from one and other.





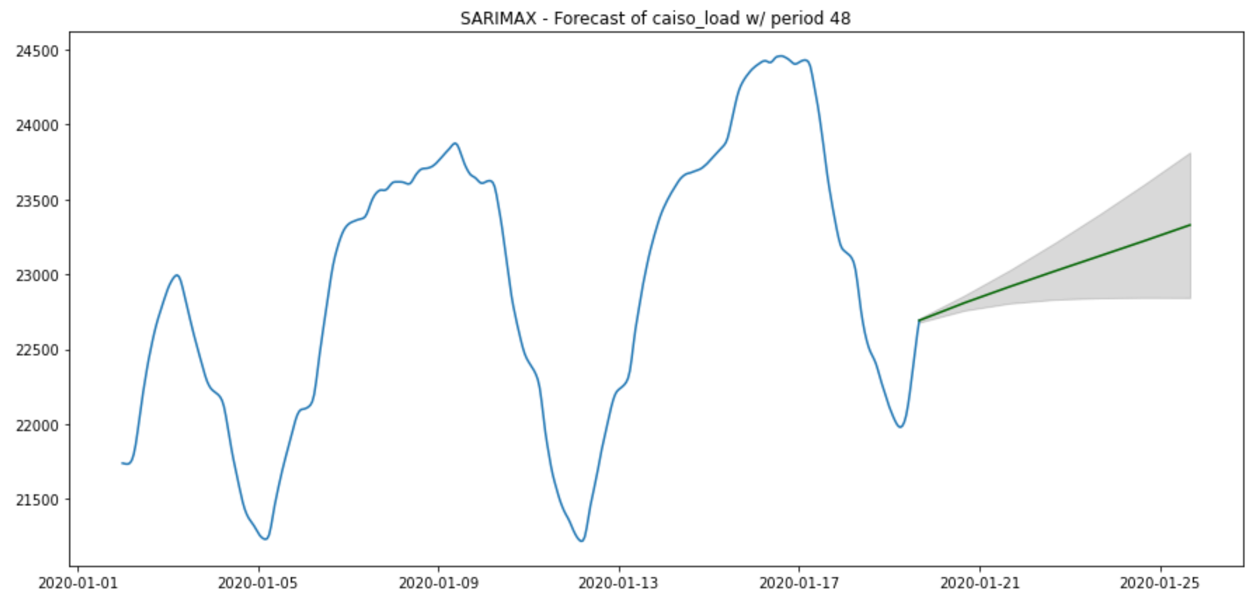
[3] SARIMAX was last but not least. The SARIMAX forecasts were implemented and the results for both Total Load and Duck can be seen below. In the end it was enough to simply produce the forecasts themselves.

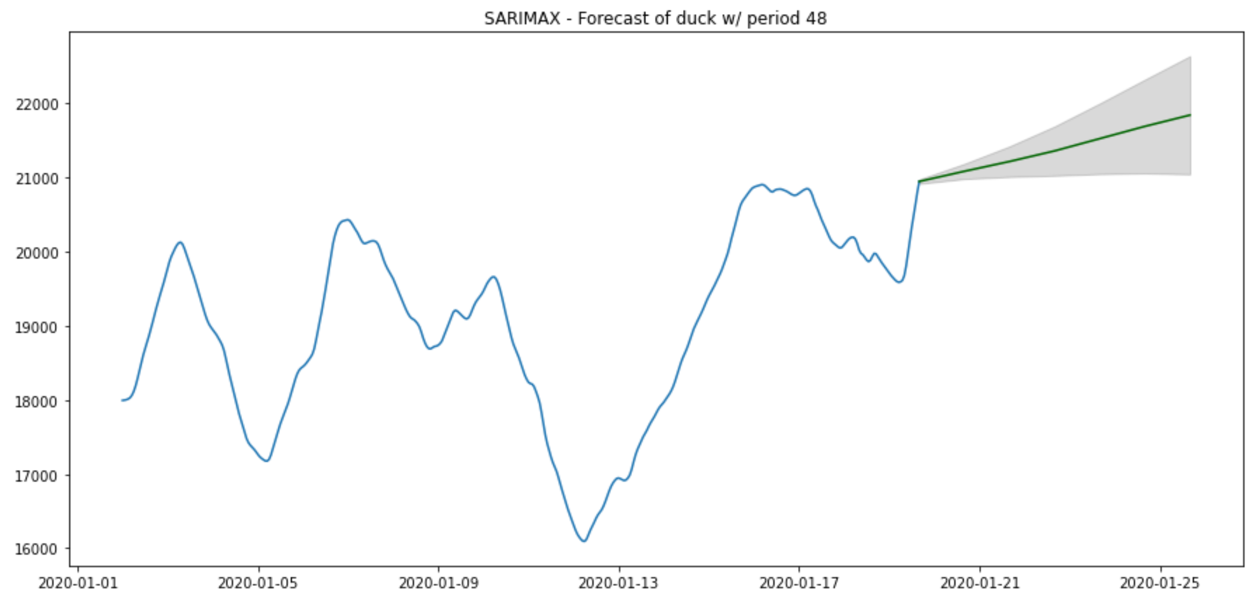
After a brief scan of the internet, existing code for GridSearch on SARIMAX was found at https://skforecast.org/0.4.2/guides/grid-search-sarimax-arima , namely the function:

    skforecast.model_selection_statsmodels.grid_search_sarimax

A great deal of time and effort was spent attempting to get SARIMAX Grid Search going, however for whatever reason, all of these attempts did not meet with a successful result.

The SARIMAX Forecasts for Total CAISO Load and Duck Curve predict a continuation of the average with a slight upward trend.





DEPLOYMENT

In the context of academic schoolwork, the deployment is simply to upload and submit the assignment and it's findings to GitHub for grading. However, in an industrial or commercial setting, the result could be deployed into production operations so that it can be run over and over again for continual and constant updating. Newly available data and forecasts can be put into reporting dashboards and other decision support systems. Thus, near real time data could be kept as up-to-date as possible for planners and decision makers. It is possible to envision an alerting or monitoring system that could be programmed to detect changing conditions and recommend a course of action based on the data and/or analysis.

Additionally the results and findings need to be ”deployed” in a social context. This is to say that the findings and insights would need to be communicated throughout the organization to relevant and interested parties. The RACI Matrix comes to mind (RACI standing for Responsible, Accountable, Consulted and Informed) as a guide to identifying a target audience who can make use of the 7-day Load and Duck forecasts. Using RACI as a guide, some would be responsible to produce and distribute forecasts, while others would be accountable to either produce or consume the forecasts in the act of carrying out their duties. Still others would need to be consulted or informed based on their roles and responsibilities. The existence of the forecasts would need to be announced as part of a deployment roll out, and making available the findings to decision makers, as well as leaders across the organization would help to inform managers, line workers and equipment operators as they conduct their work activities.

CONCLUSION

This project studied historical timeseries data from the CAISO Electrical Energy market for the purpose of making accurate forecasts. Specifically, a method has been shown to establish making 7-day forecasts on the underlying trends identified The Total Load consumed by the market as well as The Duck Curve. The Duck Curve is a recent phenomena over the past fifteen years with increasing significance as more and more electricity is generated by Solar and Wind sources. Note that Solar and Wind must either be consumed immediately, stored (e.g. put into batteries), or lost. Thus, by subtracting Solar and Wind from The Total Load, one produces The Duck Curve, which planners and operators take into account when managing the total production from all power sources in order to meet the total demand across the entire market.

The project shows a procedure to construct forecasts by combining three techniques. First Seasonal Decomposition is performed on the data in order to identify an underlying trend for both The Total Load and The Duck Curve. Then, ARIMA and SARIMAX are used on those trends in order to make 7-day forecasts. A method for exhaustive search was demonstrated (similar to Grid Search and K-fold Cross Validation) to vary the model hyper parameter inputs. These were then used to build and run the models across different segments of the timeseries to simulate and produce many different forecasts. Next these forecasts were summarized and compared with actual historical results by scoring with MSE, RMSE, MAE and MAPE. These were sorted to create a spectrum from best to worst performance (i.e. from most to least accurate forecasts).

The findings can be applied in a commercial or industrial setting for the purposes of planning and making operations more efficient and effective.