CSE 551 HW1 Report

By Group 11
Akanksha Kale (SBU ID: 113788594)
Akhila Juturu (SBU ID: 114777498)
Mandar Laxmikant Mahajan (SBU ID: 113276053)

Data preparation and cleaning

CSV data was provided for energy usage history for three homes: 'Home 1', 'Home 2' and 'Home 3'. Along with this, weather history data for each home is provided as well. We have primarily used Pandas dataframes in this assignment to read in, store and process the data.

We performed the following data cleansing steps:

Check for missing / null values

The 'cloudCover' column for Home1_weather has 1593, 2500 and 3540 missing values respectively.

```
home1_weather.isna().sum()
```

```
temperature                0
icon                       0
humidity                   0
visibility                 0
summary                    0
apparentTemperature        0
pressure                   0
windSpeed                  0
cloudCover              1593
time                       0
windBearing                0
precipIntensity            0
dewPoint                   0
precipProbability          0
dtype: int64
```

Removing cloudCover as it has 1593 null values

```
[ ]  home1_weather = home1_weather.drop(columns=['cloudCover'])
```

Checking for null values

```
home2_weather.isna().sum()
```

```
temperature               0
icon                      0
humidity                  0
visibility                0
summary                   0
apparentTemperature       0
pressure                  2
windSpeed                 3
cloudCover             2500
time                      0
windBearing               3
precipIntensity           0
dewPoint                  0
precipProbability         0
dtype: int64
```

Dropping cloudCover as 2500 values are missing

```
[ ] home2_weather = home2_weather.drop(columns=['cloudCover'])
```

```
home3_weather.isna().sum()
```

```
temperature               0
icon                      0
humidity                  0
visibility                0
summary                   0
apparentTemperature       0
pressure                  1
windSpeed                 2
cloudCover             3540
time                      0
windBearing               3
precipIntensity           0
dewPoint                  0
precipProbability         0
dtype: int64
```

```
[14] home3_weather = home3_weather.drop(columns=['cloudCover'])
```

As such a high percentage of rows are missing this feature, we have removed this column.

Check for duplicates

Home2_meter has 16 entries with duplicate timestamps. We removed the duplicates.

```
home2_meter[home2_meter.duplicated(subset = 'Date & Time', keep=False)].count()
```

```
Date & Time                 16
Usage [kW]                  16
Generation [kW]             16
WaterHeater [kW]            16
Solar [kW]                  16
Refrigerator [kW]           16
Microwave [kW]              16
Furnace [kW]                16
WaterHeater3 [kW]           16
WaterHeater2 [kW]           16
WaterHeater1 [kW]           16
Master_Bdrm [kW]            16
Front_Bdrm [kW]             16
Kit_StoveWall [kW]          16
Dishwasher_Disposal [kW]    16
Kit_SinkWall [kW]           16
Family_Rm [kW]              16
Kit_Half-Bath_Foyer [kW]    16
Washing_Machine [kW]        16
Guest_Bdrm_SmkDet [kW]      16
Dryer [kW]                  16
Basement [kW]               16
Phase_B [kW]                16
Phase_A [kW]                16
dtype: int64
```

Dropping rows with duplicate timestamps

```
home2_meter.drop_duplicates(subset = 'Date & Time', keep='first', inplace=True)
```

Home2_weather does not have duplicate values.

Remove / combine columns

In Home1_meter, values for 'use [kW]' column were exactly the same as 'House overall [kW]' column. Similarly, values for 'gen [kW]' and 'Solar [kW]' were exactly the same.

```
home1_meter['use [kW]'].equals(home1_meter['House overall [kW]'])
```

```
True
```

```
home1_meter['gen [kW]'].equals(home1_meter['Solar [kW]'])
```

```
True
```

We dropped the columns with identical data.

```
home1_meter=home1_meter.drop(columns=['use [kW]', 'gen [kW]'])
```

The above was repeated for Home 2 and Home 3.

To reduce the number of features in Home1_meter, we combined the columns for furnace energy consumption into a single 'Furnace [kW]' column. Similarly, we combined the kitchen energy consumption columns into a single 'Kitchen [kW]' column.

```
merged_daily['Furnace [kW]'] = merged_daily['Furnace 1 [kW]'] + merged_daily['Furnace 2 [kW]']
merged_daily['Kitchen [kW]'] = merged_daily['Kitchen 12 [kW]'] + merged_daily['Kitchen 14 [kW]'] + merged_daily['Kitchen 38 [kW]']
merged_daily = merged_daily.drop(columns=['Kitchen 12 [kW]','Kitchen 14 [kW]','Kitchen 38 [kW]','Furnace 2 [kW]','Furnace 1 [kW]'])
```

Under Home1_weather, the columns for ‘summary’ and ‘icon’ communicate similar information:

```
[ ]  home1_weather.summary.unique()

     array(['Clear', 'Partly Cloudy', 'Flurries', 'Light Snow', 'Overcast',
            'Foggy', 'Drizzle', 'Light Rain', 'Rain', 'Snow', 'Mostly Cloudy',
            'Heavy Snow', 'Breezy', 'Breezy and Partly Cloudy',
            'Rain and Breezy', 'Breezy and Overcast', 'Heavy Rain',
            'Light Rain and Breezy', 'Breezy and Mostly Cloudy'], dtype=object)

[ ]  home1_weather.icon.unique()

     array(['clear-night', 'clear-day', 'partly-cloudy-night', 'snow',
            'cloudy', 'fog', 'rain', 'partly-cloudy-day', 'wind'], dtype=object)
```

To avoid duplicate features, and as summary column provides more insights, we decided to drop the ‘icon’ column:

We decided to drop the icon column as summary column gives more insights

```
home1_weather=home1_weather.drop(columns=['icon'])
```

As the summary column has categorical data, we need to encode it.

Summary column has categorical data so we want to encode it

```
[ ]  le = preprocessing.LabelEncoder()

[ ]  home1_weather['summary'] = le.fit_transform(home1_weather.summary.values)
     home1_weather
```

	temperature	humidity	visibility	summary	apparentTemperature	pressure	windSpeed	time	windBearing	precipIntensity	dewPoint	precipProbability
0	20.28	0.47	10.00	4	7.41	1023.26	13.21	2014-01-01 00:00:00	284	0.0	3.46	0.0
1	19.28	0.48	10.00	4	7.91	1023.27	10.07	2014-01-01 01:00:00	284	0.0	2.98	0.0
2	18.35	0.52	10.00	4	7.90	1024.32	8.43	2014-01-01 02:00:00	258	0.0	3.77	0.0
3	17.67	0.55	10.00	4	7.47	1025.00	7.90	2014-01-01 03:00:00	246	0.0	4.41	0.0
4	16.04	0.60	10.00	4	7.42	1025.17	5.81	2014-01-01 04:00:00	224	0.0	4.78	0.0

We have performed similar operations for Home 2 and Home 3.

Resampling the data

As we have to predict energy consumption for hourly and daily intervals, we resampled the time series data using `Pandas.resample()` function.

When resampling the meter data, we used `sum()` as the aggregation function for all columns. This is because energy consumption during a larger interval will be the sum of energy consumption during smaller intervals.

```
home1_meter['Date & Time'] = pd.to_datetime(home1_meter['Date & Time'])
home1_meter_hourly = home1_meter.resample('1H', on='Date & Time').sum()
home1_meter_hourly
```

Date & Time	House overall [kW]	Dishwasher [kW]	Furnace 1 [kW]	Furnace 2 [kW]	Home office [kW]	Fridge [kW]	Wine cellar [kW]	Garage door [kW]
2014-01-01 00:00:00	3.936655	0.000082	0.270632	0.126528	0.079389	0.283442	0.008504	0.018963
2014-01-01 01:00:00	1.916776	0.000118	0.418629	0.404557	0.078096	0.068467	0.008721	0.019553
2014-01-01 02:00:00	2.327774	0.000083	0.504724	0.806428	0.077839	0.078212	0.008941	0.019866
2014-01-01 03:00:00	2.570730	0.000037	0.495085	0.912469	0.077738	0.257606	0.009040	0.020044
2014-01-01 04:00:00	2.187469	0.000060	0.456456	0.831323	0.077846	0.024826	0.008989	0.020041

The above process was repeated to obtain daily consumption.

```
home1_meter['Date & Time'] = pd.to_datetime(home1_meter['Date & Time'])
home1_meter_daily = home1_meter.resample('1D', on='Date & Time').sum()
home1_meter_daily
```

Date & Time	House overall [kW]	Dishwasher [kW]	Furnace 1 [kW]	Furnace 2 [kW]	Home office [kW]	Fridge [kW]	Wine cellar [kW]
2014-01-01	69.947198	4.259966	13.836157	13.011470	3.350778	3.216339	0.683686
2014-01-02	80.218823	2.347394	12.503589	16.091142	3.150340	3.214963	0.778618
2014-01-03	72.365610	2.376973	11.424833	16.380261	3.683052	2.412674	0.609436
2014-01-04	61.267066	0.006021	11.117193	16.581127	2.372036	3.053694	0.918202
2014-01-05	71.768049	4.509473	8.655231	11.748494	1.865972	2.428641	0.705784

When resampling the weather data, we used `mean()` as the aggregation function for columns with numerical data (e.g. temperature, pressure, etc.) For categorical columns, we used `mode()` to aggregate.

```
home1_weather_daily = home1_weather.resample('1D').agg({'temperature':'mean', 'humidity':'mean', 'visibility':'mean', 'summary':pd.Series.mode,
home1_weather_daily
```

time	temperature	humidity	visibility	summary	apparentTemperature	pressure	windSpeed	windBearing	precipIntensity	dewPoint	precip
2014-01-01	20.819167	0.549583	9.642500	4	13.285833	1026.864167	6.481250	236.250000	0.000000	6.903333	
2014-01-02	13.607917	0.806250	2.255833	12	1.391250	1019.974583	9.664167	28.083333	0.003575	8.764167	
2014-01-03	3.955000	0.656250	6.258750	4	-10.191250	1018.191250	10.720833	276.458333	0.001700	-5.270417	

The above process was repeated to obtain daily consumption.

These operations were performed for data for all three homes.

Merging dataframes

In the final steps of data cleaning, we merged the meter and weather dataframes.

```
[26] merged_daily = home1_meter_daily.merge(home1_weather_daily, left_index=True, right_index=True)
```

```
[27] merged_daily['Day of the week'] = merged_daily.index.dayofweek
     merged_daily['Day of the month'] = merged_daily.index.day
     merged_daily['Month'] = merged_daily.index.month
     merged_daily['Year'] = merged_daily.index.year
```

```
merged_daily.head()
```

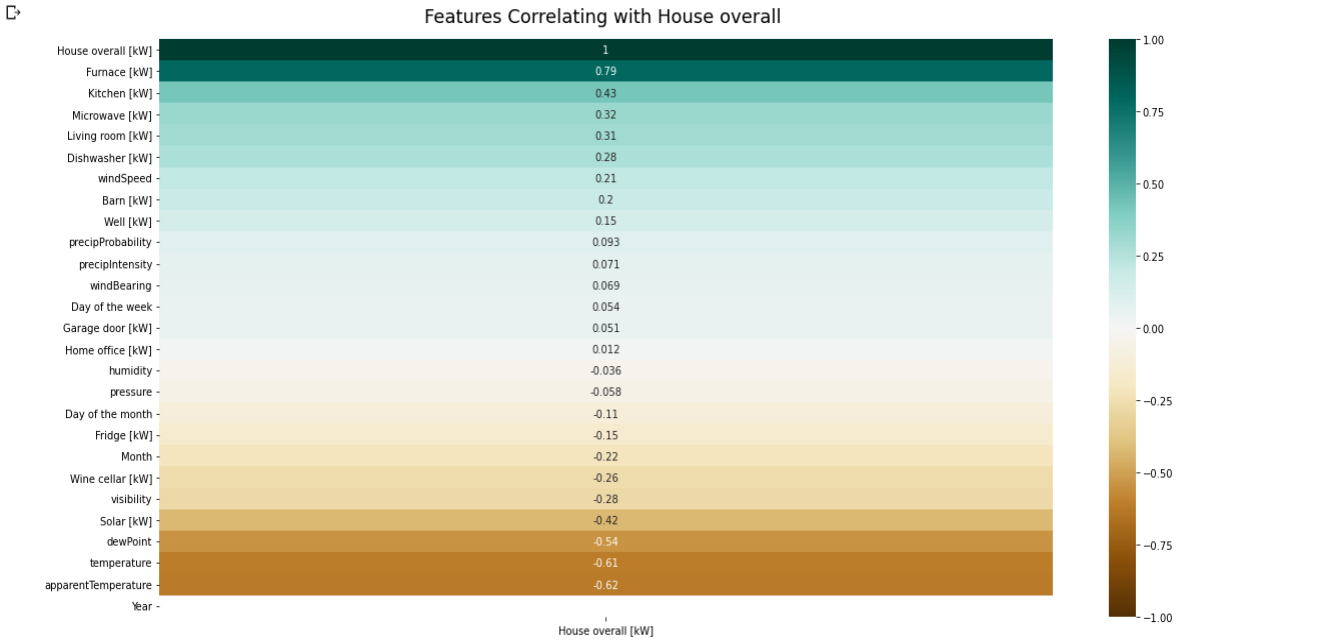
Date & Time	House overall [kW]	Dishwasher [kW]	Furnace 1 [kW]	Furnace 2 [kW]	Home office [kW]	Fridge [kW]	Wine cellar [kW]	Garage door [kW]	Kitchen 12 [kW]	Kitchen 14 [kW]	Kitchen 38 [kW]	Barn [kW]	Well [kW]	Microwave [kW]	Living room [kW]	Solar [kW]	temperature	humidity	visibility	summary	apparentTempe
2014-01-01	69.947198	4.259966	13.836157	13.011470	3.350778	3.216339	0.683686	0.533093	0.024543	0.612180	0.000379	3.393146	1.626072	0.892391	1.265130	3.508453	20.819167	0.549583	9.642500	4	13.
2014-01-02	80.218823	2.347394	12.503589	16.091142	3.150340	3.214963	0.778618	0.488344	0.278579	0.780990	0.000456	10.193153	1.032148	0.678559	2.945319	0.212557	13.607917	0.806250	2.255833	12	1.
2014-01-03	72.365610	2.376973	11.424833	16.380261	3.683052	2.412674	0.609436	0.542658	0.230869	0.959061	0.000387	9.510449	0.898288	1.412545	1.216280	3.157968	3.955000	0.656250	6.258750	4	-10.
2014-01-04	61.267066	0.006021	11.117193	16.581127	2.372036	3.053694	0.918202	0.572775	0.236326	0.595829	0.000392	7.904778	1.010425	0.625845	0.567848	5.086259	6.864167	0.600833	9.821250	4	0.
2014-01-05	71.768049	4.509473	8.655231	11.748494	1.865972	2.428641	0.705784	0.489415	0.291402	1.027364	0.000279	9.279841	1.737456	1.032148	1.243041	4.015413	20.988750	0.758750	7.417500	4	19.

Same operation was performed on home 2 and 3 data.

Find correlated variables

To identify highly (positively and negatively) correlated variables, we plotted a heatmap of correlation between features and the dependent variable ‘House overall [kW]’ which we wish to predict.

```
plt.figure(figsize=(20, 10))
heatmap_daily = sns.heatmap(merged_daily.corr()[['House overall [kW]']].sort_values(by='House overall [kW]', ascending=False), vmin=-1, vmax=1, annot=True, cmap
heatmap_daily.set_title('Features Correlating with House overall ', fontdict={'fontsize':18}, pad=16);
```



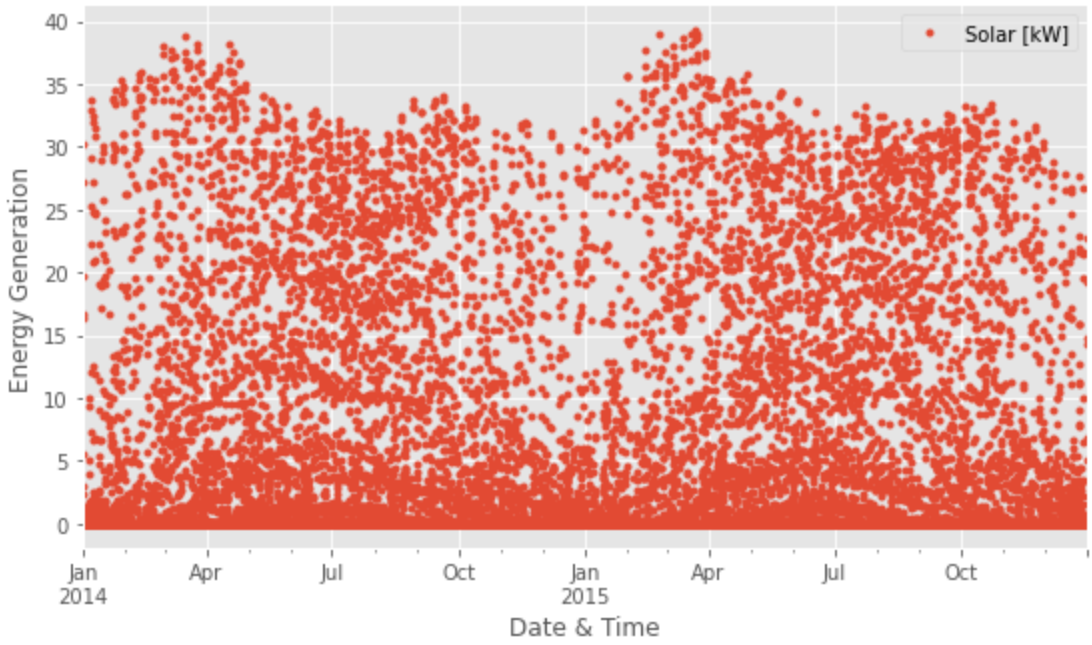
The plot shows that values from the Furnace and Kitchen columns are highly correlated with the total energy consumption for the day. This gives us the insight that most of the energy consumption comes from usage of furnaces and kitchen appliances.

Temperature has a very strong negative correlation with energy consumption. This means that when temperature is low, energy consumption is high. This can be explained by the fact that heating will drive up energy consumption during the winter months.

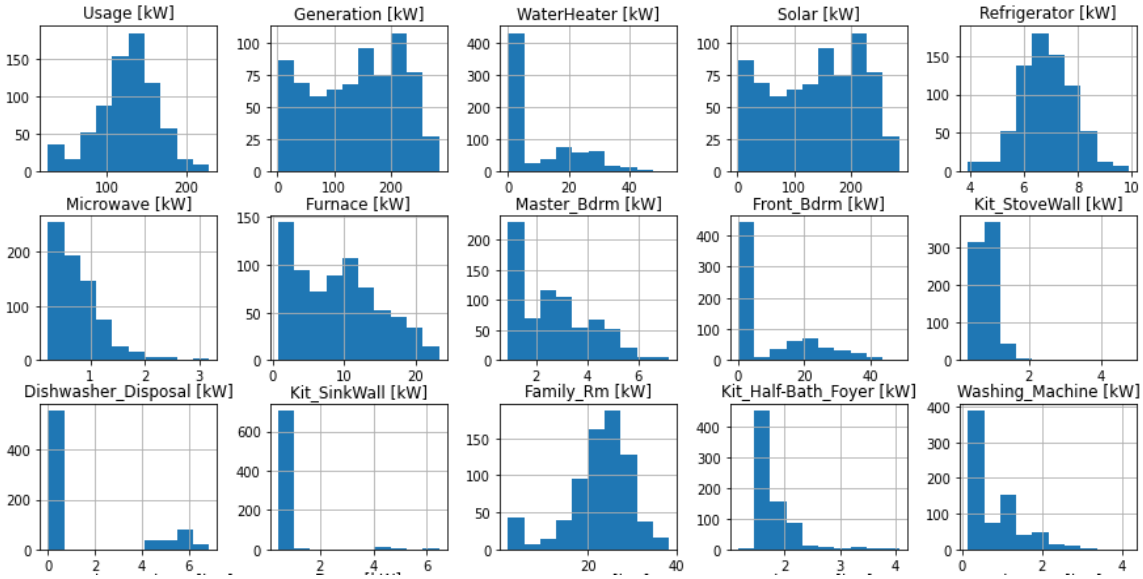
Data visualization

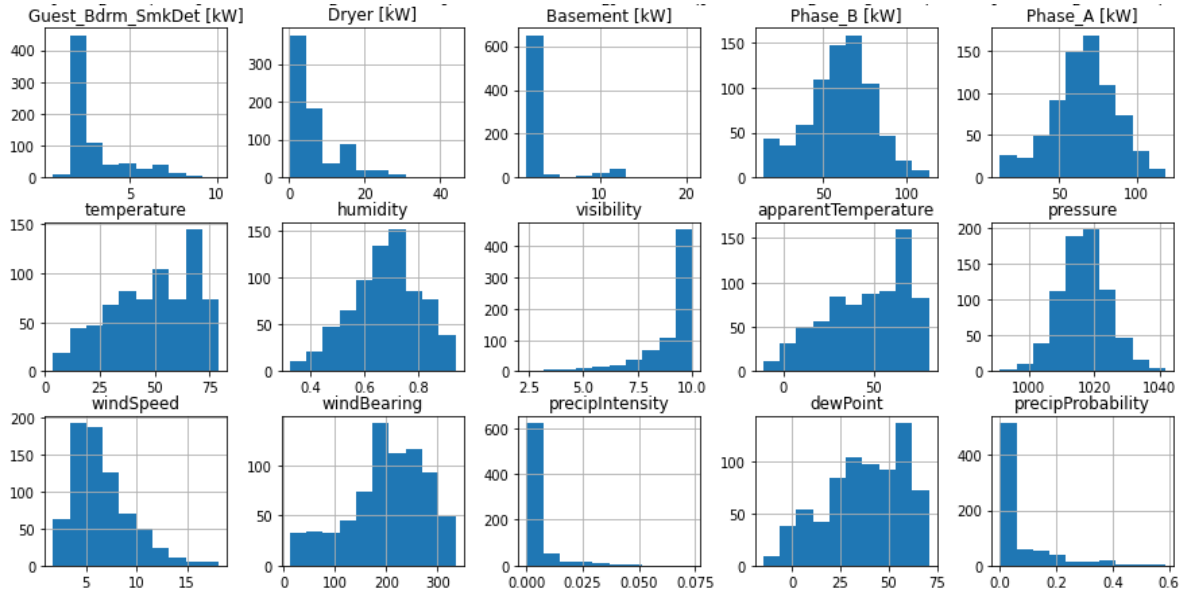
We use some visualizations to draw insights from our data. The below two points are worth mentioning:

1. Month Wise Energy Consumption Trend
   Generation from solar tends to more during the summer, especially around July - Aug and less during the colder seasons. Why would this be?
   It may be due to the consumption of more energy for cooling like ACs and
   The data might be from a tropical country where winters are not as harsh and heating might not be necessary.



2. In the figure below, the actual energy price roughly follows a normal distribution and thus, could be standardized. However, we also have to make sure that the time series does not require any other kinds of transformations. More specifically, we will check whether the time series of energy price is stationary, after visualizing its decomposed component time-series. Please refer to ARIMA section for details.





Models and parameter tuning

1. Naive / baseline model

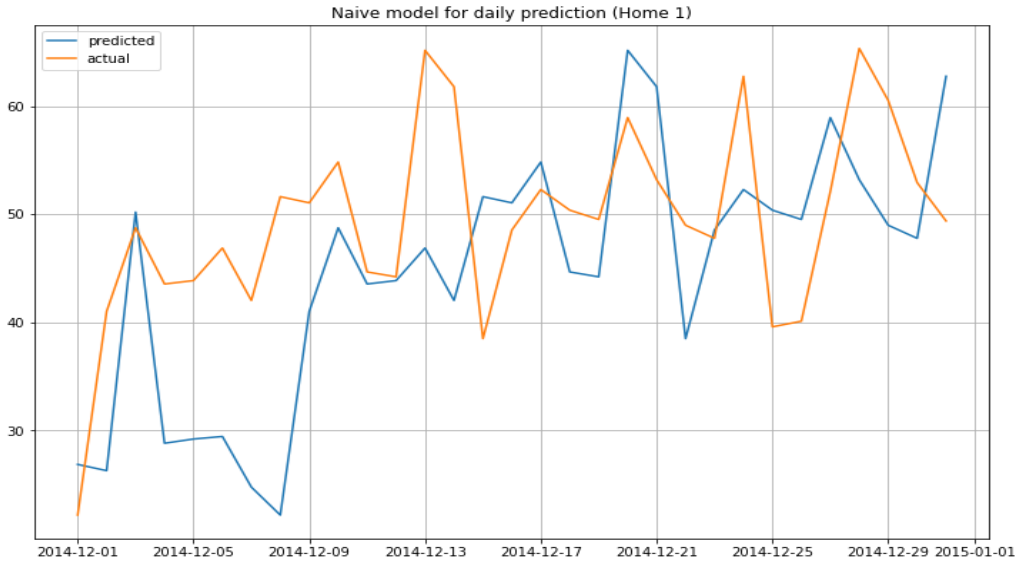
As per the instructions given in class, we have used a naive model as a baseline that uses data from one week ago to make current predictions.

Naive model for daily intervals:

Baseline MAEs and prediction plots for the three homes are as follows:

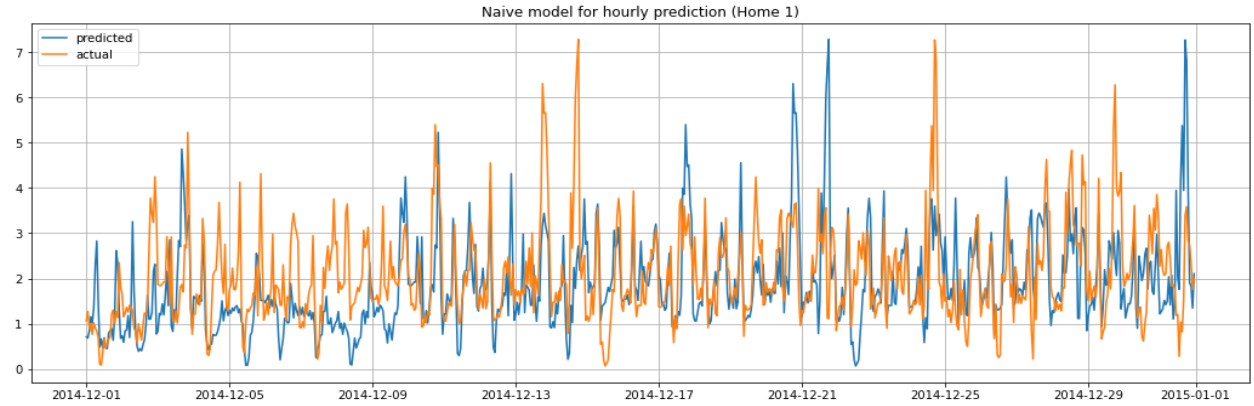
Home 1:

Daily:



Naive model MAE (daily): 9.83

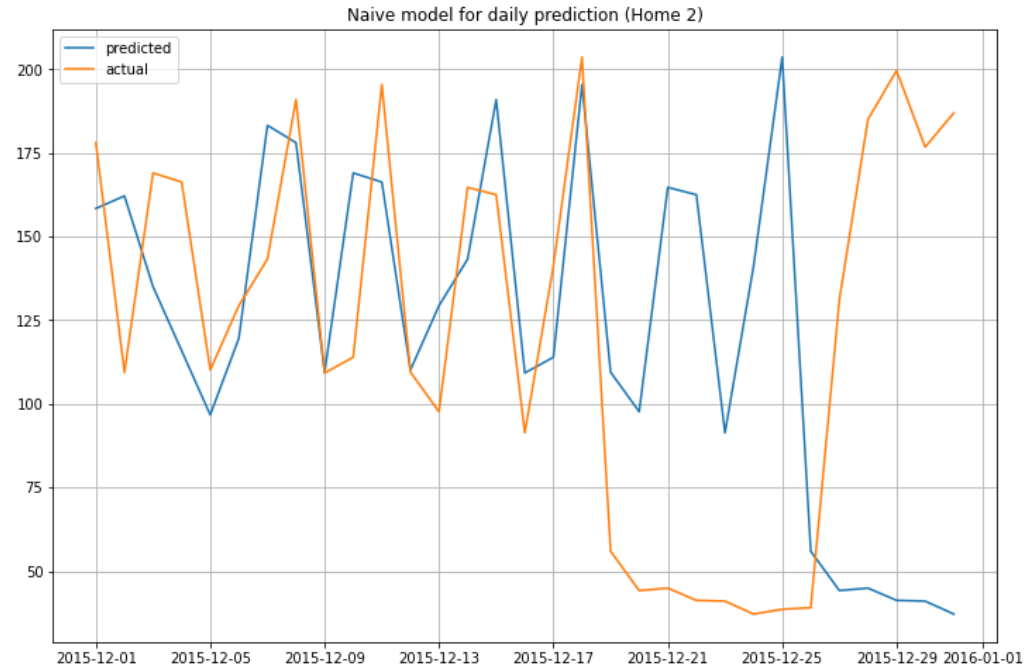
Hourly:



Naive model MAE (hourly): 0.80

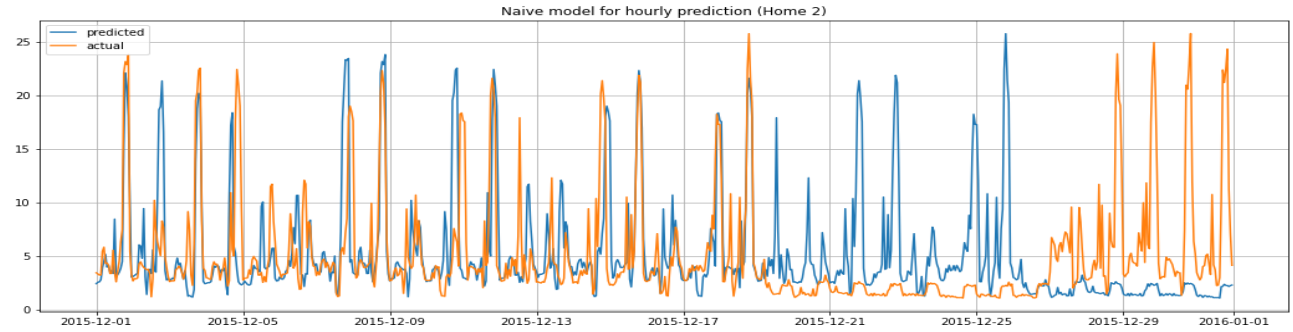
Home 2:

Daily:



Naive model MAE (daily): 58.35

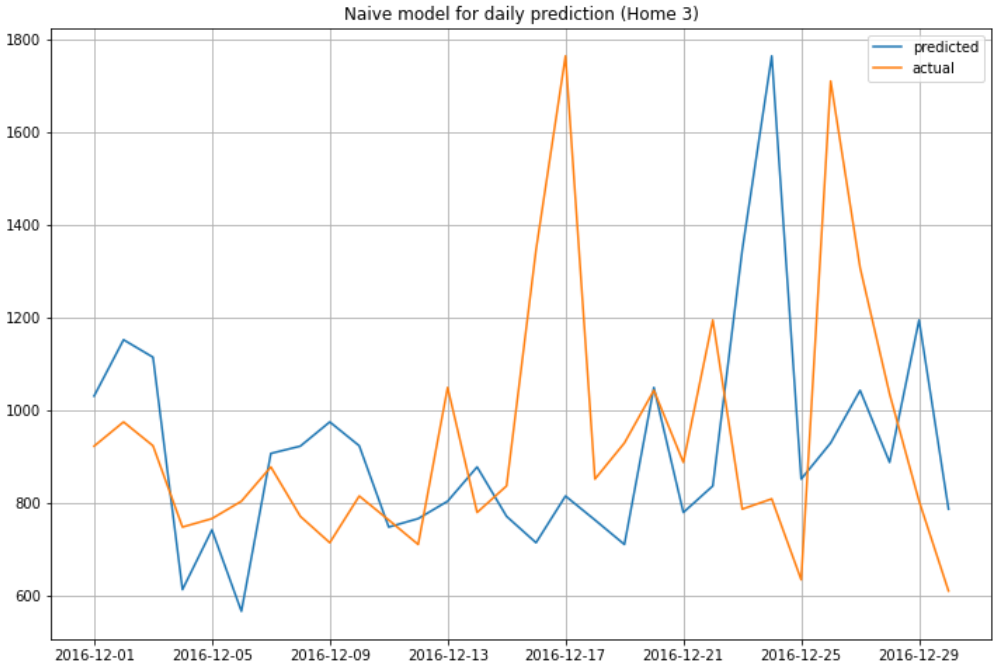
Hourly:



Naive model MAE (hourly): 3.43

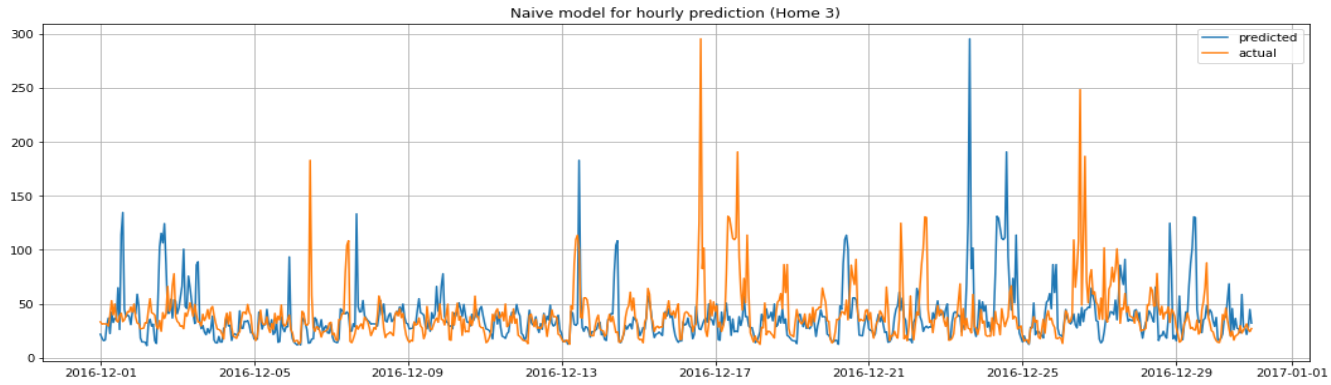
Home 3:

Daily:



Naive model MAE (daily): 258.44

Hourly:



Naive model MAE (hourly): 19.01

We can see that the predicted graphs lag behind one week behind the actual graphs.

2. Statistical model - **ARIMA**

ARIMA, short for '**Auto Regressive Integrated Moving** Average' is a class of models that 'explains' a given time series based on its own past values.

According to the name, we can split the model into smaller components as follow:

· **AR**: An **Auto-regressive** model which represents a type of random process. The output of the model is linearly dependent on its own previous value i.e. some number of lagged data points or the number of past observations.

· **MA**: A **Moving Average** model whose output is dependent linearly on the current and various past observations of a stochastic term.

· **I**: **integrated** here means the differencing step to generate stationary time series data, i.e. removing the seasonal and trend components.

ARIMA model is generally denoted as ARIMA (p, d, q) and parameter p, d, q are defined as follow:

Ø **p**: the lag order or the number of time lag of autoregressive model AR(p)

Ø **d**: degree of differencing or the number of times the data have had subtracted with past value

Ø **q**: the order of moving average model MA(q)

To build a time-series model using the ARIMA model, the dataset needs to be **stationary**. Stationary time series is when the mean and variance are constant over time. It is easier to predict when the series is stationary.

We used the **Dickey-Fuller test** to verify whether the data series is stationary or not. For our time series to be stationary, the p-value from the test has to be ≤ 0.01.

We performed following steps to build the Arima model to predict Electricity demand prediction:

Step 1 — **Check stationarity**

Step 2 — **Difference**: If the time series is not stationary, it needs to be stationarized through differencing. We need to check seasonal differencing as well using **Decomposition** technique.

Step 3 — **Select AR and MA terms**: Use the ACF and PACF to decide whether to include an AR term, MA term, or both.

Step 4 — **Build & Validate the model**: Build the model and set the number of periods to forecast & Compare the predicted values to the actuals in the validation sample.

ARIMA model for Home 1 daily predictions:

Step 1: Stationarity Test Using Dickey-Fuller Test:

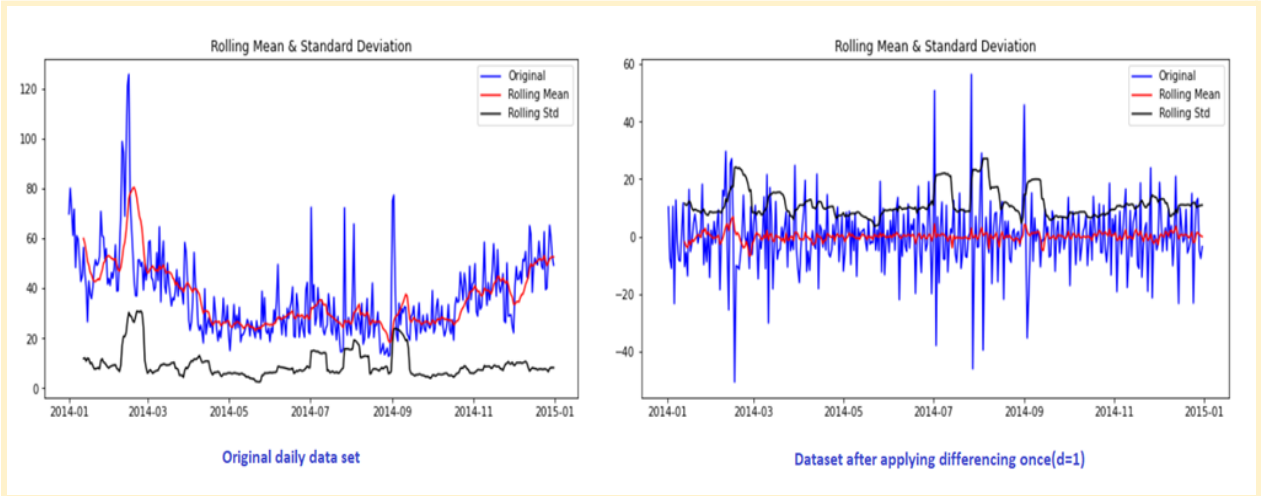
For Home 1 daily dataset returned the p-value as **0.1918441755** which is greater than 0.01.

i.e., our time series is not stationary. As ARIMA requires the dataset to be stationary, we will perform differencing in order to make our data stationary.

Step 2: We're going to try difference the dataset, to check if stationary is achieved.

After differencing, the p-value for Home 1 daily dataset is reduced to **0.0000000001** which is ≤ 0.01.

We need to check seasonal differencing as well.

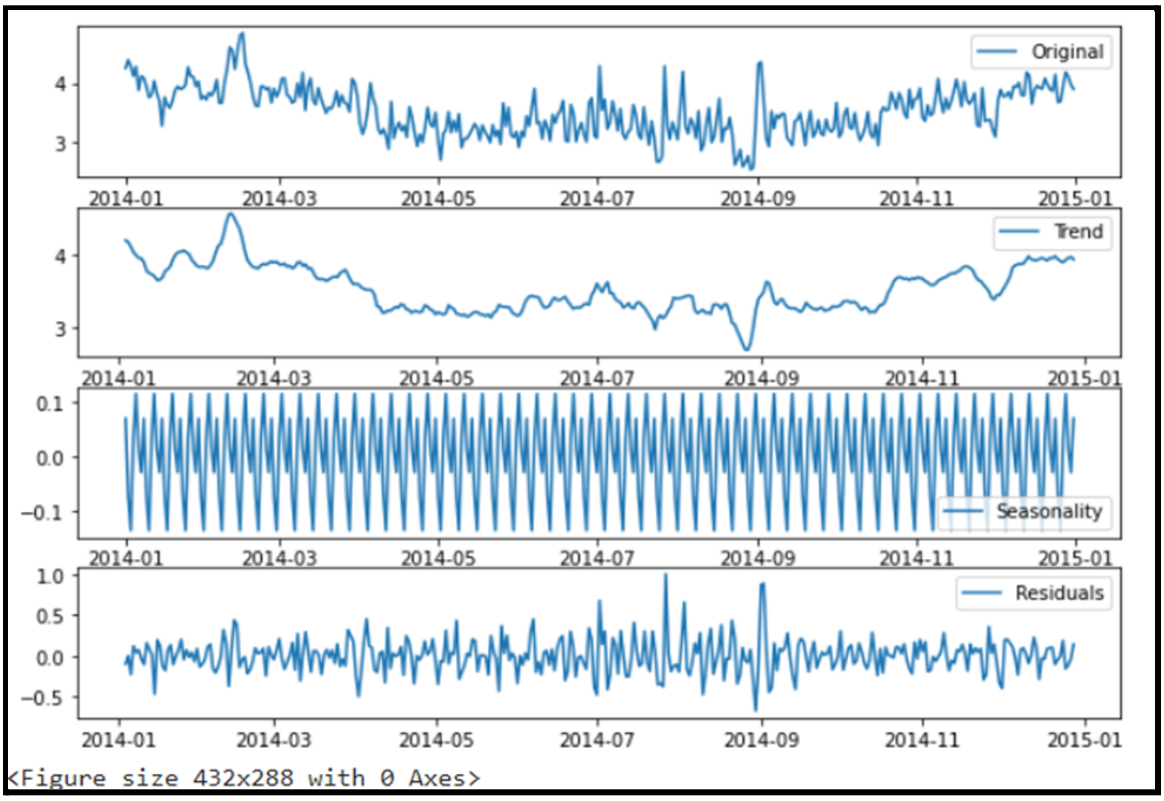


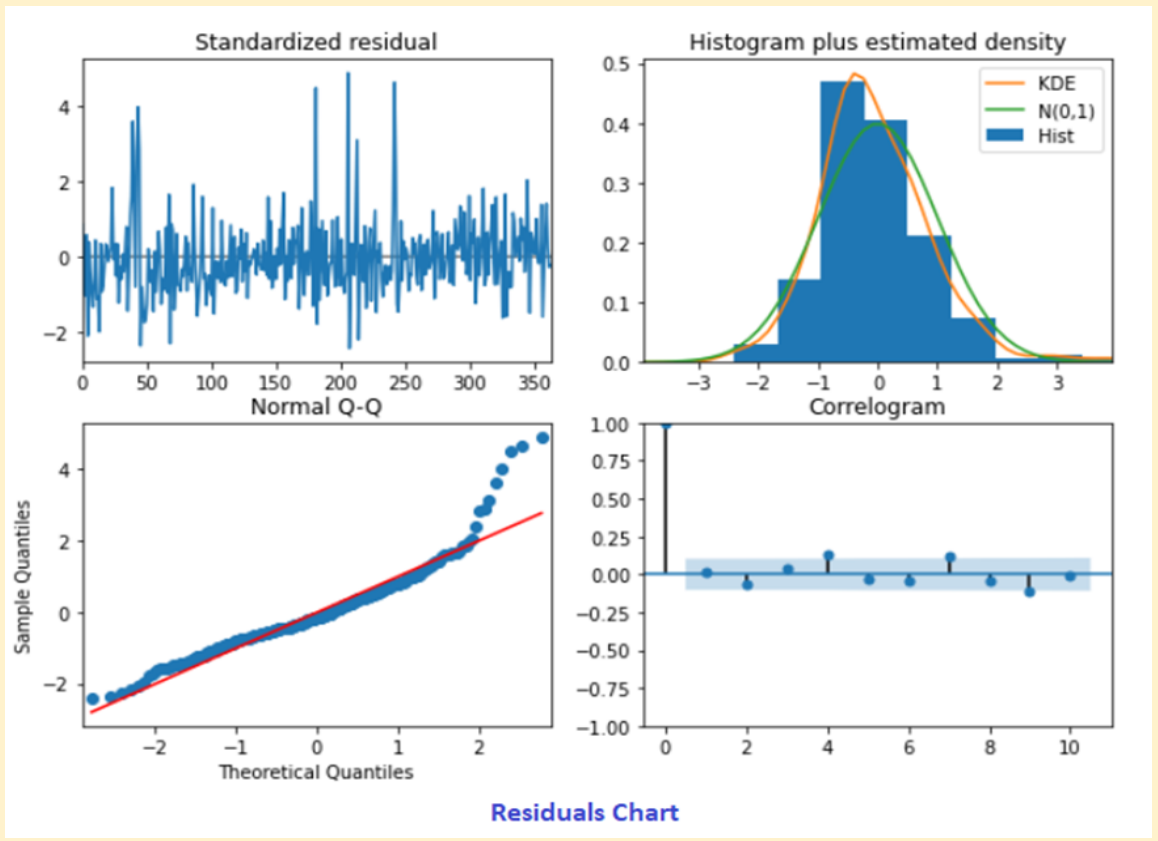
The above plot shows that mean and SD values have been stabilized around a central value after differencing.

After differencing, the normal dataset as well as the log-transformed dataset has the p-value less than 0.01. But still we can see trends and Seasonality for Home-1 daily dataset.

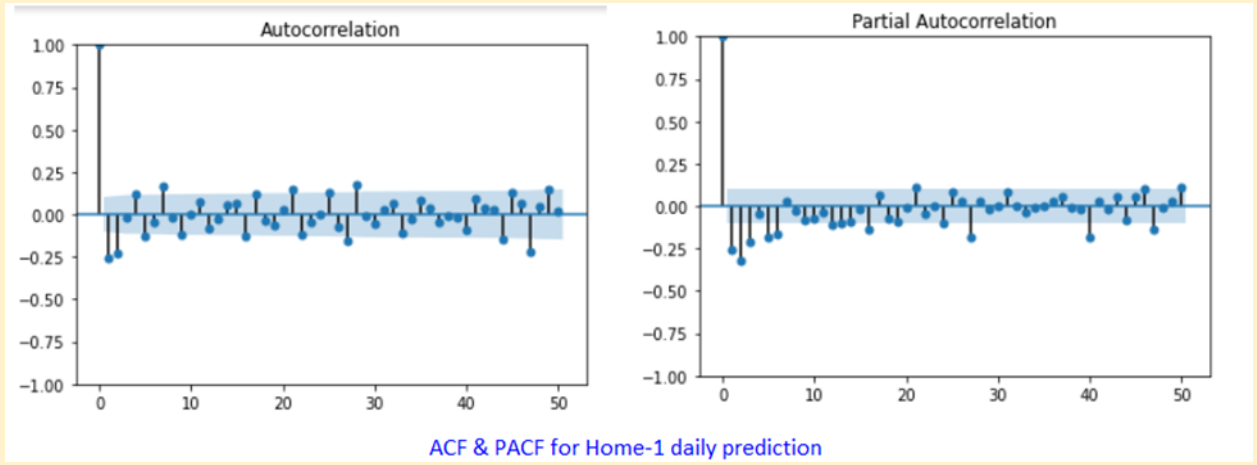
After differencing, Now the normal dataset as well as the log-transformed dataset has the p-value less than 0.01. But still we can see trend and Seasonality for Home-1 daily dataset which can be achieved using Decomposition technique.

Here we can see that the trend & seasonality are separated out from data and we can model the residuals.





Step 3: Finding the parameters (p,d,q) of the ARIMA model using ACF and PACF.



Analyzing the ACF plot, we can see any spike slightly outside of the confidence band, so we'll assume that **AR(1)**. As for the PACF plot we can see the first spike at lag=1, so we'll pick **MA(1)**. We have our model, **ARIMA (1,1,1)**

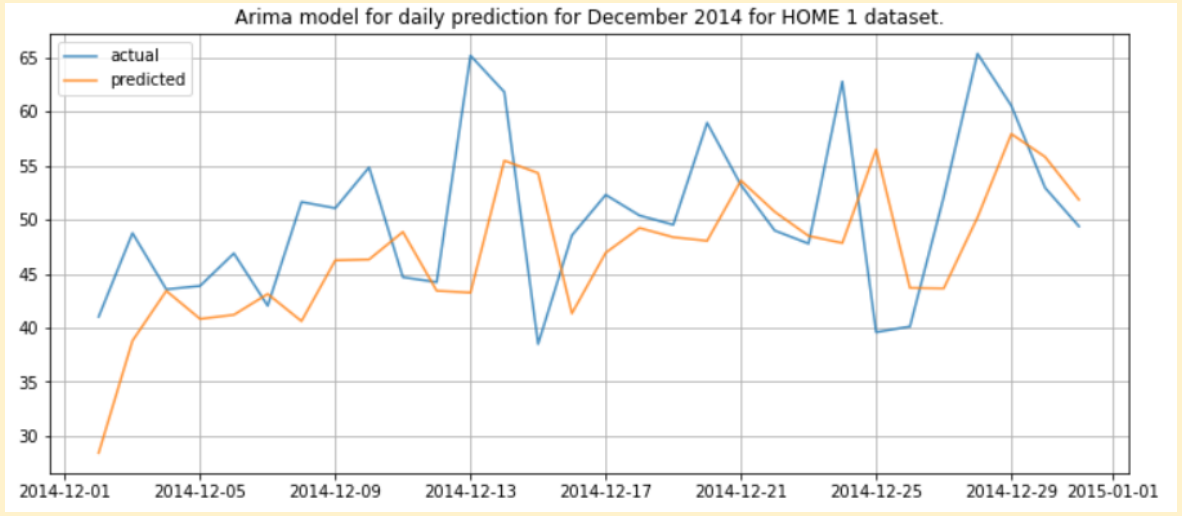
We can use auto_arima(), to search multiple combinations of p,d,q parameters and choose the best model that has the least AIC. We confirmed our finding with the result of auto-ARIMA:

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=2839.327, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=2821.003, Time=0.12 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=2793.580, Time=0.23 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=2837.335, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=2764.519, Time=0.33 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=2766.237, Time=0.80 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=2766.016, Time=0.62 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=2768.610, Time=0.43 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=2791.537, Time=0.67 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=2767.630, Time=1.55 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=2762.584, Time=0.24 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=2791.615, Time=0.08 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=2819.018, Time=0.06 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=2764.297, Time=0.38 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=2764.073, Time=0.39 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=2766.659, Time=0.15 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=2789.560, Time=0.17 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=2765.689, Time=0.58 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0]
Total fit time: 7.008 seconds
```

ARIMA Results for plot(1,1,1)

Step 4: Build the model using p,d,q values and validate the predicted & actual data.

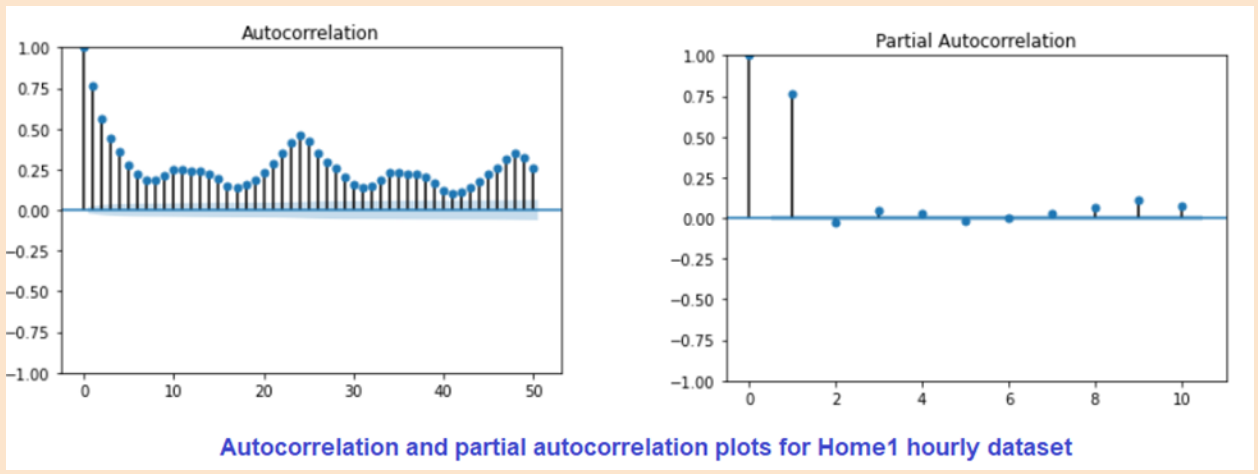


ARIMA model MAE value(daily): 6.714

ARIMA Model for hourly prediction for home1:

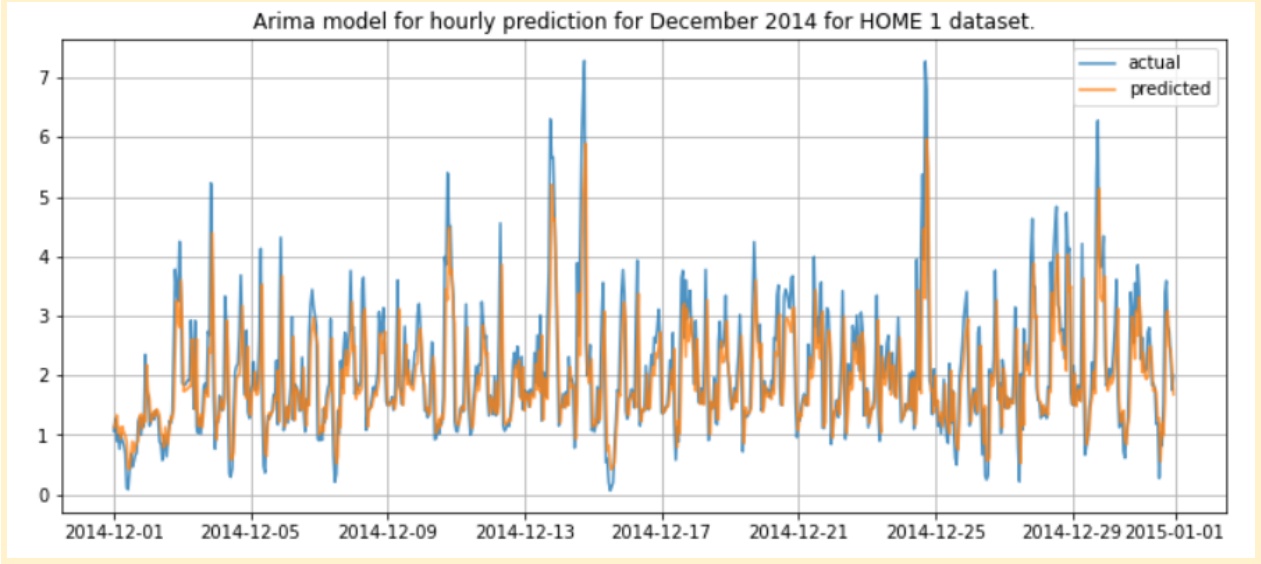
Augmented Dickey-Fuller Test Result for home 1(daily) has p-value: 0.0000000000

The p-value is ≤ 0.01. The dataset is **stationary** and the hourly plot shows that the time series is centered around a stationary value, i.e. the mean / S.D. is constant. Thus, we can use ARIMA.



Autocorrelation and partial autocorrelation plots for Home1 hourly dataset

Analyzing the ACF plot, we can see any spike slightly outside of the confidence band, so we'll assume that AR(1). As for the PACF plot we can see the first spike at lag=1, so we'll pick MA(1). We have our model, ARIMA (1,0,1)



ARIMA model MAE Value(Hourly): 0.539

Home 2 :

We performed same analysis as above for home 2, here are the results:

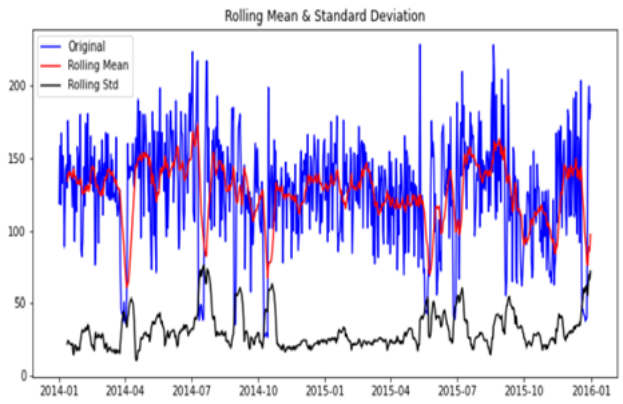
	Daily Time Intervals	Hourly Time Intervals
Dickey-Fuller Test Result	p-value: **0.0003053860**	p-value: **0.0000000000**
Differencing	NA	NA

Decomposition	Need to remove trends and seasonality from the dataset.	NA
p	**1**	**1**
d	**0**	**0**
q	**1**	**1**

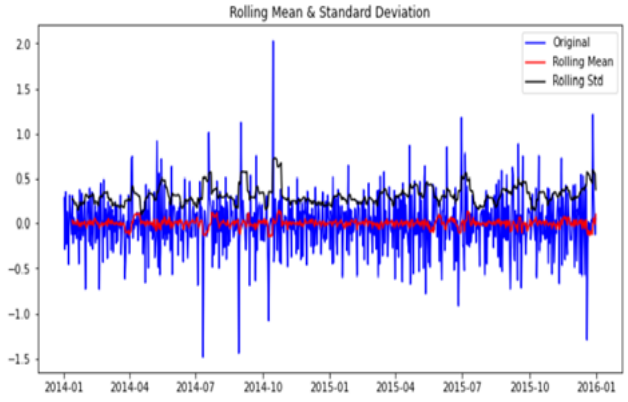
Differencing was not required as p-value < 0.01 for both daily and hourly time intervals.

<u>Daily prediction:</u>

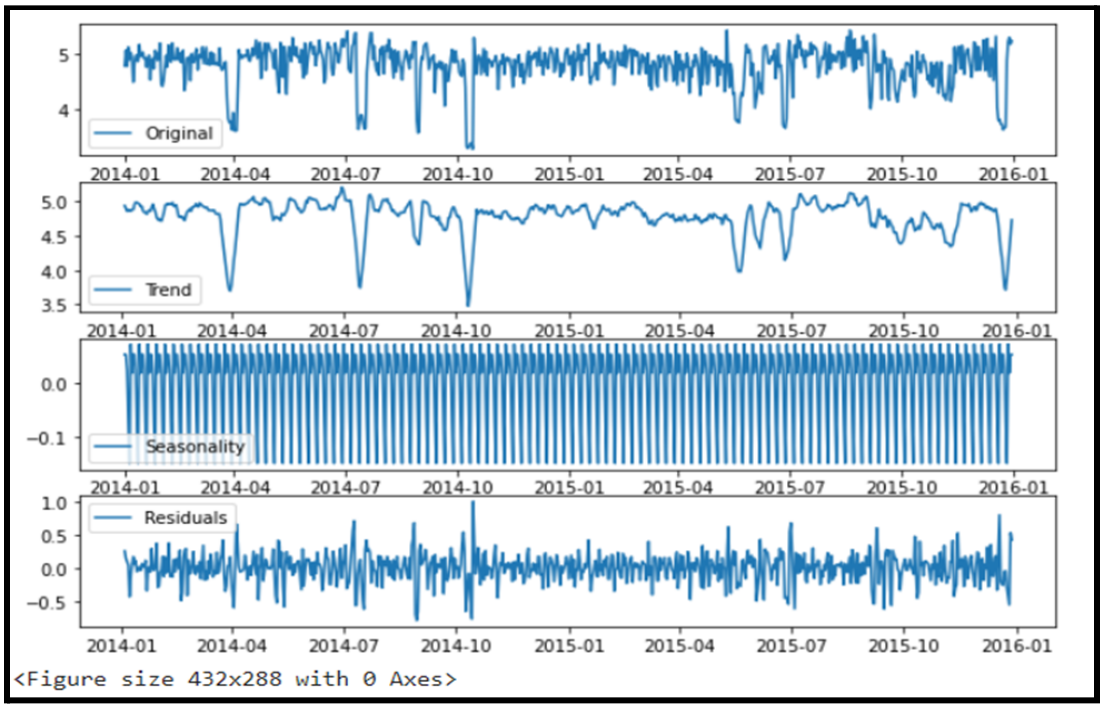
After using decomposition technique



Original dataset before decomposition

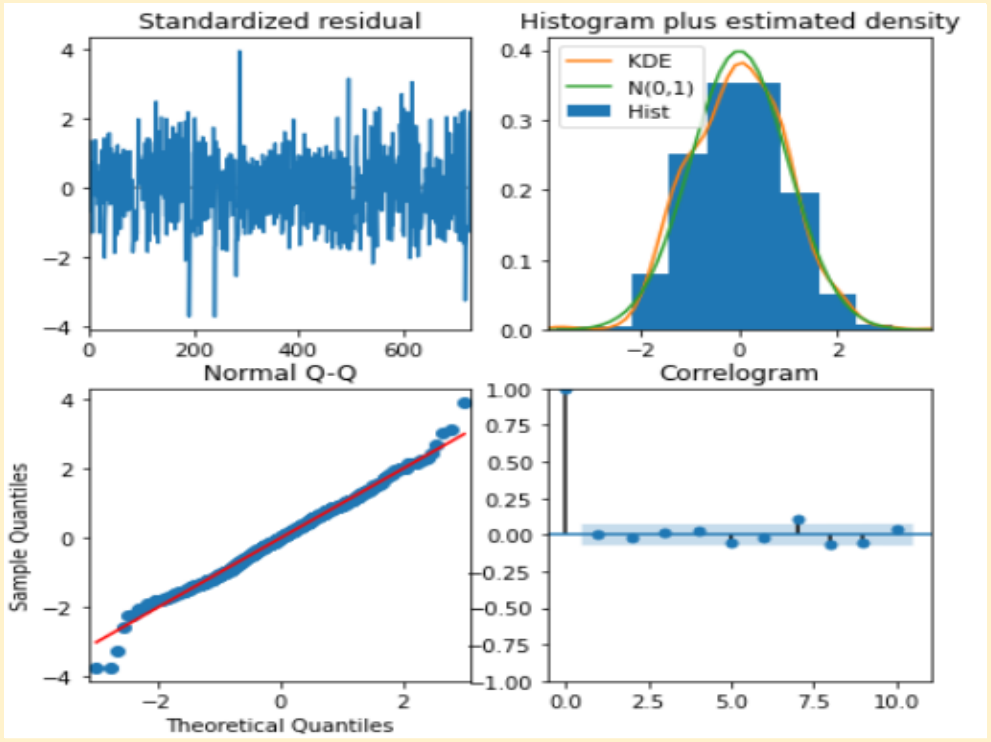


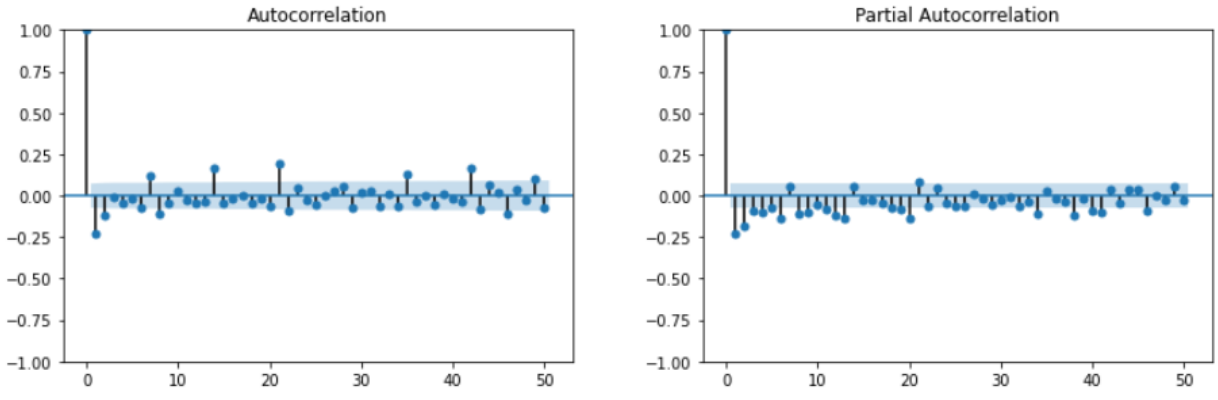
After decomposition



Here we can see that the trend & seasonality are separated out from data and we can model the residuals.

Residuals Chart

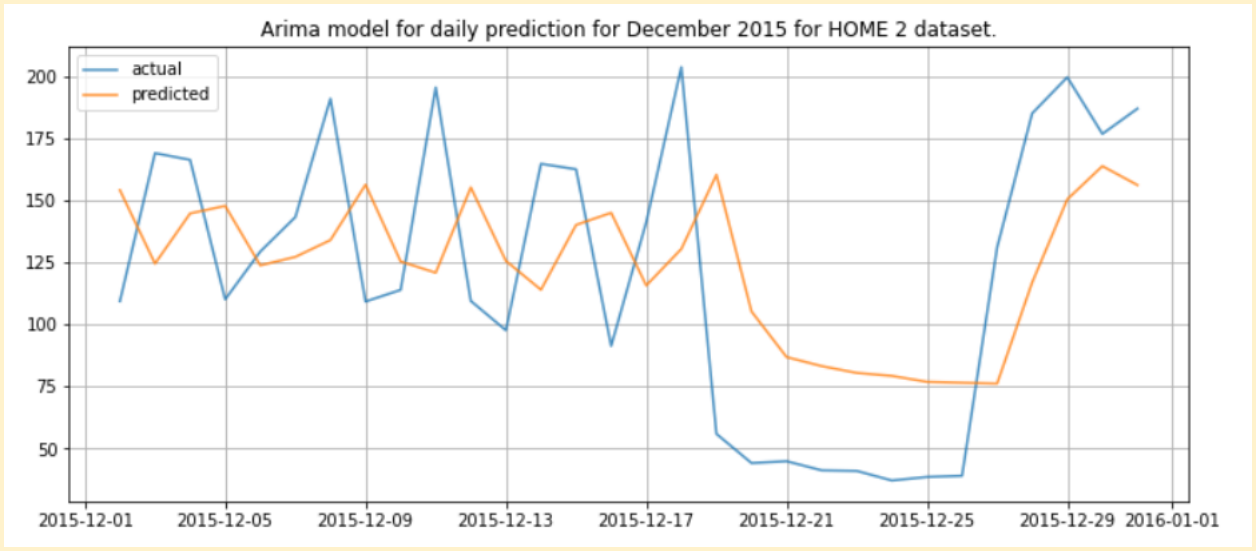




Autocorrelation and Partial Autocorrelation for Daily Home2 dataset

Analyzing the ACF plot, we can see any spike slightly outside of the confidence band, so we'll assume that **AR(1)**. As for the PACF plot we can see the first spike at lag=1, so we'll pick **MA(1)**. We have our model, ARIMA **(1,0,1)**.

ARIMA Results for plot(1,0,1)

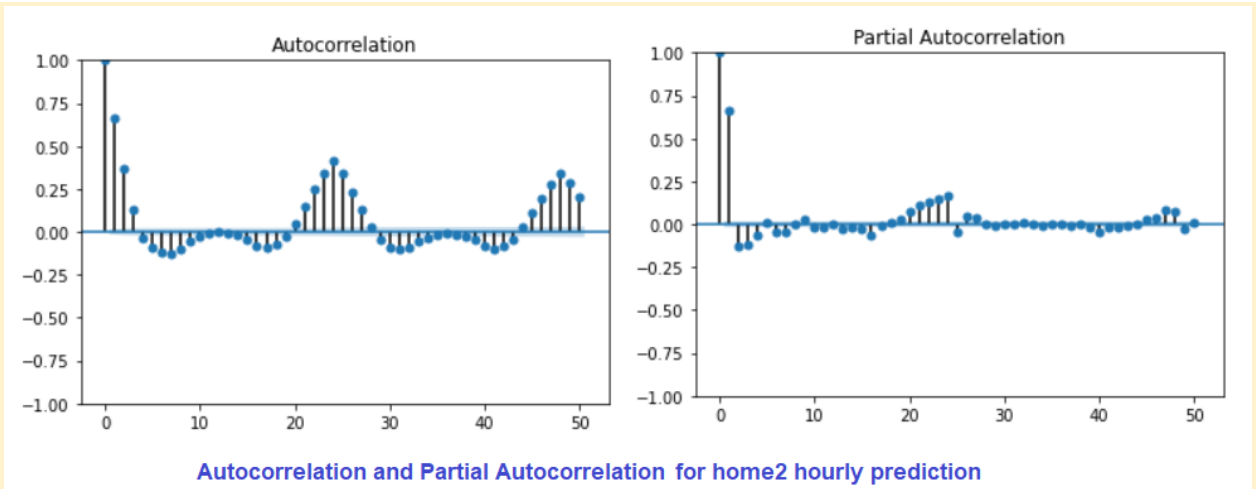


ARIMA model MAE (daily) : `42.748`

`RMSE: 47.512`

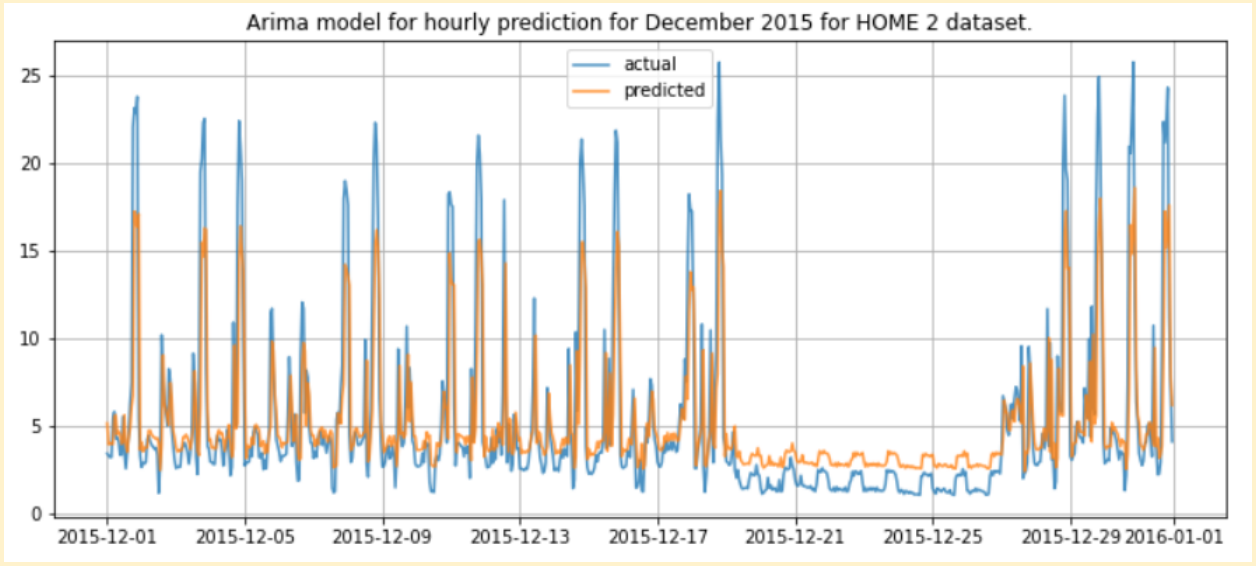
`MAPE: 0.721`

ARIMA Model for hourly prediction for home2:



Autocorrelation and Partial Autocorrelation for home2 hourly prediction

Analyzing the ACF plot, we can see any spike slightly outside of the confidence band, so we'll assume that **AR(1)**. As for the PACF plot we can see the first spike at lag=1, so we'll pick **MA(1)**. We have our model, **ARIMA (1,0,1)**.



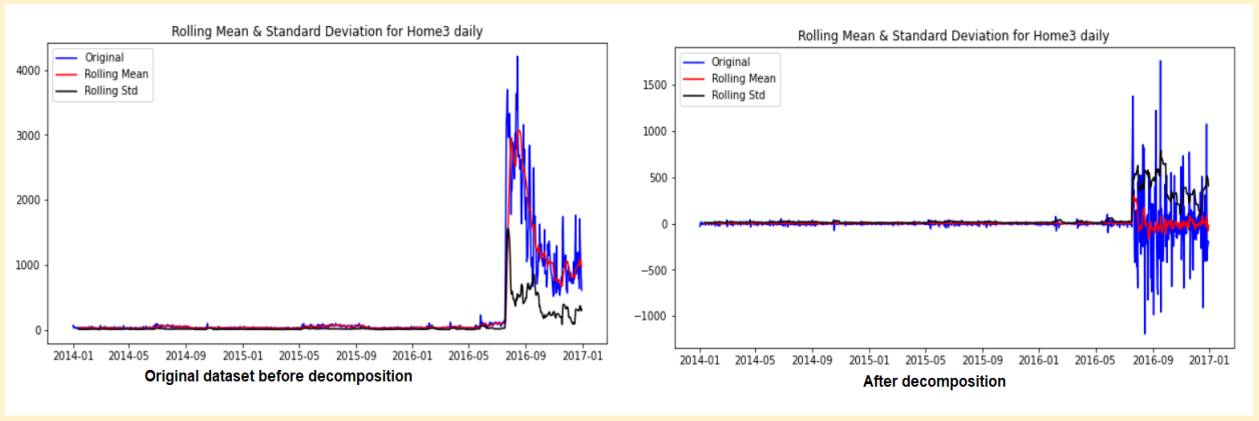
ARIMA model MAE (hourly) : `1.960`

Home 3:

	Daily Time Intervals	Hourly Time Intervals
Dickey-Fuller Test Result	p-value: 0.0659281997	p-value: 0.0033792573
Differencing	Yes	NA
Decomposition	Need to remove trends and seasonality from the dataset.	NA
p	**9**	**1**
d	**1**	**0**
q	**1**	**1**

For the daily interval, we can see p-value > 0.01. Thus, we need to perform differencing.

ARIMA Model for Daily prediction for home3:

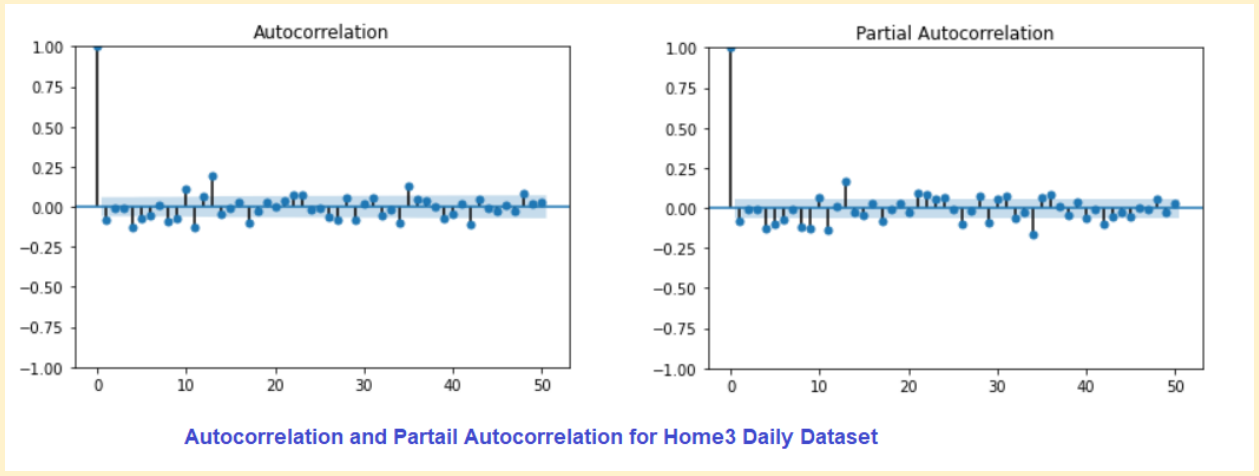


We can see a huge difference in the data for the year 2016, due to which it is hard to remove trends and it might result in higher MAE value.

The above spike in data prompted us to investigate the data for home 3, and to reason about the spike. The investigation revealed that home 3 meter data has **values for every minute** starting from 19th July 2016 18:56. As a result of resampling the data, we see huge spikes.

Date & Time	use [kW]
7/19/2016 16:00	3.375188
7/19/2016 16:30	1.908863
7/19/2016 17:00	1.849393
7/19/2016 17:30	3.402942
7/19/2016 18:00	2.961582
7/19/2016 18:30	2.00933
7/19/2016 18:56	3.511683
7/19/2016 18:57	3.055267
7/19/2016 18:58	3.05695
7/19/2016 18:59	3.183983
7/19/2016 19:00	3.6228
7/19/2016 19:01	3.620433
7/19/2016 19:02	3.615617

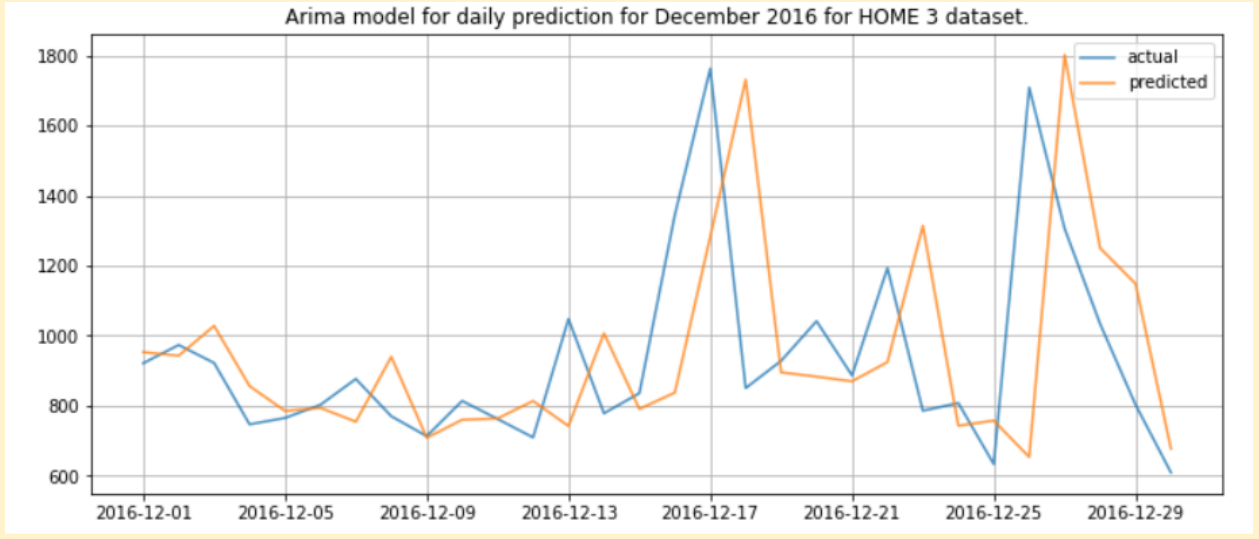
Home 3 data columns show that from 19th July 2016, energy consumption was provided per minute. This results in an extreme spike in the data during resampling. This can be seen from the mean value in the above plot. We have decided to keep this data as it is possible to have a surge of energy consumption.



Autocorrelation and Partail Autocorrelation for Home3 Daily Dataset

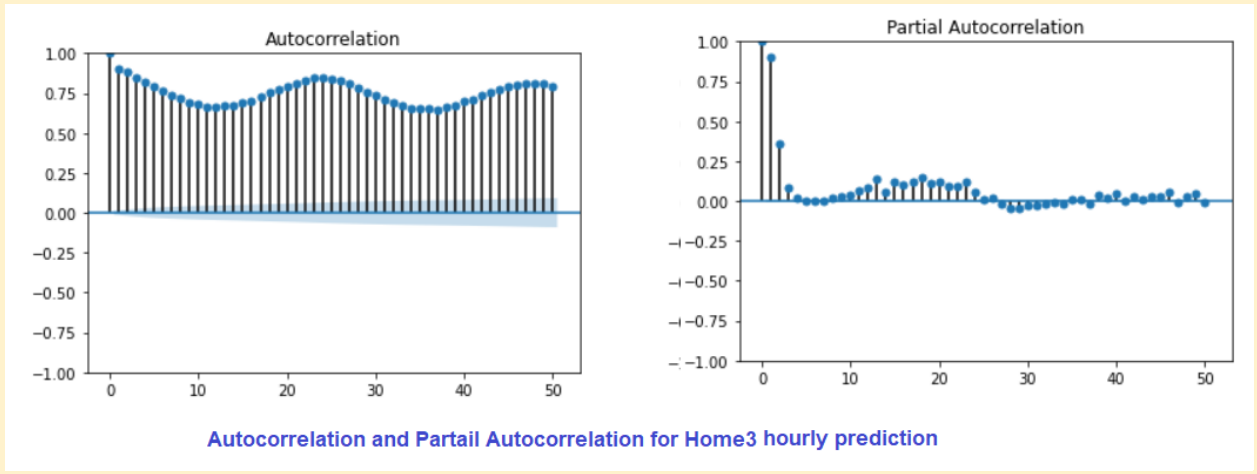
We have our model, ARIMA (9,1,1)

ARIMA Results for plot(9,1,1)



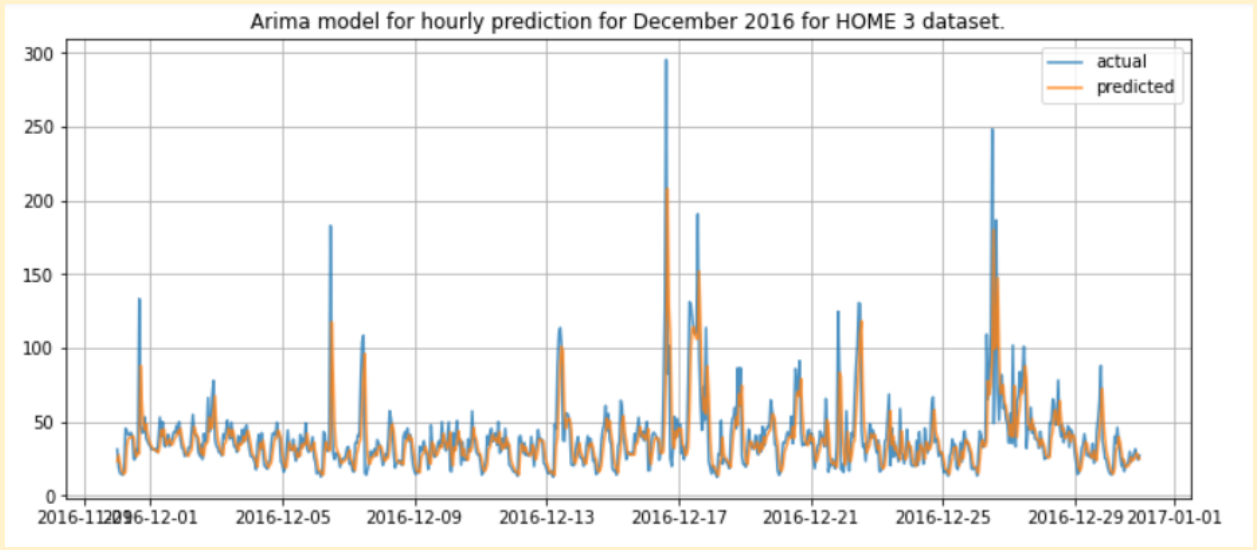
ARIMA model MAE (daily) : `219.590`

ARIMA Model for hourly prediction for home3:



Autocorrelation and Partail Autocorrelation for Home3 hourly prediction

We have our model, ARIMA (1,0,1).



ARIMA model MAE (hourly) : `11.778`

3. Deep learning model - LSTM

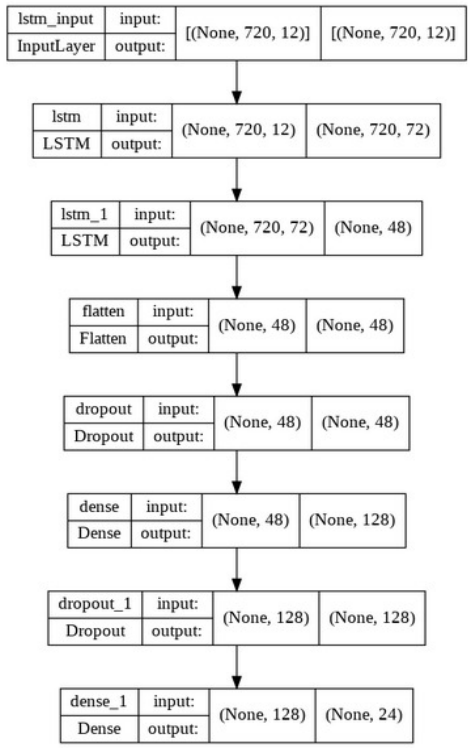
The model makes predictions for each day of 24 hours at midnight and forecasts the next 24 hours of energy demand. Usually, in traditional ML, hourly consumption sequences are used at a time followed by the next 24 hours. The model implemented by us treats each hour as independent. It considers 24 individual forecasts corresponding to each hour of the day.

Advantage of this approach:

1. The stronger auto-correlations between h0 to h23 of today and the next day, etc. is used rather than the autocorrelation between h0,h1,h2 and further etc
2. Our dataset being smaller, it considers seasonality even in just 2 or 3 years.

Design:
LSTM with 50 layers. This is a sequential model with a 'RELU' activation and has 1 Dense Layer
Shuffle is turned to False.
We adjust the output of the layers in our model to fit the input of the next layer by reshape. The Reshape in each step looks like this:
(1, len(X), 1)

The below describes the model we built:



Hyperparameter Tuning:
We define a set of parameters also called configurations. For our model we have the following params:

```
lag = 6,
LSTM_layer_depth = 50,
epochs = 10,
batch_size = 256,
```

After a lot of runs, we found that the LSTM appears to begin to become very overfit from about epoch 10-12 where the validation loss begins to rise. In all cases this is a sign the models are no longer learning against the validation set. Some options to help improve this are:

- to introduce learning rate decline
- train on longer input sequences
- increase the batch size.
- alter the depth of the layers
- reducing the no of parameters

We tried using different values and fixed the above.

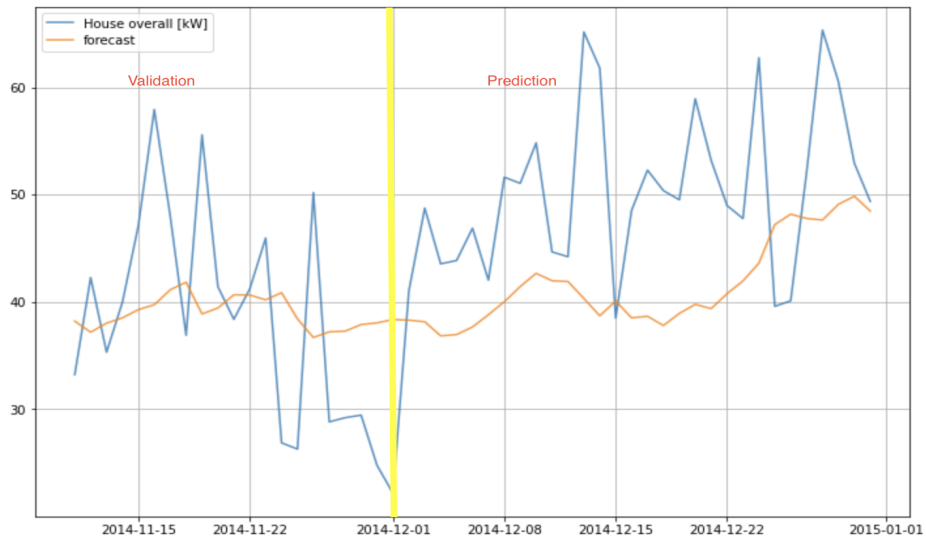
Observations:

1. The number of parameters is really an important metric for getting the right model capacity and complexity. We need to control the number of parameters of each layer in the model to handle overfitting or underfitting situations.
   What we did to prevent these situations is to adjust the number of parameters of each layer. We need to know how the number of parameters actually affects the performance of each layer.
   We did this by adjusting the acceptable correlation coefficient threshold and only considering highly correlated features. (More on this below)
2. The batch size set when fitting your model controls how many predictions we make at a time. This is not a problem in our case as we want to make the same number predictions at a time as the batch size used during training.
   This does become a problem when we wish to make fewer predictions than the batch size. For example, you may get the best results with a large batch size, but are required to make predictions for one observation at a time on something like a time series or sequence problem.

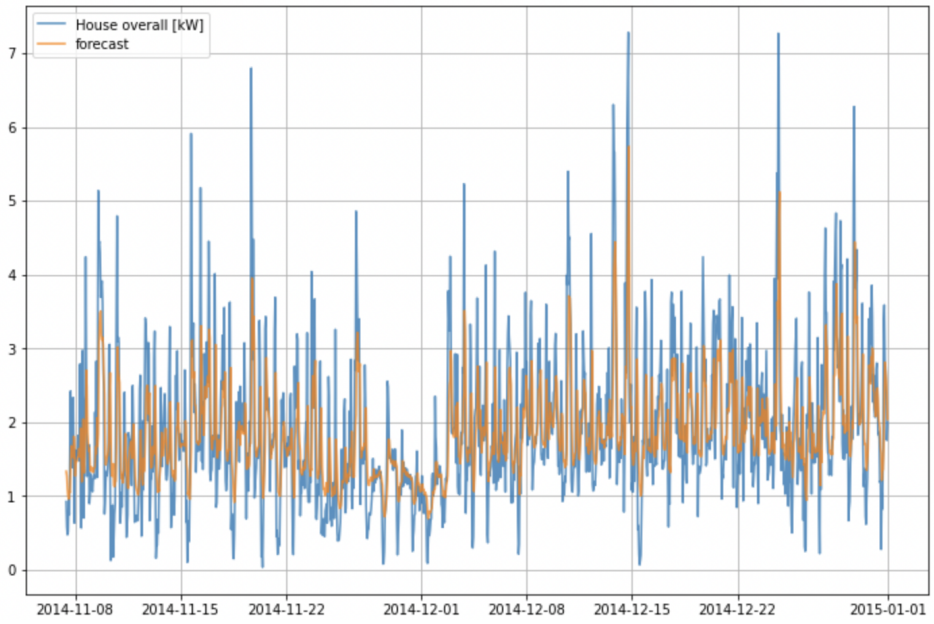
Plots of the MAE showed maximum overlap between predicted and actual with the above configurations and that reinforced our understanding.

Just playing with the hyperparameters seemed to only improve the performance of LSTM by a little. The real potential of LSTM lies in the utilization of huge data which is sequential. Given enough data, LSTM outperforms other classical and statistical models by huge margins.

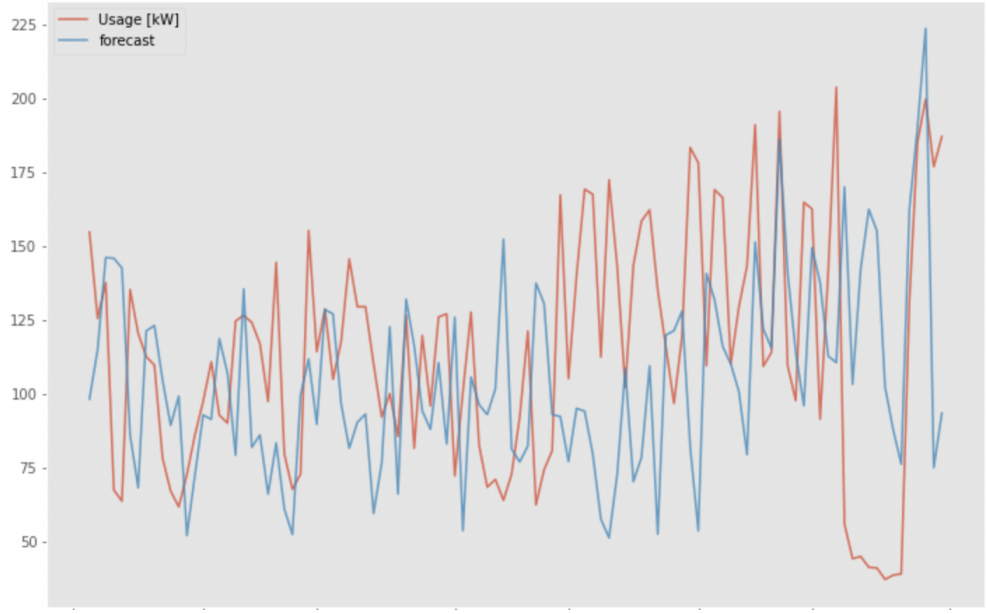
Home1 Daily



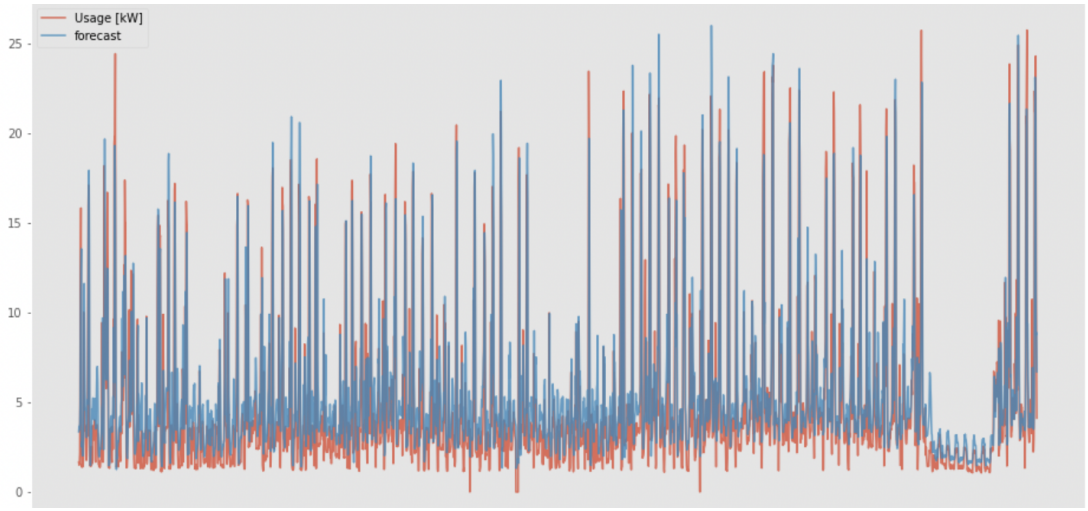
Home1 Hourly



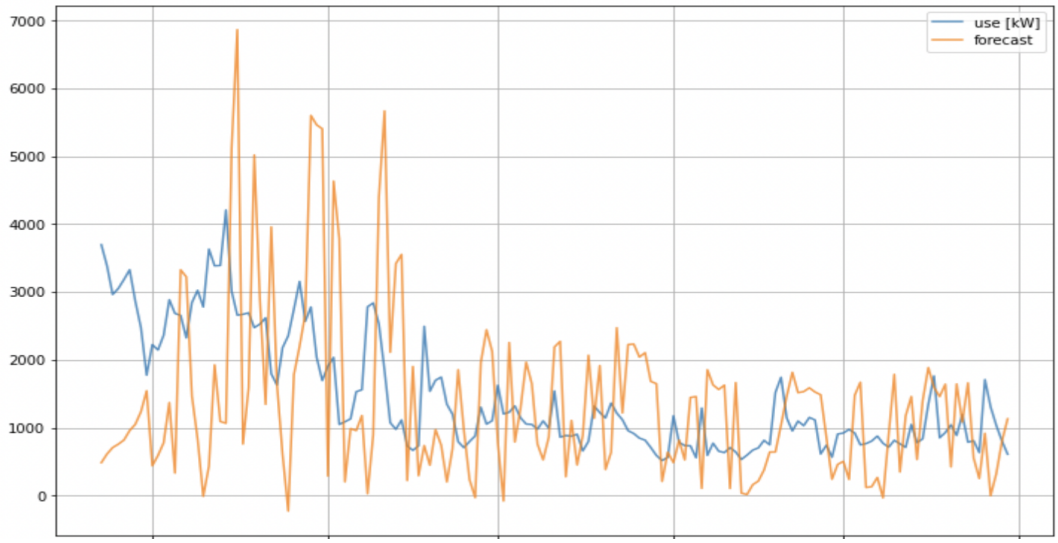
Home2 Daily



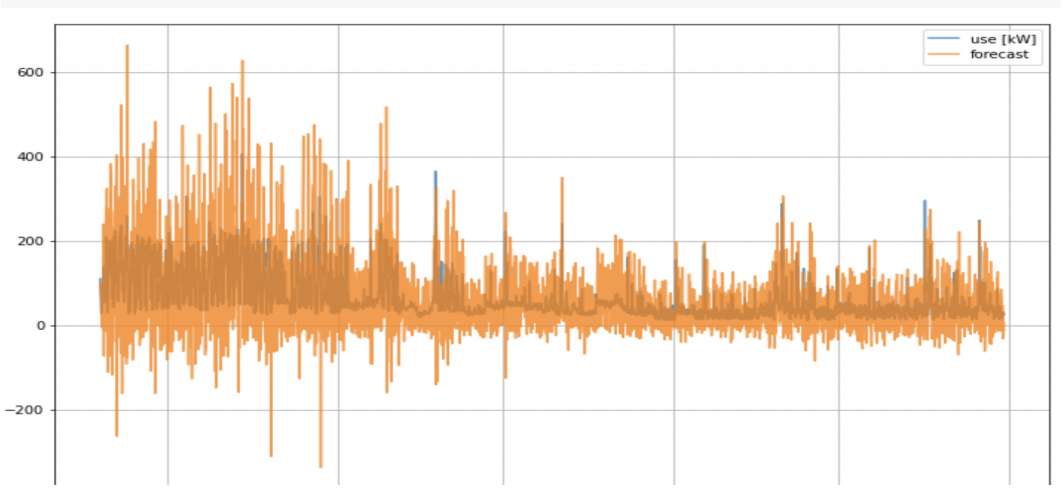
Home2 Hourly



Home3 Daily



Home3 Hourly



Performance Tuning:

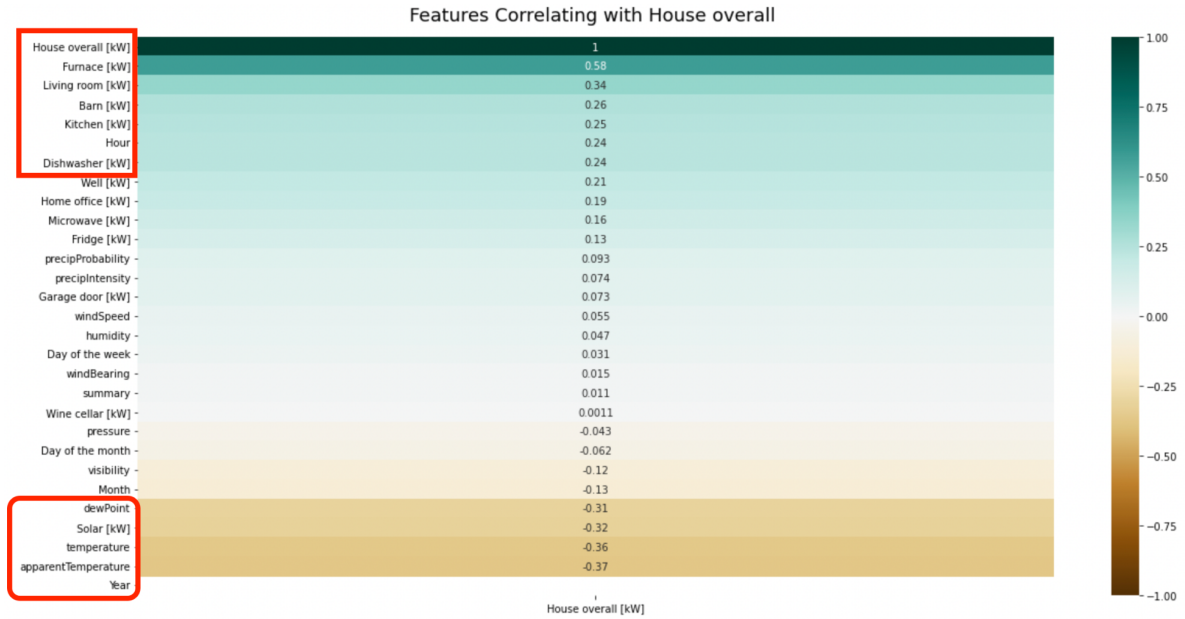
We systematically trained the LSTM with all 29 features. But it resulted in poor performance and a high MAE. We tuned the performance by using just relevant features. We found them out using the Correlation heatmap between features from weather data and energy heatmap.

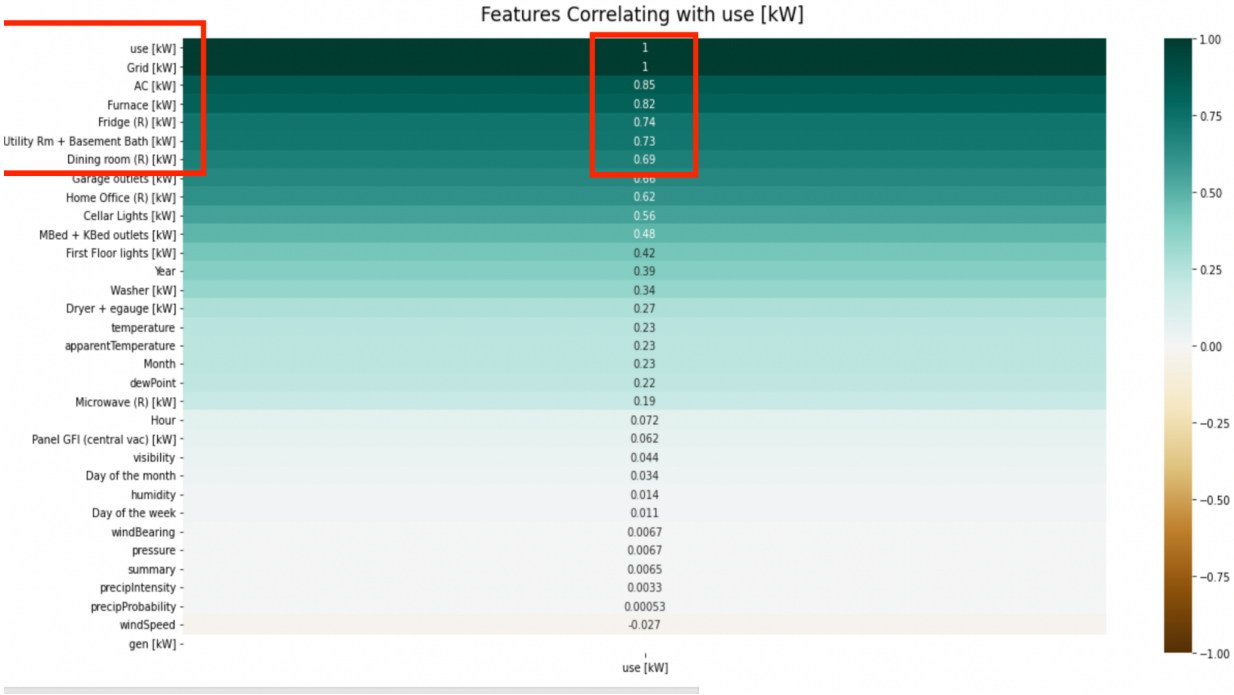
Since each Home has a different set of features we set a unique threshold for each dataset and considered features with a higher correlation coefficient than this.

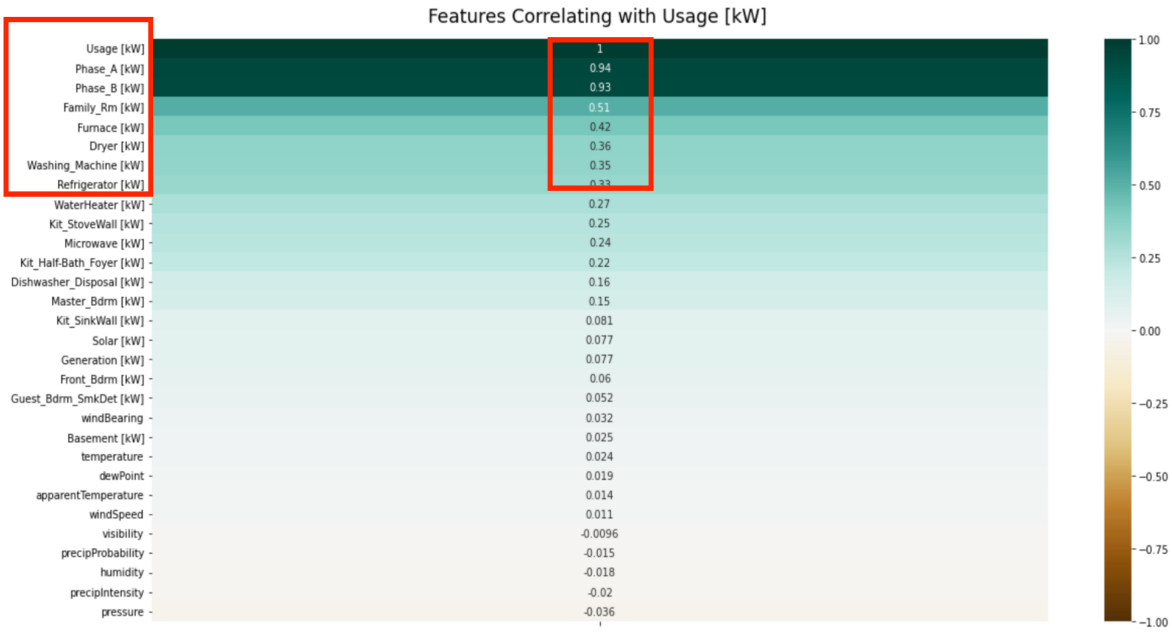
Home 1: Furnace, barn, living room, Kitchen, time of the day and dishwasher are all above the set absolute threshold 0.25

Home 2: Fridge, PhaseA, PhaseB, Family Room, Dryer, Washing Machine, Refrigerator are all above the set absolute threshold of 0.35

Home 3: AC, Furnace, Fridge. Dining Room, Utility Room / Basement are all above the set absolute threshold 0.70


Features Correlating with House overall
House overall [kW] 1
Furnace [kW] 0.58
Living room [kW] 0.34
Barn [kW] 0.26
Kitchen [kW] 0.25
Hour 0.24
Dishwasher [kW] 0.24
Well [kW] 0.21
Home office [kW] 0.19
Microwave [kW] 0.16
Fridge [kW] 0.13
precipProbability 0.093
precipIntensity 0.074
Garage door [kW] 0.073
windSpeed 0.055
humidity 0.047
Day of the week 0.031
windBearing 0.015
summary 0.011
Wine cellar [kW] 0.0011
pressure -0.043
Day of the month -0.062
visibility -0.12
Month -0.13
dewPoint -0.31
Solar [kW] -0.32
temperature -0.36
apparentTemperature -0.37
Year
House overall [kW]
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Features Correlating with use [kW]
use [kW] 1
Grid [kW] 1
AC [kW] 0.85
Furnace [kW] 0.82
Fridge (R) [kW] 0.74
Utility Rm + Basement Bath [kW] 0.73
Dining room (R) [kW] 0.69
Home Office (R) [kW] 0.62
Cellar Lights [kW] 0.56
MBed + KBed outlets [kW] 0.48
First Floor lights [kW] 0.42
Year 0.39
Washer [kW] 0.34
Dryer + egauge [kW] 0.27
temperature 0.23
apparentTemperature 0.23
Month 0.23
dewPoint 0.22
Microwave (R) [kW] 0.19
Hour 0.072
Panel GFI (central vac) [kW] 0.062
visibility 0.044
Day of the month 0.034
humidity 0.014
Day of the week 0.011
windBearing 0.0067
pressure 0.0067
summary 0.0065
precipIntensity 0.0033
precipProbability 0.00053
windSpeed -0.027
gen [kW]
use [kW]
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



Reducing the number of features and using highly correlated features improved the MAE manifold.

Results

Daily intervals:

	Home 1	Home 2	Home 3	Mean MAE
Naive daily MAE	9.83	58.35	258.44	108.87
ARIMA daily MAE	6.714	47.512	219.590	91.272
LSTM daily MAE	9.14	41.53	55.6	35.42

Hourly intervals

	Home 1	Home 2	Home 3	Mean MAE
Naive hourly MAE	0.80	3.43	19.01	7.74666667
ARIMA hourly MAE	0.539	1.960	11.778	4.759
LSTM hourly MAE	0.56	1.92	8.136	3.54

*Best performing model's MAE is highlighted

Insights and conclusion

- The baseline model provided a good starting point for our models. The ARIMA model improves over the baseline, however the LSTM model performed the best.
- We can see that the LSTM model performs significantly better for Home 3 compared to other models. The reason is that performance of LSTM models improve as the amount of available training data increases. We see that the LSTM model converges correctly even when there is a huge spike in Home 3 data in July 2016.
- Deep learning models operate via an iterative optimization algorithm, which obtains the results multiple times and selects the best results which minimize the errors. This offers a potential explanation why our LSTM model performed better than ARIMA.
- LSTM performs better on hourly plots as compared to daily as the number of data points are more in hourly plots.

Contribution and work distribution

Name	**Contribution percentage**	**Areas worked on**
Akanksha Kale (SBU ID: 113788594)	33.33%	Deep learning models, visualizations, Data cleaning, Feature Engineering
Akhila Juturu (SBU ID: 114777498)	33.33%	Data preparation, exploratory data analysis, ARIMA models
Mandar Laxmikant Mahajan (SBU ID: 113276053)	33.33%	Data cleaning and preparation, deep learning models, baseline models, visualizations, assisted with ARIMA models for home 3

References

Time Series Forecasting in Real Life: Budget forecasting with ARIMA:
https://towardsdatascience.com/time-series-forecasting-in-real-life-budget-forecasting-with-arima-d5ec57e634cb

LSTM Guide:
https://towardsdatascience.com/energy-consumption-time-series-forecasting-with-python-and-lstm-deep-learning-model-7952e2f9a796