Jobathon Nov 2022 Time Series Forecasting Approach

Table of Contents

Model(s)' Performance

Various models used in this competition and its performance on Leaderboard and validation set are as below:

Model	Private LB	Public LB	Validation Score
Final (UC Model with aggregated mean features)	436.4	264.9	210.1
UC Model with OHE Hour Features	386.6	294.3	259.21
UC Model with Holiday Features	389.3	304.5	259.17
UC Model without exogenous Features	404.5	302.1	279.2
Baseline Model	617.9	941.4	431.7

Exploratory Data Analysis

Libraries Used

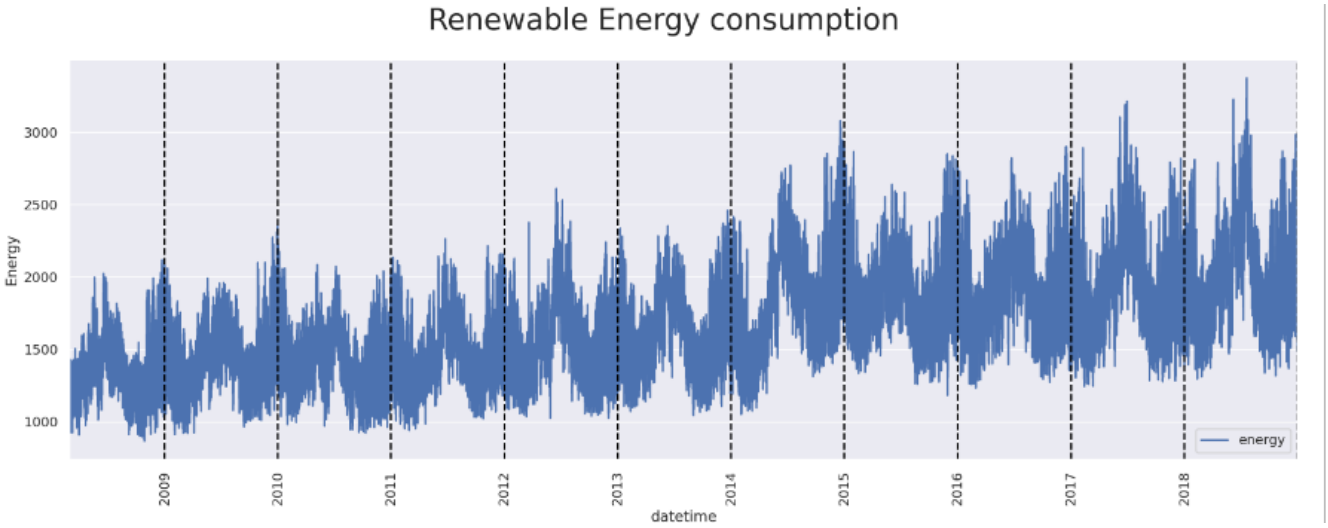
- statsmodels.tsa.seasonal_decompose
- statsmodels.graphics.tsaplots.plot_acf, statsmodels.graphics.tsaplots.plot_pacf
- pandas,numpy,matplotlib,datetime

Visualizing Data

Code to visualize

```
#visual checking of data. Plotting by Pandas method, drawing axes by Matplotlib
f, ax = plt.subplots(figsize=(18,6),dpi=200);
plt.suptitle('Renewable Energy consumption', fontsize=24);
train.plot(ax=ax,rot=90,ylabel='Energy');
xcoords = ['2008-01-01','2009-01-01','2010-01-01', '2011-01-01', '2012-01-01', '2013-01-01', '2014-01-0
           '2015-01-01', '2016-01-01', '2017-01-01', '2018-01-01', '2019-01-01']
for xc in xcoords:
    plt.axvline(x=xc, color='black', linestyle='--')
```

Let us visualize the data where each vertical dashed line corresponds to the start of the year.



The data seem to have a seasonal variation. Also, the energy seem to increase, but not significantly over time since the mean energy has almost the same value regardless of the year.
Also, within each point (I.e) each day, energy has major variation and probably could be due to different hour and this pattern could be understood when we dig further at hour level analysis subsequently.

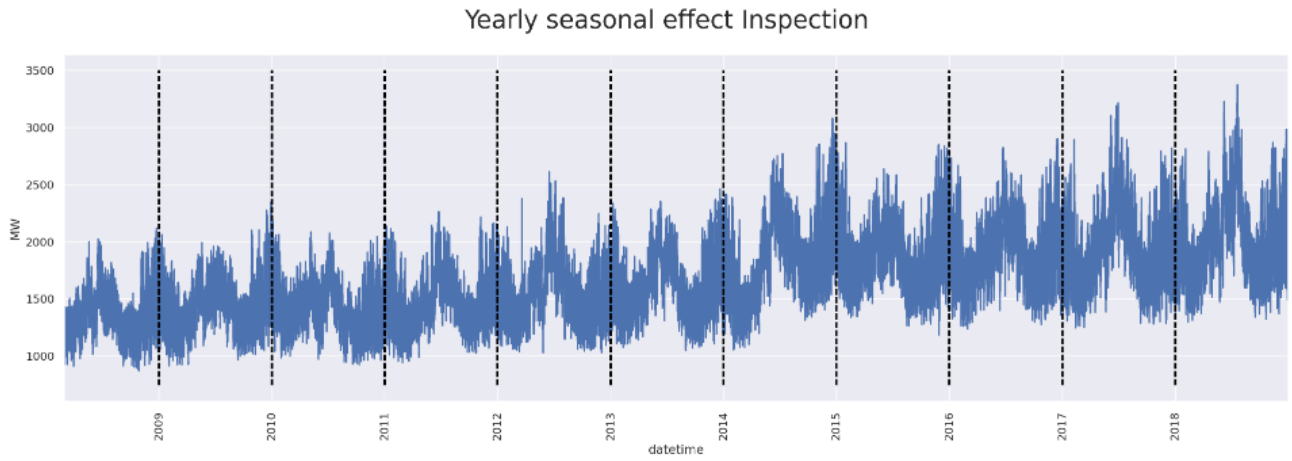
Check for Seasonality

Check for Yearly seasonality

code to check yearly seasonality

```
#checking presence of yearly seasonality
f, axes = plt.subplots(dpi=200,figsize=(18,6));
f.tight_layout(pad=3.0);
plt.suptitle('Yearly seasonal effect Inspection', y = 1.01, fontsize=24);

#plotting and drawing vertical lines at start of every year
idx = [train.loc[x:x+1*train.index.freq,:].index.values for x in train.index if x.dayofyear == 1]
train.plot(ax=axes, legend=False, rot=90, ylabel='MW');
axes.vlines(idx, axes.get_ylim()[0], axes.get_ylim()[1], colors='black', linestyles='dashed');
```

Let us visualize the data where each vertical dashed line corresponds to the start of the year.



From the above chart we can clearly see that pattern repeats every year and hence yearly seasonal effect is present.

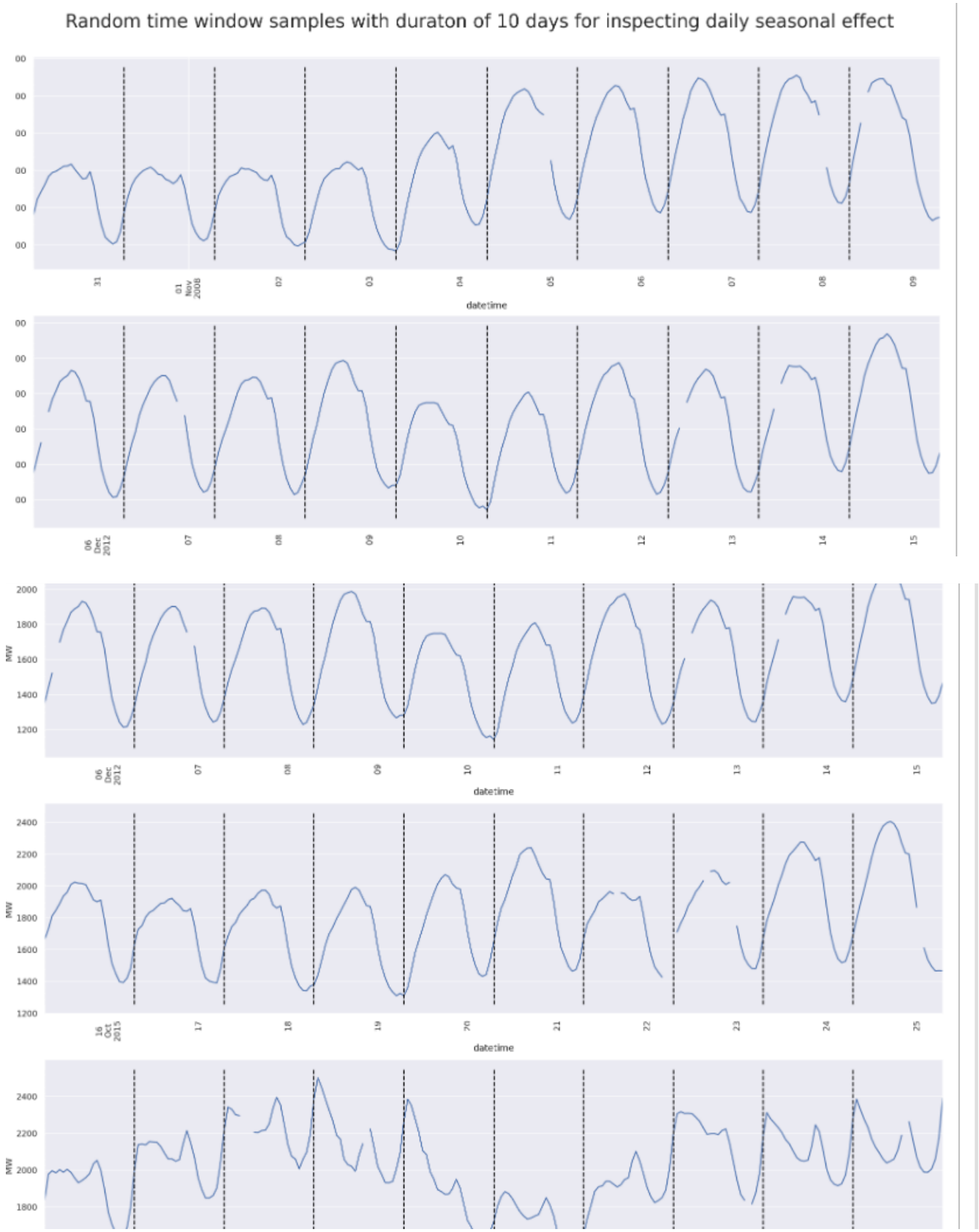
Check for 24 hours seasonality

For visualizing 24 hours data, let us select five random time windows and plot at different scales for checking any seasonal patterns:

code for visualizing 24 hours data

```
#drawing a random sample of 5 indices without repetition from sub sample, where current time is 04:00
sample = sorted([x for x in np.random.choice([x for x in train.index[:-241] if x.hour == 7],5,replace=F

#checking persistence of daily seasonality by inspecting 5 random time window of 10 days
f, axes = plt.subplots(len(sample),1,dpi=200,figsize=(18,24));
f.tight_layout(pad=3.0)
plt.suptitle('Random time window samples with duraton of 10 days for inspecting daily seasonal effect',

#plotting time windows and drawing vertical lines at 06:00 each day
for si,s in enumerate(sample):
    ids = train.index.to_list().index(s)
    idx = [train.iloc[x:x+1,:].index.values for x in range(ids, ids + 240,24)]
    train.iloc[ids:(ids+241),:].plot(ax=axes[si], legend=False, rot=90, ylabel='MW');
    axes[si].vlines(idx, axes[si].get_ylim()[0], axes[si].get_ylim()[1], colors='black', linestyles='da
```

Let us visualize the data where each vertical dashed line corresponds to 7:00 AM every day



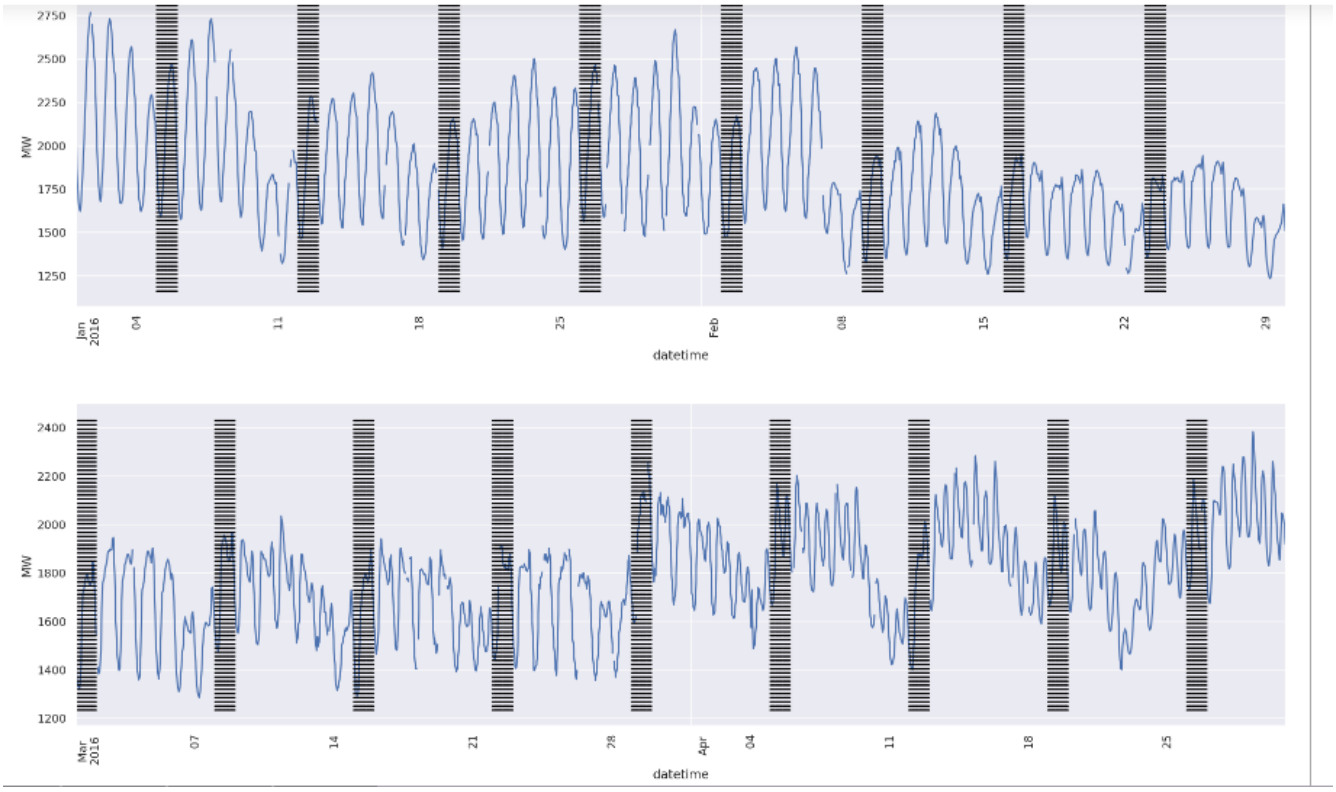
From the above chart we can clearly see that pattern repeats every day for most of the cases (i.e) every 24 hours and hence daily seasonal effect is present

Check for Weekly seasonality

To check weekly seasonality, let us consider 4 different months from one of the year (Jan -Feb 2016 and Mar-April 2016) and plot.

```
f, axes = plt.subplots(2,1,dpi=200,figsize=(18,12))
filt = [(train.index.month <=2) ,
        (train.index.month >2) & (train.index.month <=4) ]
#checking presence of weekly seasonality
for i in range(2):
    train_filt = train[(train.index.year==2016) & filt[i]]
    f.tight_layout(pad=3.0);
    plt.suptitle('Weekly seasonal effect Inspection', y = 1.01, fontsize=24);

    #plotting and drawing vertical lines at every start of week
    idx = [train_filt.loc[x:x+1*train_filt.index.freq,:].index.values for x in train_filt.index
               if (x.dayofweek == 1)]
    train_filt.plot(ax=axes[i], legend=False, rot=90, ylabel='MW')
    axes[i].vlines(idx, axes[i].get_ylim()[0], axes[i].get_ylim()[1], colors='black', linestyles='dotte
```

Let us visualize the data where each vertical dashed line always corresponds to fixed day of the week



From the above chart we can see that seasonality pattern exists for week where last 2 days (weekends) have lower energy consumption compared to other days.

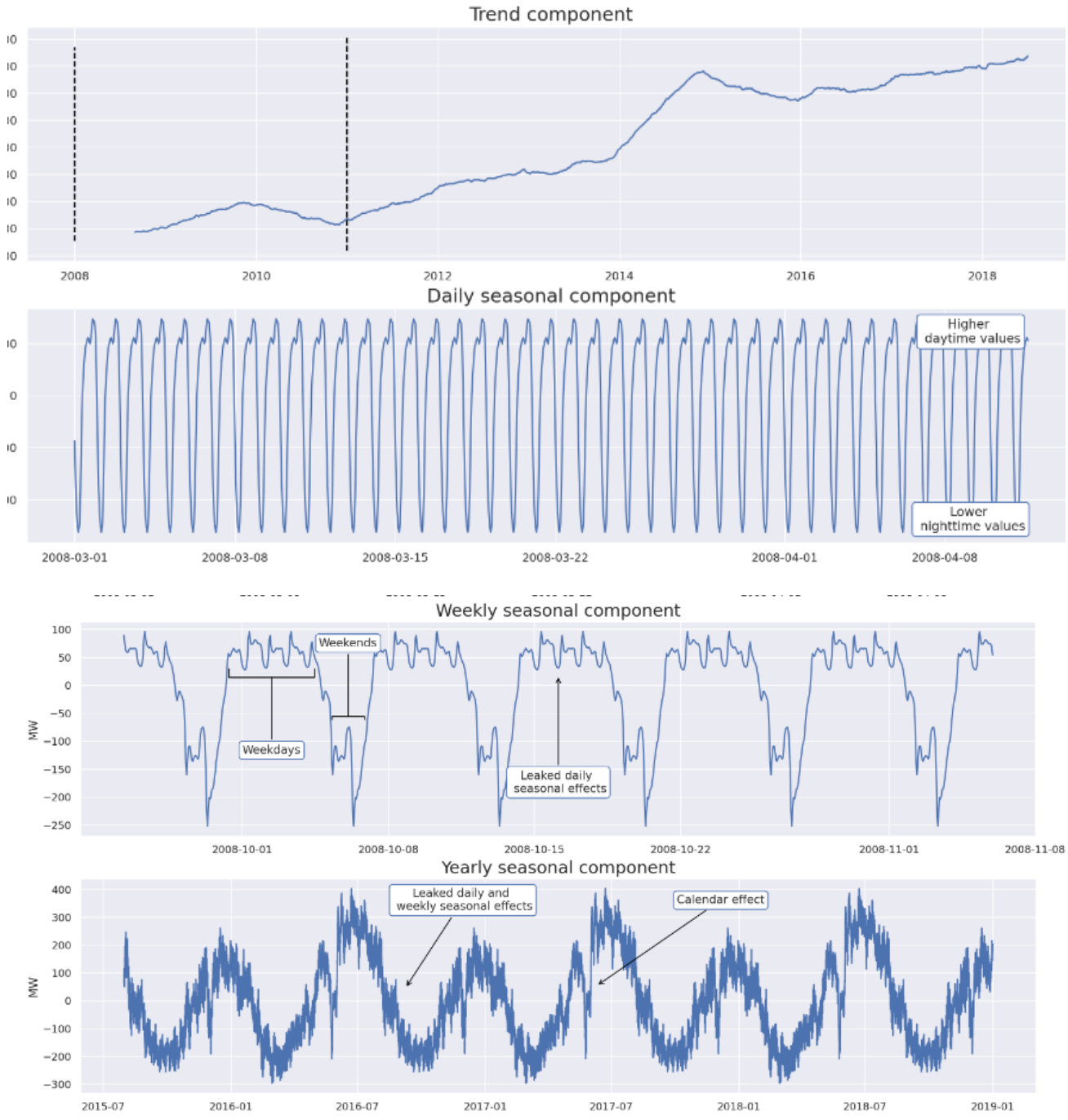
Seasonal Decomposition

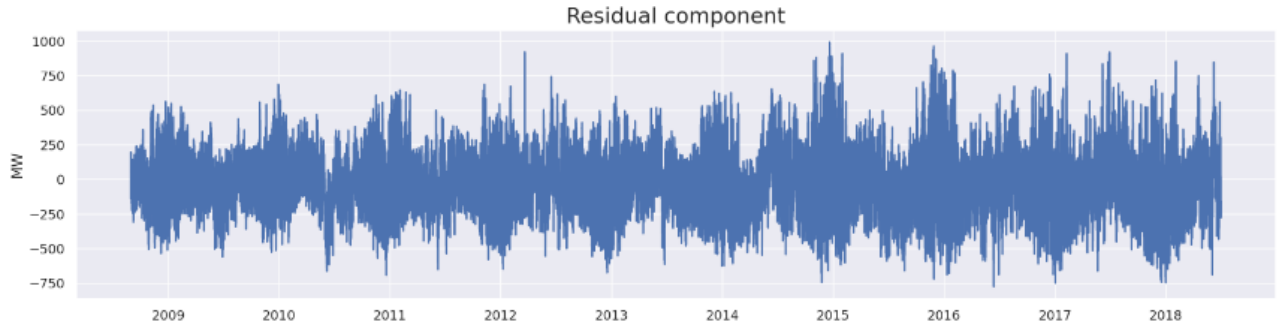
Using seasonal decomposition (from statsmodel api), one can get the seasonal and trend components of the energy data. Using this method, we can also confirm all above seasonality pattterns.

```
#extracting daily seasonality from raw time series
sd_24 = sm.tsa.seasonal_decompose(train, period=24)

#extracting weekly seasonality from time series adjusted by daily seasonality
sd_168 = sm.tsa.seasonal_decompose(train - np.array(sd_24.seasonal).reshape(-1,1), period=168)

#extracting monthly seasonality from time series adjusted by daily seasonality
sd_720 = sm.tsa.seasonal_decompose(train - np.array(sd_168.seasonal).reshape(-1,1), period=720)

#extracting yearly seasonality from time series adjusted by daily and weekly seasonality
sd_8766 = sm.tsa.seasonal_decompose(train - np.array(sd_720.seasonal).reshape(-1,1), period=8766)
```

```
f, axes = plt.subplots(5,1,figsize=(18,24),dpi=200);

#setting figure title and adjusting title position and size
plt.suptitle('Summary of seasonal decomposition', y=0.92, fontsize=24);

#plotting trend component
axes[0].plot(sd_8766.trend)
axes[0].set_title('Trend component', fontdict={'fontsize': 18});

#drawing black dashed vertical lines between y axis limits
axes[0].vlines(datetime.datetime(2008,1,1), axes[0].get_ylim()[0], axes[0].get_ylim()[1], colors='black', linestyles='dashed');
axes[0].vlines(datetime.datetime(2011,1,1), axes[0].get_ylim()[0], axes[0].get_ylim()[1], colors='black', linestyles='dashed');

#plotting daily seasonal component
axes[1].plot(sd_24.seasonal[:1000]);
axes[1].set_title('Daily seasonal component', fontdict={'fontsize': 18});

#plotting weekly seasonal component
axes[2].plot(sd_168.seasonal[5000:6000]);
axes[2].set_title('Weekly seasonal component', fontdict={'fontsize': 18});

#plotting yearly seasonality
axes[3].plot(sd_8766.seasonal[-30000:]);
axes[3].set_title('Yearly seasonal component', fontdict={'fontsize': 18});

#plotting residual of decomposition
axes[4].plot(sd_8766.resid);
axes[4].set_title('Residual component', fontdict={'fontsize': 18});

#setting label for each y axis
for a in axes:
    a.set_ylabel('MW');
```

Let us plot trend component and seasonal decomposition component for different subset of data for different seasonality.


Summary of seasonal decomposition
Trend component
2008
2010
2012
2014
2016
2018
Daily seasonal component
Higher daytime values
Lower nighttime values
2008-03-01
2008-03-08
2008-03-15
2008-03-22
2008-04-01
2008-04-08
Weekly seasonal component
Weekends
Weekdays
Leaked daily seasonal effects
MW
2008-10-01
2008-10-08
2008-10-15
2008-10-22
2008-11-01
2008-11-08
Yearly seasonal component
Leaked daily and weekly seasonal effects
Calendar effect
MW
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01



From the above charts the following are the **inferences**:

- Seasonality pattern confirmed for
  - **Daily** (24 hour)
  - **Weekly**
  - **Yearly**

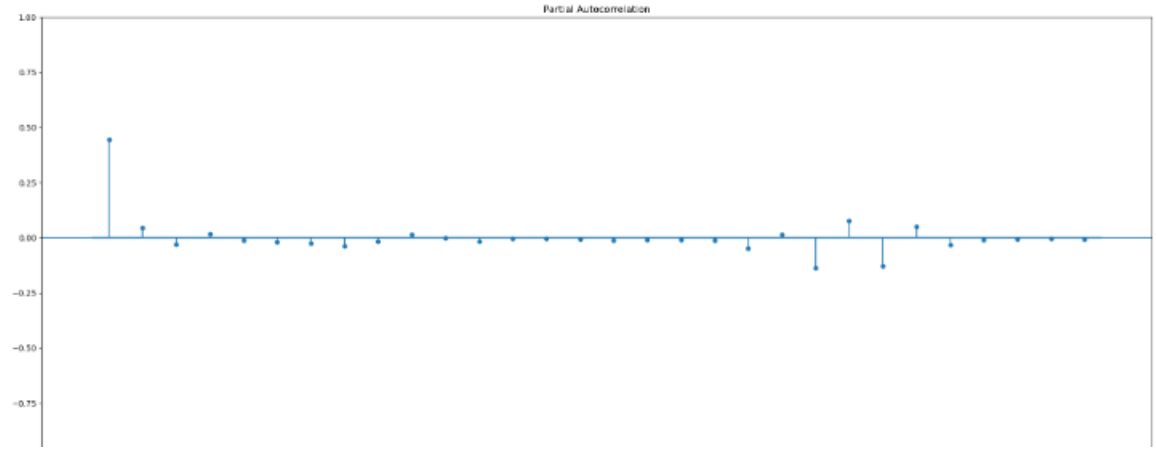
- Trend is **linear**. Trend could be either stochastic or deterministic. In this case, the slope of the trend is almost same except during 2015 where there is sudden increase, but in later years, it rolled back to the earlier slope. Hence it would be better to treat the data using **deterministic** trend.

Determination of order of Auto Regression and Moving Average

ACF and PACF plots are used to determine order of AR and MA.

PACF Plot

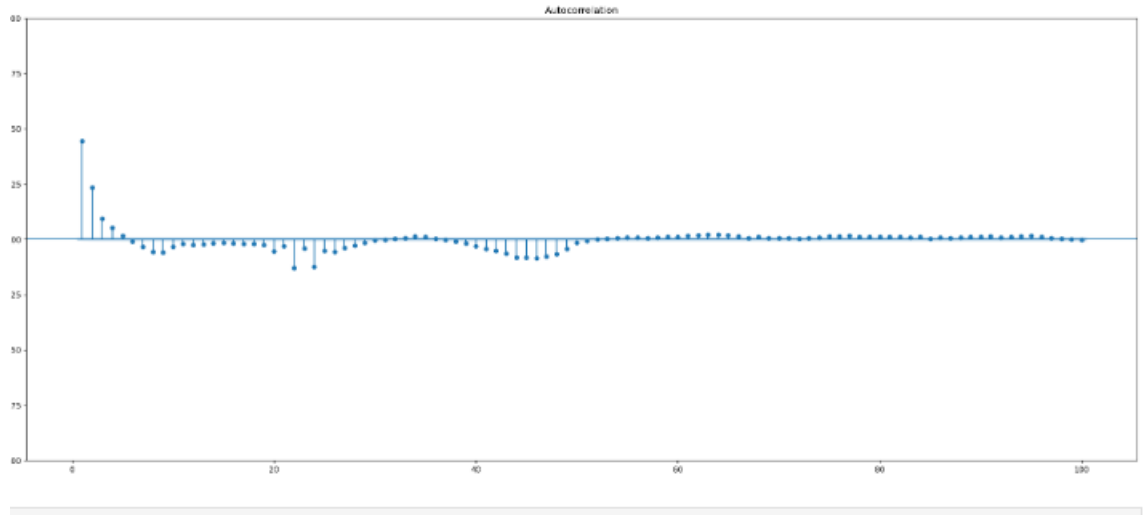
```
fig,ax=plt.subplots(figsize=(25,10))
plot_pacf(train['energy'].diff().diff(24).dropna(), ax=ax,lags = 30,zero=False)
plt.show()
```



PACF plot, there are only 2 significant lags and this plot determines AR component of ARIMA model.So AR order (p) could be atmost 2.

ACF Plot

```
fig,ax=plt.subplots(figsize=(25,10))
plot_acf(train['energy'].diff().diff(24).dropna(), ax=ax,lags = 100,zero=False)
plt.show()
```



Here ACF plot tails off while PACF plot cutsoff at certain point.
So the MA component would be zero (i.e q=0) and AR component would be 2 (p=2).

Model Development

Model Training and Evaluation

This process would be used to decide which model and features would perform optimal forecasting of the given timeseries data.

To have validation set similar to forecast horizon (ie test data has 3 years of data), train test split is performed as following:

- Validation Set: 3 years period with data from 01-Jan-2016 to 31-Dec-2018
- Train Set: Remaining 8 years with data from 01-Mar-2008 to 31-Dec-2015

Model Finalization

In this process, the model with entire train dataset is fitted and perform predictions on test dataset using this model.

Baseline Performance

- Before choosing model, a simple baseline prediction has been carried out and then the model benchmark would be compared against this baseline
    - All values in validation set predictions are simply set as single fixed value using mean of the train set values.
    - Validation Baseline RMSE: 431.73

Hence machine learning models need to perform much better on validation set compared to this baseline prediction.

Why specific model is chosen ?

Libraries Used

- sm.tsa.UnobservedComponents
- pmdarima
- RAPIDS AI cudf, cuml.tsa.arima

Constraints with direct ARIMA / SARIMA models

SARIMA model can provide state of the art solutions to time series forecasting. But it has two major drawbacks:
(1) can model only a single seasonal effect,
(2) season length cannot be too long.

But SARIMA models can still be used for multiple seasonality indirectly using manually computed Fourier transformed exogenous features. But even if it has been done, the other constraint with these models is that the computation time of the corresponding statstools API for the given data is long.

Faster Alternative to statstools ARIMA / SARIMA

So another alternative could be to use GPU enabled API from RAPIDS AI ARIMA package. It performs much faster than statstools ARIMA packages, but still it consumed in terms of hours when high seasonality such as 24 hours or more (for yearly seasonality) is used.
And of course, the performance of ARIMA model with single seasonality (just 24 hours) also poor (RMSE 422.02 just closer to baseline performance)

Model Requirement for given data
So we required a reliable model which support multiple seasonalities and at the sametime also execute faster. So the model that satisfies these constraints is Unobserved Components package.

Why the model is chosen ?

Unobserved Components package supports both requirements of supporting multiple seasonalities at relatively much faster execution time

Brief Intro on Unobserved Components Model

Unobserved Components Model (UCM) performs a time series decomposition into components such as trend, seasonal, cycle, and the regression effects due to predictor series.

The execution time comparison of ARIMA packages and the Unobserved Components package is given below.

Model	**Execution Time**
RAPIDS ARIMA	123 min (with GPU)
Unobserved Component	3 min

Configuration for Unobserved components model

- Seasonality period are 24 hours, 168 hours (weekly) and 8766 hours (yearly) respectively
- **Trend** has been set as **Local Linear Deterministic Trend** which is captured using level parameter value as *lldtrend*.
  This has been chosen because the trend is deterministic based upon the inference from Exploratory Analysis.
  Also ”Local Linear” implies that we would like to capture the ARMA process in prediction so that the model would perform better.
- ARMA regression value of “**2**” is selected from one of the best value from PACF and ACF plots during Exploratory Analysis

  Code to fit Unobserved components model for given time-series data

```
#Unobserved Components model definition
model_UC1 = sm.tsa.UnobservedComponents(y_train,
                                        autoregressive=2,
                                        level='lldtrend',
                                        exog=exog_train,
                                        cycle=False,
                                        irregular=False,
                                        stochastic_level = False,
                                        stochastic_trend = False,
                                        stochastic_freq_seasonal = [False,False,True],
                                        freq_seasonal=[{'period': 24, 'harmonics': 1},
                                                       {'period': 168, 'harmonics': 1},
                                                       {'period': 8766, 'harmonics': 2}
                                                      ]
                                       )
#fitting model to train data
model_UC1res = model_UC1.fit()
```

Checking Model Performance from Result Statistics

Result Summary can be checked using below code after performing model fit.

```
#printing statsmodels summary for model
print(model_UC1res.summary())
```

The results summary of the Unobserved Components model on validation set is as below.

From the below summary, it can be seen that the p-value for all predictors are below 0.05 and hence null hypothesis of coefficient=0 can be rejected. So it can be clearly confirmed that all predictors are very good predictors (I.e) these predictors have strong relation with the target value according to this model.

	coef	std err	z	P>|z|	[0.025	0.975]
sigma2.irregular	1.5727	1.794	0.877	0.381	-1.943	5.088
sigma2.level	578.8350	40.838	14.174	0.000	498.795	658.875
sigma2.freq_seasonal_8766(2)	3.5556	20.056	0.177	0.859	-35.753	42.864
sigma2.ar	11.5898	1.688	6.867	0.000	8.282	14.898
ar.L1	1.8119	0.016	113.499	0.000	1.781	1.843
ar.L2	-0.8156	0.016	-50.080	0.000	-0.848	-0.784
beta.month_hour_mean	0.3223	0.005	68.430	0.000	0.313	0.332
beta.quarter_hour_mean	-0.1112	0.006	-19.349	0.000	-0.123	-0.100
beta.weekofyear_mean	0.5446	0.005	102.430	0.000	0.534	0.555
beta.dayofweek_grp_mean	0.2693	0.003	87.407	0.000	0.263	0.275

RMSE of the final model on validation set: 210.1

This model performed much better than baseline model performance of 431.33

Feature Engineering

Software / Packages Used:

Tableau

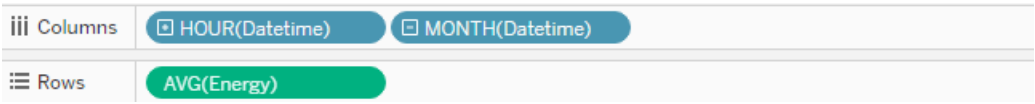
Though the time series model is capable of performing independent forecasting without any features, adding exogenous features would enforce the model to perform a dynamic regression and thereby improve the model performance.

The exploration of mean energy for each hour is done for each of the calendar group for month-wise, quarter-wise, week of year wise and day of week wise.
Tableau Visualization tool is used for exploration since it can quickly generate charts for timestamp related aggregation features.

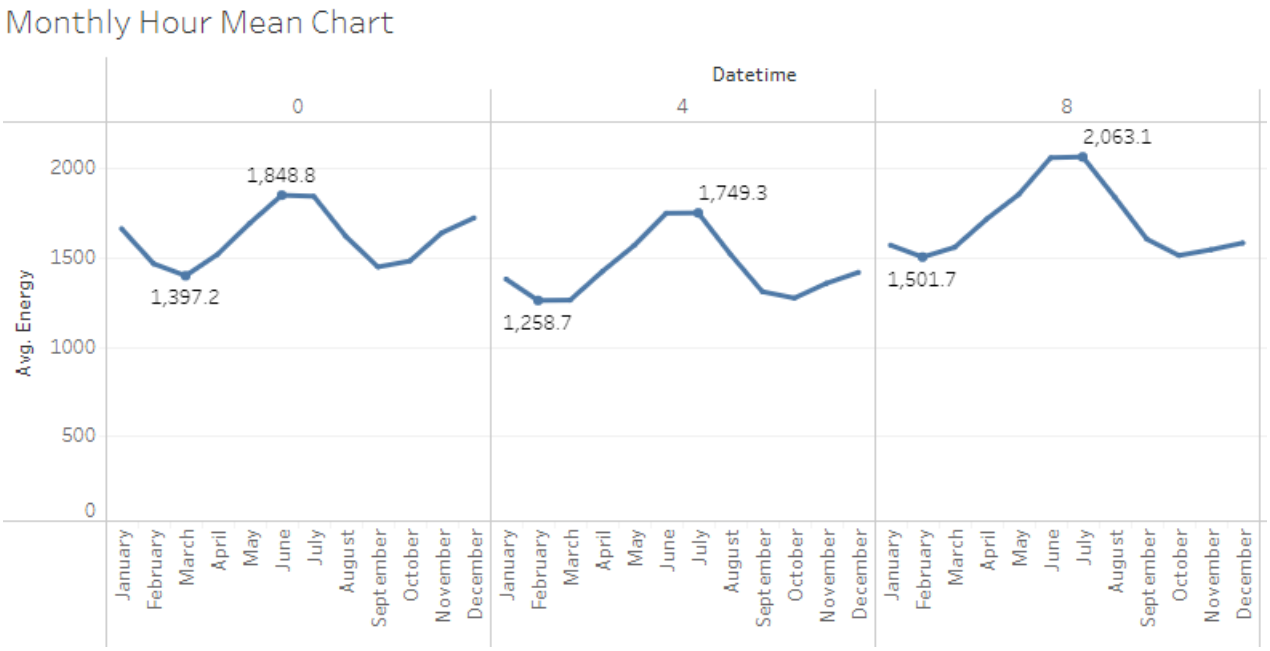
Monthly-Hourly Average Energy

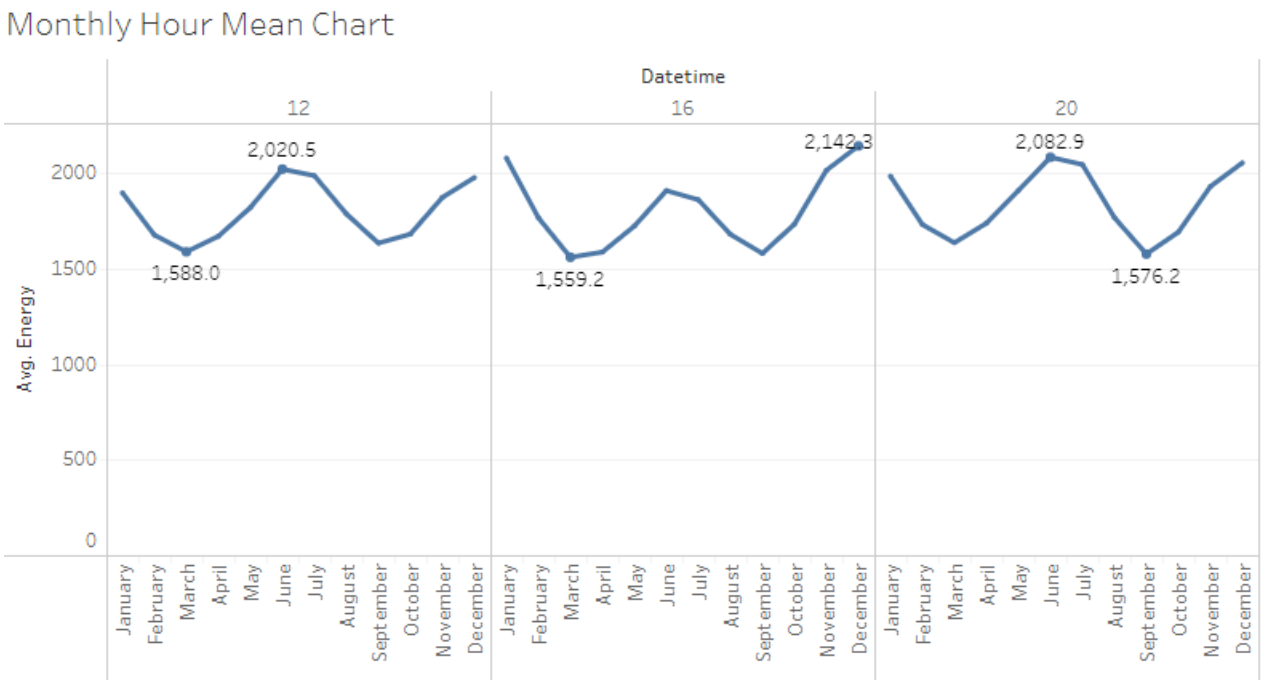
The below chart contains mean energy for month and hour combination.

Tableau selection options to generate the chart



Let us visualize the data for some of random hours for every month . Here labels corresponds to minimum and maximum energy values for the specific hour



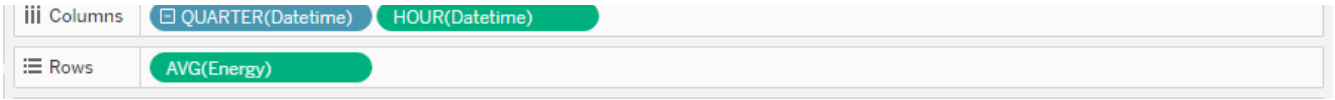


From the above charts, though at high level it looks like similar pattern, but if one spends time to look into these charts, there are considerable differentiation of the values of the target (energy) as well as the patterns for certain hours. So this feature could be useful for forecasting.

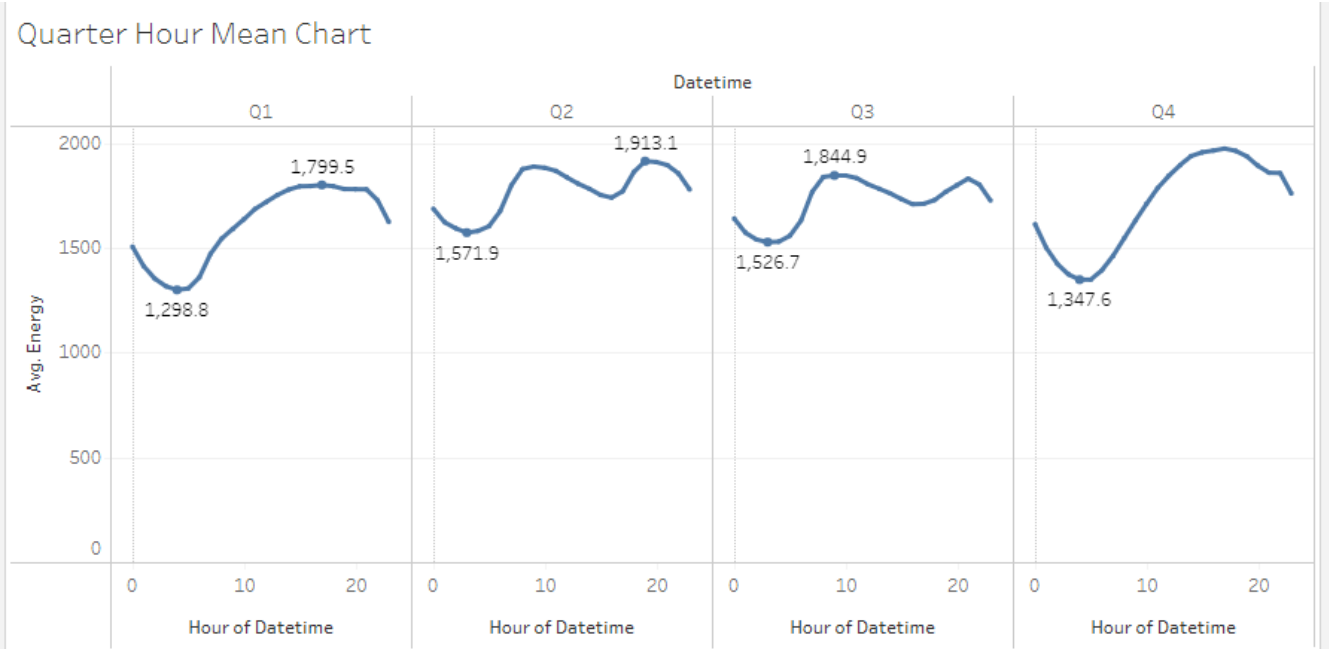
Quarterly-Hourly Average Energy

The below chart contains mean energy for quarter and hour combination.

Tableau selection options to generate the chart



Let us visualize the data for some of random hours for every quarter. Here labels corresponds to minimum and maximum energy values for the specific quarter.

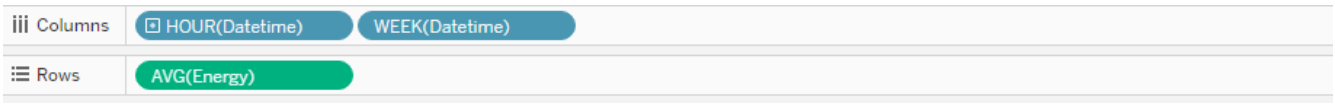


From the above charts, there is clear differentiation of the pattern between different quarters and also values of the target (energy) for different hours. So this feature could be useful for forecasting.

Week of Year-Hourly Average Energy

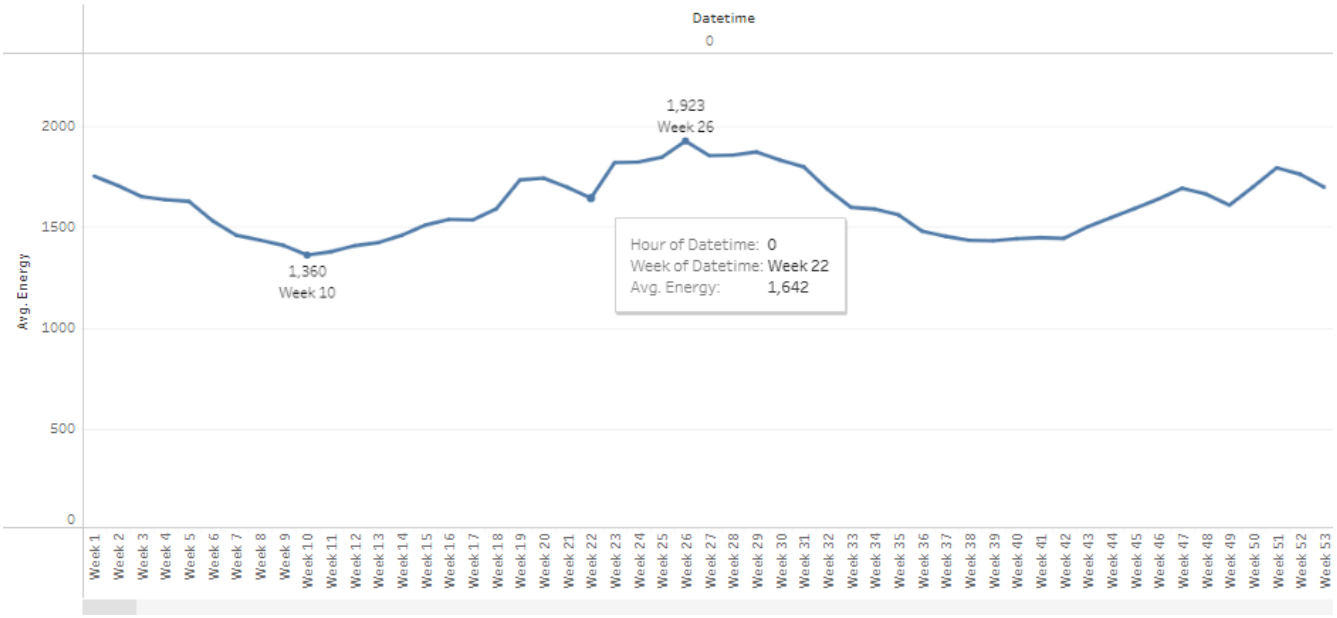
The below chart contains mean energy for week of year and hour combination.

Tableau selection options to generate the chart

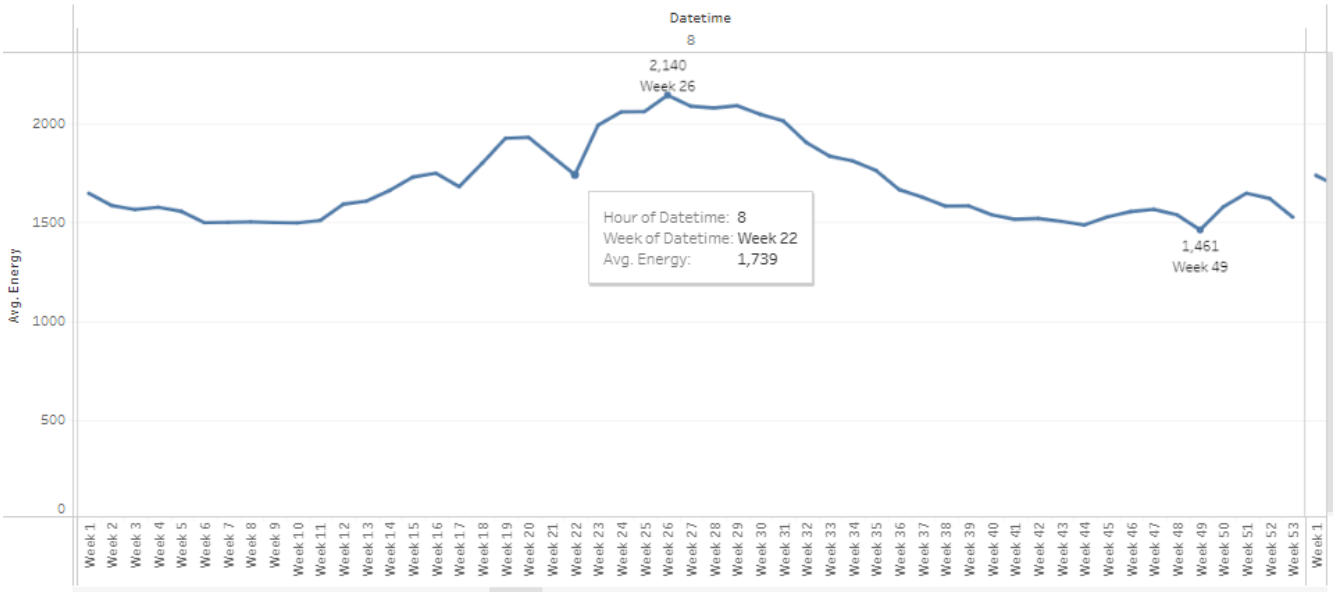


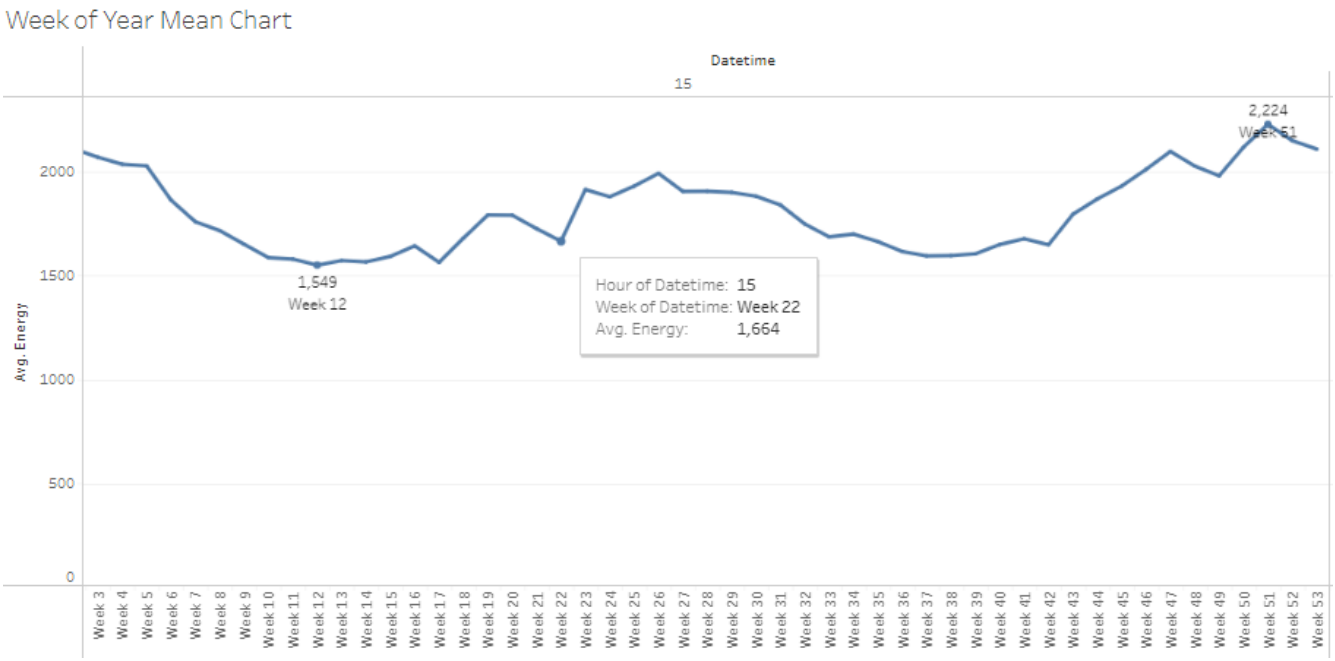
Let us visualize the data for some of random hours (0,8 and 15) for every week. Here labels corresponds to minimum and maximum energy values for the specific hour.

Week of Year Mean Chart



Week of Year Mean Chart





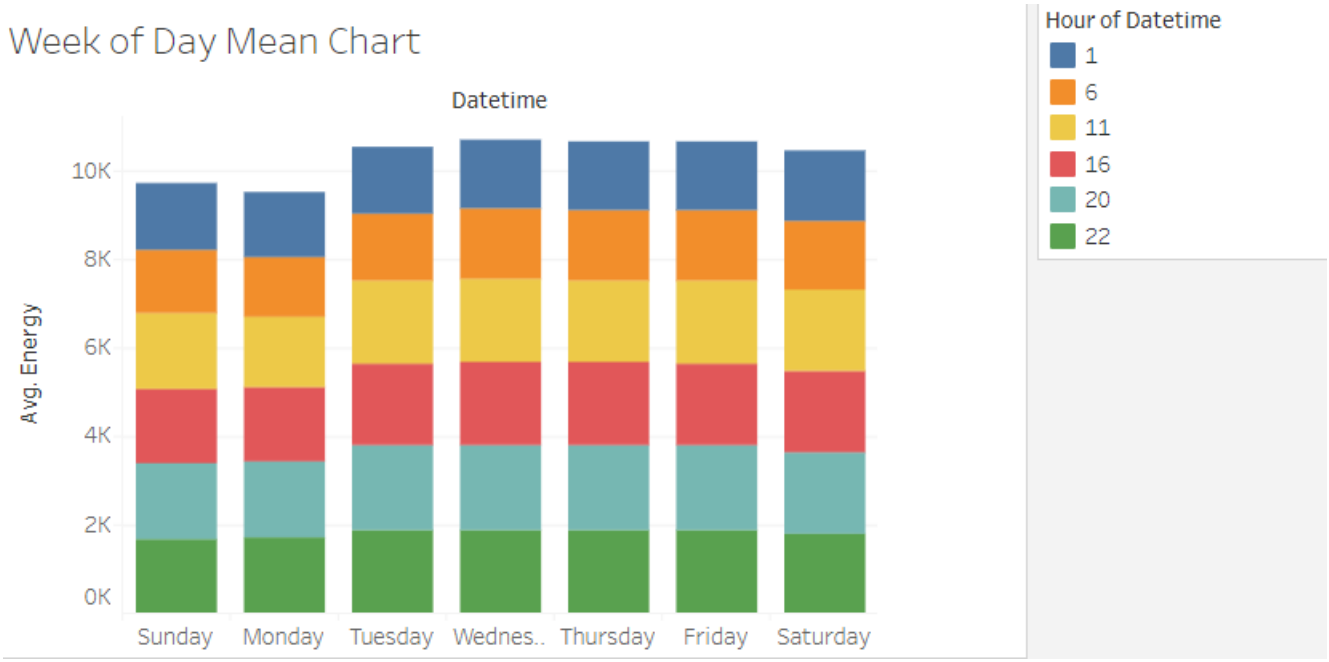
From the above charts, though at high level it looks like similar pattern, but if one spends time to look into these charts, there are considerable differentiation of the values of the target (energy) as well as some part of the pattern (especially at ends of the pattern). So this feature could be useful for forecasting.

Week Day-Hourly Average Energy

The below chart contains mean energy for week day and hour combination.

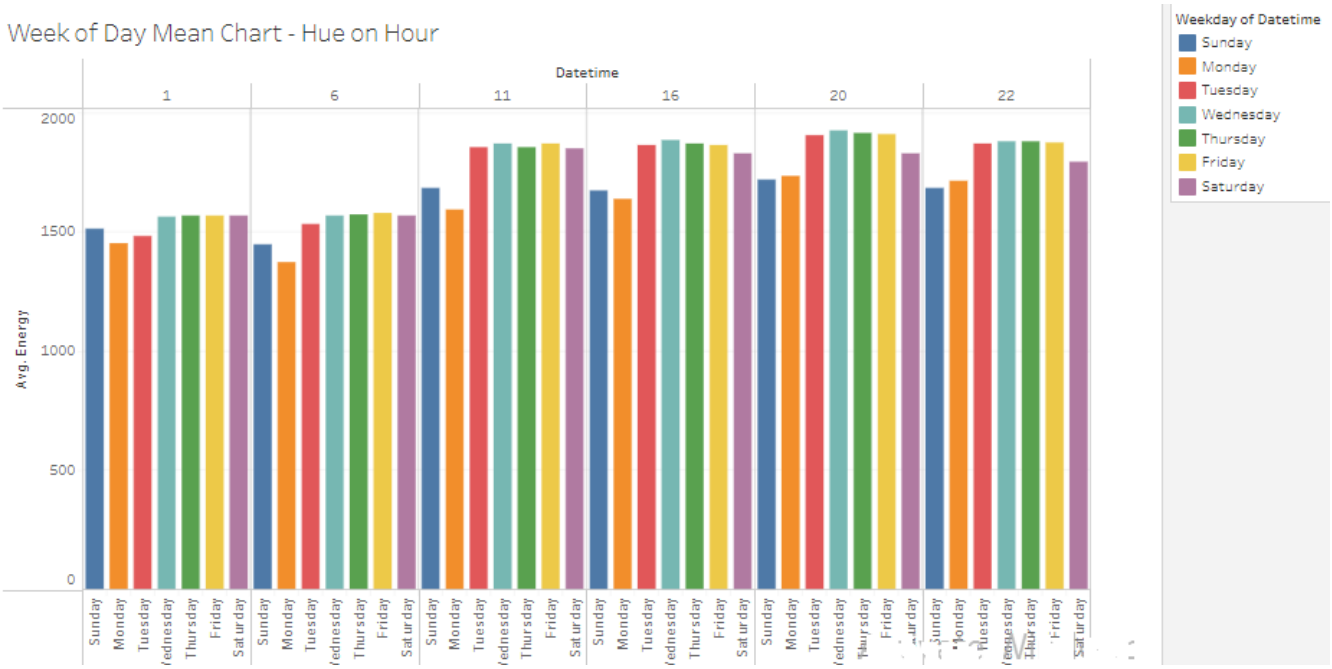
Tableau selection options to generate the chart

Let us visualize the data for every week day with colored hue on random hours.



Here from above chart, it can be seen that the days Sunday, Monday have lowest energy values, Saturday have next low energy values. So there is a clear differentiation of energy values among week day.

Let us also visualize the data for every random hour but with colored hue on week day.



From the above pattern, it can be confirmed that different hours exhibit different patterns when week day is used. So the week day feature could also be useful for forecasting.

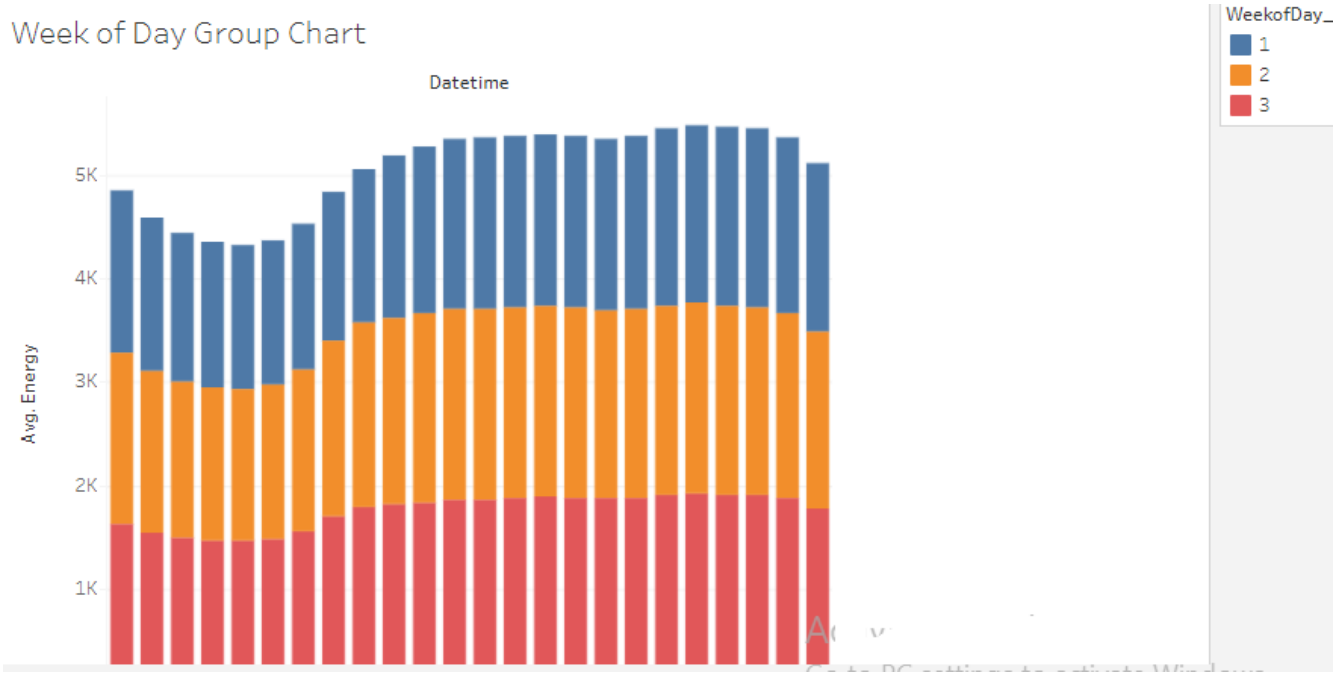
Here we could notice that other week days have almost same energy values and hence these week days may not be differentiator among themselves. To fine tune this pattern, week day can be grouped into a new feature as mentioned in next section.

Week Day Group-Hourly Average Energy

Week day can be grouped as follows:

- Group 1: Monday and Sunday
- Group 2: Saturday
- Group 3: Remaining days

With this grouping, visualization of hourly mean chart is as below with x-axis having hour starting from 0 and color hue for week day group.



Here it can be clearly seen that lot of pattern differences for various hours can be seen. So week day group hour mean feature is preferable over just week day hour mean.

Code that generate the Features

```
def gen_mean_feats(train,test,cols,newcolname):
    grouped=train.groupby(cols)[targetcol].mean().reset_index()
    grouped.columns=cols+[newcolname]
    train=train.merge(grouped,on=cols)
    test=test.merge(grouped,on=cols)
    return train,test

def gen_mean_feats_all(train,test):
    train,test=gen_mean_feats(train,test,['month','hour'],'month_hour_mean')
    train,test=gen_mean_feats(train,test,['quarter','hour'],'quarter_hour_mean')
    train,test=gen_mean_feats(train,test,['weekofyear','hour'],'weekofyear_mean'
    train,test=gen_mean_feats(train,test,['dayofweek_grp','hour'],'dayofweek_grp

    train.sort_values('datetime',inplace=True)
    train.reset_index(drop=True,inplace=True)
    test.sort_values('datetime',inplace=True)
    test.reset_index(drop=True,inplace=True)
    return train,test
```