Assignment 2 & 3

Introduction

Language used: Python

Tools used: Jupyter Notebooks and Docker

Process: Exploratory Data Analysis, Feature Engineering, Feature Selection.

Sreerag Mandakathil Sreenath
001838559
mandakathil.s@husky.neu.edu

Shreya Chudasama
001828562
chudasama.s@husky.neu.edu

Aahana Khajanchi
001824402
khajanchi.a@husky.neu.edu

Abstract

It is important to predict the energy demand and consumption of energy by houses as they contributes as the major part of energy consumption.We have conducted an in-depth analysis to provide insights on feature engineering and machine learning and predicting energy consumed by various equipment, seasonality and attributes like temperature and humidity and then predict aggregate energy use.

Exploratory Data Analysis

Performed Exploratory Data Analysis from the dataset available at https://github.com/LuisM78/Appliances-energy-prediction-data using the following libraries :-
1. Plotly
2. Matplotlib
3. Seaborn

We explored the data using the following libraries:-

1. MissingNo - which gives the missing values in our dataset
2. Pivottables - Drag and drop Pivot Tables and Charts to understand the dataset
3. PandasProfiling - It generates profile report and gives the summary
4. ipythonwidgets.interact - Automatically creating UI interface which gives controls for exploring code and data interactively
5. Logging - To log all the files

Generated Extra features:-

1. NSM - Number of Seconds from Midnight

```
In [5]: #It will give the number of rows and columns in a dataset
        data.shape

Out[5]: (19735, 29)
```

Added Number of Seconds From Midnight column

```
In [6]: logging.basicConfig(filename=logfilename, level=logging.DEBUG,
                                format='%(asctime)s - %(levelname)s - %(message)s')
        logging.debug('Added Number of Seconds From Midnight column ')

        try :
            data['NSM'] = pd.to_datetime(data['date'])
            (data['NSM'].dt.hour*60 + data['NSM'].dt.minute)*60 + data['NSM'].dt.second

        except :
            logging.ERROR('Addition of midnight column failed')
```

```
In [7]: data.shape

Out[7]: (19735, 30)
```

2. Classified Days of the week - Monday to Sunday

Generated the column weekday

```
In [17]: def dayoftheweek(day):
             if(day==0):
                 return("Monday")
             if(day==1):
                 return("Tuesday")
             if(day==2):
                 return("Wednesday")
             if(day==3):
                 return("Thurday")
             if(day==4):
                 return("Friday")
             if(day==5):
                 return("Saturday")
             if(day==6):
                 return("Sunday")
```

```
In [18]: data["dayoftheweek"] = data['date']
         data["dayoftheweek"] = data['dayoftheweek'].apply(lambda x: dayoftheweek(x.dayofweek))
         data.groupby('dayoftheweek').count()["date"]
         data.head
```

15	24.884962	24.884962	2016-01-11 19:30:00	Monday
16	35.880925	35.880925	2016-01-11 19:40:00	Monday
17	49.595305	49.595305	2016-01-11 19:50:00	Monday
18	19.001759	19.001759	2016-01-11 20:00:00	Monday
19	38.872261	38.872261	2016-01-11 20:10:00	Monday
20	46.735262	46.735262	2016-01-11 20:20:00	Monday
21	10.607126	10.607126	2016-01-11 20:30:00	Monday
22	32.583688	32.583688	2016-01-11 20:40:00	Monday
23	6.277755	6.277755	2016-01-11 20:50:00	Monday
24	13.361033	13.361033	2016-01-11 21:00:00	Monday
25	19.305705	19.305705	2016-01-11 21:10:00	Monday
26	0.669517	0.669517	2016-01-11 21:20:00	Monday
27	19.119398	19.119398	2016-01-11 21:30:00	Monday
28	43.484542	43.484542	2016-01-11 21:40:00	Monday
29	17.017450	17.017450	2016-01-11 21:50:00	Monday
...	...	...	...	...
19705	32.420348	32.420348	2016-05-27 13:10:00	Friday
19706	49.189027	49.189027	2016-05-27 13:20:00	Friday
19707	15.081162	15.081162	2016-05-27 13:30:00	Friday
19708	45.226866	45.226866	2016-05-27 13:40:00	Friday
19709	5.773431	5.773431	2016-05-27 13:50:00	Friday
19710	41.515044	41.515044	2016-05-27 14:00:00	Friday
19711	19.540642	19.540642	2016-05-27 14:10:00	Friday
19712	16.977597	16.977597	2016-05-27 14:20:00	Friday
19713	8.991420	8.991420	2016-05-27 14:30:00	Friday
19714	40.409885	40.409885	2016-05-27 14:40:00	Friday

3. Classified day as WeekDay or Weekend

Generated WeekDayType Column

```
def weekdaytype(day):
    if(day=="Saturday" or day == "Sunday"):
        return "weekend"
    else:
        return "Weekday"
```

```
data["WeekDayType"] = data["dayoftheweek"]
data["WeekDayType"] = data['WeekDayType'].apply(lambda x: weekdaytype(x))
data.groupby('WeekDayType').count()["date"]
```

```
WeekDayType
Weekday    14263
weekend     5472
Name: date, dtype: int64
```

4. TimeDelta - Represents a duration between two dates or times

```
years = data["date"].map(lambda x : x.year)
years_indata = years.unique()
print(years_indata)
```

```
[2016]
```

```
months = data["date"].map(lambda x: x.month)
months_indata = months.unique()
print(months_indata)
```

```
[1 2 3 4 5]
```

```
data["TimeDelta"] = (data["date"]-data["date"].shift()).fillna(0)
```

```
data.groupby('TimeDelta').count()["date"]
```

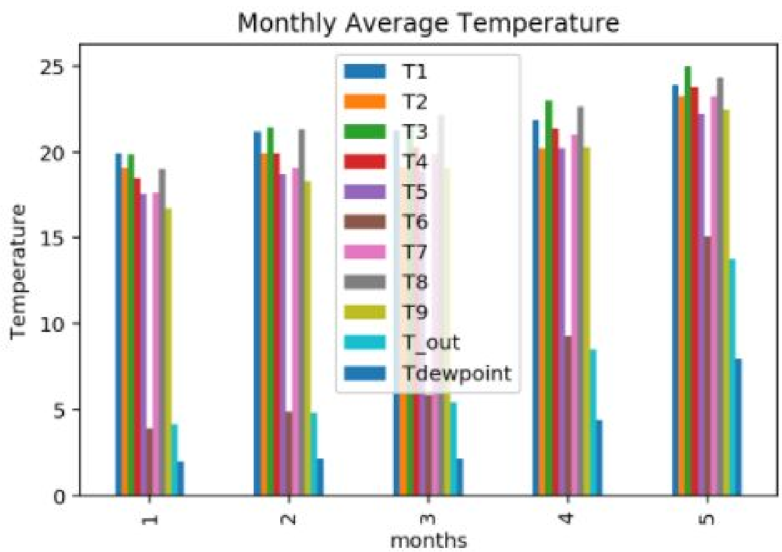
```
TimeDelta
00:00:00        1
00:10:00    19734
Name: date, dtype: int64
```

Plotted Graphs of the following:-

1. Monthly Average Temperature

Average of the total temperature was highest in the month of May

```
temp = ['T1', 'T2', 'T3', 'T4', 'T5', 'T6', 'T7', 'T8', 'T9', 'T_out', 'Tdewpoint']
df = data_new.groupby(['months'])[temp].mean()
df.plot.bar()
plt.ylabel('Temperature')
plt.title("Monthly Average Temperature")
plt.savefig("./Result/"+"Monthly Average Temperature.png")
```

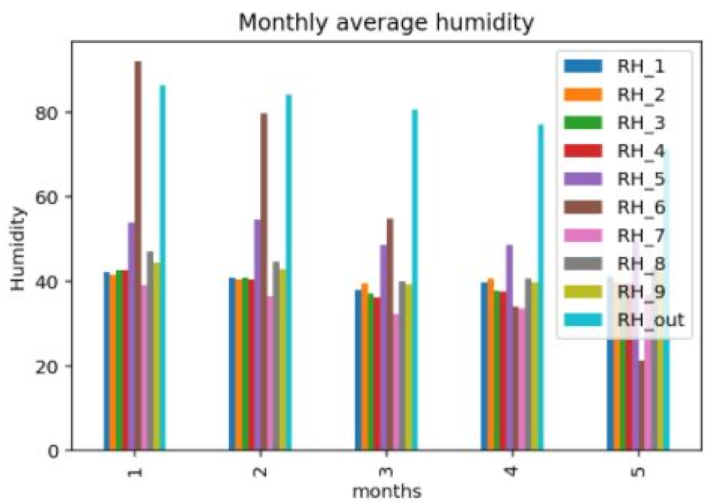


2. Monthly Average Humidity

Average of Overall Humidity was highest in the month of January

```
humidity = ['RH_1','RH_2','RH_3','RH_4','RH_5','RH_6','RH_7','RH_8','RH_9','RH_out']
```

```
df = data_new.groupby(['months'])[humidity].mean()
df.plot.bar()
plt.ylabel('Humidity')
plt.title("Monthly average humidity")
plt.savefig("./Result/"+"Monthly average humidity")
```



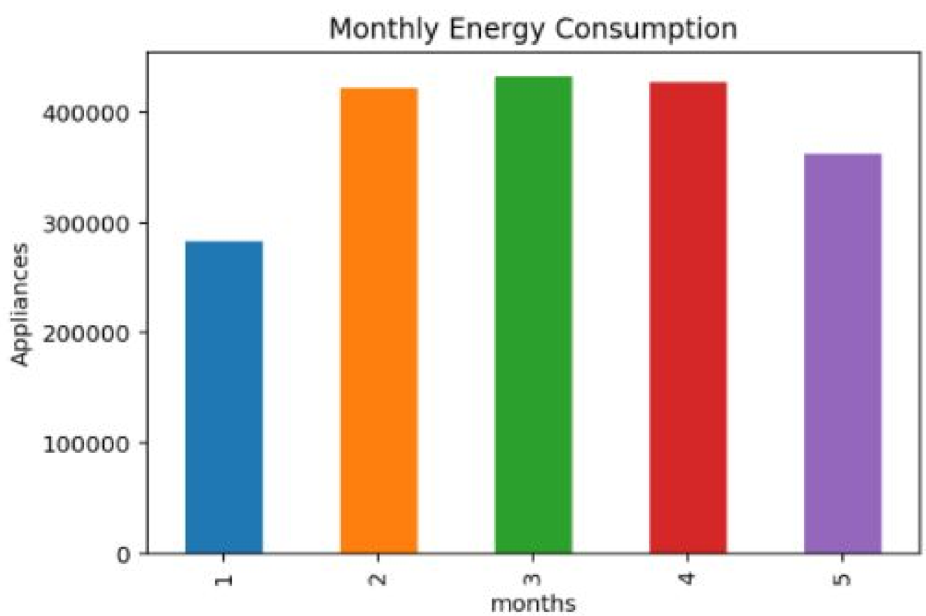
3. Monthly Energy Consumption

The monthly energy consumption is highest in March

```
#Monthly energy consumption
data_new.groupby(['months'])['Appliances'].apply(lambda x : sum(x))
```

```
months
1    283510
2    421550
3    432800
4    427200
5    362950
Name: Appliances, dtype: int64
```

```
df2 = data_new.groupby(['months'])['Appliances'].sum()
df2.plot.bar()
plt.ylabel('Appliances')
plt.title("Monthly Energy Consumption")
plt.savefig("./Result/"+"Monthly Energy Consumption.png")
```



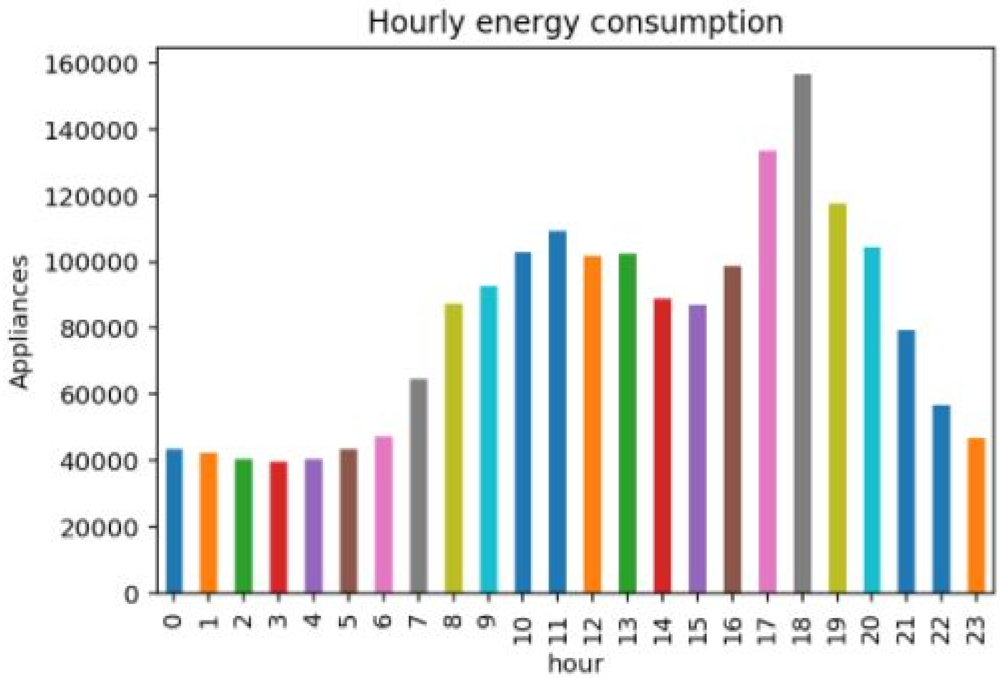
4. Hourly Energy Consumption

The hourly energy consumption is highest in evening

```
# Hourly energy consumption
data_new.groupby(['hour'])['Appliances'].apply(lambda x : sum(x))
```

```
hour
0      43390
1      42190
2      40340
3      39650
4      40570
5      43350
6      47440
7      64650
8      87250
9      92710
10    103060
11    109430
12    101630
13    102540
14     89010
15     86990
16     98560
17    133600
18    156670
19    117600
20    104380
21     79320
22     56840
23     46840
Name: Appliances, dtype: int64
```

```
df2 = data_new.groupby(['hour'])['Appliances'].sum()
df2.plot.bar()
plt.ylabel('Appliances')
plt.title("Hourly energy consumption")
plt.savefig("./Result/"+"Hourly energy consumption.png")
```



5. The function day_visual() shows the relation between any two parameters of a particular day in a month

Correlation Analysis

The function day_visual() shows the relation between any two parameters of a particular day in a month

```
%matplotlib inline
day_date = range(1, 32)
month_date = range(1, 6)
save = {False,True}
def day_visual(Day,Month, parameter1, parameter2,save):
    new_data = data[(data['date'].map(lambda x: x.day) == Day) & (data['date'].map(lambda x: x.month) == Month)]
    plt.subplot(211)
    title = "Plot of "+parameter1+" and " +parameter2+" vs time"
    plt.title("Plot of "+parameter1+" and " +parameter2+" vs time" )
    plt.plot(new_data['date'].map(lambda x: x.hour), new_data[parameter1])
    plt.xlabel("Time")
    plt.ylabel(parameter1)
    plt.subplot(212)
    plt.plot(new_data['date'].map(lambda x: x.hour), new_data[parameter2])
    plt.ylabel(parameter2)
    plt.xlabel("Time")
    if(save):
        plt.savefig("./Result/"+str(Month)+"_"+title.replace(" ","_")+".png")
    plt.show()
    return print("Correlation between ",parameter1," and ",parameter2,"is",str(new_data[parameter1].corr(new_data[parameter2])))
interact(day_visual,Day=day_date,Month=month_date, parameter1 = list(data),parameter2 = list(data), save=save)
```

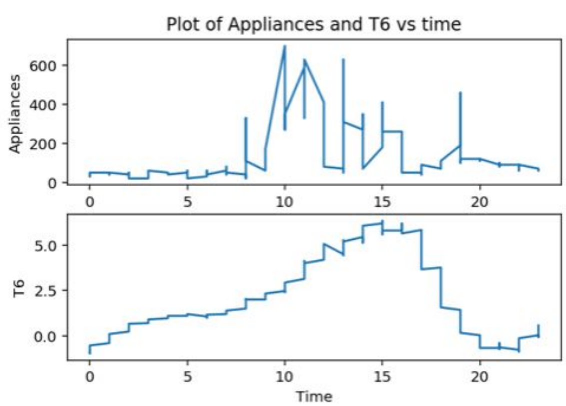
Day: 7

Month: 3

parameter1: Appliances

parameter2: T6

save: False



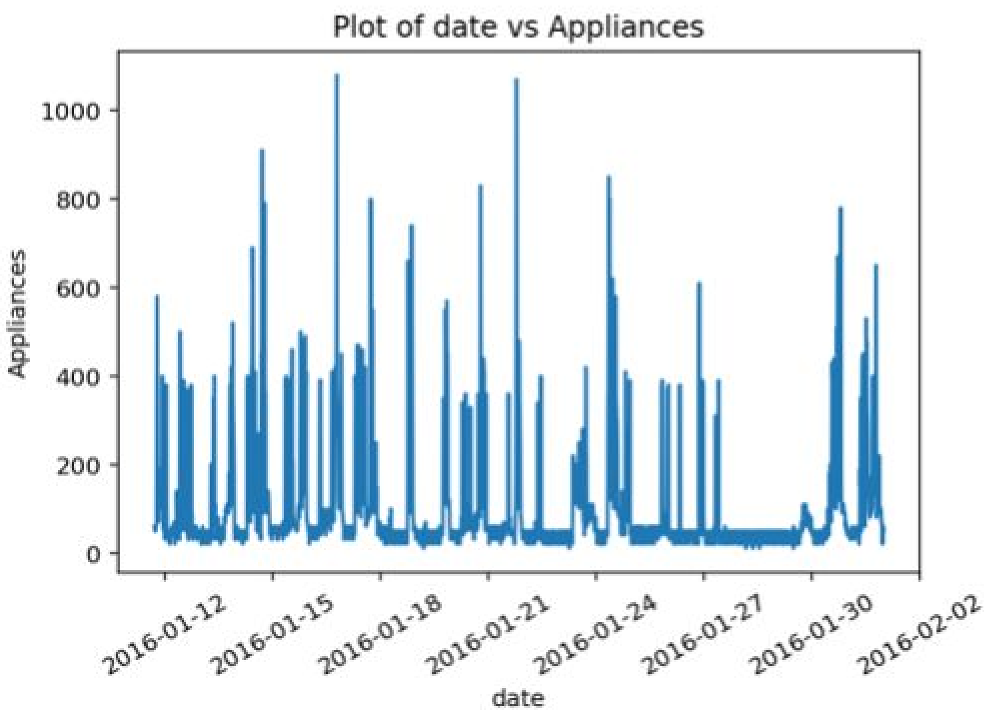

```
Correlation between  Appliances  and  T6 is 0.4732713114256746

<function __main__.day_visual>
```

6. The function month_visual shows the relation between any two parameters of a particular month

The function month_visual shows the relation between any two parameters of a particular month

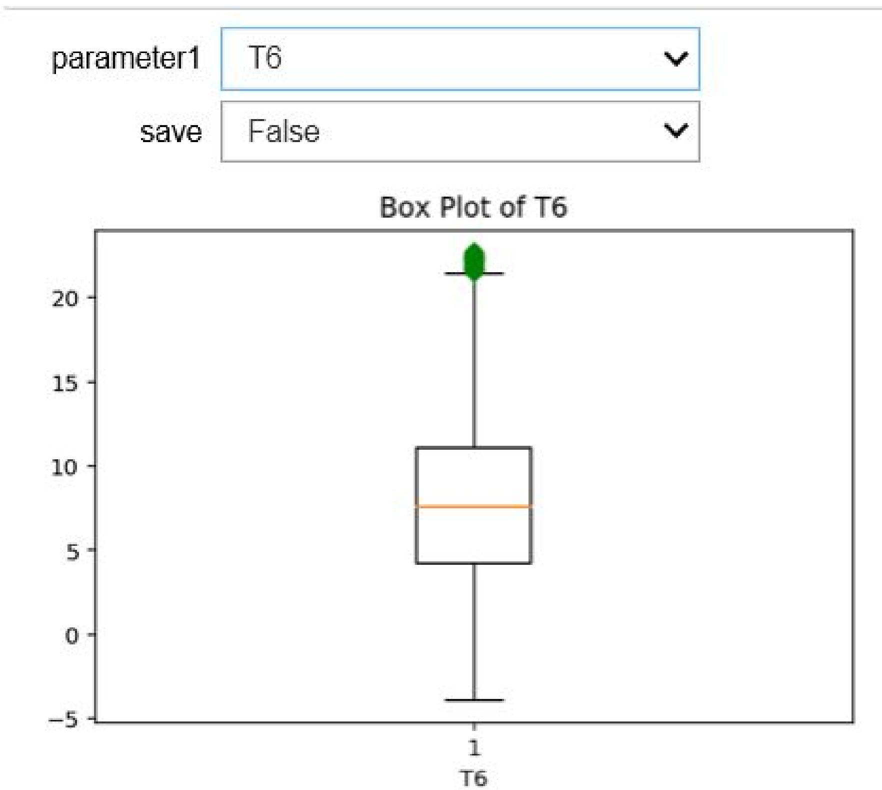
```
%matplotlib inline
day_date = range(1, 32)
month_date = range(1, 6)
save = {False,True}
def month_visual(Month, parameterx, parametery,save):
    new_data = data[(data['date'].map(lambda x: x.month) == Month)]
    title = "Plot of "+parameterx+" vs " +parametery
    plt.title(title)
    plt.plot(new_data[parameterx], new_data[parametery])
    plt.xlabel(parameterx)
    plt.ylabel(parametery)
    plt.xticks(rotation = 30)
    if(save):
        plt.savefig("./Result/"+str(Month)+"_"+title.replace(" ","_")+".png")
    plt.show()
interact(month_visual,Month=month_date, parameterx = list(data),parametery = list(data), save=save)
```



Detected the outliers using Boxplot.

The function box_visual shows the Quartile range and outliers of a particular variable

```
%matplotlib inline
day_date = range(1, 32)
month_date = range(1, 6)
save = {False,True}
def box_visual(parameter1,save):
    new_data = data
    title = "Box Plot of "+parameter1
    plt.title(title)
#     plt.figure()
    plt.boxplot(data[parameter1], 0,'gD')
    plt.xlabel(parameter1)
    if(save):
        plt.savefig("./Result/"+str(Month)+"_"+title.replace(" ","_")+".png")
    plt.show()
interact(box_visual,Month=month_date, parameter1 = list(data)[1:], save=save)
```



Detecting Outliers –

Proceed with the following steps to detect outliers in the dataset:

1. Arrange all the dataset points and calculate median
2. Calculate the upper quartile
3. Calculate the lower quartile
4. Calculate the interquartile range

   Product of numeric value of 1.5 and difference of the upper quartile (75%) and lower quartile (25%)
   - 1.5 X (Upper Quartile - Lower Quartile)
5. Calculate the inner fences for the dataset

   Set of numerical boundaries which is classified as major and minor outlier:
   - Major Outlier = Upper Quartile + Interquartile Range
   - Minor Outlier = Lower Quartile - Interquartile Range

```
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-(1.5*iqr)
    fence_high = q3+(1.5*iqr)
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] < fence_high)]
    return df_out
```

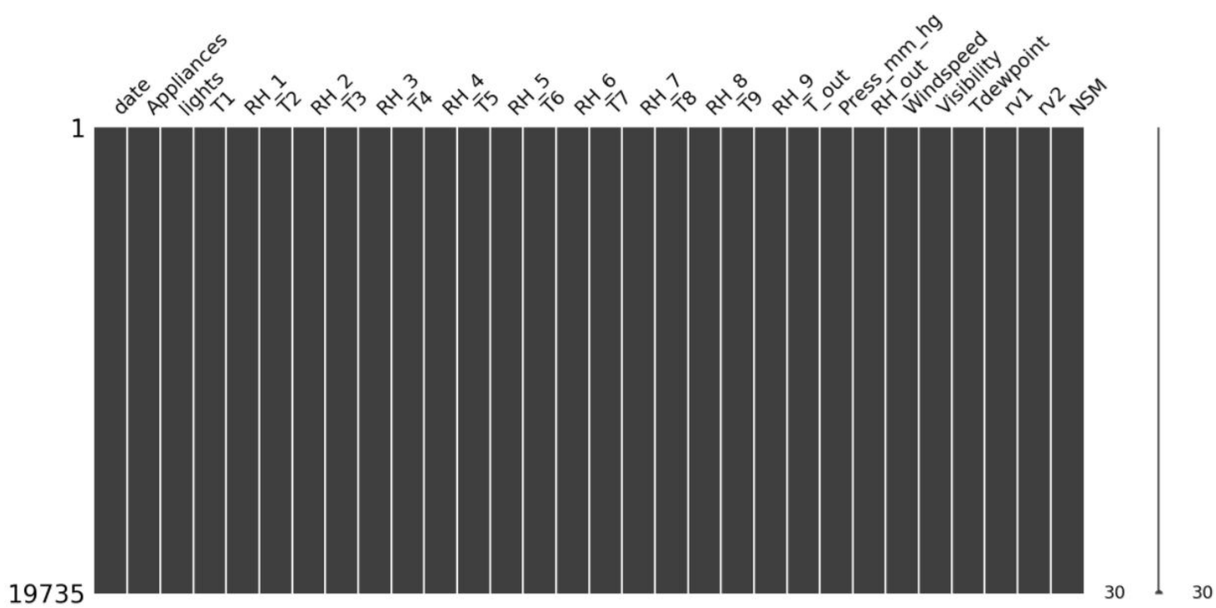
```
data_new = data
```

```
for column in list(data_new.drop(columns=['date','Appliances','lights','NSM','dayoftheweek','rv1','rv2','WeekDayType','TimeDelta'
#     print(column)
    data_new = remove_outlier(data_new, column)
data_new.info()
```

Detecting Missing Values

```
logging.basicConfig(filename=logfilename, level=logging.DEBUG,
                        format='%(asctime)s - %(levelname)s - %(message)s')
logging.debug('Checking any Missing Value ')
try :
    msno.matrix(data, figsize = (16, 7), )
    plt.savefig("./Result/CheckingMissingValue.png")

except :
    logging.ERROR("Failed to show the missing plot graph ")
```



There is no missing data, the dataset is clean

Generated the summary using PandasProfiling.

Generating profile report that summarizes the dataset

```
pandas_profiling.ProfileReport(data)
```

Overview

Dataset info

Number of variables	30
Number of observations	19735
Total Missing (%)	0.0%
Total size in memory	4.5 MiB
Average record size in memory	240.0 B

Variables types

Numeric	25
Categorical	0
Boolean	0
Date	1
Text (Unique)	1
Rejected	3
Unsupported	0

Warnings

- `T9` is highly correlated with `T7` (ρ = 0.94478) Rejected
- `T_out` is highly correlated with `T6` (ρ = 0.97479) Rejected
- `lights` has 15252 / 77.3% zeros Zeros
- `rv2` is highly correlated with `rv1` (ρ = 1) Rejected

Variables

Appliances
Numeric

Distinct count	92	Mean	97.695
Unique (%)	0.5%	Minimum	10
Missing (%)	0.0%	Maximum	1080
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

NSM
Date

Distinct count	19735	Minimum	2016-01-11 17:00:00
Unique (%)	100.0%	Maximum	2016-05-27 18:00:00
Missing (%)	0.0%		
Missing (n)	0		
Infinite (%)	0.0%		
Infinite (n)	0		

Press_mm_hg
Numeric

Distinct count	2189	Mean	755.52
Unique (%)	11.1%	Minimum	729.3
Missing (%)	0.0%	Maximum	772.3
Missing (n)	0	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		

RH_1
Numeric

Distinct count	2547	Mean	40.26
Unique (%)	12.9%	Minimum	27.023
Missing (%)	0.0%	Maximum	63.36

There are many more data in between from T1 to T9

date
Categorical, Unique

First 3 values
2016-01-23 19:30:00
2016-05-12 17:10:00
2016-04-04 05:10:00

Last 3 values
2016-02-27 05:00:00
2016-04-01 09:20:00
2016-05-10 14:20:00

lights
Numeric

Distinct count	8
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	3.8019
Minimum	0
Maximum	70
Zeros (%)	77.3%

rv1
Numeric

Distinct count	19735
Unique (%)	100.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	24.988
Minimum	0.0053217
Maximum	49.997
Zeros (%)	0.0%

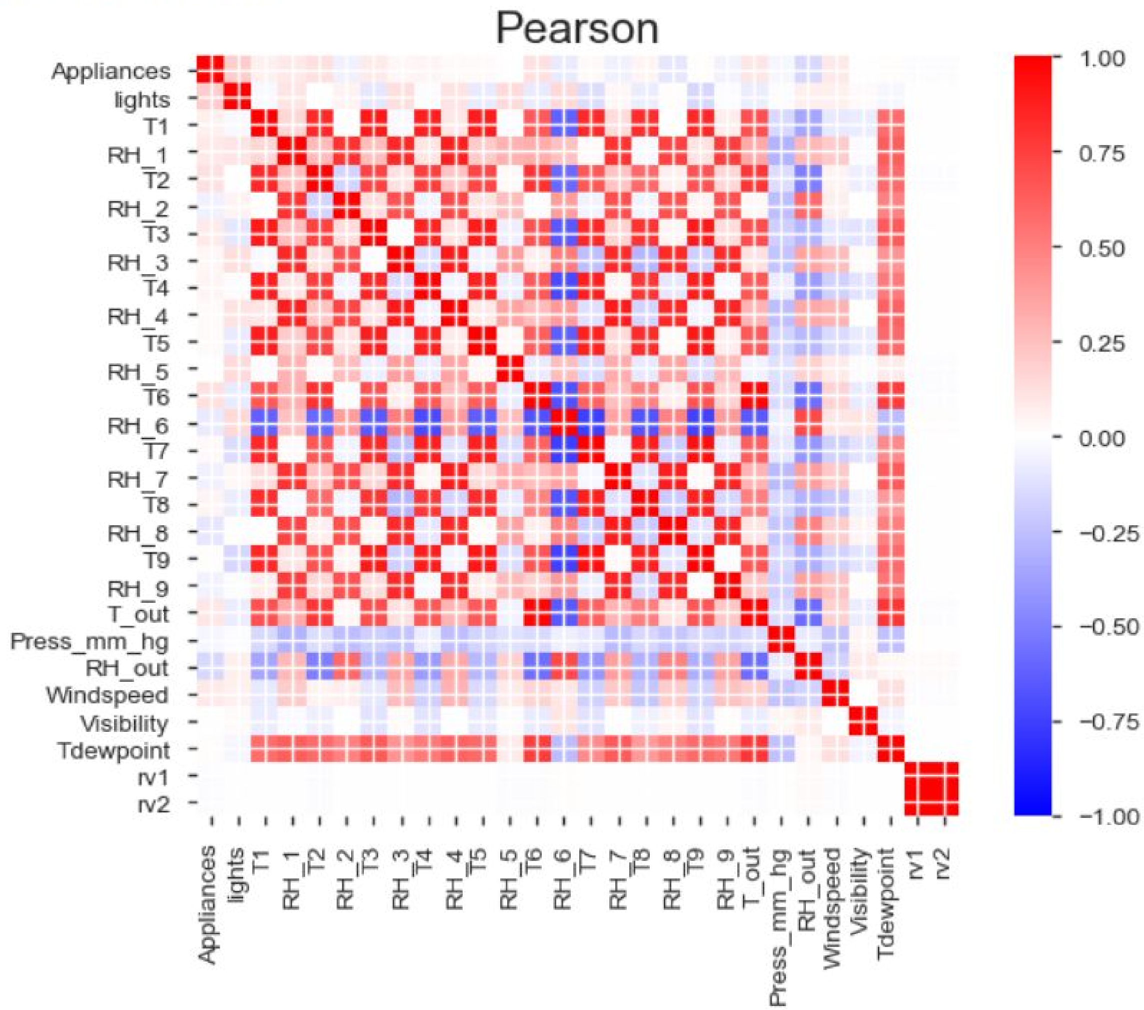
~~rv2~~
Highly correlated

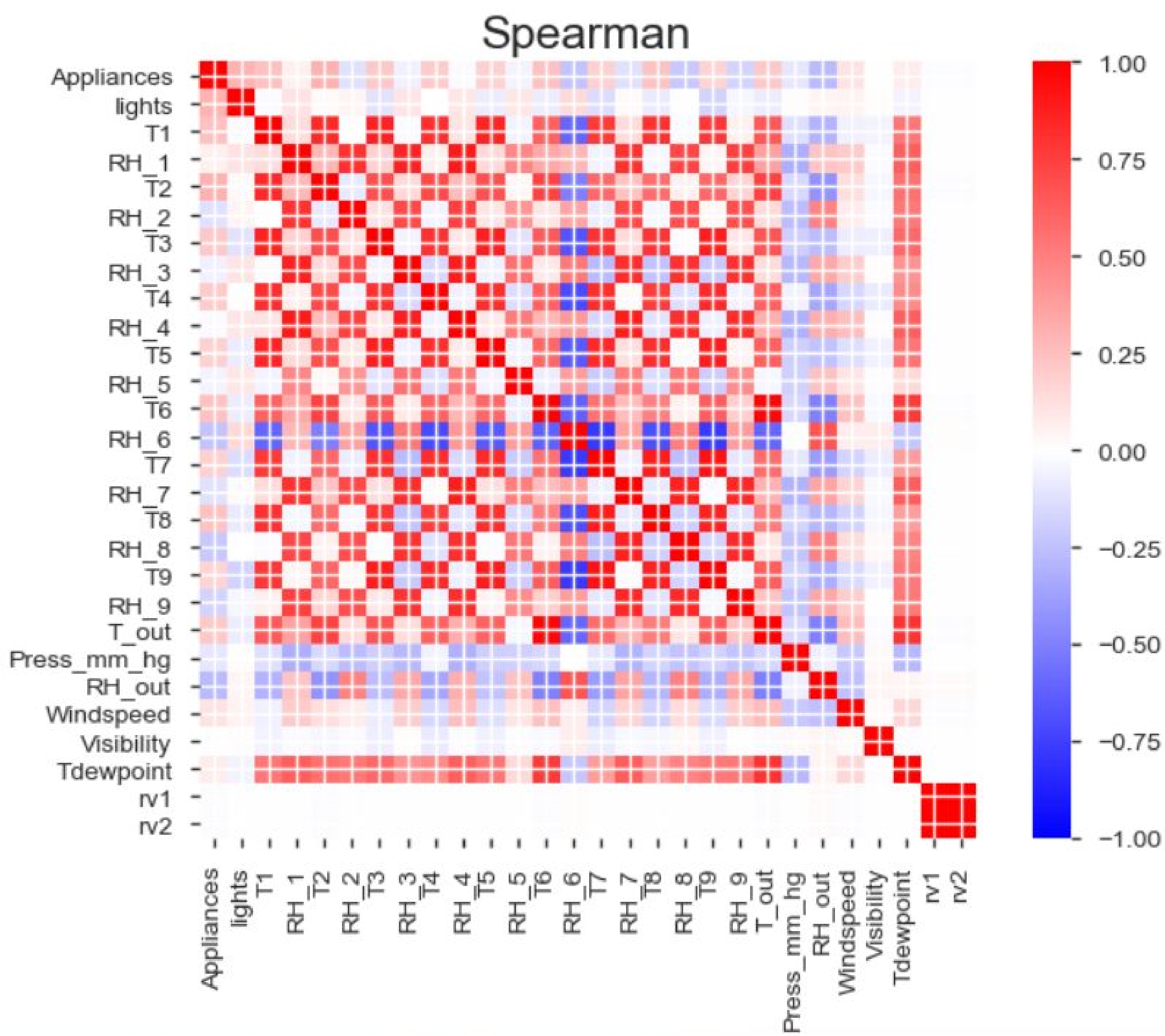
This variable is highly correlated with rv1 *and should be ignored for analysis*

Correlation	1

The correlations are shown below :

Correlations





Spearman
Appliances
lights
T1
RH_1
T2
RH_2
T3
RH_3
T4
RH_4
T5
RH_5
T6
RH_6
T7
RH_7
T8
RH_8
T9
RH_9
T_out
Press_mm_hg
RH_out
Windspeed
Visibility
Tdewpoint
rv1
rv2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Feature Engineering

The following libraries are used :-
1. Pandas
2. Matplotlib
3. Numpy
4. Seaborn
5. ipythonwidgets.interact
6. Datetime

Importing Libraries

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import time
import numpy as np
import seaborn as sns
import datetime
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

For loading dataset used pandas.read_csv

Importing our Data set

```
data= pd.read_csv("Dataset/energydata_complete.csv")
data.head()
```

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...
3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	...
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	...

To find the type of the data we have used data.info()-

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 29 columns):
date           19735 non-null object
Appliances     19735 non-null int64
lights         19735 non-null int64
T1             19735 non-null float64
RH_1           19735 non-null float64
T2             19735 non-null float64
RH_2           19735 non-null float64
T3             19735 non-null float64
RH_3           19735 non-null float64
T4             19735 non-null float64
RH_4           19735 non-null float64
T5             19735 non-null float64
RH_5           19735 non-null float64
T6             19735 non-null float64
RH_6           19735 non-null float64
T7             19735 non-null float64
RH_7           19735 non-null float64
T8             19735 non-null float64
RH_8           19735 non-null float64
T9             19735 non-null float64
RH_9           19735 non-null float64
T_out          19735 non-null float64
Press_mm_hg    19735 non-null float64
RH_out         19735 non-null float64
Windspeed      19735 non-null float64
Visibility     19735 non-null float64
Tdewpoint      19735 non-null float64
rv1            19735 non-null float64
rv2            19735 non-null float64
dtypes: float64(26), int64(2), object(1)
memory usage: 4.4+ MB
```

Feature Engineering -It's used to understand our dataset properly by generating new columns.

New Columns -

Date

Day of the Week - Monday to Sunday

WeekDayType - Weekday or Weekend

partOfTheDay - Day or Night

Active Hours

Minutes to Midnight

Seasons

Feature Engineering From Date-Time Column

Converting Date into Date Time format from String

```
# Convert argument to datetime
data["date_time"] = pd.to_datetime(data["date"],format="%Y-%m-%d %H:%M:%S")
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 30 columns):
date           19735 non-null object
Appliances     19735 non-null int64
lights         19735 non-null int64
T1             19735 non-null float64
RH_1           19735 non-null float64
T2             19735 non-null float64
RH_2           19735 non-null float64
```

```
date_time      19735 non-null datetime64[ns]
dtypes: datetime64[ns](1), float64(26), int64(2), object(1)
```

Determining The Day of the Week

```
def dayoftheweek(day):
    if(day==0):
        return("Monday")
    if(day==1):
        return("Tuesday")
    if(day==2):
        return("Wednesday")
    if(day==3):
        return("Thurday")
    if(day==4):
        return("Friday")
    if(day==5):
        return("Saturday")
    if(day==6):
        return("Sunday")
```

```
data["dayoftheweek"] = data['date_time']
data["dayoftheweek"] = data['dayoftheweek'].apply(lambda x: dayoftheweek(x.dayofweek))
data.groupby('dayoftheweek').count()["date_time"]
```

```
dayoftheweek
Friday       2845
Monday       2778
Saturday     2736
Sunday       2736
Thurday      2880
Tuesday      2880
Wednesday    2880
Name: date_time, dtype: int64
```

From above function we determined the total number of the day of the week
So, the least count is for Friday and Maximum count is for Tuesday, Wednesday and Thursday i.e. 2880

Adding Weekday or Weekend Column

```
def weekdaytype(day):
    if(day=="Saturday" or day == "Sunday"):
        return "weekend"
    else:
        return "Weekday"
```

```
data["WeekDayType"] = data["dayoftheweek"]
data["WeekDayType"] = data['WeekDayType'].apply(lambda x: weekdaytype(x))
data.groupby('WeekDayType').count()["date_time"]
```

```
WeekDayType
Weekday    14263
weekend     5472
Name: date_time, dtype: int64
```

Total number of weekdays are 14263 and weekends are 5472

Adding Day or Night

Considering Sun Rise and Sun Set

```
def partOfTheDay(time):
    day1 = pd.to_datetime('18:00:00',format="%H:%M:%S")
    day2 = pd.to_datetime('6:00:00',format="%H:%M:%S")
    if(time<day1.time() and time >= day2.time()):
        return "Day"
    else:
        return "Night"
```

```
data['timeofDay'] = data["date_time"].map(lambda x: partOfTheDay(x.time()))
```

```
data.groupby('timeofDay').count()["date_time"]
```

```
timeofDay
Day      9870
Night    9865
Name: date_time, dtype: int64
```

Calculating day and night time

Adding Active Hours

The time when human activity is present

```
def awakeTest(time):
    day1 = pd.to_datetime('8:00:00',format="%H:%M:%S")
    day2 = pd.to_datetime('22:00:00',format="%H:%M:%S")
    if(time>=day1.time() and time < day2.time()):
        return "awake"
    else:
        return "sleep"
```

```
data['activeStatus'] = data["date_time"].map(lambda x: awakeTest(x.time()))
```

```
data.groupby('activeStatus').count()["date_time"]
```

```
activeStatus
awake    11515
sleep     8220
Name: date_time, dtype: int64
```

Created function awakeTest() to find the hours spent in day and night time

Adding Minutes to Midnight Column

from 12am to 12pm (minutes)

```
data['NSM'] = pd.to_datetime(data['date_time'])
data['NSM']  = (data['NSM'].dt.hour*60 + data['NSM'].dt.minute)*60 + data['NSM'].dt.second
data['NSM'].head()
```

```
0    61200
1    61800
2    62400
3    63000
4    63600
Name: NSM, dtype: int64
```

Calculated Number of hours spent from 12am to 12pm

Adding Week of the year

```
data["weekOfTheYear"] = data['date_time'].apply(lambda x: x.isocalendar()[1])
data
```

	T3	RH_3	T4	...	Tdewpoint	rv1	rv2	date_time	dayoftheweek	WeekDayType	timeofDay	activeStatus	NSM	weekOfThe
	19.790000	44.730000	19.000000	...	5.300000	13.275433	13.275433	2016-01-11 17:00:00	Monday	Weekday	Day	awake	61200	
	19.790000	44.790000	19.000000	...	5.200000	18.606195	18.606195	2016-01-11 17:10:00	Monday	Weekday	Day	awake	61800	
	19.790000	44.933333	18.926667	...	5.100000	28.642668	28.642668	2016-01-11 17:20:00	Monday	Weekday	Day	awake	62400	

Adding Season

```
from datetime import date, datetime
Y = 2000 # dummy leap year to allow input X-02-29 (leap day)
seasons = [('winter', (date(Y,  1,  1),  date(Y,  3, 20))),
           ('spring', (date(Y,  3, 21),  date(Y,  6, 20))),
           ('summer', (date(Y,  6, 21),  date(Y,  9, 22))),
           ('autumn', (date(Y,  9, 23),  date(Y, 12, 20))),
           ('winter', (date(Y, 12, 21),  date(Y, 12, 31)))]
def get_season(now):
    if isinstance(now, datetime):
        now = now.date()
    now = now.replace(year=Y)
    return next(season for season, (start, end) in seasons
                if start <= now <= end)

print(get_season(data['date_time'][19734]))
```

```
spring
```

```
In [44]: data['season'] = data['date_time'].apply(lambda x: get_season(x))
         data
```

RH_2	T3	RH_3	T4	...	rv1	rv2	date_time	dayoftheweek	WeekDayType	timeofDay	activeStatus	NSM	weekOfTheYear	season
44.790000	19.790000	44.730000	19.000000	...	13.275433	13.275433	2016-01-11 17:00:00	Monday	Weekday	Day	awake	61200	2	winter
44.722500	19.790000	44.790000	19.000000	...	18.606195	18.606195	2016-01-11 17:10:00	Monday	Weekday	Day	awake	61800	2	winter
44.626667	19.790000	44.933333	18.926667	...	28.642668	28.642668	2016-01-11 17:20:00	Monday	Weekday	Day	awake	62400	2	winter
44.590000	19.790000	45.000000	18.890000	...	45.410389	45.410389	2016-01-11 17:30:00	Monday	Weekday	Day	awake	63000	2	winter

```
data['season'].unique()
```

```
array(['winter', 'spring'], dtype=object)
```

In our dataset we have 2 seasons, Winter and spring

Prediction Algorithm

```
#Importing Libraries
import matplotlib.pyplot as plt
import pandas as pd
import time
import numpy as np
import seaborn as sns
import datetime
import math


from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets

%matplotlib inline
```

Load Data

```
data= pd.read_csv("./Dataset/new_data_feature.csv")
```

```
data.head()
```

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...
3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	...

Libraries, Functions and Variables to Evaluate model

```
#Importing Libraries
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error
from sklearn.cross_validation import cross_val_score


rmse_dict = {}
def rmse(correct,estimated):
    rmse_val = np.sqrt(mean_squared_error(correct,estimated))
    return rmse_val

# Generating the Table Frame for metrics
evluation_table = pd.DataFrame({  'Model_desc':[],
                        'Model_param':[],
                        'r2_train': [],
                        'r2_test': [],
                        'rms_train':[],
                        'rms_test': [],
                        'mae_train': [],
                        'mae_test': [],
                        'mape_train':[],
                        'mape_test':[],
                        'cross_val_score' : []})
```

```python
# Evaluating the model
def evaluate_model(model, model_desc,model_param, X_train, y_train, X_test, y_test):
    global evluation_table

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)


    try:
        r2_train = r2_score(y_train, y_train_pred)
        r2_test = r2_score(y_test, y_test_pred)
    except:
        r2_train = "not calculated"
        r2_test = "not calculated"
    try:
        rms_train = rmse(y_train, y_train_pred)
        rms_test = rmse(y_test, y_test_pred)
    except:
        rms_train = "not calculated"
        rms_test = "not calculated"
    try:
        mae_train = mean_absolute_error(y_train, y_train_pred)
        mae_test = mean_absolute_error(y_test, y_test_pred)
    except:
        mae_train = "not calculated"
        mae_test = "not calculated"
    try:
        mape_train = np.mean(np.abs((y_train - y_train_pred) / y_train)) * 100
        mape_test = np.mean(np.abs((y_test - y_test_pred) / y_test)) * 100
    except:
        mape_train = "not calculated"
        mape_test = "not calculated"
```

```python
    model_param = pd.DataFrame({'Model_desc':[model_desc],
                               'Model_param':[model_param],
                               'r2_train': [r2_train],
                               'r2_test': [r2_test],
                               'rms_train':[rms_train],
                               'rms_test': [rms_test],
                               'mae_train': [mae_train],
                               'mae_test': [mae_test],
                               'mape_train':[mape_train],
                               'mape_test':[mape_test],
                               'cross_val_score' : "Not Calculated"})

    evluation_table = evluation_table.append([model_param])

    return evluation_table
```

	Model_desc	Model_param	cross_val_score	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	LinearRegression	LinearRegression (copy_X=True, fit_intercept=Tr...	Not Calculated	51.942311	51.777792	61.6528	58.7834	0.198272	0.212148	89.2427	91.827
0	DecisionTreeClassifier	DecisionTreeClassifier (class_weight=None, crit...	Not Calculated	37.081475	0.000000	34.2137	0	0.173300	1.000000	90.6219	0
0	RandomForestClassifier	(DecisionTreeClassifier (class_weight=None, cri...	Not Calculated	30.766113	0.000000	27.1424	0	0.417652	1.000000	77.9503	0
0	XGBClassifier	XGBClassifier (base_score=0.5, colsample_byleve...	Not Calculated	40.287799	31.518141	27.4462	23.4839	0.065761	0.357640	100.245	81.8482
0	GradientBoostingClassifier	([DecisionTreeRegressor (criterion='friedman_ms...	Not Calculated	45.717471	31.090467	37.7367	27.6261	-0.227463	0.430312	110.057	78.1591
0	Kmeans	KMeans(algorithm='auto', copy_x=True, init='k-...	Not Calculated	98.484799	96.765219	99.0614	99.0517	-0.895385	-0.902851	143.443	140.642
0	Keras-LSTM	<keras.models.Sequential object at 0x00000149A...	Not Calculated	43.573487	41.022321	not calculated	not calculated	0.035160	0.057201	not calculated	not calculated

For performing prediction we

Feature Selection

For selection

Model Validation and Selection

In this notebook we will be going through varies methods to select a model based upon the following features

- cross validation techniques
- Bias -variance tradeoff
- regularization (L1, L2, Elastic net)
- grid search options.

Definition of Validation

cross validation techniques

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.

In ML, you can use the k-fold cross-validation method to perform cross-validation. In k-fold cross-validation, you split the input data into k subsets of data (also known as folds). You train an ML model on all but one (k-1) of the subsets, and then evaluate the model on the subset that was not used for training. This process is repeated k times, with a different subset reserved for evaluation (and excluded from training) each time.

Bias -variance tradeoff

Overview of Bias and Variance In supervised machine learning an algorithm learns a model from training data.

The goal of any supervised machine learning algorithm is to best estimate the mapping function (f) for the output variable (Y) given the input data (X). The mapping function is often called the target function because it is the function that a given supervised machine learning algorithm aims to approximate.

The prediction error for any machine learning algorithm can be broken down into three parts:

1. Bias Error
2. Variance Error
3. Irreducible Error

The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

Bias Error

Bias are the simplifying assumptions made by a model to make the target function easier to learn.

Generally, parametric algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

- **Low Bias**: Suggests less assumptions about the form of the target function.
- **High-Bias**: Suggests more assumptions about the form of the target function.

Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance Error

Variance is the amount that the estimate of the target function will change if different training data was used.

The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.

Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function.

- **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
- **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

Generally, nonparametric machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Bias-Variance Trade-Off

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

You can see a general trend in the examples above:

- Parametric or linear machine learning algorithms often have a high bias but a low variance.
- Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.

The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

- The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
- The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

- Increasing the bias will decrease the variance.
- Increasing the variance will decrease the bias.

There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem

In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools to understand the behavior of machine learning algorithms in the pursuit of predictive performance.

Regularization (L1, L2, Elastic net)

Regularization is used to fix over fitting of models

$$Error_{L1} = Error + \sum_{i=0}^{N} |\beta_i|$$

$$where\ the\ \beta_i\ are\ the\ parameters$$

Regularization adds a penalty on the different parameters of the model to reduce the freedom of the model. Hence, the model will be **less likely to fit the noise** of the training data and will improve the generalization abilities of the model. We will study and compare:

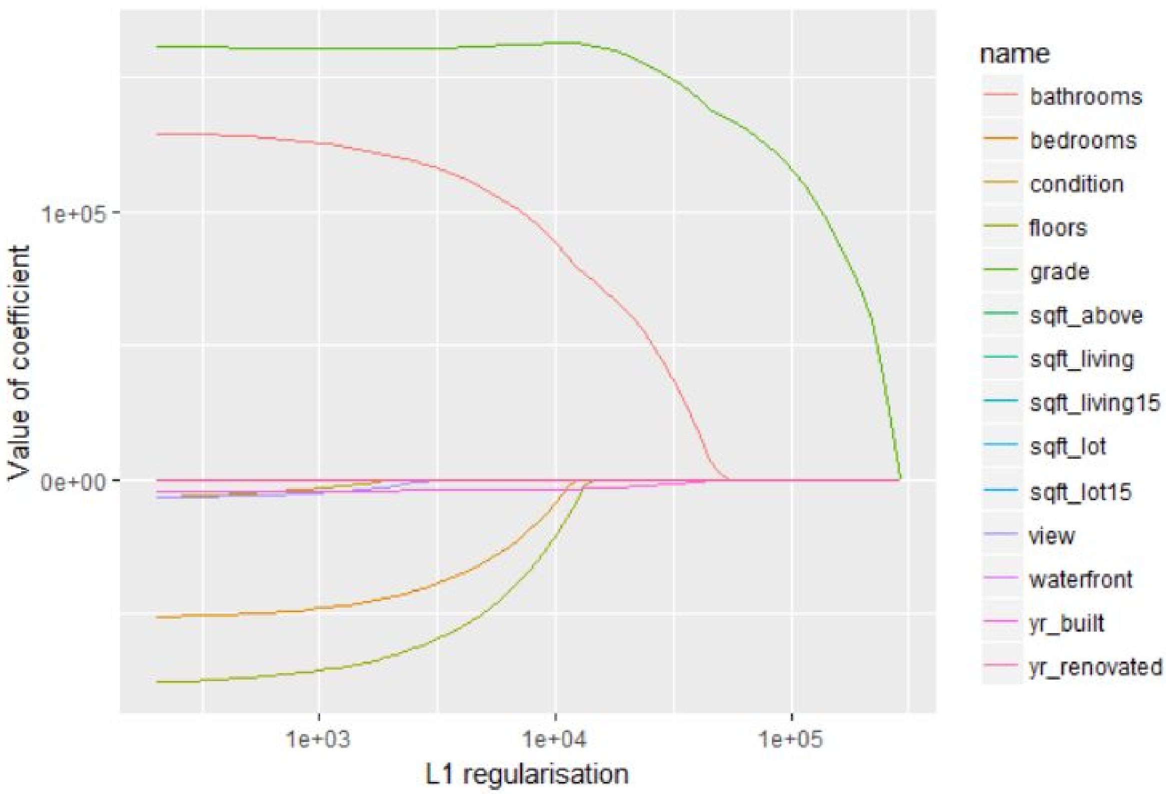
- The L1 regularization (also called Lasso)
- The L2 regularization (also called Ridge)
- The L1/L2 regularization (also called Elastic net)

You can find the R code for regularization at the end of the post.

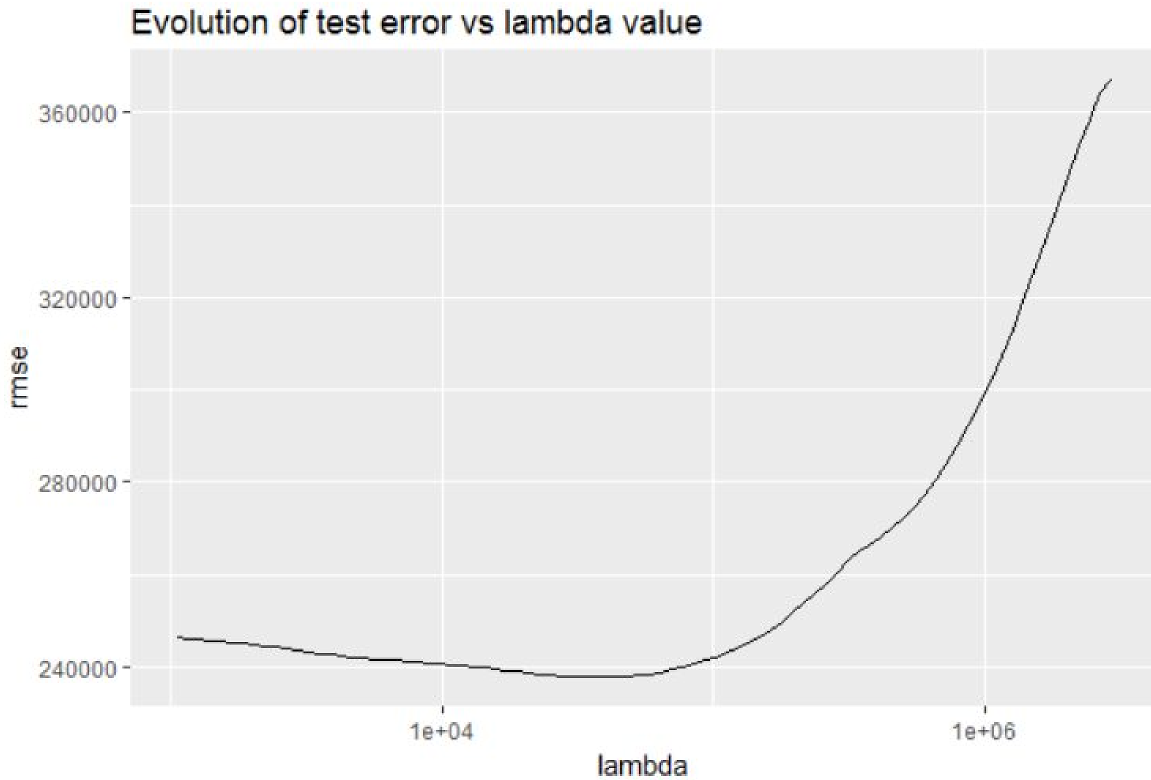
L1 Regularization (Lasso penalisation)

The L1 regularization adds a penalty equal to the sum of the absolute value of the coefficients. The L1 regularization will **shrink some parameters to zero**. Hence some variables will not play any role in the model, L1 regression can be seen as a way to select features in a model.

As lambda grows bigger, more coefficient will be cut. Below is the evolution of the value of the different coefficients while lambda is growing.



As expected, coefficients are cut one by one until no variables remain. Let's see how the test error is evolving:



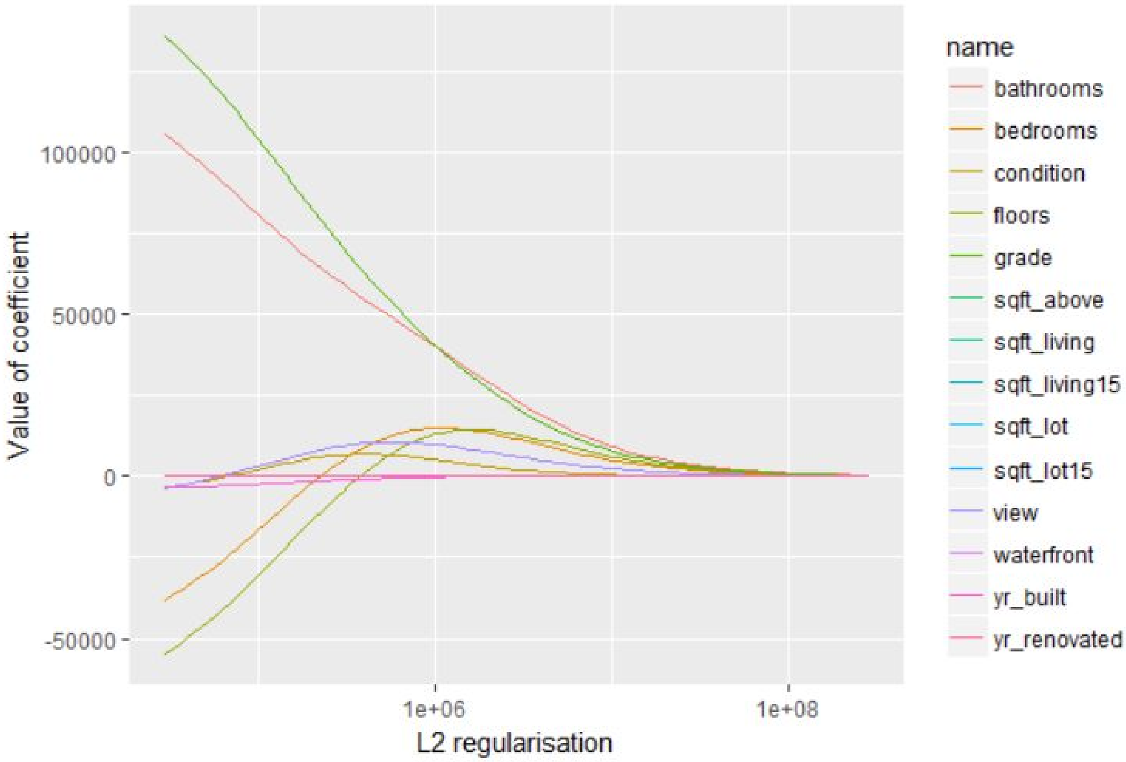
At the beginning, cutting coefficient reduces the overfitting and the generalization abilities of the model. Hence, the test error is decreasing. However, as we are cutting more and more coefficient, the test error start increasing. The model is not able to learn complex pattern with so few variables.

L2 Regularization (Ridge penalisation)

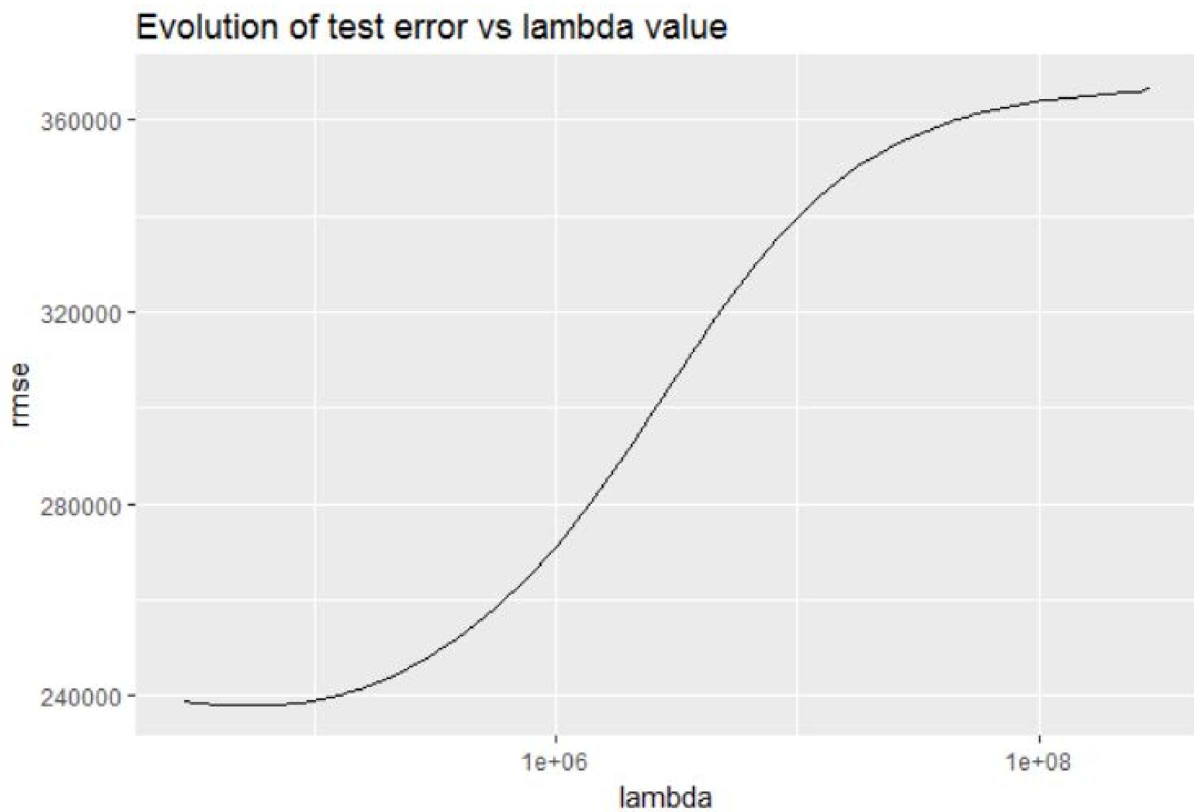
The L2 regularization adds a penalty equal to the sum of the squared value of the coefficients.

$$ERROR_{L2} = ERROR + \sum_{0}^{N} \lambda.\beta_i^2$$

The L2 regularization will force **the parameters to be relatively small**, the bigger the penalization, the smaller (and the more robust) the coefficients are.



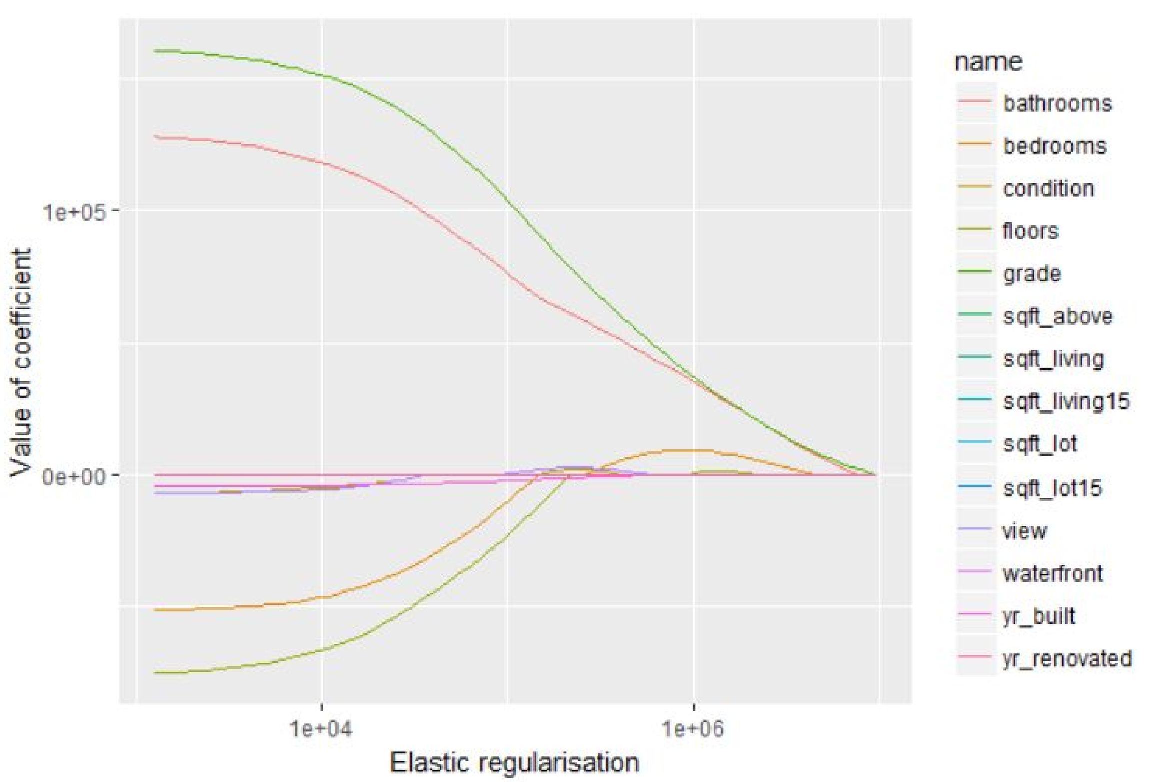
When we compare this plot to the L1 regularization plot, we notice that the coefficients decrease progressively and are not cut to zero. They slowly decrease to zero. That is the behavior we expected. Let's see how the test error evolves:



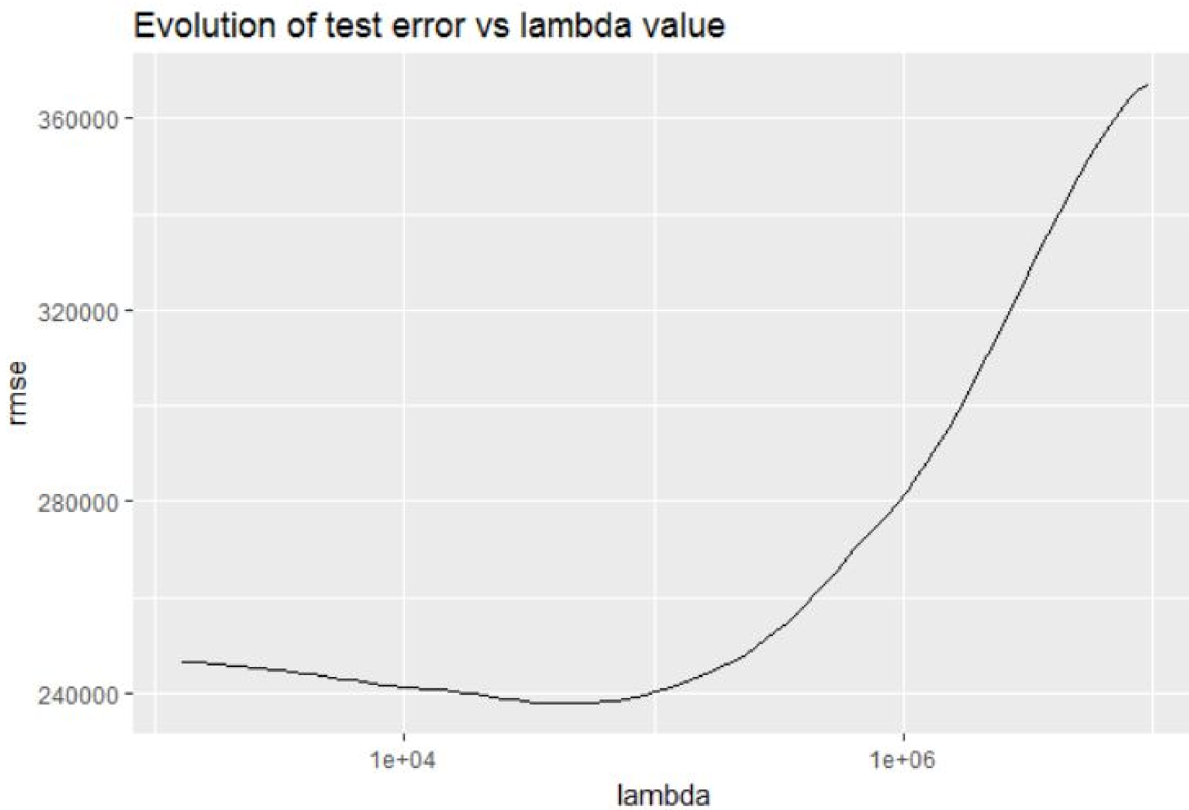
Elastic-net

Elastic-net is a mix of **both L1 and L2 regularizations**. A penalty is applied to the sum of the absolute values and to the sum of the squared values:

Lambda is a shared penalization parameter while alpha sets the ratio between L1 and L2 regularization in the Elastic Net Regularization. Hence, we expect a hybrid behavior between L1 and L2 regularization.



And that’s happening: Though coefficients are cut, the cut is less abrupt than the cut with lasso penalization alone.



A geometric perspective on regularization

The Lasso, Ridge and Elastic-net regression can also be viewed as a constraint added to the optimization process.

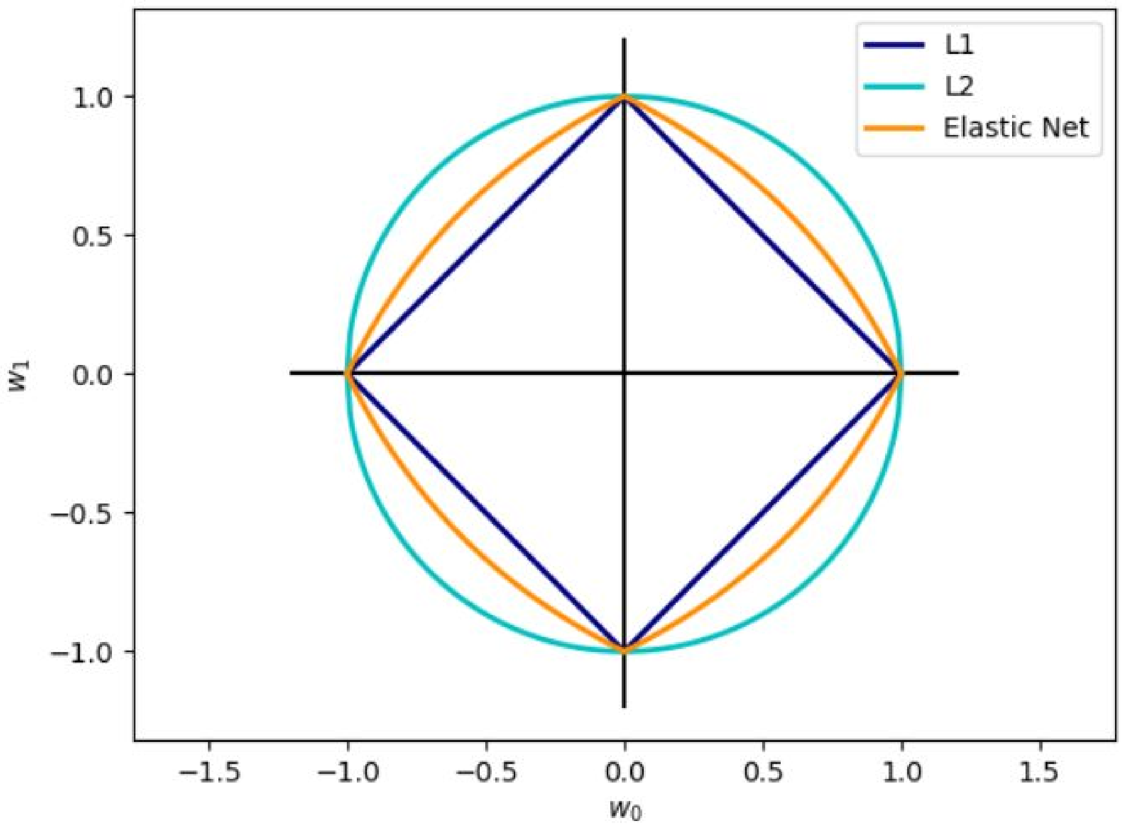
The Lasso error minimization can be rewritten:

$$arg(min_{\beta_i}(MSE)) \text{ given that } \sum |\beta_i| < t$$

And the ridge error minimization can be rewritten:

$$arg(min_{\beta_i}(MSE)) \text{ given that } \sum \beta_i^2 < t$$

When written in this way, it's clear that Lasso restricts the coefficients to a square shape (or an L1 sphere) which diagonals are equal to 2t. The ridge error restricts the coefficients to a circle (or an L2 sphere) of radius t.



Grid Search Options.

Grid search means you have a set of models (which differ from each other in their parameter values, which lie on a grid). What you do is you then train each of the models and evaluate it using cross-validation.

A search consists of:

- an estimator (regressor or classifier such as `sklearn.svm.SVC()`);

- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a score function.

Final Pipeline

It consist of entire process along with the dockerfile to create the image of each part.

Final Pipeline

```
import pandas as pd
import time
import numpy as np
import datetime
import logging
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
```

Logging Data

The dataset used is - energydata_complete

```
logfilename = 'log_pipeLine.txt'
logging.basicConfig(filename=logfilename, level=logging.DEBUG,
                        format='%(asctime)s - %(levelname)s - %(message)s')
logging.debug('Program Started')
```

```
logging.debug('Loading Data into Dataframe')
try :
    data= pd.read_csv("Dataset/energydata_complete.csv")
    logging.debug('Data Size'+str(data.shape) )

except :
    logging.ERROR('Data logging failed')
```

```
logging.debug("Tranforming date time")
data["date_time"] = pd.to_datetime(data["date"],format="%Y-%m-%d %H:%M:%S")
```

```
data["weekOfTheYear"] = data['date_time'].apply(lambda x: x.isocalendar()[1])
```

```
from datetime import date, datetime
Y = 2000 # dummy leap year to allow input X-02-29 (leap day)
seasons = [('winter', (date(Y,  1,  1),  date(Y,  3, 20))),
           ('spring', (date(Y,  3, 21),  date(Y,  6, 20))),
           ('summer', (date(Y,  6, 21),  date(Y,  9, 22))),
           ('autumn', (date(Y,  9, 23),  date(Y, 12, 20))),
           ('winter', (date(Y, 12, 21),  date(Y, 12, 31)))]
def get_season(now):
    if isinstance(now, datetime):
        now = now.date()
    now = now.replace(year=Y)
    return next(season for season, (start, end) in seasons
                if start <= now <= end)

print(get_season(data['date_time'][19734]))
```

```
spring
```

```
logging.debug('Creating column Season')
data['season'] = data['date_time'].apply(lambda x: get_season(x))
```

```
logging.debug('Creating Dummy Column')
data = pd.get_dummies(data, columns=["timeofDay","activeStatus",'dayoftheweek','WeekDayType','season'])
```

```
min_max_scaler = MinMaxScaler()
X = min_max_scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.75, test_size = 0.25)
```

```
classifier = RandomForestRegressor(n_estimators = 300, max_features=4)
classifier.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=4, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=300, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

```
print ("R-squared for Train: %.2f" %classifier.score(X_train, y_train))
print ("R-squared for Test: %.2f" %classifier.score(X_test, y_test))
```

```
R-squared for Train: 0.94
R-squared for Test: 0.60
```

The best model is RandomForestRegressor with R-squared 0.94 for Training dataset and 0.60 for Testing Dataset

Conclusion

We can conclude that there are many libraries that have really good EDA analysis tools, auto feature engineering , auto feature selection and auto ML algorithm generation kits.

With the help of the above tools we have finally selected :

RandomForestRegressor with R-squared 0.94 for Training dataset and 0.60 for Testing Dataset

We Finally dockerized the pipeline solution and put them in part 7 of the assignment.