Commercial Building Energy Consumption modeling

By : Vivek Kumar

Vyshak Srishylappa

Sankalp Jadon

TABLE OF CONTENTS

Contents **Page**

1. Part 1 - Data Ingestion 3

2. Part 2 - Data wrangling and cleaning *7*

3. Prediction 11

4. Classification 17

5. Clustering 22

File execution pattern :

- part1.py
- cleansingandpart2.R
- part2_prediction.R
- part2_classification.R
- part3.R

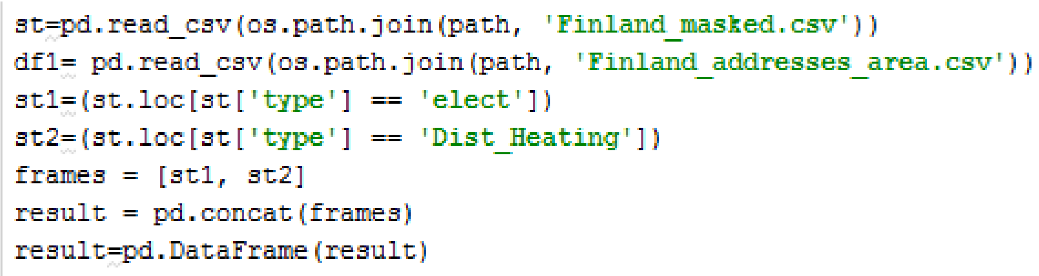
Part 1 - Data Ingestion

- Files required:
  - Part1.py
  - Finland_addresses_area.csv
  - Finland_masked.csv
- Place the files in the directory of Part1.py.
- Access to the files is done using the getwd in os package. Hence no hardcoding of filepaths.

  path=os.getcwd()

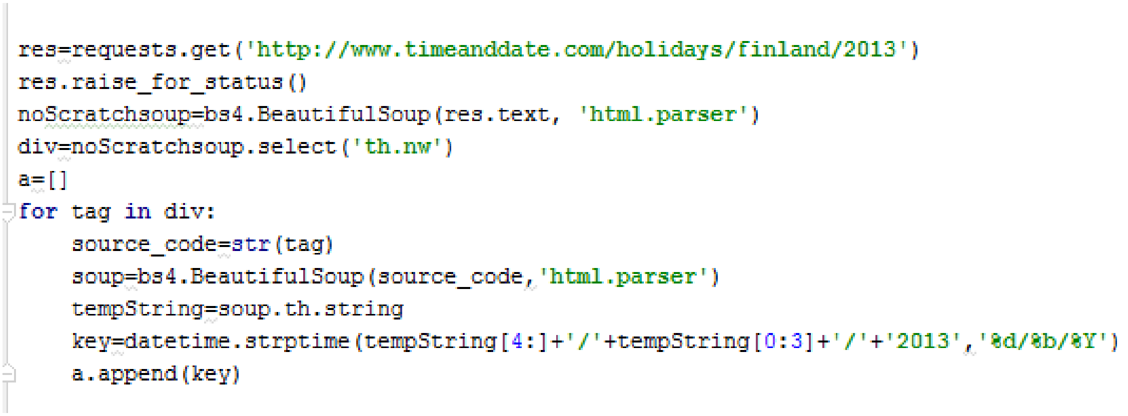
  st=pd.read_csv(os.path.join(path, 'Finland_masked.csv'))   for example.

Filter ‘elect’ and ‘Dist_Heating’ and place it in a dataframe.

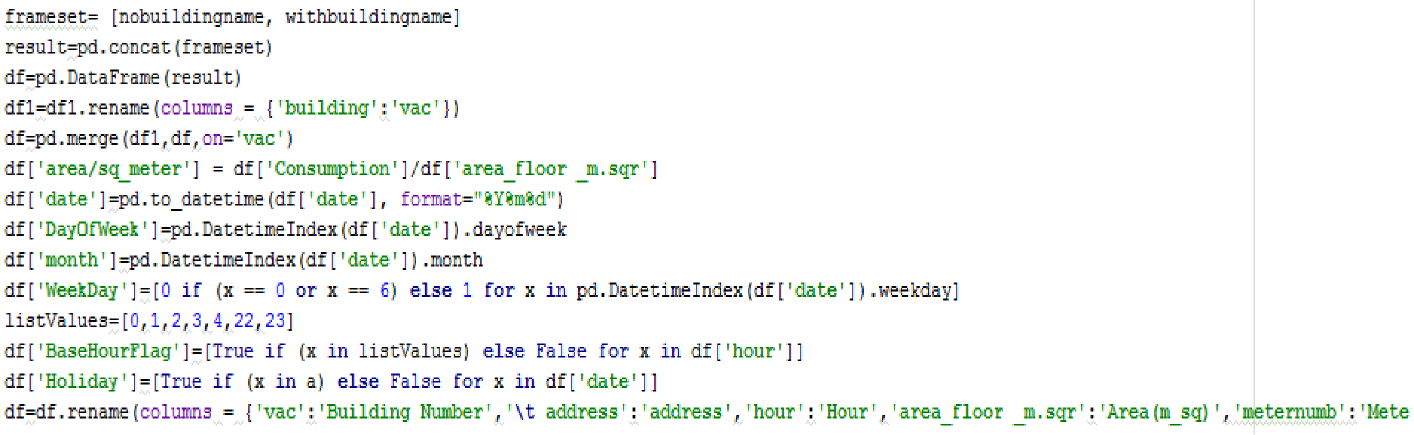


```
st=pd.read_csv(os.path.join(path, 'Finland_masked.csv'))
df1= pd.read_csv(os.path.join(path, 'Finland_addresses_area.csv'))
st1=(st.loc[st['type'] == 'elect'])
st2=(st.loc[st['type'] == 'Dist_Heating'])
frames = [st1, st2]
result = pd.concat(frames)
result=pd.DataFrame(result)
```

Derive date, DayofWeek, month and other fields. Holiday is scraped from http://www.timeanddate.com/holidays/finland/2013



```
res=requests.get('http://www.timeanddate.com/holidays/finland/2013')
res.raise_for_status()
noScratchsoup=bs4.BeautifulSoup(res.text, 'html.parser')
div=noScratchsoup.select('th.nw')
a=[]
for tag in div:
    source_code=str(tag)
    soup=bs4.BeautifulSoup(source_code,'html.parser')
    tempString=soup.th.string
    key=datetime.strptime(tempString[4:]+'/'+tempString[0:3]+'/'+'2013','%d/%b/%Y')
    a.append(key)
```

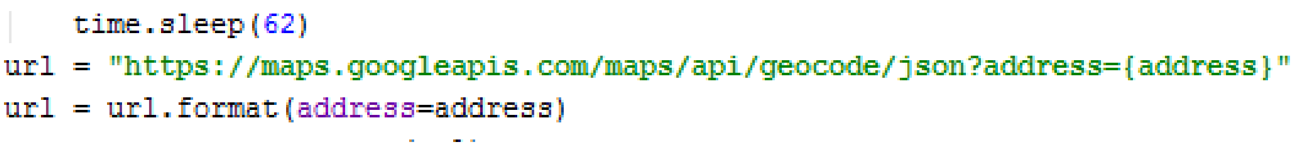


```
frameset= [nobuildingname, withbuildingname]
result=pd.concat(frameset)
df=pd.DataFrame(result)
df1=df1.rename(columns = {'building':'vac'})
df=pd.merge(df1,df,on='vac')
df['area/sq_meter'] = df['Consumption']/df['area_floor _m.sqr']
df['date']=pd.to_datetime(df['date'], format="%Y%m%d")
df['DayOfWeek']=pd.DatetimeIndex(df['date']).dayofweek
df['month']=pd.DatetimeIndex(df['date']).month
df['WeekDay']=[0 if (x == 0 or x == 6) else 1 for x in pd.DatetimeIndex(df['date']).weekday]
listValues=[0,1,2,3,4,22,23]
df['BaseHourFlag']=[True if (x in listValues) else False for x in df['hour']]
df['Holiday']=[True if (x in a) else False for x in df['date']]
df=df.rename(columns = {'vac':'Building Number','\t address':'address','hour':'Hour','area_floor _m.sqr':'Area(m_sq)','meternumb':'Mete
```

We are gathering the weather data by going through three steps –

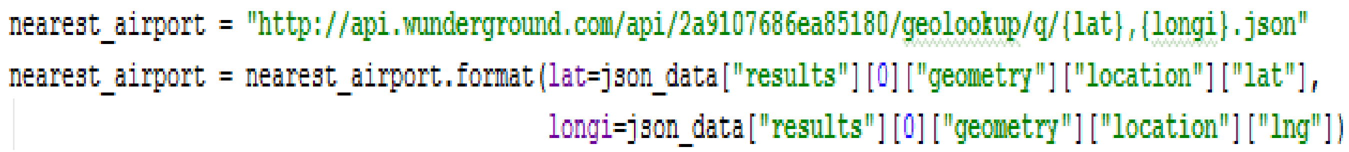
1. **Getting geocode – Latitude, Longitude from googleapis API**
2. **Getting the nearby airport code using wunderground API**
3. **Processing a loop for whole year on wunderground airport csv files**

We read the 'Finland_addresses_area.csv' file and keep data in an address list. Maps.googleapis API takes an input of address, where each address will be send by address list.



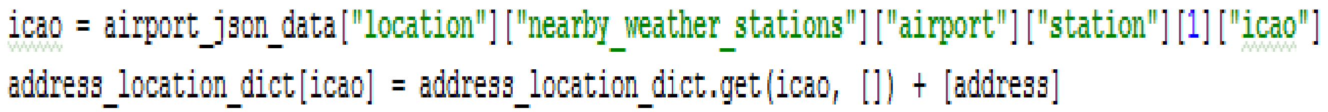
```
    time.sleep(62)
url = "https://maps.googleapis.com/maps/api/geocode/json?address={address}"
url = url.format(address=address)
```

Next, the json response will be feed as input to wunderground API.



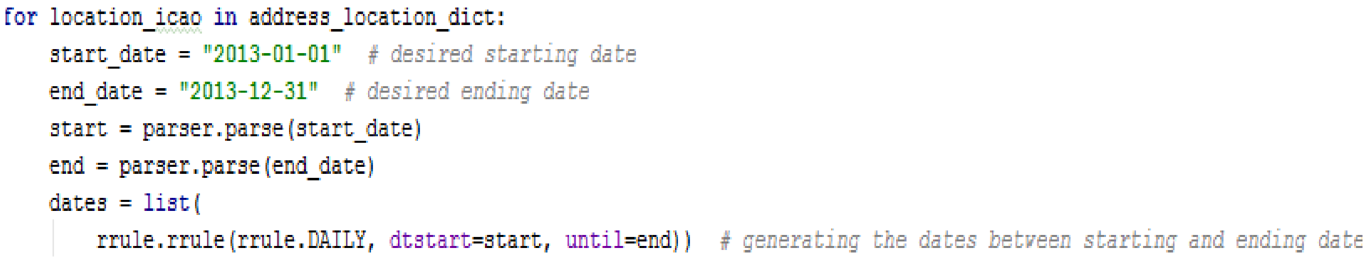
```
nearest_airport = "http://api.wunderground.com/api/2a9107686ea85180/geolookup/q/{lat},{longi}.json"
nearest_airport = nearest_airport.format(lat=json_data["results"][0]["geometry"]["location"]["lat"],
                                         longi=json_data["results"][0]["geometry"]["location"]["lng"])
```

This returns a json, where we gather the airport 'ICAO' code. Since, there are several addresses which have same airport code, we have designed a dictionary which keep tracks of all the data. Airport Code serves as the key and the list of addresses are served as values.

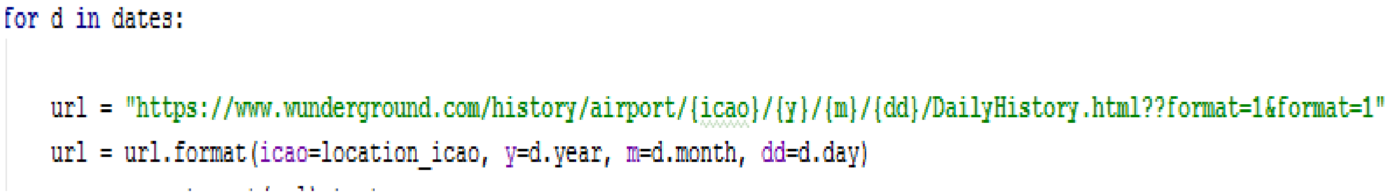


```
icao = airport_json_data["location"]["nearby_weather_stations"]["airport"]["station"][1]["icao"]
address_location_dict[icao] = address_location_dict.get(icao, []) + [address]
```

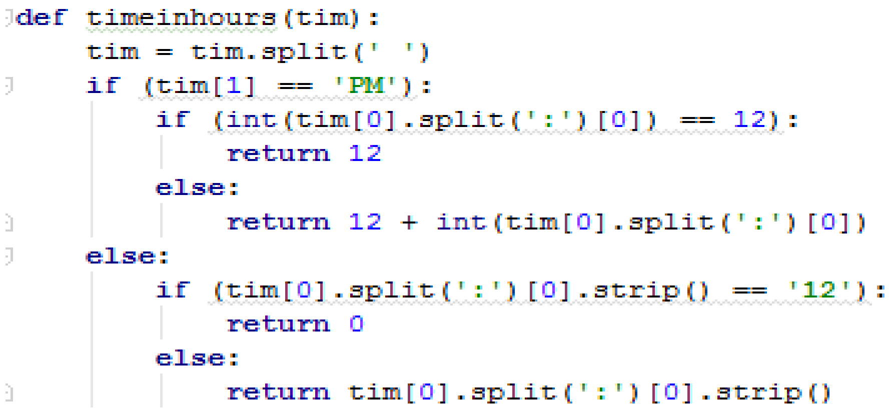
Using Pandas DataFrame, we created the column headers. Now, we are keeping a loop on year 2013, which will fetch the data from comma separated URL of wunderground.



```
for location_icao in address_location_dict:
    start_date = "2013-01-01"  # desired starting date
    end_date = "2013-12-31"  # desired ending date
    start = parser.parse(start_date)
    end = parser.parse(end_date)
    dates = list(
        rrule.rrule(rrule.DAILY, dtstart=start, until=end))  # generating the dates between starting and ending date
```



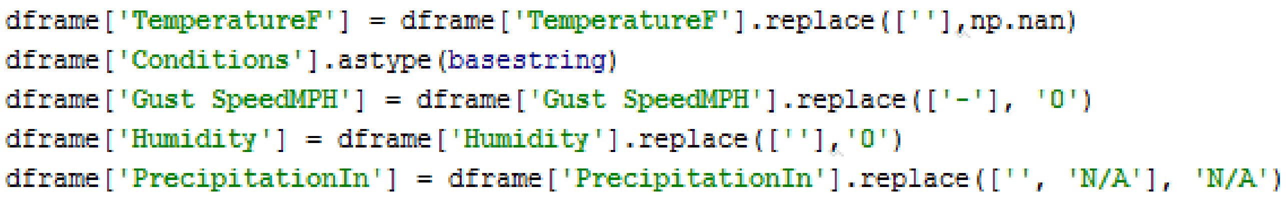
```
for d in dates:

    url = "https://www.wunderground.com/history/airport/{icao}/{y}/{m}/{dd}/DailyHistory.html??format=1&format=1"
    url = url.format(icao=location_icao, y=d.year, m=d.month, dd=d.day)
```



```
def timeinhours(tim):
    tim = tim.split(' ')
    if (tim[1] == 'PM'):
        if (int(tim[0].split(':')[0]) == 12):
            return 12
        else:
            return 12 + int(tim[0].split(':')[0])
    else:
        if (tim[0].split(':')[0].strip() == '12'):
            return 0
        else:
            return tim[0].split(':')[0].strip()
```

We need to merge the created dataframe with Finland_masked.csv file. Since one of the criteria for group by will be date, above function will create the exact same date format as we already have in Finland_masked.csv file. This will make the merger easy.

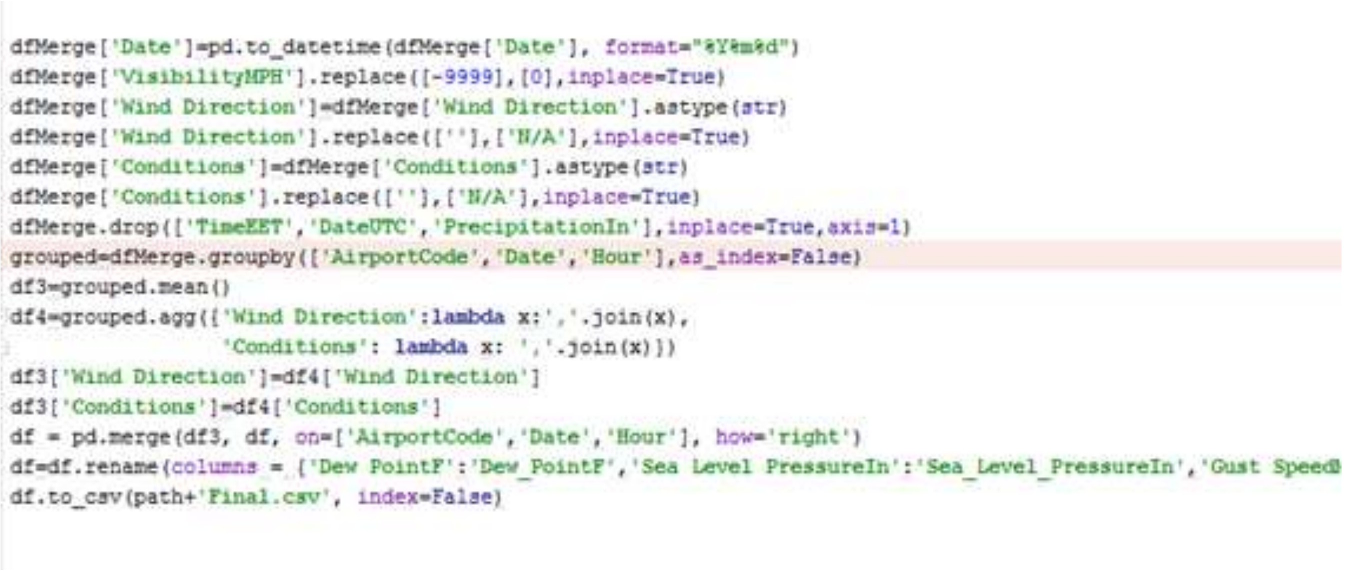
We are using BeautifulSoup to scrap the wunderground csv file and append the total data into a continuous dataframe. After the data is scraped, we are performing basic data cleanup on the columns.



```
dframe['TemperatureF'] = dframe['TemperatureF'].replace([''],np.nan)
dframe['Conditions'].astype(basestring)
dframe['Gust SpeedMPH'] = dframe['Gust SpeedMPH'].replace(['-'], '0')
dframe['Humidity'] = dframe['Humidity'].replace([''],'0')
dframe['PrecipitationIn'] = dframe['PrecipitationIn'].replace(['', 'N/A'], 'N/A')
```

We found that few of rows are blank, or have ‘-‘ or have N/A values. We have replaced the missing values with the appropriate desired values on the columns. We have performed data Cleanup and finding outliers further in the process.

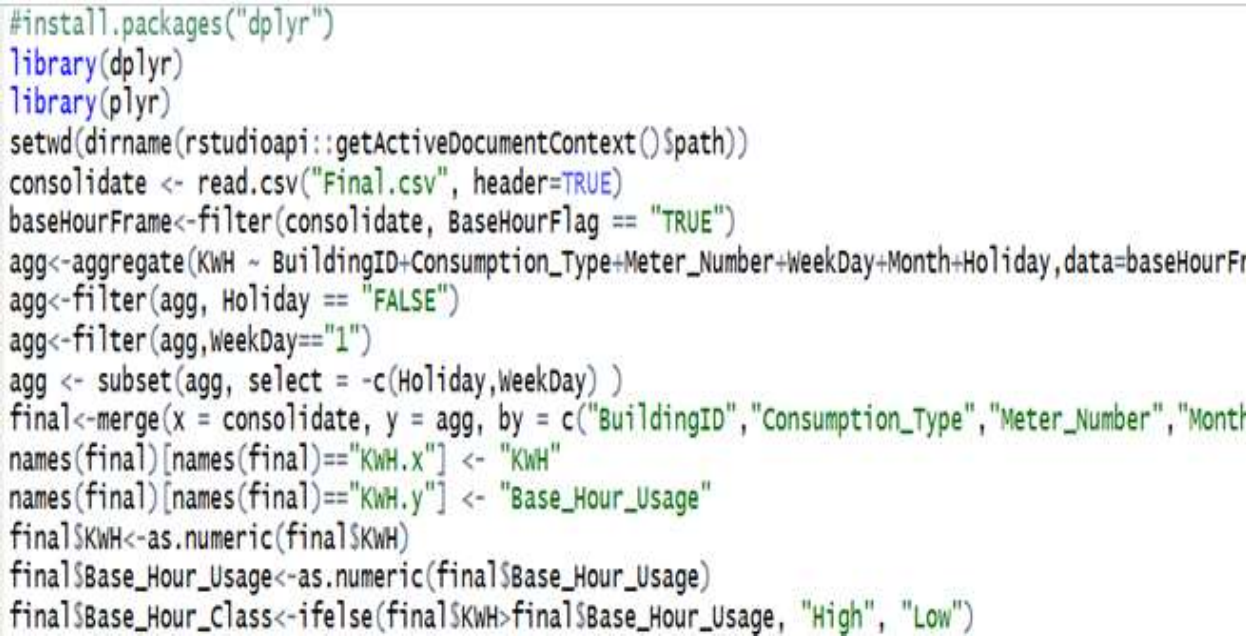
At this point, we will merge the building data and the weather data that is received from scrapping the csv webpage. We have used pandas to merge and remove the duplicates from the code. We have removed the outliers which were either very high or very low.



```
dfMerge['Date']=pd.to_datetime(dfMerge['Date'], format="%Y%m%d")
dfMerge['VisibilityMPH'].replace([-9999],[0],inplace=True)
dfMerge['Wind Direction']=dfMerge['Wind Direction'].astype(str)
dfMerge['Wind Direction'].replace([''],['N/A'],inplace=True)
dfMerge['Conditions']=dfMerge['Conditions'].astype(str)
dfMerge['Conditions'].replace([''],['N/A'],inplace=True)
dfMerge.drop(['TimeEET','DateUTC','PrecipitationIn'],inplace=True,axis=1)
grouped=dfMerge.groupby(['AirportCode','Date','Hour'],as_index=False)
df3=grouped.mean()
df4=grouped.agg({'Wind Direction':lambda x:','.join(x),
                 'Conditions': lambda x: ','.join(x)})
df3['Wind Direction']=df4['Wind Direction']
df3['Conditions']=df4['Conditions']
df = pd.merge(df3, df, on=['AirportCode','Date','Hour'], how='right')
df=df.rename(columns = {'Dew PointF':'Dew_PointF','Sea Level PressureIn':'Sea_Level_PressureIn','Gust Speed
df.to_csv(path+'Final.csv', index=False)
```

Part 2 - Data wrangling and cleaning

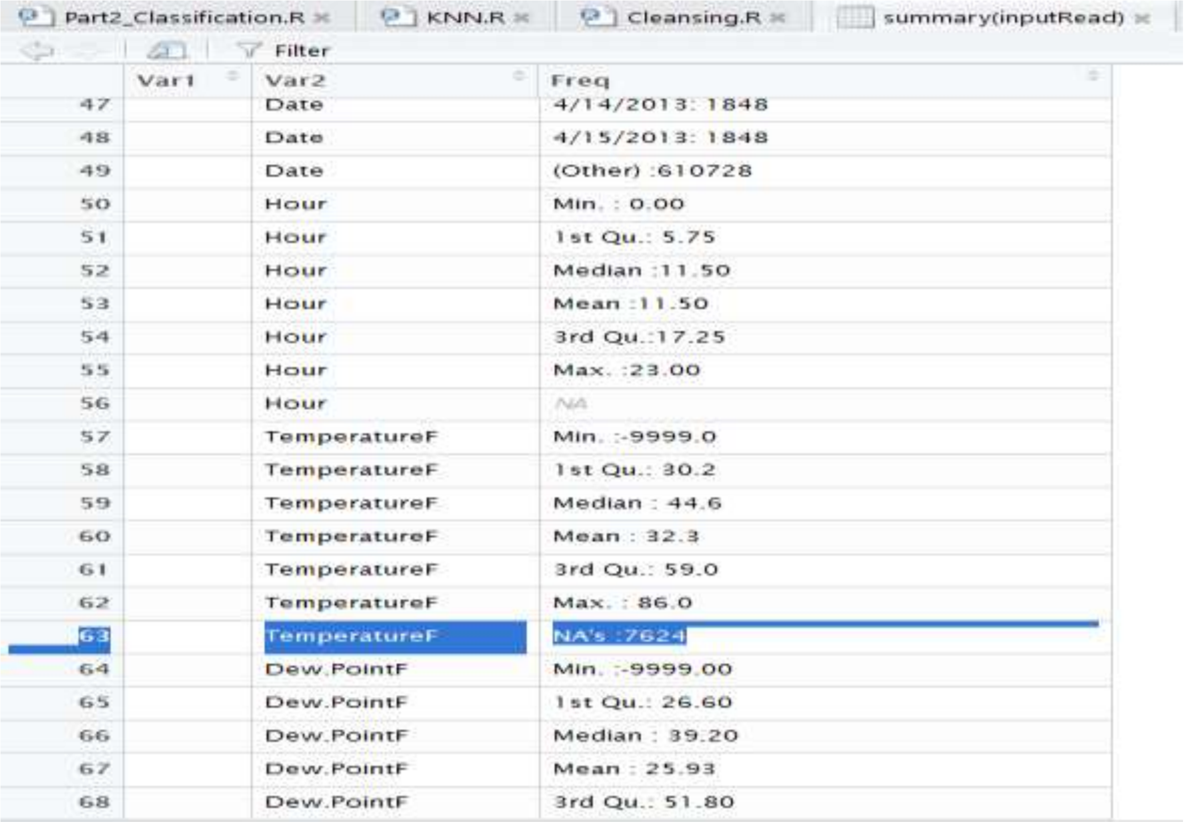
We take the input generated from Part1, and consider the rows with BaseHourFlag(0,1,2,3,4,22,23) is equal to TRUE. Take the mean of the dataset by grouping on BuildingId, Consuption_Type, Meter_Number, WeekDay, Month, Holiday. Compute the Base_Hour_Class. If the KWH consumption is greater than the Base_Hour_Usage.



```
#install.packages("dplyr")
library(dplyr)
library(plyr)
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
consolidate <- read.csv("Final.csv", header=TRUE)
baseHourFrame<-filter(consolidate, BaseHourFlag == "TRUE")
agg<-aggregate(KWH ~ BuildingID+Consumption_Type+Meter_Number+WeekDay+Month+Holiday,data=baseHourFr
agg<-filter(agg, Holiday == "FALSE")
agg<-filter(agg,WeekDay=="1")
agg <- subset(agg, select = -c(Holiday,WeekDay) )
final<-merge(x = consolidate, y = agg, by = c("BuildingID","Consumption_Type","Meter_Number","Month
names(final)[names(final)=="KWH.x"] <- "KWH"
names(final)[names(final)=="KWH.y"] <- "Base_Hour_Usage"
final$KWH<-as.numeric(final$KWH)
final$Base_Hour_Usage<-as.numeric(final$Base_Hour_Usage)
final$Base_Hour_Class<-ifelse(final$KWH>final$Base_Hour_Usage, "High", "Low")
```

We are Cleaning up the NA and removing the outliers from the dataset. We are using zoo package to cleanse the unwanted data.

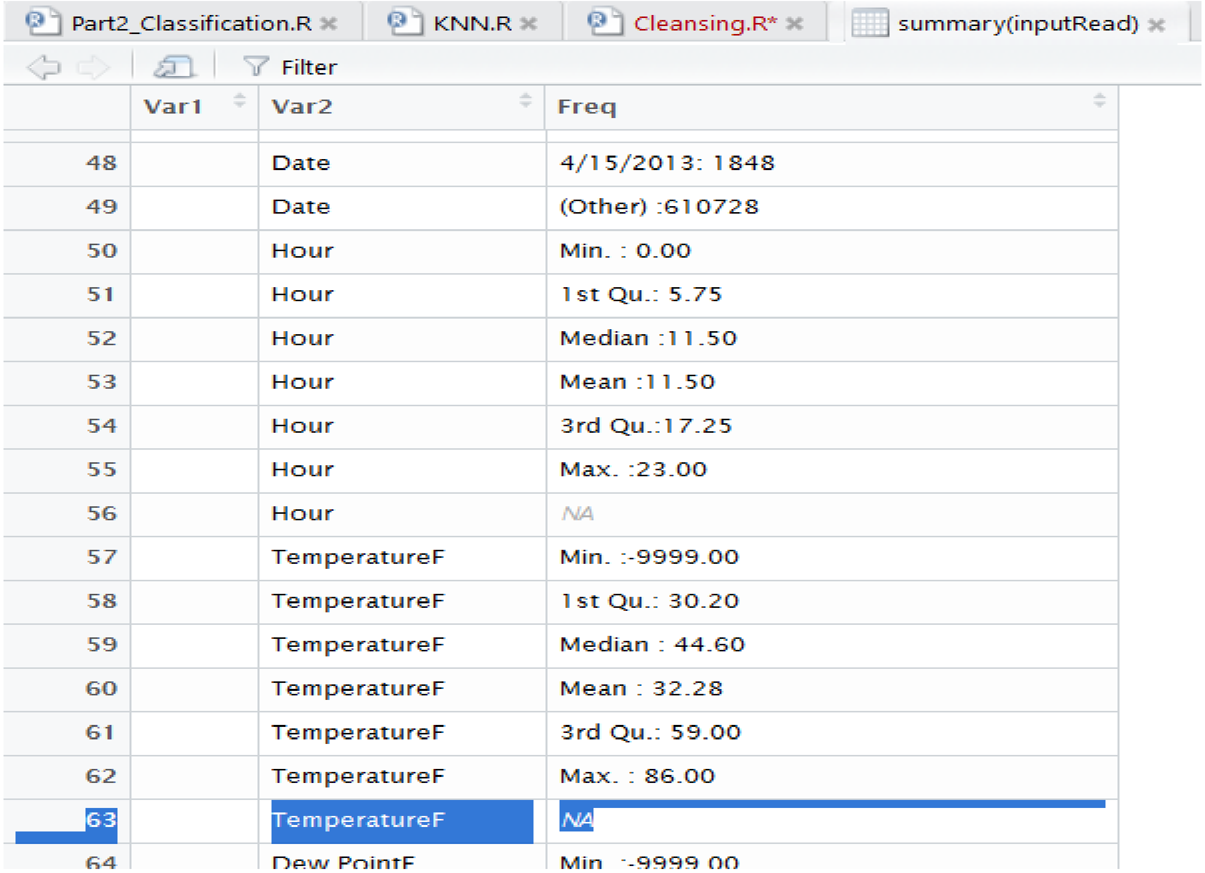
Using View(Summary(inputRead)) we identy the rows with NA.



	Var1	Var2	Freq
47		Date	4/14/2013: 1848
48		Date	4/15/2013: 1848
49		Date	(Other) :610728
50		Hour	Min. : 0.00
51		Hour	1st Qu.: 5.75
52		Hour	Median :11.50
53		Hour	Mean :11.50
54		Hour	3rd Qu.:17.25
55		Hour	Max. :23.00
56		Hour	NA
57		TemperatureF	Min. :-9999.0
58		TemperatureF	1st Qu.: 30.2
59		TemperatureF	Median : 44.6
60		TemperatureF	Mean : 32.3
61		TemperatureF	3rd Qu.: 59.0
62		TemperatureF	Max. : 86.0
63		TemperatureF	NA's :7624
64		Dew.PointF	Min. :-9999.00
65		Dew.PointF	1st Qu.: 26.60
66		Dew.PointF	Median : 39.20
67		Dew.PointF	Mean : 25.93
68		Dew.PointF	3rd Qu.: 51.80

Here we see that the Temperature has lot of NA's. We fill in the NA's with the approx function in the zoo package.

```
library(zoo)

inputRead$TemperatureF[is.na(inputRead$TemperatureF)] <-na.approx(inputRead$TemperatureF)
```



	Var1	Var2	Freq
48		Date	4/15/2013: 1848
49		Date	(Other) :610728
50		Hour	Min. : 0.00
51		Hour	1st Qu.: 5.75
52		Hour	Median :11.50
53		Hour	Mean :11.50
54		Hour	3rd Qu.:17.25
55		Hour	Max. :23.00
56		Hour	NA
57		TemperatureF	Min. :-9999.00
58		TemperatureF	1st Qu.: 30.20
59		TemperatureF	Median : 44.60
60		TemperatureF	Mean : 32.28
61		TemperatureF	3rd Qu.: 59.00
62		TemperatureF	Max. : 86.00
63		TemperatureF	NA
64		Dew PointF	Min. :-9999.00

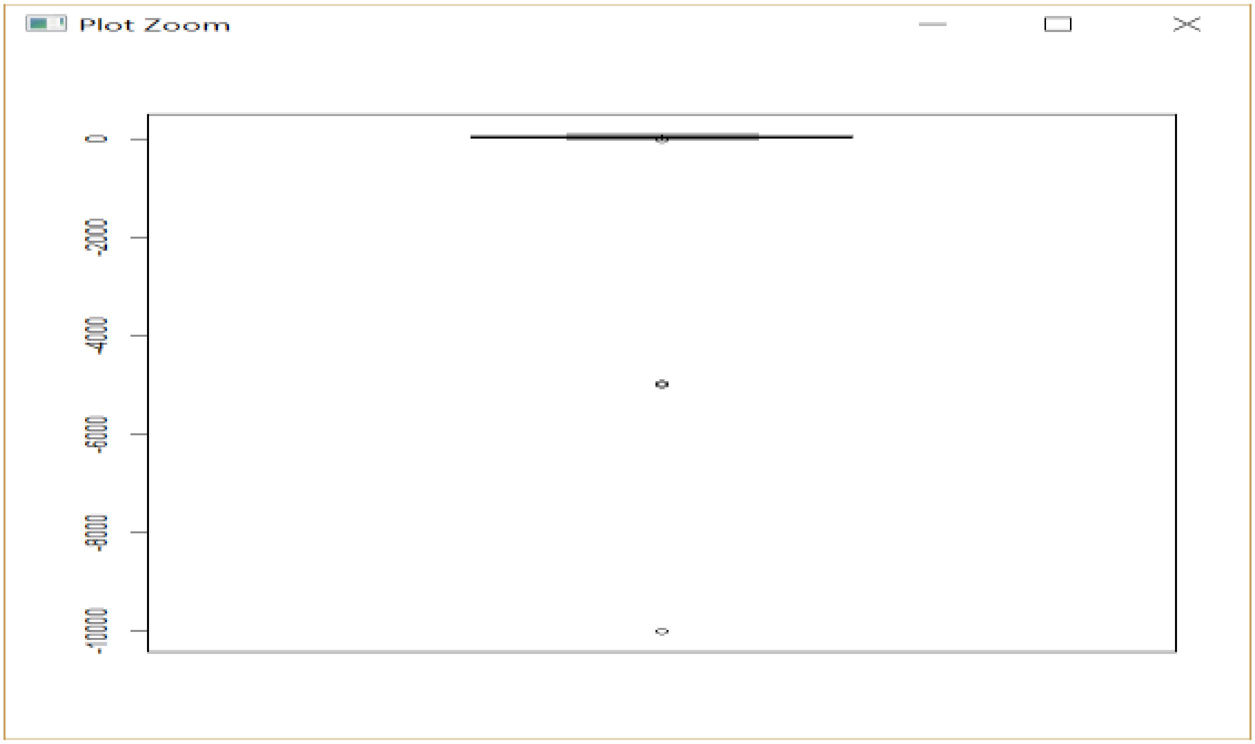
We follow the above steps to fill out na's with approximations for all the other fields.

Outliers detection.

We perform boxplot on individual fields to identy outliers.

boxplot(inputRead$TemperatureF)

We detect the outliers, which are deviating or lie far away in the boxplot. And eliminate them. In the below screenshot we see that the temperature has outliers.

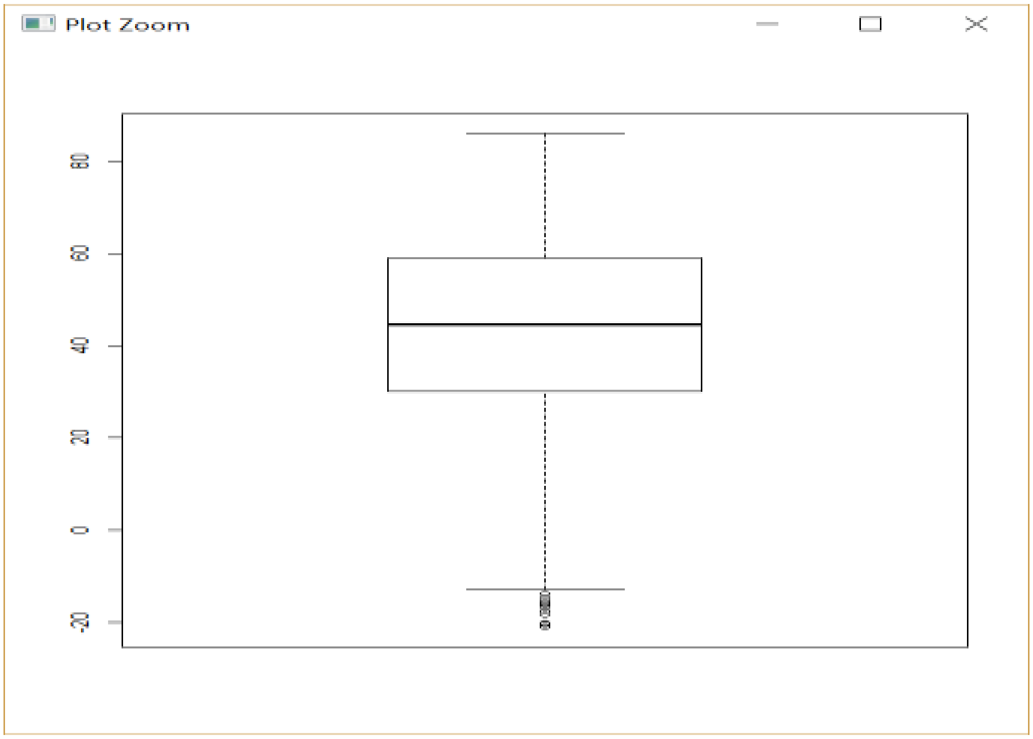


We filter out the rows with Values below -1000, to get the below box blot

inputRead[inputRead$TemperatureF < -1000,]

out <- which(inputRead$TemperatureF < -1000, arr.ind=TRUE)

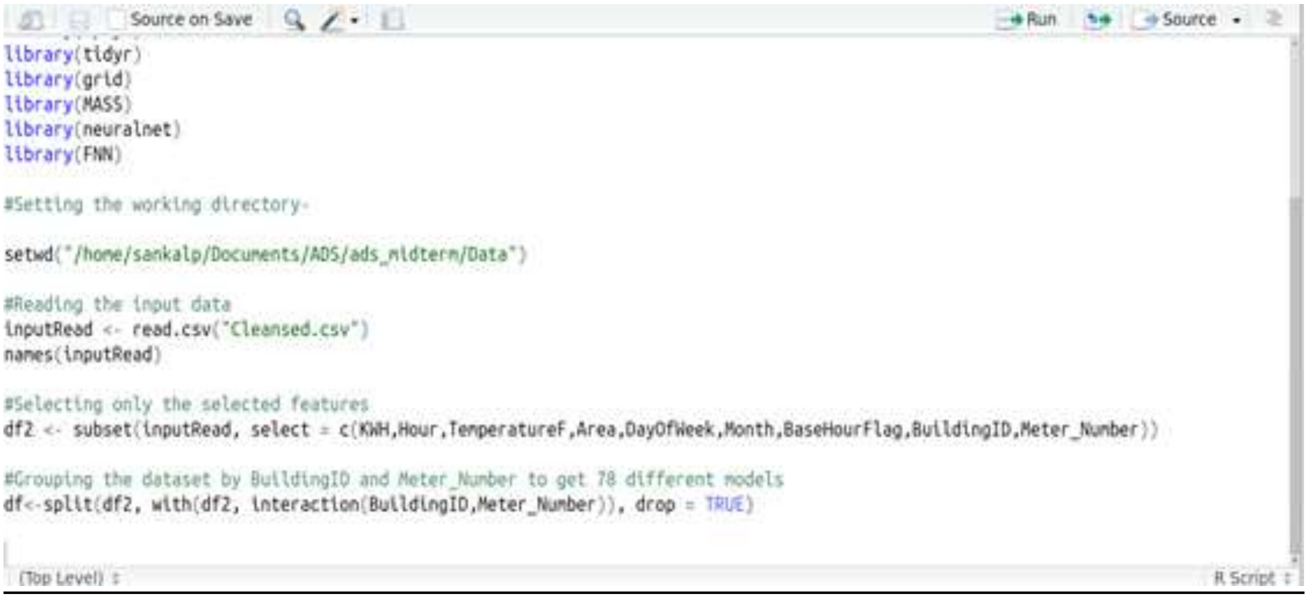
inputRead <- inputRead[-out,]



3. PREDECTION

1. In this step, we first take the dataset and group it by BuildingID and Meter Number to get 78 distinct datasets. We first take the data from the cleansed csv file and store it in inputRead. We then subset the data based on the model selection step where we chose certain features to be used for prediction. We store this subset of data in df2.

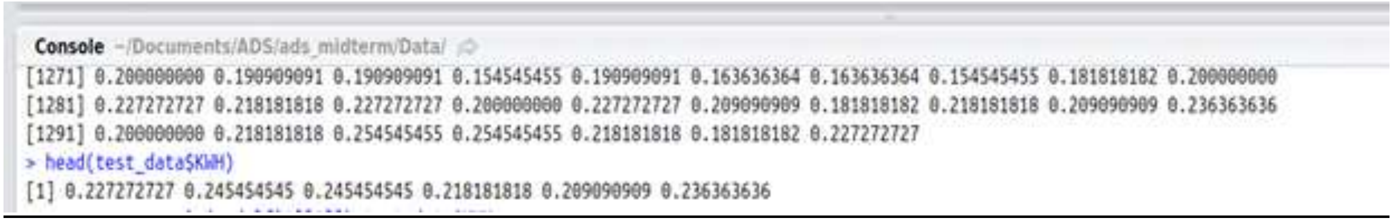
Then, we split this df2 dataset into 78 different parts by grouping the data by BuildingID and Meter Number.



```
library(tidyr)
library(grid)
library(MASS)
library(neuralnet)
library(FNN)

#Setting the working directory-

setwd("/home/sankalp/Documents/ADS/ads_midterm/Data")

#Reading the input data
inputRead <- read.csv("Cleansed.csv")
names(inputRead)

#Selecting only the selected features
df2 <- subset(inputRead, select = c(KWH,Hour,TemperatureF,Area,DayOfWeek,Month,BaseHourFlag,BuildingID,Meter_Number))

#Grouping the dataset by BuildingID and Meter_Number to get 78 different models
df<-split(df2, with(df2, interaction(BuildingID,Meter_Number)), drop = TRUE)
```

2. We apply regression for the 78 models in our dataset “df”.

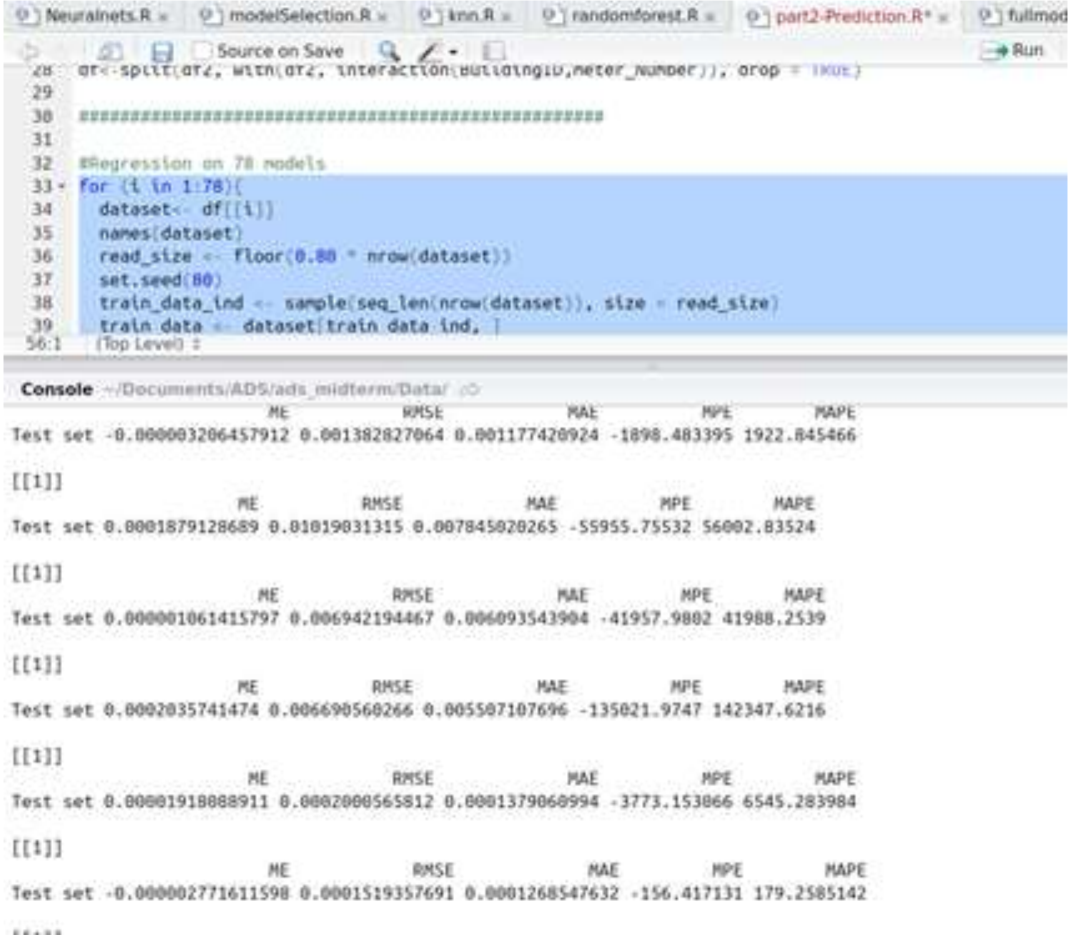
```
        for (i in 1:78){
                dataset<- df[[i]]
                names(dataset)
                read_size <- floor(0.80 * nrow(dataset))
                set.seed(80)
                train_data_ind <- sample(seq_len(nrow(dataset)), size = read_size)
                train_data <- dataset[train_data_ind, ]
                test_data <- dataset[-train_data_ind, ]
                train_data[train_data==0]<-0.000001
                test_data[test_data==0]<- 0.000001
                varnames <- c("Hour", "TemperatureF", "Area",
"DayOfWeek","Month","BaseHourFlag")
                modelfits <- vector(length(varnames), mode = "list")
                names(modelfits) <- varnames
                names(train_data)
                modelfits[[i]]<- lm(KWH~Hour+TemperatureF+Area+DayOfWeek+Month,data
= train_data)
                summary(modelfits[[i]])
                library(forecast)
                pred = predict(modelfits[[i]], test_data)
                accuracy_pred=accuracy(pred, test_data$KWH)
                x <- list(accuracy_pred)
                print(x)
                 summary(modelfits[[i]])
        }
```



```
Console ~/Documents/ADS/ads_midterm/Data/
[1271] 0.200000000 0.190909091 0.190909091 0.154545455 0.190909091 0.163636364 0.163636364 0.154545455 0.181818182 0.200000000
[1281] 0.227272727 0.218181818 0.227272727 0.200000000 0.227272727 0.209090909 0.181818182 0.218181818 0.209090909 0.236363636
[1291] 0.200000000 0.218181818 0.254545455 0.254545455 0.218181818 0.181818182 0.227272727
> head(test_data$KWH)
[1] 0.227272727 0.245454545 0.245454545 0.218181818 0.209090909 0.236363636
```

Regression summary-

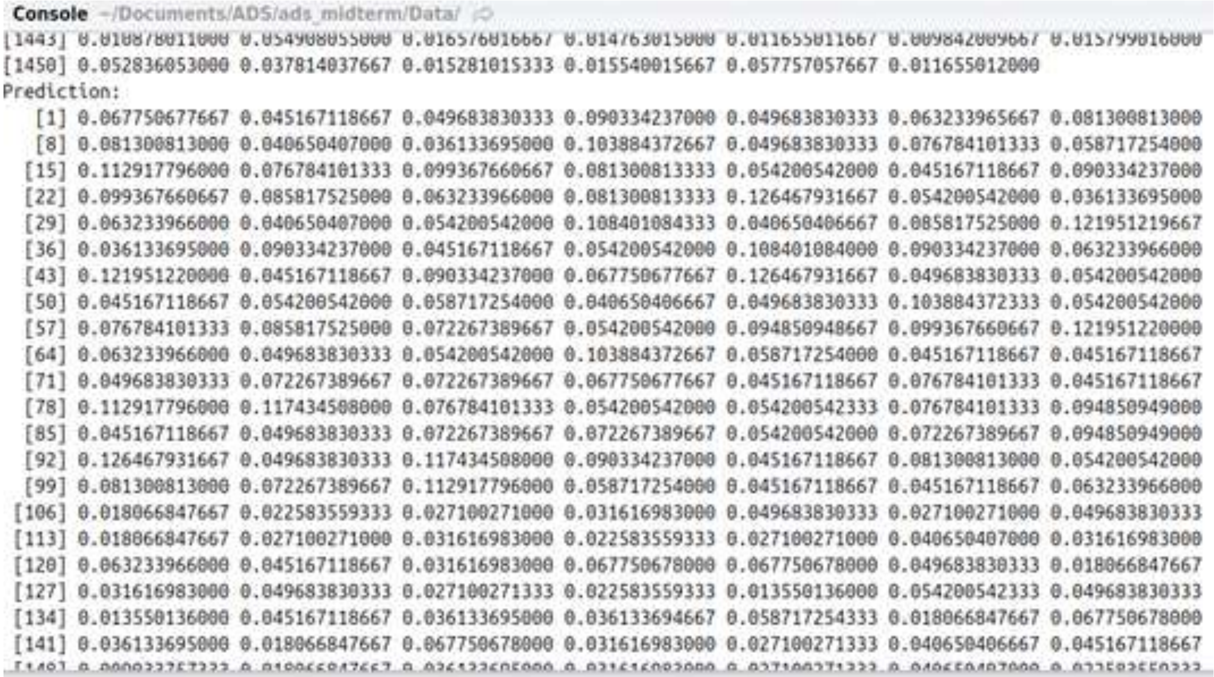
Now we apply KNN algorithm to the datasets.



```
df<-split(df2, with(df2, interaction(BuildingID,meter_Number)), drop = TRUE)

##################################################

#Regression on 78 models
for (i in 1:78){
  dataset<- df[[i]]
  names(dataset)
  read_size <- floor(0.80 * nrow(dataset))
  set.seed(80)
  train_data_ind <- sample(seq_len(nrow(dataset)), size = read_size)
  train_data <- dataset[train_data_ind, ]
```

```
Console ~/Documents/ADS/ads_midterm/Data/
                           ME           RMSE            MAE          MPE         MAPE
Test set -0.000003206457912 0.001382827064 0.001177420924 -1898.483395 1922.845466

[[1]]
                        ME          RMSE            MAE          MPE        MAPE
Test set 0.0001879128689 0.01019031315 0.007845020265 -55955.75532 56002.83524

[[1]]
                          ME           RMSE            MAE         MPE        MAPE
Test set 0.000001061415797 0.006942194467 0.006093543904 -41957.9802 41988.2539

[[1]]
                        ME           RMSE            MAE          MPE        MAPE
Test set 0.0002035741474 0.006690560266 0.005507107696 -135021.9747 142347.6216

[[1]]
                          ME            RMSE             MAE          MPE        MAPE
Test set 0.00001918888911 0.0002000565812 0.0001379060994 -3773.153866 6545.283984

[[1]]
                           ME            RMSE             MAE         MPE          MAPE
Test set -0.000002771611598 0.0001519357691 0.0001268547632 -156.417131 179.2585142
```

Prediction by KNN :

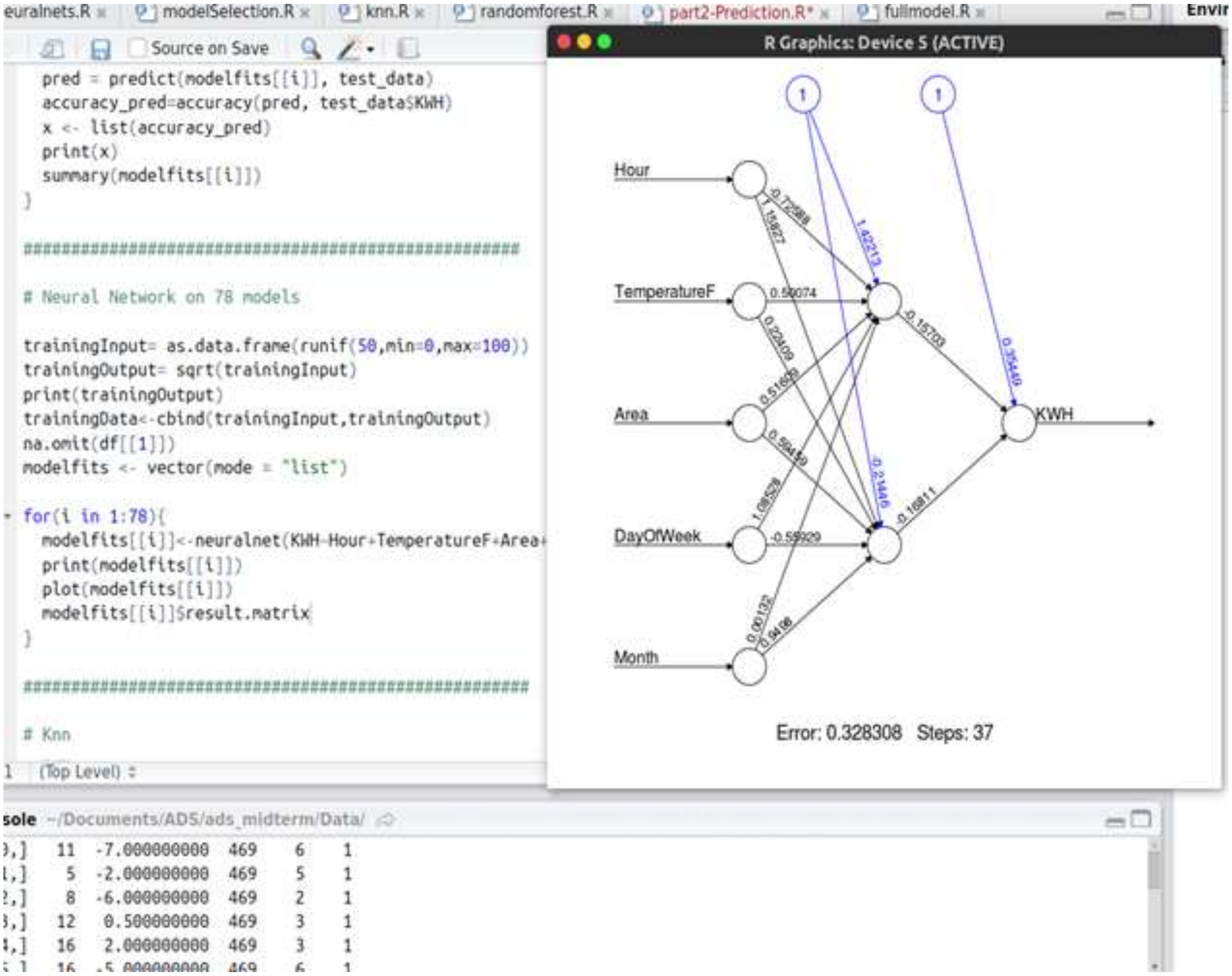


```
[1443] 0.010878011000 0.054908055000 0.016576016667 0.014763015000 0.011655011667 0.009842009667 0.015799016000
[1450] 0.052836053000 0.037814037667 0.015281015333 0.015540015667 0.057757057667 0.011655012000
Prediction:
   [1] 0.067750677667 0.045167118667 0.049683830333 0.090334237000 0.049683830333 0.063233965667 0.081300813000
   [8] 0.081300813000 0.040650407000 0.036133695000 0.103884372667 0.049683830333 0.076784101333 0.058717254000
  [15] 0.112917796000 0.076784101333 0.099367660667 0.081300813333 0.054200542000 0.045167118667 0.090334237000
  [22] 0.099367660667 0.085817525000 0.063233966000 0.081300813333 0.126467931667 0.054200542000 0.036133695000
  [29] 0.063233966000 0.040650407000 0.054200542000 0.108401084333 0.040650406667 0.085817525000 0.121951219667
  [36] 0.036133695000 0.090334237000 0.045167118667 0.054200542000 0.108401084000 0.090334237000 0.063233966000
  [43] 0.121951220000 0.045167118667 0.090334237000 0.067750677667 0.126467931667 0.049683830333 0.054200542000
  [50] 0.045167118667 0.054200542000 0.058717254000 0.040650406667 0.049683830333 0.103884372333 0.054200542000
  [57] 0.076784101333 0.085817525000 0.072267389667 0.054200542000 0.094850948667 0.099367660667 0.121951220000
  [64] 0.063233966000 0.049683830333 0.054200542000 0.103884372667 0.058717254000 0.045167118667 0.045167118667
  [71] 0.049683830333 0.072267389667 0.072267389667 0.067750677667 0.045167118667 0.076784101333 0.045167118667
  [78] 0.112917796000 0.117434508000 0.076784101333 0.054200542000 0.054200542333 0.076784101333 0.094850949000
  [85] 0.045167118667 0.049683830333 0.072267389667 0.072267389667 0.054200542000 0.072267389667 0.094850949000
  [92] 0.126467931667 0.049683830333 0.117434508000 0.090334237000 0.045167118667 0.081300813000 0.054200542000
  [99] 0.081300813000 0.072267389667 0.112917796000 0.058717254000 0.045167118667 0.045167118667 0.063233966000
 [106] 0.018066847667 0.022583559333 0.027100271000 0.031616983000 0.049683830333 0.027100271000 0.049683830333
 [113] 0.018066847667 0.027100271000 0.031616983000 0.022583559333 0.027100271000 0.040650407000 0.031616983000
 [120] 0.063233966000 0.045167118667 0.031616983000 0.067750678000 0.067750678000 0.049683830333 0.018066847667
 [127] 0.031616983000 0.049683830333 0.027100271333 0.022583559333 0.013550136000 0.054200542333 0.049683830333
 [134] 0.013550136000 0.045167118667 0.036133695000 0.036133694667 0.058717254333 0.018066847667 0.067750678000
 [141] 0.036133695000 0.018066847667 0.067750678000 0.031616983000 0.027100271333 0.040650406667 0.045167118667
 [148] 0.009033757333 0.018066847667 0.036133695000 0.031616983000 0.027100271333 0.040650407000 0.022583559333
```

Finally, we apply neural network on our dataset

```
 modelfits[[i]]<-
neuralnet(KWH~Hour+TemperatureF+Area+DayOfWeek+Month,data=df[[i]],hidden=2,threshold=0.01)
 print(modelfits[[i]])
 plot(modelfits[[i]])
 modelfits[[i]]$result.matrix
```

We run this code in for loop for each model.



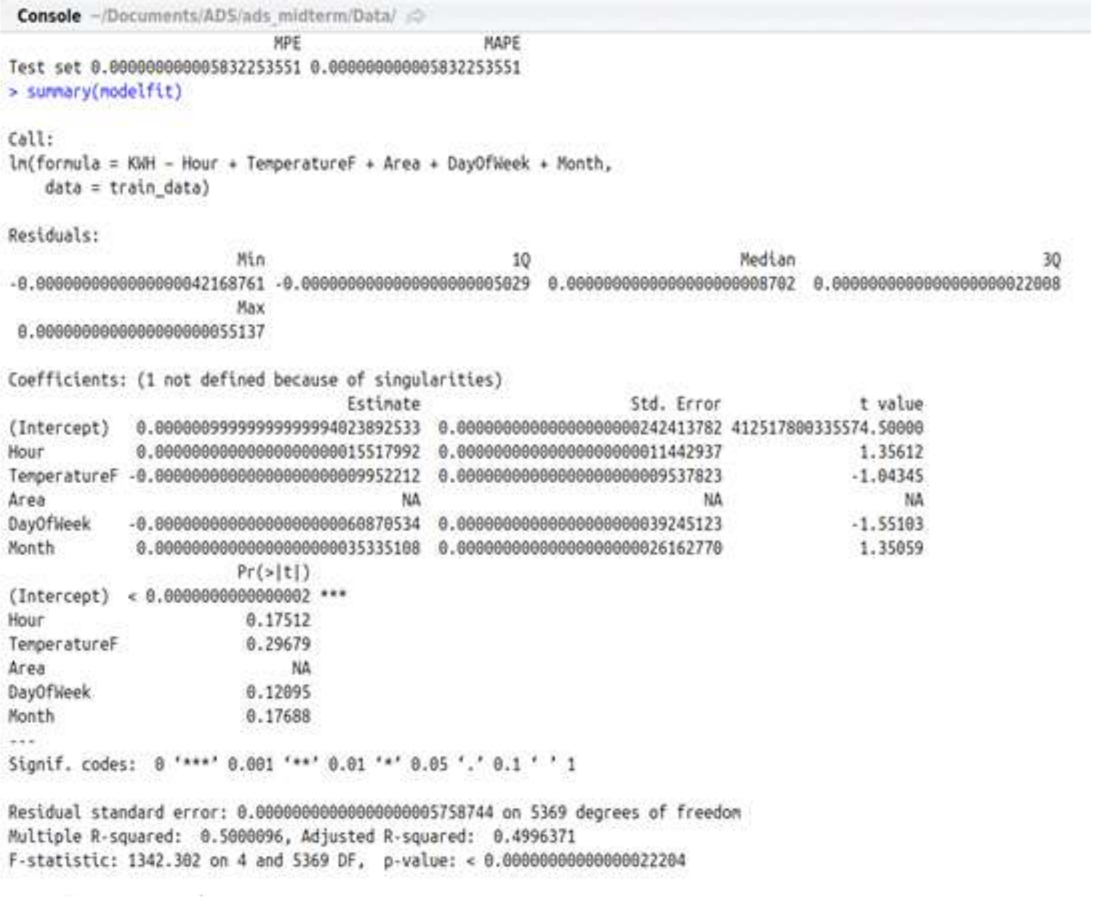
```
  pred = predict(modelfits[[i]], test_data)
  accuracy_pred=accuracy(pred, test_data$KWH)
  x <- list(accuracy_pred)
  print(x)
  summary(modelfits[[i]])
}

##########################################################

# Neural Network on 78 models

trainingInput= as.data.frame(runif(50,min=0,max=100))
trainingOutput= sqrt(trainingInput)
print(trainingOutput)
trainingData<-cbind(trainingInput,trainingOutput)
na.omit(df[[1]])
modelfits <- vector(mode = "list")

for(i in 1:78){
  modelfits[[i]]<-neuralnet(KWH~Hour+TemperatureF+Area+
  print(modelfits[[i]])
  plot(modelfits[[i]])
  modelfits[[i]]$result.matrix
}

###########################################################

# Knn
```

Error: 0.328308 Steps: 37

```
),]   11  -7.000000000  469   6   1
l,]    5  -2.000000000  469   5   1
2,]    8  -6.000000000  469   2   1
3,]   12   0.500000000  469   3   1
4,]   16   2.000000000  469   3   1
5,]   16  -5.000000000  469   6   1
```

3. We apply Regression, KNN and neural network on the entire datasets

When we apply regression on the full dataset, we get the following summary-



```
                         MPE                    MAPE
Test set 0.00000000000005832253551 0.00000000000005832253551
> summary(modelfit)

Call:
lm(formula = KWH ~ Hour + TemperatureF + Area + DayOfWeek + Month, 
    data = train_data)

Residuals:
                       Min                         1Q                     Median                         3Q 
-0.00000000000000000042168761 -0.0000000000000000000000005029  0.0000000000000000000000008702  0.0000000000000000000000022008 
                       Max 
 0.0000000000000000000000055137 

Coefficients: (1 not defined because of singularities)
                                  Estimate                     Std. Error              t value
(Intercept)   0.000000999999999999994023892533  0.000000000000000000000242413782 412517800335574.50000
Hour          0.000000000000000000000000015517992  0.000000000000000000000000011442937               1.35612
TemperatureF -0.000000000000000000000000009952212  0.000000000000000000000000009537823              -1.04345
Area                                       NA                                 NA                    NA
DayOfWeek    -0.000000000000000000000000060870534  0.000000000000000000000000039245123              -1.55103
Month         0.000000000000000000000000035335108  0.000000000000000000000000026162770               1.35059
                          Pr(>|t|)    
(Intercept)  < 0.0000000000000002 ***
Hour                      0.17512    
TemperatureF              0.29679    
Area                           NA    
DayOfWeek                 0.12095    
Month                     0.17688    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.000000000000000000000005758744 on 5369 degrees of freedom
Multiple R-squared:  0.5000096, Adjusted R-squared:  0.4996371 
F-statistic: 1342.302 on 4 and 5369 DF,  p-value: < 0.00000000000000022204
```

By applying all the algorithms on our entire dataset, we get a sense of which model is supposed to perform better based on the RMSE, MAPE, MSE.

RMSE, MAPE, MSE for all models-

1. Regression-

RMSE- 0.1374464
MAE- 0.137446
MAPE- 124232.1

2. KNN-

RMSE- 0.032355
MAE- 0.53440
MAPE- Inf

3. Neural Network-

RMSE- 0.503944
MAE- 0.342290
MAPE- Inf

After observing the RMSE values of the models, we choose KNN as our model.

4. Classification

Classification:

Logistic Regression:

In logistic Regression classification we make use of the library(caret).
We take the cleansed data, and find out the percentage of High, Low values for

```
table(inputRead$Base_Hour_Class)/nrow(inputRead)
```

Divide the train and test data, as below

```
smp_size=floor(0.54*nrow(inputRead))

set.seed(123)

train_ind<-sample(seq_len(nrow(inputRead)),size=smp_size)

train<-inputRead[train_ind,]

test<-inputRead[-train_ind,]
```

We fit the logistic Regression model using the below code:

```
fit<-
glm(Base_Hour_Class~BuildingID+Consumption_Type+Meter_Number+Month+Hou
r+TemperatureF+Dew_PointF+Humidity+Sea_Level_PressureIn+WindDirDegrees
+KWH+DayOfWeek+WeekDay,data=train,family=binomial(link="logit"))
```

Generated the Confusion Matrix and write to the CSV file using

```
cm<-confusionMatrix(test$Base_Hour_Class,pred)

tocsv <- data.frame(cbind(t(cm$overall),t(cm$byClass)))

write.csv(tocsv,file="LogisticRegression_ConfusionMatrix.csv",row.name
s=FALSE)
```

We generate the ROC curve by

```
prediction<-prediction(test$predictions,test$Base_Hour_Class)

performance<-performance(prediction,measure="tpr",x.measure="fpr")

plot(performance,main="ROC Curve",xlab="1-Specificity",ylab="Sensitivity")
```

LogisticRegression_Prediction.csv,LogisticRegression_ConfusionMatrix.csv were generated.

KNN Algorithm:

KNN alogrithm works on normalized and numerical data. Hence the columns chosen were numeric.

```
normalize <- function(x) {

  return ((x - min(x)) / (max(x) - min(x))) }


inputReadNumeric <- subset(inputRead, select=c('TemperatureF','Dew_PointF','Humidity','Sea_Level_PressureIn','VisibilityMPH','WindDirDegrees','KWH','WeekDay','DayOfWeek','Base_Hour_Class'))

inputReadNormalized <- as.data.frame(lapply(inputReadNumeric[,c('TemperatureF','Dew_PointF','Humidity','Sea_Level_PressureIn','VisibilityMPH','WindDirDegrees','KWH','WeekDay','DayOfWeek')], normalize))
```

Use library(class) to fit the KNN training algorithm as below,80 was chosen since it is the square root of the number of rows that were present the dataset.

```
     m1<-knn(train=train,test=test,cl=train_target,k=80)
```

Random Forest:

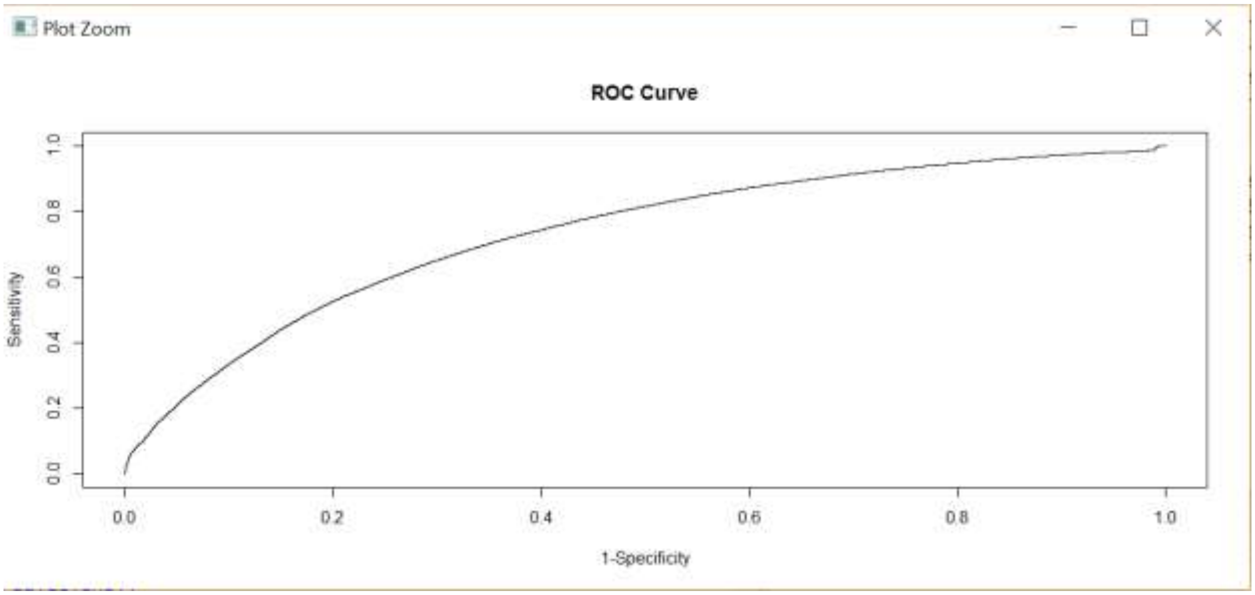
We create the train and test dataset that are created as mentioned above and We fit the data for Random Forest using

```
rforest <- randomForest(concatVal, train, ntree=100,importance=T)
```

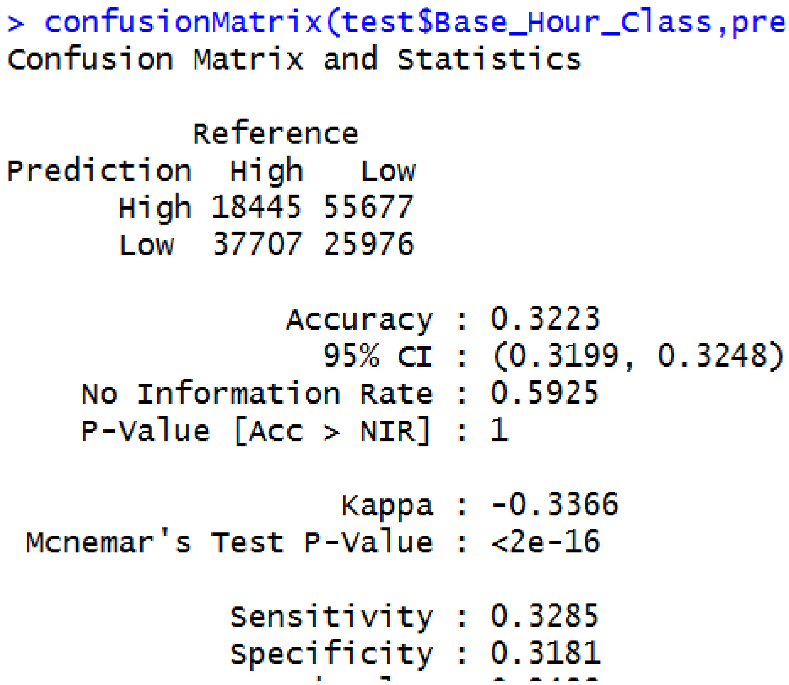
We consider 'TemperatureF','Dew_PointF','Humidity','Sea_Level_PressureIn','VisibilityMPH','WindDirDegrees','KWH','WeekDay','DayOfWeek','Base_Hour_Class' numeric values since random forest works best with Random Forest.

RandomForest_Predictions.csv, RF_ConfusionMatrix.csv were the output files generated.

3) RoC Curve for Logistic Regression

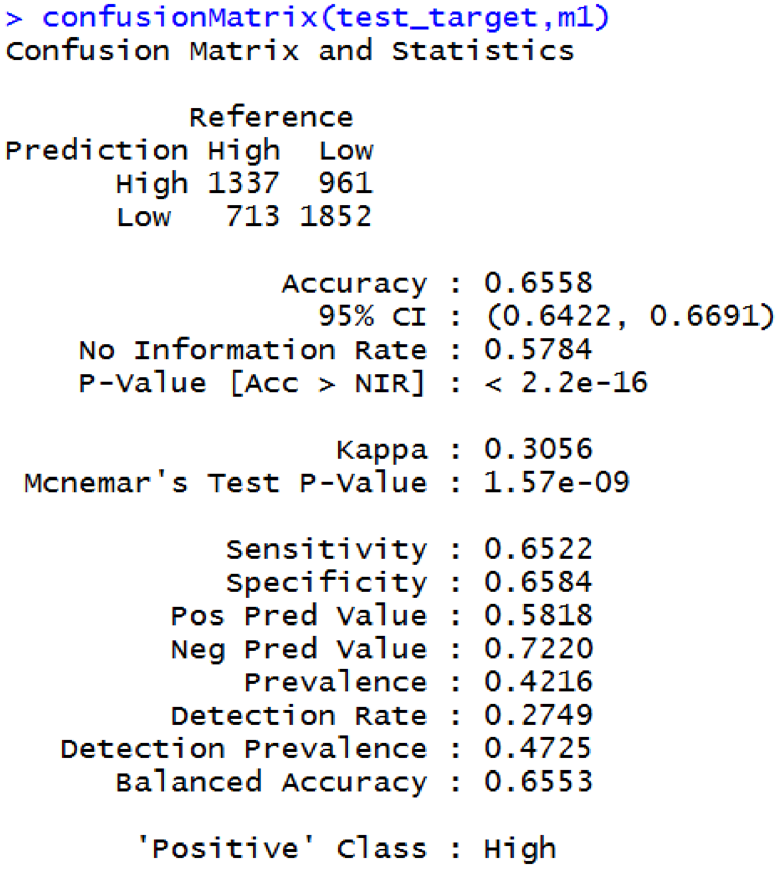


- **Confustion Matric for Logistic Regression**



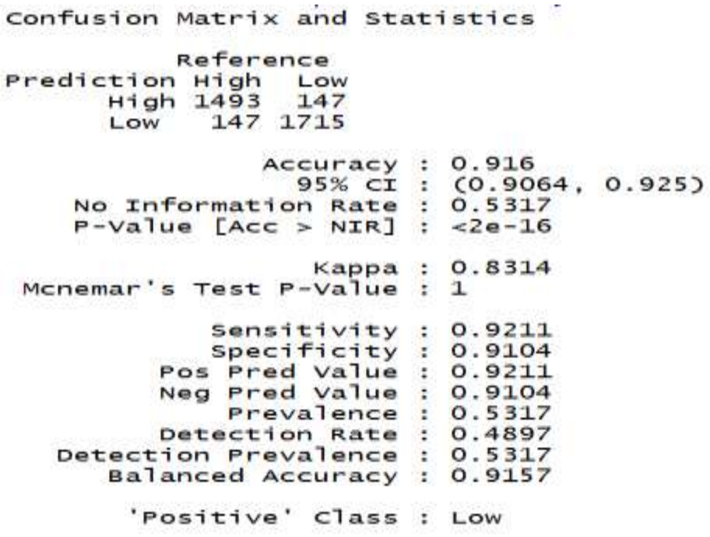
```
> confusionMatrix(test$Base_Hour_Class,pred)
Confusion Matrix and Statistics

          Reference
Prediction  High   Low
      High 18445 55677
      Low  37707 25976

               Accuracy : 0.3223
                 95% CI : (0.3199, 0.3248)
    No Information Rate : 0.5925
    P-Value [Acc > NIR] : 1

                  Kappa : -0.3366
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.3285
            Specificity : 0.3181
```

- **Confusion Matrix for KNN Algorithm**



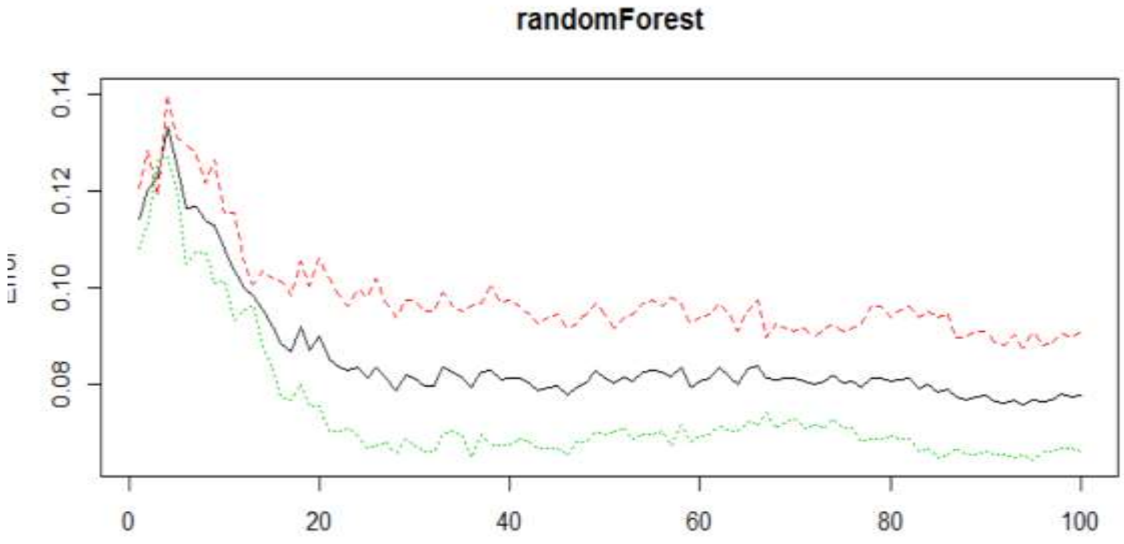
```
> confusionMatrix(test_target,m1)
Confusion Matrix and Statistics

          Reference
Prediction High  Low
      High 1337  961
      Low   713 1852

               Accuracy : 0.6558
                 95% CI : (0.6422, 0.6691)
    No Information Rate : 0.5784
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3056
 Mcnemar's Test P-Value : 1.57e-09

            Sensitivity : 0.6522
            Specificity : 0.6584
         Pos Pred Value : 0.5818
         Neg Pred Value : 0.7220
             Prevalence : 0.4216
         Detection Rate : 0.2749
   Detection Prevalence : 0.4725
      Balanced Accuracy : 0.6553

       'Positive' Class : High
```

- **Confusion Matrix for Random Forest**



```
Confusion Matrix and Statistics

          Reference
Prediction High  Low
      High 1493  147
      Low   147 1715

               Accuracy : 0.916
                 95% CI : (0.9064, 0.925)
    No Information Rate : 0.5317
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8314
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9211
            Specificity : 0.9104
         Pos Pred Value : 0.9211
         Neg Pred Value : 0.9104
             Prevalence : 0.5317
         Detection Rate : 0.4897
   Detection Prevalence : 0.5317
      Balanced Accuracy : 0.9157

       'Positive' Class : Low
```

- **ROC graph for Random Forest**

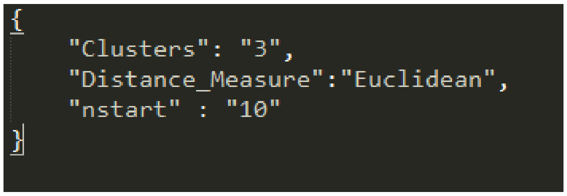


4) Going by the Confusion Matrix the predictions given by the randomForest were found to be more accurate than the other models.

Random Forest gives the high accuracy of 0.916 as compared to 0.65 and 0.32 given by KNN and Logistic Regression respectively.

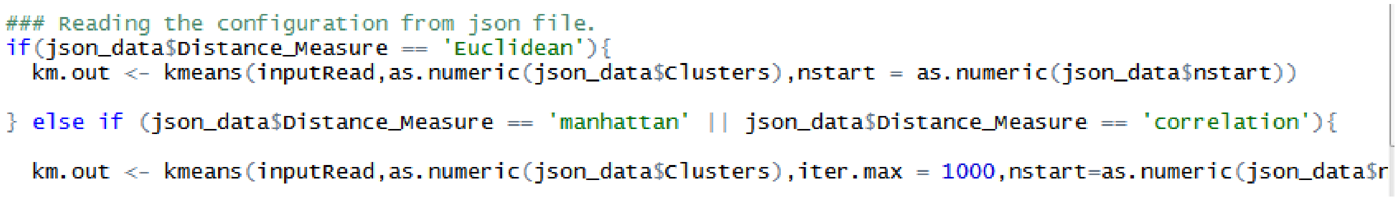
5. CLUSTERING

We created a Json file , from where we can change the configuration of the clustering. It has three option, to choose the cluster, name the distance_measure and the nstart.



```
{
    "Clusters": "3",
    "Distance_Measure":"Euclidean",
    "nstart" : "10"
}
```

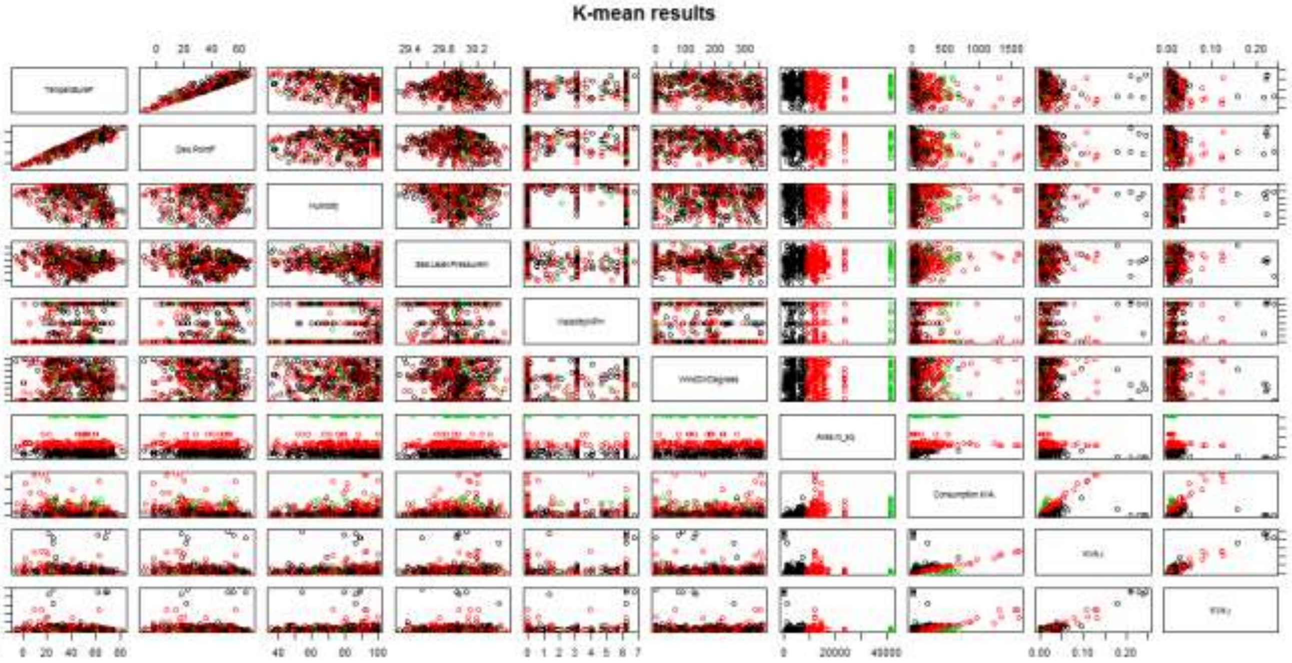
Code snippet



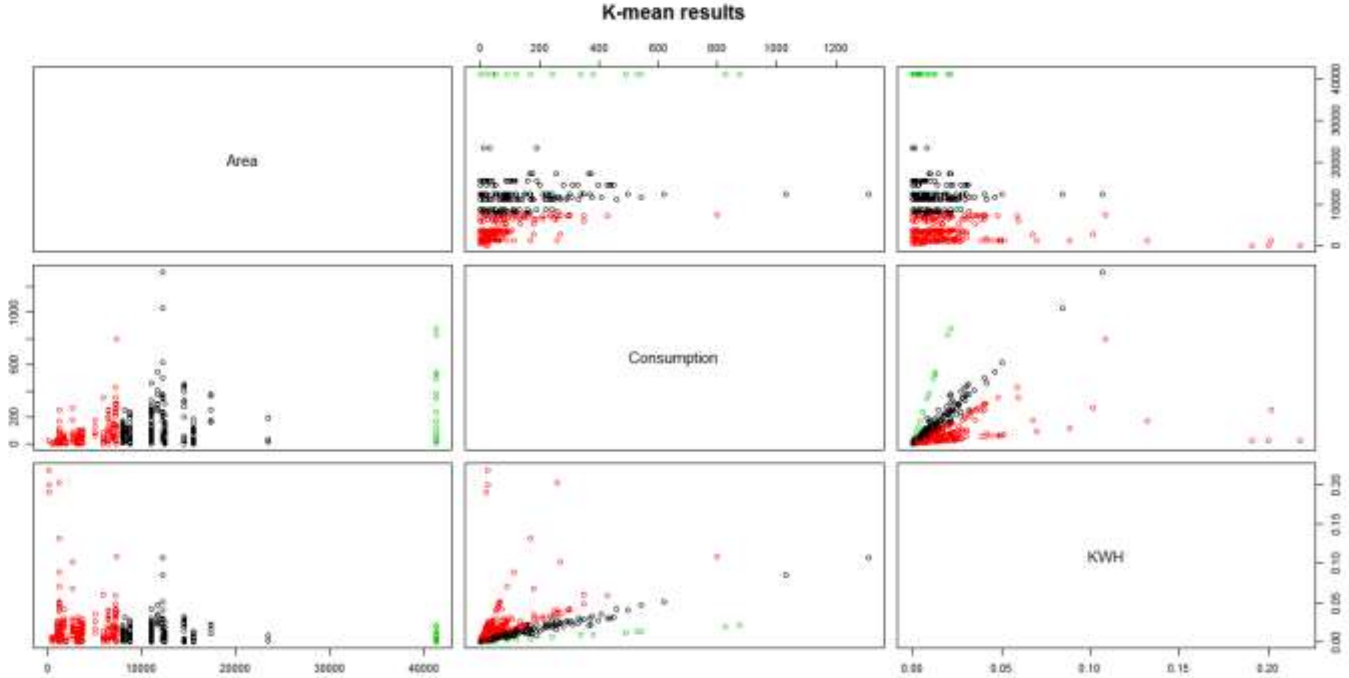
```
### Reading the configuration from json file.
if(json_data$Distance_Measure == 'Euclidean'){
  km.out <- kmeans(inputRead,as.numeric(json_data$Clusters),nstart = as.numeric(json_data$nstart))

} else if (json_data$Distance_Measure == 'manhattan' || json_data$Distance_Measure == 'correlation'){

  km.out <- kmeans(inputRead,as.numeric(json_data$Clusters),iter.max = 1000,nstart=as.numeric(json_data$r
```

K-Means

The data is huge and it has large number of rows. If we apply the k-means on total rows, it’s is difficult to understand the clustering. We deployed the graph for 10 columns. Below are the results ::



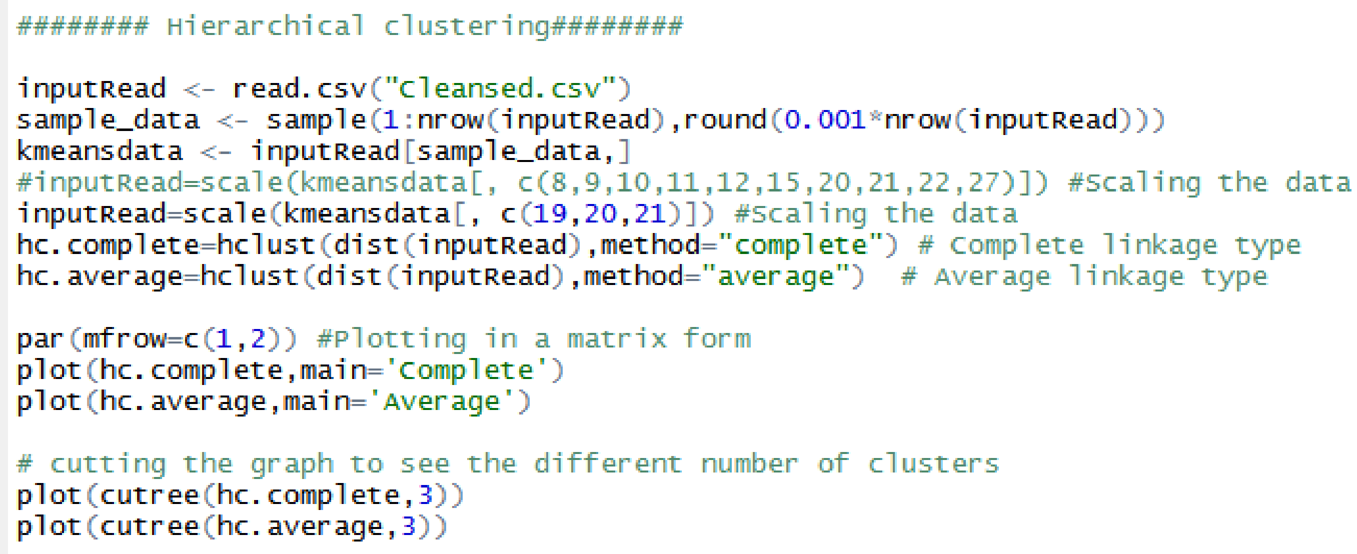
Hence, We decided to proceed only with three values, which are related to the building more directly. We choose area, consumption, KWH. Below are the cluster plot ::



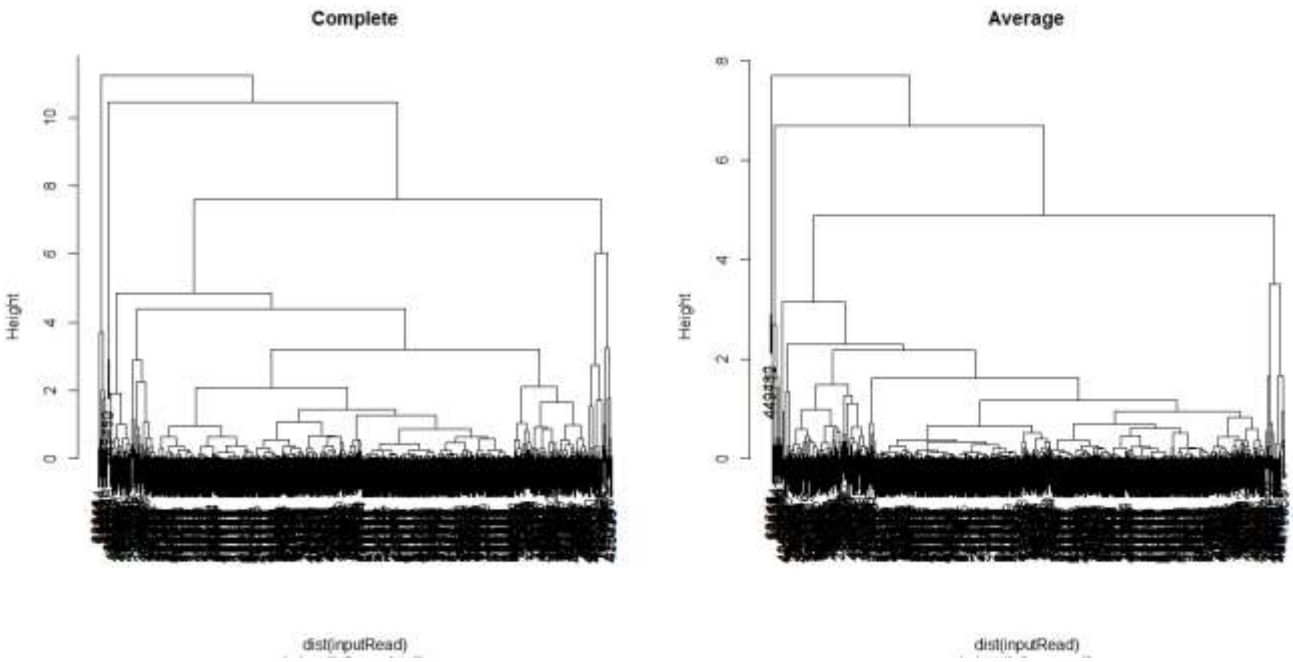
We can observe that mostly all the clusters are separated from each other and there is very less overlapping.

Hierarchical clustering

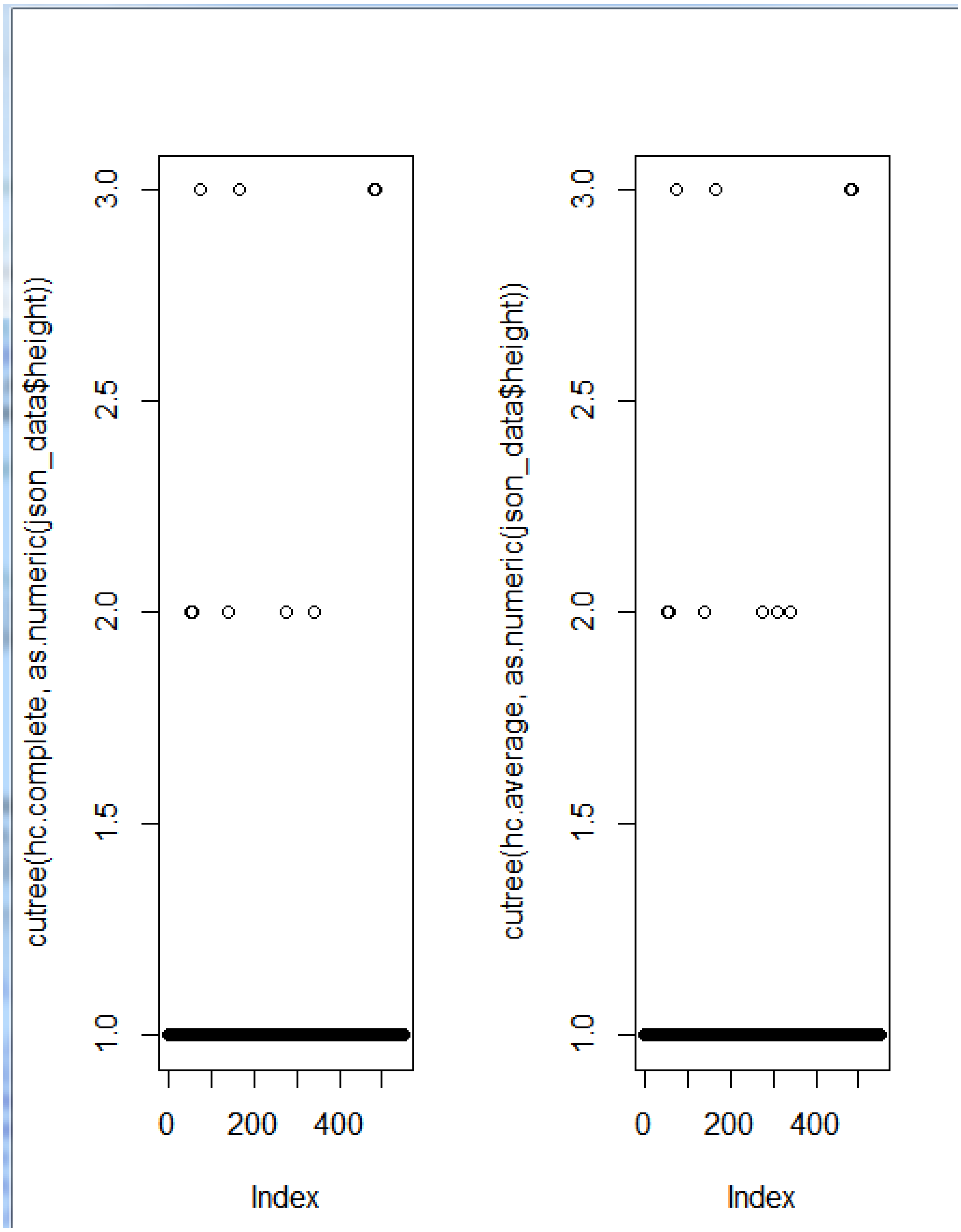
In case of Hierarchical clustering, we don’t provide the cluster size. we can get the number of clusters by cutting Dendrogram at any point.



```
######## Hierarchical clustering########

inputRead <- read.csv("Cleansed.csv")
sample_data <- sample(1:nrow(inputRead),round(0.001*nrow(inputRead)))
kmeansdata <- inputRead[sample_data,]
#inputRead=scale(kmeansdata[, c(8,9,10,11,12,15,20,21,22,27)]) #Scaling the data
inputRead=scale(kmeansdata[, c(19,20,21)]) #Scaling the data
hc.complete=hclust(dist(inputRead),method="complete") # Complete linkage type
hc.average=hclust(dist(inputRead),method="average")  # Average linkage type

par(mfrow=c(1,2)) #Plotting in a matrix form
plot(hc.complete,main='Complete')
plot(hc.average,main='Average')

# cutting the graph to see the different number of clusters
plot(cutree(hc.complete,3))
plot(cutree(hc.average,3))
```



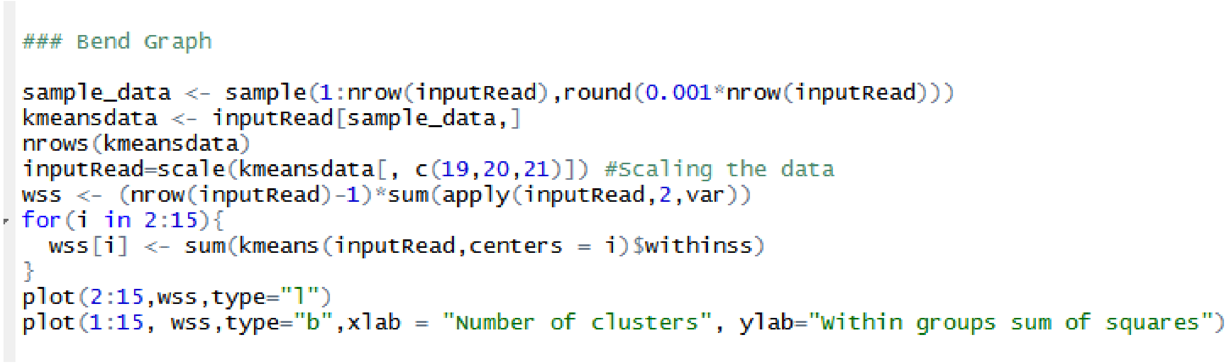
Now, we cut the graph to see the different number of clusters at a point. We choose the height as 3. Below is the plot after treecut-



We find that when the tree is cut by height of 3, we can three set of data on the plot. One lying around 1.0, another at 2.0 and few at top 3.0.

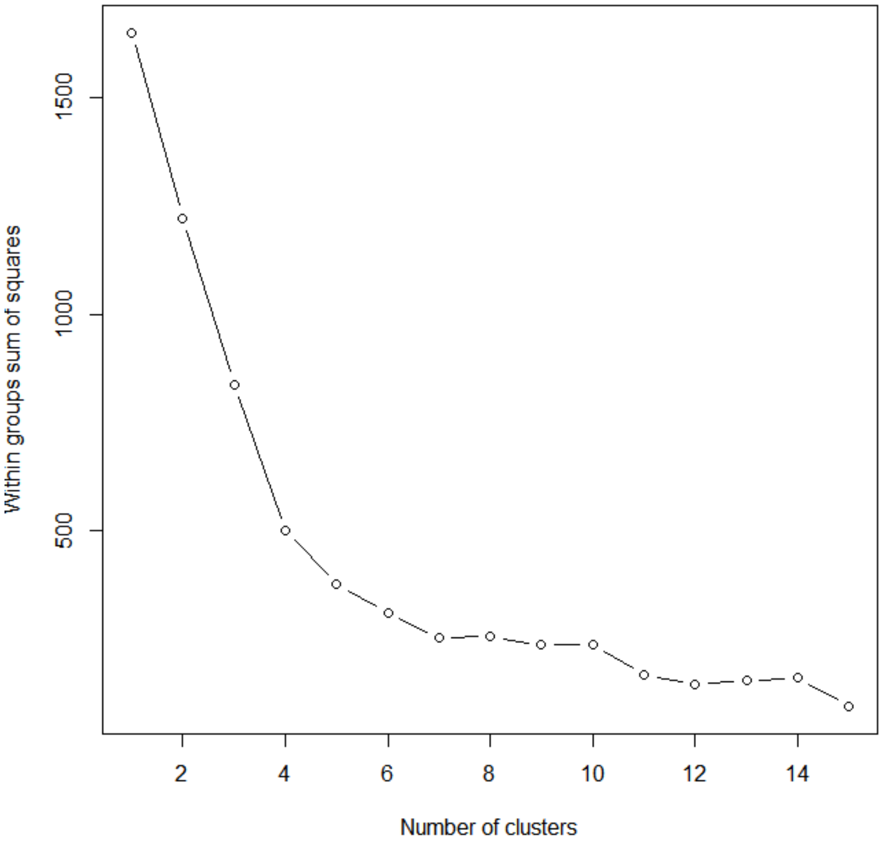
Bend Graph

Bend graph is used to find the optimal value of k. k is the number of cluster to be used.



```
### Bend Graph

sample_data <- sample(1:nrow(inputRead),round(0.001*nrow(inputRead)))
kmeansdata <- inputRead[sample_data,]
nrows(kmeansdata)
inputRead=scale(kmeansdata[, c(19,20,21)]) #Scaling the data
wss <- (nrow(inputRead)-1)*sum(apply(inputRead,2,var))
for(i in 2:15){
  wss[i] <- sum(kmeans(inputRead,centers = i)$withinss)
}
plot(2:15,wss,type="l")
plot(1:15, wss,type="b",xlab = "Number of clusters", ylab="Within groups sum of squares")
```

Plot



Going through the graph plot, take the number of cluster where the slope is more. we can see that till cluster , there is a steep slope. After cluster 4, there is change in plotting. From cluster 4-5, 5-6,6-7, we have the decreasing slope. From cluster 8, we can see the slope is not increasing effectively. Since, we see that from cluster 4 – 5, is a change in trend and we see an effective curse, we take the value of k as 5.