Part1: Data Ingestion and Wrangling

Step 1: - Extract Features

i) Columns like Week of day, Month of year, Weekday/Weekend were derived from the date field in the raw data.

ii) To get the list of holidays:

a) API call to dateandtime.com with Date and country as the parameters.
b) To get the country, another API call to google had to be called giving the address of the building as the parameter.
c) To get the location, gdata library was used which internally calls google API to get the results.

```r
#Retrieving List of Holidays
url_time_api <- "http://www.timeanddate.com/calendar/custom.html"
time_api <- read_html(url_time_api)

country_form <- xml_find_all(time_api,".//form")
country <- xml_find_all(country_form,".//option")

#index_of_country

ct_id =1
index_of_country =0
for(cntry in xml_text(country)){
  if(cntry==country_of_building){
    index_of_country  = ct_id
    break
  }
  ct_id = ct_id+1
}
if(index_of_country!=0){
  option_index <- toString(country[index_of_country])
  country_code_for_Holiday = substr(option_index,16,17)
}

yr = "2013"
params <- paste(paste("year",yr,sep="="),paste("country",country_code_for_Holiday,sep="=")
                ,paste("holm","1",sep="="),paste("hol","9",sep="="),paste("df","1",sep="="),sep="&")

#Building url to get Holiday Dynamically
url_time_api_for_country <- paste(url_time_api,params,sep = "?")

#getting List of Holidays
holiday_api <- GET(url_time_api_for_country,add_headers("Accept-Language"="en-US"))
holiday_html <- read_html(holiday_api$content)

holidays_table <- xml_find_all(holiday_html,".//table")
holidays_node <- xml_find_all(holidays_table,".//span")
```

iii) Normalize the consumption value by dividing it with the total floor area

iv) To get the weather data:

a) Get the latitude and longitude for the address
b) Pass the coordinates to the url to get the link

c) Read the XML to extract the nearest airport
d) From the nearest airport get the airport code
e) Get start date, end date from the raw data
f) Get weather data from library ‘getWeatherData’

```
for (e in df1$X..address){
# Finding Latitude & longitude for the address
  lat_lon <- geocode(e, source = "google",sensor = TRUE)
  cordinates <- paste(lat_lon$lat,lat_lon$lon,sep=",")
#Passing the cordinates to the url
  url_loc <- paste(url_api,cordinates,sep="=")
#Reading the XML
  data <- read_xml(url_loc)
#Finding the nearest airport
  data_airport <- xml_find_all(data,".//airport")
  data_air_codes <- xml_find_all(data_airport,".//icao")
  codes <- xml_contents(data_air_codes)
  code <- toString(codes[1])
  code_list[i] <- code
  i = i+1
}
```

```
endDate = format(as.Date(tail(a$date,1), format="%Y%m%d"),"%Y-%m-%d")
endDate <- as.Date(endDate)
startDate <- as.Date(startDate)


for (c_air in unique(code_list)){
  d3 <- getWeatherForDate(c_air, start_date=startDate,
                          end_date = endDate,
                          opt_detailed = TRUE,
                          opt_all_columns = TRUE)
  d3$airport_code = c_air
  d3 <- select(d3,Time,TemperatureF,Dew_PointF,Humidity,Sea_Level_PressureIn,VisibilityMPH,
               Wind_Direction,Wind_SpeedMPH,Gust_SpeedMPH,PrecipitationIn,
               Events,Conditions,WindDirDegrees,airport_code)
  weatherData <- rbind(weatherData,d3)
}
```

Step 2: - Capturing the error log

The error logs were captured and saved into an error log file.

```
for (dates_in_list in holidays_string_list){
  tryCatch(if(grep(pattern="[0-9]",x=dates_in_list)){
    holi_c[h] = dates_in_list
    h=h+1
  },
  error = function(e){
    NaN;
  }
  )
}
```

error.log - Notepad

File Edit Format View Help

```
Error: object 'ldddng' not found
Error: object 'ldddng' not found
NULL
Error in a + b : non-numeric argument to binary operator
function (..., recursive = FALSE)  .Primitive("c")
Error: object 'ldddng' not found
Error in a + b : non-numeric argument to binary operator
Error: object 'ldddng' not found
Error in "stromg" + 2 : non-numeric argument to binary operator
Error in "ssd" + 1 : non-numeric argument to binary operator
Error: object 'ldddng' not found
```

Step 3: - Removing the NA and empty values

i) There were a lot of NA or empty values in weather data. They were handled case by case according to the domain knowledge:

a) Fields like Temperature, Dew_PointF, Humidity, Sea_Level_PressureIn, VisibilityMPH and Wind_SpeedMPH were imputed by using imputation by interpolating the values.

b) WindDirDegrees was filled out with random values as they were changing randomly with no defined pattern.

c) Conditions field was replaced with the last known value as it was observed that the value of Conditions does not change frequently hourly.

```r
#Replacing NA values with linear interpolation
processedOp$TemperatureF <- na.approx(processedOp$TemperatureF,na.rm = FALSE)
processedOp$Dew_PointF <- na.approx(processedOp$Dew_PointF,na.rm = FALSE)
processedOp$Humidity <- na.approx(processedOp$Humidity,na.rm = FALSE)
processedOp$Sea_Level_PressureIn <- na.approx(processedOp$Sea_Level_PressureIn,na.rm = FALSE)
processedOp$VisibilityMPH <- na.approx(processedOp$VisibilityMPH,na.rm = FALSE)
processedOp$Wind_SpeedMPH <- na.approx(processedOp$Wind_SpeedMPH,na.rm = FALSE)


#Replacing NA values randomly
processedOp$WindDirDegrees[is.na(processedOp$WindDirDegrees)]<- sample(1:36,1)*10

#Replacing NA with last known non null value
processedOp$Conditions <-  na.locf(processedOp$Conditions,fromLast = TRUE,na.rm = FALSE)
```

Step 4: - Removing the outliers

i) There were a lot of outlier values in fields like Temperature, Dew_PointF, Humidity, Sea_Level_PressureIn, VisibilityMPH and Wind_SpeedMPH.

ii) Outliers which were beyond 1.5x IQR were replaced by NA and then later imputed along with other NA values.

```
#Remove the outliers using Boxplot
processedOp$TemperatureF[processedOp$TemperatureF %in% boxplot.stats(processedOp$TemperatureF)$out] <- NA
processedOp$Dew_PointF[processedOp$Dew_PointF %in% boxplot.stats(processedOp$Dew_PointF)$out] <- NA
processedOp$Humidity[processedOp$Humidity %in% boxplot.stats(processedOp$Humidity)$out] <- NA
processedOp$Sea_Level_PressureIn[processedOp$Sea_Level_PressureIn %in% boxplot.stats(processedOp$Sea_Level_Pr
processedOp$VisibilityMPH[processedOp$VisibilityMPH %in% boxplot.stats(processedOp$VisibilityMPH)$out] <- NA
processedOp$Wind_SpeedMPH[processedOp$Wind_SpeedMPH %in% boxplot.stats(processedOp$Wind_SpeedMPH)$out] <- NA
```

Step 4: - Merge the data

i) The address data and the consumption data are merged on the Building Name.

ii) The output of step (i) is merged with the weather data according to the nearest airport code, date and hour.

iii) The new data can be identified by the composite key of Building ID and Meter ID

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
building	X..address	BuildingID	meternum	BuildingID_	type	date	year	month	day	Day of We	hour	Base_hour	Weekday	Holiday	Consumpti	area_floor	Norm_Cor	Base_hour	Base_Hou	Temperatu	Dew_Point	Humidity	Se
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	4	TRUE	0	0	64	7380	0.008672	0.00831	High	-12.4462	-8.64516	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	19	FALSE	0	0	82	7380	0.011111	0.00831	High	28.4	28.4	100	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	5	FALSE	0	0	65	7380	0.008808	0.00831	High	-11.8923	-8.76129	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	20	FALSE	0	0	68	7380	0.009214	0.00831	High	28.4	26.6	93	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	6	FALSE	0	0	70	7380	0.009485	0.00831	High	-11.3385	-8.87742	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	7	FALSE	0	0	78	7380	0.010569	0.00831	High	-11.0615	-8.93548	77	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	21	FALSE	0	0	66	7380	0.008943	0.00831	High	28.4	24.8	86	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	8	FALSE	0	0	87	7380	0.011789	0.00831	High	-10.5077	-9.05161	77	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	22	TRUE	0	0	64	7380	0.008672	0.00831	High	26.6	24.8	93	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	9	FALSE	0	0	113	7380	0.015312	0.00831	High	-9.95385	-9.16774	77	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	23	TRUE	0	0	63	7380	0.008537	0.00831	High	26.6	23	86	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	10	FALSE	0	0	124	7380	0.016802	0.00831	High	-9.4	-9.28387	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	27	0	0	TRUE	0	0	63	7380	0.008537	0.00831	High	26.6	23	86	
Building 28	Runebergi	28096	1	28096_1	elect	1/5/2013	2013	1	5	6	1	TRUE	0	0	56	7380	0.007588	0.00831	Low	32	28.4	87	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	11	FALSE	0	0	106	7380	0.014363	0.00831	High	-4	-9.4	78	

Part2: Modelling tasks

Feature Selection:

Feature selection is done by Stepwise regression. The selected features are:-

a) Base hour flag

b) Weekday

c) Holiday

d) month

e) Temperature

f) Dew Point

Feature Transformation:

i) The columns have been normalized and some columns like Base_hour_Flag were converted into 0s and 1s for better performance.

ii) For Neural Network and KNN, the categorical variables were converted into dummy variables.

Steps to run:

i) Divide the data according to their type. i.e. Electricity and Heating, because they both have different correlation with features.

ii) For each type run the below steps separately

iii) Remove unwanted variables:

```
#Remove unwanted variables
ab$X..address.y<-NULL
ab$area_floor._m.sqr.y<-NULL
ab$BuildingID <-NULL
ab$building<-NULL
ab$meternumb<-NULL
ab$airport_code<-NULL
#ab$type<- NULL
ab$date<- NULL
ab$year<-NULL
ab$Base_hour_usage <- NULL
ab$Consumption <- NULL
ab$Base_Hour_Class <- NULL
ab$VisibilityMPH <- NULL ## more than 50% data is negative
ab$Gust_SpeedMPH<-NULL    #601105 values = '-'
ab$PrecipitationIn<-NULL  #614116 values N/A
ab$Wind_Direction <- NULL # wind dir deg is numreical for wind_direction
ab$Events <- NULL         # 350626 values empty
```

iv) For all the 78 buildings+meterID, run the below steps

v) Divide the data into Training set and Testing set. After carefully running a few combinations, we decided to take 75%-25%.

vi) Run Regression, KNN, Random Forest and Neural Network models and compute the evaluation matrices and store it in a .csv format.

KNN:

i) KNN has two types on inputs, the features and the value of k. The value of k is fed to the model using a JSON config file.

```
{
    "nearN":"8"
}
```

ii) The value of k is optimized to 8 after trial and error for best results keeping in mind the computational cost, since KNN takes up a lot of space to run.

```
#Model
knnModel<- knn(train= train_features_for_train_knn, test= test_features_for_test_knn, cl=train_features_target_knn,
```

iii) The model is run for Prediction and Classification.

iv) Performance Evaluation for Prediction of all the 78 models: -

A1	RMSE	MAE	MAPE	selected_	BuildingI	algorithm
1	0.00908	0.005921	NA	Weekday	28096_1	KNN
2	0.015995	0.009465	NA	Weekday	28108_3	KNN
3	0.007907	0.005209	Inf	Weekday	28161_1	KNN
4	0.005344	0.003309	NA	Weekday	28162_1	KNN
5	0.009499	0.004068	NA	Weekday	28168_1	KNN
6	0.004726	0.00308	NA	Weekday	30602_1	KNN
7	0.055921	0.031964	NA	Weekday	5198_1	KNN
8	0.009475	0.005836	NA	Weekday	5199_1	KNN
9	0.003796	0.00184	NA	Weekday	5286_1	KNN
10	0.003932	0.00244	NA	Weekday	5290_1	KNN
11	0.000467	0.000111	NA	Weekday	5304_1	KNN
12	0.004358	0.002868	NA	Weekday	5304_3	KNN
13	0.00764	0.005223	NA	Weekday	5306_1	KNN
14	0.003103	0.001871	NA	Weekday	5308_1	KNN
15	0.002953	0.001935	NA	Weekday	5310_1	KNN
16	0.004508	0.002762	NA	Weekday	5311_1	KNN
17	0.00855	0.005614	NA	Weekday	5313_1	KNN
18	0.009935	0.006211	NA	Weekday	5314_1	KNN
19	0.007886	0.003159	NA	Weekday	5316_1	KNN
20	0.007638	0.003026	NA	Weekday	5317_1	KNN
21	0.002771	0.001813	NA	Weekday	5318_1	KNN
22	0.001892	0.001295	NA	Weekday	5322_1	KNN
23	0.001145	0.000785	NA	Weekday	5323_1	KNN

v) Performance Evaluation for Classification of all the 78 models: -

	High	Low	Model	BuildingI	algorithm
High	1433	163	Weekday	28096_1	KNN
Low	158	298	Weekday	28096_1	KNN
High1	1302	210	Weekday	28108_3	KNN
Low1	240	300	Weekday	28108_3	KNN
High2	1155	280	Weekday	28161_1	KNN
Low2	199	418	Weekday	28161_1	KNN
High3	1213	252	Weekday	28162_1	KNN
Low3	200	387	Weekday	28162_1	KNN
High4	945	308	Weekday	28168_1	KNN
Low4	298	501	Weekday	28168_1	KNN
High5	1308	171	Weekday	30602_1	KNN
Low5	306	267	Weekday	30602_1	KNN
High6	648	390	Weekday	5198_1	KNN
Low6	369	645	Weekday	5198_1	KNN
High7	1569	110	Weekday	5199_1	KNN
Low7	188	185	Weekday	5199_1	KNN
High8	714	440	Weekday	5286_1	KNN
Low8	478	420	Weekday	5286_1	KNN
High9	1283	212	Weekday	5290_1	KNN
Low9	277	280	Weekday	5290_1	KNN
High10	3	50	Weekday	5304_1	KNN

Random Forest:

i) The value of no of trees is fed to the model using a JSON config file.

config_RF.json
Modified today at 6:33 PM

```
{
    "ntree":"100"
}
```

ii) The value of ntree is optimized to 100 after trial and error for best results keeping in mind the computational cost. The default value of ntree = 500 was taking a long training time.

```
rfm <- randomForest(train_features_target_rf ~ .,data=train_features_for_train_rf,ntree = 500)
```

iii) The model is run for Prediction and Classification.

iv) Performance Evaluation for Prediction of all the 78 models: -

RMSE	MAE	MAPE	model	BuildingI	algorithm
0.006175	0.004325	Inf	train$Bas	28096_1	Random Forest
0.010341	0.006881	Inf	train$Bas	28108_3	Random Forest
0.005108	0.003618	Inf	train$Bas	28161_1	Random Forest
0.003827	0.002589	Inf	train$Bas	28162_1	Random Forest
0.007082	0.004174	Inf	train$Bas	28168_1	Random Forest
0.002887	0.001933	Inf	train$Bas	30602_1	Random Forest
0.032596	0.020316	Inf	train$Bas	5198_1	Random Forest
0.005785	0.003929	Inf	train$Bas	5199_1	Random Forest
0.002952	0.001619	Inf	train$Bas	5286_1	Random Forest
0	0	NA	train$Bas	5288_1	Random Forest
0.002643	0.001844	Inf	train$Bas	5290_1	Random Forest
0	0	NA	train$Bas	5290_3	Random Forest
0	0	NA	train$Bas	5290_8	Random Forest
0.000211	4.70E-05	Inf	train$Bas	5304_1	Random Forest
0.002675	0.001899	Inf	train$Bas	5304_3	Random Forest
0.004813	0.00342	Inf	train$Bas	5306_1	Random Forest
0.002024	0.00136	Inf	train$Bas	5308_1	Random Forest
0.001997	0.001375	Inf	train$Bas	5310_1	Random Forest
0.002975	0.001953	Inf	train$Bas	5311_1	Random Forest
0.005614	0.003751	Inf	train$Bas	5313_1	Random Forest
0.006539	0.00452	Inf	train$Bas	5314_1	Random Forest

v) Performance Evaluation for Classification of all the 78 models: -

	High	Low	Model	BuildingI	algorithm
High	4203	603	train$Bas	28096_1	Random Forest
Low	285	1065	train$Bas	28096_1	Random Forest
High1	4181	382	train$Bas	28108_3	Random Forest
Low1	627	966	train$Bas	28108_3	Random Forest
High2	3680	753	train$Bas	28161_1	Random Forest
Low2	359	1364	train$Bas	28161_1	Random Forest
High3	3796	571	train$Bas	28162_1	Random Forest
Low3	466	1323	train$Bas	28162_1	Random Forest
High4	3317	479	train$Bas	28168_1	Random Forest
Low4	565	1795	train$Bas	28168_1	Random Forest
High5	4297	276	train$Bas	30602_1	Random Forest
Low5	815	768	train$Bas	30602_1	Random Forest
High6	2131	955	train$Bas	5198_1	Random Forest
Low6	862	2208	train$Bas	5198_1	Random Forest
High7	4723	350	train$Bas	5199_1	Random Forest
Low7	360	723	train$Bas	5199_1	Random Forest
High8	2378	932	train$Bas	5286_1	Random Forest
Low8	1695	1151	train$Bas	5286_1	Random Forest
High9	4022	442	train$Bas	5290_1	Random Forest
Low9	691	1001	train$Bas	5290_1	Random Forest
High10	127	43	train$Bas	5304_1	Random Forest

Neural Network:

```
#Model (building wise)
neuralnet <- neuralnet(predict_y ~ train$X28096_4.Base_hour_Flag +
                         train$X28096_4.Weekday+train$X28096_4.Holiday
                       + train$X28096_4.TemperatureF + as.numeric(train$'model1$X28096_4.month5') +
                       + train$X28096_4.Dew_PointF + as.numeric(train$'model1$X28096_4.month6') +
                         train$'model1$X28096_4.month7' + train$'model1$X28096_4.month8',
                       data=train, hidden=0)
```

Performance Evaluation for Prediction of all the 78 models: -

A1	RMSE	MAE	MAPE	selected_	BuildingI	algorithm
1	0.00908	0.005921	NA	Weekday	28096_1	KNN
2	0.015995	0.009465	NA	Weekday	28108_3	KNN
3	0.007907	0.005209	Inf	Weekday	28161_1	KNN
4	0.005344	0.003309	NA	Weekday	28162_1	KNN
5	0.009499	0.004068	NA	Weekday	28168_1	KNN
6	0.004726	0.00308	NA	Weekday	30602_1	KNN
7	0.055921	0.031964	NA	Weekday	5198_1	KNN
8	0.009475	0.005836	NA	Weekday	5199_1	KNN
9	0.003796	0.00184	NA	Weekday	5286_1	KNN
10	0.003932	0.00244	NA	Weekday	5290_1	KNN
11	0.000467	0.000111	NA	Weekday	5304_1	KNN
12	0.004358	0.002868	NA	Weekday	5304_3	KNN
13	0.00764	0.005223	NA	Weekday	5306_1	KNN
14	0.003103	0.001871	NA	Weekday	5308_1	KNN
15	0.002953	0.001935	NA	Weekday	5310_1	KNN
16	0.004508	0.002762	NA	Weekday	5311_1	KNN
17	0.00855	0.005614	NA	Weekday	5313_1	KNN
18	0.009935	0.006211	NA	Weekday	5314_1	KNN
19	0.007886	0.003159	NA	Weekday	5316_1	KNN
20	0.007638	0.003026	NA	Weekday	5317_1	KNN
21	0.002771	0.001813	NA	Weekday	5318_1	KNN
22	0.001892	0.001295	NA	Weekday	5322_1	KNN
23	0.001145	0.000785	NA	Weekday	5323_1	KNN

Performance Evaluation for Classification of all the 78 models: -

	High	Low	Model	BuildingI	algorithm
High	1433	163	Weekday	28096_1	KNN
Low	158	298	Weekday	28096_1	KNN
High1	1302	210	Weekday	28108_3	KNN
Low1	240	300	Weekday	28108_3	KNN
High2	1155	280	Weekday	28161_1	KNN
Low2	199	418	Weekday	28161_1	KNN
High3	1213	252	Weekday	28162_1	KNN
Low3	200	387	Weekday	28162_1	KNN
High4	945	308	Weekday	28168_1	KNN
Low4	298	501	Weekday	28168_1	KNN
High5	1308	171	Weekday	30602_1	KNN
Low5	306	267	Weekday	30602_1	KNN
High6	648	390	Weekday	5198_1	KNN
Low6	369	645	Weekday	5198_1	KNN
High7	1569	110	Weekday	5199_1	KNN
Low7	188	185	Weekday	5199_1	KNN
High8	714	440	Weekday	5286_1	KNN
Low8	478	420	Weekday	5286_1	KNN
High9	1283	212	Weekday	5290_1	KNN
Low9	277	280	Weekday	5290_1	KNN
High10	3	50	Weekday	5304_1	KNN

We decided not to take any hidden layer as the accuracy was fine and putting hidden layers was causing computational problems and it was difficult to run all 78 models with hidden layers.

Part1: Data Ingestion and Wrangling

Step 1: - Extract Features

i) Columns like Week of day, Month of year, Weekday/Weekend were derived from the date field in the raw data.

ii) To get the list of holidays:

d) API call to dateandtime.com with Date and country as the parameters.
e) To get the country, another API call to google had to be called giving the address of the building as the parameter.
f) To get the location, gdata library was used which internally calls google API to get the results.

```
#Retrieving List of Holidays
url_time_api <- "http://www.timeanddate.com/calendar/custom.html"
time_api <- read_html(url_time_api)

country_form <- xml_find_all(time_api,".//form")
country <- xml_find_all(country_form,".//option")

#index_of_country

ct_id =1
index_of_country =0
for(cntry in xml_text(country)){
  if(cntry==country_of_building){
    index_of_country  = ct_id
    break
  }
  ct_id = ct_id+1
}
if(index_of_country!=0){
  option_index <- toString(country[index_of_country])
  country_code_for_Holiday = substr(option_index,16,17)
}

yr = "2013"
params <- paste(paste("year",yr,sep="="),paste("country",country_code_for_Holiday,sep="=")
                ,paste("holm","1",sep="="),paste("hol","9",sep="="),paste("df","1",sep="="),sep="&")

#Building url to get Holiday Dynamically
url_time_api_for_country <- paste(url_time_api,params,sep = "?")

#getting List of Holidays
holiday_api <- GET(url_time_api_for_country,add_headers("Accept-Language"="en-US"))
holiday_html <- read_html(holiday_api$content)

holidays_table <- xml_find_all(holiday_html,".//table")
holidays_node <- xml_find_all(holidays_table,".//span")
```

iii) Normalize the consumption value by dividing it with the total floor area

iv) To get the weather data:

g) Get the latitude and longitude for the address
h) Pass the coordinates to the url to get the link
i) Read the XML to extract the nearest airport
j) From the nearest airport get the airport code

k) Get start date, end date from the raw data

l) Get weather data from library 'getWeatherData'

```
for (e in df1$X..address){
# Finding Latitude & longitude for the address
  lat_lon <- geocode(e, source = "google",sensor = TRUE)
  cordinates <- paste(lat_lon$lat,lat_lon$lon,sep=",")
#Passing the cordinates to the url
  url_loc <- paste(url_api,cordinates,sep="=")
#Reading the XML
  data <- read_xml(url_loc)
#Finding the nearest airport
  data_airport <- xml_find_all(data,".//airport")
  data_air_codes <- xml_find_all(data_airport,".//icao")
  codes <- xml_contents(data_air_codes)
  code <- toString(codes[1])
  code_list[i] <- code
  i = i+1
}
```

```
endDate = format(as.Date(tail(a$date,1), format="%Y%m%d"),"%Y-%m-%d")
endDate <- as.Date(endDate)
startDate <- as.Date(startDate)


for (c_air in unique(code_list)){
  d3 <- getWeatherForDate(c_air, start_date=startDate,
                          end_date = endDate,
                          opt_detailed = TRUE,
                          opt_all_columns = TRUE)
  d3$airport_code = c_air
  d3 <- select(d3,Time,TemperatureF,Dew_PointF,Humidity,Sea_Level_PressureIn,VisibilityMPH,
               Wind_Direction,Wind_SpeedMPH,Gust_SpeedMPH,PrecipitationIn,
               Events,Conditions,WindDirDegrees,airport_code)
  weatherData <- rbind(weatherData,d3)
}
```

Step 2: - Removing the NA and empty values

i) There were a lot of NA or empty values in weather data. They were handled case by case according to the domain knowledge:

a) Fields like Temperature, Dew_PointF, Humidity, Sea_Level_PressureIn, VisibilityMPH and Wind_SpeedMPH were imputed by using imputation by interpolating the values.

b) WindDirDegrees was filled out with random values as they were changing randomly with no defined pattern.

c) Conditions field was replaced with the last known value as it was observed that the value of Conditions does not change frequently hourly.

```
#Replacing NA values with linear interpolation
processedOp$TemperatureF <- na.approx(processedOp$TemperatureF,na.rm = FALSE)
processedOp$Dew_PointF <- na.approx(processedOp$Dew_PointF,na.rm = FALSE)
processedOp$Humidity <- na.approx(processedOp$Humidity,na.rm = FALSE)
processedOp$Sea_Level_PressureIn <- na.approx(processedOp$Sea_Level_PressureIn,na.rm = FALSE)
processedOp$VisibilityMPH <- na.approx(processedOp$VisibilityMPH,na.rm = FALSE)
processedOp$Wind_SpeedMPH <- na.approx(processedOp$Wind_SpeedMPH,na.rm = FALSE)


#Replacing NA values randomly
processedOp$WindDirDegrees[is.na(processedOp$WindDirDegrees)]<- sample(1:36,1)*10

#Replacing NA with last known non null value
processedOp$Conditions <-  na.locf(processedOp$Conditions,fromLast = TRUE,na.rm = FALSE)
```

Step 3: - Removing the outliers

i) There were a lot of outlier values in fields like Temperature, Dew_PointF, Humidity, Sea_Level_PressureIn, VisibilityMPH and Wind_SpeedMPH.

ii) Outliers which were beyond 1.5x IQR were replaced by NA and then later imputed along with other NA values.

```
#Remove the outliers using Boxplot
processedOp$TemperatureF[processedOp$TemperatureF %in% boxplot.stats(processedOp$TemperatureF)$out] <- NA
processedOp$Dew_PointF[processedOp$Dew_PointF %in% boxplot.stats(processedOp$Dew_PointF)$out] <- NA
processedOp$Humidity[processedOp$Humidity %in% boxplot.stats(processedOp$Humidity)$out] <- NA
processedOp$Sea_Level_PressureIn[processedOp$Sea_Level_PressureIn %in% boxplot.stats(processedOp$Sea_Level_Pr
processedOp$VisibilityMPH[processedOp$VisibilityMPH %in% boxplot.stats(processedOp$VisibilityMPH)$out] <- NA
processedOp$Wind_SpeedMPH[processedOp$Wind_SpeedMPH %in% boxplot.stats(processedOp$Wind_SpeedMPH)$out] <- NA
```

Step 4: - Merge the data

i) The address data and the consumption data are merged on the Building Name.

ii) The output of step (i) is merged with the weather data according to the nearest airport code, date and hour.

iii) The new data can be identified by the composite key of Building ID and Meter ID

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
building	X..address	BuildingID	meternum	BuildingID_	type	date	year	month	day	Day of We	hour	Base_hour	Weekday	Holiday	Consumpti	area_floor	Norm_Cor	Base_hour	Base_Hou	Temperatu	Dew_Point	Humidity	Se
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	4	TRUE	0	0	64	7380	0.008672	0.00831	High	-12.4462	-8.64516	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	19	FALSE	0	0	82	7380	0.011111	0.00831	High	28.4	28.4	100	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	5	FALSE	0	0	65	7380	0.008808	0.00831	High	-11.8923	-8.76129	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	20	FALSE	0	0	68	7380	0.009214	0.00831	High	28.4	26.6	93	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	6	FALSE	0	0	70	7380	0.009485	0.00831	High	-11.3385	-8.87742	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	7	FALSE	0	0	78	7380	0.010569	0.00831	High	-11.0615	-8.93548	77	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	21	FALSE	0	0	66	7380	0.008943	0.00831	High	28.4	24.8	86	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	8	FALSE	0	0	87	7380	0.011789	0.00831	High	-10.5077	-9.05161	77	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	22	TRUE	0	0	64	7380	0.008672	0.00831	High	26.6	24.8	93	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	9	FALSE	0	0	113	7380	0.015312	0.00831	High	-9.95385	-9.16774	77	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	26	6	23	TRUE	0	0	63	7380	0.008537	0.00831	High	26.6	23	86	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	10	FALSE	0	0	124	7380	0.016802	0.00831	High	-9.4	-9.28387	84	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	27	0	0	TRUE	0	0	63	7380	0.008537	0.00831	High	26.6	23	86	
Building 28	Runebergi	28096	1	28096_1	elect	1/5/2013	2013	1	5	6	1	TRUE	0	0	56	7380	0.007588	0.00831	Low	32	28.4	87	
Building 28	Runebergi	28096	1	28096_1	elect	########	2013	1	19	6	11	FALSE	0	0	106	7380	0.014363	0.00831	High	-4	-9.4	78	

Part2: Modelling tasks

Prediction

Feature Selection:

Feature selection is done by Stepwise regression. The selected features are:-

a) Base hour flag

b) Weekday

c) Holiday

d) month

e) Temperature

f) Dew Point

Feature Transformation:

i) The columns have been normalized and some columns like Base_hour_Flag were converted into 0s and 1s for better performance.

ii) For Neural Network and KNN, the categorical variables were converted into dummy variables.

Steps to run:

i) Divide the data according to their type. i.e. Electricity and Heating

ii) For each type run the below steps separately

iii) Remove unwanted variables:

```
#Remove unwanted variables
ab$X..address.y<-NULL
ab$area_floor._m.sqr.y<-NULL
ab$BuildingID <-NULL
ab$building<-NULL
ab$meternumb<-NULL
ab$airport_code<-NULL
#ab$type<- NULL
ab$date<- NULL
ab$year<-NULL
ab$Base_hour_usage <- NULL
ab$Consumption <- NULL
ab$Base_Hour_Class <- NULL
ab$VisibilityMPH <- NULL ## more than 50% data is negative
ab$Gust_SpeedMPH<-NULL    #601105 values = '-'
ab$PrecipitationIn<-NULL  #614116 values N/A
ab$Wind_Direction <- NULL # wind dir deg is numreical for wind_direction
ab$Events <- NULL         # 350626 values empty
```

iv) Divide the data into Training set and Testing set. After carefully running a few combinations, we decided to take 75-25.

Logistic Regression for Classification

Feature Engineering

1. Feature Transformation:

Base_hour_flag was converted to 0 & 1. This was done to create dummies.

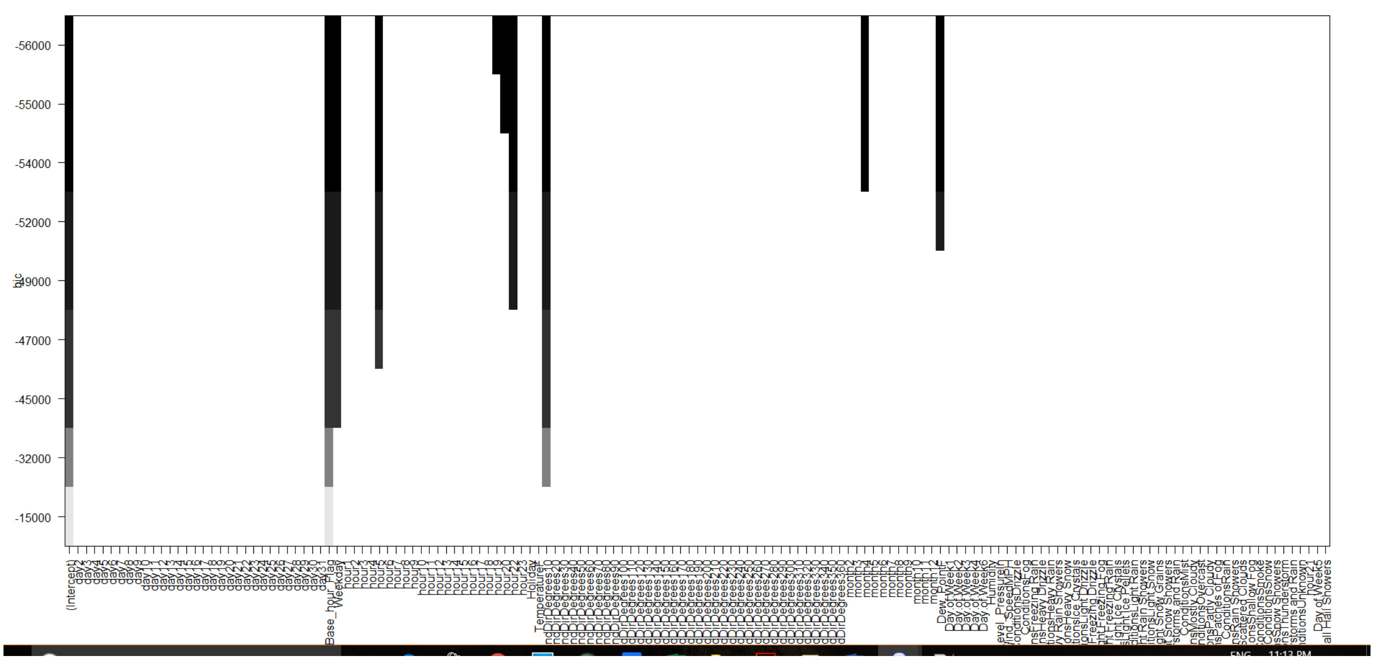
2. Feature Selection:

i) The 1st step was to drop columns which were not required in the model & also had data issues. So columns like building name, address, airport code, type etc. were eliminated since they were not required in the model while the columns like VisibilityMPH, Gust_SpeedMPH etc were removed since they had data issues. Data issues in the form of more than 50% missing data or NA values.

ii) Converting categorical variables into factors to feed it to the model. This was done so that the model could identify them as categories and not any other type.

iii) After splitting the data into train & test, initially a model was run with all the features fed to the model.

iv) Used Forward Selection for Variable Selection to determine the p values and the coefficients. We found out that the most influential features were Base_hour_Flag ,Weekday ,Holiday, TemperatureF, Dew_PointF.

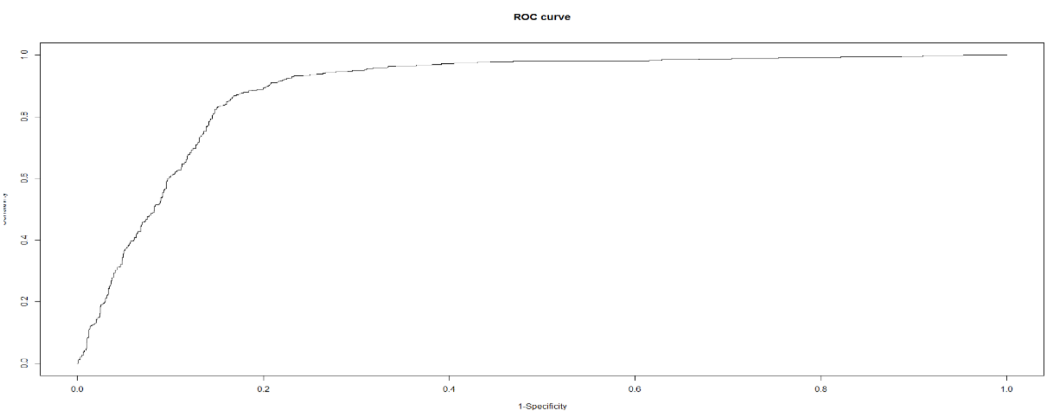


Feature selection using forward selection

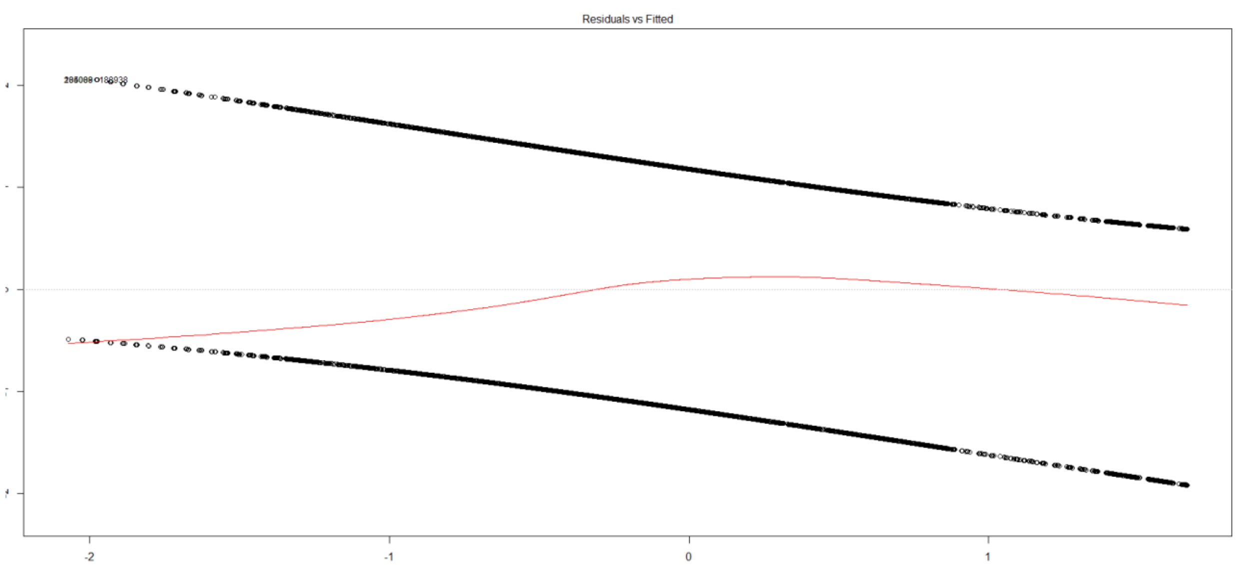
Confusion Matrix:-

	High	Low	Model
High	58170	28274	Base_hour_Flag + Weekday + Holiday + Temperature...
Low	46253	22757	Base_hour_Flag + Weekday + Holiday + Temperature...

ROC Curve :-



Residual plot:



Output file with Outlier_days tags

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	month	day	outlier_day	BuildingID_M	Day.of.Week	hour	Base_hour_Fl	Weekday	Holiday	Norm_Consu	Base_Hour_C	TemperatureF	Dew_PointF	Humidity	Sea_Level_Pr	Wind_SpeedN	Conditions	WindDirDegre
2	1	1	FALSE	28096_1	2	23	0	1	1	0.00745257	High	35.6	35.6	100	29.42	11.5	Fog	210
3	1	1	FALSE	28096_1	2	10	1	1	1	0.00704607	Low	37.4	37.4	100	29.3	11.5	Drizzle	180
4	1	1	FALSE	28096_1	2	8	1	1	1	0.00718157	Low	37.4	37.4	100	29.3	11.5	Drizzle	180
5	1	1	FALSE	28096_1	2	15	1	1	1	0.00731707	High	37.4	37.4	100	29.33	13.8	Drizzle	200
6	1	1	FALSE	28096_1	2	18	1	1	1	0.00758808	High	37.4	37.4	100	29.36	11.5	Overcast	200
7	1	1	FALSE	28096_1	2	16	1	1	1	0.00731707	High	37.4	37.4	100	29.36	11.5	Drizzle	200
8	1	1	FALSE	28096_1	2	3	0	1	1	0.00718157	Low	35.6	33.8	93	29.36	10.4	Light Rain	160
9	1	2	FALSE	28096_1	3	14	1	1	0	0.02764228	High	35.6	33.8	93	29.56	8.1	Mostly Cloudy	220
10	1	2	FALSE	28096_1	3	16	1	1	0	0.02113821	High	35.6	33.8	93	29.59	6.9	Mostly Cloudy	210
11	1	2	FALSE	28096_1	3	4	0	1	0	0.00745257	Low	35.6	35.6	100	29.47	6.9	Overcast	200
12	1	2	FALSE	28096_1	3	15	1	1	0	0.02479675	High	35.6	33.8	93	29.59	5.8	Mostly Cloudy	210
13	1	2	FALSE	28096_1	3	0	0	1	0	0.00745257	Low	35.6	35.6	100	29.44	9.2	Mist	200
14	1	3	FALSE	28096_1	4	20	1	1	0	0.01111111	High	32	30.2	93	29.56	2.3	Mostly Cloudy	120
15	1	3	FALSE	28096_1	4	7	1	1	0	0.02235772	High	32	32	100	29.59	4.6	Light Rain	260

Clustering

Problem: -

Using the building features, cluster the buildings using K-means and Hierarchical clustering.

Using Bend graphs, choose the optimal number of clusters. Discuss your cluster features in a report

Solution: -

Step 1: - Feature Engineering

a) Since the data is already clean, we have to decide what features we have to consider to cluster the buildings.

b) Since weather is not a feature of building, we drop all those features which are related to the weather and date.

c) Clustering algorithms does not take categorical variables as input. So we drop all those features which are not continuous.

d) We are now left with the following features:

i) Area
ii) Latitude
iii) Longitude
iv) Consumption – Electricity
v) Consumption - Heating

Step 2: - Normalize the inputs

i) We first tried clustering without normalization and realized that clustering was not efficient because the features were not in the same scale.

```
> unnormalized <- kmeans(selected_features[2:6],4,nstart=10)
> unnormalized$cluster
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
 3  4  3  4  4  1  3  3  4  3  3  4  4  3  3  3  1  3  4  3  4  3  3  3  1  1  4  4  2  3  4  4
```

ii) The clusters shown above shows that the function is dividing the data into several clusters even though it is specified that we need 4 clusters

iii) Normalization is done using the following function:

```
normalize<-function(x)(return((x-min(x))/(max(x)-min(x))))
```

iv) All the data now lies between 0 to1 and now we are ready to feed it into the clustering function.

area_floor._m.sqr	latitudes	longitudes	Dist_Heating	elect
0.000000000	1.00000000	0.0000000	0.00000000	0.048126965
0.190254433	0.37465775	0.8158854	0.20629348	0.156464457
0.121031313	0.34739595	0.8124752	0.00000000	0.191944164
0.294210386	0.38794060	0.8237210	1.00000000	1.000000000
0.139658978	0.27849052	0.8208910	0.28470367	0.041888283
0.372480536	0.29761006	0.8060873	0.24958443	0.168697136
0.078779500	0.21654186	0.8254039	0.05484713	0.029976806
0.076596570	0.20574810	0.8269595	0.12050888	0.092459325
0.166314972	0.22921720	0.8269104	0.23223824	0.145253503
0.022047588	0.23676461	0.8243624	0.03271915	0.027639282
0.045962793	0.18193727	0.8272852	0.00000000	0.011869151
0.281040045	0.35988989	0.8226838	0.00000000	0.668957850
0.173712678	0.33082667	0.8162436	0.36802367	0.528572677
0.085619346	0.19616090	0.8142014	0.10276593	0.109455620
0.039971864	0.28448752	0.8253073	0.00000000	0.071002155
0.070920954	0.19711646	0.8262229	0.05614769	0.061755581
0.348298528	0.25588415	0.8145947	0.58069868	0.677741169
0.088772466	0.37779442	0.8162665	0.07615400	0.133243785

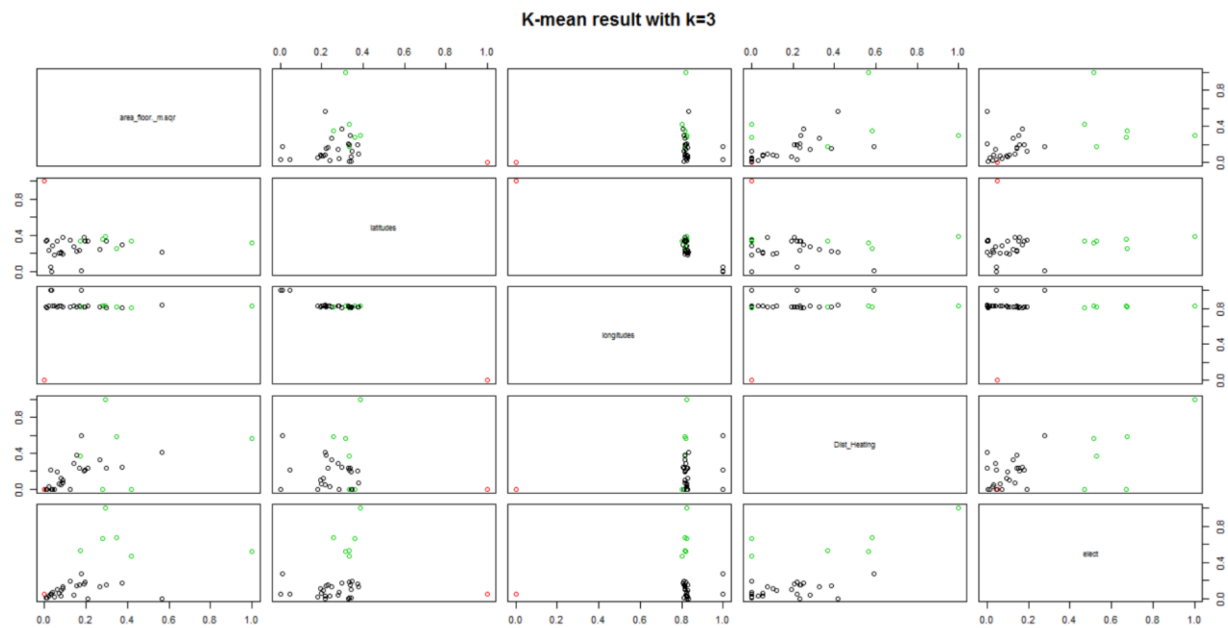
Step 3: - K-means clustering

i) In k means clustering, we have two types of inputs, the value of k and the input features.

ii) For k=3: -

```
> km.out
K-means clustering with 3 clusters of sizes 25, 1, 6

Cluster means:
  area_floor._m.sqr latitudes longitudes Dist_Heating      elect
1         0.1403507 0.2472563  0.8406094    0.1696772 0.09258102
2         0.0000000 1.0000000  0.0000000    0.0000000 0.04812696
3         0.4193691 0.3307493  0.8167882    0.4187873 0.64338942

Clustering vector:
 [1] 2 1 1 3 1 1 1 1 1 1 1 3 3 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 3 1 1 1

Within cluster sum of squares by cluster:
[1] 1.470528 0.000000 1.374788
 (between_SS / total_SS =  55.1 %)
```

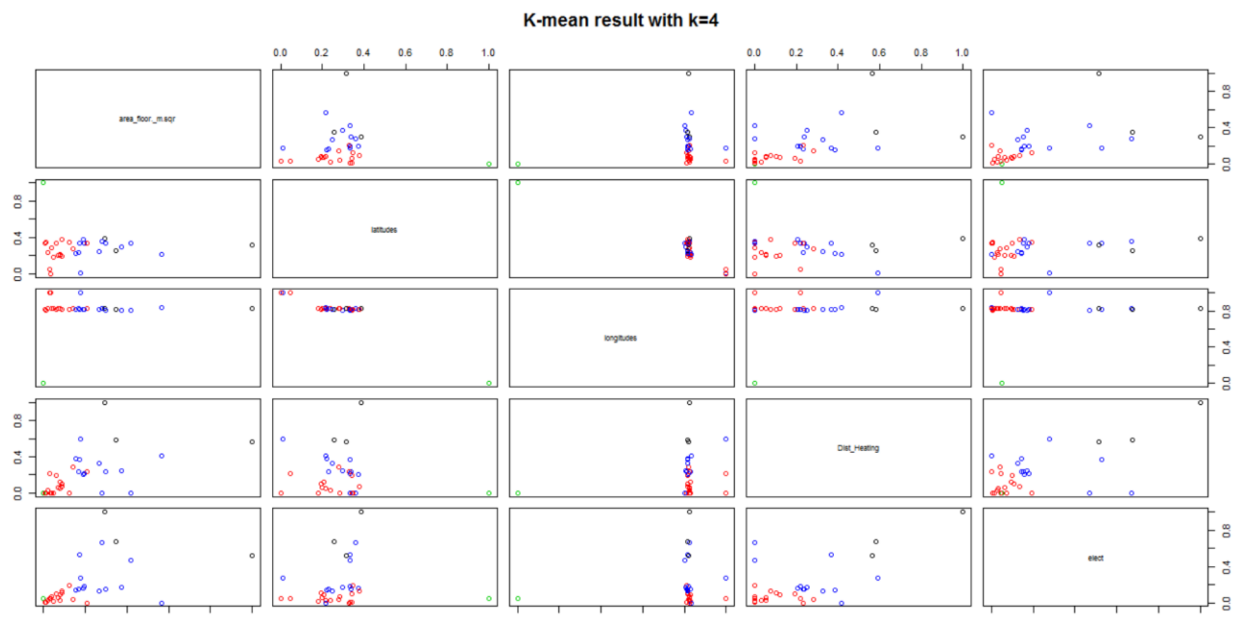
From the figure we can see that the between_ss/total_ss is around 55.1%, which might not be the best fit. We want this value to be closer to 100%. To get better results, we will have to play around with the value of k.



iii) For k=4: -

```
> km.out
K-means clustering with 4 clusters of sizes 3, 16, 1, 12

Cluster means:
  area_floor._m.sqr latitudes longitudes Dist_Heating      elect
1        0.54750297 0.3198592  0.8197188   0.71490013 0.73100773
2        0.07009023 0.2448561  0.8422900   0.08600596 0.06048617
3        0.00000000 1.0000000  0.0000000   0.00000000 0.04812696
4        0.27175249 0.2740524  0.8316808   0.26948816 0.25117168

Clustering vector:
 [1] 3 4 2 1 2 4 2 2 4 2 2 4 4 2 2 2 1 2 4 2 4 2 2 2 4 4 2 4 1 2 4 4

Within cluster sum of squares by cluster:
[1] 0.5611754 0.4577262 0.0000000 1.0566698
 (between_SS / total_SS =  67.2 %)
```

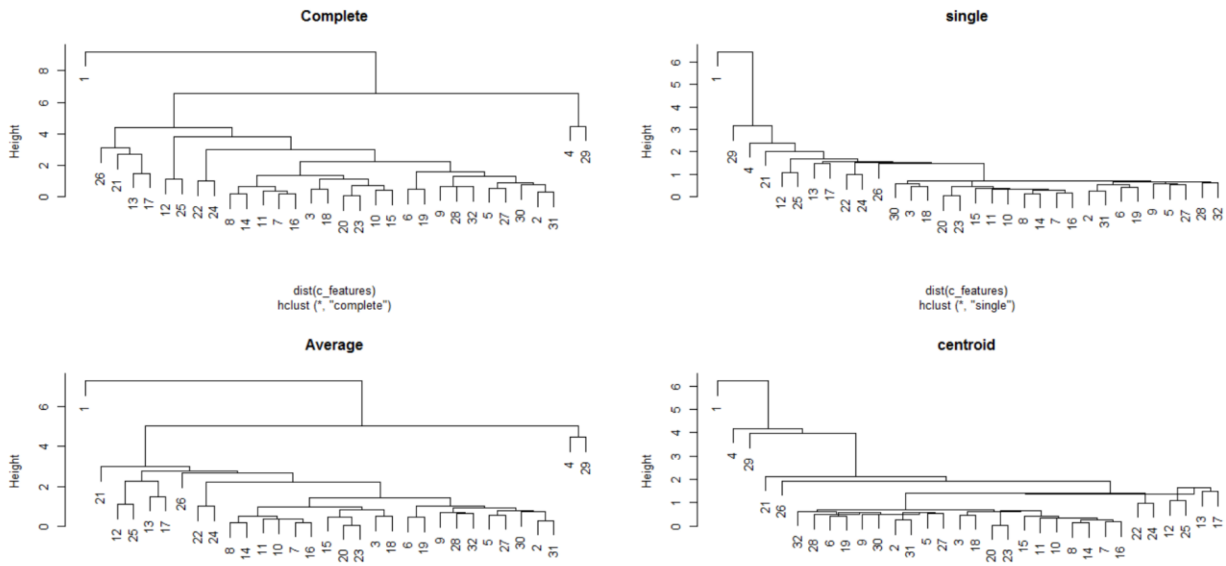
We can see that the model has become better after we changed the value of k from 3 to 4. To find the optimum value of k, we will use elbow method and bend graph done later in the report.



v) Value of iteration (nstart): Since the data points are less, the number of iterations required for convergence wont vary with the value of nstart. We have taken the value as 10. This will save computation losses.

Step 4: - Hierarchal Clustering

i) In Hierarchal clustering, we don't need to care about the value of k. We create a tree and then simply prune the tree accordingly.
ii) We have used 4 types of Linkage method:
a) Complete
b) Single
c) Average
d) Centroid
iii) The result plotted is:



iv) After pruning the tree according to the number of clusters, we get the following results:

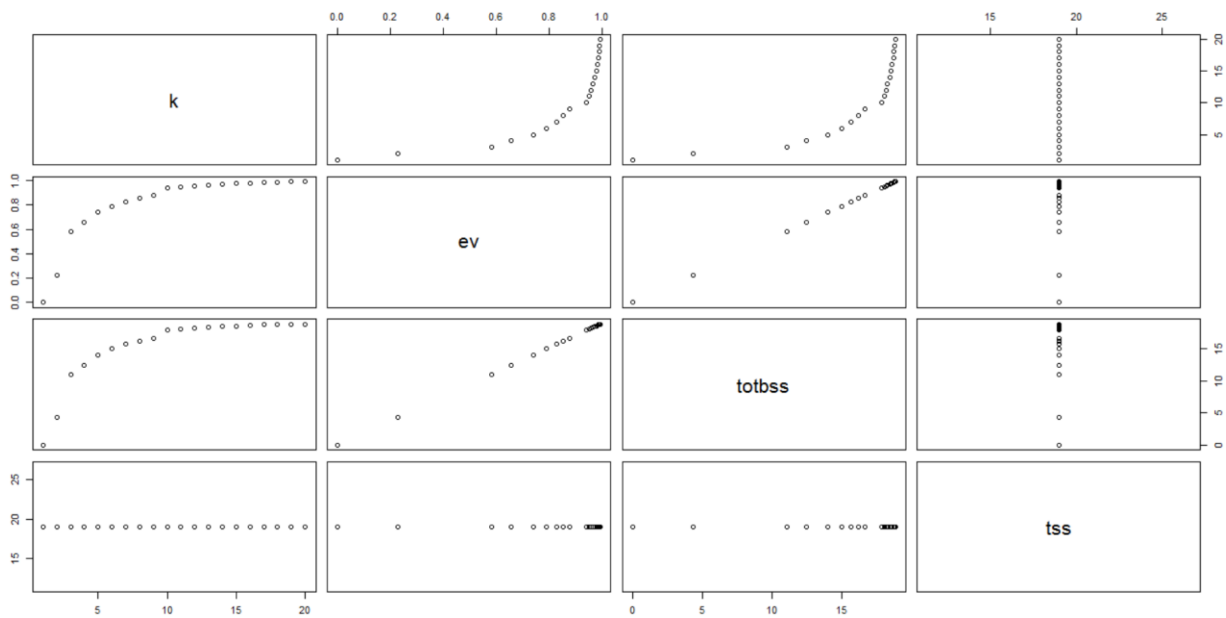
```
> cutree(hc.complete,4)
 [1] 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2
> cutree(hc.single,4)
 [1] 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2
> cutree(hc.average,4)
 [1] 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2
> cutree(hc.centroid,4)
 [1] 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2
```

Step 5: - Using Elbow method to find out the optimum numbers of clusters

i) After creating the elbow object using 'manhattan distance' method, we get the following results:

```
> print(elbow.obj)
$k
[1] 11

$ev
[1] 0.9500419

$inc.thres
[1] 0.01

$ev.thres
[1] 0.95
```

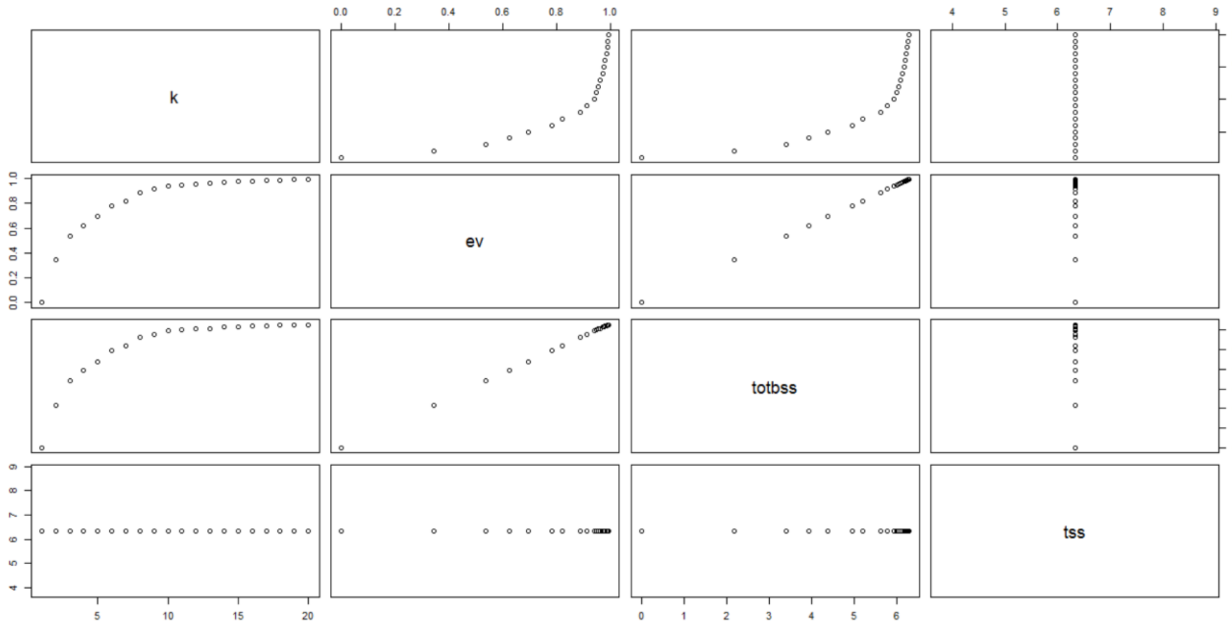
This shows that the optimum value of k should be 11. We also plot the graphs between all these values



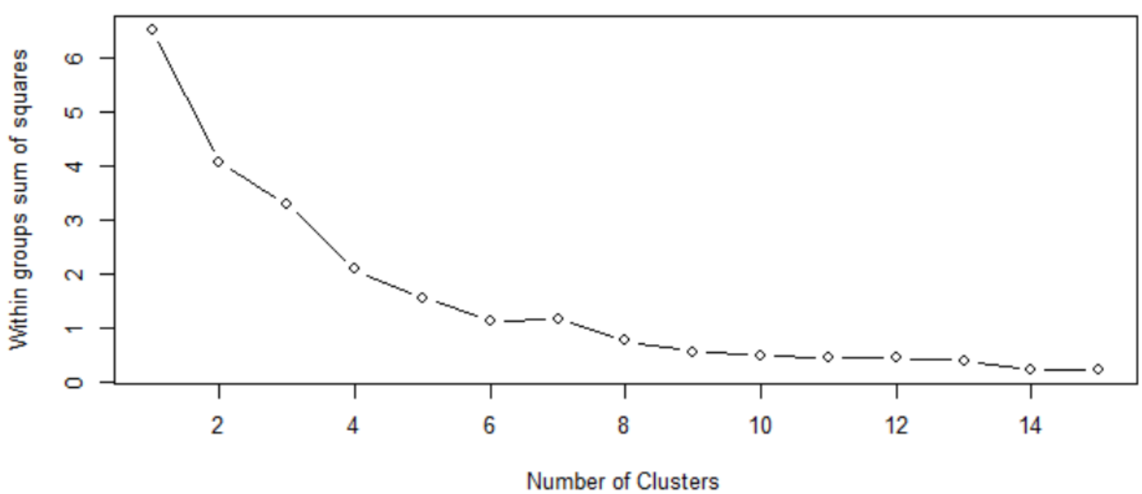
ii) After creating the elbow object using ‘Euclidian distance’ method, we get the following results:

```
> print(elbow.obj)
$k
[1] 12

$ev
[1] 0.9545588

$inc.thres
[1] 0.01

$ev.thres
[1] 0.95
```

This shows that the optimum value of k should be 12. We also plot the graphs between all these values:



Step 6: - Using Bend graph to find out the optimum numbers of clusters

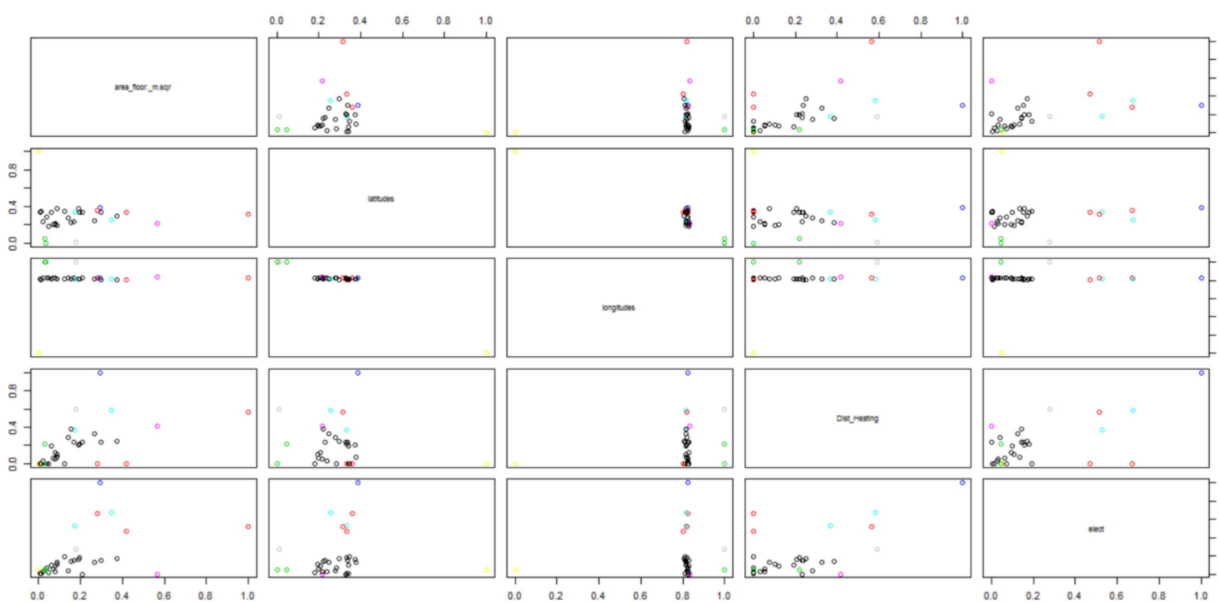


From the graph of within groups sum of squares and number of clusters, we can see that the value of k does not have much effect at a value after 10. So keeping in mind the computational cost and the efficiency of clustering, we choose k=10 as the optimum value for clustering

Step 7: - Results for k=10

```
> km.out
K-means clustering with 10 clusters of sizes 9, 2, 2, 1, 2, 1, 1, 1, 12, 1

Cluster means:
   area_floor._m.sqr  latitudes longitudes Dist_Heating      elect
1         0.22154848 0.29412768  0.8169949   0.26407275 0.12348705
2         0.34999636 0.34704579  0.8126647   0.00000000 0.56937032
3         0.03040336 0.02406185  0.9989071   0.10993639 0.04410732
4         0.29421039 0.38794060  0.8237210   1.00000000 1.00000000
5         0.26100560 0.29335541  0.8154191   0.47436118 0.60315692
6         0.56666424 0.21503786  0.8314054   0.41482239 0.00000000
7         0.00000000 1.00000000  0.0000000   0.00000000 0.04812696
8         0.17633219 0.01045415  0.9995711   0.59259048 0.27725165
9         0.05925239 0.27172025  0.8194576   0.05316577 0.06980631
10        1.00000000 0.31575295  0.8208407   0.56400169 0.51528202

Clustering vector:
 [1]  7  1  9  4  1  1  9  9  1  9  9  2  5  9  9  9  5  9  1  9  8  3  9  3  2  6  1  1 10  9  1  1

Within cluster sum of squares by cluster:
 [1] 0.12697267 0.02987601 0.02533942 0.00000000 0.05179061 0.00000000 0.00000000 0.00000000 0.15180588 0.00000000
 (between_SS / total_SS =  93.9 %)
```

We are getting a between_ss / total_ss equal to 93.9% which is pretty good. Let us plot the scatter plots and the table: -



	row.names	area_floor._m.sqr	latitudes	longitudes	Dist_Heating	elect	group
1	1	0.000000000	1.00000000	0.0000000	0.00000000	0.048126965	1
2	2	0.190254433	0.37465775	0.8158854	0.20629348	0.156464457	2
3	3	0.121031313	0.34739595	0.8124752	0.00000000	0.191944164	2
4	5	0.139658978	0.27849052	0.8208910	0.28470367	0.041888283	2
5	6	0.372480536	0.29761006	0.8060873	0.24958443	0.168697136	2
6	7	0.078779500	0.21654186	0.8254039	0.05484713	0.029976806	2
7	8	0.076596570	0.20574810	0.8269595	0.12050888	0.092459325	2
8	9	0.166314972	0.22921720	0.8269104	0.23223824	0.145253503	2
9	10	0.022047588	0.23676461	0.8243624	0.03271915	0.027639282	2
10	11	0.045962793	0.18193727	0.8272852	0.00000000	0.011869151	2
11	12	0.281040045	0.35988989	0.8226838	0.00000000	0.668957850	2
12	13	0.173712678	0.33082667	0.8162436	0.36802367	0.528572677	2
13	14	0.085619346	0.19616090	0.8142014	0.10276593	0.109455620	2
14	15	0.039971864	0.28448752	0.8253073	0.00000000	0.071002155	2
15	16	0.070920954	0.19711646	0.8262229	0.05614769	0.061755581	2
16	17	0.348298528	0.25588415	0.8145947	0.58069868	0.677741169	2
17	18	0.088772466	0.37779442	0.8162665	0.07615400	0.133243785	2
18	19	0.299231124	0.33568249	0.8100167	0.23932288	0.153253492	2
19	20	0.010744864	0.34226169	0.8094985	0.00000000	0.003142344	2
20	21	0.176332193	0.01045415	0.9995711	0.59259048	0.277251653	2
21	22	0.028547867	0.04812369	0.9978141	0.21987278	0.043809150	2
22	23	0.008707463	0.33453498	0.8106649	0.00000000	0.004959034	2
23	24	0.032258847	0.00000000	1.0000000	0.00000000	0.044405493	2
24	25	0.418952679	0.33420170	0.8026456	0.00000000	0.469782781	2
25	26	0.566664241	0.21503786	0.8314054	0.41482239	0.000000000	2
26	27	0.209949308	0.33044065	0.8244436	0.23352967	0.000000000	2
27	28	0.265783793	0.24574430	0.8160513	0.32649397	0.125887304	2
28	30	0.061873924	0.33989917	0.8148430	0.19484652	0.100228517	2
29	31	0.196366635	0.33365537	0.8188945	0.21995052	0.178074067	2
30	32	0.153896529	0.22165081	0.8137743	0.38453788	0.141865195	2
31	4	0.294210386	0.38794060	0.8237210	1.00000000	1.000000000	3
32	29	1.000000000	0.31575295	0.8208407	0.56400169	0.515282019	4

The results might look a bit unconvincing, this is due to the fact that the data points are very less.

For better visualization of the clusters, we have plotted the scatter plot color coded according to the clusters.