-

Advance Data Science

Boston Energy Forecasting

Mid Term

Lalit Jain, Sameer Goel, Lipsa Panda – Team 7

11-18-2016

1 CONTENTS

1 SUMMARY

The report summarizes the energy usage by the buildings owned by Vokia Inc. The purpose of the report is to document the Energy models and the subsequent analysis to understand and reduce energy usage to make the building energy efficient.

2 PART1: DATA INGESTION AND WRANGLING

2.1 THE DATA

- *R file available in Part 1/Part1_Ingestion.R*
- We have a data set with 1214184 records with corresponding building details. We can have an idea about the data and the data types of the variable used.

```
Console C:/Users/Lipsa/Desktop/BuidlingEnergy_Finland/
> head(rawdata)
  BuildingID        vac meternumb  type       date hour Consumption
1       5198 Building 1         1 elect 2013-01-01    0          25
2       5198 Building 1         1 elect 2013-01-01    1          25
3       5198 Building 1         1 elect 2013-01-01    2          26
4       5198 Building 1         1 elect 2013-01-01    3          25
5       5198 Building 1         1 elect 2013-01-01    4          25
6       5198 Building 1         1 elect 2013-01-01    5          25
> str(rawdata)
'data.frame':   594984 obs. of  7 variables:
 $ BuildingID : int  5198 5198 5198 5198 5198 5198 5198 5198 5198 5198 ...
 $ vac        : chr  "Building 1" "Building 1" "Building 1" "Building 1" ...
 $ meternumb  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ type       : Factor w/ 4 levels "Dist_Heating",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ date       : Date, format: "2013-01-01" "2013-01-01" "2013-01-01" ...
 $ hour       : num  0 1 2 3 4 5 6 7 8 9 ...
 $ Consumption: num  25 25 26 25 25 25 25 27 23 26 ...
>
```

- We found the raw data had 4 different types of sources of energy out of which only 2 were our matter of concern.

```
> unique(rawdata$type)
[1] elect        Dist_Heating
Levels: Dist_Heating elect React_Power Water
>
```

- Hence, we filtered the data set to fetch the energy consumption by Electricity and Heating. Once filtered, we read the data from dataset, R intuitively understands datatypes. After doing a filter with the required source types, the number records came down to 621816.

We need to check the structure of our data and then change the columns structure according to our requirements. str () function gives us the snapshot of datatypes of all the columns.

```
> str(rawdata)
'data.frame':   621816 obs. of  7 variables:
 $ BuildingID : int  5198 5198 5198 5198 5198 5198 5198 5198 5198 5198 ...
 $ vac        : Factor w/ 32 levels "                                ",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ meternumb  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ type       : Factor w/ 4 levels "Dist_Heating",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ date       : int  20130101 20130101 20130101 20130101 20130101 20130101 20130101 20130101 20130101
20130101 ...
 $ hour       : int  0 1 2 3 4 5 6 7 8 9 ...
 $ Consumption: int  25 25 26 25 25 25 25 27 23 26 ...
> 
```

- There were few values like Date, Hour, vac which needed to be converted to corresponding formats.

```
str(rawdata)

##convert to Date type and changing int columns to numeric
rawdata$date <- as.Date((strptime(rawdata$date,"%Y%m%d")),"%Y-%d-%m")
rawdata$hour <- as.numeric(rawdata$hour)
rawdata$Consumption <- as.numeric(rawdata$Consumption)
rawdata$vac <- as.character(rawdata$vac)



# View(table(rawdata$vac))
```

-

- After having an insight on the data, we analyzed there were few missing building records which we substituted with the corresponding building numbers.

```
# View(table(rawdata$vac))

###### Adding missing values by judgment
rawdata$vac[which(rawdata$BuildingID == 81909)] <- "Building 27"
rawdata$vac[which(rawdata$BuildingID == 82254)] <- "Building 9"
rawdata$vac[which(rawdata$BuildingID == 83427)] <- "Building 9"
rawdata$vac[which(rawdata$BuildingID == 84681)] <- "Building 9"
#####
```

- We added the energy consumption by taking the reading from all the meters for each building to have a confirmation that the each building with corresponding building id has a meter id reading on an hourly basis.

```
## Sum the consumption by taking  readings from all the meters
#for each type of each Building ID
require(dplyr)
rawdata.agg <- rawdata %>%  select(BuildingID,vac,meternumb,type,date,hour,Consumption)
%>% group_by(BuildingID,meternumb,vac,type,date,hour) %>%
  summarise(Consumption = sum(Consumption))
```

2.2 FEATURE ENHANCEMENT

- Now adding features like week of the day, Month of the year, Weekday or Weekend, Holiday and Base Hour Flag.

- Function to find the weekday for the data record:

```
##Function to find the Weekday
checkWeekday <- function(date){
  wday.r = if((wday(date) == 6 || wday(date) == 7)) 0 else 1
  return(wday.r)
}
```

- We used another function to calculate the base hour and set the values high for the duration 0,1,2,3,4,22,23.

```
baseHour <- function(hour){
  hour.r = if(hour %in% c(0,1,2,3,4,22,23)) 1 else 0
  return(hour.r)
}
```

- To work with date-times and time-spans: fast and user friendly parsing of date-time data, extraction and updating of components of a date-time (years, months, days, hours, minutes, and seconds), algebraic manipulation on date-time and time-span objects. The '**lubridate**' package has a consistent and memorable syntax that makes working with dates easy and fun. So, we used the following package.
- Using the above package, we derived a dataset with the above values using mutate function.

```
require(lubridate)
require(dplyr)

finlandEnergy.agg <- rawdata.agg %>% mutate(month = month(date),Day.of.Week = wday(date)) %>%
  mutate(weekday = sapply(date, function(x) checkWeekday(x))) %>%
  mutate(Base_hour_Flag = sapply(hour, function(x) baseHour(x)))
```

- After extracting the required features, we normalized the consumption with respect to the area_floor_m.sqr and made it Kwh/square meter.
- To get public holiday details, we scraped the timeanddate.com website. Code is as shown below. All the holidays are stored in holidaydates objects. We also defined a function which can take any date passed to it and it will return if the date is holiday or not.

```
country <- "finland"
year <- 2013
url <- paste("http://www.timeanddate.com/holidays/",country,"/",year,sep = "")
SOURCE <-getURL(url,encoding = "UTF-8")
PARSED <- htmlParse(SOURCE)
holidays <- xpathSApply(PARSED, "//th[@class='nw']",xmlValue)
holidaydates <- paste(holidays,year,sep = ",")
holidaydatescon <- as.Date(holidaydates,format = "%B %d,%Y")

checkHoliday <- function(date){
 ## Call the function here to get holidaydates
 hol = if(date %in% holidaydatescon) 1 else 0
 return(hol)
}
```

- We will now call the pass our data set and append a Holiday column to the dataset. Column will be evaluated using the above function defined

```
require(dplyr)
finlandEnergy.agg <- finlandEnergy.agg %>% mutate(Holiday = sapply(date, function(x) checkHoliday(x)))
```

	BuildingID	meternumb	vac	type	date	hour	Consumption	month	Day.of.Week	weekday	Base_hour_Flag	Holiday
0	5198	1	Building 1	elect	2013-01-01	19	23	1	3	1	0	1
1	5198	1	Building 1	elect	2013-01-01	20	26	1	3	1	0	1
2	5198	1	Building 1	elect	2013-01-01	21	24	1	3	1	0	1
3	5198	1	Building 1	elect	2013-01-01	22	24	1	3	1	1	1
4	5198	1	Building 1	elect	2013-01-01	23	25	1	3	1	1	1
5	5198	1	Building 1	elect	2013-01-02	0	24	1	4	1	1	0
6	5198	1	Building 1	elect	2013-01-02	1	25	1	4	1	1	0

2.3 GETTING WEATHER INFORMATION

- We will read the other file, which contains the address information of each building. To get the Weather data, we will follow the below approach
  - Using the address, find the latitude and longitude of the building using Google API. It will return the latitude and longitude of the address
  - Using the latitude and longitude information we will find the nearest city (airport code) associated to it by using wunderground api.
  - Using the nearest city, we can use wunderground library to get the Weather information related to the particular building.

- We have defined few functions to solve the above purpose:
  a) To get the Latitude & Longitude by sending the address (GeoCode.R)
  b) Getting the nearest city to the airport(AirportFunction.R)
  c) Fetching the weather data for each airport code (WeatherFunction.R)
- First we will read the address information using the csv file

```
## reading the "Area" file
area.csv <- read.csv("Finland_addresses_area.csv",header = TRUE,stringsAsFactors = FALSE)
area.csv$area_floor._m.sqr <- as.numeric(area.csv$area_floor._m.sqr)
```

- To get the Latitude & Longitude, we defined the function as shown below. **getGeoCode function** will calculate the longitude and latitude based on the address provided to the function from the google API. The output has Longitude as lat and Longitude as lng.

```
### Getting Laitude and longitude data

getGeoCode <- function(address,sensor = "false") {
  require(RJSONIO)
  root <- "http://maps.google.com/maps/api/geocode/json?"
  u <- paste0(root,"address=", address, "&sensor=false@key=AIzaSyBmO7kN8BXjz_prZnTFauynxpr4eHsADos
  target <- URLencode(u)
  dat <- fromJSON(target)
  if(length(dat$results) == 0){
    print("No Such Address found")
    return()
  }

  latitude <- dat$results[[1]]$geometry$location["lat"]
  longitude <- dat$results[[1]]$geometry$location["lng"]
  latlong <- paste(latitude,longitude,sep = " ")
  latlong <- as.character(latlong)
  return(latlong)
}
```

- The function returns the latitude and longitude as character without space which we have then made two columns of numeric type.

- Since, we have the function ready, we can call this function on our area.csv and append two columns of Latitude and Longitude next to it using the below code

```
####### Function to get Latitude and Longitude by sending Address
source("GeoCode.R") ## This functions returns latitude and longitude for any address in a character space
address.geocode <- area.csv[1:10,] %>% mutate(LatLong = sapply(X..address, function(x) getGeoCode(x)))
address.geocode <- area.csv %>% mutate(LatLong = sapply(X..address, function(x) getGeoCode(x)))

require(dplyr)
require(tidyr)
address.geocode$LatLong <- as.character(address.geocode$LatLong)

address.geocode <- address.geocode %>%  separate(LatLong,c("latitude","longitude")," ")
address.geocode$latitude <- as.numeric(address.geocode$latitude)
address.geocode$longitude <- as.numeric(address.geocode$longitude)
```

Filter

	building	X..address	area_floor._m.sqr	latitude	longitude
1	Building 1	Metsahovintie 113	110	60.21869	24.39346
2	Building 2	Otakaari 3	11697	60.18835	24.82941
3	Building 3	Konemiehentie 1	469	60.18715	24.82304
4	Building 4	Rakentajanaukio 4	8766	60.18695	24.83034
5	Building 5	Vuorimiehentie 2	11068	60.18294	24.82589
6	Building 6	Otakaari 1	41339	60.18626	24.82843
7	Building 7	Puumiehenkuja 3	2661	60.18740	24.82525
8	Building 8	Otakaari 4	8206	60.18710	24.82740
9	Building 9	Vuorimiehentie 1	6455	60.18179	24.82469
10	Building 10	Sahkomiehentie 4	7954	60.18905	24.82580
11	Building 11	Puumiehenkuja 5	5100	60.18776	24.82400
12	Building 12	Otakaari 5	12240	60.18968	24.82996
13	Building 13	Otaniementie 9	5868	60.18449	24.82846
14	Building 14	Tietotie 1	15467	60.18540	24.82061

- The second function **Airport Function** would fetch the closest city using the wunderground api. The call to the api gives us sometimes more than one nearest city associated with the latitude and longitude.
- We can handle this by calculating the distance between the actual latitude and longitude and the output latitude longitude using the function getDistance as defined below

```
## Get distance between two points on Earth.
getdistance <- function (lat1, lon1, lat2, lon2) {
   p = 0.017453292519943295    # Math.PI / 180
   a = 0.5 - cos((lat2 - lat1) * p)/2 + cos(lat1 * p) * cos(lat2 * p) * (1 - cos((lon2 - lon1) * p))/2

  return(12742 * asin(sqrt(a))) ## 2 * R; R = 6371 km
}
```

- Actual function looks like below. It takes two argument as input lat and long and call the api based on the received lat and long. Received JSON data is parsed and the result is saved in a dataframe. For each record in the dataframe, getdistance method is called to calculate the distance. At last, dataframe containing the least distance is sent back with the city code.

```
getAirpot <- function(lat,long){
  #272f5ee91491de50
  #3a3f6a4875014282
  root <- "http://api.wunderground.com/api/3a3f6a4875014282/geolookup/q/"
  u <- paste0(root,lat,",",long,".json")
  require(RJSONIO)
  target <- URLencode(u)
  dat <- fromJSON(target)
  airports <- dat$location$nearby_weather_stations$airport$station
  q = NULL
  for (i in 1:length(airports)){
      q <- rbind(q,airports[[i]])
  }

  q <- as.data.frame(q,stringsAsFactors = FALSE)
  q[,"distance"] <- NA
  for (i in 1:nrow(q)){
    if(q[i,"city"] != "" & q[i,"icao"] != ""){
      q[i,"distance"] <- getdistance(lat,long,as.numeric(q[i,"lat"]),as.numeric(q[i,"lon"]))
    }
  }
```

- Since we have the function defined now, we can call the function using the below code which will give us our required airport code

```
## Now we can get the nearest city to this airport using the function attached
source("AirportFunction.R")
#address.final <- address.geocode %>% mutate(City = sapply(latitude,longitude, function(x,y) getAirpot(x,y)))
address.geocode$City <- mapply(getAirpot,address.geocode$latitude,address.geocode$longitude)
```

	building	X..address	area_floor._m.sqr	latitude	longitude	City
1	Building 1	Metsahovintie 113	110	60.21869	24.39346	EFHK
2	Building 2	Otakaari 3	11697	60.18835	24.82941	EFHF
3	Building 3	Konemiehentie 1	469	60.18715	24.82304	EFHF
4	Building 4	Rakentajanaukio 4	8766	60.18695	24.83034	EFHF
5	Building 5	Vuorimiehentie 2	11068	60.18294	24.82589	EFHF
6	Building 6	Otakaari 1	41339	60.18626	24.82843	EFHF
7	Building 7	Puumiehenkuja 3	2661	60.18740	24.82525	EFHF
8	Building 8	Otakaari 4	8206	60.18710	24.82740	EFHF
9	Building 9	Vuorimiehentie 1	6455	60.18179	24.82469	EFHF
10	Building 10	Sahkomiehentie 4	7954	60.18905	24.82580	EFHF
11	Building 11	Puumiehenkuja 5	5100	60.18776	24.82400	EFHF
12	Building 12	Otakaari 5	12240	60.18968	24.82996	EFHF
13	Building 13	Otaniementie 9	5868	60.18449	24.82846	EFHF

- Now we can define our last Weather code function which receives a city code, min and maximum date and returns a data frame with the weather information using the wunderground library. **Please refer WeatherFunction.docx for the function**
- Since we do not want to call the Weather Function 33 times (number of records) we can find unique airport code and call the weather function and save the weather data for each station code

```
## Get Weather Data for the Airport Code
stationcode <- as.matrix(unique(address.geocode$City))


##Get WeatherData for each airport code
source("WeatherFunction.R")
for(i in stationcode){
  df <- getWeatherData(i,mindate = min(finlandEnergy.agg$date),maxdate = max(finlandEnergy.agg$date))
  assign(i,df)
}
```

2.4 MERGING DATA WITH WEATHER DATA

- Now since we have both the raw data and weather data, we can merge the dataset together respectively. Before that we need to normalize the rawdataset and make some changes
- Adding address information to the original dataset

```
## Merge data
mergeData <- merge(finlandEnergy.agg,address.geocode,by.x = "vac",by.y = "building")
```

- We will now add a normalized consumption column which will be consumption divided by the area of the building. We can name it “Consumption Kwh/Sqm”.
- Followed up by removing unwanted columns

```
## Normalizing Power consumption by area of building
require(dplyr)
finlandEnergy <- mergeData %>% mutate("consumption Kwh/sqm" = Consumption/area_floor._m.sqr)
finlandEnergy <- finlandEnergy %>% select(-c(X..address,latitude,longitude))
finlandEnergy$City <- as.character(finlandEnergy$City)
```

- Adding Base_Hr_Usage and Base_Hour_class column by average the sum of the consumption for hours in 0,1,2, and 23 and later a binary column which will be “High” if the consumption is greater than base_hr_usage and vice versa

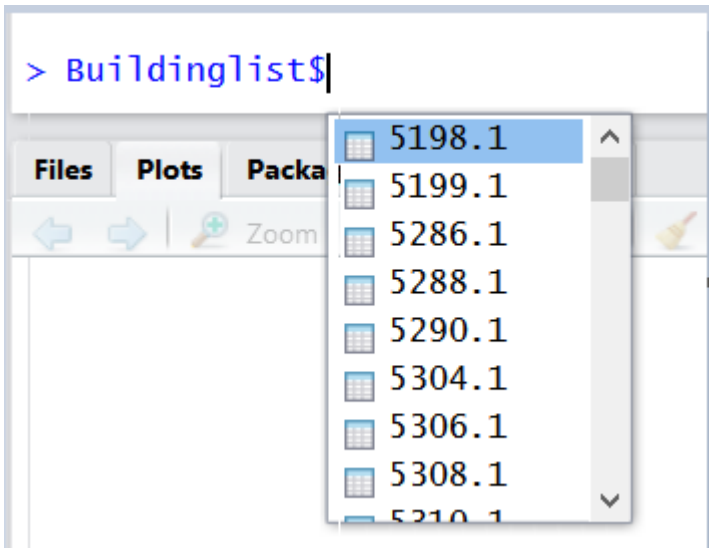
```
## Adding column base_hr_usage and Base_Hour_class
finlandEnergy <- finlandEnergy %>% group_by(BuildingID,meternumb,vac,type,date) %>%
  mutate(base_hr_usage = mean(Consumption[hour %in% c(0,1,2,23)]))

finlandEnergy <- mutate(finlandEnergy,Base_Hour_Class = ifelse(Consumption > base_hr_usage,"High","Low"))
```

```
> head(finlandEnergy)
         vac BuildingID meternumb  type       date hour Consumption month
1 Building 1       5198         1 elect 2013-01-01    0          25     1
2 Building 1       5198         1 elect 2013-01-01    1          25     1
3 Building 1       5198         1 elect 2013-01-01    2          26     1
4 Building 1       5198         1 elect 2013-01-01    3          25     1
5 Building 1       5198         1 elect 2013-01-01    4          25     1
6 Building 1       5198         1 elect 2013-01-01    5          25     1
  Day.of.Week weekday Base_hour_Flag Holiday area_floor._m.sqr City
1           3       1              1       1              110 EFHK
2           3       1              1       1              110 EFHK
3           3       1              1       1              110 EFHK
4           3       1              1       1              110 EFHK
5           3       1              1       1              110 EFHK
6           3       1              0       1              110 EFHK
  consumption Kwh/sqm base_hr_usage Base_Hour_Class
1           0.2272727         25.25             Low
2           0.2272727         25.25             Low
3           0.2363636         25.25            High
4           0.2272727         25.25             Low
5           0.2272727         25.25             Low
6           0.2272727         25.25             Low
```

- Before adding the weather data to our ‘finlandEnergy’ dataset. We need to divide our dataset with individual buildings and then add respective weather data.
- We will use split function which can divide the data frame into a grouped dataframe, which contains the set of dataframes with the combination of Building ID and meternumber.

```
## Spitting the finlandEnergy data with individual buildings
Buildinglist <- split(finlandEnergy, with(finlandEnergy, interaction(BuildingID,meternumb)), drop = TRUE)
```



- Now we can loop through our BuildingList and merge the respective weather information as shown in the below code. We can see individual datasets being created in the gobal environment with their respective weather data

```
#### Merging Weather related data with individual building type based on there location
m = 0
for(i in Buildinglist){
  m = m + 1
  build <- unique(i["vac"])
  value <- names(Buildinglist)[m]
  #value <- (strsplit(names(Buildinglist[m]),split = " ")[[1]])[2]
  if(unique(i["City"]) == "EFHK"){
    df <- merge(i,EFHK,by = c("date","hour"))
    name <- paste(build,".",value,sep = "")
    assign(name,df)
  } else if(unique(i["City"]) == "EFHF"){
    df <- merge(i,EFHF,by = c("date","hour"))
    name <- paste(build,".",value,sep = "")
    assign(name,df)
  }
}
```

Building 1.5198.1	8208 obs. of 26 variables
Building 10.5311.1	8208 obs. of 26 variables
Building 10.75700.1	8208 obs. of 26 variables
Building 10.75701.1	8208 obs. of 26 variables
Building 11.5313.1	8208 obs. of 26 variables
Building 12.5314.1	8208 obs. of 26 variables
Building 12.75712.1	8208 obs. of 26 variables
Building 12.75713.1	8208 obs. of 26 variables
Building 13.5316.1	3048 obs. of 26 variables
Building 13.75689.1	8208 obs. of 26 variables
Building 14.5317.1	8208 obs. of 26 variables
Building 14.5318.1	8208 obs. of 26 variables
Building 14.75702.1	8208 obs. of 26 variables
Building 14.75703.1	8208 obs. of 26 variables
Building 15.5322.1	8208 obs. of 26 variables
Building 15.5323.1	8208 obs. of 26 variables

- Checking a dataframe to ensure the information is as desired

```
> head(`Building 1.5198.1`)
        date hour        vac BuildingID meternumb  type Consumption month
1 2013-01-01    0 Building 1       5198         1 elect          25     1
2 2013-01-01    1 Building 1       5198         1 elect          25     1
3 2013-01-01   10 Building 1       5198         1 elect          26     1
4 2013-01-01   11 Building 1       5198         1 elect          23     1
5 2013-01-01   12 Building 1       5198         1 elect          26     1
6 2013-01-01   13 Building 1       5198         1 elect          26     1
  Day.of.Week weekday Base_hour_Flag Holiday area_floor._m.sqr City
1           3       1              1       1              110 EFHK
2           3       1              1       1              110 EFHK
3           3       1              0       1              110 EFHK
4           3       1              0       1              110 EFHK
5           3       1              0       1              110 EFHK
6           3       1              0       1              110 EFHK
  consumption Kwh/sqm base_hr_usage Base_Hour_Class TemperatureF Dew_PointF
1           0.2272727         25.25              Low         35.6       33.8
2           0.2272727         25.25              Low         33.8       32.9
3           0.2363636         25.25             High         37.4       36.5
4           0.2090909         25.25              Low         37.4       36.5
5           0.2363636         25.25             High         37.4       35.6
6           0.2363636         25.25             High         37.4       35.6
  Humidity Sea_Level_PressureIn VisibilityMPH Wind_SpeedMPH WindDirDegrees
1     93.0               29.420          2.80         13.85            160
2     96.5               29.405          2.20         15.55            145
3     96.5               29.300          4.65         16.70            190
4     96.5               29.300          4.35         14.95            190
5     93.0               29.300          4.95         16.15            190
6     93.0               29.300          3.40         17.25            190
  Conditions Wind_Direction
1 Light Rain            SSE
2       Rain             SE
3 Light Rain          South
4 Light Rain          South
5 Light Rain          South
6 Light Rain          South
>
```

- We can now merge all the individual dataset to a single dataframe and use that for analysis.

```
##### Combining the Overall dataset with weather
finland_Build_Weather = as.data.frame(NULL)
m = 0
for(i in Buildinglist){
  m = m + 1
  build <- unique(i["vac"])
  value <- names(Buildinglist)[m]
  if(exists(paste(build,".",value,sep = ""))){
    temp <- get(paste(build,".",value,sep = ""))
    finland_Build_Weather <- rbind(finland_Build_Weather,temp)
  }
}


write.csv(finland_Build_Weather,"finland_allBuilding_allWeather.csv")
```

3 PART 2: MODELLING

3.1 REQUIRED FIELDS

- We have added Base_hour_Usage and Base_Hour_Class columns in the original dataframe as mentioned in Part 1

3.2 GENERIC APPROACH: PREDICTION

We will predict consumption (KWH) for our Finland Energy Dataset using Regression, KNN, Random Forest and Neural Network models below. We will calculate predictions for each 78 Buildings we have and using all the models.

Approach for the Prediction is as follows

- We will create functions which can take a csv or object and return the coefficients of the model
- A for loop will be called which will call the defined function and give the results back
- Results will be appended and can be written down in a csv file

Note: For the functions and the result, please refer the following files in the Git repository

1. **Part2/SingleBuilding_Prediction/**

3.2.1 FEATURE SELECTION

Before proceeding with our models we have done best feature selection using four algorithms exhaustive search, forward, backward selection, stepwise regression. The best features that add to the predictive power of the model and irrelevant features removed from the model.

Performing all the feature selection methods we shortlisted below features to best predict our model.

```
hour, month, Day.of.Week, weekday, Holiday, base_hr_usage, TemperatureF, Dew_PointF,
Wind_SpeedMPH, WindDirDegrees, area_floor._m.sqr
```

3.2.2 CREATING THE FUNCTION AND READING THE INPUT

We have created functions that would take csv as a data frame or csv file.

```
returnNeuralNetwork <- function(csv){
  library(forecast)

  #write.csv(`Building 1.5198.1`,"Building 1.5198.1.csv")
  #csv <- name

  require(dplyr)
  mergeData <- csv %>% select(-c(date,consumption.Kwh.sqm,area_floor._m.sqr,vac,BuildingID,
                                 meternumb,type,Base_hour_Flag,City,Base_Hour_Class))
```

Csv data inserted into mergeData after removing the unwanted features.

3.2.3 CONVERSION OF DATA TYPE

Data types of the read file is converted as below.

```
## Converting non numeric features to numeric
mergeData$hour <- as.numeric(mergeData$hour)
mergeData$Consumption <- as.numeric(mergeData$Consumption)
mergeData$month <- as.numeric(mergeData$month)
mergeData$Day.of.Week <-  as.numeric(mergeData$Day.of.Week)
mergeData$weekday <-  as.numeric(mergeData$weekday)
mergeData$Holiday <- as.numeric(mergeData$Holiday)
```

3.2.4 SPLIT VALIDATION AND SCALING THE FEATURES

We are using the split validation and scaling of features. When you suspect that the data is not consistent. You can easily see this when you go through the results of the summary() function.

Look at the minimum and maximum values of all the (numerical) attributes. If you see that one attribute has a wide range of values, you will need to normalize your dataset, because this means that the distance will be dominated by this feature.

```
## Split validation 75%
index <- sample(1:nrow(merge.sel),round(0.75*nrow(merge.sel)))

## Scaling the data before fitting the model so that convergence is possible for
## the neurons and fitting is valid
maxs <- apply(merge.sel, 2, max)
mins <- apply(merge.sel, 2, min)

scaled <- as.data.frame(scale(merge.sel, center = mins, scale = maxs - mins))

train_ <- scaled[index,]
test_ <- scaled[-index,]
```

With above function we have computed maxs and mins of the numerical features and excluding the categorical features and then re-distributed the values.

3.2.5 BASELINE MODEL

In the absence of any predictor, all we have is the dependent variable (Consumption). What would be our best guess if we had to predict the amount of Consumption, on a given day, in the test set? It is the mean of the Consumption values, in the training data, is the best value.

```
best.guess <- mean(train.sample$Consumption)
RMSE.baseline <- sqrt(mean((best.guess-train.val$Consumption)^2))
MAE.baseline <- mean(abs(best.guess-train.val$Consumption))
```

This value will give us the fair idea on how are the predictions compared to RMSE baseline and RME baseline, in further steps.

3.2.6 DIFFERENT MACHINE LEARNING ALGORITHMS AND OUTPUT

3.2.6.1 REGRESSION

Linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables.

```
## Linear Regression model
set.seed(2014)
lin.reg <- lm(formula = Consumption ~ hour + month + Day.of.Week + weekday + Holiday
              + base_hr_usage + TemperatureF + Dew_PointF + Wind_SpeedMPH + WindDirDegrees,
              data = train.sample)
```

ModelName	Type	RMSE.baseline	MAE.baseline	RMSE.model	MAE.model	ME.model
Building 15.5322.1	elect	8.467102198	7.558388858	6.954511765	5.75478239	0.196602531
Building 15.5323.1	elect	3.973045095	3.294424494	2.939427692	2.348576806	0.042807802
Building 16.5325.1	elect	13.96259979	11.44155382	11.26379846	9.271670225	0.687662357
Building 17.5329.1	elect	41.38109743	36.08841546	37.52142531	31.75930489	1.524817713
Building 18.5332.1	elect	5.722252612	4.539875444	5.287784406	4.188473829	0.146295825

3.2.6.2 KNN

KNN or k-nearest neighbors' algorithm is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored, labeled instances. More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure.

This similarity measure is typically expressed by a distance measure such as the Euclidean distance, cosine similarity or the Manhattan distance.

```
# KNN Model
knn.model <- FNN::knn.reg(train=train_[,-2], test = test_[,-2], y=train_$Consumption, k=3,
                          algorithm = c("brute"))
```

ModelName	Type	RMSE.baseline	MAE.baseline	RMSE.model	MAE.model	ME.model	MAPE.model
Building 15.5322.1	elect	8.566021407	7.616323541	3.464640846	2.234080572	-0.153508772	-3.329403632
Building 15.5323.1	elect	3.995853228	3.291692791	1.970007405	1.446231319	-0.132391163	-7.401196357
Building 16.5325.1	elect	13.59206972	11.00029164	5.956387958	3.987979207	-0.13222872	-1.523139496
Building 17.5329.1	elect	40.82474504	35.74359252	16.27416149	9.890350877	0.236679662	-3.784144712
Building 18.5332.1	elect	5.606651057	4.403393035	2.581370169	1.699317739	-0.023879142	-4.057447685

3.2.6.3 RANDOM FOREST

The random forest starts with a standard machine learning technique called a "decision tree". This is a type of additive model that makes predictions by combining decisions from a sequence of base models.

```
## Random forest model
rf <- randomForest(Consumption ~ hour + month + Day.of.Week + weekday + Holiday + base_hr_usage
                   + TemperatureF + Dew_PointF + Wind_SpeedMPH + WindDirDegrees,
                   data = train.sample,
                   importance = TRUE,
                   ntree=ntree)
```

ModelName	Type	RMSE.baseline	MAE.baseline	RMSE.model	MAE.model	ME.model	MAPE.model
Building 25.75691.1	Dist_Heating	32.82932495	26.87878203	5.212448068	2.964157185	-0.275736746	NA
Building 5.75692.1	Dist_Heating	80.73522699	68.33697047	12.84373214	9.491859391	-0.147864664	NA
Building 17.75693.1	Dist_Heating	73.86375697	61.947157	10.79116273	6.085606664	-0.301580956	NA
Building 15.75694.1	Dist_Heating	24.82729727	20.37207992	3.690838085	2.060656126	-0.101346727	NA
Building 23.75696.1	Dist_Heating	47.31552233	36.97521874	8.733326385	4.428842162	-0.022518036	NA

3.2.6.4 NEURAL NETWORK

Neural network terminology is inspired by the biological operations of specialized cells called neurons. A neuron is a cell that has several inputs that can be activated by some outside process.

The artificial equivalent of a neuron is a node (also sometimes called neurons, but I will refer to them as nodes to avoid ambiguity) that receives a set of weighted inputs, processes their sum with its activation function, and passes the result of the activation function to nodes further down the graph.

```
## Neural Network
library(neuralnet)
n <- names(train_)
f <- as.formula(paste("Consumption ~", paste(n[!n %in% c("Consumption","Sea_Level_PressureIn",
                                                         "Humidity","VisibilityMPH")], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=1,linear.output=T,stepmax = 1e+06)
```

ModelName	Type	RMSE.baseline	MAE.baseline	RMSE.model	MAE.model	ME.model	MAPE.model
Building 15.5322.1	elect	8.706718785	7.753462604	6.914013175	5.717025712	0.136473021	#NAME?
Building 15.5323.1	elect	3.989457874	3.256236487	2.843549035	2.223737603	0.064368267	NA
Building 16.5325.1	elect	14.16388014	11.46764152	11.50504002	9.35579146	0.485704457	NA
Building 17.5329.1	elect	41.0670715	36.05751564	37.47580708	30.25979977	-0.32254956	#NAME?
Building 18.5332.1	elect	5.680665022	4.492937998	5.215237282	4.141898801	0.132680813	#NAME?

3.2.7 STORING AND RETURNING THE RESULTS

We are storing the results as below, where ModelName, Type, RMSE.baseline, MAE.baseline.

```
### Putting the results into the data frame
results = as.data.frame(NULL)
results[1,"ModelName"] <- unique(paste(csv$vac,csv$BuildingID,csv$meternumb,sep = "."))
results[1,"Type"] <- unique(csv$type)
results[1,"RMSE.baseline"] <- RMSE.baseline
results[1,"MAE.baseline"] <- MAE.baseline
```

Calculating the results from the different machine learning algorithms.

```
## Evaluating the model's accuracy
resultset <- data.frame(actual = test.r, prediction = pr.nn_)
accuracy <- accuracy(resultset$prediction, resultset$actual)
```

When we have all the output values, we will return it from the function

```
  ## Adding coefficients to the results tab
  results[1,"RMSE.model"] <- accuracy[2]
  results[1,"MAE.model"] <- accuracy[3]
  results[1,"ME.model"] <- accuracy[1]
  results[1,"MAPE.model"] <- accuracy[4]


  return(results)
}
```

3.2.8 META EVALUATOR

The above function is run repeatedly for all the datasets using below code. We will execute 4 ‘for’ loops for every Algorithm for 78 different datasets which computes the MAPE, MAE, RMSE values along with Baseline values.

Below is the sample code for Neural network ‘for’ loop, we have similar loop for Regression, KNN and Random Forest.

```
### Evaluating Neural Network Model on 78 different datasets
source("Part2_NeuralNetwork.R")
neuralnetwork <- as.data.frame(NULL)
m = 0
for(i in Buildinglist){
  m = m + 1
  build <- unique(i["vac"])
  value <- names(Buildinglist)[m]
  if(exists(paste(build,".",value,sep = ""))){
    name <- get(paste(build,".",value,sep = ""))
    print(paste(build,".",value,sep = ""))
    colnames(name)[which(names(name) == "consumption Kwh/sqm")] <- "consumption.Kwh.sqm"
    modelcoeff <- returnNeuralNetwork(csv = name)
    neuralnetwork <- rbind(neuralnetwork,modelcoeff)
  }
}

write.csv(neuralnetwork,"radomForest_All.csv")
```

Note: For the functions and the result, please refer the following files in the Git repository

1. **Part2/SingleBuilding_Prediction/ Part2_ModelEvaluation.R**

3.2.9 PREDICTION MODEL EVALUATION: WHICH MODEL TO CHOOSE

After we have the output from all the files for all the model, the baseline RMSE value will give fair idea on how are the predictions compared to RMSE value of the model. We had compared results of all the models decided **Random Forest** gives us much better result. Random forest output is as shown below

		ModelName	Type	RMSE.baseline	MAE.baseline	RMSE.model	MAE.model	ME.model
2	1	Building 1.5198.1	elect	5.593393414	3.585660526	2.296510362	1.341585558	0.101906894
3	2	Building 2.5199.1	elect	111.3696726	92.55094138	14.86867479	9.832327826	-0.249996209
4	3	Building 3.5286.1	elect	1.689228989	1.006380821	1.15939283	0.717441172	-0.000629313
6	5	Building 5.5290.1	elect	48.74529957	43.57629363	5.741472423	4.050662687	-0.018138881
7	6	Building 6.5304.1	elect	19.38991521	9.905761645	1.637801408	0.27580233	-0.02235513
8	7	Building 7.5306.1	elect	21.36808962	16.89628277	4.608367276	2.456296579	0.015680021
9	8	Building 8.5308.1	elect	30.50202382	27.17686877	4.053022083	2.612748694	-0.124539742

3.2.10 OUTLIER ANALYSIS RANDOM FOREST

Note: For the functions and the result, please refer the following files in the Git repository

1. **Part2/SingleBuilding_Prediction/ RandomForest_Outlier**

We have done outlier detection in below steps.

3.2.10.1.1 COMPUTE PREDICTIONS AND RESIDUALS

We predict the values using our Random Forest model. We will then calculate the residuals. Since Random Forest does not give us residuals directly, we can calculate Residuals as shown below, which is nothing but absolute difference of actual verses predicted values.

```
## Predicting the model
test.pred.forest <- predict(rf,train.val)
accuracy <- accuracy(test.pred.forest, train.val$Consumption)

## Computing the predictions and residuals. Saving the results
output <- data.frame(Actual_Kwh = train.val$Consumption,
                     Predicted_Kwh = test.pred.forest,
                     Residuals = abs(train.val$Consumption-test.pred.forest))
```

Difference from ‘Actual_Kwh’ to ‘Predicted_Kwh’ gives us ‘Residuals’.

3.2.10.1.2 STANDARD DEVIATION ON THE RESIDUALS

Standard deviation is calculated using base r function sd().

```
## Calculating the standard deviation on the residuals
sd <- sd(test.pred.forest, na.rm = FALSE)
```

3.2.10.1.3 TAG POINTS AS OUTLIERS

We can compute another column called outliers which gives us Outlier if the residual is greater than 2 times the standard deviation

```
## Calculating outlier
output <- output %>% mutate(Outliers = ifelse(output$Residuals > 2*sd,"Outlier",""))
```

In below example SD was computed as 32.78 for Building 5 unit 5310 meter 1. So any value greater than 65.56 will be marked as an outlier.

Actual_Kwh	Predicted_Kwh	Residuals	Outliers
108	109.3138333	1.313833333	
183	143.5371167	39.46288331	
239	163.7221055	75.27789451	Outlier
97	98.0135	1.0135	
167	166.804	0.196	

From the example we can see that residual 75 < 2*32.78 hence it is marked as an outlier. Overall outliers detected from all the buildings were around 2%.

3.3 GENERIC APPROACH: CLASSIFCATION

Note: For the functions and the result, please refer the following files in the Git repository

1. **Part2/SingleBuilding_Classifcation/**"RespectiveModel"

Generic steps remain the same, however the feature selection and the formula changes for the model. We will discuss the algorithm used in the below section for all the Machine Learning algorithm used for classifcation

3.3.1 DIFFERENT MACHINE LEARNING ALGORITHMS AND OUTPUT

Note: For the functions and the result, please refer the following files in the Git repository

1. **Part2/SingleBuilding_Classification/**

3.3.1.1 LOGISTIC REGRESSION

Binary Logistic Regression **is a special type** of regression where binary response variable is related to a set of explanatory variables, which can be discrete and/or continuous. We ae using **glm()** function to predict the Base_Class_Hour in this case of classification.

```
## Neural Network for classification
n <- names(train_)
f <- as.formula(paste("Base_Class ~", paste(n[!n %in% c("Base_Class","Sea_Level_PressureIn",
                                                        "Humidity","VisibilityMPH")], collapse = " + ")))
logit <- glm(f,family = "binomial",data = train_)
```

In this case, our function will result the Confusion matrix as shown below and also the ROC curve will be saved as described later

Model	Type	Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
Building 1.5198.1	2	1	1	0.998203915	1	0.515107212	0
Building 2.5199.1	2	1	1	0.998203915	1	0.828947368	6.57E-168
Building 3.5286.1	2	1	1	0.998203915	1	0.505847953	0
Building 5.5290.1	2	1	1	0.998203915	1	0.712475634	7.66E-303
Building 6.5304.1	2	1	1	0.998068493	1	0.944968553	1.25E-47
Building 7.5306.1	2	1	1	0.998203915	1	0.696393762	3.458459520888

3.3.1.2 KNN

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). Here we are using caret library with a train control of 3 times repeated cross validation. Model is trained with train control and also a preprocess of center and scale is applied to the features.

```
## KNN Classification model
ctrl <- trainControl(method="repeatedcv",repeats = 3)
knn.class <- train(Base_Hour_Class ~ hour + Consumption + month + Day.of.Week + weekday + Holiday
                   + base_hr_usage + TemperatureF + Dew_PointF + Wind_SpeedMPH + WindDirDegrees,
                   data= train.sample,
                   method = "knn",
                   trControl = ctrl,
                   preProcess = c("center","scale"),
                   tuneLength = 1)
```

Model	Type	Type	Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
Building 1.	2	Elect	0.785018	0.569462	0.764342896	0.8046695	0.527405603
Building 2.	2	Elect	0.872107	0.531944	0.854982057	0.887889233	0.830085262
Building 3.	2	Elect	0.813033	0.625039	0.793316908	0.831622876	0.515834348
Building 5.	2	Elect	0.844093	0.617019	0.825629013	0.861316513	0.710718636
Building 6.	2	Elect	0.980996	0.838034	0.972820497	0.98723675	0.938401048
Building 7.	2	Elect	0.819732	0.586021	0.800268164	0.838045481	0.681485993

3.3.1.3 RANDOM FOREST

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest). We use caret library to compute the Random Forest similar to what we did earlier in KNN.

```
## RF classification model
ctrl <- trainControl(method="repeatedcv",repeats = 3)
rf.class <- train(Base_Hour_Class ~ hour + Consumption + month + Day.of.Week + weekday + Holiday + base_hr_usage
                  + TemperatureF + Dew_PointF + Wind_SpeedMPH + WindDirDegrees,
                  data= train.sample,
                  method = "rf",
                  trControl = ctrl,
                  preProcess = c("center","scale"),
                  tuneLength = 1)
```

ModelName	Type	Accuracy
Building 14.5318.1	elect	96.46772229
Building 15.5322.1	elect	97.32034105
Building 15.5323.1	elect	99.51278928
Building 16.5325.1	elect	96.95493301
Building 17.5329.1	elect	98.17295981

3.3.1.4 NEURAL NETWORK

Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record.

We are using NeuralNet library to train the model with a hidden layer of 3/2 and threshold of 0.1.

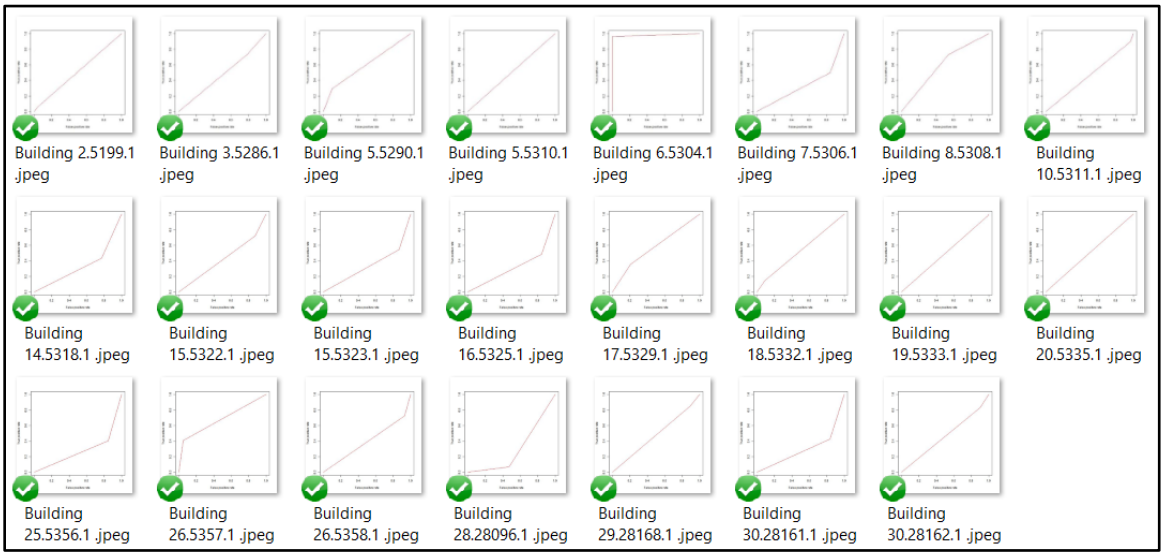
```
## Neural Network for classification
library(neuralnet)
n <- names(train_)
f <- as.formula(paste("Base_Class ~", paste(n[!n %in% c("Consumption","Day.of.Week","month",
                                                        "TemperatureF","Base_Class","Sea_Level_PressureIn",
                                                        "weekday","Humidity","VisibilityMPH","Dew_PointF",
                                                        "Holiday","Wind_SpeedMPH","WindDirDegrees")],
                                            collapse = " + ")))
nn <- neuralnet(f,data=train_,linear.output = F,hidden = c(4,2),stepmax = 1e+05, threshold = 0.1)
```

ModelName	Type	Accuracy
Building 5.5290.1	elect	72.12%
Building 6.5304.1	elect	96.91%
Building 5.5310.1	elect	80.31%
Building 11.5313.1	elect	80.36%
Building 13.5316.1	elect	94.49%

3.3.1.5 ROC CHARTS

We have calculated ROC for all the models. Below is sample code to generate the ROC.

```
knn_roc <-  roc(train.val$Base_Hour_Class, knnPredict[,1], levels = rev(train.val$Base_Hour_Class))
plotName <- paste("Knn_Roc/", results[1],".jpeg")
jpeg(file = plotName)
myplot <- plot(knn_roc, type="S", print.thres= 0.5)
dev.off()
```



3.3.1.6 OUTLIER DETECTION

For the best model, we have computed as follows:

3.3.1.6.1 COMPUTED PREDICTION FLAG

We have computed the prediction from our dataset so that we can compare it against the actual value.

```
# Prediction flag
merge.sel_OL <- merge.sel
merge.sel_OL$predicted_base_hour_flag <- predict(knn.class,newdata = merge.sel)
```

If we have the mismatch between predicted and Actual value we have a new column called ‘Score’ which has value 1 for mismatch and 0 for no mismatch.

```
merge.sel_OL <- merge.sel_OL %>%
                mutate(score = ifelse(merge.sel_OL$Base_Hour_Class
                                      == merge.sel_OL$predicted_base_hour_flag, 0 ,1))
```

3.3.1.6.2 OUTLIER_DAY

We have computed ‘outlier_day’ column as If there is a mismatch in flags for 6 or more hours, we tagged all rows for that day as True else False.

```
# Detectig outlier day
new_data <- merge.sel_OL %>%
                group_by(date) %>%
                mutate(Outlier_day = ifelse(sum(score) > 5, "TRUE", "FALSE"))
```

date	hour	Consumpt	month	Day.of.We	weekday	Holiday	base_hr_u	Base_Hou	Temperatu	Dew_Poin	Humidity	Sea_Level	VisibilityM	Wind_Spe	WindDirD	predicted_	score	Outlier_day
1/1/2013	7	27	1	3	1	1	25.25	High	37.4	35.6	93	29.3	6.2	13.85	180	Low	1	TRUE
1/1/2013	8	23	1	3	1	1	25.25	Low	37.26667	36.06667	94.66667	29.30667	5.1	14.83333	183.3333	High	1	TRUE
1/1/2013	9	26	1	3	1	1	25.25	High	37.4	35.6	93	29.3	3.7	13.25	180	High	0	TRUE
1/2/2013	0	24	1	4	1	0	25.5	Low	36.5	35.6	96.5	29.42	1.2	9.25	190	Low	0	FALSE
1/2/2013	1	25	1	4	1	0	25.5	Low	35.6	35.6	100	29.42	1.85	8.65	195	Low	0	FALSE
1/2/2013	10	53	1	4	1	0	25.5	High	34.7	33.8	96.5	29.5	6.2	6.35	205	High	0	FALSE
1/2/2013	11	66	1	4	1	0	25.5	High	35.6	32.9	90	29.53	6.2	9.8	235	High	0	FALSE

3.4 GENERIC APPROACH: ALL BUILDINGS (PREDICTION AND CLASSIFICTION)

Note: For the functions and the result, please refer the following files in the Git repository

1. **Part2/ AllBuilding_Script&Result**

For one dataset as a file, we will the follow the same approach as described in part 3.3.2 and 3.3.3.

Code for performing all the machine algorithm on the single dataset is located at **Part2/ AllBuilding_Script&Result/ Part2_AllBuildings_AllModel.R**

3.4.1 CLASSIFICATION OUTPUT – (KNN EXAMPLE)

	hour	mont	Day.of.	week	Base_	Holiday	area_floor	consumpti	base_hr_u	Temperatu	Dew_Poin	Humidity	Sea_Level	VisibilityM	Wind_Spe	WindDirDegrees	Base_Hour_Class	Predicted_Base_Hour_Class
7	0.608696	0	0.333	1	0	1	0	0.378788	0.011827	0.544974	0.658988	0.961686	0.322404	0.139842	0.49509	0.546296296	Low	Low
12	0.826087	0	0.333	1	0	1	0	0.348485	0.011827	0.546218	0.670157	1	0.338798	0.242744	0.285271	0.583333333	Low	Low
15	0.913043	0	0.333	1	0	1	0	0.363636	0.011827	0.546218	0.651309	0.91954	0.355191	0.258575	0.286822	0.569444444	Low	Low
18	0.130435	0	0.333	1	1	1	0	0.378788	0.011827	0.512605	0.632461	1	0.338798	0.242744	0.446512	0.416666667	Low	High
19	0.173913	0	0.333	1	1	1	0	0.378788	0.011827	0.529412	0.632461	0.91954	0.322404	0.292876	0.410853	0.416666667	Low	High
38	0.869565	0	0.5	1	0	0	0	0.409091	0.011944	0.513228	0.613613	0.908046	0.464481	0.324538	0.235659	0.583333333	High	High
51	0.434783	0	0.667	1	0	0	0	0.5	0.012061	0.495798	0.575916	0.844828	0.464481	0.324538	0.213953	0.763888889	High	High
56	0.652174	0	0.667	1	0	0	0	0.439394	0.012061	0.495798	0.58534	0.885057	0.464481	0.203166	0.125581	0.611111111	High	High
60	0.826087	0	0.667	1	0	0	0	0.409091	0.012061	0.478992	0.575916	0.91954	0.448087	0.242744	0.125581	0.472222222	High	High
63	0.913043	0	0.667	1	0	0	0	0.409091	0.012061	0.495798	0.594764	0.91954	0.448087	0.176781	0.125581	0.347222222	High	High
64	0.956522	0	0.667	1	1	0	0	0.409091	0.012061	0.495798	0.594764	0.91954	0.448087	0.137203	0.125581	0.319444444	High	Low
65	1	0	0.667	1	1	0	0	0.393939	0.012061	0.495798	0.613613	1	0.448087	0.08971	0.16124	0.361111111	High	Low
74	0.043478	0	0.833	0	1	0	0	0.378788	0.012061	0.495798	0.613613	1	0.448087	0.081794	0.125581	0.319444444	Low	High
79	0.608696	0	0.833	0	0	0	0	0.454545	0.012061	0.510115	0.617103	0.938697	0.484517	0.211961	0.082687	0.666666667	High	High
82	0.73913	0	0.833	0	0	0	0	0.348485	0.012061	0.512605	0.613613	0.91954	0.489071	0.121372	0.089922	0.847222222	Low	Low

3.4.2 PREDICTION OUTPUT – (KNN EXAMPLE)

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
hour	month	Day.of.We	weekday	Base_hour	Holiday	area_floor	base_hr_u	Temperatu	Dew_Poin	Humidity	Sea_Level	VisibilityM	Wind_Spe	WindDirDe	consumpti	Pred_Consumptio	
0.043478	0	0.333333	1	1	1	0	0.011827	0.512605	0.623037	0.95977	0.363388	0.113456	0.482171	0.402778	0.227273	0.024973	
0.434783	0	0.333333	1	0	1	0	0.011827	0.546218	0.660733	0.95977	0.306011	0.242744	0.517829	0.527778	0.236364	0.024628	
0.565217	0	0.333333	1	0	1	0	0.011827	0.546218	0.651309	0.91954	0.306011	0.176781	0.534884	0.527778	0.236364	0.025016	
0.913043	0	0.333333	1	0	1	0	0.011827	0.546218	0.651309	0.91954	0.355191	0.258575	0.286822	0.569444	0.218182	0.020955	
1	0	0.333333	1	1	1	0	0.011827	0.546218	0.651309	0.91954	0.371585	0.176781	0.322481	0.541667	0.227273	0.021577	
0.217391	0	0.333333	1	0	1	0	0.011827	0.537815	0.651309	0.95977	0.314208	0.324538	0.339535	0.458333	0.227273	0.022058	
0.26087	0	0.333333	1	0	1	0	0.011827	0.546218	0.660733	0.95977	0.306011	0.324538	0.427907	0.5	0.227273	0.023231	
0	0	0.5	1	1	0	0	0.011944	0.537815	0.651309	0.95977	0.371585	0.060686	0.286822	0.527778	0.218182	0.024142	
0.73913	0	0.5	1	0	0	0	0.011944	0.512605	0.613613	0.91954	0.464481	0.324538	0.268217	0.583333	0.3	0.024383	
0.869565	0	0.5	1	0	0	0	0.011944	0.513228	0.613613	0.908046	0.464481	0.324538	0.235659	0.583333	0.245455	0.023855	
0.913043	0	0.5	1	0	0	0	0.011944	0.512605	0.613613	0.91954	0.448087	0.274406	0.285271	0.569444	0.245455	0.024682	
0	0	0.666667	1	1	0	0	0.012061	0.512605	0.613613	0.91954	0.448087	0.324538	0.213953	0.694444	0.227273	0.023982	
0.043478	0	0.666667	1	1	0	0	0.012061	0.512605	0.613613	0.91954	0.448087	0.324538	0.196899	0.680556	0.245455	0.023732	
0.086957	0	0.666667	1	1	0	0	0.012061	0.510115	0.613613	0.915709	0.455373	0.549692	0.234625	0.703704	0.227273	0.024316	
0.869565	0	0.666667	1	0	0	0	0.012061	0.481481	0.594066	0.97318	0.453552	0.331574	0.141602	0.416667	0.245455	0.023769	
0.086957	0	0.833333	0	1	0	0	0.012061	0.495798	0.613613	0.992337	0.455373	0.094107	0.10646	0.351852	0.236364	0.020306	
0.913043	0	0.833333	0	0	0	0	0.012061	0.512605	0.613613	0.91954	0.538251	0.324538	0.268217	0.916667	0.236364	0.021793	
0.130435	0	0.833333	0	1	0	0	0.012061	0.512605	0.613613	0.91954	0.464481	0.121372	0.142636	0.472222	0.227273	0.020553	

4 COMPARISON and CONCLUSION

Each algorithm has certain pros and cons, we have summarized the observed behavior in the table below.

Algorithm	KNN	Linear regression	Logistic regression	Random Forests
Problem Type	Either	Regression	Classification	Either
Results interpretable	Yes	Yes	Somewhat	A little
Average predictive accuracy	Lower	Lower	Lower	Higher
Training speed	Fast	Fast	Fast	Slow
Prediction speed	Depends on n	Fast	Fast	Moderate
Amount of parameter tuning needed	Minimal	None (excluding regularization)	None (excluding regularization)	Some
Performs well with small observations?	No	Yes	Yes	No
Automatically learns feature interactions?	No	No	No	Yes
Parametric?	No	Yes	Yes	No
Features might need scaling?	Yes	No (unless regularized)	No (unless regularized)	No

END