MIDTERM

Team 5

Team members:

Dipti Pamnani
Jayesh Samyani
Suraj Sharma

Under guidance of :

Prof. Sri Krishnamurthy
Ashwin Dinoriya

Team 5 **Midterm**

Introduction

Abstract:

Midterm is focused to understand data coming from energy usage in Finland at Taria and Pasi Inc. and they are been approached by Vokia Inc. . They want to help monitor and reduce energy consumption of 78 buildings. Vokia Inc. owns these buildings and wants to understand and reduce energy usage and wants to make the buildings more energy efficient.

This task needs to process or cleanse data using different data wrangling and ingestion techniques. After cleaning data we have to build models for every unique data, which is combination of building ID and Meter Number, using different techniques for prediction(Linear regression, KNN, Random Forest and Neural Networks) and classification(Logistic regression, KNN, Random Forest and Neural Networks)) and clustering(k-means, Hierarchical) .

After building models, we have to analyze the performance of each model of 78 models using performance matrix and confusion matrix and detect outliers if any. Then based on performance we have to select the ideal technique for entire model set. For generalization, we need to build model on entire data set and predict, classify based on Consumption per squaremeter and Base hour class respectively.

GOAL

Use the data science skills get the weather data to do feature engineering and build models for prediction, classification and clustering and evaluate based on performance. Try to suggest using these techniques which is efficient way to reduce energy consumption based on features available.

Dipti Pamnani **Jayesh Samyani** **Suraj Sharma**

Part1: Data Ingestion and Wrangling

- Using R script ingest the data.

- Extract features like week of day, month of year, weekend, holiday, Base hour flag(Hours 0,1,2,3,4,22,23)

- Normalize Consumption with respect to area_floor _m.sqr to ensure you are working with Kwh/square meter.

- The historical hourly weather information for each of the building. To fetch weather data, we need to get the exact location of the building using address of the given building.
- After retrieving exact location of building, we need to find closest airport using geosphere formula using geo latitude and longitude of each building.
- Gather weather data for each building from closest airport location.

  Following are the columns of weather data
- **TimeEET,TemperatureF,Dew PointF,Humidity,Sea Level PressureIn,VisibilityMPH,Wind Direction,Wind SpeedMPH,Gust SpeedMPH,PrecipitationIn,Events,Conditions,WindDirDegrees,Date**

- Clean the data received from weather data using different techniques

Dipti Pamnani **Jayesh Samyani** **Suraj Sharma**

Team 5 **Midterm**

Follow the following steps for data ingestion and wrangling

1. Read the file and filter data with type of elect and Dist_heating
2. Fill all the missing values
3. Normalize data for consumption- consumption per squaremeter
4. Fetch date , month,year and other features(Base hr usage and base hr flag)
5. Calcutaing Base Hr Class based on actual Consumption and base hr usage
6. This is the function to retrieve the geolocation from the address
7. Fetch the geolocation data for the given addresses using the above function
8. Code to get the nearest airport for each Building
9. Fetch weather data for the airports
10. merge weather data for all unique airports corresponding to the building data

Dipti Pamnani **Jayesh Samyani** **Suraj Sharma**

1. Read the file and filter data with type of elect and Dist_heating

```
data <- read.csv("Finland_masked.csv",header = TRUE,stringsAsFactors = FALSE)
address <- read.csv("Finland_addresses_area.csv",header = TRUE,stringsAsFactors =
FALSE)
datatemp <- data

datatemp1=datatemp %>% group_by(vac,type) %>% filter(type=="elect" | type
=="Dist_Heating")
elect=datatemp1%>% filter(type=="elect" )
unique(x = elect$BuildingID)
heat=datatemp1%>% filter(type=="Dist_Heating" )
unique(x = heat$BuildingID)
unique(x = datatemp1$type=="elect")
```

2. Fill all the missing values

```
datatemp1$vac[datatemp1$BuildingID== 81909] <- "Building 27"
datatemp1$vac[datatemp1$BuildingID== 82254] <- "Building 9"
datatemp1$vac[datatemp1$BuildingID== 83427] <- "Building 9"
datatemp1$vac[datatemp1$BuildingID== 84681] <- "Building 9"

datatemp2 <-
transform(datatemp1,uniquekey=paste0(datatemp1$BuildingID,datatemp1$meternum
b))
datatemp2 <- plyr::rename(datatemp2, c("vac"="building"))
```

3. Normalize data for consumption- consumption per squaremeter

```
datatemp3 <- merge(x = datatemp2, y = address, by = "building", all.x = TRUE)
datatemp3$Consumption_per_squaremeter <-
(datatemp3$Consumption/datatemp3$area_floor._m.sqr)
```

4.Fetch date , month,year and other features(Base hr usage and base hr flag)

```
tempdate <- datatemp3[,5]


tempdate2 <- as.Date(as.character(tempdate), "%Y%m%d")
```

```
# fetching details from date column
datemonth <- format(tempdate2,"%m")
dateyear <- format(tempdate2,"%Y")
dateday <- format(tempdate2,"%d")

dayofweek <- weekdays(tempdate2)


ifweekend <- is.weekend(tempdate2)


holiday <-  as.data.frame(is.holiday(tempdate2,2013-05-01))
datedata <-
data.frame("Date"=tempdate2,"Month"=datemonth,"Day"=dateday,"Year"=dateyear,"
Day of Week"=dayofweek,"Weekend"=ifweekend)

datatemp4 <- cbind(datatemp3,datedata)
datatemp4$Base_hour_Flag <- ifelse((datatemp3$hour==0) |(datatemp3$hour==1)
|(datatemp3$hour==2) |(datatemp3$hour==3)
|(datatemp3$hour==4)|(datatemp3$hour==22)|(datatemp3$hour==23), "True",
"False")
datatemp4$Base_hour_Flag <- as.logical(datatemp4$Base_hour_Flag)
class(datatemp4$Base_hour_Flag)

datatemp4$Holiday <- ifelse((datatemp4$date==20130101)
|(datatemp4$date==20130106) |(datatemp4$date==20130329)
|(datatemp4$date==20130331) |(datatemp4$date==20130401)
|(datatemp4$date==20130501) |(datatemp4$date==20130509)
|(datatemp4$date==20130512) |(datatemp4$date==20130519)
|(datatemp4$date==20130621) |(datatemp4$date==20130622)
|(datatemp4$date==20131102) |(datatemp4$date==20131110)
|(datatemp4$date==20131206) |(datatemp4$date==20131224)
|(datatemp4$date==20131225) |(datatemp4$date==20131226) ,"True","False")
unique(x = datatemp2$uniquekey)

fetching features details like base hr usage and base hr flag

daily_base_hr_usage_dataset <-datatemp4 %>%
filter(datatemp4$Base_hour_Flag=="TRUE")


daily_base_hr_usage_dataset <-as.data.frame(daily_base_hr_usage_dataset %>%
group_by(BuildingID,type,meternumb,Weekend,Month,Holiday)%>%summarize(Base_h
r_usage=mean(Consumption_per_squaremeter)))
```

```
datatemp5 <- merge(x = datatemp4, y = daily_base_hr_usage_dataset, by =
c("BuildingID","type","meternumb","Weekend","Month","Holiday"), all.x = TRUE)
```

5. Calcutaing Base Hr Class based on actual Consumption and base hr usage

```
datatemp5$Base_Hour_Class <-
ifelse(datatemp5$Consumption_per_squaremeter>datatemp5$Base_hr_usage,"High","
Low")
datatemp5$Base_Hour_Class <- as.vector(datatemp5$Base_Hour_Class)
class(daily_base_hr_usage_dataset)
write.csv(datatemp5,file="Midterm_version2.csv")
datatemp5 <- plyr::rename(datatemp5, c("hour"="Hour"))
datatemp5 <- plyr::rename(datatemp5, c("X..address"="Address"))


MyData <- read.csv(file="TempFinal.csv", header=TRUE, sep=",")
MyData <- plyr::rename(MyData, c("X..address"="Address"))
datatemp6 <- merge(x = datatemp5, y = MyData, by =
c("building","Date","Hour","Address"), all.x = TRUE)
write.csv(datatemp6,file="Midterm_version3.csv")
```

6. This is the function to retrieve the geolocation from the address

```
url <- function(address, return.call = "json", sensor = "false") {
 root <- "http://maps.google.com/maps/api/geocode/"
 u <- paste(root, return.call, "?address=", address, "&sensor=", sensor, sep = "")
 return(URLencode(u))
}
geoCode <- function(address,verbose=FALSE) {
 if(verbose) cat(address,"\n")
 u <- url(address)
 doc <- getURL(u)
 x <- fromJSON(doc,simplify = FALSE)
 if(x$status=="OK") {
```

```
    lat <- x$results[[1]]$geometry$location$lat
    lng <- x$results[[1]]$geometry$location$lng
    location_type  <- x$results[[1]]$geometry$location_type
    formatted_address  <- x$results[[1]]$formatted_address
    return(c(lat, lng, location_type, formatted_address))
    Sys.sleep(1)
  } else {
    return(c(NA,NA,NA, NA))
  }
```

7. . Fetch the geolocation data for the given addresses using the above function

```
i =1
addresses1 = NULL
while(i <= 10)
{
  #if(i %% 9 == 0) Sys.sleep(3)
  addresses1 = c(addresses1,findata[i,2])
  i = i+1
}
locations1  <- ldply(addresses1, function(x) geoCode(x))
addresses2 = NULL
while(i > 10 && i<=20)
{
  #if(i %% 9 == 0) Sys.sleep(3)
  addresses2 = c(addresses2,findata[i,2])
  i = i+1
}
locations2  <- ldply(addresses2, function(x) geoCode(x))
addresses3 = NULL
while(i > 20 && i<=30)
{
  #if(i %% 9 == 0) Sys.sleep(3)
  addresses3 = c(addresses3,findata[i,2])
  i = i+1
}
locations3  <- ldply(addresses3, function(x) geoCode(x))
addresses4 = NULL
while(i > 30 && i<=33)
{
  #if(i %% 9 == 0) Sys.sleep(3)
```

```
  addresses4 = c(addresses4,findata[i,2])
  i = i+1
}
locations4  <- ldply(addresses4, function(x) geoCode(x))
locations=NULL
locations <- rbind(locations1,locations2,locations3,locations4)
names(locations)  <- c("lat","lon","location_type", "formatted")
#Geolocation data retrieved for all the specified addresses
```

8. Code to get the nearest airport for each Building

```
i <- 1
lat <- NULL
lon <- NULL
xml.url1 <- data.frame(NULL,stringsAsFactors = FALSE)
xml.url <- data.frame(NULL,stringsAsFactors = FALSE)
locdata <- data.frame(NULL,stringsAsFactors = FALSE)
locdata.t <- data.frame(NULL,stringsAsFactors = FALSE)

#function to calculate the minimum geolocation distance
earth.dist <- function (long1, lat1, long2, lat2)
{
  rad <- pi/180
  a1 <- lat1 * rad
  a2 <- long1 * rad
  b1 <- lat2 * rad
  b2 <- long2 * rad
  dlon <- b2 - a2
  dlat <- b1 - a1
  a <- (sin(dlat/2))^2 + cos(a1) * cos(b1) * (sin(dlon/2))^2
  c <- 2 * atan2(sqrt(a), sqrt(1 - a))
  R <- 6378.145
  d <- R * c
  return(d)
}
airport <- c("")
airportlat <- c("")
airportlon <- c("")
airportdata <- data.frame(airport,airportlat,airportlon,stringsAsFactors = FALSE)
while(i <= nrow(totaldata))
```

```
{
  lat <- totaldata$lat[i]
  lon <- totaldata$lon[i]

  xml.url1[i,1] <-
paste("http://api.wunderground.com/auto/wui/geo/GeoLookupXML/index.xml?query=
",lat,sep = "")
  xml.url[i,1] <- paste(xml.url1[i,1],lon,sep=",")
  xmlfile <- xmlParse(xml.url[i,1])
  xmltop = xmlRoot(xmlfile)
  xmlfileresult <- t(xmlSApply(xmltop[["nearby_weather_stations"]][["airport"]],
xmlValue))
  nodes <- getNodeSet(xmltop,"//airport/station")
  locdata<- as.data.frame(lapply(nodes, function(x) xmlSApply(x,
xmlValue)),stringsAsFactors = FALSE)
  locdata.t <- t(locdata)
  locdata.t <- data.frame(locdata.t,stringsAsFactors = FALSE)
  locdata.t <- locdata.t[!locdata.t$icao=="",]
  locdata.t$lat <- as.numeric(locdata.t$lat)
  locdata.t$lon <- as.numeric(locdata.t$lon)
  nearest <- locdata.t[which.min(earth.dist(lon,lat,locdata.t$lon,locdata.t$lat)),]
  nearest <- nearest[,c(4:6)]
  names(nearest) <- c("airport","airportlat","airportlon")
  airportdata <- rbind(airportdata,nearest)
  i <- i+1
}
airportdata <- airportdata[-1,]
```

9.. **Fetch weather data for the airports**

```
install.packages("weatherData")
library(weatherData)
install.packages("dplyr")
library(dplyr)
install.packages("zoo")
library(zoo)

Date<-as.data.frame(seq(as.Date("2013/1/1"), as.Date("2013/12/31"),by = "days"))
Date<-as.data.frame(Date[rep(row.names(Date),times=24),])
names(Date)<-c("Date")
Date<-as.data.frame(Date[order(as.Date(Date$Date, format="%Y/%m/%d")),])
names(Date)<-c("Date")
```

```
Hour<-as.data.frame(c(rep(0:23,times=365)))
names(Hour)<-c("Hour")
df1<-cbind(Date,Hour)

#etch unique values for the airports in the Bilding data
airports <- data.frame(unique(airportdata$airport),stringsAsFactors = FALSE)
datalist = list()

#Retrive weather data for each airport
i=1
while(i <= nrow(airports))
{
  d3<- getWeatherForDate(airports[i,1], start_date="2013-01-01",
                    end_date = "2013-12-31",
                    opt_detailed = TRUE,opt_custom_columns = T,
                    custom_columns=c(1,2,3,4,5,6,7,8,9,10,11,12,13))
  #head(d3)
  dates <- format(as.POSIXct(strptime(d3$Time,"%Y-%m-%d %H:%M:%S",tz="")) ,format
= "%m/%d/%Y")
  hours <- format(as.POSIXct(strptime(d3$Time,"%Y-%m-%d %H:%M:%S",tz="")) ,format
= "%H")
  d3$Date <- dates
  d3$Date<- as.Date(d3$Date,format = "%m/%d/%Y")
  d3$Hour <- as.numeric(hours)
  d3$Humidity <- as.numeric(d3$Humidity) #NAs introduced
  d3$Wind_SpeedMPH <- as.numeric(d3$Wind_SpeedMPH) #NAs introduced
  # replace all NAs with the previous value
  na.locf(d3$Wind_SpeedMPH)
  d3 <- d3[,-c(1,2)]
  d3 = d3 %>% group_by(Date,Hour) %>%
summarize(TemperatureF=mean(TemperatureF),
                              Dew_PointF=mean(Dew_PointF),
                              Humidity=mean(Humidity),
                              Sea_Level_PressureIn = mean(Sea_Level_PressureIn),
                              VisibilityMPH = mean(VisibilityMPH),
                              Wind_SpeedMPH = mean(Wind_SpeedMPH),
                              WindDirDegrees = mean(WindDirDegrees),
                              Conditions = max(Conditions),
                              Wind_Direction = max(Wind_Direction),
                              Gust_SpeedMPH = max(Gust_SpeedMPH),
                              PrecipitationIn = max(PrecipitationIn),
                              Events = max(Events))

  d3 <- merge(df1,d3,by=c("Date","Hour"),all.x = TRUE)
```

```
 d3$Airport <- airports[i,1]

 d3$TemperatureF <- na.locf(d3$TemperatureF)
 d3$Dew_PointF <- na.locf(d3$Dew_PointF)
 d3$Humidity <- na.locf(d3$Humidity)
 d3$Sea_Level_PressureIn <- na.locf(d3$Sea_Level_PressureIn)
 d3$VisibilityMPH <- na.locf(d3$VisibilityMPH)
 d3$Wind_SpeedMPH <- na.locf(d3$Wind_SpeedMPH)
 d3$WindDirDegrees <- na.locf(d3$WindDirDegrees)
 d3$Conditions <- na.locf(d3$Conditions)
 d3$Wind_Direction <- na.locf(d3$Wind_Direction)
 d3$Gust_SpeedMPH <- na.locf(d3$Gust_SpeedMPH)
 d3$PrecipitationIn <- na.locf(d3$PrecipitationIn)
 d3$Events <- na.locf(d3$Events)

 datalist[[i]] <- d3

 i <- i+1
}
```

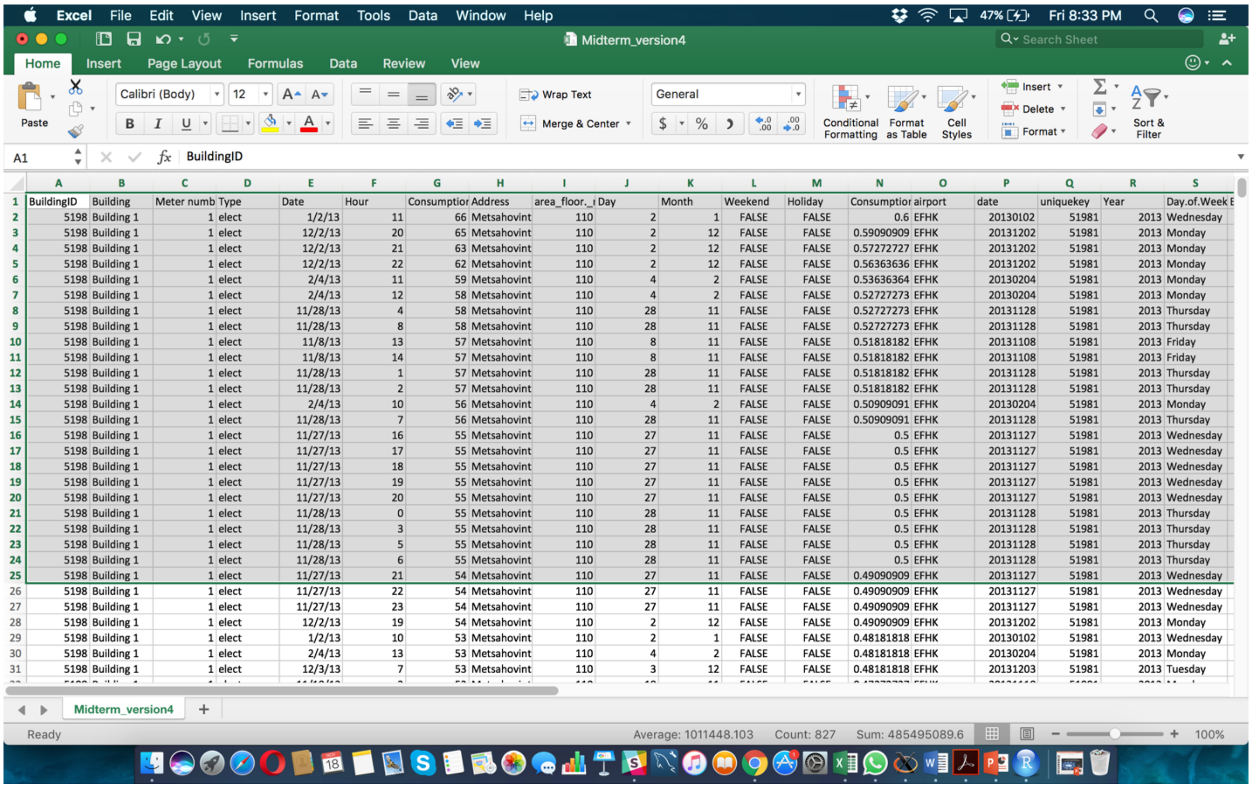
10 **merge weather data for all unique airports corresponding to the building data**

```
big_data = do.call(rbind, datalist)
write.csv(big_data,"complete_weather_data.csv")
```

Output after Data wrangling and ingestion

Team 5 **Midterm**

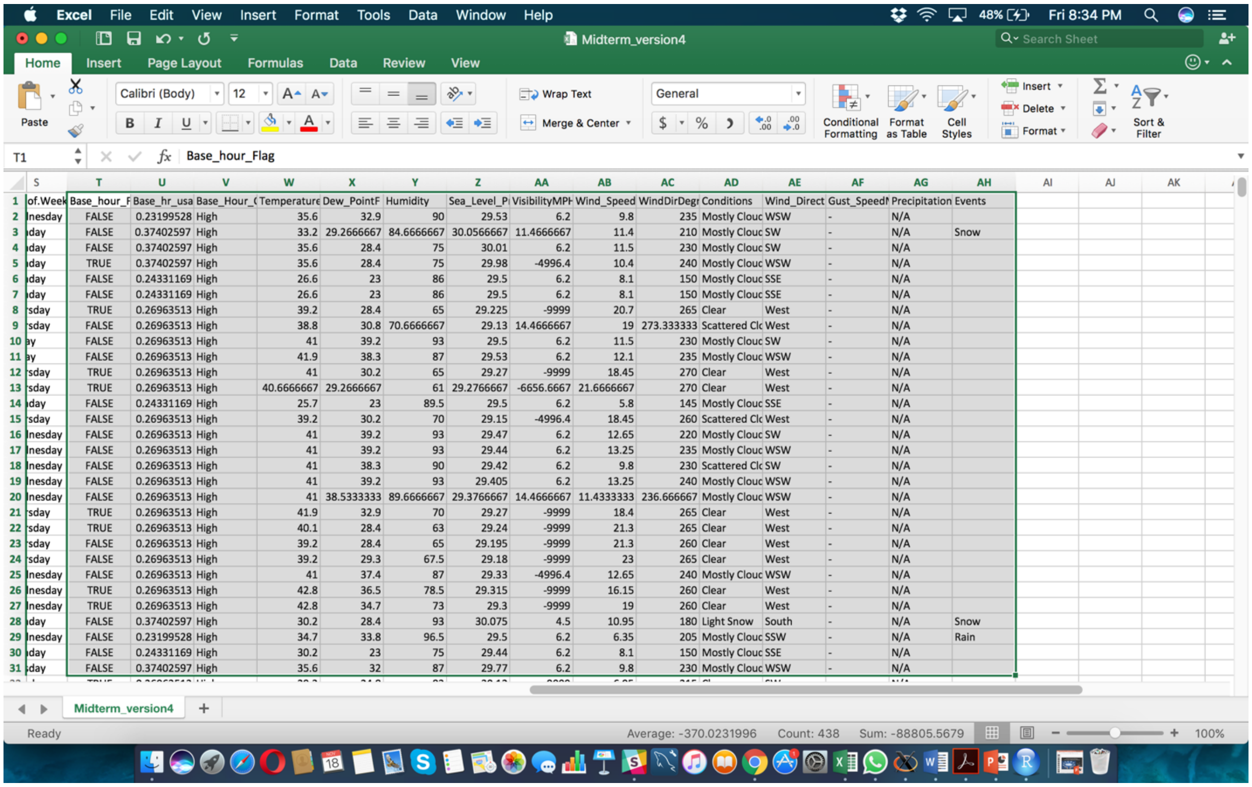
<u>OUTPUT</u>



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	BuildingID	Building	Meter numb	Type	Date	Hour	Consumptior	Address	area_floor._ı	Day	Month	Weekend	Holiday	Consumptior	airport	date	uniquekey	Year	Day.of.Week
2	5198	Building 1	1	elect	1/2/13	11	66	Metsahovint	110	2	1	FALSE	FALSE	0.6	EFHK	20130102	51981	2013	Wednesday
3	5198	Building 1	1	elect	12/2/13	20	65	Metsahovint	110	2	12	FALSE	FALSE	0.59090909	EFHK	20131202	51981	2013	Monday
4	5198	Building 1	1	elect	12/2/13	21	63	Metsahovint	110	2	12	FALSE	FALSE	0.57272727	EFHK	20131202	51981	2013	Monday
5	5198	Building 1	1	elect	12/2/13	22	62	Metsahovint	110	2	12	FALSE	FALSE	0.56363636	EFHK	20131202	51981	2013	Monday
6	5198	Building 1	1	elect	2/4/13	11	59	Metsahovint	110	4	2	FALSE	FALSE	0.53636364	EFHK	20130204	51981	2013	Monday
7	5198	Building 1	1	elect	2/4/13	12	58	Metsahovint	110	4	2	FALSE	FALSE	0.52727273	EFHK	20130204	51981	2013	Monday
8	5198	Building 1	1	elect	11/28/13	4	58	Metsahovint	110	28	11	FALSE	FALSE	0.52727273	EFHK	20131128	51981	2013	Thursday
9	5198	Building 1	1	elect	11/28/13	8	58	Metsahovint	110	28	11	FALSE	FALSE	0.52727273	EFHK	20131128	51981	2013	Thursday
10	5198	Building 1	1	elect	11/8/13	13	57	Metsahovint	110	8	11	FALSE	FALSE	0.51818182	EFHK	20131108	51981	2013	Friday
11	5198	Building 1	1	elect	11/8/13	14	57	Metsahovint	110	8	11	FALSE	FALSE	0.51818182	EFHK	20131108	51981	2013	Friday
12	5198	Building 1	1	elect	11/28/13	1	57	Metsahovint	110	28	11	FALSE	FALSE	0.51818182	EFHK	20131128	51981	2013	Thursday
13	5198	Building 1	1	elect	11/28/13	2	57	Metsahovint	110	28	11	FALSE	FALSE	0.51818182	EFHK	20131128	51981	2013	Thursday
14	5198	Building 1	1	elect	2/4/13	10	56	Metsahovint	110	4	2	FALSE	FALSE	0.50909091	EFHK	20130204	51981	2013	Monday
15	5198	Building 1	1	elect	11/28/13	7	56	Metsahovint	110	28	11	FALSE	FALSE	0.50909091	EFHK	20131128	51981	2013	Thursday
16	5198	Building 1	1	elect	11/27/13	16	55	Metsahovint	110	27	11	FALSE	FALSE	0.5	EFHK	20131127	51981	2013	Wednesday
17	5198	Building 1	1	elect	11/27/13	17	55	Metsahovint	110	27	11	FALSE	FALSE	0.5	EFHK	20131127	51981	2013	Wednesday
18	5198	Building 1	1	elect	11/27/13	18	55	Metsahovint	110	27	11	FALSE	FALSE	0.5	EFHK	20131127	51981	2013	Wednesday
19	5198	Building 1	1	elect	11/27/13	19	55	Metsahovint	110	27	11	FALSE	FALSE	0.5	EFHK	20131127	51981	2013	Wednesday
20	5198	Building 1	1	elect	11/27/13	20	55	Metsahovint	110	27	11	FALSE	FALSE	0.5	EFHK	20131127	51981	2013	Wednesday
21	5198	Building 1	1	elect	11/28/13	0	55	Metsahovint	110	28	11	FALSE	FALSE	0.5	EFHK	20131128	51981	2013	Thursday
22	5198	Building 1	1	elect	11/28/13	3	55	Metsahovint	110	28	11	FALSE	FALSE	0.5	EFHK	20131128	51981	2013	Thursday
23	5198	Building 1	1	elect	11/28/13	5	55	Metsahovint	110	28	11	FALSE	FALSE	0.5	EFHK	20131128	51981	2013	Thursday
24	5198	Building 1	1	elect	11/28/13	6	55	Metsahovint	110	28	11	FALSE	FALSE	0.5	EFHK	20131128	51981	2013	Thursday
25	5198	Building 1	1	elect	11/27/13	21	54	Metsahovint	110	27	11	FALSE	FALSE	0.49090909	EFHK	20131127	51981	2013	Wednesday
26	5198	Building 1	1	elect	11/27/13	22	54	Metsahovint	110	27	11	FALSE	FALSE	0.49090909	EFHK	20131127	51981	2013	Wednesday
27	5198	Building 1	1	elect	11/27/13	23	54	Metsahovint	110	27	11	FALSE	FALSE	0.49090909	EFHK	20131127	51981	2013	Wednesday
28	5198	Building 1	1	elect	12/2/13	19	54	Metsahovint	110	2	12	FALSE	FALSE	0.49090909	EFHK	20131202	51981	2013	Monday
29	5198	Building 1	1	elect	1/2/13	10	53	Metsahovint	110	2	1	FALSE	FALSE	0.48181818	EFHK	20130102	51981	2013	Wednesday
30	5198	Building 1	1	elect	2/4/13	13	53	Metsahovint	110	4	2	FALSE	FALSE	0.48181818	EFHK	20130204	51981	2013	Monday
31	5198	Building 1	1	elect	12/3/13	7	53	Metsahovint	110	3	12	FALSE	FALSE	0.48181818	EFHK	20131203	51981	2013	Tuesday

Dipti Pamnani **Jayesh Samyani** **Suraj Sharma**

Team 5 **Midterm**



	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH
1	of.Week	Base_hour_F	Base_hr_usa	Base_Hour_(	Temperature	Dew_PointF	Humidity	Sea_Level_Pı	VisibilityMPH	Wind_Speed	WindDirDegı	Conditions	Wind_Direct	Gust_SpeedM	Precipitation	Events
2	lnesday	FALSE	0.23199528	High	35.6	32.9	90	29.53	6.2	9.8	235	Mostly Clouc	WSW	-	N/A	
3	lday	FALSE	0.37402597	High	33.2	29.2666667	84.6666667	30.0566667	11.4666667	11.4	210	Mostly Clouc	SW	-	N/A	Snow
4	lday	FALSE	0.37402597	High	35.6	28.4	75	30.01	6.2	11.5	230	Mostly Clouc	SW	-	N/A	
5	lday	TRUE	0.37402597	High	35.6	28.4	75	29.98	-4996.4	10.4	240	Mostly Clouc	WSW	-	N/A	
6	lday	FALSE	0.24331169	High	26.6	23	86	29.5	6.2	8.1	150	Mostly Clouc	SSE	-	N/A	
7	lday	FALSE	0.24331169	High	26.6	23	86	29.5	6.2	8.1	150	Mostly Clouc	SSE	-	N/A	
8	rsday	TRUE	0.26963513	High	39.2	28.4	65	29.225	-9999	20.7	265	Clear	West	-	N/A	
9	rsday	FALSE	0.26963513	High	38.8	30.8	70.6666667	29.13	14.4666667	19	273.333333	Scattered Clc	West	-	N/A	
10	ay	FALSE	0.26963513	High	41	39.2	93	29.5	6.2	11.5	230	Mostly Clouc	SW	-	N/A	
11	ay	FALSE	0.26963513	High	41.9	38.3	87	29.53	6.2	12.1	235	Mostly Clouc	WSW	-	N/A	
12	rsday	TRUE	0.26963513	High	41	30.2	65	29.27	-9999	18.45	270	Clear	West	-	N/A	
13	rsday	TRUE	0.26963513	High	40.6666667	29.2666667	61	29.2766667	-6656.6667	21.6666667	270	Clear	West	-	N/A	
14	lday	FALSE	0.24331169	High	25.7	23	89.5	29.5	6.2	5.8	145	Mostly Clouc	SSE	-	N/A	
15	rsday	FALSE	0.26963513	High	39.2	30.2	70	29.15	-4996.4	18.45	260	Scattered Clc	West	-	N/A	
16	lnesday	FALSE	0.26963513	High	41	39.2	93	29.47	6.2	12.65	220	Mostly Clouc	SW	-	N/A	
17	lnesday	FALSE	0.26963513	High	41	39.2	93	29.44	6.2	13.25	235	Mostly Clouc	WSW	-	N/A	
18	lnesday	FALSE	0.26963513	High	41	38.3	90	29.42	6.2	9.8	230	Scattered Clc	SW	-	N/A	
19	lnesday	FALSE	0.26963513	High	41	39.2	93	29.405	6.2	13.25	240	Mostly Clouc	WSW	-	N/A	
20	lnesday	FALSE	0.26963513	High	41	38.5333333	89.6666667	29.3766667	14.4666667	11.4333333	236.666667	Mostly Clouc	WSW	-	N/A	
21	rsday	TRUE	0.26963513	High	41.9	32.9	70	29.27	-9999	18.4	265	Clear	West	-	N/A	
22	rsday	TRUE	0.26963513	High	40.1	28.4	63	29.24	-9999	21.3	265	Clear	West	-	N/A	
23	rsday	FALSE	0.26963513	High	39.2	28.4	65	29.195	-9999	21.3	260	Clear	West	-	N/A	
24	rsday	FALSE	0.26963513	High	39.2	29.3	67.5	29.18	-9999	23	265	Clear	West	-	N/A	
25	lnesday	FALSE	0.26963513	High	41	37.4	87	29.33	-4996.4	12.65	240	Mostly Clouc	WSW	-	N/A	
26	lnesday	TRUE	0.26963513	High	42.8	36.5	78.5	29.315	-9999	16.15	260	Clear	West	-	N/A	
27	lnesday	TRUE	0.26963513	High	42.8	34.7	73	29.3	-9999	19	260	Clear	West	-	N/A	
28	lday	FALSE	0.37402597	High	30.2	28.4	93	30.075	4.5	10.95	180	Light Snow	South	-	N/A	Snow
29	lnesday	FALSE	0.23199528	High	34.7	33.8	96.5	29.5	6.2	6.35	205	Mostly Clouc	SSW	-	N/A	Rain
30	lday	FALSE	0.24331169	High	30.2	23	75	29.44	6.2	8.1	150	Mostly Clouc	SSE	-	N/A	
31	lday	FALSE	0.37402597	High	35.6	32	87	29.77	6.2	9.8	230	Mostly Clouc	WSW	-	N/A	

Dipti Pamnani **Jayesh Samyani** **Suraj Sharma**

Part 2: Modeling tasks.

1. Prediction

Linear Regression: In statistics, **linear regression** is an approach for modeling the relationship between a scalar dependent variable *y* and one or more explanatory(or independent variables) denoted *X*. The case of one explanatory variable is called *simple linear regression*. For more than one explanatory variable, the process is called *multiple linear regression*

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

To find performance matrix for 78 models. we have to find unique 78 dataset buildings based on building Id and Meter number

CODE:

Steps followed are techniques

1. Extract 78 unique combinations
2. Build 78 Models
3. Select each model and perform splitting of the data into 70% training and 30 % testing
4. Train the model using training data
5. Predict the results on testdata
6. Evaluate Performance matrix of model

1. Extract 78 unique combinations

```
unique.models <- unique(df1$uniquekey)
unique.models <- as.numeric(levels(unique.models))[unique.models]
unique.models <- sort(unique.models)

N<-length(unique.models)
list_dataFrames <- vector("list", N)
list_models <- vector("list", N)
```

2. Build 78 Models

```
j <- 0
for(i in unique.models){
        j <- j+1
        assign(paste('dfmodel', j, sep=''), df1[df1$uniquekey==i,])
        tempdf<-df1[df1$uniquekey==i,]
        list_dataFrames[[j]]<-tempdf

}
names(list_dataFrames) <- paste("dfmodel", 1:N, sep = "")
i<-0
str(dfmodel1)
for (name in names(list_dataFrames)) {
        i<-i+1
        print(i)
        # if(i!=4&&i!=6&&i!=7){
        data.frame.model <- list_dataFrames[[name]]
        data.frame.model[["uniquekey"]] <- NULL

       str(dfmodel1)
```

3.Select each model and perform splitting of the data into 70% training and 30 % testing

```
        sample <- sample.split (data.frame.model, SplitRatio= 0.7)
        data.frame.train <-subset(data.frame.model,sample==TRUE)
        data.frame.test <- subset(data.frame.model,sample==FALSE)
```

4. Train the model using training data

```
        linearmodel <-lm(Consumption_per_squaremeter ~ .,data = data.frame.train)
```

5.Predict the results on testdata

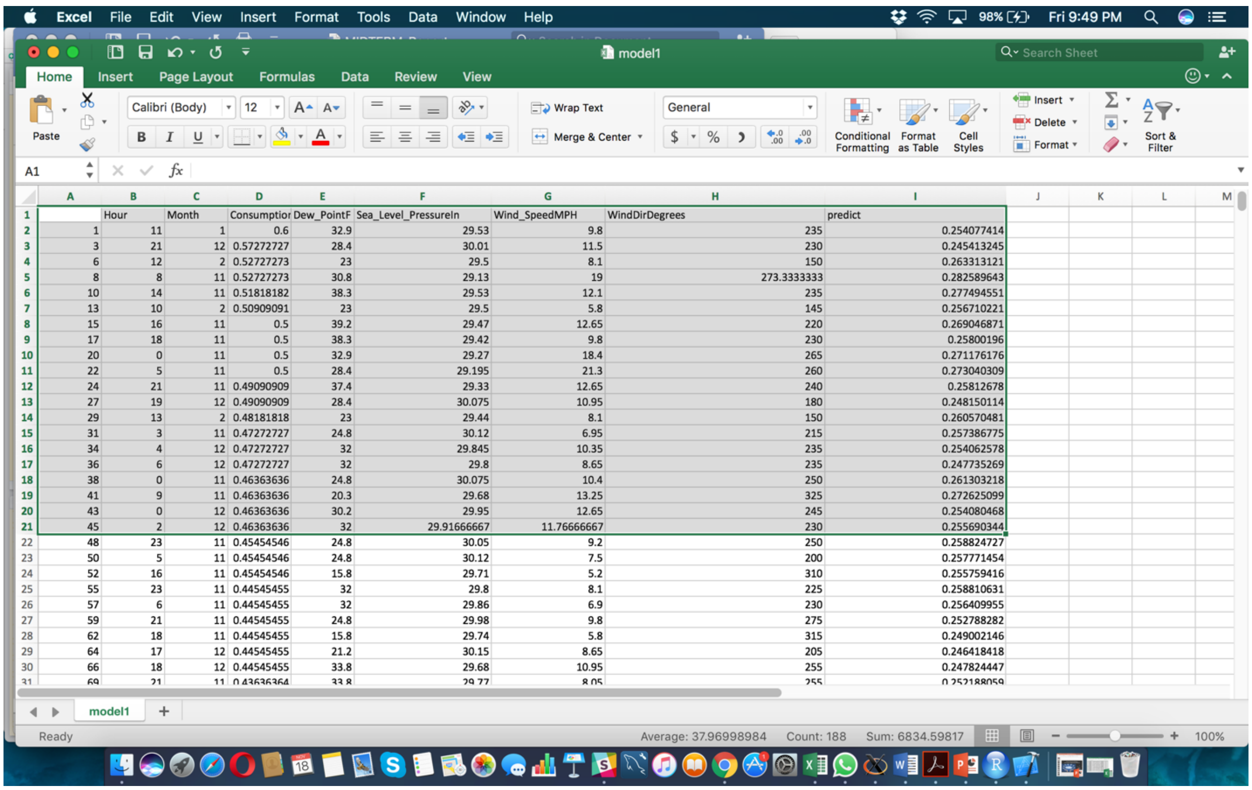
```
data.frame.test$predict <- predict(linearmodel,newdata=data.frame.test)
summary(linearmodel)
coef(linearmodel)
```

5. Evaluate Performance matrix of model

```
accuracyresult <-
as.data.frame(accuracy(data.frame.test$predict,data.frame.test$Consumption_per_squaremeter))

write.csv(data.frame.test,file =
paste0("Prediction/LinearRegression/outputcsv/model",i,".csv"))

tocsv <- accuracyresult
#
 write.table(tocsv, file =
paste0("Prediction/LinearRegression/PerformanceMatrix",i,".csv"),row.names=FALSE, na="",
sep=",",append = TRUE)

}
```

Output CSV:

Predict column determines consumption_per_sqauremeter for each row.



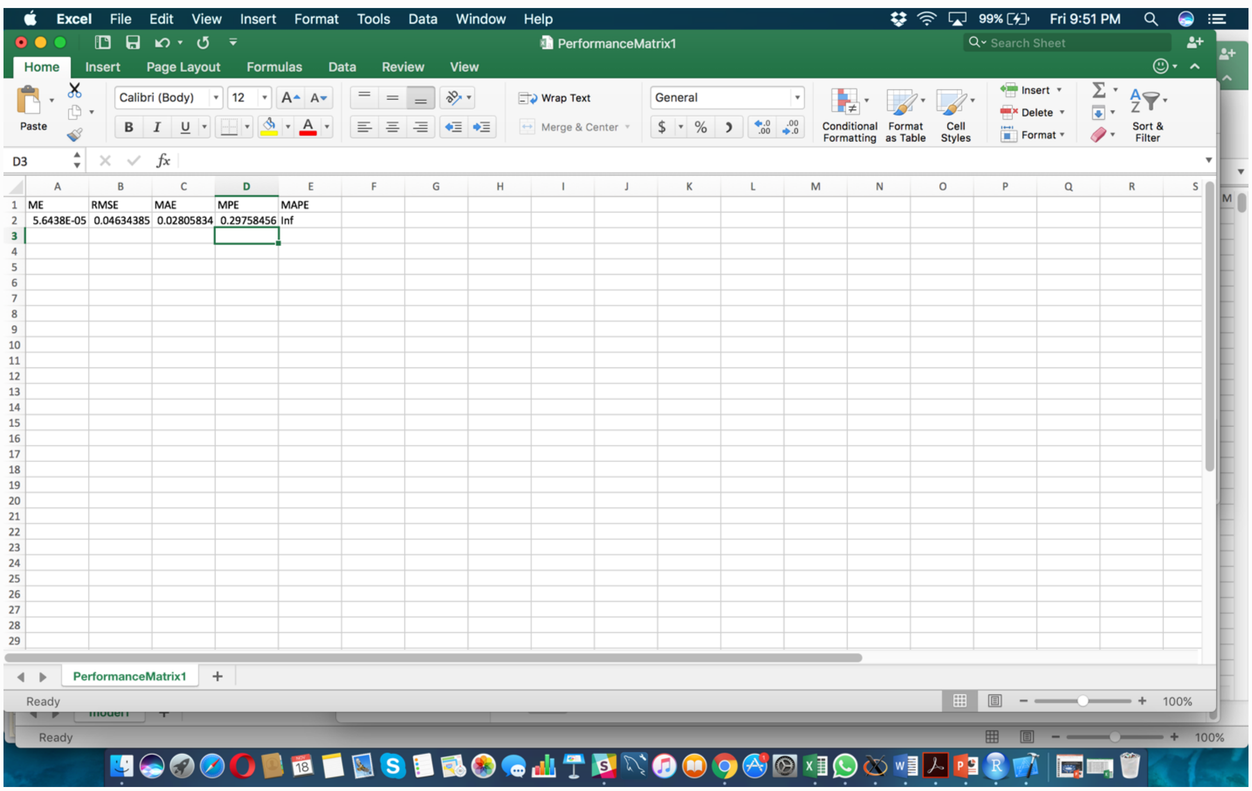
	Hour	Month	Consumptior	Dew_PointF	Sea_Level_PressureIn	Wind_SpeedMPH	WindDirDegrees	predict
1	11	1	0.6	32.9	29.53	9.8	235	0.254077414
3	21	12	0.57272727	28.4	30.01	11.5	230	0.245413245
6	12	2	0.52727273	23	29.5	8.1	150	0.263313121
8	8	11	0.52727273	30.8	29.13	19	273.3333333	0.282589643
10	14	11	0.51818182	38.3	29.53	12.1	235	0.277494551
13	10	2	0.50909091	23	29.5	5.8	145	0.256710221
15	16	11	0.5	39.2	29.47	12.65	220	0.269046871
17	18	11	0.5	38.3	29.42	9.8	230	0.25800196
20	0	11	0.5	32.9	29.27	18.4	265	0.271176176
22	5	11	0.5	28.4	29.195	21.3	260	0.273040309
24	21	11	0.49090909	37.4	29.33	12.65	240	0.25812678
27	19	12	0.49090909	28.4	30.075	10.95	180	0.248150114
29	13	2	0.48181818	23	29.44	8.1	150	0.260570481
31	3	11	0.47272727	24.8	30.12	6.95	215	0.257386775
34	4	12	0.47272727	32	29.845	10.35	235	0.254062578
36	6	12	0.47272727	32	29.8	8.65	235	0.247735269
38	0	11	0.46363636	24.8	30.075	10.4	250	0.261303218
41	9	11	0.46363636	20.3	29.68	13.25	325	0.272625099
43	0	12	0.46363636	30.2	29.95	12.65	245	0.254080468
45	2	12	0.46363636	32	29.91666667	11.76666667	230	0.255690344
48	23	11	0.45454546	24.8	30.05	9.2	250	0.258824727
50	5	11	0.45454546	24.8	30.12	7.5	200	0.257771454
52	16	11	0.45454546	15.8	29.71	5.2	310	0.255759416
55	23	11	0.44545455	32	29.8	8.1	225	0.258810631
57	6	11	0.44545455	32	29.86	6.9	230	0.256409955
59	21	11	0.44545455	24.8	29.98	9.8	275	0.252788282
62	18	11	0.44545455	15.8	29.74	5.8	315	0.249002146
64	17	12	0.44545455	21.2	30.15	8.65	205	0.246418418
66	18	12	0.44545455	33.8	29.68	10.95	255	0.247824447
69	21	11	0.43636364	33.8	29.77	8.05	255	0.252188059

Dipti Pamnani **Jayesh Samyani** **Suraj Sharma**

Performance Matrix (Accuracy):

All the error parameters are less .

RMSE is less which determines that variance of prediction and actual values are less and hence the accuracy for the given model is quite high



1. <u>Linear Regression</u>:

<u>Linear Regression</u>: In statistics, **linear regression** is an approach for modeling the relationship between a scalar dependent variable *y* and one or more explanatory(or independent variables) denoted *X*. The case of one explanatory variable is called *simple linear regression*. For more than one explanatory variable, the process is called *multiple linear regression*

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

To find performance matrix for 78 models. we have to find unique 78 dataset buildings based on building Id and Meter number

Steps followed are techniques

1. Extract 78 unique combinations
2. Build 78 Models
3. Select each model and perform splitting of the data into 70% training and 30 % testing
4. Train the model using training data
5. Predict the results on testdata
6. Evaluate Performance matrix of model

Prediction:

This algorithm is a supervised learning algorithm, where the destination is known, but the path to the destination is not. It does not create a model; instead it creates predictions from close data on-demand when a prediction is required. A similarity measure (such as Euclidean distance) is used to locate close data in order to make predictions.

Prediction for our dataset:

We have to predict the consumption (KWH/sq. m) for each building (heat and electricity)

Steps Performed:

1) **Get the Data**

   The dataset that we generated in step 1 is used. This dataset contains the complete data for the building (Heat and Electricity Consumption, Nearest Airport, Building Area, Base Hour Usage, and Weather Conditions)

2) **Know the Data**

   Now that the data is loaded into RStudio, we try to get a thorough understanding of what the data is about, and what each column signifies.

   The initial overview of the data shows that the each building is associated with different meters for calculating the heat and electricity consumption, for each hour of the day for the year 2013.

   We have a unique key column, which assigns a key for each building's meter. (78 unique ids)

   Thus, this column is of key importance to divide the entire dataset into 78 different subsets for the purpose of predicting and classifying the data.

3) **Prepare the Data**

   Now we prepare the data for our model.

```
df1<-read.csv("Midterm_version4.csv")
str(df1)

df1$uniquekey <- as.factor(df1$uniquekey)

df1[["date"]] <- NULL
df1[["Gust_SpeedMPH"]] <- NULL
df1[["PrecipitationIn"]]<-NULL
df1[["Events"]] <- NULL
```

   The data from the csv is stored in data frame df1. The uniquekey column is converted from a numeric value to a factor column in order to get the unique data in the column.

   We then make the date column (repeated column in the dataset) ,Gust_SpeedMPH, Precipation and Events column are removed as maximum number of rows in the dataset for these columns are NULL. So, these columns will not help much in our model.

```
unique.models <- unique(df1$uniquekey)
unique.models <- as.numeric(levels(unique.models))[unique.models]
unique.models <- sort(unique.models)
N<-length(unique.models)
j <- 0
for(i in unique.models){
  j <- j+1
  assign(paste('dfmodel', j, sep=''), df1[df1$uniquekey==i,])

}
```

The above piece of code makes 78 different models based on the uniquekey column.

```
k= 1
while(k <= N)
{
  dfnames <- paste("dfmodel", 1:N, sep = "")
  k = k+1
}
```

We then store the names for each model created in a list called “dfnames”, so that we can iterate through the names and retrieve data for that particular model.

```
for (name in dfnames) {
  i<-i+1
  print(i)
  print(name)
  data.frame.model <- get(name)
set.seed(101)

data.frame.model <- data.frame.model[,-c(1:5)] #Remove buildingID,building,meter number,type,date
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)] #Remove address and area

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)] #Remove airport, unique key, year

data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)


abc <- model.matrix(~.+0, data=data.frame.model)

data.frame.model_n <- as.data.frame(abc)

sample <- sample.split (data.frame.model_n, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model_n,sample==TRUE)
data.frame.model_test <- subset(data.frame.model_n,sample==FALSE)

fit <- knnreg(data.frame.model_train[,-7],data.frame.model_train[,7],k=3)
predictions <- predict(fit, data.frame.model_test[,-7])
acc <- accuracy(predictions,data.frame.model_test$Consumption_per_squaremeter)
print(acc)

}
```

- In the above piece of code we iterate through all 78 models by taking the name stored in the dfnames list.
- We then remove the columns that will not help in predicting the value.

For example, the building ID, building, meter number and type of meter columns have same data for a particular model (The basis of segregating the model)

- We then convert the columns like Hour, Month, Day, Weekend, and Holiday to factor type, so that we can use this in an efficient manner for the model.

- KNN model works with numeric data, thus to convert our data frame to have numeric data, we convert the data frame to a matrix.

```
abc <- model.matrix(~.+0, data=data.frame.model)
```

- This line of code converts the data frame to a numeric matrix by splitting each factor into a column and assigning 1 wherever TRUE for that case.
- Split the data into training and testing datasets.

4) Prepare the Model

```
fit <- knnreg(data.frame.model_train[,-7],data.frame.model_train[,7],k=3)
predictions <- predict(fit, data.frame.model_test[,-7])
acc <- accuracy(predictions,data.frame.model_test$Consumption_per_squaremeter)
print(acc)
```

Here we prepare a model to predict the consumption_per_squaremeter column with k=3. We chose k=3 as we observe better values for accuracy with k=3. The accuracy measures did not differ much with k=4, hence we kept the k value to 3.

Classification:
To predict the Base_Hour_Class flag for the dataset, i.e. we need to classify whether the corresponding data would fall under “High” class or “Low” class. Based on our predicted classification, we then check the accuracy of the model b comparing it with the actual values.
Steps/Procedure:

1) Step 1 and Step 2 remain the same as we did for Prediction model.

- We read the data from the csv and split the dataset into 78 models based on the uniquekey column
- Factor the columns and convert the data frame to matrix so that the factor data can be used more efficiently with the numeric data, instead of just considering the numeric data to build the classification model.

```
data.frame.model <- data.frame.model[,-c(1:5)]
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)]

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)]
data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)
```

Converting the data into matrix

```
data.frame.model_train_n <- model.matrix(~.+0, data=data.frame.model_train)
data.frame.model_test_n <- model.matrix(~.+0, data=data.frame.model_test)

data.frame.model_train_n <- as.data.frame(data.frame.model_train_n)
data.frame.model_test_n <- as.data.frame(data.frame.model_test_n)
```

2) Prepare the data

```
set.seed(101)

sample <- sample.split (data.frame.model, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model,sample==TRUE)
data.frame.model_test <- subset(data.frame.model,sample==FALSE)
```

Split the data into training and testing dataset.

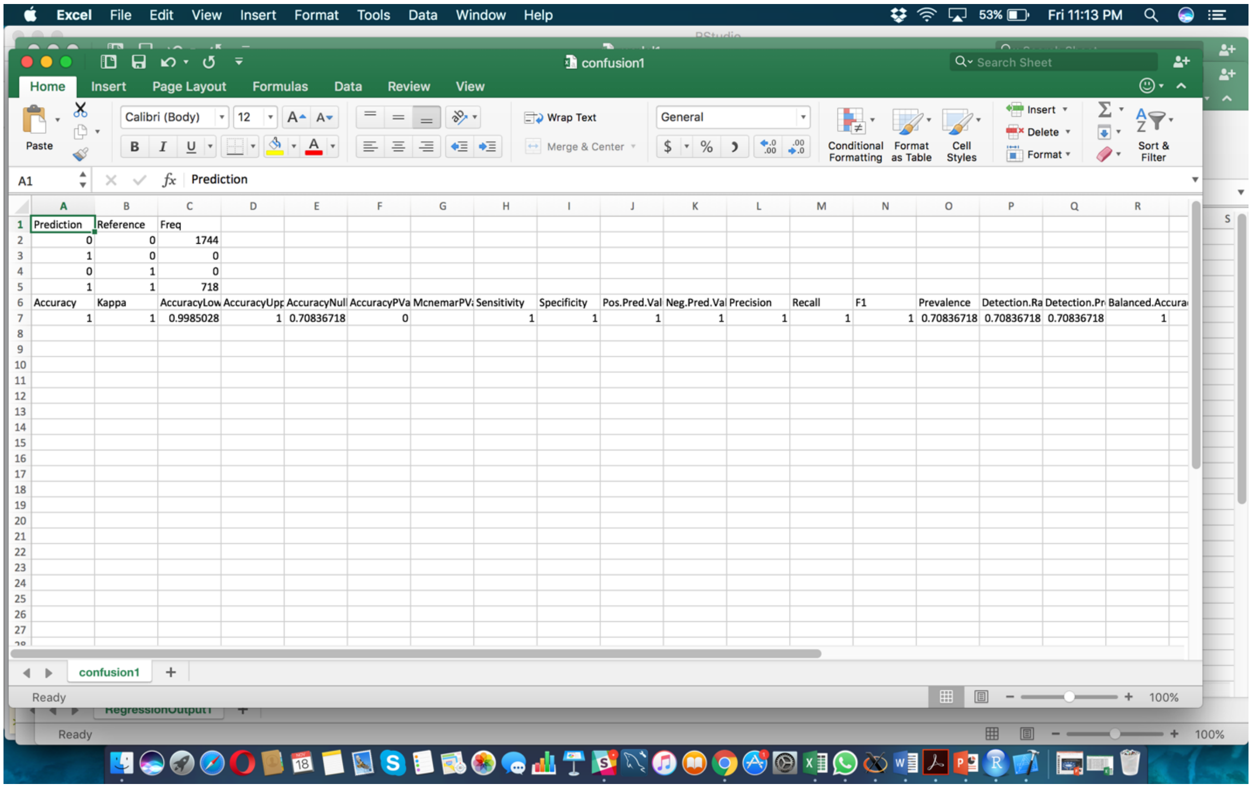
For classification model, we need to specify which the classification column. We do so by specifying labels. In our data set we wish to predict on the column Consumption per Square meter which is column 11, hence the data frames for training and testing labels are constructed which can now be used for building a model.

```
data.frame.model_train_labels <- data.frame.model[1:nrow(data.frame.model_train_n), 11]
data.frame.model_test_labels <- data.frame.model[1:nrow(data.frame.model_test_n), 11]
```

3) Build the Model

```
linearmodel <-lm(Consumption_per_squaremeter ~ .,data = data.frame.train)

data.frame.test$predict <- predict(linearmodel,newdata=data.frame.test)
summary(linearmodel)
coef(linearmodel)
```

4. Confusion matrix



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Prediction	Reference	Freq															
2	0	0	1744															
3	1	0	0															
4	0	1	0															
5	1	1	718															
6	Accuracy	Kappa	AccuracyLow	AccuracyUpp	AccuracyNull	AccuracyPVa	McnemarPVa	Sensitivity	Specificity	Pos.Pred.Val	Neg.Pred.Val	Precision	Recall	F1	Prevalence	Detection.Ra	Detection.Pr	Balanced.Accura
7	1	1	0.9985028	1	0.70836718	0		1	1	1	1	1	1	1	0.70836718	0.70836718	0.70836718	1

CODE:

1 Extract 78 unique combinations

```
unique.models <- unique(df1$uniquekey)
unique.models <- as.numeric(levels(unique.models))[unique.models]
unique.models <- sort(unique.models)

N<-length(unique.models)
list_dataFrames <- vector("list", N)
list_models <- vector("list", N)
```

2 Build 78 Models

```
j <- 0
for(i in unique.models){
      j <- j+1
      assign(paste('dfmodel', j, sep=''), df1[df1$uniquekey==i,])
      tempdf<-df1[df1$uniquekey==i,]
      list_dataFrames[[j]]<-tempdf
```

```
}
names(list_dataFrames) <- paste("dfmodel", 1:N, sep = "")
i<-0
str(dfmodel1)
for (name in names(list_dataFrames)) {
    i<-i+1
    print(i)
    # if(i!=4&&i!=6&&i!=7){
    data.frame.model <- list_dataFrames[[name]]
    data.frame.model[["uniquekey"]] <- NULL

   str(dfmodel1)
```

3.Select each model and perform splitting of the data into 70% training and 30 % testing

```
    sample <- sample.split (data.frame.model, SplitRatio= 0.7)
    data.frame.train <-subset(data.frame.model,sample==TRUE)
    data.frame.test <- subset(data.frame.model,sample==FALSE)
```

4. Train the model using training data

```
    linearmodel <-lm(Consumption_per_squaremeter ~ .,data = data.frame.train)
```

5.Predict the results on testdata

```
    data.frame.test$predict <- predict(linearmodel,newdata=data.frame.test)
    summary(linearmodel)
    coef(linearmodel)
```

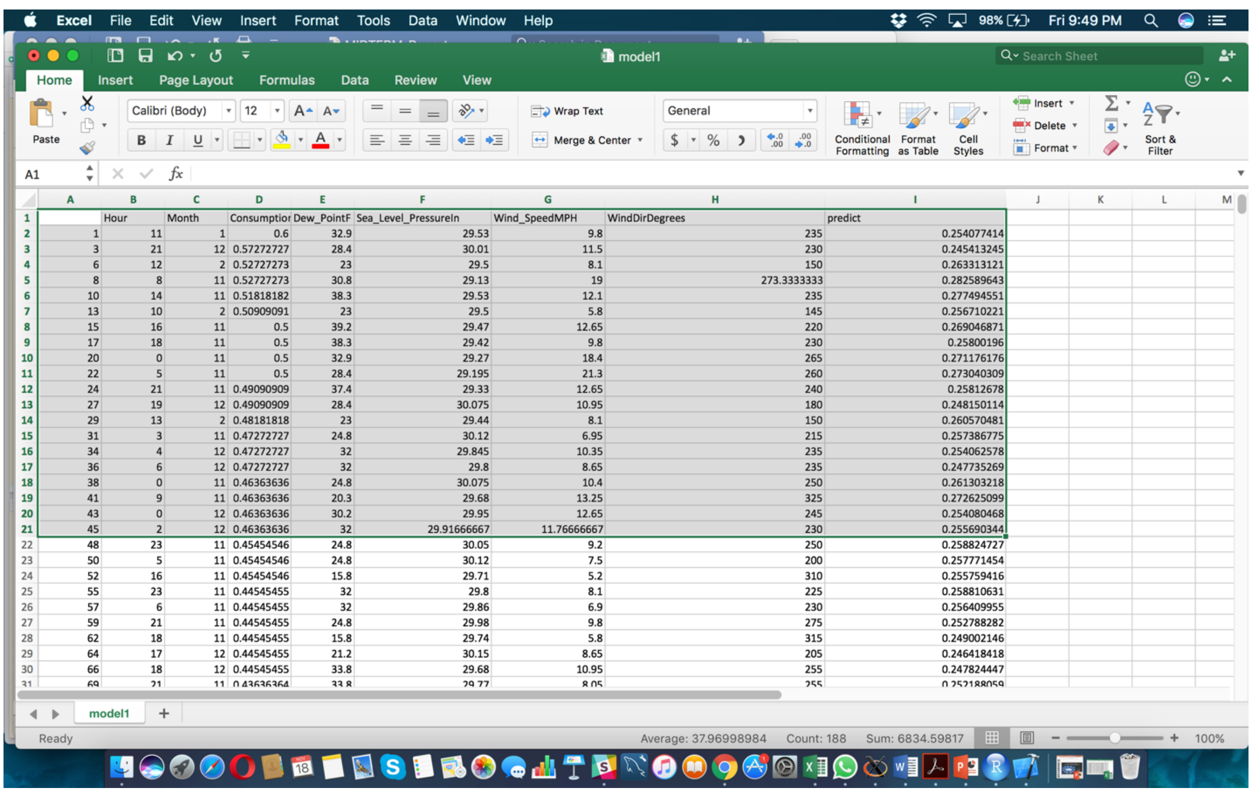
6 Evaluate Performance matrix of model

```
   accuracyresult <-
as.data.frame(accuracy(data.frame.test$predict,data.frame.test$Consumption_per_squaremeter))

    write.csv(data.frame.test,file =
paste0("Prediction/LinearRegression/outputcsv/model",i,".csv"))

    tocsv <- accuracyresult
    #
```

```
write.table(tocsv, file =
paste0("Prediction/LinearRegression/PerformanceMatrix",i,".csv"),row.names=FALSE, na="",
sep=",",append = TRUE)

}
```

Output CSV:

Predict column determines consumption_per_sqauremeter for each row.

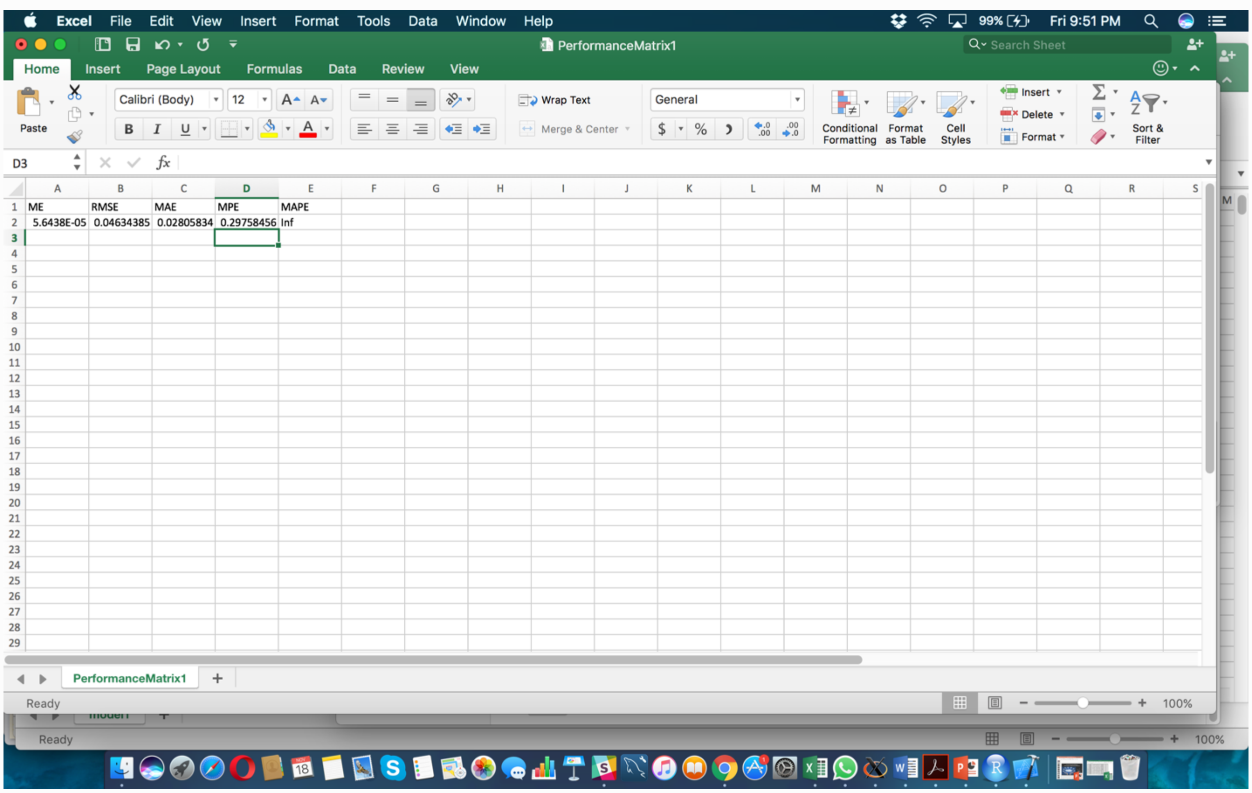


	Hour	Month	Consumptior	Dew_PointF	Sea_Level_PressureIn	Wind_SpeedMPH	WindDirDegrees	predict
1	11	1	0.6	32.9	29.53	9.8	235	0.254077414
3	21	12	0.57272727	28.4	30.01	11.5	230	0.245413245
6	12	2	0.52727273	23	29.5	8.1	150	0.263313121
8	8	11	0.52727273	30.8	29.13	19	273.3333333	0.282589643
10	14	11	0.51818182	38.3	29.53	12.1	235	0.277494551
13	10	2	0.50909091	23	29.5	5.8	145	0.256710221
15	16	11	0.5	39.2	29.47	12.65	220	0.269046871
17	18	11	0.5	38.3	29.42	9.8	230	0.25800196
20	0	11	0.5	32.9	29.27	18.4	265	0.271176176
22	5	11	0.5	28.4	29.195	21.3	260	0.273040309
24	21	11	0.49090909	37.4	29.33	12.65	240	0.25812678
27	19	12	0.49090909	28.4	30.075	10.95	180	0.248150114
29	13	2	0.48181818	23	29.44	8.1	150	0.260570481
31	3	11	0.47272727	24.8	30.12	6.95	215	0.257386775
34	4	12	0.47272727	32	29.845	10.35	235	0.254062578
36	6	12	0.47272727	32	29.8	8.65	235	0.247735269
38	0	11	0.46363636	24.8	30.075	10.4	250	0.261303218
41	9	11	0.46363636	20.3	29.68	13.25	325	0.272625099
43	0	12	0.46363636	30.2	29.95	12.65	245	0.254080468
45	2	12	0.46363636	32	29.91666667	11.76666667	230	0.255690344
48	23	11	0.45454546	24.8	30.05	9.2	250	0.258824727
50	5	11	0.45454546	24.8	30.12	7.5	200	0.257771454
52	16	11	0.45454546	15.8	29.71	5.2	310	0.255759416
55	23	11	0.44545455	32	29.8	8.1	225	0.258810631
57	6	11	0.44545455	32	29.86	6.9	230	0.256409955
59	21	11	0.44545455	24.8	29.98	9.8	275	0.252788282
62	18	11	0.44545455	15.8	29.74	5.8	315	0.249002146
64	17	12	0.44545455	21.2	30.15	8.65	205	0.246418418
66	18	12	0.44545455	33.8	29.68	10.95	255	0.247824447
69	21	11	0.43636364	33.8	29.77	8.05	255	0.252188059

Performance Matrix (Accuracy):

All the error parameters are less .

RMSE is less which determines that variance of prediction and actual values are less and hence the accuracy for the given model is quite high



2 . KNN (k-Nearest Neighbor) Algorithm

Prediction:
This algorithm is a supervised learning algorithm, where the destination is known, but the path to the destination is not. It does not create a model; instead it creates predictions from close data on-demand when a prediction is required. A similarity measure (such as Euclidean distance) is used to locate close data in order to make predictions.

Prediction for our dataset:
We have to predict the consumption (KWH/sq. m) for each building (heat and electricity)

Steps Performed:

1) **Get the Data**

   The dataset that we generated in step 1 is used. This dataset contains the complete data for the building (Heat and Electricity Consumption, Nearest Airport, Building Area, Base Hour Usage, and Weather Conditions)
2) **Know the Data**

   Now that the data is loaded into RStudio, we try to get a thorough understanding of what the data is about, and what each column signifies.
   The initial overview of the data shows that the each building is associated with different meters for calculating the heat and electricity consumption, for each hour of the day for the year 2013.
   We have a unique key column, which assigns a key for each building's meter. (78 unique ids)
   Thus, this column is of key importance to divide the entire dataset into 78 different subsets for the purpose of predicting and classifying the data.
3) **Prepare the Data**

   Now we prepare the data for our model.

```
df1<-read.csv("Midterm_version4.csv")
str(df1)

df1$uniquekey <- as.factor(df1$uniquekey)

df1[["date"]] <- NULL
df1[["Gust_SpeedMPH"]] <- NULL
df1[["PrecipitationIn"]]<-NULL
df1[["Events"]] <- NULL
```

   The data from the csv is stored in data frame df1. The uniquekey column is converted from a numeric value to a factor column in order to get the unique data in the column.

We then make the date column (repeated column in the dataset) ,Gust_SpeedMPH, Precipation and Events column are removed as maximum number of rows in the dataset for these columns are NULL. So, these columns will not help much in our model.

```
unique.models <- unique(df1$uniquekey)
unique.models <- as.numeric(levels(unique.models))[unique.models]
unique.models <- sort(unique.models)
N<-length(unique.models)
j <- 0
for(i in unique.models){
  j <- j+1
  assign(paste('dfmodel', j, sep=''), df1[df1$uniquekey==i,])

}
```

The above piece of code makes 78 different models based on the uniquekey column.

```
k= 1
while(k <= N)
{
  dfnames <- paste("dfmodel", 1:N, sep = "")
  k = k+1
}
```

We then store the names for each model created in a list called "dfnames", so that we can iterate through the names and retrieve data for that particular model.

```
for (name in dfnames) {
  i<-i+1
  print(i)
  print(name)
  data.frame.model <- get(name)
set.seed(101)

data.frame.model <- data.frame.model[,-c(1:5)] #Remove buildingID,building,meter number,type,date
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)] #Remove address and area

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)] #Remove airport, unique key, year

data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)


abc <- model.matrix(~.+0, data=data.frame.model)

data.frame.model_n <- as.data.frame(abc)

sample <- sample.split (data.frame.model_n, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model_n,sample==TRUE)
data.frame.model_test <- subset(data.frame.model_n,sample==FALSE)

fit <- knnreg(data.frame.model_train[,-7],data.frame.model_train[,7],k=3)
predictions <- predict(fit, data.frame.model_test[,-7])
acc <- accuracy(predictions,data.frame.model_test$Consumption_per_squaremeter)
print(acc)

}
```

- In the above piece of code we iterate through all 78 models by taking the name stored in the dfnames list.
- We then remove the columns that will not help in predicting the value.

For example, the building ID, building, meter number and type of meter columns have same data for a particular model (The basis of segregating the model)

- We then convert the columns like Hour, Month, Day, Weekend, and Holiday to factor type, so that we can use this in an efficient manner for the model.
- KNN model works with numeric data, thus to convert our data frame to have numeric data, we convert the data frame to a matrix.

```
abc <- model.matrix(~.+0, data=data.frame.model)
```

- This line of code converts the data frame to a numeric matrix by splitting each factor into a column and assigning 1 wherever TRUE for that case.
- Split the data into training and testing datasets.

4) **Prepare the Model**

```
fit <- knnreg(data.frame.model_train[,-7],data.frame.model_train[,7],k=3)
predictions <- predict(fit, data.frame.model_test[,-7])
acc <- accuracy(predictions,data.frame.model_test$Consumption_per_squaremeter)
print(acc)
```

Here we prepare a model to predict the consumption_per_squaremeter column with k=3. We chose k=3 as we observe better values for accuracy with k=3. The accuracy measures did not differ much with k=4, hence we kept the k value to 3.

Classification:
To predict the Base_hr_class for the dataset, i.e. we need to classify whether the corresponding data would fall under "High" class or "Low" class. Based on our predicted classification, we then check the accuracy of the model b comparing it with the actual values.
Steps/Procedure:

5) **Step 1 and Step 2 remain the same as we did for Prediction model.**
   - We read the data from the csv and split the dataset into 78 models based on the uniquekey column
   - Factor the columns and convert the data frame to matrix so that the factor data can be used more efficiently with the numeric data, instead of just considering the numeric data to build the classification model.

```
data.frame.model <- data.frame.model[,-c(1:5)]
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)]

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)]
data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)
```

Converting the data into matrix

```
data.frame.model_train_n <- model.matrix(~.+0, data=data.frame.model_train)
data.frame.model_test_n <- model.matrix(~.+0, data=data.frame.model_test)

data.frame.model_train_n <- as.data.frame(data.frame.model_train_n)
data.frame.model_test_n <- as.data.frame(data.frame.model_test_n)
```

6) Prepare the data

```
set.seed(101)

sample <- sample.split (data.frame.model, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model,sample==TRUE)
data.frame.model_test <- subset(data.frame.model,sample==FALSE)
```

Split the data into training and testing dataset.
For classification model, we need to specify which the classification column. We do so by specifying labels. In our data set we wish to classify on the column Base_hr_class which is column 11, hence the data frames for training and testing labels are constructed which can now be used for building a model.

```
data.frame.model_train_labels <- data.frame.model[1:nrow(data.frame.model_train_n), 11]
data.frame.model_test_labels <- data.frame.model[1:nrow(data.frame.model_test_n), 11]
```

7) Build the Model

```
data.frame.model_test_pred <- knn(train = data.frame.model_train, test = data.frame.model_test,
                                  cl = data.frame.model_train_labels, k=3)


table = CrossTable(x= data.frame.model_test_labels, y=data.frame.model_test_pred,prop.chisq = FALSE)
accuracy = mean(data.frame.model_test_pred == data.frame.model$Base_Hour_Class[data.frame.model_test_labels])
print(accuracy)
```

We build the classification model to determine the Base_hr_class and then we check the accuracy of our prediction by making a CrossTable.

```
                             | data.frame.model_test_pred
data.frame.model_test_labels |      High |       Low | Row Total |
-----------------------------|-----------|-----------|-----------|
                        High |      1507 |       604 |      2111 |
                             |     0.714 |     0.286 |     0.858 |
                             |     0.905 |     0.759 |           |
                             |     0.612 |     0.245 |           |
-----------------------------|-----------|-----------|-----------|
                         Low |       158 |       192 |       350 |
                             |     0.451 |     0.549 |     0.142 |
                             |     0.095 |     0.241 |           |
                             |     0.064 |     0.078 |           |
-----------------------------|-----------|-----------|-----------|
                Column Total |      1665 |       796 |      2461 |
                             |     0.677 |     0.323 |           |
-----------------------------|-----------|-----------|-----------|
```

This is the cross table generated for each model.

Here 1507 out of 2461 records have been accurately predicted (TN->True Negatives) as High class, Also 192 out of 2461 were accurately predicted (TP-> True Positives).
Thus, a total of 1699 out of 2461 have been predicted correctly.
There were 158 records of False Negatives meaning that these records which belong to Low class but have been predicted as High class.
Similarly there were 604 cases of False Positives (FP) meaning 604 records which belong to High class have been predicted as Low class.The accuracy of the model is calculated as (1507+192)/2461 ~ 69%

3 . Neural Network

Prediction:

Neural Network can be defined as a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'.

Prediction for our dataset:

We have to predict the consumption (KWH/sq. m) for each building (heat and electricity)

Steps Performed:

1) **Get the Data**

   The dataset that we generated in step 1 is used. This dataset contains the complete data for the building (Heat and Electricity Consumption, Nearest Airport, Building Area, Base Hour Usage, and Weather Conditions)

2) **Know the Data**

   Now that the data is loaded into RStudio, we try to get a thorough understanding of what the data is about, and what each column signifies.

   The initial overview of the data shows that the each building is associated with different meters for calculating the heat and electricity consumption, for each hour of the day for the year 2013.

   We have a unique key column, which assigns a key for each building's meter. (78 unique ids)

   Thus, this column is of key importance to divide the entire dataset into 78 different subsets for the purpose of predicting and classifying the data.

3) **Prepare the Data**

   Now we prepare the data for our model.

```
df1<-read.csv("Midterm_version4.csv")
str(df1)

df1$uniquekey <- as.factor(df1$uniquekey)

df1[["date"]] <- NULL
df1[["Gust_SpeedMPH"]] <- NULL
df1[["PrecipitationIn"]]<-NULL
df1[["Events"]] <- NULL
```

   The data from the csv is stored in data frame df1. The uniquekey column is converted from a numeric value to a factor column in order to get the unique data in the column.

   We then make the date column (repeated column in the dataset) ,Gust_SpeedMPH, Precipation and Events column are removed as maximum number of rows in the dataset for these columns are NULL. So, these columns will not help much in our model.

```
unique.models <- unique(df1$uniquekey)
unique.models <- as.numeric(levels(unique.models))[unique.models]
unique.models <- sort(unique.models)
N<-length(unique.models)
j <- 0
for(i in unique.models){
  j <- j+1
  assign(paste('dfmodel', j, sep=''), df1[df1$uniquekey==i,])

}
```

The above piece of code makes 78 different models based on the uniquekey column.

```
k= 1
while(k <= N)
{
  dfnames <- paste("dfmodel", 1:N, sep = "")
  k = k+1
}
```

We then store the names for each model created in a list called "dfnames", so that we can iterate through the names and retrieve data for that particular model.

```
for (name in dfnames) {
  i<-i+1
  print(i)
  print(name)
  data.frame.model <- get(name)
set.seed(101)

data.frame.model <- data.frame.model[,-c(1:5)] #Remove buildingID,building,meter number,type,date
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)] #Remove address and area

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)] #Remove airport, unique key, year

data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)


abc <- model.matrix(~.+0, data=data.frame.model)

data.frame.model_n <- as.data.frame(abc)

sample <- sample.split (data.frame.model_n, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model_n,sample==TRUE)
data.frame.model_test <- subset(data.frame.model_n,sample==FALSE)

fit <- knnreg(data.frame.model_train[,-7],data.frame.model_train[,7],k=3)
predictions <- predict(fit, data.frame.model_test[,-7])
acc <- accuracy(predictions,data.frame.model_test$Consumption_per_squaremeter)
print(acc)

}
```

- In the above piece of code we iterate through all 78 models by taking the name stored in the dfnames list.
- We then remove the columns that will not help in predicting the value.

For example, the building ID, building, meter number and type of meter columns have same data for a particular model (The basis of segregating the model)

- We then convert the columns like Hour, Month, Day, Weekend, and Holiday to factor type, so that we can use this in an efficient manner for the model.

- KNN model works with numeric data, thus to convert our data frame to have numeric data, we convert the data frame to a matrix.

```
abc <- model.matrix(~.+0, data=data.frame.model)
```

- This line of code converts the data frame to a numeric matrix by splitting each factor into a column and assigning 1 wherever TRUE for that case.
- Split the data into training and testing datasets.

4) **Prepare the Model**

```
fit <- knnreg(data.frame.model_train[,-7],data.frame.model_train[,7],k=3)
predictions <- predict(fit, data.frame.model_test[,-7])
acc <- accuracy(predictions,data.frame.model_test$Consumption_per_squaremeter)
print(acc)
```

Here we prepare a model to predict the consumption_per_squaremeter column with k=3. We chose k=3 as we observe better values for accuracy with k=3. The accuracy measures did not differ much with k=4, hence we kept the k value to 3.

Classification:

To predict the Consumption per squaremeter flag for the dataset, i.e. we need to classify whether the corresponding data would fall under "High" class or "Low" class. Based on our predicted classification, we then check the accuracy of the model b comparing it with the actual values.

Steps/Procedure:

5) **Step 1 and Step 2 remain the same as we did for Prediction model.**
   - We read the data from the csv and split the dataset into 78 models based on the uniquekey column
   - Factor the columns and convert the data frame to matrix so that the factor data can be used more efficiently with the numeric data, instead of just considering the numeric data to build the classification model.

```
data.frame.model <- data.frame.model[,-c(1:5)]
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)]

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)]
data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)
```

Converting the data into matrix

```
data.frame.model_train_n <- model.matrix(~.+0, data=data.frame.model_train)
data.frame.model_test_n <- model.matrix(~.+0, data=data.frame.model_test)

data.frame.model_train_n <- as.data.frame(data.frame.model_train_n)
data.frame.model_test_n <- as.data.frame(data.frame.model_test_n)
```

6) Prepare the data

```
set.seed(101)

sample <- sample.split (data.frame.model, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model,sample==TRUE)
data.frame.model_test <- subset(data.frame.model,sample==FALSE)
```

Split the data into training and testing dataset.
For classification model, we need to specify which the classification column. We do so by specifying labels. In our data set we wish to classify on the column Consumption per squaremeter which is column 11, hence the data frames for training and testing labels are constructed which can now be used for building a model.

```
data.frame.model_train_labels <- data.frame.model[1:nrow(data.frame.model_train_n), 11]
data.frame.model_test_labels <- data.frame.model[1:nrow(data.frame.model_test_n), 11]
```

7) Build the Model

```
f <-  Base_hour_Flag ~  Meter.number + Hour  + Consumption + area_floor._m.sqr.x + Day +
Month + Weekend + Holiday   + Consumption_per_squaremeter + Base_hr_usage


nn <- neuralnet(f,data=train,hidden=c(3,2),linear.output=FALSE)
```

We build the classification model to determine the Base Hr Flag and then we check the accuracy of our prediction by making a CrossTable.

4. Logistic Regression

Classification:

Logistic regression is a regression model where the dependent variable (DV) is categorical. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. The important point here to note is that in linear regression, the expected values of the response variable are modeled based on combination of values taken by the predictors. In logistic regression Probability or Odds of the response taking a particular value is modeled based on combination of values taken by the predictors.

To predict the Base_Hour_Class flag for the dataset, i.e. we need to classify whether the corresponding data would fall under “High” class or “Low” class. Based on our predicted classification, we then check the accuracy of the model b comparing it with the actual values.
Steps/Procedure:

4) **Step 1 and Step 2 remain the same as we did for Prediction model.**
   - We read the data from the csv and split the dataset into 78 models based on the uniquekey column
   - Factor the columns and convert the data frame to matrix so that the factor data can be used more efficiently with the numeric data, instead of just considering the numeric data to build the classification model.

```
data.frame.model <- data.frame.model[,-c(1:5)]
data.frame.model$Hour <- as.factor(data.frame.model$Hour)
data.frame.model <- data.frame.model[,-c(3,4)]

data.frame.model$Day <- as.factor(data.frame.model$Day)
data.frame.model$Month <- as.factor(data.frame.model$Month)
data.frame.model$Weekend <- as.factor(data.frame.model$Weekend)
data.frame.model$Holiday <- as.factor(data.frame.model$Weekend)
data.frame.model <- data.frame.model[,-c(8:10)]
data.frame.model$Base_hour_Flag <- as.numeric(data.frame.model$Base_hour_Flag)
```

Converting the data into matrix

```
data.frame.model_train_n <- model.matrix(~.+0, data=data.frame.model_train)
data.frame.model_test_n <- model.matrix(~.+0, data=data.frame.model_test)

data.frame.model_train_n <- as.data.frame(data.frame.model_train_n)
data.frame.model_test_n <- as.data.frame(data.frame.model_test_n)
```

5) **Prepare the data**

```
set.seed(101)

sample <- sample.split (data.frame.model, SplitRatio= 0.7)
data.frame.model_train <-subset(data.frame.model,sample==TRUE)
data.frame.model_test <- subset(data.frame.model,sample==FALSE)
```

Split the data into training and testing dataset.
For classification model, we need to specify which the classification column. We do so by specifying labels. In our data set we wish to classify on the column Base_Hour_Class which is column 11, hence the data frames for training and testing labels are constructed which can now be used for building a model.

```
data.frame.model_train_labels <- data.frame.model[1:nrow(data.frame.model_train_n), 11]
data.frame.model_test_labels <- data.frame.model[1:nrow(data.frame.model_test_n), 11]
```

6) **Build the Model**

logisticmodel <-glm(Base_Hour_Class ~ .,data = data.frame.train,family=binomial(link="logit"))

We build the classification model to determine the Base_Hour_Class and then we check the accuracy of our prediction by making a CrossTable.

This is the cross table generated for each model.

5 .Random Forest Algorithm:

Random Forest is an ensemble learning based classification and regression technique. It is one of the commonly used predictive modelling and machine learning technique.
In a normal decision tree, one decision tree is built and in a random forest algorithm number of decision trees are built during the process. A vote from each of the decision trees is considered in deciding the final class of a case or an object, this is called ensemble process. This is a democratic process. Since, many decision trees are built and used in a process of Random Forest algorithm, it is called a forest.

1) Random Forest Classification:

The problem asks us to predict the class Base_Hour_Class variable which tells us whether the consumption for hours {0,1,2,3,4,22, 23} is above the mean usage("High") or below the mean usage("Low").

We followed following steps to make class predictions for Base_Hour_Class column.

STEP 1: Install packages and libraries

The following packages and libraries are required to for random forest classification.

```
# 1) Install packages and libraries*************************

install.packages("e1071")
install.packages("caret")
install.packages("ROCR")
install.packages("randomForest")
install.packages("bigrfc")
install.packages("ranger")
install.packages("miscTools")
library(randomForest)
library(caTools)
library(e1071)
library(caret)
library(ROCR)
library(miscTools)
library(dplyr)

#**********************************************************
```

STEP 2: Exploratory analysis on data

Here we structure the data and remove duplicate and columns with NULL values.

```
# 2) Reading data and structuring the data****************

df1<-read.csv("Midterm_version4.csv")

df1$Hour <- as.factor(df1$Hour)
df1$Day <- as.factor(df1$Day)
df1$Month <- as.factor(df1$Month)
df1$uniquekey <- as.factor(df1$uniquekey)
df1$Year <- as.factor(df1$Year)
df1[["date"]] <- NULL
df1[["Gust_SpeedMPH"]] <- NULL
df1[["PrecipitationIn"]]<-NULL
df1[["Events"]] <- NULL

str(df1)

#************************************************************
```

STEP 3: Making unique subsets of data:

In this step we divide data into 78 unique combinations of BuildingID_MeterNo (uniquekey column in our dataset)

```
# Making 78 unique subsets of data based on BuildingID_meter(uniquek

unique.models <- unique(df1$uniquekey)
unique.models <- as.numeric(levels(unique.models))[unique.models]
unique.models <- sort(unique.models)

N<-length(unique.models)

#Making List of data frames which are subset of original***

list_dataFrames <- vector("list", N)
list_models_class <- vector("list", N)
accuracy<-NULL
j <- 0
for(i in unique.models){
  j <- j+1
  assign(paste('dfmodel', j, sep=''), df1[df1$uniquekey==i,])
  tempdf<-df1[df1$uniquekey==i,]
   list_dataFrames[[j]]<-tempdf

 }
 names(list_dataFrames) <- paste("dfmodel", 1:N, sep = "")

 #************************************************************
```

STEP 4: Variable selection for classification:

In this step we extract features based on their importance in creating the model.
Here we take few sample subsets and perform random forest classification taking all features into consideration. Once the model is built we extract important features based on the decreasing Gini values.

GINI: GINI importance measures the average gain of purity by splits of a given variable. If the variable is useful, it tends to split mixed labeled nodes into pure single class nodes.

```
# Running random forest on entire data using all features

fit <- randomForest(Base_Hour_Class ~ .,mtry=floor(sqrt(ncol(data.frame.train))),data.frame.train,ntree

# Plotting the fit
plot(fit)

# Variable Importance Plot
varImpPlot(fit,
           sort = T,
           main="Variable Importance",
           n.var=5)

# Variable Importance Table based on Gini Scores

var.imp <- data.frame(importance(fit,type=2))
```

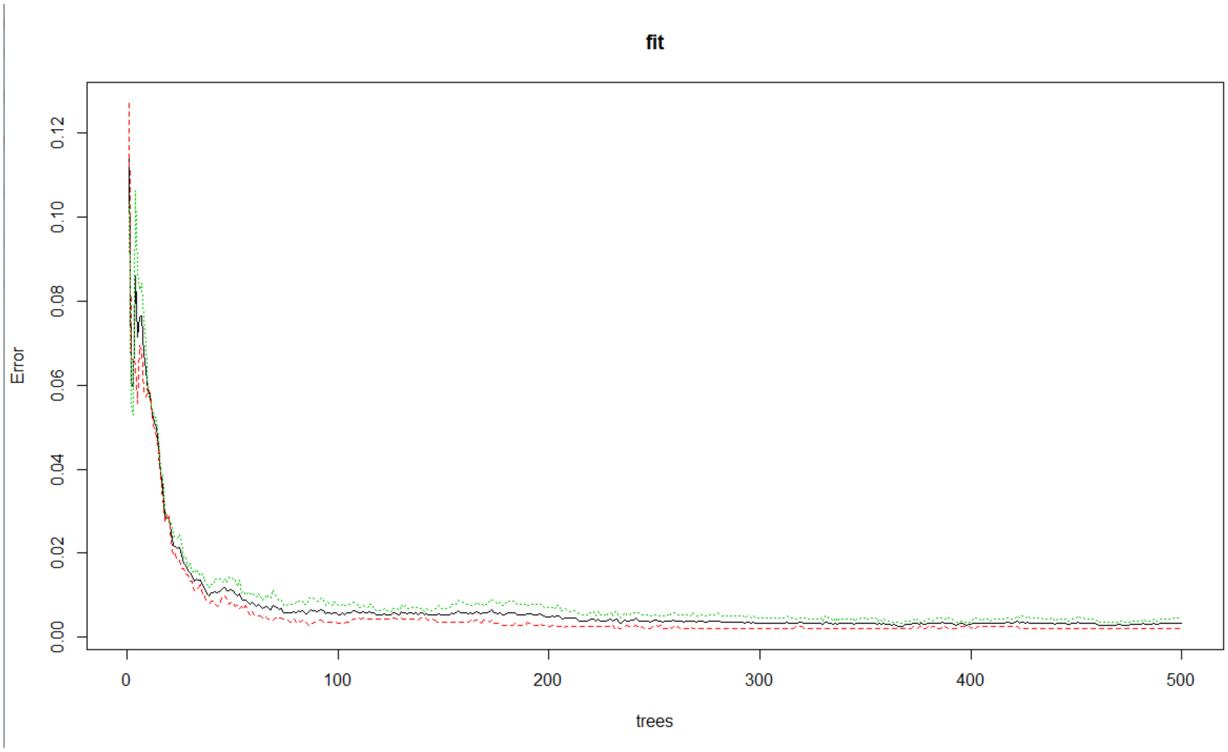
Importance Table:

	MeanDecreaseGini	Variables
1	811.395472	Consumption
2	792.754617	Consumption_per_squaremeter
3	277.105290	Base_hr_usage
4	198.998189	Month
5	158.135987	Day
6	102.864263	Hour
7	63.472388	Wind_Direction
8	49.869569	Dew_PointF
9	48.599827	TemperatureF
10	39.403700	Day.of.Week

As we can see Consumption and Consumption_per_squaremeter are determined very important

We select top 4 features for further modelling.

Plot(fit)



The above plot shows the as we increase the number of trees the above 100 there is no significant decrease in the error.

STEP 5: Iterating over list of subsets and building RF models.
We iterate over the 78 different subsets of data for each building and generate 78 different models with respect to features selected in the above step

```
# Modeling on training data
fit <- randomForest(Base_Hour_Class ~ .,mtry=floor(sqrt(ncol(data.frame.train))),
                    data.frame.train,ntree=500, importance=TRUE,test=data.frame.test)

list_models_class[[i]]<-fit
```

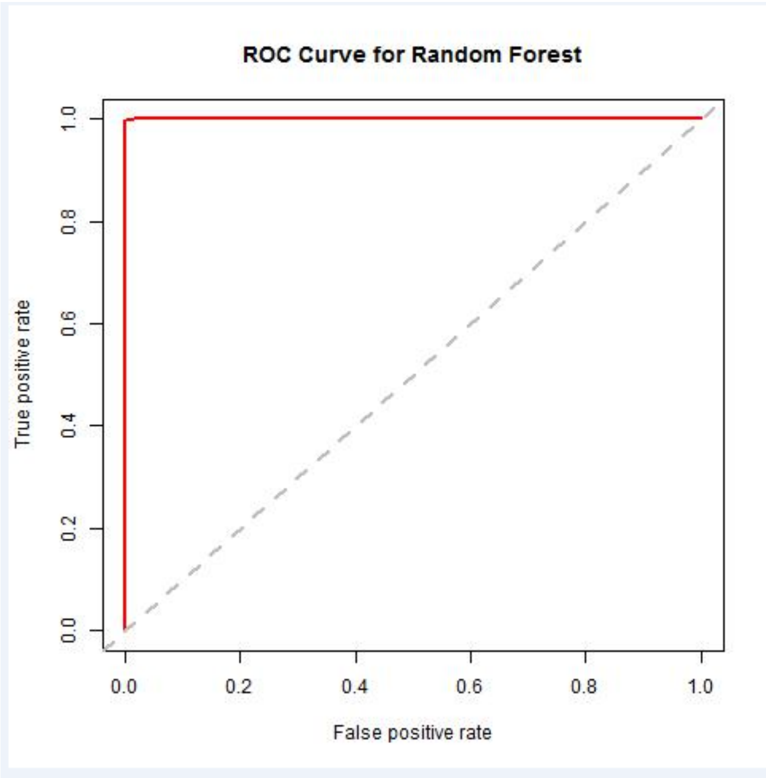
Here we choose **mtry** as the **square root of number of columns** which in our case is 2 by default its 5.
We keep the number of trees as **500.**
STEP 6: Perform prediction for each building
Once the model is built we perform prediction to determine class of each testing dataset

```
#Class prediction on testing data
data.frame.test$predict <- predict(fit,newdata=data.frame.test)
```

Performance Evaluation:
ROC Curves:

```
#Making ROC curves for each model
mypath <- file.path(file = paste0("Classification/randomForest/ROC/ROC",i,".png"))
jpeg(mypath)
par(mfrow=c(1,1))
prediction.obj = prediction(predict.prob,data.frame.test$Base_Hour_Class)
performance.obj = performance(prediction.obj,"tpr","fpr")
plot(performance.obj,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")
dev.off()
}
```

The above code gernates ROC curves which tells us how accurate our model is in prediction. The area under the curve should be as high as possible.



The graph shows the model has almost accurately predicted all the labels. The graph has maximum area under the curve with negligible false positive values.

Predictior	Reference	Freq	
High	High	71	
Low	High	1	
High	Low	1	
Low	Low	2382	
Accuracy	Kappa	AccuracyLower	AccuracyUpper
0.999185	0.985691	0.997060283	0.999901325

We can see only two values were wrongly predicted giving a near perfect model.

STEP 7: Perform prediction on entire dataset.

We perform training and testing on the entire dataset. Here we use **ranger()** function to perform fast random forest modelling on large datasets.

```
# Here we use ranger() function which speeds up random Forest for large datasets
fit<-ranger(Base_Hour_Class~ Consumption+Consumption_per_squaremeter+Base_hr_usage+Month+Day,
            mtry=floor(sqrt(ncol(data.frame.train))),num.trees = 500,data=data.frame.train,write.forest = TRUE)
```

We perform similar performance evaluation on the entire data set.

Predictior	Reference	Freq
High	High	111278
Low	High	22
High	Low	36
Low	Low	88534

We can see we got excellent performance metrics using random forest classification on the entire dataset.
The accuracy measure is **>95%**

2) Random Forest Prediction.

The initial steps for data exploration and developing subsets of data remains the same for prediction.

STEP 1: Varibale selection:

We take few sample data sets for feature selection and sort them on the basis of decreasing importance score

```
> fit$importance
                      IncNodePurity
BuildingID               0.00000000
Building                 0.00000000
Meter.number             0.00000000
Type                     0.00000000
Hour                     0.40418660
Consumption              6.45037232
Address                  0.00000000
area_floor._m.sqr.x      0.00000000
Day                      0.67971571
Month                    1.23406880
Weekend                  0.18905141
Holiday                  0.01715509
airport                  0.00000000
Year                     0.00000000
Day.of.Week              0.24054966
Base_hour_Flag           0.02507942
Base_hr_usage            1.69890729
Base_Hour_Class          1.96460715
TemperatureF             0.27913757
Dew_PointF               0.19441503
Humidity                 0.15447481
Sea_Level_PressureIn     0.20870039
VisibilityMPH            0.04959188
Wind_SpeedMPH            0.12963779
WindDirDegrees           0.13137216
Conditions               0.16502664
Wind_Direction           0.26824020
```

We can see that Consumption , Base_Hour_Class ,Base_Hour_usage and month are important determining features.

STEP 2: Build 78 models with features selected.

```
fit<-randomForest(Consumption_per_squaremeter ~ Consumption+Month+Base_hr_usage,
                  data=data.frame.train[,cols],
                  mtry=floor(sqrt(ncol(data.frame.train[,cols]))),ntree=500)
print(fit)
```

STEP 3: Predict Test Data.

```
data.frame.test$predict <- predict(fit, data.frame.test)
```

STEP 4: Compute Performance Metrics.

Compute MAE,RMSE and MAPE for each building data prediction.

```
r2 <- rSquared(data.frame.test$Consumption_per_squaremeter, data.frame.test$Consumptio

error <- data.frame.test$Consumption_per_squaremeter - data.frame.test$predict

rmse <- sqrt(mean((error)^2))

mae<- mean(abs(error))

mape <- mean(abs(error/data.frame.test$Consumption_per_squaremeter))

performance.metrics <- data.frame(mape,mae,rmse)
```

C	D
mae	rmse
2.36E-05	0.000127

As seen the performance measure for mean absolute error and rmse is very low depicting strong performance of the model.

STEP 5: Perform random forest prediction on entire data set

Similar to classification we use ranger() function to perform random forest prediction on entire dataset. We use 500 trees to perform the prediction.

```
> fit<-ranger(Consumption_per_squaremeter ~ Consumption+Month+Base_hr_usage,
+               mtry=floor(sqrt(ncol(data.frame.train[,cols]))),
+               num.trees = 500,data=data.frame.train[,cols],write.forest = TRUE)
Growing trees.. Progress: 12%. Estimated remaining time: 3 minutes, 39 seconds.
Growing trees.. Progress: 27%. Estimated remaining time: 2 minutes, 48 seconds.
Growing trees.. Progress: 42%. Estimated remaining time: 2 minutes, 11 seconds.
Growing trees.. Progress: 55%. Estimated remaining time: 1 minute, 43 seconds.
```

Performance Evaluation.

R-squared:

```
> fit$r.squared
[1] 0.9987357
```

mae	rmse
0.00024	0.00099

Conclusion:

This algorithm can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing. It has methods for balancing errors in data sets where classes are imbalanced. The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.

5. K-means

Clustering

k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.
The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center.

Build clusters and based on elbow method we can that 4 clusters will be ideal but we can predict good with 2 clusters also.

```
km.out <- kmeans(df2,3,nstart=10)
  df2$clustertagkmeans <-  km.out$cluster
   km.out$size
 #  plot(df2, col=(km.out$cluster), main="K-mean result with k=3")

   wss <- (nrow(df2)-1)*sum(apply(df2,2,var))
   for (i in 2:15) wss[i] <- sum(kmeans(df2,centers=i)$withinss)
   plot(1:15, wss, type="b", xlab="Number of Clusters",
        ylab="Within groups sum of squares")#Scatterplot
```

6. Hierarchical clustering

Clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. The idea is to build a binary tree of the data that successively merges similar groups of points.

Code:

```
datamatrix.for.hcluster <- model.matrix(~.+0, data=df2)

transpose.datamatrix.for.hcluster <- t(datamatrix.for.hcluster)

hc.complete=hclust(dist(transpose.datamatrix.for.hcluster ),method="complete") #
Complete linkage type

hc.average=hclust(dist(transpose.datamatrix.for.hcluster ),method="average")  #
Average linkage type

df2$clustertaghierarchical <-   hc.average$cluster

par(mfrow=c(1,2)) #Plotting in a matrix form

plot(hc.complete,main='Complete')

plot(hc.average,main='Average')


write.csv(df2,file = paste0("Clustering/Hierarchical/outputcsv/model",i,".csv"))


cutree(hc.complete,3)

cutree(hc.average,3)
```

Which algorithm to choose

Here are the algorithms:

- K-nearest neighbors (KNN)
- Linear regression
- Logistic regression
- Random Forests
- Neural networks

Here are the dimensions for comparison:

- Problem type (classification/regression)
- Results interpretable by you?
- Easy to explain algorithm to others?
- Average predictive accuracy
- Training speed
- Prediction speed
- Amount of parameter tuning needed (excluding feature selection)
- Performs well with small number of observations?
- Handles lots of irrelevant features well (separates signal from noise)?
- Automatically learns feature interactions?
- Gives calibrated probabilities of class membership?
- Parametric?
- Features might need scaling?

Based on above parameters we can say that every technique has its own advantages and disadvantages, it depends on the situation.

If you need more accuracy go for Random forest and Neural Networks

If you need it to fast, use linear for prediction and logistic for classification.

Conclusion:

Based on the observations and performance metrics of all modelling techniques. We found that Random Forest provides near accurate predictions greater than 90%

Dipti Pamnani Jayesh Samyani Suraj Sharma