Verdigris Project

Yufei Gui

Data Processing

Step 1: load data and parse the data variable.

In this part, we are working on parsing the date variable into calendar date and minute. The date and minute are in numeric form. For example, the date is encoded as 20160727 and minute is encoded as 301. 301 represents 3:01am.

Step 2: Take absolute value of all the data points.

In this project, since we are looking at energy consumption, so we will take the absolute values of all the data points.

Step 3: Check whether there is any missing values.

In this part, we found that there is no missing value in the dataset.

```
## Work Directory
setwd("/Users/Victoria_G/Desktop/verdigris-ml-takehome")

## Read data from local csv files
data.raw <- read.csv("minutely_power.csv")
data.raw <- data.frame(data.raw)  ## List to data frame

## data preprocessing

## For dates
dates <- format(as.POSIXct(strptime(data.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%Y-%m-%d")
dates <- as.character(dates)
dates <- gsub("-","",dates)
dates <- as.numeric(dates)

## For minutes and hours
hour_min <- format(as.POSIXct(strptime(data.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%H:%M")
hour_min <- as.character(hour_min)
hour_min <- gsub(":","",hour_min)
hour_min <- as.numeric(hour_min)

## Hours
#hours <- format(as.POSIXct(strptime(data.raw$X,"%Y-%m-%d
```

```
%H:%M:%S+%S:%S",tz="")) ,format = "%H")
#hours<-as.numeric(hours)

## Minutes
#mins <- format(as.POSIXct(strptime(data.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%M")
#mins<-as.numeric(mins)

## Add back to the original dataset
data <- data.frame("date" = dates, "hour_min" = hour_min,
data.raw[,2:ncol(data.raw)])

## Problem with negative values - take absolute values - need further
validations
data <- abs(data)

## Check if there is any NA values in the dataset
index <- which(is.na(data))
print(length(index))   ## Good - no NA

## [1] 0
```

Step 4 Binning

In this part, we defined a function to try to organize the data and find the driving force from the data. The basic idea of this function is that we want to calculate the 15-min average energy use for 96 time points a day during each month. There are several steps.

Firstly, we sort the data by minute and divide each day's data points into 96 segments(15 min intervals). Among each segments, calculate the average energy use for each circuit.

For each time point(15 min interval point), we calculate the monthwide average energy use for each circuit. Since there are 96 time intervals and 71 circuit, one time column, we will finally generate the 96X72 dataframe. The row is the 15-min time interval point. For example, 0 represents 0:00am and 1345 represents 13:45pm. The columns are the 71 circuits.

This function will also return a vector having length 96. This vector is the monthwide average energy use for each time interval point adding the energy consumption of all 71 circuits.

```
## Check the number of columns in the dataset
n <- ncol(data)
print(n-2)     ## There are 71 circuits recorded

## [1] 71

## Check the earliest and the latest hours recorded
print(max(hour_min))

## [1] 2359

print(min(hour_min))  ## Everyday starts from 00:00 to 23:59
```

```
## [1] 0

## For each month: 4 time intervals in an hour, 96 time intervals in a day;
71 circuits
minutes.unique <- unique(hour_min)
minutes.unique <- sort(minutes.unique)


#define a function that could do the binning each month data
bin <- function(data, month){
  data.average_month <- matrix(0, ncol = 72, nrow = 96)  ## Initialization:
72 columns because of the addition of one column recording time
  if (month == 20161200){
    month2=20170100
  }else{
    month2=month+100
  }
  index <- which(data$date > month & data$date < month2)
  data.month <- data[index,]

  ## sort rows by minutes
  data.month <- data.month[ order(data.month[,2], data.month[,1]), ]
  i = 1
  j = 15
  k = 1
  while (k < 97){
    time <- minutes.unique[i:j]
    index <- which(data.month$hour_min %in% time)
    data.useful <- data.month[index,]
    data.average_month[k,1] <- min(time)
    for (l in 2:72){
      data.average_month[k,l] <- mean(data.useful[,l+1])
    }
    k = k + 1
    i = i + 15
    j = j + 15
  }

  ## End of Binning
  value.month <- rep(0,nrow(data.average_month))

  for (i in 1:length(value.month)){
    value.month[i] <- sum(data.average_month[i,2:ncol(data.average_month)])
  }

  # Find the maximum energy use time point
  peak_index <- which.max(value.month)
  #peak_time<-data.average_month[index,1]
```

```
  return(list(v1=value.month,v2=data.average_month, v3=peak_index))
}
```

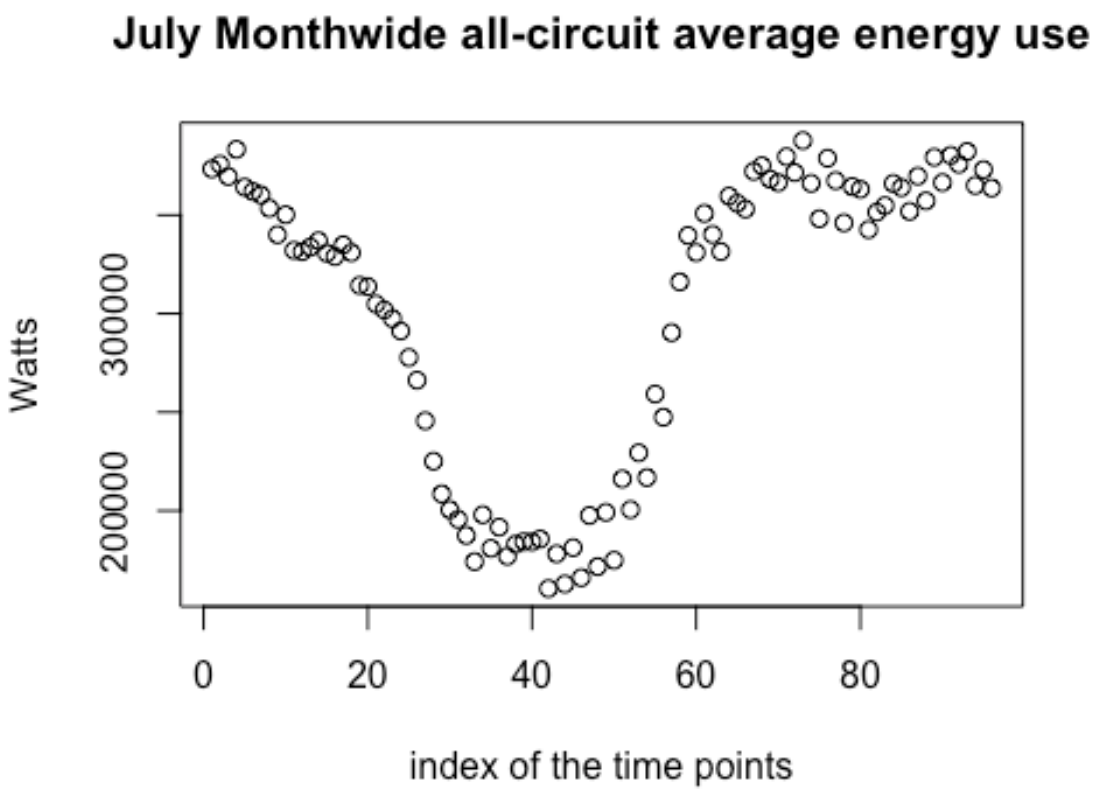
Peak Analysis

In this part, I will take July, 2016 as an example to illustrate how I approach the peak detection problem and how I find the driving force.

The basic idea for the peak detection in this part is that we can get the monthwide all-circuit average energy use for each month.

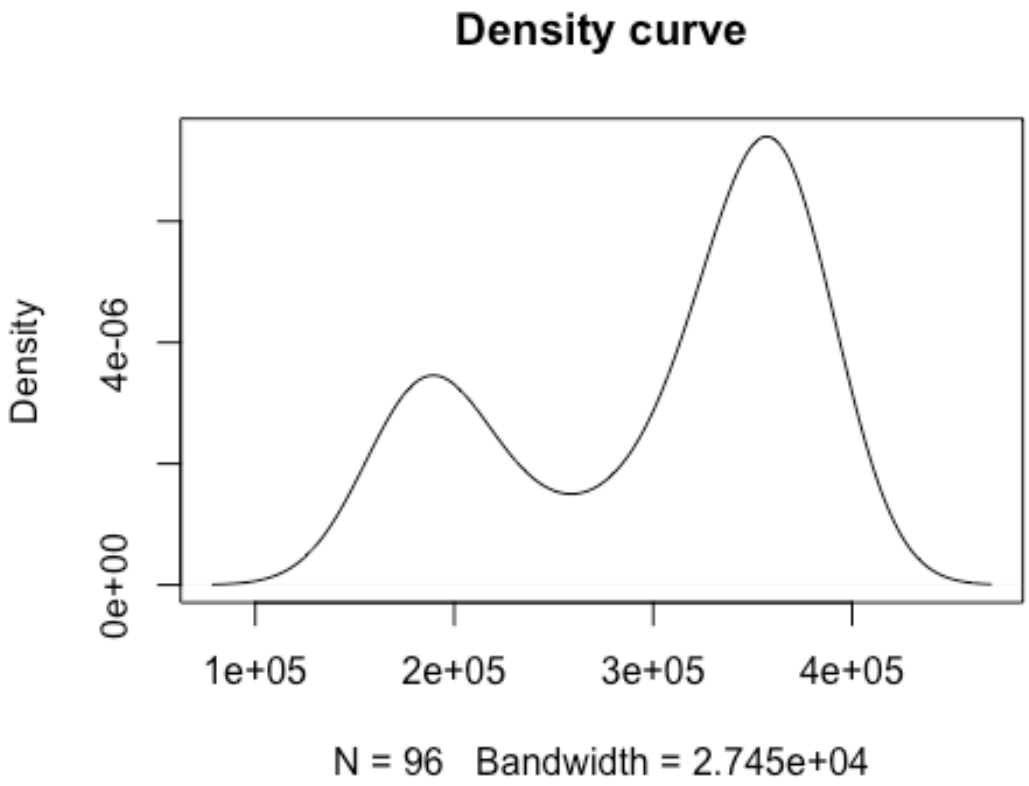
We first take the July,2016 monthwide all-circuit average energy use data and find maximum energy consumption time point. According to the definition provided in the instruction, "**A peak to this customer is the largest average power consumed over a 15 minute interval by month**", this point could be regarded as the peak point in July.

```
result_july<-bin(data, 20160700)
# The monthwide all-circuit average energy use data
value.201607<-result_july$v1
# The 96X72 dataframe
data.average07 <- result_july$v2

index <- which.max(value.201607)

cat("The peak time point in July 2016 is",data.average07[index,1])

## The peak time point in July 2016 is 1800

cat("The maximum monthwide all-circuit average energy use
is",value.201607[index],"watts")

## The maximum monthwide all-circuit average energy use is 387787.4 watts

# Draw the plots for monthwide all-circuit average energy use for each time
interval points of the day
plot(value.201607 , xlab="index of the time points", ylab="Watts", main="July
Monthwide all-circuit average energy use")
```



Summary:
As we can see from the plot, the energy use of decreased from 0am to 10am, and it reached its minimum at 10:15am. However, it began to increase afterwards. According to the graph, it reached its maximum around 6pm.

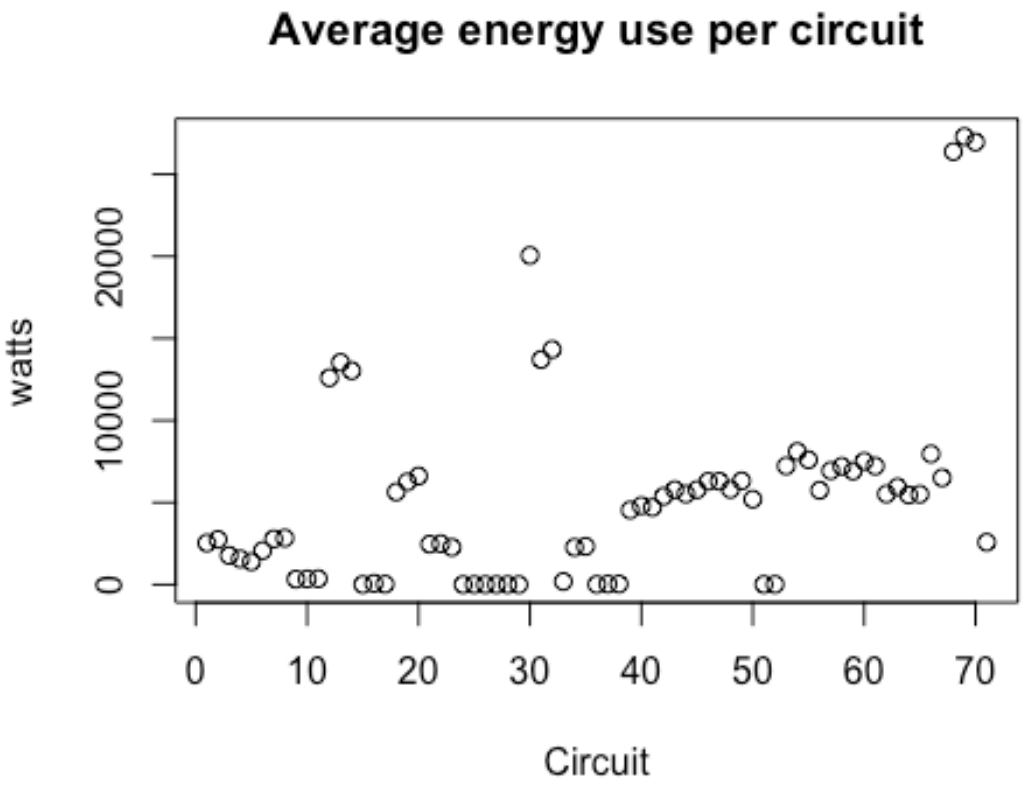
```
plot(main="Density curve",density(value.201607))
```



Summary:
From the density curve, we can find that the distribution of the monthwide all-circuit average energy use is a mixture of two normal distributions.

After finding the peak time point, we can dig into it and try to get the insight of the driving force behind. So we plot the Average energy use per circuit. This is the plot about each circuit's energy use at the peak time period.

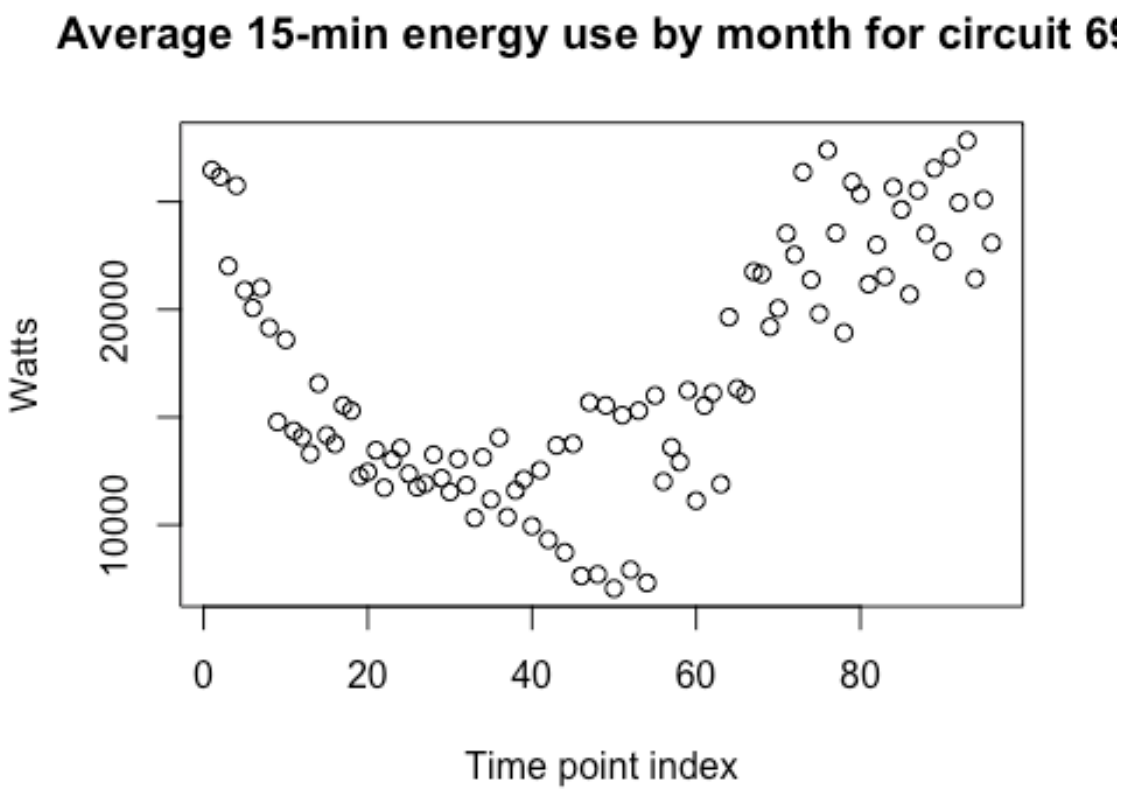
```
plot(data.average07[index,2:ncol(data.average07)], main="Average energy use
per circuit",xlab="Circuit", ylab="watts")
```



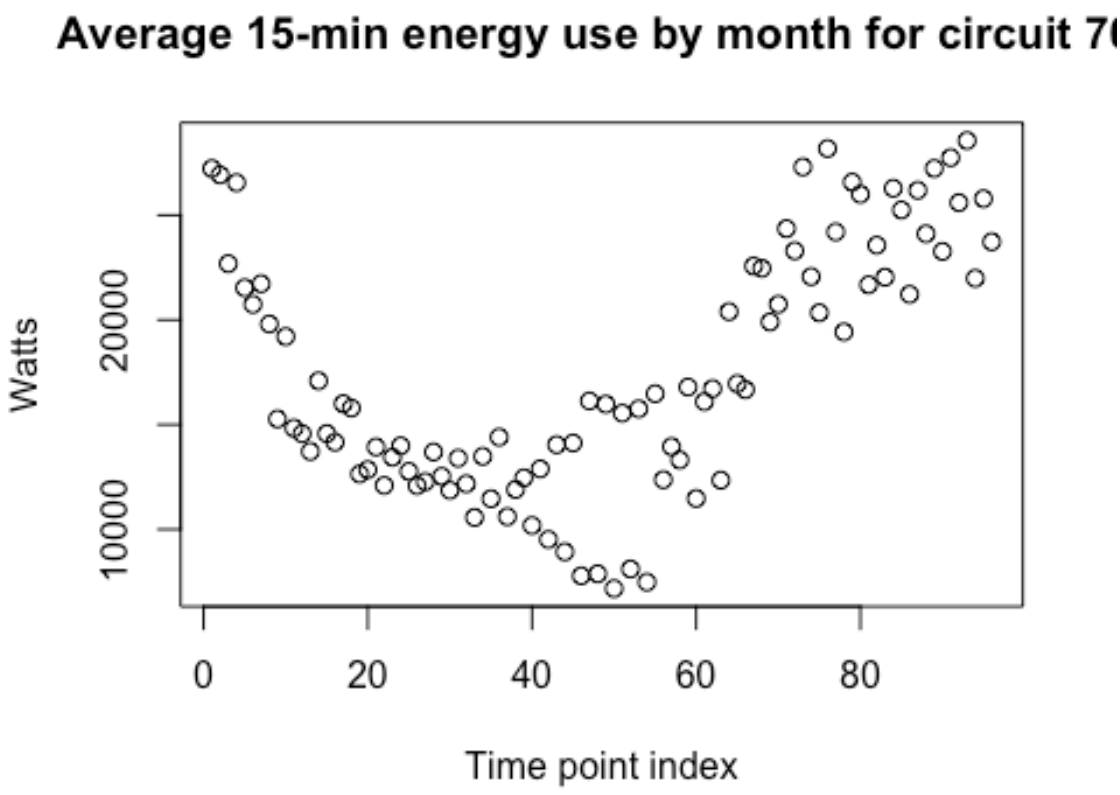
Data insights for the driving force of the Peaks

As we can see from the plot, the circuit 69,70, and 71 have extremely high values at 6:00pm during July, 2016. It may be the potential driving forces for the peak energy use. However, we cannot make conclusions right now, we should also compare the distribution of this circuit vertically during all the time points in the day. If there is no significant change among each time point, then we would say that this circuit may not be the dirving force. However, if there are significant change among each time point during a day, then we may consider this circuit could be the driving force for the inefficient energy use.

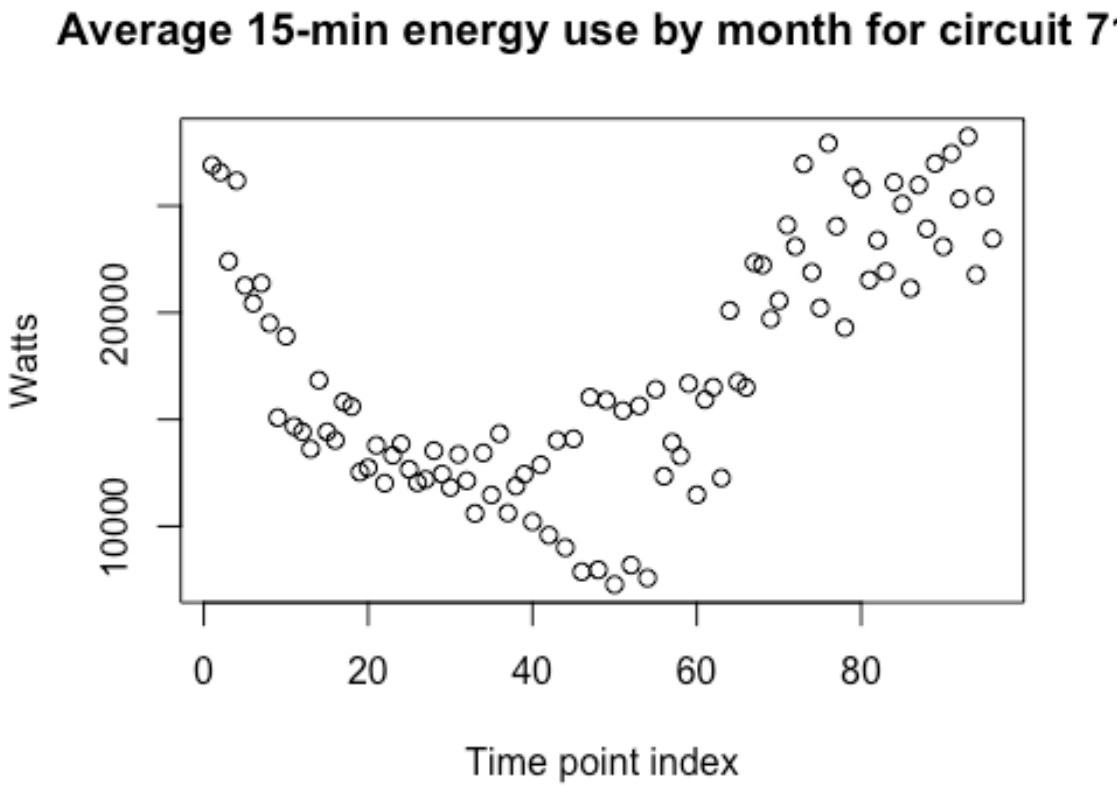
```
plot(data.average07[,69],main="Average 15-min energy use by month for circuit
69 ", xlab="Time point index", ylab="Watts")
```




```r
plot(data.average07[,70],main="Average 15-min energy use by month for circuit
70 ", xlab="Time point index", ylab="Watts")
```




```r
plot(data.average07[,71],main="Average 15-min energy use by month for circuit
71 ", xlab="Time point index", ylab="Watts")
```



It is not hard to find that the energy consumption does change among the different time periods of a day. So we can conclude that the energy use on circuit 69, 70 and 71 are the main driving force. Circuit 69, 70 and 71 corresponds to the circuit X8568, X8570,and X8572. In addition, there are many ways that we can define what the driving force is. It is reasonable that only the energy consumptions may not be the only driving force of the inefficient use. There might be some other factors that will also have the effect on the peak points. We may also consider the weather conditions. For example, the tmeperature and the precipitation during that month.

Data Preparation for Energy forecast

Part I. Generate the monthwide average energy use during a day

Goal: Generate the power consumed over a 15 minute interval for each circuit for each month dataset. Next, combine all 7 month datasets. Finally, by using the results of the peak analysis, we generate a response variable. We find peak time points and label the peak time point as 1, which represents it is a peak point and 0 otherwise.

```
#The data is from July,2016 to Jan, 2017
max(data$date)
```

```
## [1] 20170114

min(data$date)

## [1] 20160727

# generate by month dataset --power consumed over a 15 minute interval by
month for each circuit during a day

month <- c(20160700,20160800,20160900,20161000,20161100,20161200,20170100)

data.201607<-bin(data,month[1])$v2
data.201608<-bin(data,month[2])$v2
data.201609<-bin(data,month[3])$v2
data.201610<-bin(data,month[4])$v2
data.201611<-bin(data,month[5])$v2
data.201612<-bin(data,month[6])$v2
data.201701<-bin(data,month[7])$v2

data.list<-
c(data.201607,data.201608,data.201609,data.201610,data.201611,data.201612,dat
a.201701)

label=rep(0,nrow(data.201607))

#Generate the response variable----the label
peak_index<-bin(data,month[1])$v3
label.full<-label
label.full[peak_index]<-1

for (i in 2:7){
  peak_index<-bin(data,month[i])$v3
  label.month<-label
  label.month[peak_index]<-1
  label.full<-c(label.full,label.month)
}


#Merge the by month dataset
data.full<-
rbind(data.201607,data.201608,data.201609,data.201610,data.201611,data.201612
,data.201701)
data.full<-data.frame(data.full)


# Add dates and label columns to the full data
data.full["Label"]<-label.full
data.full["hour"]<-rep(seq(0,23),each=4)
data.full["min"]<- rep(c(0,15,30,45),168)
data.full["year_mon"]<- rep(month, each=96)
```

```
data.full["month"]<- rep(c(7,8,9,10,11,12,1), each=96)
data.full["year"]<-c(rep(2016,576),rep(2017, 96))
colnames(data.full)[1]<-"hour_min"

#Write the data into csv file
write.csv(data.full,file="monthwide_average.csv",row.names = FALSE)
```

Part II.Data Processing and Impuation for Weather Data

Also we want to combine the weather information into our feature space. For the weather data, we want to parse the date variable and merge it with the previous average energy data set.

Step 1. Data Processing for Weather Data

```
## Read data from local csv files
data.weather.raw <- read.csv("weather.csv")
data.weather.raw  <- data.frame(data.weather.raw )  ## List to data frame

## data preprocessing

## For dates
dates <- format(as.POSIXct(strptime(data.weather.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%Y-%m-%d")
dates <- as.character(dates)
dates <- gsub("-","",dates)
dates <- as.numeric(dates)

## For minutes and hours
minutes <- format(as.POSIXct(strptime(data.weather.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%M")
minutes <- as.character(minutes)
minutes <- gsub(":","",minutes)
minutes <- as.numeric(minutes)

## Hours
hour <- format(as.POSIXct(strptime(data.weather.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%H")
hour<-as.numeric(hour)

## year
year<-format(as.POSIXct(strptime(data.weather.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%Y")
year<-as.numeric(year)

#month
month<-format(as.POSIXct(strptime(data.weather.raw$X,"%Y-%m-%d
%H:%M:%S+%S:%S",tz="")) ,format = "%m")
month<-as.numeric(month)
```

```
## Add back to the original dataset
data.weather <- data.frame("date" = dates,
"year"=year,"month"=month,"hour"=hour,
data.weather.raw[,2:ncol(data.weather.raw)])
```

Step2: Data Imputation

There are about 5448 missing values in the weather datasets. And also we list the missing value rate for each column. We found that variable "ozone" and "precipType" have high missing rate. So we decide to delete this variable. And then implement random forest algorithm to impute the rest of the data.

```
# check the total amount of missing rate
index <- which(is.na(data.weather))
print(length(index))

## [1] 5448

# calculate the missing rate for each column
colMeans(is.na(data.weather))

##                 date                 year                month
##        0.0000000000        0.0000000000        0.0000000000
##                 hour apparentTemperature          cloudCover
##        0.0000000000        0.0000000000        0.2063959604
##            dewPoint             humidity                 icon
##        0.0000000000        0.0000000000        0.0000000000
##               ozone     precipIntensity   precipProbability
##        0.9377235430        0.0000000000        0.0000000000
##          precipType             pressure             summary
##        0.0000000000        0.0000000000        0.0000000000
##         temperature                 time          visibility
##        0.0000000000        0.0000000000        0.0008415737
##         windBearing           windSpeed
##        0.0006311803        0.0006311803

# delete the "ozone" variable
data.weather["ozone"]<-NULL
data.weather["precipType"]<-NULL

# Data impuation with random forest
library(missForest)

## Loading required package: randomForest

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.
```

```
## Loading required package: foreach

## Loading required package: itertools

## Loading required package: iterators

result<-missForest(data.weather)

##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
##   missForest iteration 3 in progress...done!

data.weather.imp<-data.frame(result$ximp)

# write the imputed data to csv file
write.csv(data.weather.imp,file="weather_imputed.csv",row.names = FALSE)
```