Boston Electricity

Sameer, Lalit, Lipsa

October 25, 2016

Table of Contents

## 1. Introduction

This report demonstrates an algorithm to predict energy usage of Boston city. We will try to infer relationship from various factors that potentially impacts the enery consumption in the city.

### 1.1 About the data

We are provided with two data sets, one is from Mildred School colleted in 2014 and other is weather data pulled from <https://www.wunderground.com> using API.

### 1.1 Mildred school energy usage dataset

This data contains Boston energy usage captured at every 5 mins interval.

#### 1.1.1 Data description

We are given with two flat files of 4MB each. We import the datasets from our working directory as:

#### 1.1.2 Description of rows and columns

To understand about Mildred School data details, we took a closer look at the 4 variables Account, Date, Channel and Units recorded at every 5 min time interval.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Type | Range | Description |
| Account | Character | 26908650026 | Account number belonging to Mildred School |
| Date | Date | Jan-Dec'14 | Complete 2014 year data |
| Channel | Factor | 7 Channels | All Channels linked to the account, including Mildred School |
| Units | Factor | 2 Units | Units in which usage is recorded. kWh and kVARh |

### 1.2 Dataset from wunderground.com

This website contains weather data and we can pull this data using API

#### 1.2.1 Data description

We are required to pull Temperature, Dew\_PointF, Humidity, Sea\_Level\_PressureIn, VisibilityMPH, Wind\_Direction, Wind\_SpeedMPH, Conditions and WindDirDegrees data for Boston (KBOS) for 2014 Year.

#### 1.2.2 Description of rows and columns

The weather data is closely understood by looking a columns and their charateristics.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable | Type | Unit | Range | Description |
| TemperatureF | Numeric | Fahrenheit | 0 - 100 | Temprature in atmosphere |
| Dew\_PointF | Numeric | Fahrenheit | -20 - 80 | Temperature to which the air needs to be cooled to make the air saturated |
| Humidity | Numeric | Percent(%) | 10 - 100 | The water content of the air |
| Sea\_Level\_PressureIn | Numeric | Pascal | 28 - 32 | Atmosphere exerts a pressure on the surface of the earth at sea level |
| VisibilityMPH | Numeric | Miles/Hour | 0 - 10 |
| Wind\_Direction | Numeric | Direction | N E W S |
| Wind\_SpeedMPH | Numeric | Miles/Hour | 0 - 50 |
| Conditions | Numeric | Condition | -- |
| WindDirDegrees | Numeric | Degrees | 0 - 360 |

## 2. Goal

We will build a forcasting model that will predict the energy usage in future years by computing two datasets. We will apply multi-linear regression to model Power usage as a function of multiple variables. To supply this model we need to first do data wrangling, then Algorithm implementation and finally forcast.

## 3. Process Flow

Data wrangling -> Feature selection -> Algorithm Implemetation -> Forcasting Model -> Prediction

### 3.1 Installing Packages and libraries

To process the data we will import required packages and libraries

#### 3.1.1 dplyr package

We will install and load the dplyr package that contains additional functions for data manipulation using data frames. It allows us to order rows, select rows, select variables, modify variables and summarize variables. We will be using functions like distinct, filter, group by, n\_distinct from this package.

library(dplyr)

#### 3.1.2 tidyr package

It's designed specifically for data tidying (not general reshaping or aggregating) and works well with 'dplyr' data pipelines.

library(tidyr)

#### 3.1.3 lubridate package

Functions to work with date-times and time-spans: fast and user friendly parsing of date-time data, extraction and updating of components of a date-time (years, months, days, hours, minutes, and seconds), algebraic manipulation on date-time and time-span objects. The 'lubridate' package has a consistent and memorable syntax that makes working with dates easy and fun.

library(lubridate)

#### 3.1.4 weatherData package

Functions that help in fetching weather data from websites. Given a location and a date range, these functions help fetch weather data (temperature, pressure etc.) for any weather related analysis.

library(weatherData)

### 3.2 Data wrangling and prepration

Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw" form into another format that allows for more convenient consumption of the data with the help of semi-automated tools.

#### 3.2.1 Primary dataset filteration

As the dataset contains channels 7 different channels and we are interested in only kwh units for Mildred School we will do intial dataset filtering and merging.

* filter(): Applies linear filtering to a univariate time series or to each series separately of a multivariate time series.
* rbind(): Take a sequence of vector, matrix or data-frame arguments and combine by columns or rows, respectively.

# Filtering  
rd1 <- filter(rd1, Units == "kWh",Channel == "MILDRED SCHOOL 1")  
rd2 <- filter(rd2, Units == "kWh",Channel == "MILDRED SCHOOL 1")  
  
# Merging  
data <- rbind(rd1,rd2)

#### 3.2.2 Datatype conformity

As we read the data from dataset, r intutively understands datatypes. We need to check structure of our data and then change the columns structure according to our requirements.

str() function gives us the snapshot of datatypes of all the columns

str(data[1:4])

## 'data.frame': 365 obs. of 4 variables:  
## $ Account: num 2.69e+10 2.69e+10 2.69e+10 2.69e+10 2.69e+10 ...  
## $ Date : Factor w/ 365 levels "1/1/2014","1/10/2014",..: 1 12 23 26 27 28 29 30 31 2 ...  
## $ Channel: Factor w/ 7 levels "507115423 1 kWh",..: 6 6 6 6 6 6 6 6 6 6 ...  
## $ Units : Factor w/ 3 levels "kVARh","kWh",..: 2 2 2 2 2 2 2 2 2 2 ...

We will now convert the datatype of these columns. We are using as.Date() and as.charcter() function which are available in base r library.

data$Date <- as.Date(data$Date,"%m/%d/%Y")  
data$Channel <- as.character(data$Channel)  
data$Units <- as.character(data$Units)  
  
# Re-check structure of data  
str(data[1:4])

## 'data.frame': 365 obs. of 4 variables:  
## $ Account: num 2.69e+10 2.69e+10 2.69e+10 2.69e+10 2.69e+10 ...  
## $ Date : Date, format: "2014-01-01" "2014-01-02" ...  
## $ Channel: chr "MILDRED SCHOOL 1" "MILDRED SCHOOL 1" "MILDRED SCHOOL 1" "MILDRED SCHOOL 1" ...  
## $ Units : chr "kWh" "kWh" "kWh" "kWh" ...

As we can see that the data is widely spread and for any analysis to be done we need to convert the data into long format. We will first find the poisitions of the columns and also save the columnnames into array.

* grep(): Search for matches to argument pattern within each element of a character vector

column\_Pos <- grep("X",colnames(data))

This wills store the names of the columns for the observations

columnnamesval <- grep("X",colnames(data),value = TRUE)

### 3.3 Data Gathering

We need to gather right data

#### 3.3.1 Gathering the data in long format

We need to gather data in long format. \* gather(): \* head(): Returns the first parts of a vector, matrix, table, data frame or function.

Data.long <- gather(data,key,values,column\_Pos)  
head(Data.long)

## Account Date Channel Units key values  
## 1 26908650026 2014-01-01 MILDRED SCHOOL 1 kWh X0.05 11.13  
## 2 26908650026 2014-01-02 MILDRED SCHOOL 1 kWh X0.05 10.17  
## 3 26908650026 2014-01-03 MILDRED SCHOOL 1 kWh X0.05 10.47  
## 4 26908650026 2014-01-04 MILDRED SCHOOL 1 kWh X0.05 9.99  
## 5 26908650026 2014-01-05 MILDRED SCHOOL 1 kWh X0.05 18.10  
## 6 26908650026 2014-01-06 MILDRED SCHOOL 1 kWh X0.05 9.96

#### 3.3.2 Derive data

Now we have the data in the long format we can apply some aggregations on the data for each hour which will comprise of 12 observations for each hour. We will create 24 new columns which will be sum of 12 observations of each hour using dplyr summarize and group by function.

Columns to be derived:

|  |  |
| --- | --- |
| Column Name | Description |
| kWh | Sum of 12 observations (5 min intervals rolled up to hourly) |
| month | 1-12 => Jan-Dec - Derived from dates |
| day | 1-31 - Derived from dates |
| year | Derived from dates |
| hour | 0-23 - Derived for each record corresponding to the hour of observation |
| Day of Week | 0-6 -Sun-Sat - Derived from dates |
| Weekday | 1- Yes 0- No - Derived from dates |
| Peakhour | 7AM-7PM - 1 ; 7PM-7AM - 0 |

Below is a user defined function to find the Weekday

checkWeekday <- function(date){  
 a = if((wday(date) == 6 || wday(date) == 7)) 0 else 1  
 return(a)  
}

We Will use dplyr chaining function to group the data first and summarize the columns for each hour using the 12 observations/each hour \* group\_by(): It breaks down a dataset into specified groups of rows. \* summarise(): Summarise multiple values to a single value. \* mutate(): Mutate adds new variables and preserves existing; transmute drops existing variables.

aggData <- Data.long %>% group\_by(Account,Date,Channel,Units) %>%   
 summarise('0' = sum(values[key %in% columnnamesval[1:12]]),  
 '1' = sum(values[key %in% columnnamesval[13:24]]),  
 '2' = sum(values[key %in% columnnamesval[25:36]]),  
 '3' = sum(values[key %in% columnnamesval[37:48]]),  
 '4' = sum(values[key %in% columnnamesval[49:60]]),  
 '5' = sum(values[key %in% columnnamesval[61:72]]),  
 '6' = sum(values[key %in% columnnamesval[73:84]]),  
 '7' = sum(values[key %in% columnnamesval[85:96]]),  
 '8' = sum(values[key %in% columnnamesval[97:108]]),  
 '9' = sum(values[key %in% columnnamesval[109:120]]),  
 '10' = sum(values[key %in% columnnamesval[121:132]]),  
 '11' = sum(values[key %in% columnnamesval[133:144]]),  
 '12' = sum(values[key %in% columnnamesval[145:156]]),  
 '13' = sum(values[key %in% columnnamesval[157:168]]),  
 '14' = sum(values[key %in% columnnamesval[169:180]]),  
 '15' = sum(values[key %in% columnnamesval[181:192]]),  
 '16' = sum(values[key %in% columnnamesval[193:204]]),  
 '17' = sum(values[key %in% columnnamesval[205:216]]),  
 '18' = sum(values[key %in% columnnamesval[217:228]]),  
 '19' = sum(values[key %in% columnnamesval[229:240]]),  
 '20' = sum(values[key %in% columnnamesval[241:252]]),  
 '21' = sum(values[key %in% columnnamesval[253:264]]),  
 '22' = sum(values[key %in% columnnamesval[265:276]]),  
 '23' = sum(values[key %in% columnnamesval[277:288]])) %>%  
 mutate(month = lubridate::month(Date),day = day(Date),year = year(Date),'Day of Week' = wday(Date)) %>%  
 mutate(weekday = sapply(Date, function(x) checkWeekday(x)))

Gathering the column again in long format so that data is in clean consistent format. This way we have the kWH value for every hour and for every day of the year 2014

aggData.long <- gather(aggData,hour,Kwh,5:28)

We will change the hour column into numeric

aggData.long$hour <- as.numeric(aggData.long$hour)

Now lets derive peak hour. Using the table definition we can define a function based on hours like below

cal\_PeakHour <- function(hour){  
 p = if(hour > 6 & hour < 20) 1 else 0  
 return(p)  
}

* sapply(): Each element of which is the result of applying FUN to the corresponding element of X.

aggData.long$PeakHour <- sapply(aggData.long$hour,function(x) cal\_PeakHour(x))

The structure is now aligned with our requirement now lets pull up the other dataset

#### 3.3.3 Data from weather API

We will use the WeatherData Package which use wunderground.com data.

Finding minium and max date for our observed dataset

mindate <- min(aggData.long$Date)  
maxdate <- max(aggData.long$Date)

Converting into date

mindate <- as.Date("01/01/2014", "%m/%d/%Y")  
maxdate <- as.Date("12/31/2014", "%m/%d/%Y")

Lets get the station code for Boston

getStationCode("Boston")

## [[1]]  
## Station State airportCode  
## 656 Boston MA KBOS  
##   
## [[2]]  
## [1] "USA MA BOSTON KBOS BOS 72509 42 22N 071 01W 6 X U A 0 US"  
## [2] "USA MA BOSTON/TAUNTON KBOX BOX 41 57N 071 08W 36 X F 8 US"  
## [3] "USA MA BOSTON/RFC KTAR TAR 41 57N 071 08W 36 R 8 US"

station\_id is "KBOS"

* getWeatherForDate(): Getting data for a range of dates, it has certain parameters
* station\_id: is a valid 3- or 4-letter Airport code or a valid Weather Station ID (example: "KBOS" for Boston).
* start\_date: string representing a date in the past ("YYYY-MM-DD", all numeric)
* end\_date : Ifanintervalistobespeci???ed,end\_dateisastringrepresentingadateinthepast ("YYYY-MM-DD", all numeric) and greater than the start date
* opt\_detailed: Boolen ???ag to indicate if detailed records for the station are desired. (default FALSE). By default only one records per date is returned.
* opt\_custom\_columns: Boolen ???ag to indicate if only a user-speci???ed set of columns are to be returned. (default FALSE) If TRUE, then the desired columns must be speci???ed via custom\_columns
* custom\_columns: Vector of integers speci???ed by the user to indicate which columns to fetch. The Date column is always returned as the ???rst column.

#WeatherData <- getWeatherForDate("KBOS", start\_date=mindate,  
# end\_date = maxdate,  
# opt\_detailed=T,opt\_custom\_columns=T,  
# custom\_columns=c(2:13))  
WeatherData <- read.csv("WeatherData\_new.csv", header = TRUE)

Data looks like this

head(WeatherData)

## X Time TemperatureF Dew\_PointF Humidity  
## 1 1 2014-01-01 00:54:00 23.0 5.0 46  
## 2 2 2014-01-01 01:54:00 21.9 3.9 46  
## 3 3 2014-01-01 02:54:00 21.9 3.9 46  
## 4 4 2014-01-01 03:54:00 21.9 3.0 44  
## 5 5 2014-01-01 04:54:00 21.0 3.0 46  
## 6 6 2014-01-01 05:54:00 21.0 3.0 46  
## Sea\_Level\_PressureIn VisibilityMPH Wind\_Direction Wind\_SpeedMPH  
## 1 30.20 10 WNW 8.1  
## 2 30.23 10 WNW 11.5  
## 3 30.25 10 WSW 12.7  
## 4 30.27 10 WSW 11.5  
## 5 30.29 10 West 9.2  
## 6 30.30 10 West 11.5  
## Gust\_SpeedMPH PrecipitationIn Events Conditions WindDirDegrees  
## 1 - N/A <NA> Clear 290  
## 2 - N/A <NA> Partly Cloudy 290  
## 3 - N/A <NA> Clear 240  
## 4 19.6 N/A <NA> Clear 250  
## 5 - N/A <NA> Clear 260  
## 6 20.7 N/A <NA> Clear 270  
## date hour  
## 1 2014-01-01 0  
## 2 2014-01-01 1  
## 3 2014-01-01 2  
## 4 2014-01-01 3  
## 5 2014-01-01 4  
## 6 2014-01-01 5

Lets use Lubridate package to find the date and hour

WeatherData$date = date(WeatherData$Time)  
WeatherData$hour = hour(WeatherData$Time)