Boston Electricity

Sameer, Lalit, Lipsa

October 25, 2016

Table of Contents

[1. Introduction 1](#_Toc465152832)

[1.1 About the data 1](#_Toc465152833)

[1.1 Mildred school energy usage dataset 1](#_Toc465152834)

[1.2 Dataset from wunderground.com 2](#_Toc465152835)

[2. Goal 2](#_Toc465152836)

[3. Process Flow 3](#_Toc465152837)

[3.1 Installing Packages and libraries 3](#_Toc465152838)

[3.2 Data wrangling and prepration 3](#_Toc465152839)

[3.3 Data Gathering 5](#_Toc465152840)

[Data looks like this 11](#_Toc465152841)

## 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))

Lets have a look at the structure of our dataset now

str(aggData.long)

## Classes 'grouped\_df', 'tbl\_df', 'tbl' and 'data.frame': 8760 obs. of 12 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" ...  
## $ month : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ day : int 1 2 3 4 5 6 7 8 9 10 ...  
## $ year : num 2014 2014 2014 2014 2014 ...  
## $ Day of Week: num 4 5 6 7 1 2 3 4 5 6 ...  
## $ weekday : num 1 1 0 0 1 1 1 1 1 0 ...  
## $ hour : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Kwh : num 132 121 125 121 216 ...  
## $ PeakHour : num 0 0 0 0 0 0 0 0 0 0 ...  
## - attr(\*, "vars")=List of 3  
## ..$ : symbol Account  
## ..$ : symbol Date  
## ..$ : symbol Channel  
## - attr(\*, "drop")= logi TRUE  
## - attr(\*, "indices")=List of 365  
## ..$ : int 0 365 730 1095 1460 1825 2190 2555 2920 3285 ...  
## ..$ : int 1 366 731 1096 1461 1826 2191 2556 2921 3286 ...  
## ..$ : int 2 367 732 1097 1462 1827 2192 2557 2922 3287 ...  
## ..$ : int 3 368 733 1098 1463 1828 2193 2558 2923 3288 ...  
## ..$ : int 4 369 734 1099 1464 1829 2194 2559 2924 3289 ...  
## ..$ : int 5 370 735 1100 1465 1830 2195 2560 2925 3290 ...  
## ..$ : int 6 371 736 1101 1466 1831 2196 2561 2926 3291 ...  
## ..$ : int 7 372 737 1102 1467 1832 2197 2562 2927 3292 ...  
## ..$ : int 8 373 738 1103 1468 1833 2198 2563 2928 3293 ...  
## ..$ : int 9 374 739 1104 1469 1834 2199 2564 2929 3294 ...  
## ..$ : int 10 375 740 1105 1470 1835 2200 2565 2930 3295 ...  
## ..$ : int 11 376 741 1106 1471 1836 2201 2566 2931 3296 ...  
## ..$ : int 12 377 742 1107 1472 1837 2202 2567 2932 3297 ...  
## ..$ : int 13 378 743 1108 1473 1838 2203 2568 2933 3298 ...  
## ..$ : int 14 379 744 1109 1474 1839 2204 2569 2934 3299 ...  
## ..$ : int 15 380 745 1110 1475 1840 2205 2570 2935 3300 ...  
## ..$ : int 16 381 746 1111 1476 1841 2206 2571 2936 3301 ...  
## ..$ : int 17 382 747 1112 1477 1842 2207 2572 2937 3302 ...  
## ..$ : int 18 383 748 1113 1478 1843 2208 2573 2938 3303 ...  
## ..$ : int 19 384 749 1114 1479 1844 2209 2574 2939 3304 ...  
## ..$ : int 20 385 750 1115 1480 1845 2210 2575 2940 3305 ...  
## ..$ : int 21 386 751 1116 1481 1846 2211 2576 2941 3306 ...  
## ..$ : int 22 387 752 1117 1482 1847 2212 2577 2942 3307 ...  
## ..$ : int 23 388 753 1118 1483 1848 2213 2578 2943 3308 ...  
## ..$ : int 24 389 754 1119 1484 1849 2214 2579 2944 3309 ...  
## ..$ : int 25 390 755 1120 1485 1850 2215 2580 2945 3310 ...  
## ..$ : int 26 391 756 1121 1486 1851 2216 2581 2946 3311 ...  
## ..$ : int 27 392 757 1122 1487 1852 2217 2582 2947 3312 ...  
## ..$ : int 28 393 758 1123 1488 1853 2218 2583 2948 3313 ...  
## ..$ : int 29 394 759 1124 1489 1854 2219 2584 2949 3314 ...  
## ..$ : int 30 395 760 1125 1490 1855 2220 2585 2950 3315 ...  
## ..$ : int 31 396 761 1126 1491 1856 2221 2586 2951 3316 ...  
## ..$ : int 32 397 762 1127 1492 1857 2222 2587 2952 3317 ...  
## ..$ : int 33 398 763 1128 1493 1858 2223 2588 2953 3318 ...  
## ..$ : int 34 399 764 1129 1494 1859 2224 2589 2954 3319 ...  
## ..$ : int 35 400 765 1130 1495 1860 2225 2590 2955 3320 ...  
## ..$ : int 36 401 766 1131 1496 1861 2226 2591 2956 3321 ...  
## ..$ : int 37 402 767 1132 1497 1862 2227 2592 2957 3322 ...  
## ..$ : int 38 403 768 1133 1498 1863 2228 2593 2958 3323 ...  
## ..$ : int 39 404 769 1134 1499 1864 2229 2594 2959 3324 ...  
## ..$ : int 40 405 770 1135 1500 1865 2230 2595 2960 3325 ...  
## ..$ : int 41 406 771 1136 1501 1866 2231 2596 2961 3326 ...  
## ..$ : int 42 407 772 1137 1502 1867 2232 2597 2962 3327 ...  
## ..$ : int 43 408 773 1138 1503 1868 2233 2598 2963 3328 ...  
## ..$ : int 44 409 774 1139 1504 1869 2234 2599 2964 3329 ...  
## ..$ : int 45 410 775 1140 1505 1870 2235 2600 2965 3330 ...  
## ..$ : int 46 411 776 1141 1506 1871 2236 2601 2966 3331 ...  
## ..$ : int 47 412 777 1142 1507 1872 2237 2602 2967 3332 ...  
## ..$ : int 48 413 778 1143 1508 1873 2238 2603 2968 3333 ...  
## ..$ : int 49 414 779 1144 1509 1874 2239 2604 2969 3334 ...  
## ..$ : int 50 415 780 1145 1510 1875 2240 2605 2970 3335 ...  
## ..$ : int 51 416 781 1146 1511 1876 2241 2606 2971 3336 ...  
## ..$ : int 52 417 782 1147 1512 1877 2242 2607 2972 3337 ...  
## ..$ : int 53 418 783 1148 1513 1878 2243 2608 2973 3338 ...  
## ..$ : int 54 419 784 1149 1514 1879 2244 2609 2974 3339 ...  
## ..$ : int 55 420 785 1150 1515 1880 2245 2610 2975 3340 ...  
## ..$ : int 56 421 786 1151 1516 1881 2246 2611 2976 3341 ...  
## ..$ : int 57 422 787 1152 1517 1882 2247 2612 2977 3342 ...  
## ..$ : int 58 423 788 1153 1518 1883 2248 2613 2978 3343 ...  
## ..$ : int 59 424 789 1154 1519 1884 2249 2614 2979 3344 ...  
## ..$ : int 60 425 790 1155 1520 1885 2250 2615 2980 3345 ...  
## ..$ : int 61 426 791 1156 1521 1886 2251 2616 2981 3346 ...  
## ..$ : int 62 427 792 1157 1522 1887 2252 2617 2982 3347 ...  
## ..$ : int 63 428 793 1158 1523 1888 2253 2618 2983 3348 ...  
## ..$ : int 64 429 794 1159 1524 1889 2254 2619 2984 3349 ...  
## ..$ : int 65 430 795 1160 1525 1890 2255 2620 2985 3350 ...  
## ..$ : int 66 431 796 1161 1526 1891 2256 2621 2986 3351 ...  
## ..$ : int 67 432 797 1162 1527 1892 2257 2622 2987 3352 ...  
## ..$ : int 68 433 798 1163 1528 1893 2258 2623 2988 3353 ...  
## ..$ : int 69 434 799 1164 1529 1894 2259 2624 2989 3354 ...  
## ..$ : int 70 435 800 1165 1530 1895 2260 2625 2990 3355 ...  
## ..$ : int 71 436 801 1166 1531 1896 2261 2626 2991 3356 ...  
## ..$ : int 72 437 802 1167 1532 1897 2262 2627 2992 3357 ...  
## ..$ : int 73 438 803 1168 1533 1898 2263 2628 2993 3358 ...  
## ..$ : int 74 439 804 1169 1534 1899 2264 2629 2994 3359 ...  
## ..$ : int 75 440 805 1170 1535 1900 2265 2630 2995 3360 ...  
## ..$ : int 76 441 806 1171 1536 1901 2266 2631 2996 3361 ...  
## ..$ : int 77 442 807 1172 1537 1902 2267 2632 2997 3362 ...  
## ..$ : int 78 443 808 1173 1538 1903 2268 2633 2998 3363 ...  
## ..$ : int 79 444 809 1174 1539 1904 2269 2634 2999 3364 ...  
## ..$ : int 80 445 810 1175 1540 1905 2270 2635 3000 3365 ...  
## ..$ : int 81 446 811 1176 1541 1906 2271 2636 3001 3366 ...  
## ..$ : int 82 447 812 1177 1542 1907 2272 2637 3002 3367 ...  
## ..$ : int 83 448 813 1178 1543 1908 2273 2638 3003 3368 ...  
## ..$ : int 84 449 814 1179 1544 1909 2274 2639 3004 3369 ...  
## ..$ : int 85 450 815 1180 1545 1910 2275 2640 3005 3370 ...  
## ..$ : int 86 451 816 1181 1546 1911 2276 2641 3006 3371 ...  
## ..$ : int 87 452 817 1182 1547 1912 2277 2642 3007 3372 ...  
## ..$ : int 88 453 818 1183 1548 1913 2278 2643 3008 3373 ...  
## ..$ : int 89 454 819 1184 1549 1914 2279 2644 3009 3374 ...  
## ..$ : int 90 455 820 1185 1550 1915 2280 2645 3010 3375 ...  
## ..$ : int 91 456 821 1186 1551 1916 2281 2646 3011 3376 ...  
## ..$ : int 92 457 822 1187 1552 1917 2282 2647 3012 3377 ...  
## ..$ : int 93 458 823 1188 1553 1918 2283 2648 3013 3378 ...  
## ..$ : int 94 459 824 1189 1554 1919 2284 2649 3014 3379 ...  
## ..$ : int 95 460 825 1190 1555 1920 2285 2650 3015 3380 ...  
## ..$ : int 96 461 826 1191 1556 1921 2286 2651 3016 3381 ...  
## ..$ : int 97 462 827 1192 1557 1922 2287 2652 3017 3382 ...  
## ..$ : int 98 463 828 1193 1558 1923 2288 2653 3018 3383 ...  
## .. [list output truncated]  
## - attr(\*, "group\_sizes")= int 24 24 24 24 24 24 24 24 24 24 ...  
## - attr(\*, "biggest\_group\_size")= int 24  
## - attr(\*, "labels")='data.frame': 365 obs. of 3 variables:  
## ..$ Account: num 2.69e+10 2.69e+10 2.69e+10 2.69e+10 2.69e+10 ...  
## ..$ Date : Date, format: "2014-01-01" ...  
## ..$ Channel: chr "MILDRED SCHOOL 1" "MILDRED SCHOOL 1" "MILDRED SCHOOL 1" "MILDRED SCHOOL 1" ...  
## ..- attr(\*, "vars")=List of 3  
## .. ..$ : symbol Account  
## .. ..$ : symbol Date  
## .. ..$ : symbol Channel  
## ..- attr(\*, "drop")= logi TRUE

The structure seems to align 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))

## Data looks like this

#head(WeatherData)

Lets use Lubridate package to find the date and hour

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