Exploratory Data Analysis with R

Microsoft Data Science Team

May 27, 2016

# Importing Data

There are many ways of getting your data into R, each with different strengths. When working data saved in text files or csv's, you can use the standard R function (available in the utitls package, loaded with every R installation), read.table and it's cousin read.csv. When working with very large data, see the data.table::fread function, or the readr package.

hvac <- read.csv("HVAC.csv", header = T, stringsAsFactors = F)  
building <- read.csv("building.csv", header = T, stringsAsFactors = F)  
head(hvac)

## Date Time TargetTemp ActualTemp System SystemAge BuildingID  
## 1 6/1/13 0:00:01 66 58 13 20 4  
## 2 6/2/13 1:00:01 69 68 3 20 17  
## 3 6/3/13 2:00:01 70 73 17 20 18  
## 4 6/4/13 3:00:01 67 63 2 23 15  
## 5 6/5/13 4:00:01 68 74 16 9 3  
## 6 6/6/13 5:00:01 67 56 13 28 4

head(building)

## BuildingID BuildingMgr BuildingAge HVACproduct Country  
## 1 1 M1 25 AC1000 USA  
## 2 2 M2 27 FN39TG France  
## 3 3 M3 28 JDNS77 Brazil  
## 4 4 M4 17 GG1919 Finland  
## 5 5 M5 3 ACMAX22 Hong Kong  
## 6 6 M6 9 AC1000 Singapore

dim(hvac)

## [1] 8000 7

dim(building)

## [1] 20 5

We used a couple of additional arguments to make sure our data was imported in the corret format. First, since our csv files have a header row, we set the header argument to the value TRUE. Secondally, and more importantly, we set the argument stringsAsFactors to FALSE. Convertign factors into other data types is a frequent stress-inducing problem in R. By setting stringsAsFactors to FALSE, we can ensure that our character columns (columns containing strings) are not converted to factor columns. If we later want to use factors, we can conver the character columns into factors, which is usually met with less errors than converting factors into characters/numerics.

# Summary Statistics with R

The return value for the read.csv function is a data.frame, which is a special kind of list. The summary function has a specific method for data.frames, which provides summary statistics for each column.

summary(building)

## BuildingID BuildingMgr BuildingAge HVACproduct   
## Min. : 1.00 Length:20 Min. : 3.00 Length:20   
## 1st Qu.: 5.75 Class :character 1st Qu.:13.75 Class :character   
## Median :10.50 Mode :character Median :19.00 Mode :character   
## Mean :10.50 Mean :18.70   
## 3rd Qu.:15.25 3rd Qu.:25.00   
## Max. :20.00 Max. :28.00   
## Country   
## Length:20   
## Class :character   
## Mode :character   
##   
##   
##

summary(hvac)

## Date Time TargetTemp ActualTemp   
## Length:8000 Length:8000 Min. :65.0 Min. :55.00   
## Class :character Class :character 1st Qu.:66.0 1st Qu.:61.00   
## Mode :character Mode :character Median :67.0 Median :68.00   
## Mean :67.5 Mean :67.64   
## 3rd Qu.:69.0 3rd Qu.:74.00   
## Max. :70.0 Max. :80.00   
## System SystemAge BuildingID   
## Min. : 1.00 Min. : 1.00 Min. : 1.00   
## 1st Qu.: 5.00 1st Qu.: 8.00 1st Qu.: 6.00   
## Median :10.00 Median :15.00 Median :10.00   
## Mean :10.38 Mean :15.35 Mean :10.51   
## 3rd Qu.:15.00 3rd Qu.:23.00 3rd Qu.:16.00   
## Max. :20.00 Max. :30.00 Max. :20.00

The output provides five point summary statistics for the numeric columns as well as the average. For character data, there is less information provided. If we convert the character columns into factor variables, we will see truncated tabulations of their values.

All data types in R have a handy as. function to convert from one data type to another. In order to convert multiple columns at once, we'll use the lapply function and select the columns we wan to convert.

build\_fctrs <- c("BuildingID", "Country")  
building[build\_fctrs] <- lapply(building[build\_fctrs], as.factor)  
hvac$BuildingID <- as.factor(hvac$BuildingID)  
summary(hvac)

## Date Time TargetTemp ActualTemp   
## Length:8000 Length:8000 Min. :65.0 Min. :55.00   
## Class :character Class :character 1st Qu.:66.0 1st Qu.:61.00   
## Mode :character Mode :character Median :67.0 Median :68.00   
## Mean :67.5 Mean :67.64   
## 3rd Qu.:69.0 3rd Qu.:74.00   
## Max. :70.0 Max. :80.00   
##   
## System SystemAge BuildingID   
## Min. : 1.00 Min. : 1.00 12 : 426   
## 1st Qu.: 5.00 1st Qu.: 8.00 10 : 425   
## Median :10.00 Median :15.00 18 : 421   
## Mean :10.38 Mean :15.35 6 : 415   
## 3rd Qu.:15.00 3rd Qu.:23.00 7 : 415   
## Max. :20.00 Max. :30.00 16 : 414   
## (Other):5484

summary(building)

## BuildingID BuildingMgr BuildingAge HVACproduct   
## 1 : 1 Length:20 Min. : 3.00 Length:20   
## 2 : 1 Class :character 1st Qu.:13.75 Class :character   
## 3 : 1 Mode :character Median :19.00 Mode :character   
## 4 : 1 Mean :18.70   
## 5 : 1 3rd Qu.:25.00   
## 6 : 1 Max. :28.00   
## (Other):14   
## Country   
## Finland : 2   
## Argentina: 1   
## Australia: 1   
## Belgium : 1   
## Brazil : 1   
## Canada : 1   
## (Other) :13

We should also convert the Date column into a POSIXct date object.

head(hvac)

## Date Time TargetTemp ActualTemp System SystemAge BuildingID  
## 1 6/1/13 0:00:01 66 58 13 20 4  
## 2 6/2/13 1:00:01 69 68 3 20 17  
## 3 6/3/13 2:00:01 70 73 17 20 18  
## 4 6/4/13 3:00:01 67 63 2 23 15  
## 5 6/5/13 4:00:01 68 74 16 9 3  
## 6 6/6/13 5:00:01 67 56 13 28 4

hvac$Date\_Time <- as.POSIXct(paste(hvac$Date,   
 hvac$Time, sep = " "),   
 format = "%m/%d/%y %H:%M:%S")

## Warning in strptime(x, format, tz = tz): unable to identify current timezone 'C':  
## please set environment variable 'TZ'

## Warning in strptime(x, format, tz = tz): unknown timezone 'localtime'

hvac$Date <- as.Date(hvac$Date\_Time)  
hvac$Time <- strftime(hvac$Date\_Time, format = "%H:%M:%S")  
head(hvac)

## Date Time TargetTemp ActualTemp System SystemAge BuildingID  
## 1 2013-06-01 00:00:01 66 58 13 20 4  
## 2 2013-06-02 01:00:01 69 68 3 20 17  
## 3 2013-06-03 02:00:01 70 73 17 20 18  
## 4 2013-06-04 03:00:01 67 63 2 23 15  
## 5 2013-06-05 04:00:01 68 74 16 9 3  
## 6 2013-06-06 05:00:01 67 56 13 28 4  
## Date\_Time  
## 1 2013-06-01 00:00:01  
## 2 2013-06-02 01:00:01  
## 3 2013-06-03 02:00:01  
## 4 2013-06-04 03:00:01  
## 5 2013-06-05 04:00:01  
## 6 2013-06-06 05:00:01

To make your life easier when working with dates, take a look at the [lubridate](http://cran.r-project.org/web/packages/lubridate/index.html) package.

## Merging Data

Let's merge the hvac and buildings data.frames together.

build\_temps <- merge(building, hvac, by = "BuildingID")  
head(build\_temps)

## BuildingID BuildingMgr BuildingAge HVACproduct Country Date  
## 1 1 M1 25 AC1000 USA 2013-06-06  
## 2 1 M1 25 AC1000 USA 2013-06-11  
## 3 1 M1 25 AC1000 USA 2013-06-26  
## 4 1 M1 25 AC1000 USA 2013-06-02  
## 5 1 M1 25 AC1000 USA 2013-06-27  
## 6 1 M1 25 AC1000 USA 2013-06-06  
## Time TargetTemp ActualTemp System SystemAge Date\_Time  
## 1 12:43:51 70 64 19 30 2013-06-06 12:43:51  
## 2 17:45:56 70 77 7 2 2013-06-11 17:45:56  
## 3 01:45:56 69 73 18 5 2013-06-26 01:45:56  
## 4 10:43:51 67 55 6 12 2013-06-02 10:43:51  
## 5 00:00:01 65 57 18 21 2013-06-27 00:00:01  
## 6 07:00:01 68 60 10 24 2013-06-06 07:00:01

## Exploratory Data Analysis

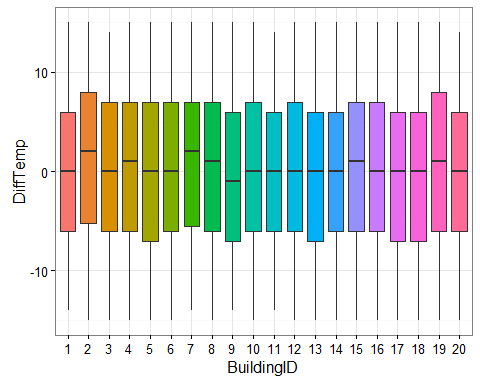
Let's visualize the temperature by sites. We will use the awesome [ggplot2](https://cran.r-project.org/web/packages/ggplot2/index.html) package for our visualizations.

Let's calculate the difference between ActualTemp and TargetTemp.

build\_temps$DiffTemp <- with(build\_temps, ActualTemp - TargetTemp)

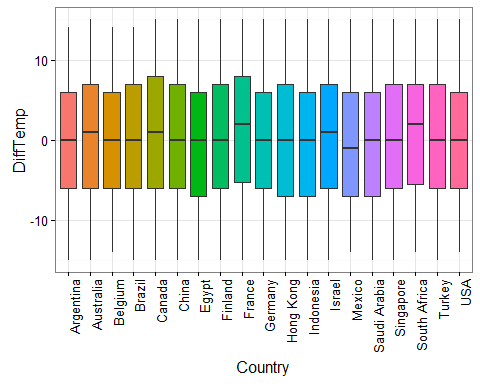
Let's use the ggplot library to visualize the temperature differences as boxplots. the ggplot2 package makes this easy to do. We pass the aesthetics to plot, in this case, the building id and the difference in temperature. Just to make the different IDs easier to compare, we'll fill by id as wel.

library(ggplot2)  
ggplot(build\_temps, aes(x = BuildingID, y = DiffTemp, fill = BuildingID)) + geom\_boxplot() + guides(fill = F) + theme\_bw()



Similarily for country:

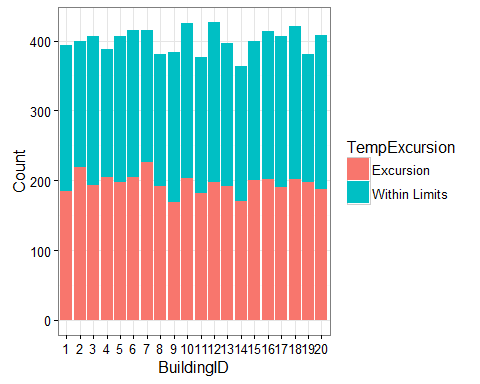
ggplot(build\_temps, aes(x = Country, y = DiffTemp, fill = Country)) + geom\_boxplot() + guides(fill = F) + theme\_bw() + theme(axis.text.x = element\_text(angle = 90, hjust = 1))



Suppose we just want to see which buildings had the most frequent excursions in temperature, where we'll define excursion as any data point where ActualTemp > TargetTemp.

We will use the dplyr package to aggregate the data.

library(dplyr)  
build\_temps <- build\_temps %>% mutate(TempExcursion = ifelse(DiffTemp > 0, "Excursion", "Within Limits"), TempFlag = ifelse(DiffTemp > 0, 1, 0))  
excursions <- build\_temps %>% group\_by(TempExcursion, BuildingID) %>% summarize(Count = n())  
ggplot(excursions, aes(x = BuildingID, y = Count, fill = TempExcursion)) + geom\_bar(stat = 'identity', position = 'stack') + theme\_bw()



# Predicting Modeling with R

Let's try to estimate the probability that there'll be a temperature excursion given building temperature attributes.

## Split Data into Training and Validation Set

In order to evaluate our model, we will need a set that we can use to estimate/train our model, and then another set that we can test our model. The test set should be separate from our training set.

The base package in R has a handy function named split for splitting data by the values of one of it's columns, and it returns a named list with an element for each value in the split column.

range(build\_temps$Date)

## [1] "2013-06-01" "2013-06-30"

build\_temps$SplitValue <- ifelse(build\_temps$Date > "2013-06-15", "Test", "Train")  
split\_df <- split(build\_temps, build\_temps$SplitValue)  
names(split\_df)

## [1] "Test" "Train"

lapply(split\_df, dim)

## $Test  
## [1] 3995 16  
##   
## $Train  
## [1] 4005 16

lapply(split\_df, function(x) range(x$Date))

## $Test  
## [1] "2013-06-16" "2013-06-30"  
##   
## $Train  
## [1] "2013-06-01" "2013-06-15"

## Train a Binomial Logistic Regression Model

Every modeling function in R starts with a formula to estimate. We will create our model formula, and then pass that to the glm function in the stats package to estimate a binary classification model, using the logistic link function.

temp\_formula <- TempFlag ~ BuildingID + BuildingMgr + BuildingAge + HVACproduct + Country + Time + System + SystemAge  
train\_model <- glm(temp\_formula, family = 'binomial', data = split\_df[["Train"]])

## Score Model on Test Set

Now that we have our trained model, we can score it on our test test.

predictions <- predict(train\_model, newdata = split\_df[["Test"]],   
 type = 'response')

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =  
## ifelse(type == : prediction from a rank-deficient fit may be misleading

summary(predictions)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.1434 0.4249 0.4911 0.4908 0.5607 0.7489

We will use the pROC package to estimate our model's accuracy.

library(pROC)

## Warning: package 'pROC' was built under R version 3.2.5

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

roc\_curve <- roc(split\_df[["Test"]]$TempFlag, predictions)

## Warning in roc$se: partial match of 'se' to 'sensitivities'

## Warning in roc$sp: partial match of 'sp' to 'specificities'

auc\_value <- auc(roc\_curve)

## Warning in roc$se: partial match of 'se' to 'sensitivities'  
  
## Warning in roc$se: partial match of 'sp' to 'specificities'

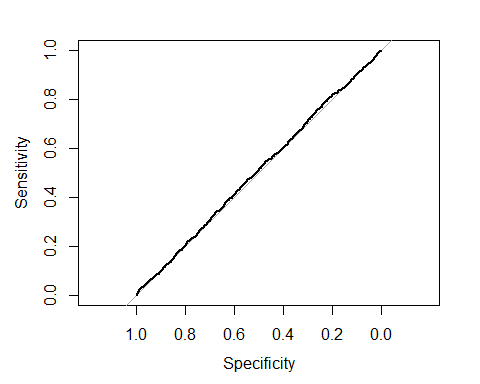
auc\_value

## Area under the curve: 0.5059

plot(roc\_curve)

## Warning in x$se: partial match of 'se' to 'sensitivities'

## Warning in x$sp: partial match of 'sp' to 'specificities'



##   
## Call:  
## roc.default(response = split\_df[["Test"]]$TempFlag, predictor = predictions)  
##   
## Data: predictions in 2052 controls (split\_df[["Test"]]$TempFlag 0) < 1943 cases (split\_df[["Test"]]$TempFlag 1).  
## Area under the curve: 0.5059