Analysis

Eduardo Ariño de la Rubia

December 14, 2014

First we load some libraries and make sure the code can take advantage of 6 of the cores on my laptop (it'll get it nice and hot!)

```
library(caret)
library(lubridate)
library(dplyr)
#library(doMC)
#registerDoMC(cores = 6)
```

Then we read the data

```
data <- read.csv("data.csv", stringsAsFactors=FALSE)
str(data)
```

```
## 'data.frame':    60779 obs. of  5 variables:
##  $ Timestamp                      : chr  "2013-01-01T00:15:00Z" "2013-01-01T00:30:00Z" "2013-01-01T0
##  $ Offset.from.UTC                : chr  "-03:00" "-03:00" "-03:00" "-03:00" ...
##  $ Energy.usage.per.interval..kWh.: num  38.2 38.2 38.2 38.2 37.4 ...
##  $ Interval..minutes.             : int  15 15 15 15 15 15 15 15 15 15 ...
##  $ Site.ID                        : int  38391 38391 38391 38391 38391 38391 38391 38391 38391 38391
```

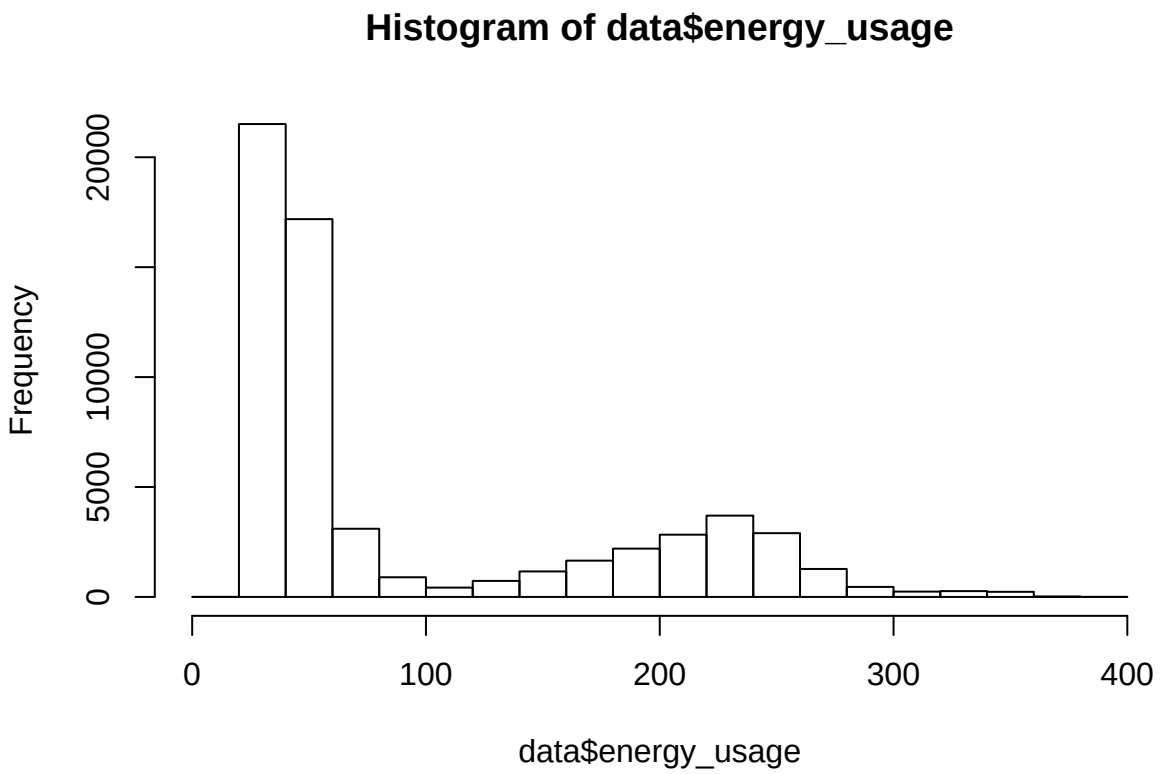
Not a lot to work with, we have 60779 rows and really, only two variables to work with, Timestamp and Energy usage. I hate the column names, so I'm going to go ahead and fix it.

```
colnames(data) <- c("timestamp", "utc_offset",
                    "energy_usage", "interval", "site_id")
```

That's better.

So, let's look at the distribution of data for the energy usage, that will give us an idea as to what our thesis should be:

```
hist(data$energy_usage)
```



So, what do we have to engineer features from? Well, the only real thing we have is the Timestamp variable. So, let's use lubridate and engineer atomic variables. I will be using Hadley's dplyr for this. Since wday returns 7 for sunday and 1 for saturday, it kind of messes up some stuff, so I'm going to go ahead and make 1 monday and 7 sunday.

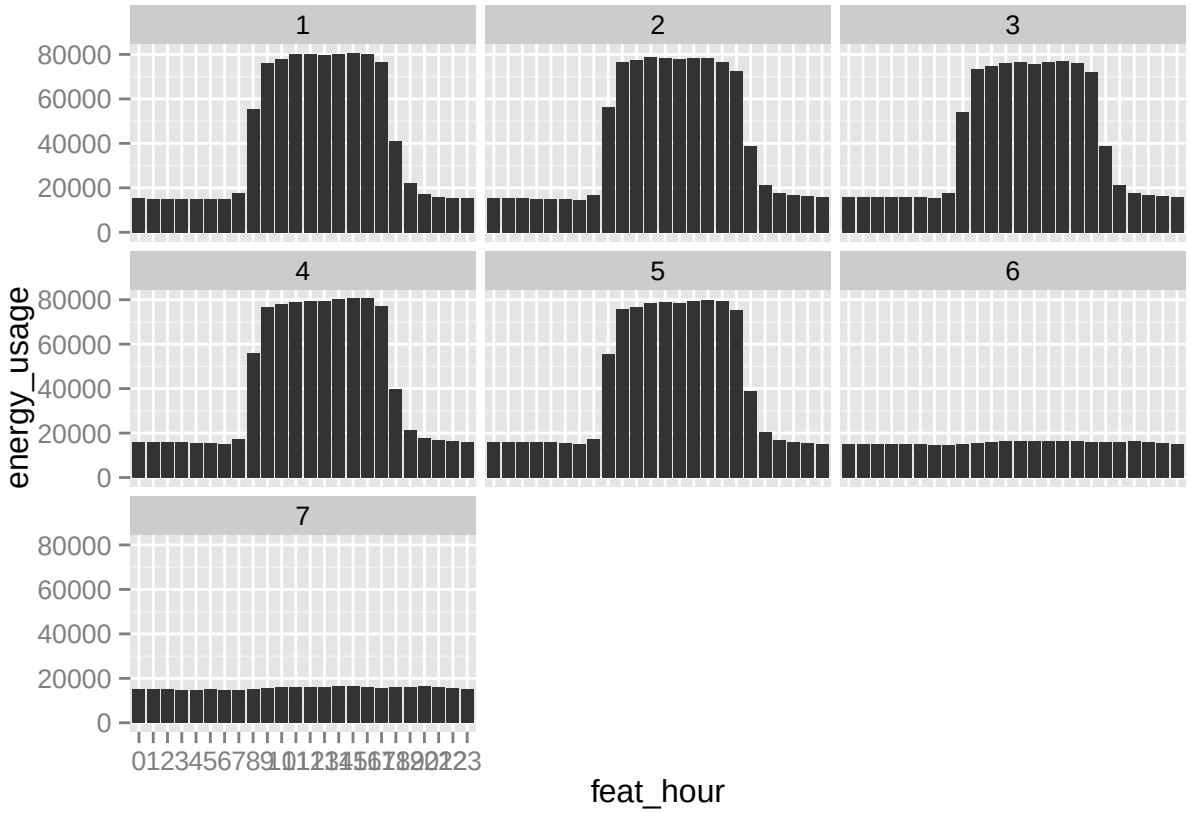
```
data %>% mutate(tstamp=ymd_hms(timestamp)) %>%
  mutate(feat_hour=hour(tstamp),
         feat_month=month(tstamp),
         feat_day_of_week=wday(tstamp)) %>%
  mutate(feat_day_of_week=feat_day_of_week-1) %>%
  mutate(feat_day_of_week=ifelse(feat_day_of_week == 0, 7, feat_day_of_week)) %>%
          mutate(feat_weekend=feat_day_of_week %in% c(6, 7)) %>%
  mutate(feat_hour=as.factor(feat_hour),
          feat_month=as.factor(feat_month),
          feat_day_of_week=as.factor(feat_day_of_week),
          feat_weekend=as.factor(feat_weekend)) -> data
```

So, what features did we extract:

- feat_hour - the hour of the reading
- feat_day_of_week - the day of the week
- feat_month - the month of the reading
- feat_weekend - a binary feature for whether things are a weekend or not

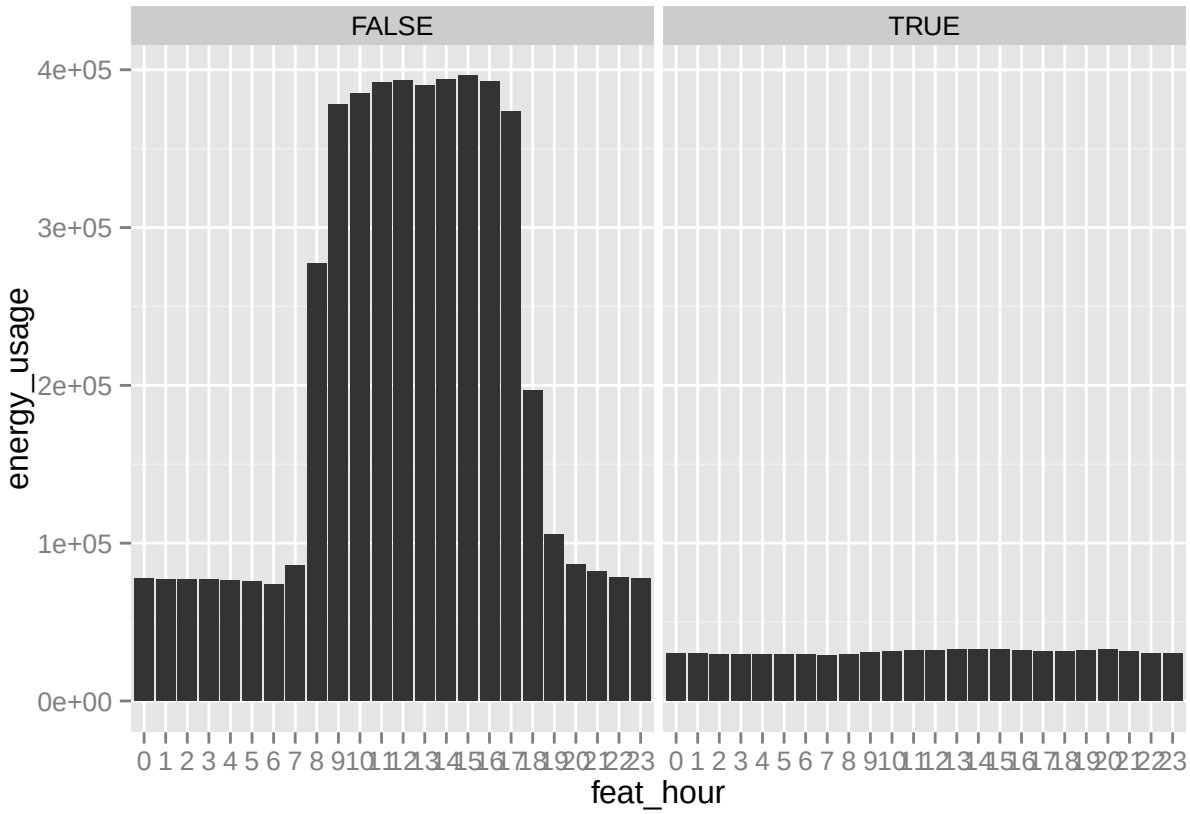
Ok, so now let's visualize the data with some of these new features, for starters let's look by hour of the day by day of week.

```
ggplot(data, aes(x=feat_hour, y=energy_usage)) +
  geom_bar(stat="identity") +
  facet_wrap(~ feat_day_of_week)
```



Pretty clear, this bank isn't open on Sundays or Saturdays (day 1 and 7). So, the feat_day_of_week feature is going to be silly predictive. We also created the "weekend" feature, which is a BINARY feature, which are incredibly powerful predictively, let's see what that graph looks like.

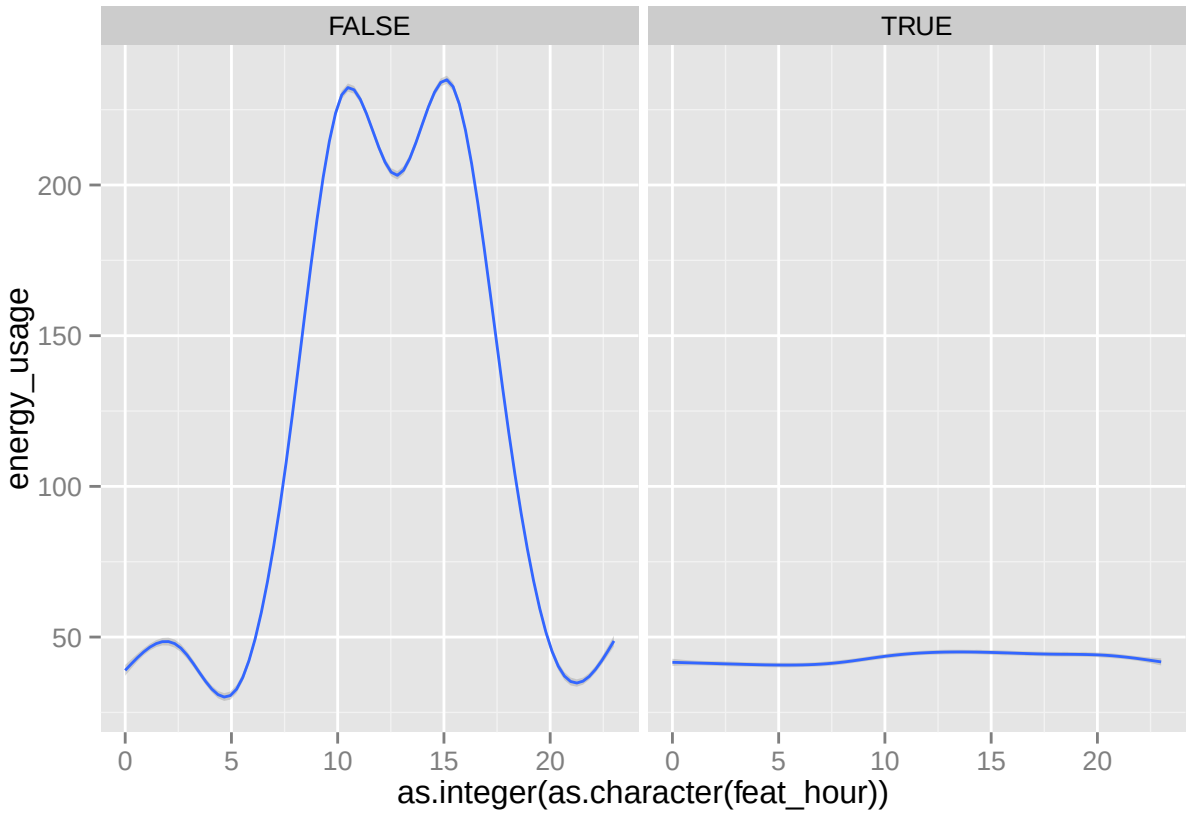
```
ggplot(data, aes(x=feat_hour, y=energy_usage)) +
  geom_bar(stat="identity") +
  facet_wrap(~ feat_weekend)
```



Yeah, there you go. Almost certainly feat_weekend is going to be part of this.

However, the curve we're trying to predict is very much that, it's a curve, so this isn't really a fit for a linear regression.

```
ggplot(data, aes(x=as.integer(as.character(feat_hour)), y=energy_usage)) +
  geom_smooth() +
  facet_wrap(~ feat_weekend)
```



The problem as you stated it is a bit tough:

```
I'm trying to extract features from it and train a model on this sample data that predicts whether the l
```

Well, we don't have a target label for "is open" or "is closed", so the first thing we can do is train a *regressor* instead of a classifier, and say, given the features we've extracted, how good a job can we do in predicting the kWH that will be used?

Let's bust out some caret, first we create a train/test split:

```
inTrain <- createDataPartition(data$energy_usage, p=.8, list=FALSE)
train <- data[inTrain,]
test <- data[-inTrain,]
```

Since we're trying to do some regression, we don't have a lot of predictors. We could pick something more complicated, but life is short. We're trying to fit non-linear data, but let's see how far a simple lm gets us.

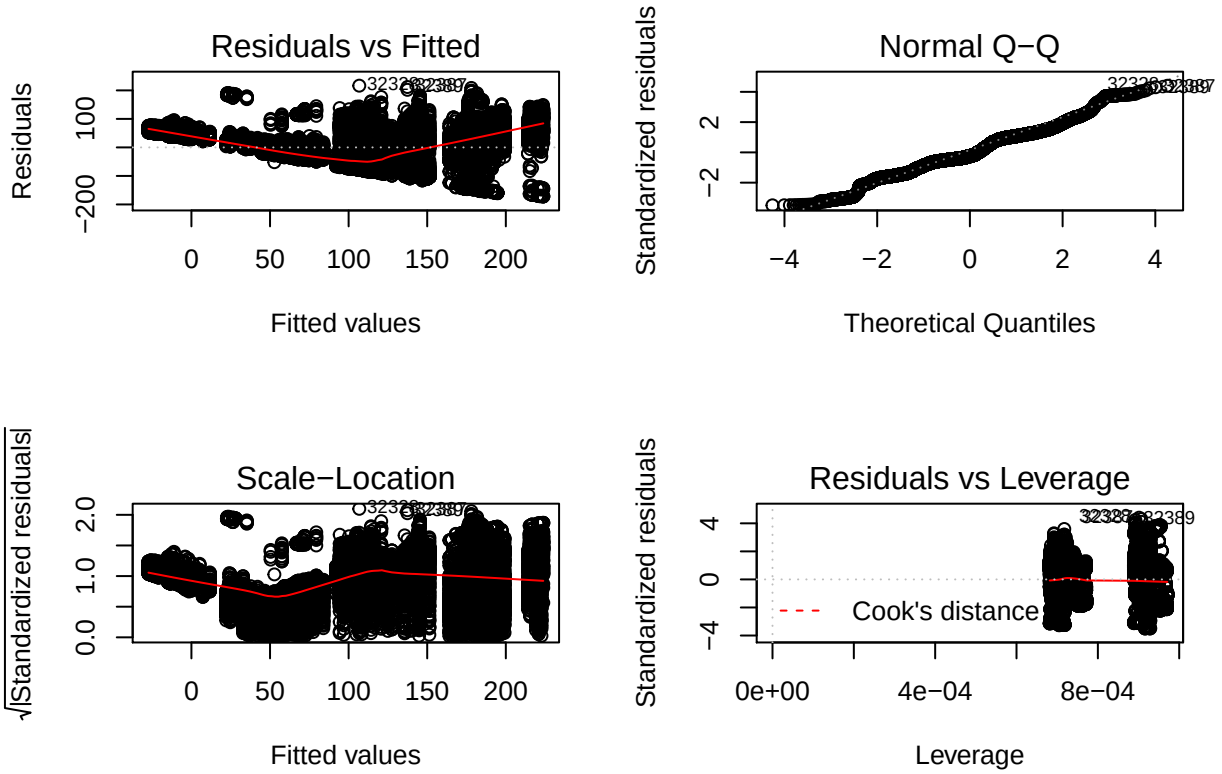
```
lm.fit <- train(energy_usage ~ feat_hour + feat_weekend + feat_month,
                data=train, method="lm")

summary(lm.fit$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -172.00  -28.29  -12.08   43.74  216.49
##
## Coefficients:
##                    Estimate Std. Error  t value Pr(>|t|)
## (Intercept)        59.61166    1.29520   46.025  < 2e-16 ***
## feat_hour1         -0.44269    1.54640   -0.286  0.77467
## feat_hour2          0.03918    1.54490    0.025  0.97977
## feat_hour3          0.47216    1.54849    0.305  0.76043
## feat_hour4         -0.84306    1.55021   -0.544  0.58656
## feat_hour5         -0.21542    1.54735   -0.139  0.88928
## feat_hour6         -0.71354    1.54795   -0.461  0.64483
## feat_hour7          2.92347    1.54546    1.892  0.05854 .
## feat_hour8         79.25720    1.55449   50.986  < 2e-16 ***
## feat_hour9        119.92893    1.54490   77.629  < 2e-16 ***
## feat_hour10       123.55671    1.54603   79.919  < 2e-16 ***
## feat_hour11       125.13335    1.55118   80.670  < 2e-16 ***
## feat_hour12       125.28228    1.54623   81.024  < 2e-16 ***
## feat_hour13       125.31671    1.55060   80.818  < 2e-16 ***
## feat_hour14       127.06533    1.54209   82.398  < 2e-16 ***
## feat_hour15       125.73438    1.55003   81.117  < 2e-16 ***
## feat_hour16       126.89773    1.55312   81.705  < 2e-16 ***
## feat_hour17       118.45585    1.55061   76.393  < 2e-16 ***
## feat_hour18        48.63142    1.54850   31.405  < 2e-16 ***
## feat_hour19        11.80504    1.54528    7.639 2.22e-14 ***
## feat_hour20         4.68914    1.54755    3.030  0.00245 **
## feat_hour21         2.96652    1.54946    1.915  0.05556 .
## feat_hour22         0.98575    1.55137    0.635  0.52517
## feat_hour23         0.71165    1.54680    0.460  0.64546
## feat_weekendTRUE -72.18154    0.49576 -145.599  < 2e-16 ***
## feat_month2         8.00999    1.03286    7.755 8.99e-15 ***
## feat_month3         1.66351    1.00755    1.651  0.09874 .
## feat_month4        -3.20174    1.01492   -3.155  0.00161 **
## feat_month5       -11.33053    1.01034  -11.215  < 2e-16 ***
## feat_month6       -13.91351    1.01479  -13.711  < 2e-16 ***
## feat_month7        -3.24232    1.00951   -3.212  0.00132 **
## feat_month8         5.98554    1.00909    5.932 3.02e-09 ***
## feat_month9        12.48190    1.03998   12.002  < 2e-16 ***
## feat_month10       36.01107    1.23437   29.174  < 2e-16 ***
## feat_month11       37.15966    1.25504   29.608  < 2e-16 ***
## feat_month12       -1.45715    1.23227   -1.182  0.23702
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 49.2 on 48589 degrees of freedom
## Multiple R-squared:  0.6555, Adjusted R-squared:  0.6553
## F-statistic:  2642 on 35 and 48589 DF,  p-value: < 2.2e-16
```

```
par(mfrow=c(2,2))
plot(lm.fit$finalModel)
```



An adjusted R-squared of 0.6552566 is not bad. 65% of variance is predictable just using the hour of the day, whether it's a weekend, and what month it is!

Ok, so I guess the next question is, let's engineer a label to "classify" against, we're going to make an is_open label. We're literally just going to make it by adding a column where if energy_usage > 60 kWh, it's open.

```
data %>% mutate(is_open=as.factor(ifelse(energy_usage > 60,
                                         TRUE, FALSE))) -> data
```

Ok, so let's now train a simple classifer on the previous values on this new is_open categorical value. First we create the new train/test split.

```
inTrain <- createDataPartition(data$is_open, p=.8, list=FALSE)
train <- data[inTrain,]
test <- data[-inTrain,]
```

So how good are we at predicting whether it's open (by which we mean more than 60 kWh used) using the hour, whether it's a weekend, and the month?

```
glm.fit <- train(as.factor(is_open) ~ feat_hour + feat_weekend + feat_month,
                 data=train, method="glm", trControl=trainControl(method="cv"))
```

Ok, our accuracy is 0.9362045! So, with the hour, the weekend, and the month, we can predict if we're going to use <60 kWh with close to 94% accuracy on the training set. How do we do on the test set?

```
confusionMatrix(test$is_open, predict(glm.fit, test))
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  7505  234
##      TRUE    558 3858
##
##                Accuracy : 0.9348
##                  95% CI : (0.9303, 0.9392)
##     No Information Rate : 0.6633
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8569
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9308
##             Specificity : 0.9428
##          Pos Pred Value : 0.9698
##          Neg Pred Value : 0.8736
##              Prevalence : 0.6633
##          Detection Rate : 0.6174
##    Detection Prevalence : 0.6367
##       Balanced Accuracy : 0.9368
##
##        'Positive' Class : FALSE
##
```

I'd say it's a good start.