R&I Kaggle Competition

Albert Xavier Lopez Barrantes and Alejandro Encalado Masia

28 de octubre de 2018

Introduction to the "Google Analytics Customer Revenue Prediction".

> The 80/20 rule has proven true for many businesses-only a small percentage of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies. RStudio, the developer of free and open tools for R and enterprise-ready products for teams to scale and share work, has partnered with Google Cloud and Kaggle to demonstrate the business impact that thorough data analysis can have. In this competition, we are challenged to analyze a Google Merchandise Store (also known as GStore, where Google swag is sold) customer dataset to predict revenue per customer. Hopefully, the outcome will be more actionable operational changes and a better use of marketing budgets for those companies who choose to use data analysis on top of GA data.

Since this competition is organized by Google in colaboration with R Studio, we are going to work on this dataset using R and this report will be made in Markdown. Notice that in this work we will try to predict the existence of transaction, not the money spend by transaction.

With all of this, let's start by defining the working directory and downloading the libraries we are going to use:

```
library(tidyverse)
library(tictoc)
library(jsonlite)
library(magrittr)
library(dplyr)
library(naniar)
library(reshape2)
library(ggplot2)
library(Matrix)
library(xgboost)
library(knitr)
```

In this competition, Kaggle gives us the dataset split in two files, "train" and "test", but we will work only with train set because we are going to split it in two parts in order to validate our models. Due to the size of the file, we implemented a simple function that is gives us the time of execution, so we can visualize the remaining time. Also, as the raw data contains some variables stored in JSON format, we are going to define some functions in order to clean the dataset:

```
# Function to visualize the dataset upload
load_info <- function(file){

  message(paste("Starting the upload..."))
  tic()
  train <- read.csv(file, header = TRUE)
  message(paste("------------------"))
  message(paste("End of the upload..."))
  toc()
  message(paste("------------------"))

  return(train)
```

```
}

# Function for remove NAs
remove_nas <- function(y, list){

  is_na_val <- function(x) x %in% list
  y <- mutate_if(y, is.factor,is.character)
  y <- mutate_if(y, is.logical,is.character)
  y <- y %>% mutate_all(funs(ifelse(is_na_val(.), NA, .)))
  y[is.na(y)] <- 0

  return(y)

}

# Function for the JSON format
clean_json <- function(data){

  message(paste("Starting the transformation for JSON format..."))

  library(jsonlite)
  flatten_json <- . %>%
    str_c(., collapse = ",") %>%
    str_c("[", ., "]") %>%
    fromJSON(flatten = T)



  parse <- . %>%
    bind_cols(flatten_json(.$device)) %>%
    bind_cols(flatten_json(.$geoNetwork)) %>%
    bind_cols(flatten_json(.$trafficSource)) %>%
    bind_cols(flatten_json(.$totals)) %>%
    select(-device, -geoNetwork, -trafficSource, -totals)

  tic()
  train <- parse(data)
  message(paste("-------------------"))
  message(paste("End of the transformation."))
  toc()
  message(paste("-------------------"))
  return(train)

}


# Function to handle useless information
trash_info <- function(x) {
  trash_list <- c()
  for (header in colnames(x)) {
    if (nrow(unique(x[paste(as.character(header))])) == 1) {
      trash_list <- c(trash_list,as.character(header))
    }
```

```
  }
  return(trash_list)
}

list_not_set <- c("not available in demo dataset", "(not provided)",
                                    "(not set)", "<NA>", "unknown.unknown","(none)")
```

Once defined the functions, we are going to upload the datasets and make use of the functions above to obtain information stored in JSON format.

```
# Loading dataset and cleaninng

train <- load_info("train.csv")
```

```
## Starting the upload...
```

```
## ------------------
```

```
## End of the upload...
```

```
## 34.099 sec elapsed
```

```
## ------------------
```

```
# Cleaning the JSON format
train <- clean_json(train)
```

```
## Starting the transformation for JSON format...
```

```
## ------------------
```

```
## End of the transformation.
```

```
## 66.772 sec elapsed
```

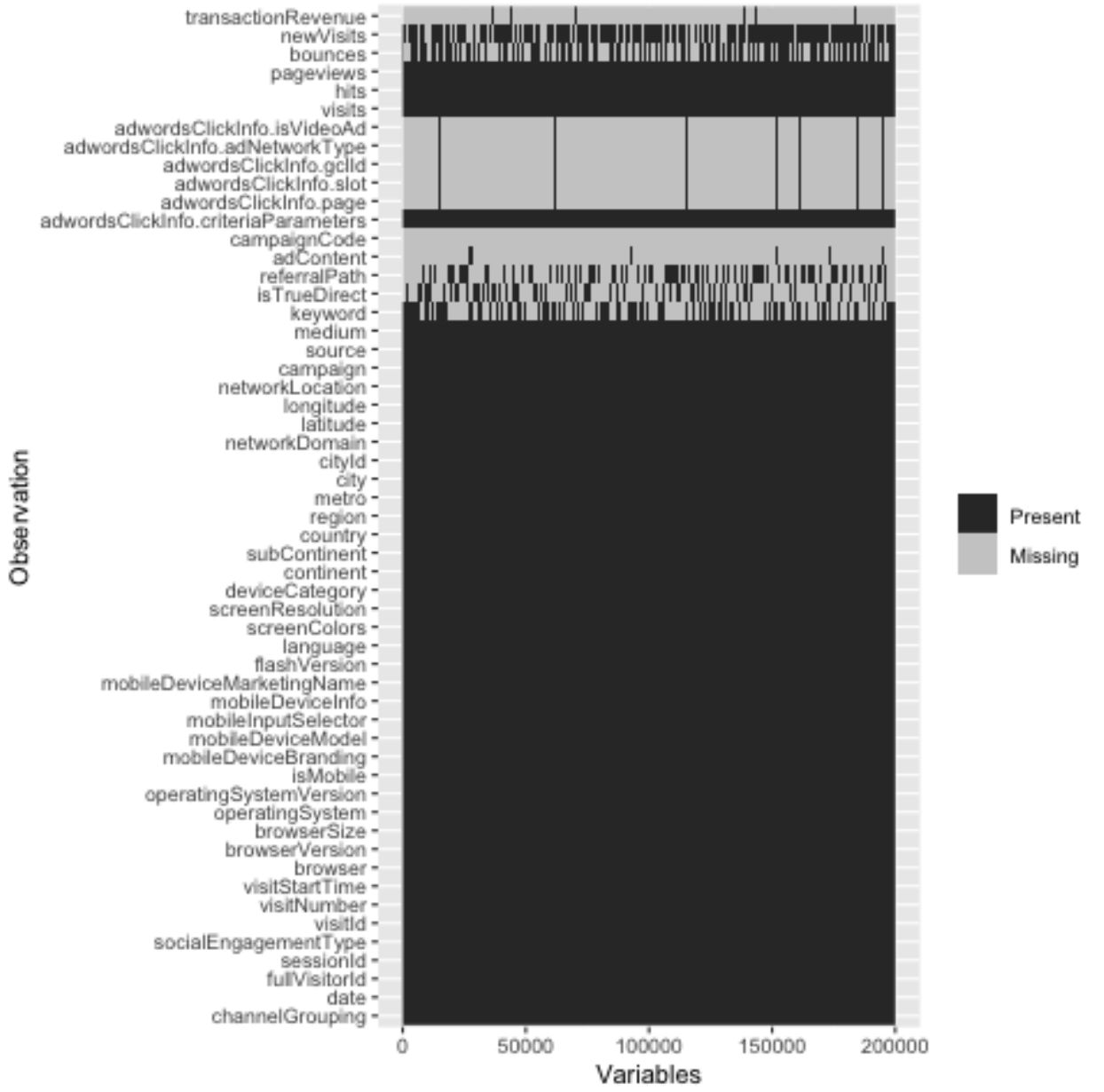
```
## ------------------
```

Now we have the dataset uploaded and ready to work on it, it's time to visualize the variables. To do so, we are going to perform a ggplot for missing values in each variable to identify the gaps in the dataset. Notice we just plot a subsample of 200.000 observation from the total of 900000 observation from the test dataset, the overall conclusions will be the same and will require less computation time:

```
png("g1.png")
g1 <- head(train,200000) %>%
  is.na %>% melt %>%
  ggplot(data = .,aes(y = Var1,x = Var2)) +
  geom_raster(aes(fill = value)) + coord_flip() +
  scale_fill_grey(name = "",labels = c("Present","Missing")) +
  labs(x = "Observation",y = "Variables")
print(g1)
dev.off()
```

```
## pdf
##   2
```

```
knitr::include_graphics("g1.png")
```



In this plot we can identify some variables which have almost all the observations with empty values, like for example all variables related to "Ads". Also we identify many variables that even not having any missing values, the values inside are not available like for example "data not available for this demo". All in all, we are going to use functions from the beggining to clean this kind of data. Finally we are going to transform the dataset into tibbles, a format for dataframes provided by "tidyverse" package which performs faster in big datasets.

```
train.tash <- trash_info(train)
trash.list <- names(train) %in% trash_info(train)
train <- train[!trash.list]
train <- remove_nas(train,list_not_set)

# Defining the dataset as a tibble
train.tib <- as.tibble(train)
```

We ended this piece of code by defining the dataset as.tibble. This function from the package "tidyverse" is more efficient when manipulating large datasets in R. Once we have the dataset uploaded and ready to work on it, it's time to visualize the variables. The advantage of working with tibbles is we can define the piece of dataset in the preview print and visualize very fast what type of variable each one is.

```
print(train.tib, n=7, width=Inf)
```

```
## # A tibble: 903,653 x 36
##   channelGrouping     date fullVisitorId sessionId    visitId visitNumber
##   <lgl>              <int>         <dbl> <lgl>          <int>       <int>
## 1 FALSE           20160902       1.13e18 FALSE     1472830385           1
## 2 FALSE           20160902       3.77e17 FALSE     1472880147           1
## 3 FALSE           20160902       3.90e18 FALSE     1472865386           1
## 4 FALSE           20160902       4.76e18 FALSE     1472881213           1
## 5 FALSE           20160902       2.73e16 FALSE     1472822600           2
## 6 FALSE           20160902       2.94e18 FALSE     1472807194           1
## 7 FALSE           20160902       1.91e18 FALSE     1472817241           1
##   visitStartTime browser    operatingSystem isMobile deviceCategory
##            <int> <chr>      <chr>           <lgl>    <chr>
## 1     1472830385 Chrome     Windows         FALSE    desktop
## 2     1472880147 Firefox    Macintosh       FALSE    desktop
## 3     1472865386 Chrome     Windows         FALSE    desktop
## 4     1472881213 UC Browser Linux           FALSE    desktop
## 5     1472822600 Chrome     Android         FALSE    mobile
## 6     1472807194 Chrome     Windows         FALSE    desktop
## 7     1472817241 Chrome     Windows         FALSE    desktop
##   continent subContinent    country        region              metro
##   <chr>     <chr>           <chr>          <chr>               <chr>
## 1 Asia      Western Asia    Turkey         Izmir               0
## 2 Oceania   Australasia     Australia      0                   0
## 3 Europe    Southern Europe Spain          Community of Madrid 0
## 4 Asia      Southeast Asia  Indonesia      0                   0
## 5 Europe    Northern Europe United Kingdom 0                   0
## 6 Europe    Southern Europe Italy          0                   0
## 7 Asia      Southern Asia   Pakistan       0                   0
##   city   networkDomain campaign source medium  keyword
##   <chr>  <chr>         <chr>    <chr>  <chr>   <chr>
## 1 Izmir  ttnet.com.tr  0        google organic 0
## 2 0      dodo.net.au   0        google organic 0
## 3 Madrid 0             0        google organic 0
## 4 0      0             0        google organic google + online
## 5 0      0             0        google organic 0
## 6 0      fastwebnet.it 0        google organic 0
## 7 0      0             0        google organic 0
##   isTrueDirect referralPath adContent campaignCode adwordsClickInfo.page
##   <lgl>        <chr>        <chr>     <chr>        <chr>
## 1 FALSE        0            0         0            0
## 2 FALSE        0            0         0            0
## 3 FALSE        0            0         0            0
## 4 FALSE        0            0         0            0
## 5 FALSE        0            0         0            0
## 6 FALSE        0            0         0            0
## 7 FALSE        0            0         0            0
##   adwordsClickInfo.slot adwordsClickInfo.gclId
##   <chr>                 <chr>
```

```
## 1 0                     0
## 2 0                     0
## 3 0                     0
## 4 0                     0
## 5 0                     0
## 6 0                     0
## 7 0                     0
##   adwordsClickInfo.adNetworkType adwordsClickInfo.isVideoAd hits
##   <chr>                          <lgl>                      <chr>
## 1 0                              FALSE                      1
## 2 0                              FALSE                      1
## 3 0                              FALSE                      1
## 4 0                              FALSE                      1
## 5 0                              FALSE                      1
## 6 0                              FALSE                      1
## 7 0                              FALSE                      1
##   pageviews bounces newVisits transactionRevenue
##   <chr>     <chr>   <chr>     <chr>
## 1 1         1       1         0
## 2 1         1       1         0
## 3 1         1       1         0
## 4 1         1       1         0
## 5 1         1       0         0
## 6 1         1       1         0
## 7 1         1       1         0
## # ... with 9.036e+05 more rows
```

Taking into account the information we get from the summary, we will to transform character variables into categorical variables to avoid problems when fitting models. Also, in this section we are going to avoid variables which have almost all observations with missing values, like we saw before, and those variables giving the same information.

```
MT.train <- select(train.tib,

################################################################################
##     add here new variables, they are already clean (NA -> 0)            ##
################################################################################


                   visitNumber,
                   medium,
                   isTrueDirect,
                   hits,
                   pageviews,
                   bounces,
                   newVisits,
                   country,
                   operatingSystem,
                   deviceCategory,
                   browser,
                   subContinent,
                   date,
                   transactionRevenue)
```

```
#############################################################################

##      Most relevant information in subContinent variable               ##

#############################################################################


 MT.train$aux_ind <- 0


 MT.train$aux_ind[MT.train$subContinent == "Northern America"] <- 1
 MT.train$aux_ind[MT.train$subContinent == "South America"] <- 1
 MT.train$aux_ind[MT.train$subContinent == "Eastern Asia"] <- 1




 MT.train$subContinent[MT.train$aux_ind == 0] <- "other"


 MT.train <- select(MT.train, -aux_ind)



#############################################################################

MT.train <- mutate(MT.train,

#############################################################################
##                   Formating variables                                   ##
#############################################################################

                   transactionRevenue = as.double(transactionRevenue),

                   visitNumber = as.integer(visitNumber),
                   medium = as.factor(medium),
                   isTrueDirect = as.integer(isTrueDirect),
                   hits = as.integer(hits),
                   pageviews = as.integer(pageviews),
                   bounces = as.integer(bounces),
                   newVisits = as.integer(newVisits),
                   country = as.factor(country),
                   subContinent = as.factor(subContinent),
                   operatingSystem = as.factor(operatingSystem),
                   deviceCategory=as.factor(deviceCategory),
                   browser=as.factor(browser),
                   date = as.Date(as.character(date),"%Y%m%d"))
```

```
##############################################################################
##     Conversion of transaction revenue                                    ##
##############################################################################

MT.train$transaction[MT.train$transactionRevenue > 0] <- 1
MT.train$transaction[MT.train$transactionRevenue == 0] <- 0
MT.train <- select(MT.train , -transactionRevenue)
```

At this point we have all work done on the dataset and it's ready to be used and apply some models on it. we decided we had to become the problem into a binary solution first.

As we commented bellow, we are going to predict number if there is transaction or not.The reason is only 1% of the observations buy something in the GStore, so if we try to predict an amount of money spend it would be impossible from the beginning since most of them don't spend anything. So we created the variable "transaction" to use as a target variable in a binary problem. After this decision, the problem becomes easier in conceptual terms since we know we have to apply models for a binary classification problem. Of course, the first one that came into our mind was the Logistic Model.

Logistic Regression Model

The first model we have choosen is the logistic regression model. The model is based on simple regression, but instead of predict a continous variable, it will try to predict a binary value.

```
# Subsamples
train1<-MT.train

#Logistic function
logis<-glm(transaction ~ medium + deviceCategory + hits + pageviews + subContinent + newVisits
           + visitNumber, data=train1, family="binomial")
summary(logis)
```

```
##
## Call:
## glm(formula = transaction ~ medium + deviceCategory + hits +
##     pageviews + subContinent + newVisits + visitNumber, family = "binomial",
##     data = train1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.1138  -0.0300  -0.0192   5.4573
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -7.577182   0.168073 -45.083  < 2e-16 ***
## mediumaffiliate             -2.209818   0.351269  -6.291 3.16e-10 ***
## mediumcpc                   -0.279361   0.061444  -4.547 5.45e-06 ***
## mediumcpm                   -0.466465   0.101477  -4.597 4.29e-06 ***
## mediumorganic               -0.323725   0.033654  -9.619  < 2e-16 ***
## mediumreferral               0.251877   0.031803   7.920 2.37e-15 ***
## deviceCategorymobile        -1.081621   0.042464 -25.472  < 2e-16 ***
## deviceCategorytablet        -1.228830   0.100323 -12.249  < 2e-16 ***
## hits                        -0.199656   0.003222 -61.965  < 2e-16 ***
```

```
## pageviews                      0.376510   0.004509  83.500  < 2e-16 ***
## subContinentNorthern America  3.444931   0.165178  20.856  < 2e-16 ***
## subContinentSouth America     0.524568   0.226850   2.312   0.0208 *
## subContinentother            -0.118190   0.182360  -0.648   0.5169
## newVisits                    -1.067247   0.023904 -44.647  < 2e-16 ***
## visitNumber                  -0.013241   0.001237 -10.705  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 123358  on 903652  degrees of freedom
## Residual deviance:  69211  on 903638  degrees of freedom
## AIC: 69241
##
## Number of Fisher Scoring iterations: 10
```

```
#Predictions
p1<-predict(logis, train1, type = "response")
head(p1)
```

```
##            1            2            3            4            5
## 0.0001333059 0.0001333059 0.0001333059 0.0001333059 0.0001296755
##            6
## 0.0001333059
```

```
#Classification error:
pred1<-ifelse(p1>0.5, 1, 0)
confusion.matrix<-table(Predicted= pred1, Actual=train1$transaction)
accuracy <- diag(confusion.matrix)/sum(confusion.matrix)

print(confusion.matrix)
```

```
##          Actual
## Predicted      0      1
##         0 890111   9391
##         1   2027   2124
```

```
print(accuracy[1])
```

```
##         0
## 0.9850142
```

Overall, we ended having the variables with more p-value, which mean they have more power of disrimination. From this first model we can see how the number of hits sombedy does when navigating in the website, the number of pageviews and the device they are using, can be explaining part of the possible outcome.

We have set the initial break point to build our predictions as 0.5 and doing so que get the confusion matrix above. Having 11.000 of posibles "1" outcomes, almost 80% of them are well classified, but then the model predicts many "1" that shouldn't be. This started to made us think about the imbalance of the dataset and how difficult is for any model to perform well when the training set has that imbalance.

Extreme gradient boosting Model (XGB)

Our second option is a model from the package xgboost. The model we have chose it is called extreme gradient boosting. This model is based on gradient descent, for each iteration, it is computed a "weak" learner

based on gradient descent algorithm. It is called extreme because it improves the computation in comparison with other packages related to this algorithm. For instance, xgboost package supports parallel computation and it is supposed to be arround 10 times faster than the other gradient boosting packages.

The computation is performed in order to find a model $F(\bar{x})$ where the mean squared error is minimum. $MSE(x) = \frac{1}{N}\sum_1^N (y_i - F(x_i))^2$ Where y i s the target value, which is the quantity we want to predict.

Gradient boosting algorithm assumes that at some point of the iteration it exist a model where $F_m(\bar{x}) = F_{m-1}(\bar{x}) + h(\bar{x}) = \bar{y}$. So, at each step, $h(\bar{x})$ changes, and it is used to improve the next iteration.

```
#Creation train and validation sets with the same proportions of transaction #

MT.train.transaction <- filter(MT.train, transaction == 1)
ind_transaction <- sample(2, nrow(MT.train.transaction),
                          replace = T, prob = c(0.5,0.5))

MT.train.no.transaction <- filter(MT.train, transaction == 0)
ind_no.transaction <- sample(2, nrow(MT.train.no.transaction),
                             replace = T, prob = c(0.5,0.5))

GB.train.1 <- MT.train.transaction[ind_transaction == 1,]
GB.train.2 <- MT.train.no.transaction[ind_no.transaction == 1,]

GB.train <- rbind(GB.train.1,GB.train.2)

ind_rand_train <- sample(nrow(GB.train), nrow(GB.train), replace = F)

GB.train <- cbind(GB.train, ind_rand_train)
GB.train <- arrange(GB.train, ind_rand_train)

GB.val.1 <- MT.train.transaction[ind_transaction == 2,]
GB.val.2 <- MT.train.no.transaction[ind_no.transaction == 2,]

GB.val <- rbind(GB.val.1,GB.val.2)

ind_rand_val <- sample(nrow(GB.val), nrow(GB.val), replace = F)
GB.val <- cbind(GB.val, ind_rand_val)
GB.val <- arrange(GB.val, ind_rand_val)

# Substracting the variables we don't need for gradient boosting algorithm
GB.train <- select(GB.train,-isTrueDirect, -ind_rand_train, -date, -country)
GB.val <- select(GB.val,-isTrueDirect, -ind_rand_val,-date,-country)
```

```
#Format factors into dummy variables
trainm <- sparse.model.matrix(transaction ~. -transaction, data = GB.train )
valm <- sparse.model.matrix(transaction ~. -transaction, data = GB.val )

#Setting the labels
train_label <- GB.train$transaction
val_label <- GB.val$transaction

#Creating the matrix for the model
train_matrix <-xgb.DMatrix(data = as.matrix(trainm), label = train_label)
val_matrix <-xgb.DMatrix(data = as.matrix(valm), label = val_label)
```

```r
#Running x gradient boosting model & making the prediction table
model <- xgboost(data = train_matrix, nrounds = 10)
```

```
## [1]  train-rmse:0.356482
## [2]  train-rmse:0.258452
## [3]  train-rmse:0.192885
## [4]  train-rmse:0.150544
## [5]  train-rmse:0.124489
## [6]  train-rmse:0.109331
## [7]  train-rmse:0.100953
## [8]  train-rmse:0.096386
## [9]  train-rmse:0.093972
## [10] train-rmse:0.092698
```

```r
predictionxgb <- predict(model, valm)

predictionxgb <- ifelse(predictionxgb> 0.4 , 1, 0)

#Confusion matrix and accuracy
confusion.matrix <- table(Predicted = predictionxgb, Actual = GB.val$transaction)
accuracy <- diag(confusion.matrix)/sum(confusion.matrix)

print(confusion.matrix)
```

```
##           Actual
## Predicted      0      1
##         0 443960   4012
##         1   1734   1816
```

```r
print(accuracy[1])
```

```
##         0
## 0.9832522
```

We observed that this model gets stuck when the error reaches 0.09 and althought the accuracy is high (~0.98%), we cannot conclude this is not the best prediction we could get. Just ~44% transactions are predicted, and the aim of the problem is to find who would buy products, not who wouldn't. So, or gradient boosting is not the proper model, or we didn't perform a proper feature engineering.

Conclusions

In our opinion, the point is that we haven't select the correct variables to do our prediction, as we shown bellow, the criteria was that all the variables we don't know what they are were refused, and some of those we select, were changed in order to no overfit the models. For instance, country variable was discarted because the information is already contained in subContinent. And we renamed all the subcontinents that has no impact on the transaction. That is why we just take into account transactions comming from North America, South America and Eastern Asia.

The model performance of xgboost and logistic regression is almost the same for the variables we have chosen, the accuracy is arround 0.98%, however, as we said, this is not what we want, but we have seen that both models tells us the same.

All in all, we can conclude that the key to have good predictions is to feed the model with the correct with "good" variables and observations. And with the dataset we have chosen for this exercice we have done our best. Hopefully, in the future we will try to improve our criteria to select the correct variables and observations.

```r
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.14
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] bindrcpp_0.2.2  knitr_1.20      xgboost_0.71.2  Matrix_1.2-14
##  [5] reshape2_1.4.3  naniar_0.4.0.0  magrittr_1.5    jsonlite_1.5
##  [9] tictoc_1.0      forcats_0.3.0   stringr_1.3.1   dplyr_0.7.5
## [13] purrr_0.2.4     readr_1.1.1     tidyr_0.8.1     tibble_1.4.2
## [17] ggplot2_3.0.0   tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.4  haven_1.1.2       lattice_0.20-35
##  [4] colorspace_1.3-2  htmltools_0.3.6   yaml_2.2.0
##  [7] utf8_1.1.4        rlang_0.2.0       pillar_1.2.3
## [10] glue_1.2.0        withr_2.1.2       modelr_0.1.2
## [13] readxl_1.1.0      bindr_0.1.1       plyr_1.8.4
## [16] munsell_0.4.3     gtable_0.2.0      cellranger_1.1.0
## [19] rvest_0.3.2       evaluate_0.11     labeling_0.3
## [22] broom_0.5.0       Rcpp_0.12.19      scales_0.5.0
## [25] backports_1.1.2   hms_0.4.2         digest_0.6.15
## [28] stringi_1.2.2     visdat_0.5.1      grid_3.5.0
## [31] rprojroot_1.3-2   cli_1.0.0         tools_3.5.0
## [34] lazyeval_0.2.1    crayon_1.3.4      pkgconfig_2.0.1
## [37] data.table_1.11.8 xml2_1.2.0        lubridate_1.7.4
## [40] assertthat_0.2.0  rmarkdown_1.10    httr_1.3.1
## [43] rstudioapi_0.8    R6_2.2.2          nlme_3.1-137
## [46] compiler_3.5.0
```