Modeling yellow-cab taxi data from NYC

Sergio Ramirez

August 30, 2018

In this document, we focus on applying the insights obtained from the previous analytic document by transforming data in a better shape, and creating a simple predictive model from that cleaned data.

In this case, we take a subsample of 1 millions rows from each monthly dataset in order to speed up computations in this preliminary analysis. As future work, data can be furtherly investigated using a proper set of big data platforms, such as Spark.

Data load and preparation

```
raw_data_march <- read.csv("~/Documentos/others/yellow_2017-03_1M.csv", header = F)
raw_data_june <- read.csv("~/Documentos/others/yellow_2017-06_1M.csv", header = F)
raw_data_november <- read.csv("~/Documentos/others/yellow_2017-11_1M.csv", header = F)

colnames_all <- c("VendorID","tpep_pickup_datetime","tpep_dropoff_datetime","passenger_count","trip_dis
raw_train <- rbind(raw_data_march, raw_data_june, raw_data_november)
colnames(raw_train) <- colnames_all

rm(raw_data_march)
rm(raw_data_june)
rm(raw_data_november)
```

Re-format features

First we transform datetime columns into POSIXct features in order to compute trip duration variable (time difference between DropOff (DO) and PickUp (PU) datetimes). We also coerce categorical variables to factor type.

```
parse_datetime <- function(data) {
  data$tpep_pickup_datetime <- as.POSIXct(as.character(data$tpep_pickup_datetime), format = "%Y-%m-%d %
  data$tpep_dropoff_datetime <- as.POSIXct(as.character(data$tpep_dropoff_datetime), format = "%Y-%m-%d
  data
}

raw_train <- parse_datetime(raw_train)
summary(raw_train)
```

```
##     VendorID     tpep_pickup_datetime
##  Min.   :1.000   Min.   :2017-03-01 00:00:12
##  1st Qu.:1.000   1st Qu.:2017-03-17 06:50:13
##  Median :2.000   Median :2017-06-02 12:34:49
##  Mean   :1.545   Mean   :2017-05-10 12:57:00
##  3rd Qu.:2.000   3rd Qu.:2017-06-17 18:46:02
##  Max.   :2.000   Max.   :2017-11-30 23:59:58
##  tpep_dropoff_datetime         passenger_count trip_distance
##  Min.   :2017-03-01 00:01:21   Min.   :0.000   Min.   :   0.000
##  1st Qu.:2017-03-17 07:06:30   1st Qu.:1.000   1st Qu.:   0.980
```

```
##  Median :2017-06-02 12:53:43   Median :1.000   Median :   1.610
##  Mean   :2017-05-10 13:13:37   Mean   :1.621   Mean   :   2.935
##  3rd Qu.:2017-06-17 19:00:47   3rd Qu.:2.000   3rd Qu.:   3.040
##  Max.   :2017-12-01 19:55:21   Max.   :9.000   Max.   :6545.780
##    RatecodeID     store_and_fwd_flag  PULocationID    DOLocationID
##  Min.   : 1.000   N:2090463          Min.   :  1.0   Min.   :  1.0
##  1st Qu.: 1.000   Y:   9537          1st Qu.:114.0   1st Qu.:107.0
##  Median : 1.000                      Median :162.0   Median :162.0
##  Mean   : 1.044                      Mean   :162.8   Mean   :160.7
##  3rd Qu.: 1.000                      3rd Qu.:233.0   3rd Qu.:233.0
##  Max.   :99.000                      Max.   :265.0   Max.   :265.0
##   payment_type    fare_amount           extra             mta_tax
##  Min.   :1.000   Min.   :  -259.00   Min.   :-53.7100   Min.   :-0.5000
##  1st Qu.:1.000   1st Qu.:     6.50   1st Qu.:  0.0000   1st Qu.: 0.5000
##  Median :1.000   Median :     9.50   Median :  0.0000   Median : 0.5000
##  Mean   :1.332   Mean   :    13.14   Mean   :  0.3396   Mean   : 0.4973
##  3rd Qu.:2.000   3rd Qu.:    14.50   3rd Qu.:  0.5000   3rd Qu.: 0.5000
##  Max.   :4.000   Max.   :171861.78   Max.   : 11.5000   Max.   :54.5100
##    tip_amount       tolls_amount      improvement_surcharge
##  Min.   : -7.000   Min.   :-15.0000   Min.   :-0.3000
##  1st Qu.:  0.000   1st Qu.:  0.0000   1st Qu.: 0.3000
##  Median :  1.360   Median :  0.0000   Median : 0.3000
##  Mean   :  1.867   Mean   :  0.3233   Mean   : 0.2996
##  3rd Qu.:  2.450   3rd Qu.:  0.0000   3rd Qu.: 0.3000
##  Max.   :337.060   Max.   :548.8700   Max.   : 1.0000
##   total_amount
##  Min.   :  -259.30
##  1st Qu.:     8.70
##  Median :    11.80
##  Mean   :    16.47
##  3rd Qu.:    17.80
##  Max.   :171863.58
```

Now, we will do some creative stuff with feature engineering. We create one variable that describes the duration of trips, and others to model stational patterns in pickup and dropoff events. For instance, we believe people are happier during weekends so they will likely give higher tips those days.

Day of month is not included because we think monthly seasonality is extremely scarce (https://robjhyndman.com/hyndsight/monthly-seasonality/). Also we have not added variables related to year seasonality as our data do not extende beyond a single year unit. Finally, absolute datetime variable (timestamp) has also been incorporated to the model.

```
featura_engineering <- function(data) {
  data %>% mutate(
    trip_duration = as.numeric(difftime(tpep_dropoff_datetime, tpep_pickup_datetime, units = "secs")),
    hourDO = as.numeric(strftime(tpep_dropoff_datetime, format = "%H")),
    hourPU = as.numeric(strftime(tpep_pickup_datetime, format = "%H")),
    weekday = as.numeric(strftime(tpep_dropoff_datetime, format = "%w")),
    timestamp = as.numeric(tpep_dropoff_datetime)
  )
}


# Coerce types to include time-based features in encoding.
categ_features <- c("hourDO", "hourPU", "weekday", "PULocationID", "DOLocationID", "VendorID", "Ratecod
```

```r
coerce_types <- function(data) {
  data[categ_features] <- lapply(data[categ_features], as.factor)
  # Remove datetime features from data
  data[,-which(names(data) %in% c("tpep_dropoff_datetime", "tpep_pickup_datetime"))]
}

raw_train <- featura_engineering(raw_train)
raw_train <- coerce_types(raw_train)

summary(raw_train)
```

```
##  VendorID    passenger_count trip_distance      RatecodeID
##  1: 955014   Min.   :0.000   Min.   :   0.000   1 :2040295
##  2:1144986   1st Qu.:1.000   1st Qu.:   0.980   2 :  46941
##              Median :1.000   Median :   1.610   3 :   4399
##              Mean   :1.621   Mean   :   2.935   4 :   1149
##              3rd Qu.:2.000   3rd Qu.:   3.040   5 :   7154
##              Max.   :9.000   Max.   :6545.780   6 :     16
##                                                 99:     46
##  store_and_fwd_flag  PULocationID      DOLocationID     payment_type
##  N:2090463          237    :  80365   161    :  76613   1:1420612
##  Y:   9537          161    :  77887   236    :  76034   2: 664911
##                     236    :  73111   237    :  70508   3:  11220
##                     186    :  72037   170    :  67549   4:   3257
##                     162    :  71444   230    :  64561
##                     230    :  70518   162    :  62728
##                     (Other):1654638   (Other):1682007
##   fare_amount           extra              mta_tax
##  Min.   :  -259.00   Min.   :-53.7100   Min.   :-0.5000
##  1st Qu.:     6.50   1st Qu.:  0.0000   1st Qu.: 0.5000
##  Median :     9.50   Median :  0.0000   Median : 0.5000
##  Mean   :    13.14   Mean   :  0.3396   Mean   : 0.4973
##  3rd Qu.:    14.50   3rd Qu.:  0.5000   3rd Qu.: 0.5000
##  Max.   :171861.78   Max.   : 11.5000   Max.   :54.5100
##
##    tip_amount       tolls_amount      improvement_surcharge
##  Min.   : -7.000   Min.   :-15.0000   Min.   :-0.3000
##  1st Qu.:  0.000   1st Qu.:  0.0000   1st Qu.: 0.3000
##  Median :  1.360   Median :  0.0000   Median : 0.3000
##  Mean   :  1.867   Mean   :  0.3233   Mean   : 0.2996
##  3rd Qu.:  2.450   3rd Qu.:  0.0000   3rd Qu.: 0.3000
##  Max.   :337.060   Max.   :548.8700   Max.   : 1.0000
##
##   total_amount       trip_duration          hourDO
##  Min.   :  -259.30   Min.   : -3167.0   1      : 131659
##  1st Qu.:     8.70   1st Qu.:   398.0   2      : 128054
##  Median :    11.80   Median :   673.0   3      : 118968
##  Mean   :    16.47   Mean   :   996.8   4      : 117071
##  3rd Qu.:    17.80   3rd Qu.:  1111.0   0      : 116133
##  Max.   :171863.58   Max.   :109096.0   5      : 110168
##                                         (Other):1377947
##      hourPU        weekday      timestamp
##  1      : 130161   0:281065   Min.   :1.488e+09
##  2      : 123553   1:235687   1st Qu.:1.490e+09
```

```
##  0       : 119787   2:249416   Median :1.496e+09
##  4       : 117552   3:292069   Mean   :1.494e+09
##  3       : 117175   4:341468   3rd Qu.:1.498e+09
##  21      : 108938   5:367787   Max.   :1.512e+09
##  (Other):1382834   6:332508
```

Outliers were analyzed and identified in the previous document but here we grab some extra information from location-based features. We noticed that some drop-off locations were never used as pick-up locations, somekind of weird.

Beyond that, we filter out those registers we believe can be cleary stated as outliers. For example, negative distances and durations, ghostly trips (no passengers on board), etc. Feature values for outliers may be set to NaN as our algorithms is able to deal with these values, but we believe it is much faster to just remove them.

```
# How many distinct locations
raw_train %>% distinct(PULocationID) %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   253
```

```
raw_train %>% distinct(DOLocationID) %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   262
```

```
# it seems there is some drop-off locations are not present in the pickup set, somekind of weird
distinctPU <- raw_train %>% distinct(PULocationID)
distinctDO <- raw_train %>% distinct(DOLocationID)

# how many outliers we have, not missing data
outliers <- raw_train %>% filter(passenger_count <= 0 | passenger_count >= 10 | trip_distance <= 0 & tri

# less than 1% of data are outliers, we can force them to NaN, or even safely remove them
nrow(outliers) / nrow(raw_train)
```

```
## [1] 0.003531905
```

```
# cleaned data
clean_data_from_outliers <- function(data) {
  data %>% filter(!(passenger_count <= 0 | passenger_count >= 10 | trip_distance <= 0 & trip_distance >
}

raw_train <- clean_data_from_outliers(raw_train)

raw_train %>% count()
```

```
## # A tibble: 1 x 1
##         n
##     <int>
## 1 2092583
```

```
summary(raw_train)
```

```
##  VendorID    passenger_count trip_distance     RatecodeID
##  1: 952177   Min.   :1.000   Min.   :  0.000   1 :2034006
```

```
##  2:1140406   1st Qu.:1.000   1st Qu.:  0.980   2 :  46575
##              Median :1.000   Median :  1.620   3 :   4356
##              Mean   :1.621   Mean   :  2.931   4 :   1120
##              3rd Qu.:2.000   3rd Qu.:  3.040   5 :   6504
##              Max.   :9.000   Max.   :202.600   6 :     16
##                                                99:      6
##  store_and_fwd_flag  PULocationID      DOLocationID     payment_type
##  N:2083135          237    :  80155   161    :  76454   1:1417998
##  Y:   9448          161    :  77673   236    :  75910   2: 661343
##                     236    :  72937   237    :  70377   3:  10319
##                     186    :  71806   170    :  67394   4:   2923
##                     162    :  71259   230    :  64363
##                     230    :  70261   162    :  62594
##                     (Other):1648492   (Other):1675491
##   fare_amount         extra          mta_tax         tip_amount
##  Min.   :  0.01   Min.   :-0.50   Min.   :0.0000   Min.   :  0.000
##  1st Qu.:  6.50   1st Qu.: 0.00   1st Qu.:0.5000   1st Qu.:  0.000
##  Median :  9.50   Median : 0.00   Median :0.5000   Median :  1.360
##  Mean   : 13.05   Mean   : 0.34   Mean   :0.4979   Mean   :  1.869
##  3rd Qu.: 14.50   3rd Qu.: 0.50   3rd Qu.:0.5000   3rd Qu.:  2.450
##  Max.   :975.00   Max.   : 6.52   Max.   :7.6800   Max.   :337.060
##
##   tolls_amount      improvement_surcharge  total_amount
##  Min.   : -5.7600   Min.   :0.0           Min.   :  0.10
##  1st Qu.:  0.0000   1st Qu.:0.3           1st Qu.:  8.75
##  Median :  0.0000   Median :0.3           Median : 11.80
##  Mean   :  0.3222   Mean   :0.3           Mean   : 16.38
##  3rd Qu.:  0.0000   3rd Qu.:0.3           3rd Qu.: 17.80
##  Max.   :548.8700   Max.   :1.0           Max.   :975.30
##
##  trip_duration         hourDO            hourPU        weekday
##  Min.   :    1.0   1      : 131218   1      : 129722   0:279957
##  1st Qu.:  399.0   2      : 127652   2      : 123143   1:234667
##  Median :  673.0   3      : 118612   0      : 119380   2:248543
##  Mean   :  876.5   4      : 116707   4      : 117196   3:291166
##  3rd Qu.: 1109.0   0      : 115758   3      : 116809   4:340411
##  Max.   :10786.0   5      : 109831   21     : 108509   5:366503
##                    (Other):1372805   (Other):1377824   6:331336
##    timestamp
##  Min.   :1.488e+09
##  1st Qu.:1.490e+09
##  Median :1.496e+09
##  Mean   :1.494e+09
##  3rd Qu.:1.498e+09
##  Max.   :1.512e+09
##
```

```
# as we can see by removing these noisy examples, we have removed suspicious values in features with le
```

Number of outliers discovered in this first analysis are below 1% of data, so it's safely to filter them out. By addressing outlier values in most intuitive features, we have been able to remove at the same time not-as-clear outliers in other variables, such as: mta_tax, or extra.

Finally, categorical features should be one-hot-encoded to avoid our model finds out false relationships among numerical values; the only ones accepted by XGBoost. Here we found a problem with the number of binary

variables generated as there are more than 2 hundreds different locations. In order to deal with memory performance nuances we have relied on a schema based on sparse features to overcome these problems.

```
# one-hot-encoding categorical features
library(Matrix)
encode_data <- function(dataset) {
  sparse.model.matrix(timestamp + passenger_count + trip_distance + fare_amount + extra + mta_tax +  to

}

raw_train <- raw_train %>% arrange(timestamp)
split.position <- nrow(raw_train) * 0.8

raw_train <- encode_data(raw_train)
backup.train <- raw_train

y_train <- raw_train[1:split.position,"tip_amount"]
y_test <- raw_train[split.position:nrow(raw_train),"tip_amount"]
raw_test <- raw_train[split.position:nrow(raw_train),-which(raw_train@Dimnames[[2]] %in% c("tip_amount"
raw_train <- raw_train[1:split.position,-which(raw_train@Dimnames[[2]] %in% c("tip_amount"))]



saveRDS(raw_train, file="/home/sramirez/taxi_ohe_train_X.Rda")
saveRDS(y_train, file="/home/sramirez/taxi_ohe_train_y.Rda")
saveRDS(raw_test, file="/home/sramirez/test_ohe_X.Rda")
saveRDS(y_test, file="/home/sramirez/test_ohe_y.Rda")

nrow(raw_train)
```

```
## [1] 1674066
```

```
nrow(raw_test)
```

```
## [1] 418517
```

```
length(y_train)
```

```
## [1] 1674066
```

```
length(y_test)
```

```
## [1] 418517
```

```
#summary(raw_train)
```

Learning phase

Now, we learn from data from the three months specified in the requirements. Our first idea was to use a validation set to select the best configuration for the most important parameter in XGBoost: the number of rounds. However, because of the huge amount of data and the rapidness of this analysis we would rather focus on a simple 80/20 hold-out validation process. Before performing split, we sort the 3-months dataset by timestamp. By doing so, we guarantee the validation process will be less biased as we are including time information in some input features (hour, timestamp, etc.).

A better validation process could be performed by relying on a big data platform such as Spark. Nevertheless, this process demands longer time than a rapid validation with a reduce subset of the original dataset.

About the classification algorithm chosen, we have relied on XGBoost because of its competitive time performance, and its great predictive capabilities. Authors of the algorithm proved that it was possible to train model with millions of data in a single machine. So we think it is the perfect fit for our purposes. Parameter values were set by default as specified in the documentation. We have only tweaked subsample and colsample_bytree to put more emphasis on avoiding overfitting.

```
## [1]  train-rmse:2.378909 valid-rmse:2.382842
## [2]  train-rmse:2.024261 valid-rmse:1.999122
## [3]  train-rmse:1.815969 valid-rmse:1.773064
## [4]  train-rmse:1.702772 valid-rmse:1.650840
## [5]  train-rmse:1.633465 valid-rmse:1.575248
## [6]  train-rmse:1.589816 valid-rmse:1.527996
## [7]  train-rmse:1.566935 valid-rmse:1.504103
## [8]  train-rmse:1.551919 valid-rmse:1.488768
## [9]  train-rmse:1.543459 valid-rmse:1.480191
## [10] train-rmse:1.534358 valid-rmse:1.477322
## [11] train-rmse:1.528798 valid-rmse:1.474108
## [12] train-rmse:1.526060 valid-rmse:1.471550
## [13] train-rmse:1.520038 valid-rmse:1.469827
## [14] train-rmse:1.517139 valid-rmse:1.468013
## [15] train-rmse:1.512511 valid-rmse:1.467954
## [16] train-rmse:1.510755 valid-rmse:1.466882
## [17] train-rmse:1.509346 valid-rmse:1.466015
## [18] train-rmse:1.508178 valid-rmse:1.465213
## [19] train-rmse:1.503876 valid-rmse:1.465859
## [20] train-rmse:1.502519 valid-rmse:1.465523
## [21] train-rmse:1.500707 valid-rmse:1.464828
## [22] train-rmse:1.499388 valid-rmse:1.464135
## [23] train-rmse:1.496674 valid-rmse:1.463634
## [24] train-rmse:1.495968 valid-rmse:1.463333
## [25] train-rmse:1.492649 valid-rmse:1.463707
## [26] train-rmse:1.487866 valid-rmse:1.466566
## [27] train-rmse:1.487150 valid-rmse:1.466334
## [28] train-rmse:1.486386 valid-rmse:1.465869
## [29] train-rmse:1.485369 valid-rmse:1.465717
## [30] train-rmse:1.484109 valid-rmse:1.465121
## [31] train-rmse:1.483559 valid-rmse:1.464708
## [32] train-rmse:1.481853 valid-rmse:1.464507
## [33] train-rmse:1.481460 valid-rmse:1.464182
## [34] train-rmse:1.480598 valid-rmse:1.464132
## [35] train-rmse:1.480135 valid-rmse:1.463717
## [36] train-rmse:1.479686 valid-rmse:1.463100
## [37] train-rmse:1.475839 valid-rmse:1.462459
## [38] train-rmse:1.475526 valid-rmse:1.462245
## [39] train-rmse:1.474772 valid-rmse:1.461727
## [40] train-rmse:1.474210 valid-rmse:1.461398
## [41] train-rmse:1.473590 valid-rmse:1.461117
## [42] train-rmse:1.470228 valid-rmse:1.462092
## [43] train-rmse:1.469793 valid-rmse:1.461168
## [44] train-rmse:1.468234 valid-rmse:1.461032
## [45] train-rmse:1.467522 valid-rmse:1.461001
## [46] train-rmse:1.464826 valid-rmse:1.460772
## [47] train-rmse:1.464506 valid-rmse:1.460659
## [48] train-rmse:1.463473 valid-rmse:1.460527
```

```
## [49] train-rmse:1.462428 valid-rmse:1.460479
## [50] train-rmse:1.462117 valid-rmse:1.460390
## [51] train-rmse:1.461888 valid-rmse:1.460445
## [52] train-rmse:1.461509 valid-rmse:1.460290
## [53] train-rmse:1.459865 valid-rmse:1.459854
## [54] train-rmse:1.459376 valid-rmse:1.459727
## [55] train-rmse:1.459047 valid-rmse:1.459576
## [56] train-rmse:1.458260 valid-rmse:1.459504
## [57] train-rmse:1.457548 valid-rmse:1.459295
## [58] train-rmse:1.456392 valid-rmse:1.459094
## [59] train-rmse:1.455737 valid-rmse:1.459129
## [60] train-rmse:1.455138 valid-rmse:1.458984
## [61] train-rmse:1.454983 valid-rmse:1.458986
## [62] train-rmse:1.454221 valid-rmse:1.458883
## [63] train-rmse:1.452093 valid-rmse:1.462506
## [64] train-rmse:1.451776 valid-rmse:1.462395
## [65] train-rmse:1.450840 valid-rmse:1.462021
## [66] train-rmse:1.450538 valid-rmse:1.461927
## [67] train-rmse:1.449910 valid-rmse:1.461865
## [68] train-rmse:1.449563 valid-rmse:1.462133
## [69] train-rmse:1.449304 valid-rmse:1.461966
## [70] train-rmse:1.449043 valid-rmse:1.461682
## [71] train-rmse:1.447198 valid-rmse:1.461355
## [72] train-rmse:1.446660 valid-rmse:1.461305
## [73] train-rmse:1.446476 valid-rmse:1.461194
## [74] train-rmse:1.446088 valid-rmse:1.461134
## [75] train-rmse:1.445669 valid-rmse:1.460953
## [76] train-rmse:1.445447 valid-rmse:1.460985
## [77] train-rmse:1.444881 valid-rmse:1.460939
## [78] train-rmse:1.443044 valid-rmse:1.460889
## [79] train-rmse:1.442902 valid-rmse:1.460778
## [80] train-rmse:1.442330 valid-rmse:1.460493
## [81] train-rmse:1.438261 valid-rmse:1.460191
## [82] train-rmse:1.436058 valid-rmse:1.459685
## [83] train-rmse:1.434530 valid-rmse:1.459983
## [84] train-rmse:1.434389 valid-rmse:1.459904
## [85] train-rmse:1.433369 valid-rmse:1.459723
## [86] train-rmse:1.431797 valid-rmse:1.460068
## [87] train-rmse:1.431576 valid-rmse:1.459980
## [88] train-rmse:1.430154 valid-rmse:1.460693
## [89] train-rmse:1.429487 valid-rmse:1.460679
## [90] train-rmse:1.428259 valid-rmse:1.460637
## [91] train-rmse:1.428015 valid-rmse:1.460641
## [92] train-rmse:1.426838 valid-rmse:1.460893
## [93] train-rmse:1.426703 valid-rmse:1.460864
## [94] train-rmse:1.426400 valid-rmse:1.460815
## [95] train-rmse:1.425985 valid-rmse:1.460971
## [96] train-rmse:1.424136 valid-rmse:1.460600
## [97] train-rmse:1.423548 valid-rmse:1.460378
## [98] train-rmse:1.423032 valid-rmse:1.460424
## [99] train-rmse:1.422834 valid-rmse:1.460399
## [100]    train-rmse:1.422722 valid-rmse:1.460309

## [1] TRUE
```

Validate predictions

```
# Reload model and test
xgb.model <- xgb.load("/home/sramirez/git/taxi_NYC_exploration/last_xgbmodel_taxiNYC")
X_test <- readRDS(file="/home/sramirez/test_ohe_X.Rda")
y_test <- readRDS(file="/home/sramirez/test_ohe_y.Rda")

# predict values in test set
y_test_pred <- predict(xgb.model, X_test)
err_test <- RMSE(y_test_pred, y_test)
residual.vector <- abs(y_test_pred - y_test)
naive.residual.vector <- abs(mean(y_train) - y_test)
paste("RMSE for test =", err_test)
```

```
## [1] "RMSE for test = 1.46030885461823"
```

```
paste("Absolute mean error for test =", mean(residual.vector))
```

```
## [1] "Absolute mean error for test = 0.569231341708166"
```

```
paste("Standard deviation error for test =", sd(residual.vector))
```

```
## [1] "Standard deviation error for test = 1.34479810814143"
```

```
paste("RMSE for naive test =", RMSE(mean(y_train), y_test))
```

```
## [1] "RMSE for naive test = 2.66066110665573"
```

```
paste("Absolute mean error for naive test =", mean(naive.residual.vector))
```

```
## [1] "Absolute mean error for naive test = 1.64415158863437"
```

```
paste("Standard deviation error for naive test =", sd(naive.residual.vector))
```

```
## [1] "Standard deviation error for naive test = 2.09186365085641"
```

```
summary(y_test)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   1.430   1.911   2.490 250.000
```

According to the results, our first model is able to make predictions with a mean error around 50 cents. This allow us to assert that we are able to make better predictions than the naïve model (that based on mean value in train data), which is a great start! Although the model can be improved with feature engineering, for instance, by adding aggregations by locations, number of passengers, etc. Furthermore, by using extra months from previous years we can include more features focused on stationality, such as dayOfYear, month, etc.

Finally, we will check what features are considered more relevant for our model:

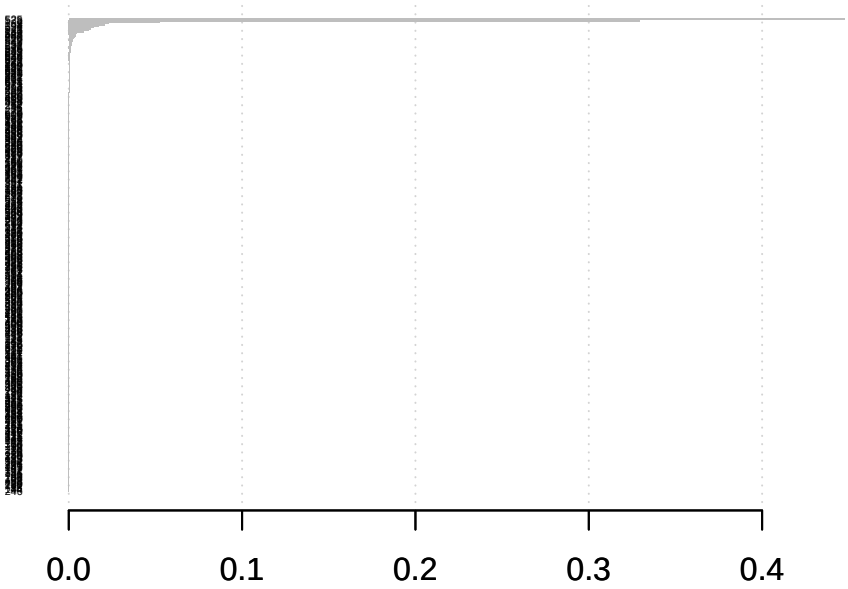
```
importance_matrix <- xgb.importance(model = xgb.model)

print(importance_matrix)
```

```
##      Feature         Gain        Cover    Frequency
##   1:     525 4.474631e-01 1.253202e-01 0.2219895288
##   2:     522 3.294548e-01 2.074804e-02 0.0759162304
##   3:       2 5.257740e-02 1.484895e-02 0.0214659686
##   4:     139 2.315669e-02 1.525431e-02 0.0293193717
##   5:       3 2.083769e-02 1.693877e-02 0.0094240838
##  ---
```

```
## 341:     197 8.107677e-10 4.067932e-08 0.0002617801
## 342:      18 2.447945e-10 1.144106e-08 0.0002617801
## 343:      85 4.081603e-11 3.813686e-09 0.0002617801
## 344:     144 1.307072e-11 1.779720e-08 0.0002617801
## 345:     246 3.089777e-12 3.813686e-09 0.0002617801
```

```
xgb.plot.importance(importance_matrix = importance_matrix)
```




```
raw_train@Dimnames[[2]][as.numeric(importance_matrix$Feature[0:10])]
```

```
##  [1] "payment_type4"   "DOLocationID265" "VendorID2"
##  [4] "PULocationID137" "RatecodeID2"     "DOLocationID137"
##  [7] "DOLocationID264" "payment_type2"   "RatecodeID4"
## [10] "PULocationID131"
```