cmp713-project

Hazem Alabiad & Betül Bayrak

25/01/2021

Introduction

Tipping someone is a common thing in our community, especially if this person is a taxi driver, waiter, or delivery officer; The tip amount varies depending on many conditions; In the case of a taxi driver, it might depend on the passenger count, trip distance, trip hour, et cetera. In this project, we aim to study and predict the tip percentage of the total amount paid to taxi drivers. We used the data provided by NYC Taxi & Limousine Commission (TLC) from this link. First, we explore our data to know what our data looks like, find potential outliers, obtain general and basic statistics, for instance, observations count, variables count, NAs' count, et cetera. Second, we apply what is called data pre-processing and data engineering that include removing unnecessary or NAs' rows, columns, convert the data type to a more suitable one and introducing new variables to get our data ready to be fed our model that is going to predict the tip amount given other variables. Third, we build our models that utilize the pre-processed, clean data to predict a variable "tip percentage in our case". Lastly, we measure the performance or accuracy of our model using many approaches and mathematical formulas and plot the predictions vs. actual tip percentage to decide on which model did the best.

Loading Libraries

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
library(ggmap)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```r
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
```

```
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(nnet)
library(hrbrthemes)
```

```
## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use these themes.
```

```
##       Please use hrbrthemes::import_roboto_condensed() to install Roboto Condensed and
```

```
##       if Arial Narrow is not on your system, please see https://bit.ly/arialnarrow
```

```r
library(rpart.plot)
```

```
## Loading required package: rpart
```

```r
library(mlbench)
library(e1071)
library(rpart)
library(DMwR2)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```r
library(caret)
```

```
## Loading required package: lattice
```

```r
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
library(fpc)
library(Hmisc)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: Formula
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following object is masked from 'package:e1071':
##
```

```
##     impute

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units

library(tibble)
library(knitr)
library(MLmetrics)

##
## Attaching package: 'MLmetrics'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE

## The following object is masked from 'package:base':
##
##     Recall

library(earth) # fit MARS models

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos

##
## Attaching package: 'TeachingDemos'

## The following objects are masked from 'package:Hmisc':
##
##     cnvrt.coords, subplot
```

Custom Functions

```
`%notin%` <- Negate(`%in%`)

pretty_df_print <- function (df) kable_styling(kable(df))

top_least_freq_and_neg_vals <- function (col) {
  t <- table(col) %>% as.data.frame() %>% arrange(desc(Freq))
  non_pos <- as.data.frame(col)
  non_pos <- non_pos[non_pos <=0, ]
  print("Top frequent values:")
  print.data.frame(head(t))
  print("Least frequent values:")
  print.data.frame(tail(t))
```

```r
  print("Negative values:")
  print.data.frame(non_pos)
  print("/////////////////////////////////////////////////////")
}

na_count_df <- function (df) {
  return(as.data.frame(t(data.frame(sapply(df, function(col) sum(is.na(col)))))))
}

iqr_outliers_min_max <- function(df) {
  iqr_df <- as.data.frame( t(data.frame(df %>% sapply(function (x) {quantile(x, c(0.25,
      0.75))}))) ) %>% cbind((data.frame(df %>% sapply(IQR))))
  colnames(iqr_df) <- c("Q1", "Q3", "IQR")
  iqr_df$minVal <- iqr_df$Q1 - 1.5*iqr_df$IQR
  iqr_df$maxVal <- iqr_df$Q3 + 1.5*iqr_df$IQR
  print.data.frame(iqr_df %>% select("minVal", "maxVal"))
}

iqr_outliers_finder <- function(x){
  Q1 <- quantile(x, 0.25)
  Q3 <- quantile(x, 0.75)
  IQR <- Q3 - Q1
  Vl <- Q1 - 1.5 * IQR
  Vr <- Q3 + 1.5 * IQR
  return (x[which(x < Vl | x > Vr)])
}

get_freq_df_from_vector <- function (vec, col.name) {
  freq_df <- as.data.frame(table(vec))
  colnames(freq_df)[1] <- col.name
  freq_df <- freq_df[order(freq_df[,1], decreasing = T),]
  if(nrow(freq_df) <= 10) {
    print.data.frame(freq_df)}
  else {
    cat("Highest values :\n")
    print.data.frame(head(freq_df))
    cat("Lowest values :\n")
    print.data.frame(tail(freq_df))
  }
}

dbscan_outliers_finder <- function(data, ...) {
  require(fpc, quietly=TRUE)
  cl <- dbscan(data, ...)
  posOuts <- which(cl$cluster == 0)
  list(positions = posOuts,
       outliers = data[posOuts,],
       dbscanResults = cl)
}
```

Data loading

```r
st <- proc.time()
green <- read.csv("data/green_tripdata_2020-06.csv")
green <- green %>% rbind(read.csv("data/green_tripdata_2020-01.csv"))
print(proc.time() - st)
```

```
##    user  system elapsed
##   5.824   0.157   6.005
```

```r
cat('There are [', nrow(green), '] observations, and [',ncol(green),'] variables')
```

```
## There are [ 510879 ] observations, and [ 20 ] variables
```

Data exploratory & Preprocessing

Dataframe head

```r
head(green)
```

```
##   VendorID lpep_pickup_datetime lpep_dropoff_datetime store_and_fwd_flag
## 1        1  2020-06-01 00:22:07   2020-06-01 00:39:03                  N
## 2        2  2020-06-01 00:09:05   2020-06-01 00:22:46                  N
## 3        2  2020-06-01 00:20:05   2020-06-01 00:23:33                  N
## 4        2  2020-06-01 00:30:50   2020-06-01 00:37:49                  N
## 5        2  2020-06-01 00:03:05   2020-06-01 00:16:46                  N
## 6        2  2020-06-01 00:14:05   2020-06-01 00:23:57                  N
##   RatecodeID PULocationID DOLocationID passenger_count trip_distance
## 1          1          255           14               1          0.00
## 2          1          166          141               1          3.43
## 3          1           75           42               1          1.03
## 4          1           74           42               1          1.22
## 5          1          152          247               1          2.59
## 6          1          244          200               1          5.95
##   fare_amount extra mta_tax tip_amount tolls_amount ehail_fee
## 1        28.2   0.0     0.5       0.00          0.0        NA
## 2        13.0   0.5     0.5       3.41          0.0        NA
## 3         5.0   0.5     0.5       0.00          0.0        NA
## 4         7.0   0.5     0.5       0.00          0.0        NA
## 5        11.5   0.5     0.5       0.00          0.0        NA
## 6        17.5   0.5     0.5       6.00          2.8        NA
##   improvement_surcharge total_amount payment_type trip_type
## 1                   0.3        29.00            1         1
## 2                   0.3        20.46            1         1
## 3                   0.3         6.30            2         1
## 4                   0.3         8.30            2         1
## 5                   0.3        12.80            2         1
## 6                   0.3        27.60            1         1
##   congestion_surcharge
## 1                 0.00
## 2                 2.75
## 3                 0.00
## 4                 0.00
## 5                 0.00
## 6                 0.00
```

Dataframe tail

```r
tail(green)
```

```
##        VendorID lpep_pickup_datetime lpep_dropoff_datetime store_and_fwd_flag
## 510874       NA  2020-01-31 23:08:00   2020-01-31 23:26:00
## 510875       NA  2020-01-31 23:29:00   2020-01-31 23:47:00
## 510876       NA  2020-01-31 23:57:00   2020-02-01 00:23:00
## 510877       NA  2020-01-31 23:57:00   2020-02-01 00:10:00
## 510878       NA  2020-01-31 23:27:00   2020-02-01 00:04:00
## 510879       NA  2020-01-31 23:36:00   2020-02-01 00:01:00
##        RatecodeID PULocationID DOLocationID passenger_count trip_distance
## 510874         NA           14          206              NA          7.51
```

```
## 510875         NA          167           32               NA          4.58
## 510876         NA           81           69               NA          6.55
## 510877         NA          244          241               NA          3.34
## 510878         NA           68           17               NA          8.92
## 510879         NA           22          124               NA         13.51
##        fare_amount extra mta_tax tip_amount tolls_amount ehail_fee
## 510874       21.46  2.75       0          0        14.99        NA
## 510875       23.21  2.75       0          0         0.00        NA
## 510876       27.27  2.75       0          0         0.00        NA
## 510877       25.95  2.75       0          0         0.00        NA
## 510878       30.39  2.75       0          0         0.00        NA
## 510879       42.20  2.75       0          0         0.00        NA
##        improvement_surcharge total_amount payment_type trip_type
## 510874                   0.3        39.50           NA        NA
## 510875                   0.3        26.26           NA        NA
## 510876                   0.3        30.32           NA        NA
## 510877                   0.3        29.00           NA        NA
## 510878                   0.3        33.44           NA        NA
## 510879                   0.3        45.25           NA        NA
##        congestion_surcharge
## 510874                   NA
## 510875                   NA
## 510876                   NA
## 510877                   NA
## 510878                   NA
## 510879                   NA
```

Brief info on the data

```
str(green)
```

```
## 'data.frame':    510879 obs. of  20 variables:
##  $ VendorID             : int  1 2 2 2 2 2 2 2 2 2 ...
##  $ lpep_pickup_datetime : chr  "2020-06-01 00:22:07" "2020-06-01 00:09:05" "2020-06-01
00:20:05" "2020-06-01 00:30:50" ...
##  $ lpep_dropoff_datetime: chr  "2020-06-01 00:39:03" "2020-06-01 00:22:46" "2020-06-01
00:23:33" "2020-06-01 00:37:49" ...
##  $ store_and_fwd_flag   : chr  "N" "N" "N" "N" ...
##  $ RatecodeID           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ PULocationID         : int  255 166 75 74 152 244 93 85 82 174 ...
##  $ DOLocationID         : int  14 141 42 42 247 200 95 188 70 127 ...
##  $ passenger_count      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ trip_distance        : num  0 3.43 1.03 1.22 2.59 5.95 2.25 0.65 3.5 3.38 ...
##  $ fare_amount          : num  28.2 13 5 7 11.5 17.5 11 5 12 11.5 ...
##  $ extra                : num  0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ mta_tax              : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ tip_amount           : num  0 3.41 0 0 0 6 0 0 2.66 0 ...
##  $ tolls_amount         : num  0 0 0 0 0 2.8 0 0 0 0 ...
##  $ ehail_fee            : logi  NA NA NA NA NA NA ...
##  $ improvement_surcharge: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
##  $ total_amount         : num  29 20.5 6.3 8.3 12.8 ...
##  $ payment_type         : int  1 1 2 2 2 1 2 2 1 2 ...
##  $ trip_type            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ congestion_surcharge : num  0 2.75 0 0 0 0 0 0 0 0 ...
```

Unnecessary Variables Deletion

```
num.vars <- ncol(green)
green <- green %>% subset(select = -c( VendorID, store_and_fwd_flag))
cat("[", num.vars - ncol(green) , "] variables were delete")
```

```
## [ 2 ] variables were delete
```

Summary of the data

```r
describe(green)
```

```
## green
##
##  18  Variables      510879  Observations
## --------------------------------------------------------------------------------
## lpep_pickup_datetime
##        n  missing distinct
##   510879        0   385504
##
## lowest : 2008-12-31 22:06:48 2008-12-31 23:11:00 2009-01-01 00:07:51 2009-01-01 00:08:09
2009-01-01 00:34:50
## highest: 2020-06-30 23:42:13 2020-06-30 23:51:12 2020-06-30 23:52:09 2020-06-30 23:57:39
2020-06-30 23:59:37
## --------------------------------------------------------------------------------
## lpep_dropoff_datetime
##        n  missing distinct
##   510879        0   385627
##
## lowest : 2008-12-31 23:12:08 2009-01-01 00:17:31 2009-01-01 00:45:10 2009-01-01 00:58:27
2009-01-01 01:00:23
## highest: 2020-07-01 10:24:18 2020-07-01 12:00:04 2020-07-01 13:27:36 2020-07-01 14:51:21
2020-07-01 15:59:47
## --------------------------------------------------------------------------------
## RatecodeID
##        n  missing distinct     Info     Mean      Gmd
##   370150   140729        7    0.083    1.107   0.2087
##
## lowest :  1  2  3  4  5, highest:  3  4  5  6 99
##
## Value           1      2      3      4      5      6     99
## Frequency  359569    727    189    269   9390      4      2
## Proportion  0.971  0.002  0.001  0.001  0.025  0.000  0.000
## --------------------------------------------------------------------------------
## PULocationID
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0      256    0.999    108.1    78.84       18       33
##      .25      .50      .75      .90      .95
##       52       82      166      226      244
##
## lowest :   1   3   4   5   6, highest: 261 262 263 264 265
## --------------------------------------------------------------------------------
## DOLocationID
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0      260        1    128.6    87.77       21       37
##      .25      .50      .75      .90      .95
##       62      129      193      238      250
##
## lowest :   1   2   3   4   5, highest: 261 262 263 264 265
## --------------------------------------------------------------------------------
## passenger_count
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   370150   140729       10    0.353    1.298   0.5511        1        1
##      .25      .50      .75      .90      .95
##        1        1        1        2        3
##
```

```
## lowest : 0 1 2 3 4, highest: 5 6 7 8 9
##
## Value           0      1      2      3      4      5      6      7      8
## Frequency     693 320143  26611   5312   1716   9976   5666     10     18
## Proportion  0.002  0.865  0.072  0.014  0.005  0.027  0.015  0.000  0.000
##
## Value           9
## Frequency       5
## Proportion  0.000
## ---------------------------------------------------------------------------
## trip_distance
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0     3776        1    8.818    14.21     0.23     0.60
##      .25      .50      .75      .90      .95
##     1.10     2.12     4.61     9.27    13.48
##
## lowest :    -33.69    -33.29    -29.17    -26.66    -26.08
## highest: 134349.12 134355.40 134359.97 148387.95 162291.50
##
## Value           0   6000  26000  34000  42000  44000  48000  50000  84000
## Frequency  510851      1      1      1      1      4      1      3      1
## Proportion      1      0      0      0      0      0      0      0      0
##
## Value       96000 132000 134000 148000 162000
## Frequency       1      1     11      1      1
## Proportion      0      0      0      0      0
##
## For the frequency table, variable is rounded to the nearest 2000
## ---------------------------------------------------------------------------
## fare_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0     6501        1     16.3    13.08     4.50     5.00
##      .25      .50      .75      .90      .95
##     7.00    12.00    21.37    33.40    44.15
##
## lowest : -210.00 -200.00 -180.00  -97.67  -83.82
## highest:  275.00  300.00  335.50  630.00  753.00
## ---------------------------------------------------------------------------
## extra
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0       16    0.864   0.7613    1.052     0.00     0.00
##      .25      .50      .75      .90      .95
##     0.00     0.50     1.00     2.75     2.75
##
## lowest : -4.50 -1.00 -0.50  0.00  0.50, highest:  3.50  3.75  4.50  5.50  8.25
##
## Value       -4.50  -1.00  -0.50   0.00   0.50   0.80   1.00   2.50   2.75
## Frequency       4    234    333 253938  88855      1  75201     28  83708
## Proportion  0.000  0.000  0.001  0.497  0.174  0.000  0.147  0.000  0.164
##
## Value        3.00   3.25   3.50   3.75   4.50   5.50   8.25
## Frequency       4   2083      4   2067    129   4019    271
## Proportion  0.000  0.004  0.000  0.004  0.000  0.008  0.001
## ---------------------------------------------------------------------------
## mta_tax
##        n  missing distinct     Info     Mean      Gmd
##   510879        0        4    0.553   0.3771   0.1866
##
## Value       -0.50   0.00   0.50   3.55
## Frequency    1114 123396 386364      5
## Proportion  0.002  0.242  0.756  0.000
## ---------------------------------------------------------------------------
## tip_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0     1549    0.737   0.9838    1.493     0.00     0.00
```

```
##      .25      .50      .75      .90      .95
##     0.00     0.00     1.76     2.94     4.01
##
## lowest :  -2.80  -1.29  -0.99  -0.86  -0.76, highest: 300.08 333.00 449.60 450.00 480.00
## --------------------------------------------------------------------------------
## tolls_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0      106     0.17   0.3747   0.7103     0.00     0.00
##      .25      .50      .75      .90      .95
##     0.00     0.00     0.00     0.00     6.12
##
## lowest : -6.12 -2.80  0.00  0.80  1.00, highest: 35.00 39.74 45.75 48.88 96.12
## --------------------------------------------------------------------------------
## improvement_surcharge
##        n  missing distinct     Info     Mean      Gmd
##   510879        0        3    0.106   0.2883  0.02258
##
## Value        -0.3    0.0    0.3
## Frequency    1172  17570 492137
## Proportion  0.002  0.034  0.963
## --------------------------------------------------------------------------------
## total_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   510879        0     7599        1     19.4     14.6     5.80     6.80
##      .25      .50      .75      .90      .95
##     9.30    14.80    25.06    38.54    49.49
##
## lowest : -210.30 -200.30 -180.30  -88.50  -80.30
## highest:  454.90  462.80  498.80  636.92  753.80
## --------------------------------------------------------------------------------
## payment_type
##        n  missing distinct     Info     Mean      Gmd
##   370150   140729        5    0.746    1.456   0.5126
##
## lowest : 1 2 3 4 5, highest: 1 2 3 4 5
##
## Value           1      2      3      4      5
## Frequency  204709 162727   2043    653     18
## Proportion  0.553  0.440  0.006  0.002  0.000
## --------------------------------------------------------------------------------
## trip_type
##        n  missing distinct     Info     Mean      Gmd
##   370149   140730        2    0.071    1.024  0.04715
##
## Value           1      2
## Frequency  361206   8943
## Proportion  0.976  0.024
## --------------------------------------------------------------------------------
## congestion_surcharge
##        n  missing distinct     Info     Mean      Gmd
##   370150   140729        5    0.429   0.4751   0.7861
##
## lowest : -2.75  0.00  0.75  2.50  2.75, highest: -2.75  0.00  0.75  2.50  2.75
##
## Value       -2.75   0.00   0.75   2.50   2.75
## Frequency       5 306179      6    163  63797
## Proportion  0.000  0.827  0.000  0.000  0.172
## --------------------------------------------------------------------------------
##
## Variables with all observations missing:
##
## [1] ehail_fee
```

Count NAs' in each column

```r
na_count_df(green)
```

```
##                                          lpep_pickup_datetime
## sapply.df..function.col..sum.is.na.col...                    0
##                                          lpep_dropoff_datetime RatecodeID
## sapply.df..function.col..sum.is.na.col...                     0     140729
##                                          PULocationID DOLocationID
## sapply.df..function.col..sum.is.na.col...            0            0
##                                          passenger_count trip_distance
## sapply.df..function.col..sum.is.na.col...          140729             0
##                                          fare_amount extra mta_tax tip_amount
## sapply.df..function.col..sum.is.na.col...           0     0       0          0
##                                          tolls_amount ehail_fee
## sapply.df..function.col..sum.is.na.col...            0    510879
##                                          improvement_surcharge total_amount
## sapply.df..function.col..sum.is.na.col...                     0            0
##                                          payment_type trip_type
## sapply.df..function.col..sum.is.na.col...       140729    140730
##                                          congestion_surcharge
## sapply.df..function.col..sum.is.na.col...               140729
```

Remove NAs’

As we can see from the output above: `passenger_count`, `payment_type`, `trip_type`, `congestion_surcharge` have ~ (24 K) NAs’ `fare_amount`, `extra`, `mta_tax`, `tip_amount`, `tolls_amount`, `improvement_surcharge`, `total_amount` have negative values. Negative values: we believe have been entered mistakenly as negative values so, we need to fix this problem by obtaining the absolute values. NAs’: We need to drop rows with NAs as they critical in the classification. As well as, we need to remove `ehail_fee` variable as it’s all NAs’. For `tip_amount` we realized that there are many `0` values, we need to get rid of them.

```r
green <- green %>% within(rm(ehail_fee))
num.rows <- nrow(green)
green <- green[!is.na(green$trip_type) & green$trip_distance != 0,]
cat("[", num.rows - nrow(green), "] observations were deleted out of [", num.rows, "]",
    "which means ~ (", round((num.rows - nrow(green))/num.rows, digits = 2), ")")
```

```
## [ 158626 ] observations were deleted out of [ 510879 ] which means ~ ( 0.31 )
```

```r
as.data.frame(t(data.frame(sapply(green, function(col) sum(is.na(col))))))
```

```
##                                             lpep_pickup_datetime
## sapply.green..function.col..sum.is.na.col...                    0
##                                             lpep_dropoff_datetime RatecodeID
## sapply.green..function.col..sum.is.na.col...                     0          0
##                                             PULocationID DOLocationID
## sapply.green..function.col..sum.is.na.col...            0            0
##                                             passenger_count trip_distance
## sapply.green..function.col..sum.is.na.col...               0             0
##                                             fare_amount extra mta_tax
## sapply.green..function.col..sum.is.na.col...           0     0       0
##                                             tip_amount tolls_amount
## sapply.green..function.col..sum.is.na.col...          0            0
##                                             improvement_surcharge total_amount
## sapply.green..function.col..sum.is.na.col...                     0            0
##                                             payment_type trip_type
## sapply.green..function.col..sum.is.na.col...            0         0
##                                             congestion_surcharge
## sapply.green..function.col..sum.is.na.col...                    0
```

Check top & least frequent and negative values

```r
describe(green)
```

```
## green
##
##  17  Variables      352253  Observations
## --------------------------------------------------------------------------------
## lpep_pickup_datetime
##        n  missing distinct
##   352253        0   327696
##
## lowest : 2009-01-01 00:08:09 2009-01-01 00:34:50 2009-01-01 00:45:53 2009-01-01 00:54:13
2009-01-01 01:19:36
## highest: 2020-06-30 23:37:20 2020-06-30 23:42:13 2020-06-30 23:51:12 2020-06-30 23:52:09
2020-06-30 23:57:39
## --------------------------------------------------------------------------------
## lpep_dropoff_datetime
##        n  missing distinct
##   352253        0   327343
##
## lowest : 2009-01-01 00:17:31 2009-01-01 00:45:10 2009-01-01 00:58:27 2009-01-01 01:00:23
2009-01-01 01:28:23
## highest: 2020-07-01 10:24:18 2020-07-01 12:00:04 2020-07-01 13:27:36 2020-07-01 14:51:21
2020-07-01 15:59:47
## --------------------------------------------------------------------------------
## RatecodeID
##        n  missing distinct     Info     Mean      Gmd
##   352253        0        6    0.067    1.085   0.1664
##
## lowest : 1 2 3 4 5, highest: 2 3 4 5 6
##
## Value           1      2      3      4      5      6
## Frequency  344198    577    143    266   7066      3
## Proportion  0.977  0.002  0.000  0.001  0.020  0.000
## --------------------------------------------------------------------------------
## PULocationID
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0      237    0.997    101.7    73.53       20       33
##      .25      .50      .75      .90      .95
##       49       75      134      223      244
##
## lowest :   3   4   5   6   7, highest: 260 262 263 264 265
## --------------------------------------------------------------------------------
## DOLocationID
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0      258        1    129.5    87.92       24       41
##      .25      .50      .75      .90      .95
##       65      129      193      239      255
##
## lowest :   1   3   4   5   6, highest: 261 262 263 264 265
## --------------------------------------------------------------------------------
## passenger_count
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0       10    0.362    1.308   0.5671        1        1
##      .25      .50      .75      .90      .95
##        1        1        1        2        3
##
## lowest : 0 1 2 3 4, highest: 5 6 7 8 9
##
## Value           0      1      2      3      4      5      6      7      8
```

```
## Frequency     610 303129  26191   5248   1691   9740   5630      6      6
## Proportion  0.002  0.861  0.074  0.015  0.005  0.028  0.016  0.000  0.000
##
## Value          9
## Frequency      2
## Proportion  0.000
## ---------------------------------------------------------------------------
## trip_distance
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0     3006        1    3.182    3.499     0.50     0.67
##      .25      .50      .75      .90      .95
##     1.02     1.73     3.20     6.09     8.46
##
## lowest :     -3.05     -1.46     -1.33     -1.02     -1.01
## highest:     80.15     84.97    124.13    130.68 134121.50
##
## Value           0 134000
## Frequency  352252      1
## Proportion      1      0
##
## For the frequency table, variable is rounded to the nearest 1000
## ---------------------------------------------------------------------------
## fare_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0      749    0.999    11.92    8.692      4.0      5.0
##      .25      .50      .75      .90      .95
##      6.5      9.0     14.0     22.0     29.0
##
## lowest : -200.00  -80.00  -52.00  -50.00  -46.96
## highest:  228.50  233.50  335.50  630.00  753.00
## ---------------------------------------------------------------------------
## extra
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0       10    0.842   0.4201   0.5453      0.0      0.0
##      .25      .50      .75      .90      .95
##      0.0      0.0      0.5      1.0      1.0
##
## lowest : -4.50 -1.00 -0.50  0.00  0.50, highest:  1.00  2.75  3.25  3.75  4.50
##
## Value       -4.50  -1.00  -0.50   0.00   0.50   1.00   2.75   3.25   3.75
## Frequency       3    184    287 179563  88132  74436   5401   2077   2062
## Proportion  0.000  0.001  0.001  0.510  0.250  0.211  0.015  0.006  0.006
##
## Value        4.50
## Frequency     108
## Proportion  0.000
## ---------------------------------------------------------------------------
## mta_tax
##        n  missing distinct     Info     Mean      Gmd
##   352253        0        4    0.067   0.4873  0.02499
##
## Value       -0.50   0.00   0.50   3.55
## Frequency     892   7174 344182      5
## Proportion  0.003  0.020  0.977  0.000
## ---------------------------------------------------------------------------
## tip_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0     1493    0.843    1.252    1.749     0.00     0.00
##      .25      .50      .75      .90      .95
##     0.00     0.00     2.00     3.41     4.76
##
## lowest :  -1.29   0.00   0.01   0.02   0.03, highest: 300.08 333.00 449.60 450.00 480.00
## ---------------------------------------------------------------------------
## tolls_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
```

```
##   352253        0       70    0.071   0.1536   0.3006        0        0
##      .25      .50      .75      .90      .95
##        0        0        0        0        0
##
## lowest : -6.12  0.00  1.00  2.00  2.29, highest: 35.00 39.74 45.75 48.88 96.12
## ---------------------------------------------------------------------------------
## improvement_surcharge
##        n  missing distinct     Info     Mean      Gmd
##   352253        0        3    0.027   0.2965 0.006947
##
## Value        -0.3    0.0    0.3
## Frequency     914   2282 349057
## Proportion  0.003  0.006  0.991
## ---------------------------------------------------------------------------------
## total_amount
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   352253        0     4637        1    14.98    10.46     5.38     6.30
##      .25      .50      .75      .90      .95
##     8.19    11.30    17.55    27.55    36.06
##
## lowest : -200.30  -80.00  -57.30  -52.80  -50.80
## highest:  454.90  462.80  498.80  636.92  753.80
## ---------------------------------------------------------------------------------
## payment_type
##        n  missing distinct     Info     Mean      Gmd
##   352253        0        5    0.749    1.467   0.5126
##
## lowest : 1 2 3 4 5, highest: 1 2 3 4 5
##
## Value           1      2      3      4      5
## Frequency  190434 159617   1606    585     11
## Proportion  0.541  0.453  0.005  0.002  0.000
## ---------------------------------------------------------------------------------
## trip_type
##        n  missing distinct     Info     Mean      Gmd
##   352253        0        2    0.056    1.019  0.03761
##
## Value           1      2
## Frequency  345500   6753
## Proportion  0.981  0.019
## ---------------------------------------------------------------------------------
## congestion_surcharge
##        n  missing distinct     Info     Mean      Gmd
##   352253        0        5    0.445   0.4978   0.8155
##
## lowest : -2.75  0.00  0.75  2.50  2.75, highest: -2.75  0.00  0.75  2.50  2.75
##
## Value       -2.75   0.00   0.75   2.50   2.75
## Frequency       5 288462      6    126  63654
## Proportion  0.000  0.819  0.000  0.000  0.181
## ---------------------------------------------------------------------------------
```

Handling `0` values

`passenger_count` variable has some `0` which must be an input error. We can fix this issue by replacing `0` with the rounded mean of all records. `total_amount` variable has `0` which cannot be handled nor replaced. So, we go with deleting them.

```
avg <- as.integer(round(mean(green$passenger_count), digits = 0))
green$passenger_count <- ifelse(green$passenger_count != 0, green$passenger_count, avg)
green <- green[green$total_amount > 0,]
```

```r
str(green)
```

```
## 'data.frame':    350533 obs. of  17 variables:
##  $ lpep_pickup_datetime : chr  "2020-06-01 00:09:05" "2020-06-01 00:20:05" "2020-06-01
00:30:50" "2020-06-01 00:03:05" ...
##  $ lpep_dropoff_datetime: chr  "2020-06-01 00:22:46" "2020-06-01 00:23:33" "2020-06-01
00:37:49" "2020-06-01 00:16:46" ...
##  $ RatecodeID           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ PULocationID         : int  166 75 74 152 244 93 85 82 174 29 ...
##  $ DOLocationID         : int  141 42 42 247 200 95 188 70 127 92 ...
##  $ passenger_count      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ trip_distance        : num  3.43 1.03 1.22 2.59 5.95 ...
##  $ fare_amount          : num  13 5 7 11.5 17.5 11 5 12 11.5 61 ...
##  $ extra                : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ mta_tax              : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ tip_amount           : num  3.41 0 0 0 6 0 0 2.66 0 2.75 ...
##  $ tolls_amount         : num  0 0 0 0 2.8 0 0 0 0 0 ...
##  $ improvement_surcharge: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
##  $ total_amount         : num  20.5 6.3 8.3 12.8 27.6 ...
##  $ payment_type         : int  1 2 2 2 1 2 2 1 2 1 ...
##  $ trip_type            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ congestion_surcharge : num  2.75 0 0 0 0 0 0 0 0 0 ...
```

Convert the negative values to positive by obtaining `abs`

```r
green$fare_amount <- abs(green$fare_amount)
green$extra <- abs(green$extra)
green$mta_tax <- abs(green$mta_tax)
green$tip_amount <- abs(green$tip_amount)
green$tolls_amount <- abs(green$tolls_amount)
green$improvement_surcharge <- abs(green$improvement_surcharge)
green$total_amount <- abs(green$total_amount)
```

Convert categorical variables to factor type

```r
green$RatecodeID <- factor(green$RatecodeID)
green$PULocationID <- factor(green$PULocationID)
green$DOLocationID <- factor(green$DOLocationID)
green$payment_type <- factor(green$payment_type)
green$trip_type <- factor(green$trip_type)
```

```r
str(green)
```

```
## 'data.frame':    350533 obs. of  17 variables:
##  $ lpep_pickup_datetime : chr  "2020-06-01 00:09:05" "2020-06-01 00:20:05" "2020-06-01
00:30:50" "2020-06-01 00:03:05" ...
##  $ lpep_dropoff_datetime: chr  "2020-06-01 00:22:46" "2020-06-01 00:23:33" "2020-06-01
00:37:49" "2020-06-01 00:16:46" ...
##  $ RatecodeID           : Factor w/ 6 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ PULocationID         : Factor w/ 237 levels "3","4","5","6",..: 148 68 67 136 219 85 78
75 156 24 ...
##  $ DOLocationID         : Factor w/ 258 levels "1","3","4","5",..: 135 41 41 240 193 94 182
69 121 91 ...
##  $ passenger_count      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ trip_distance        : num  3.43 1.03 1.22 2.59 5.95 ...
##  $ fare_amount          : num  13 5 7 11.5 17.5 11 5 12 11.5 61 ...
##  $ extra                : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

```
## $ mta_tax                : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
## $ tip_amount             : num  3.41 0 0 0 6 0 0 2.66 0 2.75 ...
## $ tolls_amount           : num  0 0 0 0 2.8 0 0 0 0 0 ...
## $ improvement_surcharge: num  0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 0.3 ...
## $ total_amount           : num  20.5 6.3 8.3 12.8 27.6 ...
## $ payment_type           : Factor w/ 5 levels "1","2","3","4",..: 1 2 2 2 1 2 2 1 2 1 ...
## $ trip_type              : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
## $ congestion_surcharge : num  2.75 0 0 0 0 0 0 0 0 0 ...
```

Detecting outliers

Using IQR

After we calculate the outliers using IQR, we inspect each variable to double check and remove manually what we, as exports, believe that it is an outlier.

```
iqr_outliers_min_max(green %>% select_if(is.numeric))
```

```
##                       minVal maxVal
## passenger_count        1.000  1.000
## trip_distance         -2.225  6.455
## fare_amount           -4.750 25.250
## extra                 -0.750  1.250
## mta_tax                0.500  0.500
## tip_amount            -3.000  5.000
## tolls_amount           0.000  0.000
## improvement_surcharge  0.300  0.300
## total_amount          -5.650 31.550
## congestion_surcharge   0.000  0.000
```

```
system.time(iqr.outliers <- green %>% select_if(is.numeric) %>% lapply(iqr_outliers_finder))
```

```
##    user  system elapsed
##   0.153   0.025   0.179
```

Now, let us inspect variable values ### passenger_count

```
get_freq_df_from_vector(iqr.outliers$passenger_count, "passenger_count")
```

```
##   passenger_count  Freq
## 8               9     2
## 7               8     6
## 6               7     6
## 5               6  5620
## 4               5  9695
## 3               4  1680
## 2               3  5224
## 1               2 26024
```

We can remove observations with `passenger_count > 6` as it is not logical to fit 6 or more people in a taxi.

```
num.obs <- nrow(green)
green <- green[green$passenger_count <= 6, ]
cat("[", num.obs-nrow(green), "] Observation have been deleted! Which is ~ (", round((num.obs-
        nrow(green))/num.obs, digits = 2), ")")
```

```
## [ 14 ] Observation have been deleted! Which is ~ ( 0 )
```

trip_distance

```
get_freq_df_from_vector(iqr.outliers$trip_distance, "trip_distance")
```

```
## Highest values :
##      trip_distance Freq
## 2345      134121.5    1
## 2344        130.68    1
## 2343        124.13    1
## 2342         84.97    1
## 2341         80.15    1
## 2340         79.69    1
## Lowest values :
##   trip_distance Freq
## 6           6.5  235
## 5          6.49   69
## 4          6.48   84
## 3          6.47   76
## 2          6.46   82
## 1         -3.05    1
```

1. Drop the row with `trip_distance == 134121.5`
2. Take the absolute value of the negative ones

```
green$trip_distance <- abs(green$trip_distance)
green <- green[green$trip_distance < 150, ]

num.obs <- nrow(green)
cat("[", num.obs-nrow(green), "] Observation have been deleted! Which is ~ (", round((num.obs-
        nrow(green))/num.obs, digits = 3), ")")
```

```
## [ 0 ] Observation have been deleted! Which is ~ ( 0 )
```

fare_amount

```
get_freq_df_from_vector(iqr.outliers$fare_amount, "fare_amount")
```

```
## Highest values :
##     fare_amount Freq
## 413         753    1
## 412         630    1
## 411       335.5    1
## 410       233.5    1
## 409       228.5    1
## 408         225    3
## Lowest values :
##   fare_amount Freq
## 6        25.5 1162
## 5       25.49    1
## 4       25.47    1
## 3       25.44    1
## 2       25.41    1
## 1        25.3    1
```

`753` is an outlier as the trip distance is only `7.49` => drop `335.5` is an outlier as the trip distance is only `6.97` => drop Others seem to be accepted

```
green <- green[green$fare_amount %notin% c(753, 335.5),]
```

```r
num.obs <- nrow(green)
cat("[", num.obs-nrow(green), "] Observation have been deleted! Which is ~ (", round((num.obs-
        nrow(green))/num.obs, digits = 3), ")")
```

```
## [ 0 ] Observation have been deleted! Which is ~ ( 0 )
```

extra

```r
get_freq_df_from_vector(iqr.outliers$extra, "extra")
```

```
##   extra Freq
## 4   4.5  108
## 3  3.75 2062
## 2  3.25 2077
## 1  2.75 5401
```

mta_tax

```r
get_freq_df_from_vector(iqr.outliers$mta_tax, "mta_tax")
```

```
##   mta_tax Freq
## 2    3.55    5
## 1       0 6346
```

tip_amount

```r
get_freq_df_from_vector(iqr.outliers$tip_amount, "tip_amount")
```

```
## Highest values :
##      tip_amount Freq
## 1000        480    1
## 999         450    1
## 998       449.6    1
## 997         333    1
## 996      300.08    1
## 995      282.88    1
## Lowest values :
##   tip_amount Freq
## 6       5.06  117
## 5       5.05   21
## 4       5.04   59
## 3       5.03   21
## 2       5.02    3
## 1       5.01  324
```

We remove observations with `tip_amount > 0.7 of the`total_amount`

```r
green <- green[(green$tip_amount/green$total_amount) <= 0.7,]

num.obs <- nrow(green)
cat("[", num.obs-nrow(green), "] Observation have been deleted! Which is ~ (", round((num.obs-
        nrow(green))/num.obs, digits = 3), ")")
```

```
## [ 0 ] Observation have been deleted! Which is ~ ( 0 )
```

tolls_amount

```
get_freq_df_from_vector(iqr.outliers$tolls_amount, "tolls_amount")
```

```
## Highest values :
##    tolls_amount Freq
## 68        96.12    1
## 67        48.88    1
## 66        45.75    1
## 65        39.74    1
## 64           35    1
## 63        31.99    1
## Lowest values :
##   tolls_amount Freq
## 6          2.8  433
## 5         2.75   12
## 4         2.64    2
## 3         2.29  197
## 2            2    3
## 1            1    3
```

96.12, 48.88, 35.0 and all values with percentage 0.9 or higher of the total_amount are outliers to be removed.

```
green <- green[green$tolls_amount/green$total_amount < 0.5, ]

num.obs <- nrow(green)
cat("[", num.obs-nrow(green), "] Observation have been deleted! Which is ~ (", round((num.obs-
        nrow(green))/num.obs, digits = 3), ")")
```

```
## [ 0 ] Observation have been deleted! Which is ~ ( 0 )
```

improvement_surcharge

```
get_freq_df_from_vector(iqr.outliers$improvement_surcharge, "improvement_surcharge")
```

```
##   improvement_surcharge Freq
## 1                     0 1476
```

total_amount

```
get_freq_df_from_vector(iqr.outliers$total_amount, "total_amount")
```

```
## Highest values :
##      total_amount Freq
## 2645        753.8    1
## 2644       636.92    1
## 2643        498.8    1
## 2642        462.8    1
## 2641        454.9    1
## 2640        357.3    1
## Lowest values :
##   total_amount Freq
## 6        31.63    1
## 5        31.62   18
## 4         31.6    7
## 3        31.59   14
```

```
## 2        31.57    1
## 1        31.56   61
```

`congestion_surcharge`

```r
get_freq_df_from_vector(iqr.outliers$congestion_surcharge, "congestion_surcharge")
```

```
##   congestion_surcharge  Freq
## 3                 2.75 63654
## 2                  2.5   126
## 1                 0.75     6
```

Save valid data "after dropping outliers"

```r
system.time(saveRDS(green, file = "data/valid_data.rds"))
```

```
##    user  system elapsed
##   1.662   0.012   1.681
```

Load valid data

```r
system.time(green <- readRDS("data/valid_data.rds"))
```

```
##    user  system elapsed
##   0.437   0.011   0.449
```

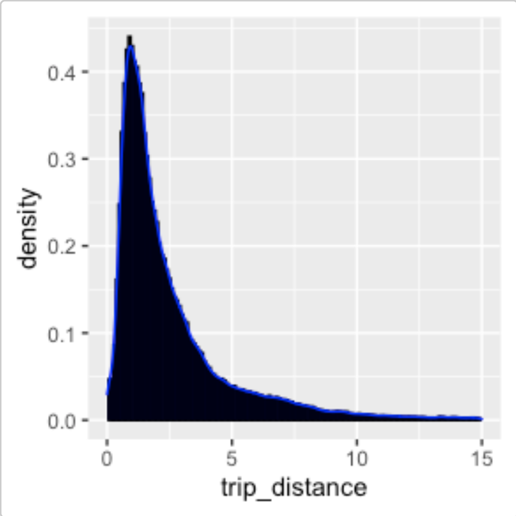
Density distribution of `trip_distance`

```r
ggplot(green, aes(x=trip_distance)) +
        geom_histogram(aes(y=..density..), binwidth=.1, colour="black", fill="white") +
        geom_density(alpha=.2, colour="blue", fill="#000066")+  xlim(0, 15)
```

```
## Warning: Removed 5478 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 5478 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



Classification

Add mean and median of trip_distance grouped hour of pickup time

```
st <- proc.time()
green$pickup_hour <- as.integer(format(strptime(green$lpep_pickup_datetime, "%Y-%m-%d
        %H:%M:%S"),"%H"))
hourly_trip_distance <- data.frame( green %>%
                                  group_by(pickup_hour) %>%
                                  summarise(mean_trip_dist = mean(trip_distance),
                                            median_trip_dist = median(trip_distance)) %>%
        ungroup())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

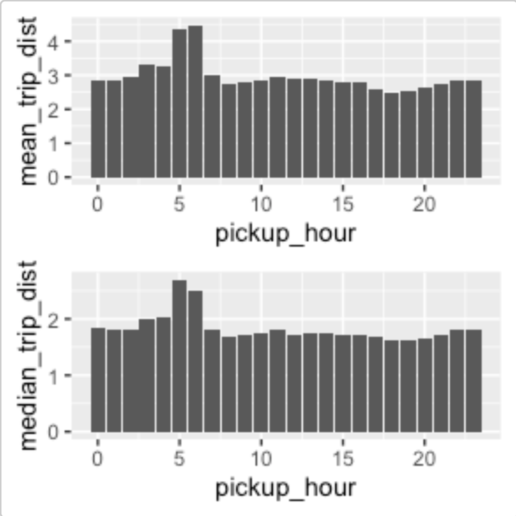
```
head(hourly_trip_distance)
```

```
##   pickup_hour mean_trip_dist median_trip_dist
## 1           0       2.833005            1.850
## 2           1       2.848688            1.800
## 3           2       2.931595            1.810
## 4           3       3.302185            2.000
## 5           4       3.242655            2.025
## 6           5       4.333602            2.700
```

```
print(proc.time() - st)
```

```
##    user  system elapsed
##   1.176   0.025   1.203
```

```
# Mean trip distance plot
m.trip.dist.plt <- ggplot(hourly_trip_distance, aes(x=pickup_hour, y=mean_trip_dist)) +
        geom_bar(stat = "identity")
# Median trip distance plot
M.trip.dist.plt <- ggplot(hourly_trip_distance, aes(x=pickup_hour, y=median_trip_dist)) +
        geom_bar(stat = "identity")
grid.arrange(m.trip.dist.plt, M.trip.dist.plt, ncol=1, nrow =2)
```



From above, we conclude that the longest trips are at 5 & 6 in the morning. In rest of the day's hours trips are somehow close in terms of distance traveled.
//

`tip_percentage` is the tip percentage based on `total_amount`

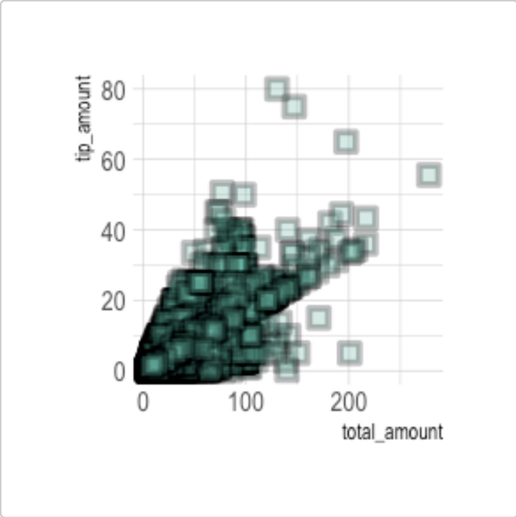
```
st <- proc.time()
green$tip_percentage <- ifelse(green$tip_amount==0.0 | green$total_amount==0.0 , 0.0,
                        round(green$tip_amount/green$total_amount,3))
# Remove all zero percentage as they are not going to help us in classification
num.obs <- nrow(green)
green <- green[green$tip_percentage > 0,]
cat("[", num.obs-nrow(green), "] Observation have been deleted! Which is ~ (", round((num.obs-
        nrow(green))/num.obs, digits = 3), ")")
```

```
## [ 188251 ] Observation have been deleted! Which is ~ ( 0.537 )
```

```
print(proc.time() - st)
```

```
##    user  system elapsed
##   0.122   0.024   0.146
```

```
ggplot(green, aes(x=total_amount, y=tip_amount)) +
    geom_point(
        color="black",
        fill="#69b3a2",
        shape=22,
        alpha=0.3,
        size=3,
        stroke =2
        ) +
    theme_ipsum()
```



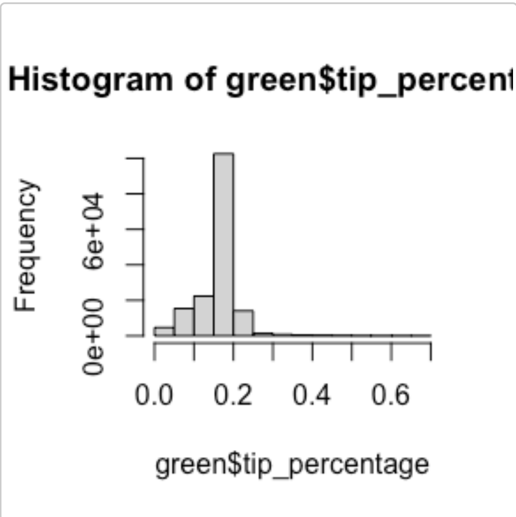
geom_point is useful when we want to compare two continuous variables. //

```
cat("Average tip percentage of the total amount ~ (",
    round((sum(green$tip_amount)/sum(green$total_amount)),4)*100 ," %)")
```

```
## Average tip percentage of the total amount ~ ( 15.59  %)
```

Histogram of `tip_percentage`

```
hist(green$tip_percentage)
```



Classification using Decision Tree ### Split data into training & testing

```
sample.size <- floor(0.75*nrow(green))
s <- sample(seq_len(nrow(green)), sample.size)
numeic.cols <- green %>% select_if(is.numeric)
train.set <- numeic.cols[s,]
test.set <- numeic.cols[-s, ]
```

```
dt.model <- rpartXse(tip_percentage ~ ., train.set, se = 0.5)
dt.predicted <- round(predict(dt.model, test.set), digits = 3)
saveRDS(file = "data/dt_model.rds", object = dt.model)
saveRDS(file = "data/dt_pred.rds", object = dt.predicted)
head(dt.predicted)
```

```
##     20     63     65     92     93    104
```

```
## 0.167 0.057 0.093 0.167 0.167 0.065
```

```r
print(proc.time() - st, paste("\n"))
```

```
##    user  system elapsed
##  22.025   0.269  22.372
```

Load DT Model

```r
dt.model <- readRDS("data/dt_model.rds")
dt.predicted <- readRDS("data/dt_pred.rds")
```

Predicted vs original tip percentage using `Random Forest Tree`

```r
dt.perf.matrix <- as.data.frame(test.set$tip_percentage) %>% cbind(dt.predicted)
colnames(dt.perf.matrix) <- c("actual", "pred")
dt.perf.matrix["error"] <- dt.perf.matrix["actual"]-dt.perf.matrix["pred"]
dt.perf.matrix["error/actual"] <- abs(dt.perf.matrix["error"]/dt.perf.matrix["actual"])
head(dt.perf.matrix)
```

```
##     actual  pred error error/actual
## 20   0.167 0.167 0.000   0.00000000
## 63   0.059 0.057 0.002   0.03389831
## 65   0.093 0.093 0.000   0.00000000
## 92   0.167 0.167 0.000   0.00000000
## 93   0.167 0.167 0.000   0.00000000
## 104  0.066 0.065 0.001   0.01515152
```

Measuring performance using Mean Absolute Percentage Error (MAPE)

```r
dt.mape <- mean(dt.perf.matrix$`error/actual`)
cat("Error percentage: (", round(dt.mape, 4), ") \nSuccess percentage: (", round(100-dt.mape,
        4),")")
```

```
## Error percentage: ( 0.0048 )
## Success percentage: ( 99.9952 )
```

```r
dt.mse <- mse(test.set$tip_percentage, dt.predicted)
dt.mae <- mae(test.set$tip_percentage, dt.predicted)
dt.rmse <- rmse(test.set$tip_percentage, dt.predicted)
dt.r2 <- R2(test.set$tip_percentage, dt.predicted, form = "traditional")

cat(" MAE:", dt.mae, "\n", "MSE:", dt.mse, "\n",
    "RMSE:", dt.rmse, "\n", "R-squared:", dt.r2)
```

```
##  MAE: 0.0004403405
##  MSE: 8.424944e-06
##  RMSE: 0.002902575
##  R-squared: 0.9968075
```

As we can see from above, if we accept only 3 digits after the point, which may cause some lose in data thus error, we obtained a very good success rate using RT. // ## Using Support Vector Machine (SVM)

```
st <- proc.time()
svm.model <- svm(tip_percentage ~ ., train.set)
svm.predicted <- round(predict(svm.model, test.set), digits = 3)
saveRDS(object = svm.model, file = "data/svm_model.rds")
saveRDS(object = svm.predicted, file = "data/svm_pred.rds")
print(proc.time() - st)
```

```
##    user  system elapsed
## 174.749   0.700 175.792
```

Load SVM Model

```
svm.model <- readRDS("data/svm_model.rds")
svm.predicted <- readRDS("data/svm_pred.rds")
```

Measuring performance using Mean Absolute Percentage Error (MAPE)

```
svm.perf.matrix <- as.data.frame(test.set$tip_percentage) %>% cbind(svm.predicted)
colnames(svm.perf.matrix) <- c("actual", "pred")
svm.perf.matrix["error"] <- svm.perf.matrix["actual"]-svm.perf.matrix["pred"]
svm.perf.matrix["error/actual"] <- abs(svm.perf.matrix["error"]/svm.perf.matrix["actual"])
head(svm.perf.matrix)
```

```
##     actual  pred error error/actual
## 20   0.167 0.166 0.001  0.005988024
## 63   0.059 0.055 0.004  0.067796610
## 65   0.093 0.091 0.002  0.021505376
## 92   0.167 0.163 0.004  0.023952096
## 93   0.167 0.166 0.001  0.005988024
## 104  0.066 0.064 0.002  0.030303030
```

```
svm.mape <- mean(svm.perf.matrix$`error/actual`)
cat("Error percentage: (", round(svm.mape, 4), ") \nSuccess percentage: (", round(100-
        svm.mape, 4),")")
```

```
## Error percentage: ( 0.0516 )
## Success percentage: ( 99.9484 )
```

```
svm.mse <- mse(test.set$tip_percentage, svm.predicted)
svm.mae <- mae(test.set$tip_percentage, svm.predicted)
svm.rmse <- rmse(test.set$tip_percentage, svm.predicted)
svm.r2 <- R2(test.set$tip_percentage, svm.predicted, form = "traditional")

cat(" MAE:", svm.mae, "\n", "MSE:", svm.mse, "\n",
    "RMSE:", svm.rmse, "\n", "R-squared:", svm.r2)
```

```
##  MAE: 0.00232149
##  MSE: 3.388009e-05
##  RMSE: 0.005820661
##  R-squared: 0.9869203
```

Using Neural Network

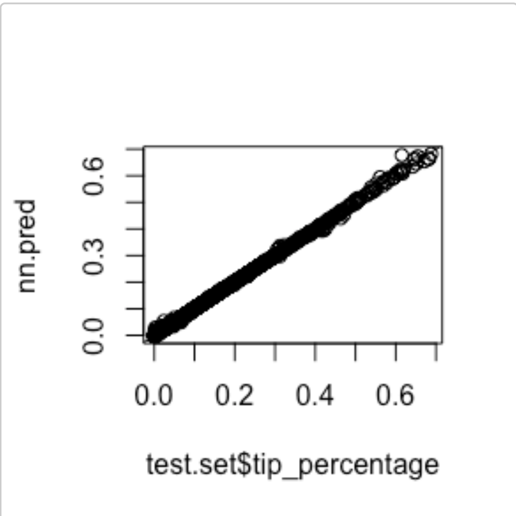
```
st <- proc.time()
nn.model <- nnet(tip_percentage ~ ., train.set,
           linout=TRUE,
           trace=FALSE,
           size=6,
           decay=0.01,
           maxit=2000)
nn.pred <- predict(nn.model, test.set)
saveRDS(nn.model, "data/nn_model.rds")
saveRDS(nn.pred, "data/nn_pred.rds")
print(proc.time() - st)
```

```
##    user  system elapsed
##  31.972   0.184  32.276
```

Load NN model

```
nn.model <- readRDS("data/nn_model.rds")
nn.pred <- readRDS("data/nn_pred.rds")
```

```
plot(test.set$tip_percentage, nn.pred)
abline(0, 1)
```



From the plot above, we realize that when the `tip_percentage` is larger the errors, or the residuals become larger. //

Starting `H2O` Scalable platform that parallelize many machine learning algorithms

```
system.time(h2oInstance <- h2o.init()) # start H2O instance locally
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##
/var/folders/dw/1n2ny0fj771bb5fgcnlrfdl80000gn/T//Rtmpp74DHW/file1e52d726a20/h2o_betulbayrak_started_from_

##
/var/folders/dw/1n2ny0fj771bb5fgcnlrfdl80000gn/T//Rtmpp74DHW/file1e52529561c5/h2o_betulbayrak_started_from

##
```

```
##
## Starting H2O JVM and connecting: ........... Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         10 seconds 5 milliseconds
##     H2O cluster timezone:       Europe/Istanbul
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.32.0.1
##     H2O cluster version age:    3 months and 21 days !!!
##     H2O cluster name:           H2O_started_from_R_betulbayrak_uik133
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.56 GB
##     H2O cluster total cores:    0
##     H2O cluster allowed cores:  0
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder,
Core V4
##     R Version:                  R version 4.0.3 (2020-10-10)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (3 months and 21 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

```
##    user  system elapsed
##   0.246   0.076  16.021
```

Using H2O build a deep neural network with the following parameters

```
st <- proc.time()
trH <- as.h2o(train.set,"trH")
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
tsH <- as.h2o(test.set,"tsH")
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
mdl <- h2o.deeplearning(x=1:11, y=12, training_frame=trH,hidden = c(100, 100, 100, 100, 100,
        100, 100), epochs = 1000)
```

```
##
  |
  |                                                                      |   0%
  |
  |                                                                      |   1%
  |
  |=                                                                     |   1%
  |
  |=                                                                     |   2%
  |
  |======================================================================| 100%
```

```
preds <- as.vector(h2o.predict(mdl,tsH))
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

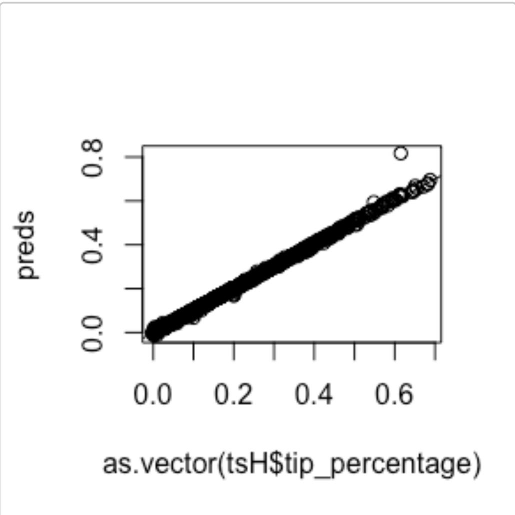
```
print(proc.time() - st)
```

```
##    user  system elapsed
##   2.787   0.157  93.180
```

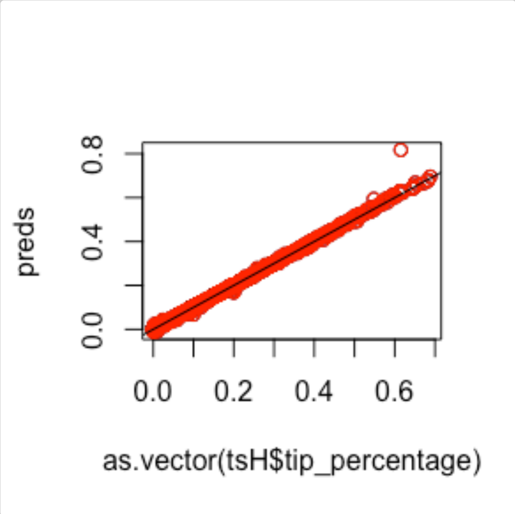
```
mean(abs(preds - as.vector(tsH$tip_percentage)))
```

```
## [1] 0.0004807213
```

```
plot(as.vector(tsH$tip_percentage), preds)
abline(0, 1)
```




```
plot(as.vector(tsH$tip_percentage), preds)
points(as.vector(tsH$tip_percentage), preds, col = "red")
abline(0, 1)
```




```
h2o.shutdown(prompt = F);
```

```
mars1 <- earth(
tip_percentage ~ .,
data = train.set)
```

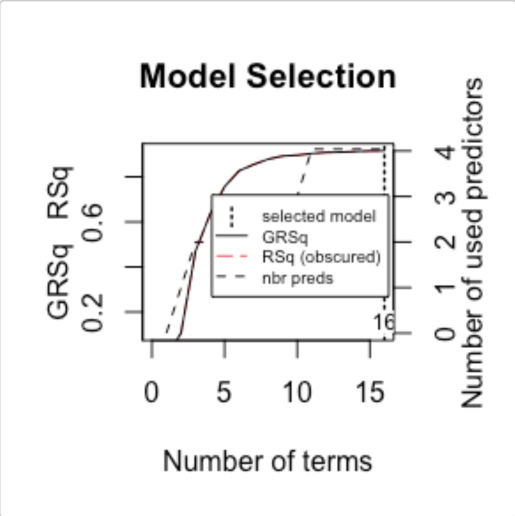
```
print(mars1)
```

```
## Selected 16 of 16 terms, and 4 of 11 predictors
## Termination condition: Reached nk 23
## Importance: tip_amount, total_amount, fare_amount, tolls_amount, ...
## Number of terms at each degree of interaction: 1 15 (additive model)
## GCV 0.0002224128    RSS 27.02916    GRSq 0.9151765    RSq 0.9152183
```

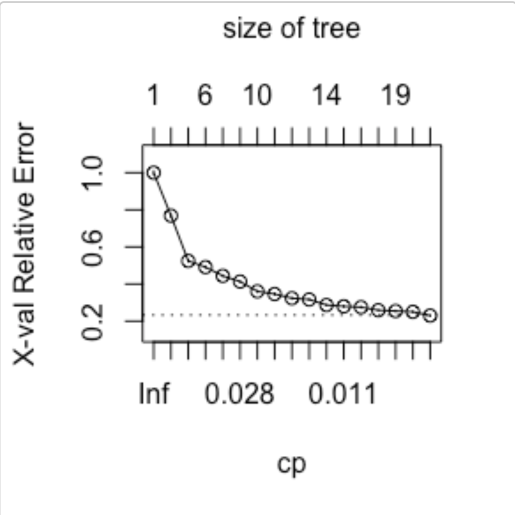
```
summary(mars1) %>% .$coefficients #%>% head(10)
```

```
##                         tip_percentage
## (Intercept)                0.075024794
## h(tip_amount-1.47)         0.070856665
## h(1.47-tip_amount)        -0.098625575
## h(total_amount-12.51)     -0.008962479
## h(12.51-total_amount)      0.006912660
## h(tip_amount-2.44)        -0.024750395
## h(fare_amount-11.5)       -0.007578864
## h(11.5-fare_amount)        0.013890228
## h(tip_amount-4.14)        -0.012601374
## h(fare_amount-21.5)        0.006267890
## h(tolls_amount-3)          0.012704593
## h(3-tolls_amount)         -0.002412357
## h(fare_amount-6.5)         0.010420321
## h(tip_amount-8.8)         -0.012387461
## h(fare_amount-37)          0.002614235
## h(total_amount-30.05)     -0.004122108
```

```
plot(mars1, which = 1)
```




```
optimal_tree <- rpart(
formula = tip_percentage ~ .,
data = train.set,
method = "anova",
control = list(minsplit = 11, maxdepth = 8, cp = 0.01)
)
plotcp(optimal_tree)
```



Conclusion

We conclude that data is the chief factor that levels up the model's accuracy so, we first need to be aware of the data and understand it as much as possible. After that, we build the regression model using many available ones such as SVM, Decision Tree, and DNN to predict the tip percentage of the total paid amount given other variables. All the used models gave us similar accuracies of ~ 0.99 that indicates successful training using the provided data.