JANUARY 2021

Project Report On Mobile Ad Click Prediction

Overview

Machine Learning has a variety of applications in the real world whether it is for predictive analysis or automating things or decision making for business purposes. Having knowledge of Machine Learning is different but applying it to business purposes for making profit, and learning something new from the data is the main motive.Today, a company wants to know the CTR (Click Through Rate) in order to identify whether spending their money on digital advertising is worth it or not.A higher CTR represents more interest in that specific campaign, whereas a lower CTR can show that your ad may not be as relevant. High CTRs are important because they show that more people are clicking through to your website. Along with this high CTRs also help to get better ad positions for less money on online platforms like Google, Bing etc.But higher CTR's may not always mean higher profits! Many times higher CTR's can be deceitful because if a keyword isn't pertinent to your business or isn't going to generate sales, leads, branding gains, etc. then a high click-through rate for that term is actually bad for business. Most times users click the ad but do not get converted into customers (Users who download the apps). With the rapid increase in the advertising network, click prediction needs huge data analysis. The advertisement prediction is considered to be one of the most lucrative stories in the domain of machine learning.

Problem Statement

Predicting whether a user will download an app after clicking a mobile app advertisement.

Objectives

General Objectives

The general objective of this project is with the help of various machine learning algorithms we are able to classify the user data in a better way and are able to make accurate predictions so as to whether users make the choice of downloading apps or not.

Specific Objectives

The specific objectives of this project are:

- Applying the three machine learning algorithms: Random Forest, Regression Tree, Logistic Regression Model as a part to recognise whether the user input is fruitful or not.
- Drawing contrast between all the algorithms and selecting the best out of them based on various aspects primarily the confusion matrix and accuracy parameters.
- Analysis of the theoretical and literatures relevant to the project and using their results as a reference point for our study.
- To associate the knowledge aspects in training different machine learning models to learn from the large set of user click input data set.

Approach/ Methodology

Algorithm

We approach this project by dividing our problem statement into four significant parts.

- We first deal with data munging and the testing of three machine learning models (Logistic Regression Model, Regression tree model with most significant variables and Random Forest Model) using a training sample and testing with 1E+07 rows of the sample.
- We perform exploratory data analysis at this point and work on cleaning the data by dealing with missing values and reducing the quantity of various factors to balance the train target feature.
- In the second part of the approach we will deal with tidying the training dataset by calculating the number of batches and further transforming the training dataset accordingly.
- In the third part, the tidied training dataset will be taken with the best model acquired among the three machine learning models to train the model further, here we change some features to factor and create a random forest model.In the fourth part of the approach the trained model will be applied to our test dataset and afterward, the predicted results will be matched to produce the final result.

Models

The following models have been used in the project:

- **Logistic regression with most significant variables:**
  Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no). A logistic regression model predicts P(Y=1) as a function of X.

- **Regression tree model with most significant variables:**
  A regression tree is built through a process known as binary recursive partitioning, which is an iterative process that splits the data into partitions or branches, and then continues splitting each partition into smaller groups as the method moves up each branch.Initially, all records in the Training Set are grouped into the same partition. The algorithm then begins allocating the data into the first two partitions or branches, using every possible binary split on every field.

- **Random Forest Model:**
  Random forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classesor mean/average prediction of the individual trees. Random decision forests correct the decision tree's habit of overfitting to their training set.

Prerequisites

Packages/Tools used

We use R studio to write our code and train and test our model. The R packages used in this project are as follows:

dplyr: This package provides us a set of tools for efficiently manipulating datasets in R, it is the next iteration of plyr.

lubridate: This package makes it easier to work with dates and time.

ggplot2: We use this package for data visualization and for making graphs in our project

ggthemes: Provides a variety of themes for the charts which we can use for our project

mltools: This package helps in the exploratory data analysis

phasedata.table: It offers fast and memory efficient - file reader and writer, aggregations, updates, equi, non-equi, rolling, range and interval joins, in a short and flexible syntax, for faster development.

Caret: The caret package (short for Classification And REgression Training) contains functions to streamline the model training process for complex regression and classification problems.

ROCR: This package is used for evaluating and visualizing the performance of scoring classifiers in the statistical language

R.knitr: It is a General-Purpose Package for Dynamic Report Generation in R.

rmarkdown: R Markdown supports dozens of static and dynamic output formats we have used in the project.

Experimental Investigation

Exploratory Data Analysis

- Training set: 1,00,000 obs. Of 8 variables
- Test set : 10,00,00,000 obs. Of 8 variables
- We found no missing values in the train and the test set.
- The target variable(is_attributed) is categorical along with the other variables like IP, variables, app, device, OS and channel.
- The target variable is_attributed has 99773 values of class 0 and 227 values of class 1 in the training dataset.
- From the ips analysis,we found out that there are around 34857 unique and 65143 duplicate ips in the training dataset.
- The click_time and the attributed_time feature of the training dataset has been taken from 6th November 2017 to 9th November 2017.
- From the click_time analysis, we found that the click_time attribute of the training data has days that are weekdays. 6th = Monday, 7th = Tuesday,8th =Wednesday, 9th =Thursday

- The click_time attribute of the test dataset has only 10th November 2017's data.

- From the click_time analysis, we found out that 6th November 2017 has only partial data in the training dataset.

- From the attributed_time_features analysis, we found out that the number of clicks in function of the day on the 6th day is 5011,7th day is 32393, 8th day is 34035 and 9th day is 28561 in the training dataset.

- From the attributed time feature analysis, it was found that the number of app downloads on a particular day is shown by the attributed_day function. The number of downloads on the corresponding days are as follows:

  6th Nov,2017 – 4
  7th Nov,2017 – 76
  8th Nov,2017 – 85
  9th Nov,2017 – 62

- As the number of app downloads are less on the days 6th and 9th, we remove those two days from the dataset.

- The number of downloads on 7th and 8th decreases after 16 hours.

- The app attribute is used for marketing purpose. From the app_id analysis, it was found that, we have around 17500 number of data of class type 1 in the training dataset.

- The device attribute gives the device type id of user's mobile phone. From the device attribute analysis, we found that the number of devices of the corresponding class types in the training dataset are as follows:

Device	n
1	94338
2	4345
0	541

and many more.

- We also found out that 58% of all devices in the training dataset are of type 1.

- The OS attribute is the OS id of user mobile phone. From the OS attribute analysis, we found out that 40% of the systems are represented by type 19 and 13 in the training dataset.

- The channel attribute is the channel id of mobile ad publisher.

Data Preprocessing

- In the Ips Analysis, we observe that the yes proportions seems to behave like a sinusoid. Therefore, we add another column according to the yes_prop values. We do this for the train_set and the test_set both.

- We introduced a new feature containing the day of the click_time while doing the click_time analysis. We do this for the train_set and the test_set both.

- While doing the attributed time features analysis, we introduced two new features namely, attributed_day and attributed_hour. We do this for the train_set and the test_set both.

- Then we introduce another feature named app_fac which includes all the unique app_ids. We do this for the train_set and the test_set both.

- In this step, we introduce a new feature named device_fac which contains two classes – one for type 1(majority) and the other for including all the other types. We do this for the train_set and the test_set both.

- Here we introduce another new feature named os_fac, which contains 4 classes- one for type 13 os, one for type 19 os, one for os greater than type 19 and one with all the rest of the types. We do this for the train_set and the test_set both.

- We introduce another variable named channel_fac, which contains channel_id within intervals of 0,135,236,401. We do this for the train_set and the test_set both.

- We then check for the missing values in the newly added features and find out that there are missing values in the features attributed_time, attributed_day and attributed_hour. In order to deal with the missing values, don't include these attributes while training the data.

- In the next step, we reduce the quantity of the class 0 in the target variable(is_attributed) in order to balance the dataset.

Model 1: Logistic Regression with most significant variables

- **Model Building** :
    - Using labels() as an extractor function we first obtain a named feature vector of labels from the test_set .
    - We then assign the variable model2 with a logistic model model using the glm() function which is used to fit generalized logistic models, specified by giving a symbolic description of the linear predictor and a description of the error distribution. In our case the data used to create a logistic regression model is from train_set1 .
    - Within the glm() function , we further create a new variable is_attributed which is a sum of the attributes repetitions, device_fac and os_fac . The family of the data here is binomial, making this a binomial regression model.
    - Following this we obtain a summary of our model using the summary() function .
    - In order to make predictions of our model we use the predict() function , which takes our logistic regression model and the values of the predictor variable (that we want response values for , hence the type is set to “response”)as arguments.

- We assign the predict() function to a new variable called predictions2.

- Then using the round() function we round the value of our predictions2 variables to a specified number of decimal places (in our case 0 as the default is 0), Now our model has been rounded to the nearest whole number.

- We evaluate our Logistic Regression Model by creating a confusion matrix using the ConfusionMatrix() function which calculates the cross tabulation of observed and predicted classes with associated statistics , where the reference is test_set $ is_attributed (the factor of classes used as true results).

- Finally we plot the ROC curve and the precision recall curve using plot_utils.R

Model 2: Regression Tree Model with most significant variables

- **Model Building** :
    - **varImp** is a function used to calculate the variable importance for objects produced by "**train**" and method specific methods.
    - In our case the varImp does the same and gives a list of variables according to their **rpart** variable importance.
    - "**rpart**" stands for recursive partitioning and regression trees. It helps in fitting a model and has a number of arguments such as the formula, weight, data, method etc.
    - In our case **os_fac** is omitted when considering the most significant variable while training the model based on the varImp list.
    - Next the model is trained with the above and is used as an argument in the "**predict**" function corresponding to the test set values.
    - "predict" function returns a vector of predicted responses from a fitted rpart object.

- **Model Prediction :**

  - In order to make predictions of our model we use the predict() function , which takes our random forest model and the values of the predictor variable (that we want the labels assigned to that value, hence the type is set to class)as arguments .

  - Now these predictions are summarized and depicted with the help of a confusion matrix.

  - The "confusionmatrix" is used to calculate a cross-tabulation of observed and predicted classes with associated statistics, the default case character string for the factor level that corresponds to a "positive" result is set to 1.

Model 3: Random Forest model balanced by SMOTE

- **Model Building** :
    - The first step is to import the package DMwR- which includes functions and data accompanying the book "Data Mining with R, learning with case studies" by Luis Torgo, CRC Press 2010 into the workspace. In this particular use case, we are using this package in order use the SMOTE function.
    - Then we import the package random Forest – which is used when we want to include Breiman's random forest algorithm for classification and regression.
    - SMOTE is a well-known algorithm to balance the uneven proportion of cases that are available for each class of the problem. The general idea of this method is to artificially generate new examples of the minority class using the nearest neighbours of these cases. Furthermore, the majority class examples are also under-sampled, leading to a more balanced dataset. We have used SMOTE to balance the number of class 0 and class 1 in the target variable.
    - After that, we use the randomForest() function which implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression.

- Within the randomForest() function , we further create a new variable is_attributed which is a sum of the attributes repetitions_fac multiplied by app and channel multiplied by app_fac. We then include, the train_set1 which contains the variables in the dataframe. We have taken 30 trees and the minimum size of each terminal node is 1.

- In order to make predictions of our model we use the predict() function , which takes our random forest model and the values of the predictor variable (that we want the labels assigned to that value, hence the type is set to class)as arguments .

- We evaluate our Random Forest Model by creating a confusion matrix using the ConfusionMatrix() function which calculates the cross tabulation of observed and predicted classes with associated statistics , where the reference is test_set $ is_attributed (the factor of classes used as true results) .

Inference

- **Model 1:Logistic Regression with most significant variables**
    - It can be inferred from the model that the accuracy in prediction is almost near to 83%.
        - Around 8300K values have correctly been categorized at true negatives
        - 12K as true positives
        - 1700K as false negatives and the rest as false positives.
    - While considering the most significant variables (excluding the os_fac), its balanced accuracy comes out to be around 75% which is quite average when compared to other models.
    - As the sensitivity stands at almost 67% therefore the ratio of correctly +ve identified subjects by test against all +ve subjects in reality is still better than the Regression Tree model but not so good as compared to the Random Forest Model , therefore the model still has scope for improvement.
    - In terms of specificity we are less accurate in predicting the wrong clicks ratioed to all wrong clicks in reality, which stands at 83% whereas for the other 2 models stands at 97%.

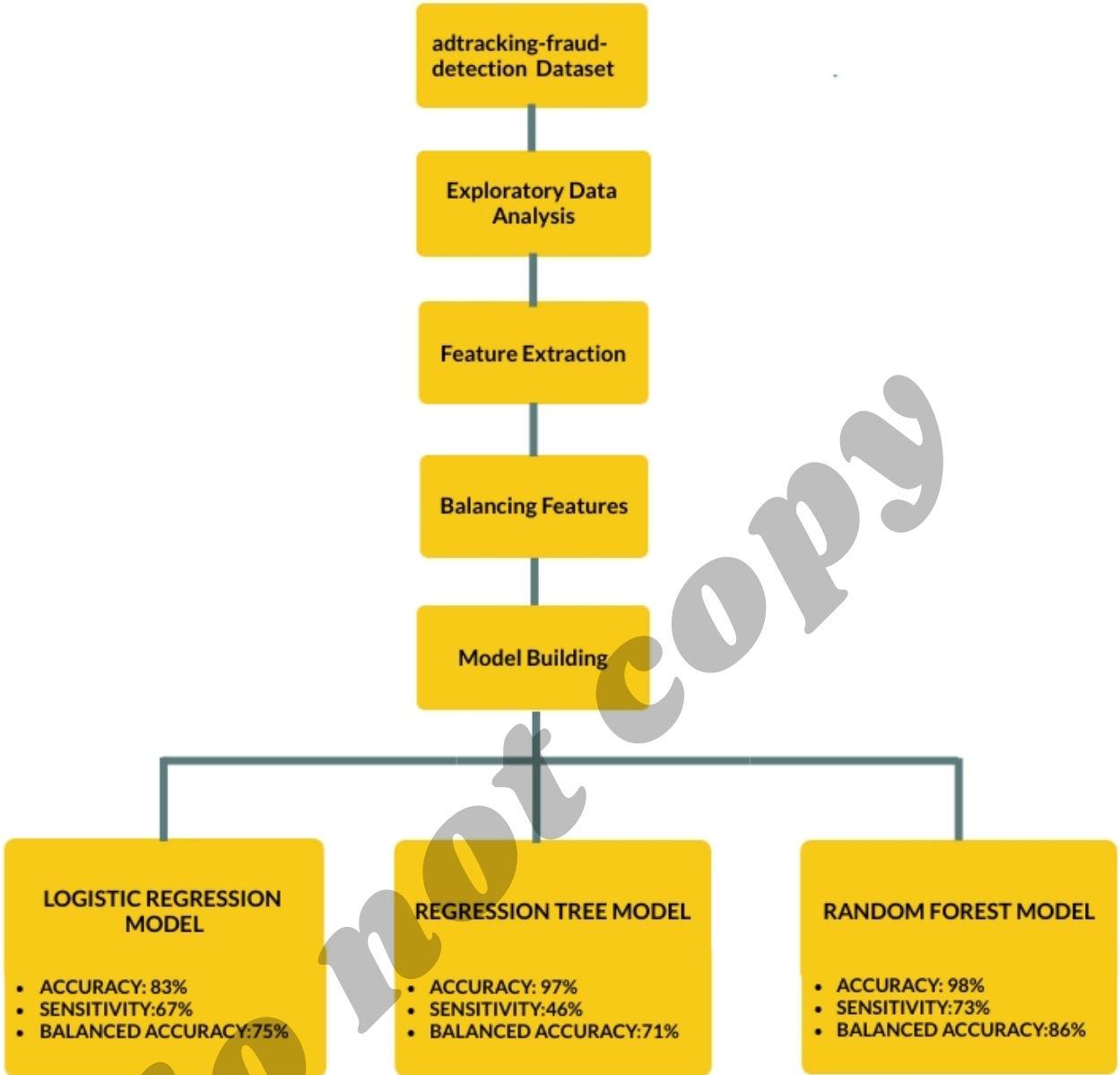
- **Model 2: Regression Tree Model with most significant variables**
    - It can be inferred from the model that the accuracy in prediction is almost near to 97%.
        - Around 9700K values have correctly been categorised at true negatives
        - 8600 as true positives
        - 293K as false negatives and the rest as false positives.
    - While considering the most significant variables (excluding the os_fac), its balanced accuracy comes out to be around 71% which is quite average when compared to other models.
    - As the sensitivity stands at almost 46% therefore the ratio of correctly +ve identified subjects by test against all +ve subjects in reality is still quite low, therefore the model still has scope for improvement, as we are still unable to predict the correct fraudulent clicks.
    - In terms of specificity we are quite accurate in predicting the wrong clicks ratioed to all wrong clicks in reality, which stands at 97%.

- **Model 3: Random Forest model balanced by SMOTE**
    - It can be inferred from the model that the accuracy in prediction is almost near to 98.4%.
        - Around 9800K values have correctly been categorised at true negatives
        - 4954 as true positives
        - 179K as false negatives and the rest as false positives
    - While considering the most significant variables (excluding the os_fac), its balanced accuracy comes out to be around 86% which is quite average when compared to other models.
    - As the sensitivity stands at almost 73% therefore the ratio of correctly +ve identified subjects by test against all +ve subjects in reality is quite high, but there is still scope for improvement, as we are still unable to predict the correct fraudulent clicks.
    - In terms of specificity we are quite accurate in predicting the wrong clicks ratioed to all wrong clicks in reality, which stands at 98%.

Conclusion/Outcome

As seen from the inferences above, we conclude that the random forest model is the best out of all the three models in terms of performance with an accuracy of 98%, sensitivity of 73%, and specificity of 98% which is significantly better than the other two models.

Flow Chart



Screenshots

Data Preprocessing:

Code:

```
setwd("F:/project")

# Packages
library(dplyr)
library(lubridate)
library(ggplot2)
library(ggthemes)
library(mltools)
library(data.table)
library(caret)
library(ROCR)
library(knitr)
library(rmarkdown)

# Read the data sets
train_set <- read.csv(file = 'train_sample.csv', header = T)
#test_set <- fread(file = 'test.csv', header = T)

# The train dataset named train.csv can be found on the web site
# https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data
test_set <- fread(file = 'train.csv', header = T, nrows = 1e7)
```

Output:

```
> setwd("F:/project")
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> library(lubridate)

Attaching package: 'lubridate'

The following objects are masked from 'package:base':

    date, intersect, setdiff, union

> library(ggplot2)
Keep up to date with changes at https://www.tidyverse.org/blog/
```

Code:

```
Console  Terminal  Jobs
F:/project/
warning message:
package 'ROCR' was built under R version 4.0.3
> library(knitr)
warning message:
package 'knitr' was built under R version 4.0.3
> library(rmarkdown)
warning message:
package 'rmarkdown' was built under R version 4.0.3
> train_set <- read.csv(file = 'train_sample.csv', header = T)
> test_set <- fread(file = 'train.csv', header = T, nrows = 1e7)
Error: Unable to establish connection with R session
> # Missing values
>
> any(is.na(train_set))
[1] FALSE
>
> any(is.na(test_set))
[1] FALSE
>
> dim(train_set)
[1] 100000      8
>
```

Output:

```
23  # Missing values
24
25  any(is.na(train_set))
26
27  any(is.na(test_set))
28
29  dim(train_set)
30
31
31:1  (Top Level)  R Script
```

Code:

```
project1.R*   Source on Save   Run   Source
42
43  # The target variable is categorical
44  train_set$is_attributed <- as.factor(train_set$is_attributed)
45  test_set$is_attributed <- as.factor(test_set$is_attributed)
46  table(train_set$is_attributed)
47
48  summary(train_set)
49
50  summary(test_set)
51
52
53
```

Output:

```
F:/project/
> train_set$is_attributed <- as.factor(train_set$is_attributed)
> test_set$is_attributed <- as.factor(test_set$is_attributed)
> table(train_set$is_attributed)

    0     1
99773   227
>
> summary(train_set)
       ip              app              device              os              channel       click_time
 Min.   :     9   Min.   :  1.00   Min.   :   0.00   Min.   :  0.00   Min.   :  3.0   Length:100000
 1st Qu.: 40552   1st Qu.:  3.00   1st Qu.:   1.00   1st Qu.: 13.00   1st Qu.:145.0   Class :character
 Median : 79827   Median : 12.00   Median :   1.00   Median : 18.00   Median :258.0   Mode  :character
 Mean   : 91256   Mean   : 12.05   Mean   :  21.77   Mean   : 22.82   Mean   :268.8
 3rd Qu.:118252   3rd Qu.: 15.00   3rd Qu.:   1.00   3rd Qu.: 19.00   3rd Qu.:379.0
 Max.   :364757   Max.   :551.00   Max.   :3867.00   Max.   :866.00   Max.   :498.0
 attributed_time    is_attributed
 Length:100000      0:99773
 Class :character   1:  227
 Mode  :character


>
```

Code:

```
53
54  # Unique values of the ip feature
55  length(unique(train_set$ip))
56  length(unique(test_set$ip))
57
58  head(rev(sort(table(train_set$ip))))
59
60  head(rev(sort(table(test_set$ip))))
61
62
63
64
65
66
67
68
```


Output:

```
Console  Terminal  Jobs
F:/project/
> # Unique values of the ip feature
> length(unique(train_set$ip))
[1] 34857
> length(unique(test_set$ip))
[1] 68740
>
> head(rev(sort(table(train_set$ip))))

 5348  5314 73487 73516 53454 114276
  669   616   439   399   280    219
>
> head(rev(sort(table(test_set$ip))))

73516 73487  5314  5348 53454 105560
51711 51215 35073 35004 25381 23289
>
```

Code:

```
64
65  # Duplicated ips
66  dupl_ips_train <- train_set[duplicated(train_set$ip), 1]
67  length(dupl_ips_train)
68
69  length(unique(dupl_ips_train))
70
71  round(prop.table(table(train_set$is_attributed[train_set$ip %in%
72                                                 unique(dupl_ips_train)])) * 100, 2)
73
74  dupl_ips_test <- train_set[duplicated(test_set$ip), 1]
75  length(dupl_ips_test)
76
77  length(unique(dupl_ips_test))
78
79  round(prop.table(table(train_set$is_attributed[train_set$ip %in%
80                                                 unique(dupl_ips_test)])) * 100, 2)
81
82
83
```

Output:

```
F:/project/
> # Duplicated ips
> dupl_ips_train <- train_set[duplicated(train_set$ip), 1]
> length(dupl_ips_train)
[1] 65143
>
> length(unique(dupl_ips_train))
[1] 17434
>
> round(prop.table(table(train_set$is_attributed[train_set$ip %in%
+                                                unique(dupl_ips_train)])) * 100, 2)

    0     1
99.91  0.09
>
> dupl_ips_test <- train_set[duplicated(test_set$ip), 1]
> length(dupl_ips_test)
[1] 9931260
>
> length(unique(dupl_ips_test))
[1] 32051
>
> round(prop.table(table(train_set$is_attributed[train_set$ip %in%
+                                                unique(dupl_ips_test)])) * 100, 2)

   0    1
99.8  0.2
>
```

Code:

```
project1.R*
Source on Save   Run   Source
81
82
83
84  # Repeated ips in order
85  n_dupl_ips_train <- train_set %>%
86    count(ip, wt = n() ) %>%
87    arrange(desc(n))
88
89  head(n_dupl_ips_train)
90
91
92  n_dupl_ips_test <- test_set %>%
93    count(ip, wt = n() ) %>%
94    arrange(desc(n))
95
96  head(n_dupl_ips_test)
97
98
99
100
101
102
103
104
105
106
```

Output:

```
Console  Terminal  Jobs
F:/project/
> head(n_dupl_ips_train)
      ip   n
1   5348 669
2   5314 616
3  73487 439
4  73516 399
5  53454 280
6 114276 219
>
>
> n_dupl_ips_test <- test_set %>%
+   count(ip, wt = n() ) %>%
+   arrange(desc(n))
>
> head(n_dupl_ips_test)
       ip     n
1:  73516 51711
2:  73487 51215
3:   5314 35073
4:   5348 35004
5:  53454 25381
6: 105560 23289
```

Code:

```
 98
 99  # Verifying the total of lines
100  sum(n_dupl_ips_train$n)
101
102  sum(n_dupl_ips_test$n)
103
104
105
```

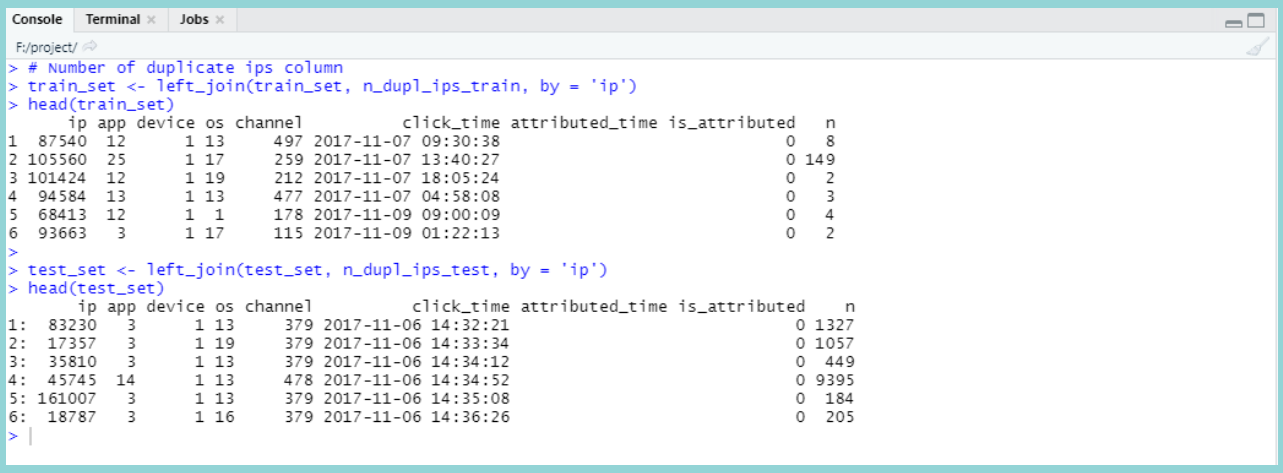
Output:

```
Console  Terminal  Jobs
F:/project/
> # Verifying the total of lines
> sum(n_dupl_ips_train$n)
[1] 100000
>
> sum(n_dupl_ips_test$n)
[1] 10000000
>
```

Code:

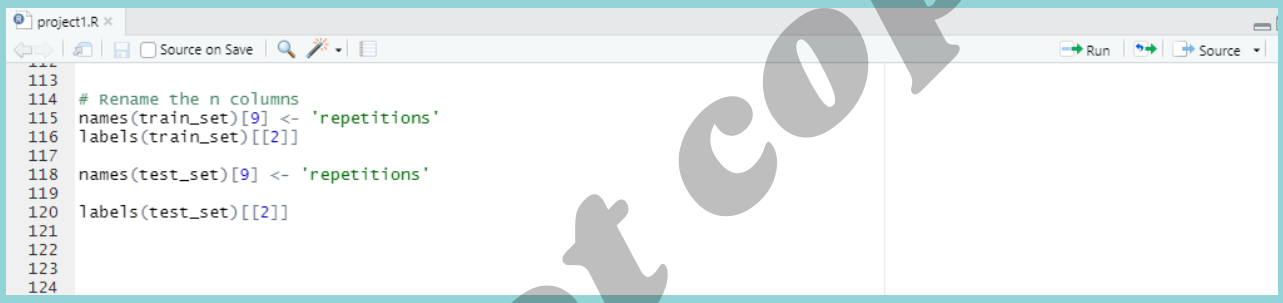
```
project1.R
Source on Save   Run   Source
103
104  # Number of duplicate ips column
105  train_set <- left_join(train_set, n_dupl_ips_train, by = 'ip')
106  head(train_set)
107
108  test_set <- left_join(test_set, n_dupl_ips_test, by = 'ip')
109  head(test_set)
110
111
112
113
114
115
```

Output:



```
Console  Terminal  Jobs
F:/project/
> # Number of duplicate ips column
> train_set <- left_join(train_set, n_dupl_ips_train, by = 'ip')
> head(train_set)
      ip app device os channel          click_time attributed_time is_attributed   n
1  87540  12      1 13     497 2017-11-07 09:30:38                           0   8
2 105560  25      1 17     259 2017-11-07 13:40:27                           0 149
3 101424  12      1 19     212 2017-11-07 18:05:24                           0   2
4  94584  13      1 13     477 2017-11-07 04:58:08                           0   3
5  68413  12      1  1     178 2017-11-09 09:00:09                           0   4
6  93663   3      1 17     115 2017-11-09 01:22:13                           0   2
>
> test_set <- left_join(test_set, n_dupl_ips_test, by = 'ip')
> head(test_set)
       ip app device os channel          click_time attributed_time is_attributed    n
1:  83230   3      1 13     379 2017-11-06 14:32:21                           0 1327
2:  17357   3      1 19     379 2017-11-06 14:33:34                           0 1057
3:  35810   3      1 13     379 2017-11-06 14:34:12                           0  449
4:  45745  14      1 13     478 2017-11-06 14:34:52                           0 9395
5: 161007   3      1 13     379 2017-11-06 14:35:08                           0  184
6:  18787   3      1 16     379 2017-11-06 14:36:26                           0  205
>
```

Code:



```
project1.R
Source on Save   Run   Source
113
114  # Rename the n columns
115  names(train_set)[9] <- 'repetitions'
116  labels(train_set)[[2]]
117
118  names(test_set)[9] <- 'repetitions'
119
120  labels(test_set)[[2]]
121
122
123
124
```

Output:

```
Console  Terminal  Jobs
F:/project/
> names(train_set)[9] <- 'repetitions'
> labels(train_set)[[2]]
[1] "ip"             "app"            "device"         "os"             "channel"        "click_time"
[7] "attributed_time" "is_attributed"  "repetitions"
>
> names(test_set)[9] <- 'repetitions'
>
> labels(test_set)[[2]]
[1] "ip"             "app"            "device"         "os"             "channel"        "click_time"
[7] "attributed_time" "is_attributed"  "repetitions"
>
```

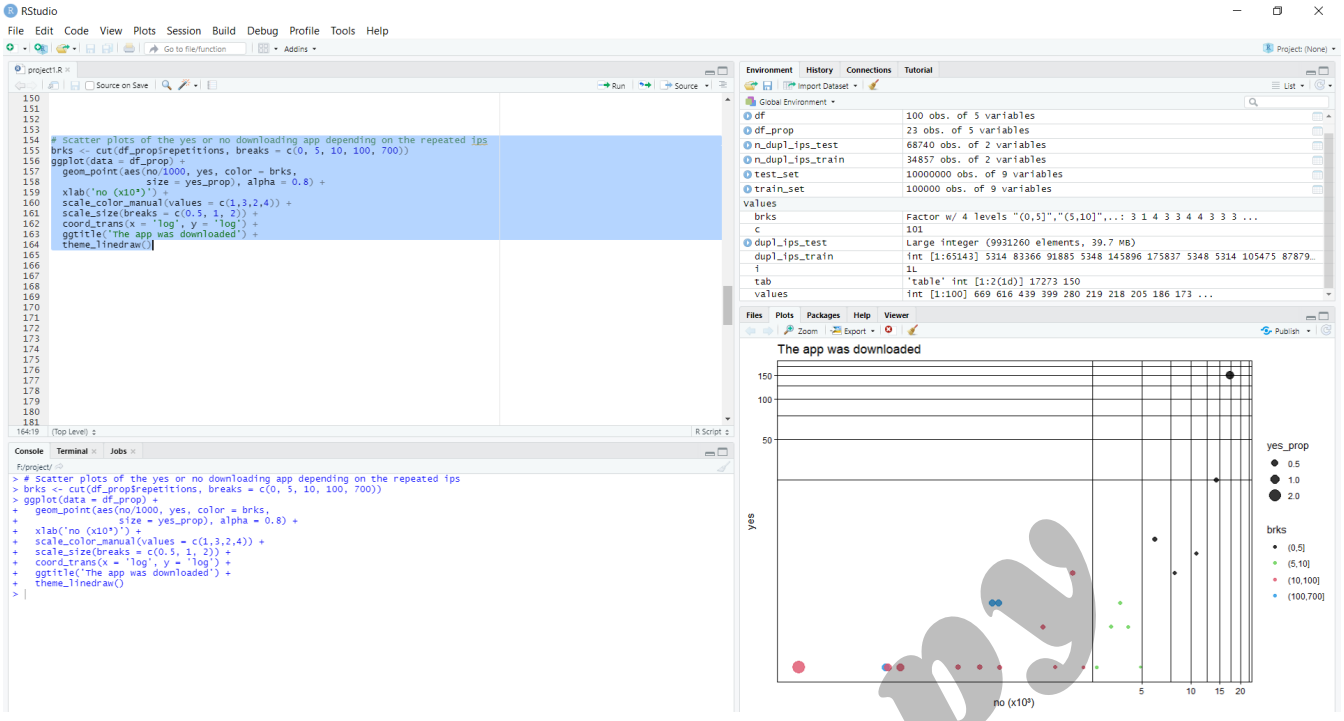
Code:

```r
#Ips analysis
# The number of ips repeated depending on the number of repetitions
c = 1
values <- unique(n_dupl_ips_train$n)
df <- data.frame(repetitions = rep(NA, length(values)))

for (i in values) {
  df$repetitions[c] <- i
  tab <- table(train_set$is_attributed[train_set$repetitions == i])
  df$no[c] <- tab[1]
  df$no_prop[c] <- round(tab[1] * 100 / sum(tab), 2)
  df$yes[c] <- tab[2]
  df$yes_prop[c] <- round(tab[2] * 100 / sum(tab), 2)
  c = c + 1
}

# Verifying the number rows of  train data set is correct
sum(df$no, df$yes)

# Filter and sorting df in relation to the proportion of yes to the app
df_prop <- df %>%
  filter(yes_prop > 0) %>%
  arrange(desc(yes_prop))

df_prop
```

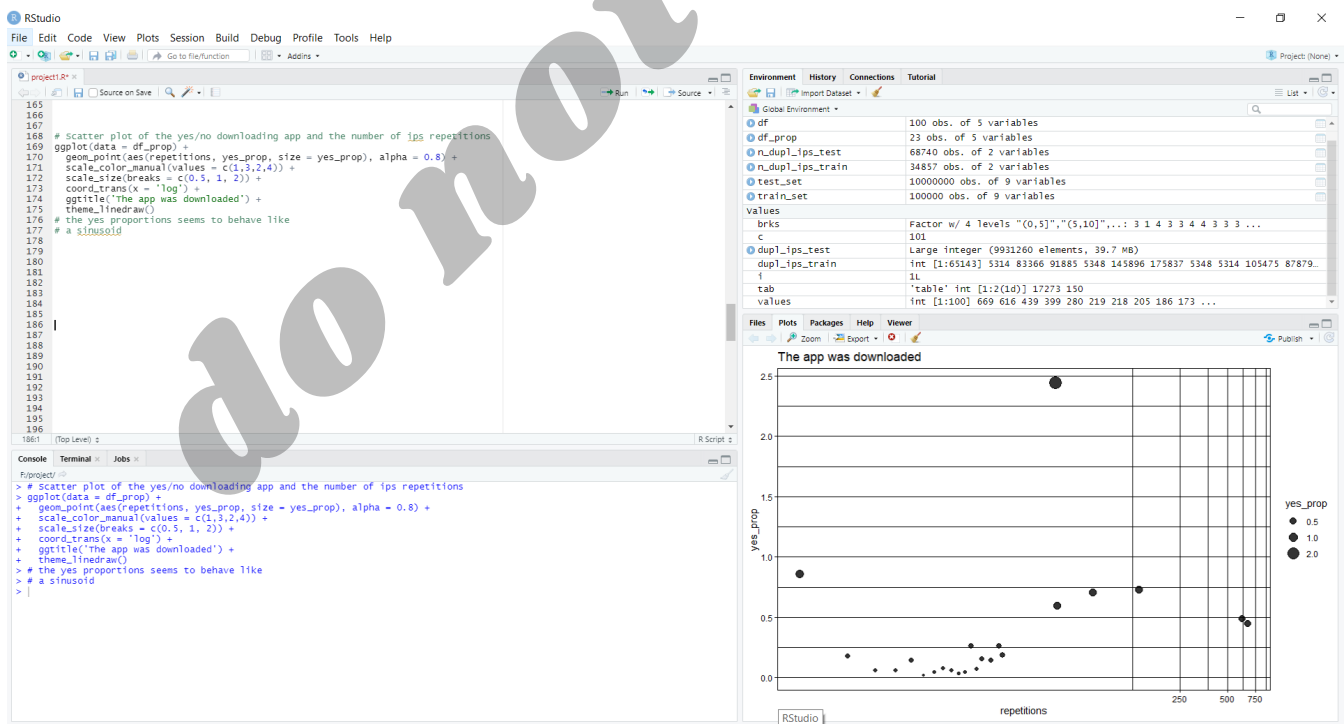
Output:

```
+   c = c + 1
+ }
>
> # Verifying the number rows of  train data set is correct
> sum(df$no, df$yes)
[1] 100000
>
> # Filter and sorting df in relation to the proportion of yes to the app
> df_prop <- df %>%
+   filter(yes_prop > 0) %>%
+   arrange(desc(yes_prop))
>
> df_prop
   repetitions    no no_prop yes yes_prop
1           41    40   97.56   1     2.44
2            1 17273   99.14 150     0.86
3          137   136   99.27   1     0.73
4           70   139   99.29   1     0.71
5           42   167   99.40   1     0.60
6          616   613   99.51   3     0.49
7          669   666   99.55   3     0.45
8           18   377   99.74   1     0.26
9           12  1891   99.74   5     0.26
```

Code:



Output:



Code:

project1.R

```
181
182
183
184 # Iserting another columns according to the yes_prop values
185 gt <- df_prop$repetitions[df_prop$yes_prop > 0.4]
186
187 train_set <- train_set %>%
188   mutate(yes_prop = ifelse(repetitions %in% gt, 1, 0))
189 # I forget that did not have the
190 # is_attributed features in test data set.
191 # I will make classes for repititions
192 # feature.
193
194 # repetitions classes
195 train_set$repetitions_fac <- cut(train_set$repetitions,
196                                  breaks = c(0,5,nrow(train_set)),
197                                  labels = c(1, 2))
198
199 test_set$repetitions_fac <- cut(test_set$repetitions,
200                                 breaks = c(0,5,nrow(test_set)),
201                                 labels = c(1, 2))
202
203
204
205
206
```

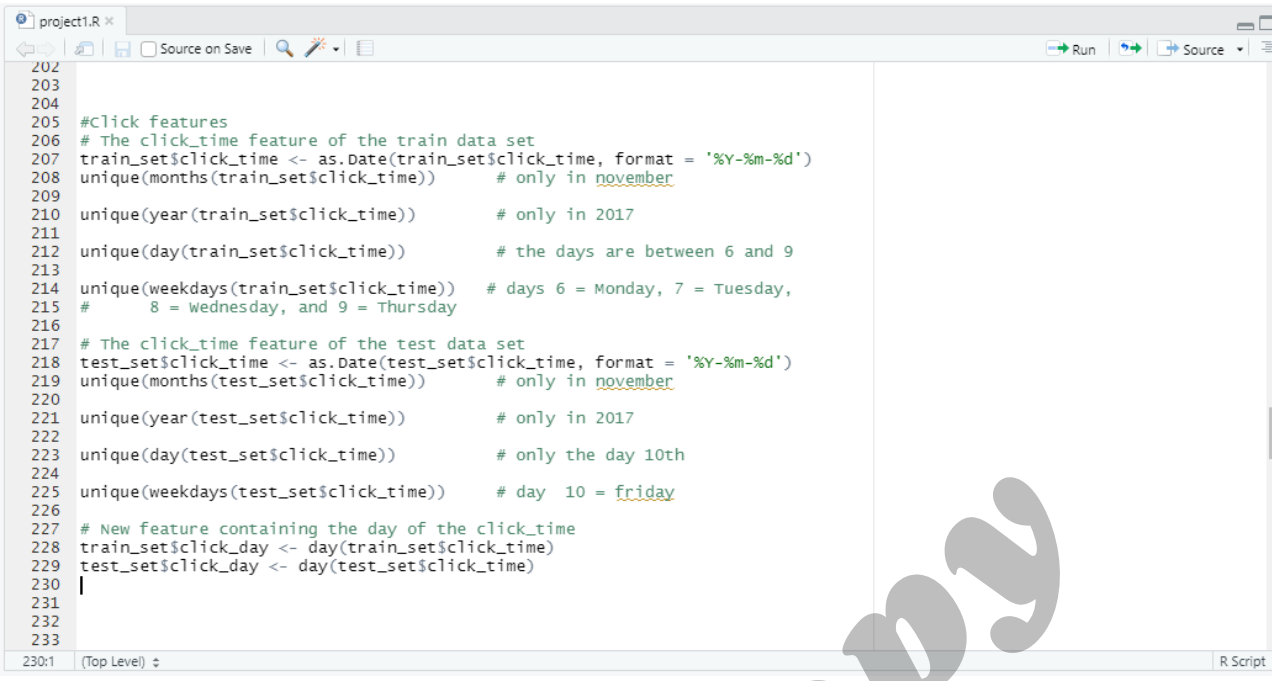
Output:

Console Terminal Jobs

F:/project/

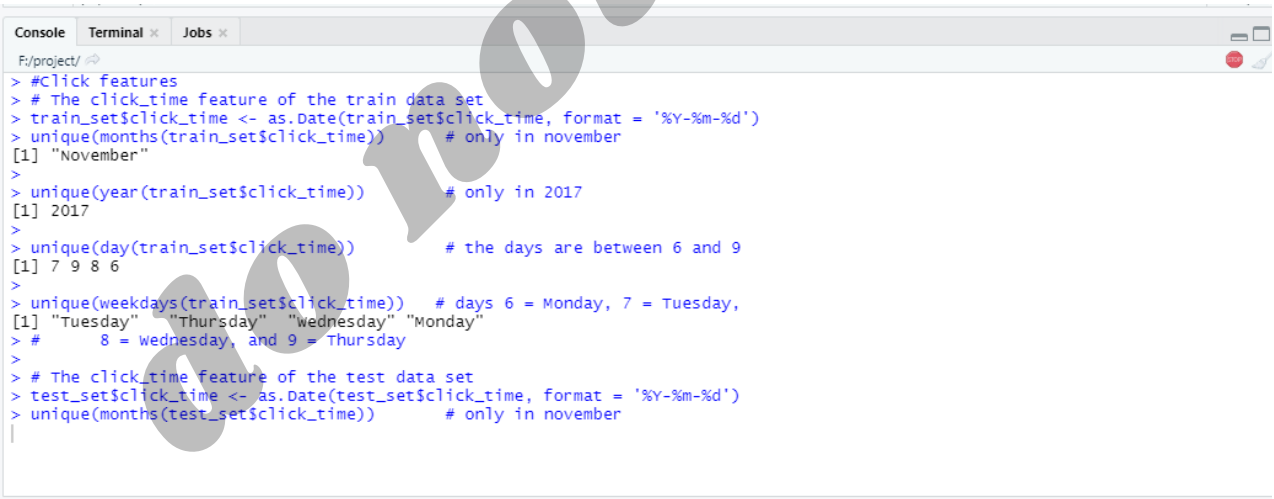
```
> # Iserting anoter columns according to the yes_prop values
> gt <- df_prop$repetitions[df_prop$yes_prop > 0.4]
>
> train_set <- train_set %>%
+   mutate(yes_prop = ifelse(repetitions %in% gt, 1, 0))
> # I forget that did not have the
> # is_attributed features in test data set.
> # I will make classes for repititions
> # feature.
>
> # repetitions classes
> train_set$repetitions_fac <- cut(train_set$repetitions,
+                                  breaks = c(0,5,nrow(train_set)),
+                                  labels = c(1, 2))
>
> test_set$repetitions_fac <- cut(test_set$repetitions,
+                                 breaks = c(0,5,nrow(test_set)),
+                                 labels = c(1, 2))
>
```

Code:



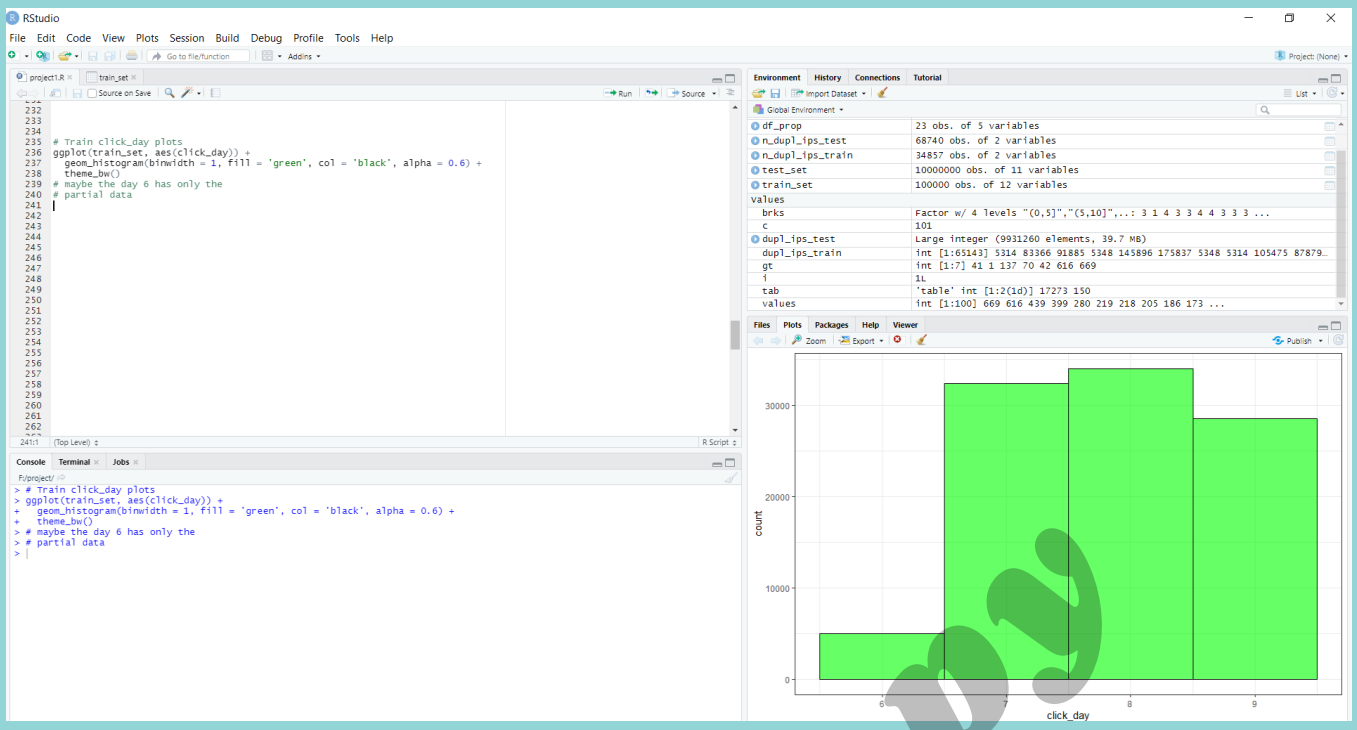
```
#Click features
# The click_time feature of the train data set
train_set$click_time <- as.Date(train_set$click_time, format = '%Y-%m-%d')
unique(months(train_set$click_time))      # only in november

unique(year(train_set$click_time))        # only in 2017

unique(day(train_set$click_time))         # the days are between 6 and 9

unique(weekdays(train_set$click_time))    # days 6 = Monday, 7 = Tuesday,
#       8 = Wednesday, and 9 = Thursday

# The click_time feature of the test data set
test_set$click_time <- as.Date(test_set$click_time, format = '%Y-%m-%d')
unique(months(test_set$click_time))       # only in november

unique(year(test_set$click_time))         # only in 2017

unique(day(test_set$click_time))          # only the day 10th

unique(weekdays(test_set$click_time))     # day  10 = friday

# New feature containing the day of the click_time
train_set$click_day <- day(train_set$click_time)
test_set$click_day <- day(test_set$click_time)
```

Output:



```
> #Click features
> # The click_time feature of the train data set
> train_set$click_time <- as.Date(train_set$click_time, format = '%Y-%m-%d')
> unique(months(train_set$click_time))      # only in november
[1] "November"
>
> unique(year(train_set$click_time))        # only in 2017
[1] 2017
>
> unique(day(train_set$click_time))         # the days are between 6 and 9
[1] 7 9 8 6
>
> unique(weekdays(train_set$click_time))    # days 6 = Monday, 7 = Tuesday,
[1] "Tuesday"   "Thursday"  "Wednesday" "Monday"
> #       8 = Wednesday, and 9 = Thursday
>
> # The click_time feature of the test data set
> test_set$click_time <- as.Date(test_set$click_time, format = '%Y-%m-%d')
> unique(months(test_set$click_time))       # only in november
```

Code:



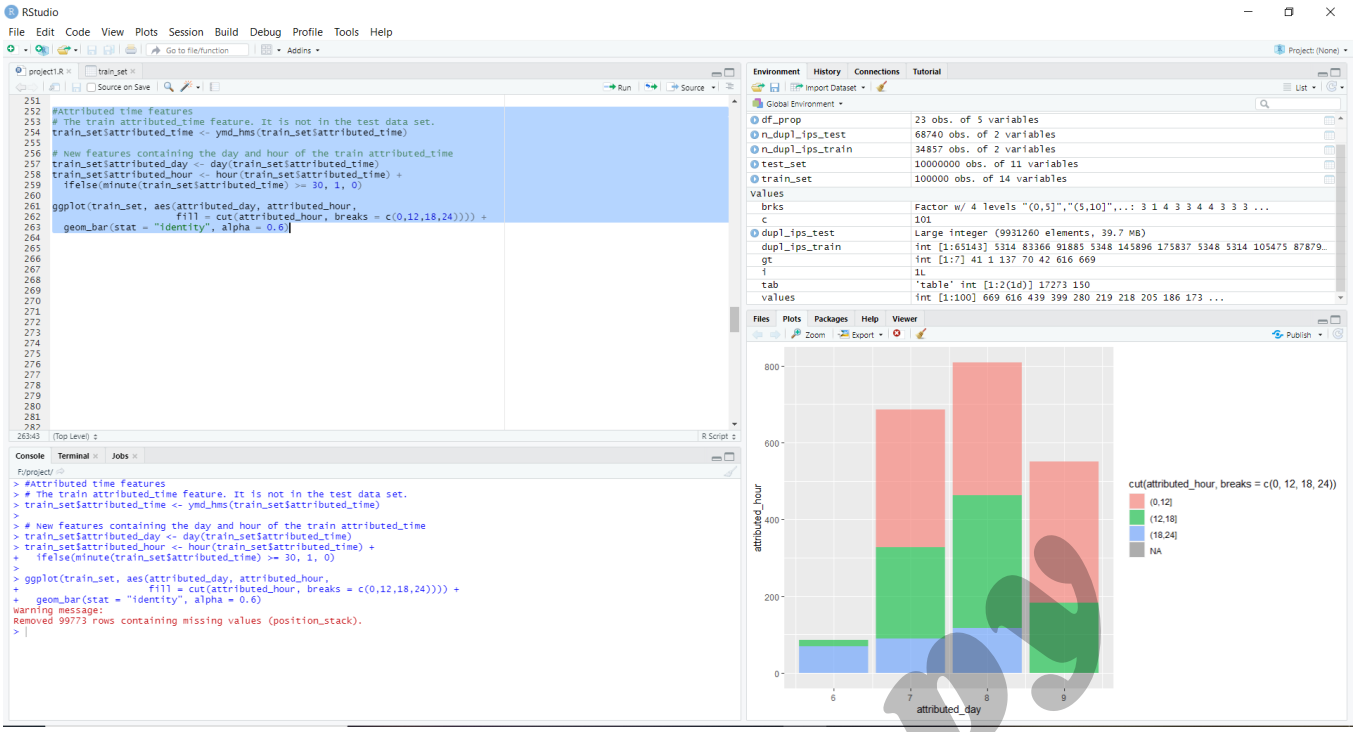
Code:

```
242
243  # Number of clicks in function of the day (train)
244  train_set %>%
245    count(click_day)
246
```

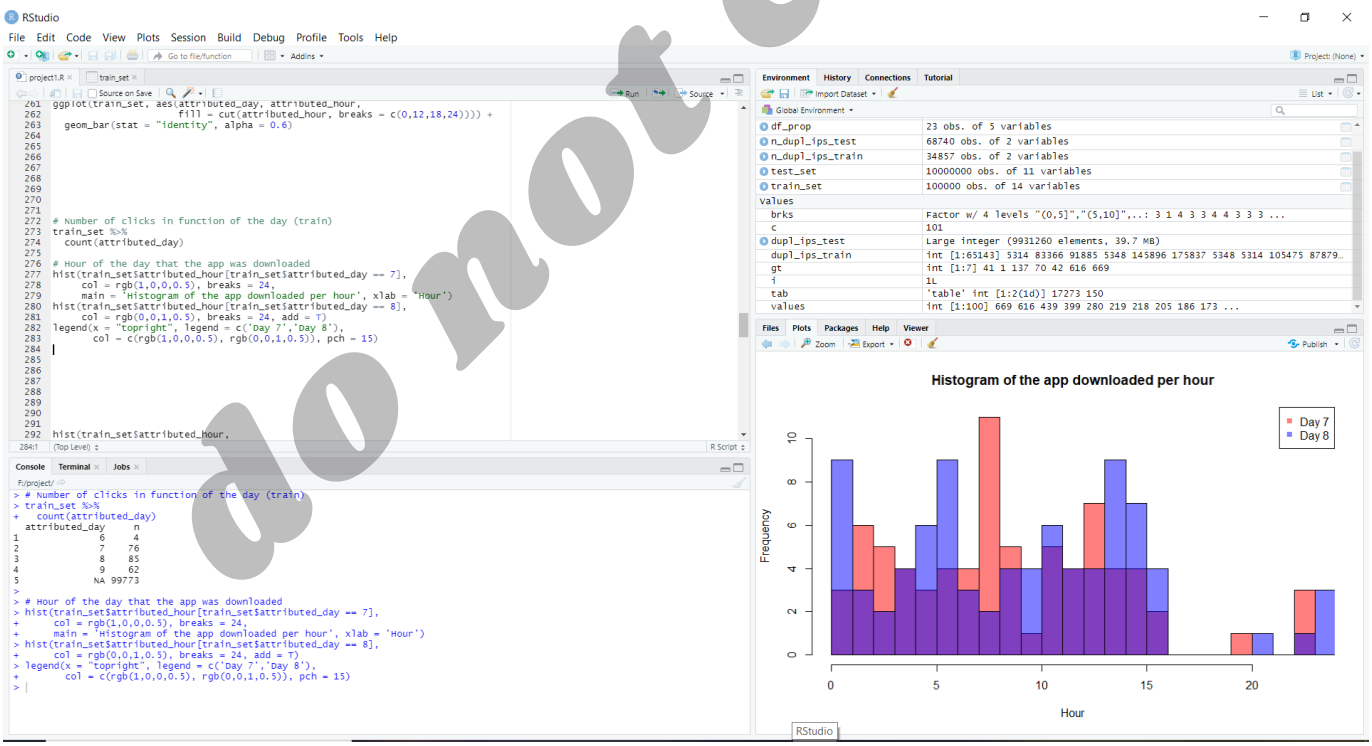
Output:

```
Console  Terminal  Jobs
F:/project/
> # Number of clicks in function of the day (train)
> train_set %>%
+   count(click_day)
  click_day     n
1         6  5011
2         7 32393
3         8 34035
4         9 28561
>
```

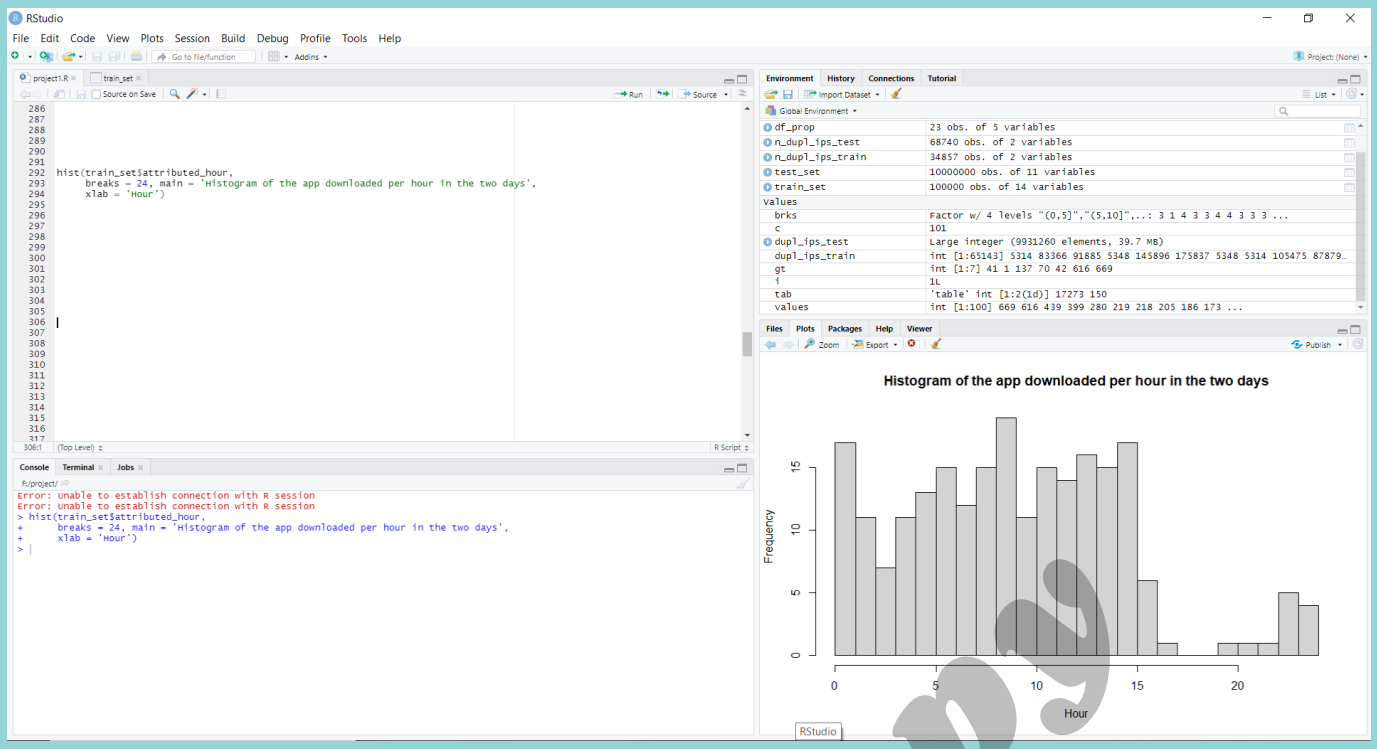
Code/Output:



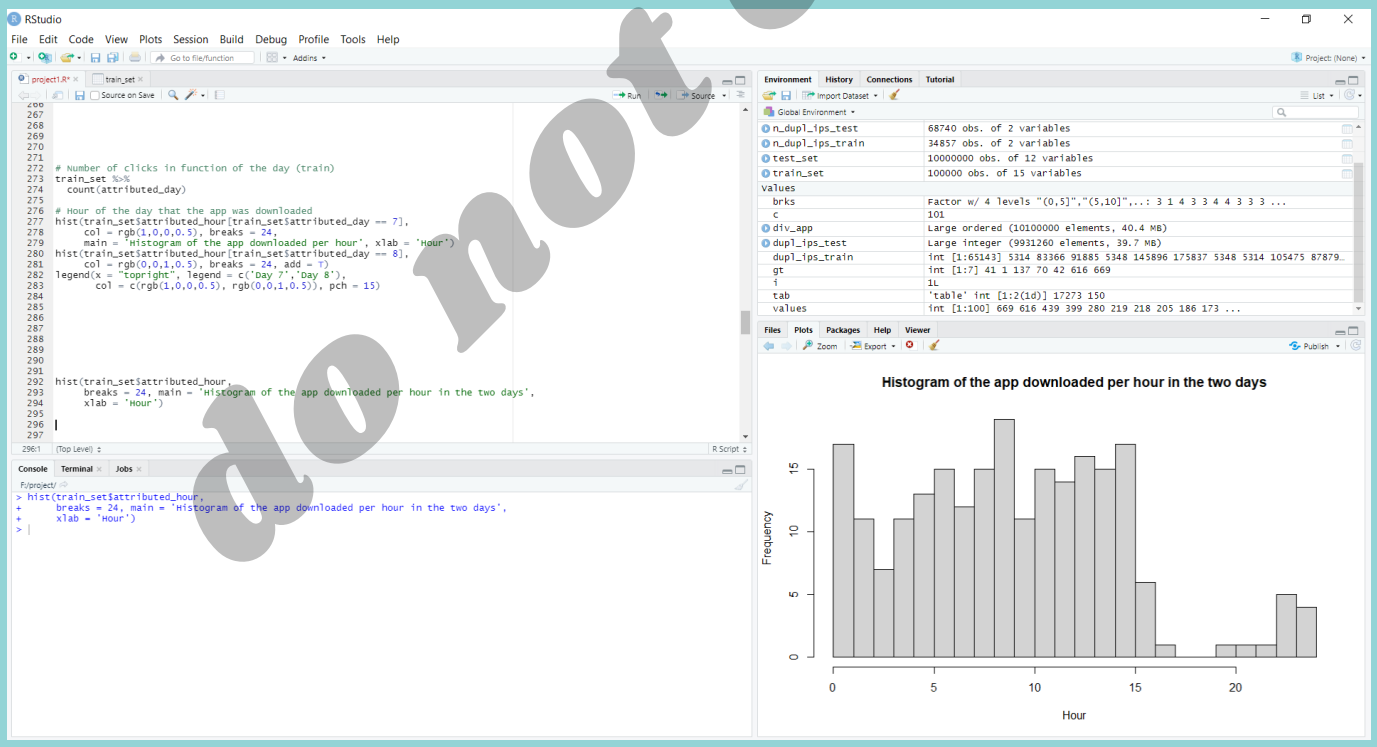
Code:



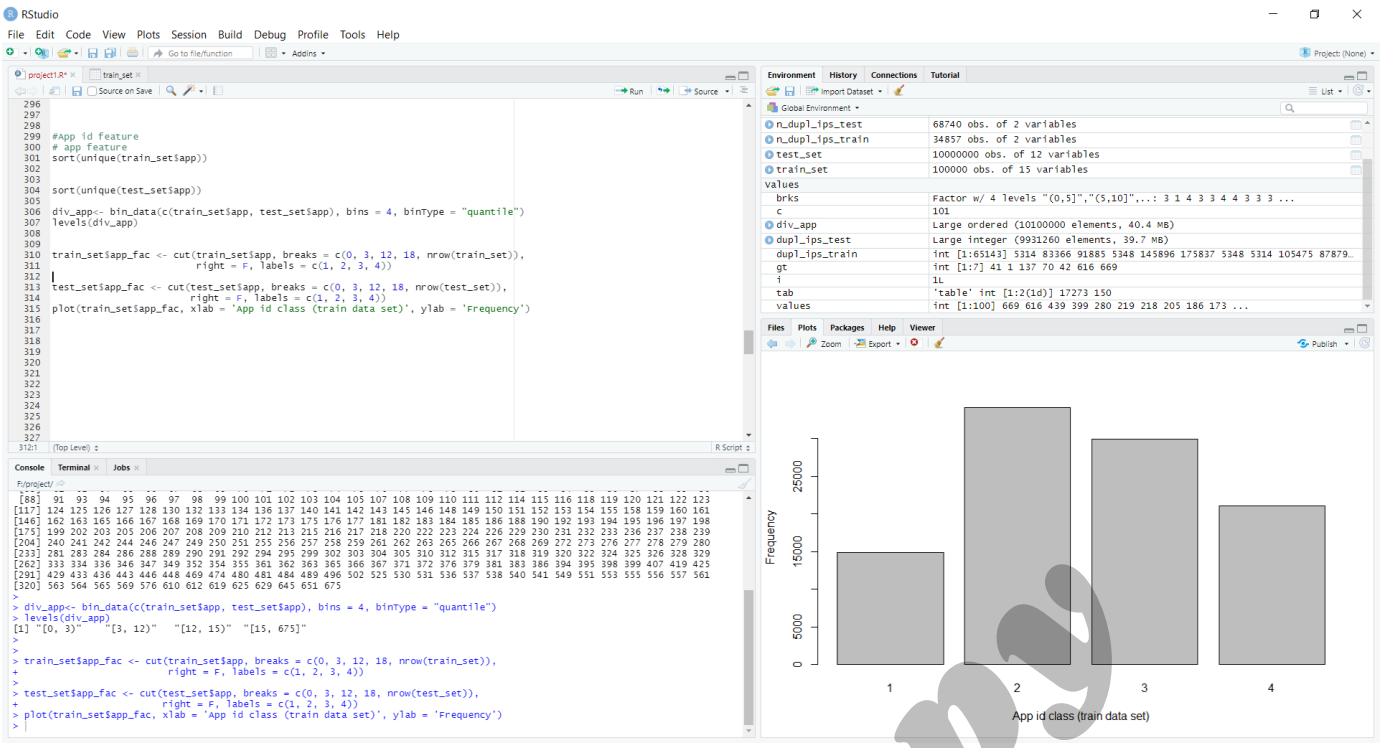
Output:



Code:

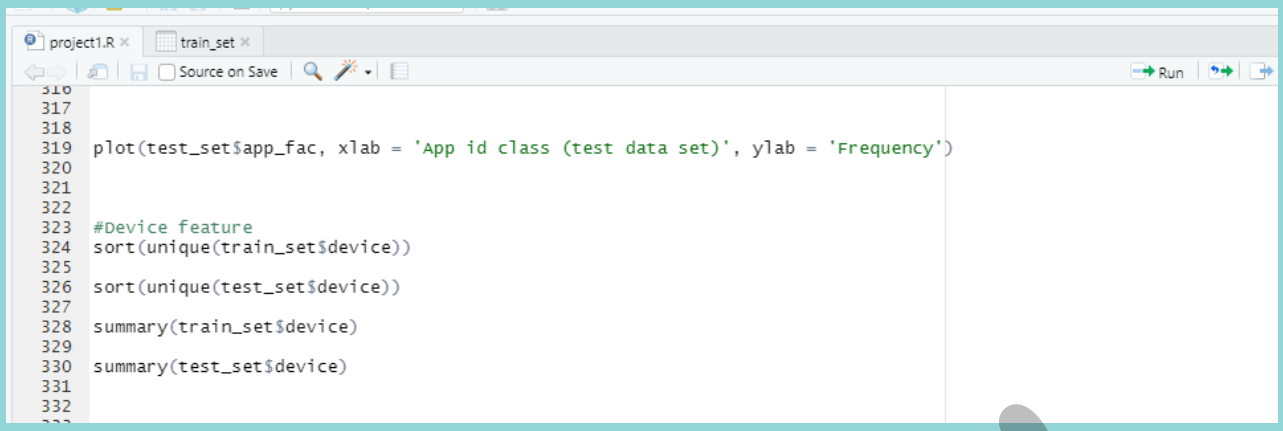


Code:



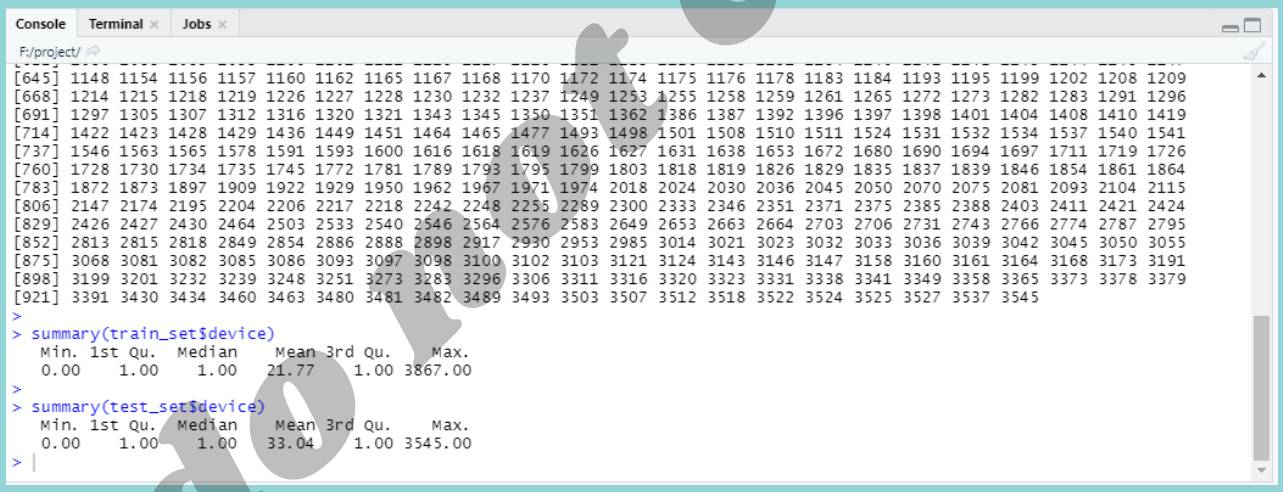
Code:

Code:



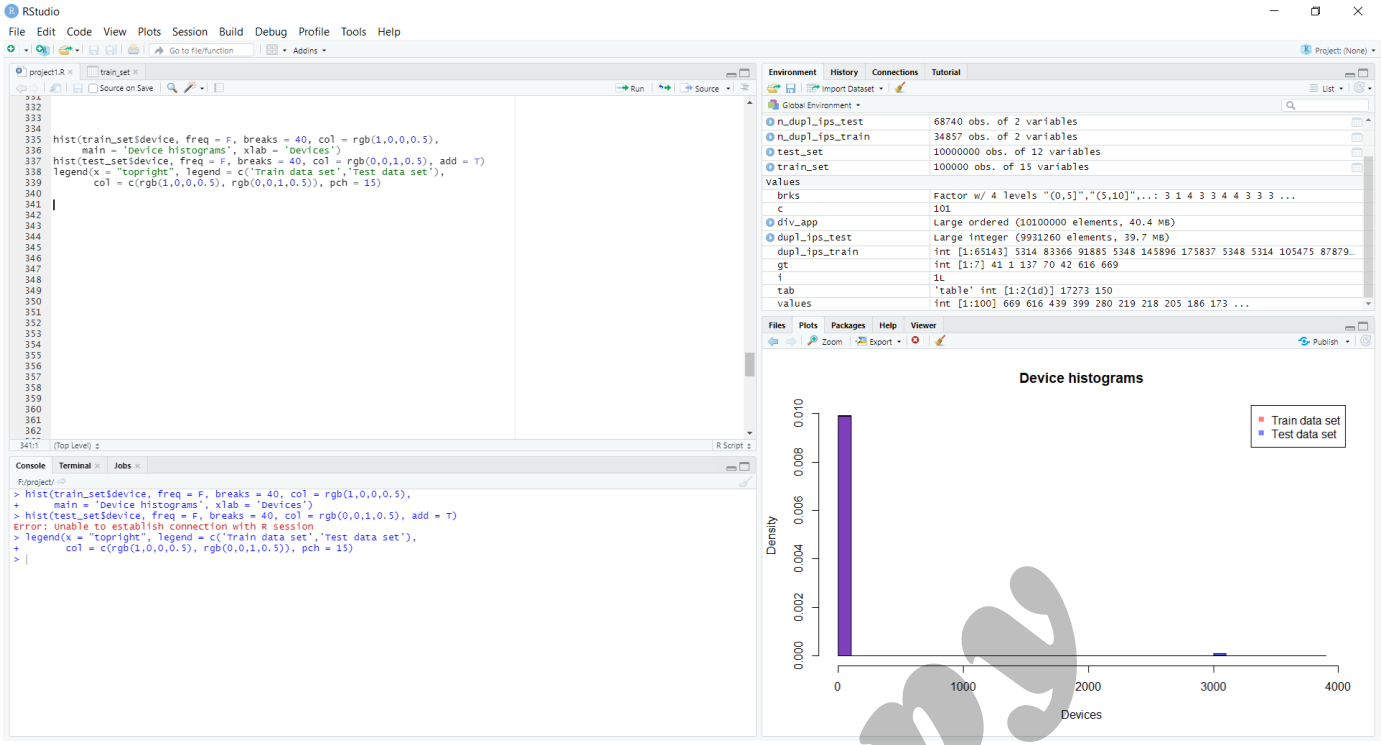
```
plot(test_set$app_fac, xlab = 'App id class (test data set)', ylab = 'Frequency')


#Device feature
sort(unique(train_set$device))

sort(unique(test_set$device))

summary(train_set$device)

summary(test_set$device)
```

Output:



```
[645] 1148 1154 1156 1157 1160 1162 1165 1167 1168 1170 1172 1174 1175 1176 1178 1183 1184 1193 1195 1199 1202 1208 1209
[668] 1214 1215 1218 1219 1226 1227 1228 1230 1232 1237 1249 1253 1255 1258 1259 1261 1265 1272 1273 1282 1283 1291 1296
[691] 1297 1305 1307 1312 1316 1320 1321 1343 1345 1350 1351 1362 1386 1387 1392 1396 1397 1398 1401 1404 1408 1410 1419
[714] 1422 1423 1428 1429 1436 1449 1451 1464 1465 1477 1493 1498 1501 1508 1510 1511 1524 1531 1532 1534 1537 1540 1541
[737] 1546 1563 1565 1578 1591 1593 1600 1616 1618 1619 1626 1627 1631 1638 1653 1672 1680 1690 1694 1697 1711 1719 1726
[760] 1728 1730 1734 1735 1745 1772 1781 1789 1793 1795 1799 1803 1818 1819 1826 1829 1835 1837 1839 1846 1854 1861 1864
[783] 1872 1873 1897 1909 1922 1929 1950 1962 1967 1971 1974 2018 2024 2030 2036 2045 2050 2070 2075 2081 2093 2104 2115
[806] 2147 2174 2195 2204 2206 2217 2218 2242 2248 2255 2289 2300 2333 2346 2351 2371 2375 2385 2388 2403 2411 2421 2424
[829] 2426 2427 2430 2464 2503 2533 2540 2546 2564 2576 2583 2649 2653 2663 2664 2703 2706 2731 2743 2766 2774 2787 2795
[852] 2813 2815 2818 2849 2854 2886 2888 2898 2917 2930 2953 2985 3014 3021 3023 3032 3033 3036 3039 3042 3045 3050 3055
[875] 3068 3081 3082 3085 3086 3093 3097 3098 3100 3102 3103 3121 3124 3143 3146 3147 3158 3160 3161 3164 3168 3173 3191
[898] 3199 3201 3232 3239 3248 3251 3273 3283 3296 3306 3311 3316 3320 3323 3331 3338 3341 3349 3358 3365 3373 3378 3379
[921] 3391 3430 3434 3460 3463 3480 3481 3482 3489 3493 3503 3507 3512 3518 3522 3524 3525 3527 3537 3545

> summary(train_set$device)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    1.00    1.00   21.77    1.00 3867.00
>
> summary(test_set$device)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    1.00    1.00   33.04    1.00 3545.00
>
```

Code:



Code:

```
a <- train_set %>%
  count(device, sort = T)
head(a)


b <- test_set %>%
  count(device, sort = T)
head(b)

# Type 1 device proportion
( a[1,2]/sum(a) )

( b[1,2]/sum(b) )

# Making two classes of devices: one for type 1 and the other for the others
class_device <- function(x) {ifelse(x == 1, 1, 2)}

train_set$device_fac <- as.factor(class_device(train_set$device))
levels(train_set$device_fac)

test_set$device_fac <- as.factor(class_device(test_set$device))
levels(test_set$device_fac)
```

Code:

```
372
373 #OS feature
374 sort(unique(train_set$os))
375
376
377 sort(unique(test_set$os))
378
379 summary(train_set$os)
380
381 summary(test_set$os)
382
383
384
```

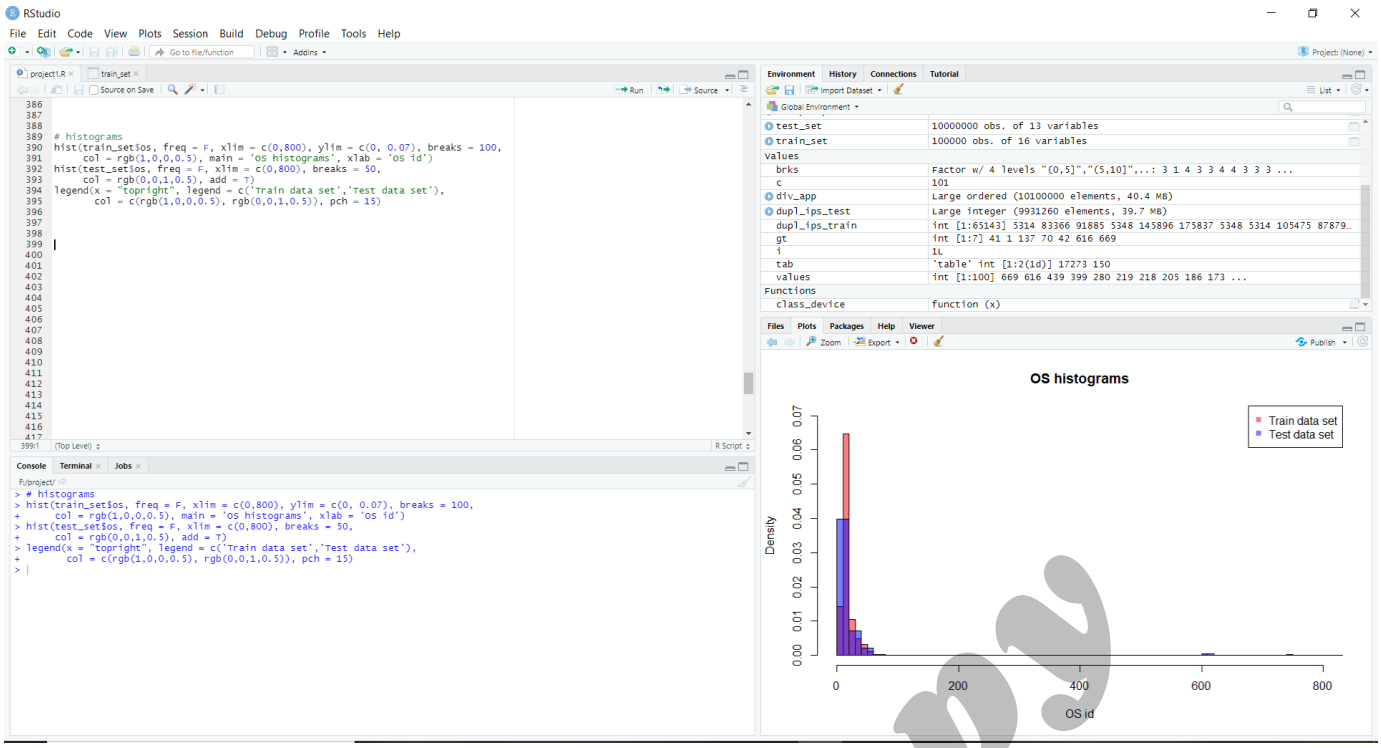
Output

```
> summary(train_set$os)
  [1]   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28
 [30]  29  30  31  32  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  52  53  54  55  56  57  58  59
 [59]  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  83  84  85  86  87  88  89
 [88]  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 117 118 119
[117] 120 123 124 125 126 127 128 129 130 132 133 134 135 136 137 138 140 141 142 143 145 146 147 148 149 150 151 152 153
[146] 155 156 158 159 160 161 162 164 168 169 171 172 173 174 175 177 178 183 184 185 188 190 192 193 196 197 198 207 208
[175] 209 213 214 215 216 217 219 223 226 228 229 231 234 236 237 241 243 244 245 248 250 251 252 254 255 256 260 261 262
[204] 265 268 272 274 277 280 284 286 294 297 300 302 305 306 325 326 329 336 338 342 346 355 380 404 407 408 411 414 421
[233] 438 465 505 508 512 514 531 541 552 559 566 573 584 602 603 607 610 612 616 617 619 620 622 630 636 640 645 647 649
[262] 651 653 656 657 669 672 675 681 684 686 687 688 690 692 700 701 702 704 705 707 712 715 716 726 736 737 739 742 743
[291] 744 745
>
> summary(train_set$os)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   13.00   18.00   22.82   19.00  866.00
>
> summary(test_set$os)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0    13.0    18.0    24.6    19.0   745.0
>
```

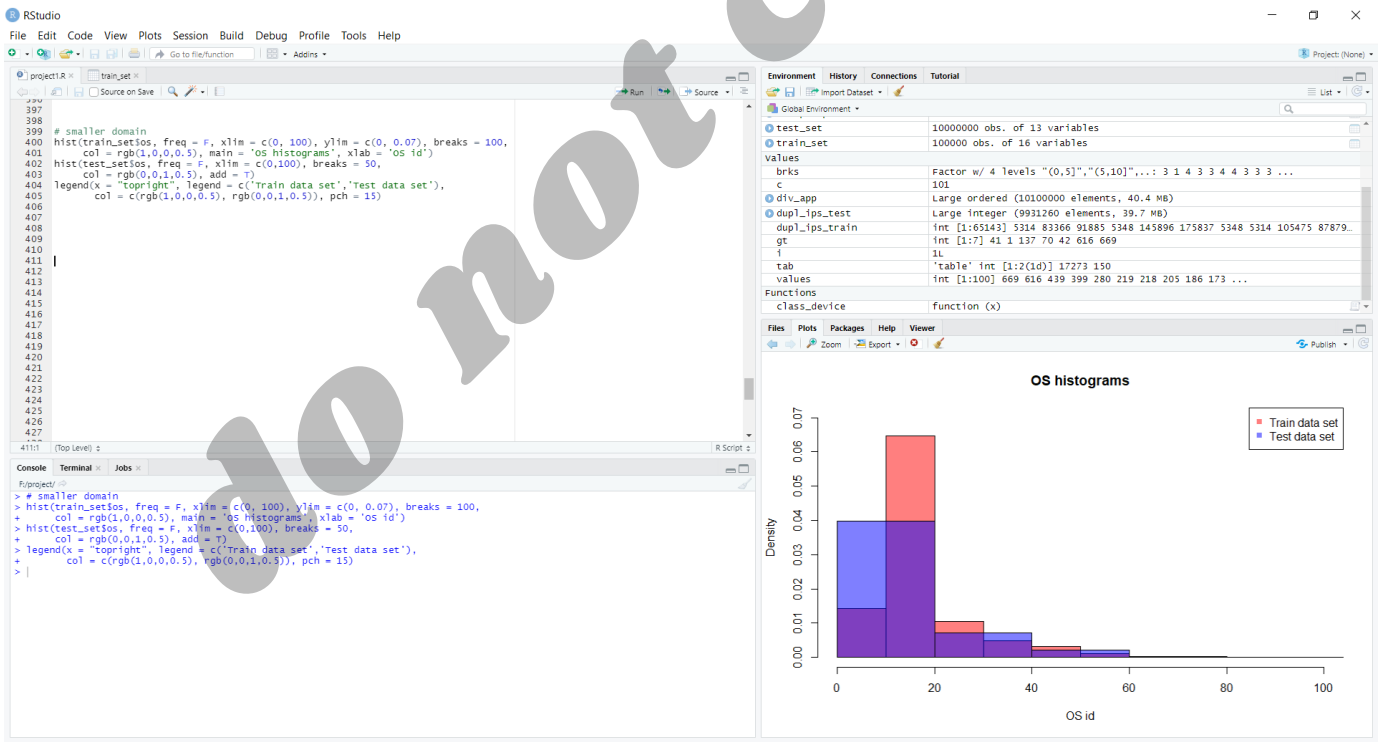
Code:

```
386
387
388
389 # histograms
390 hist(train_set$os, freq = F, xlim = c(0,800), ylim = c(0, 0.07), breaks = 100,
391      col = rgb(1,0,0,0.5), main = 'OS histograms', xlab = 'OS id')
392 hist(test_set$os, freq = F, xlim = c(0,800), breaks = 50,
393      col = rgb(0,0,1,0.5), add = T)
394 legend(x = "topright", legend = c('Train data set','Test data set'),
395        col = c(rgb(1,0,0,0.5), rgb(0,0,1,0.5)), pch = 15)
396
397
398
399
400
401
402
```

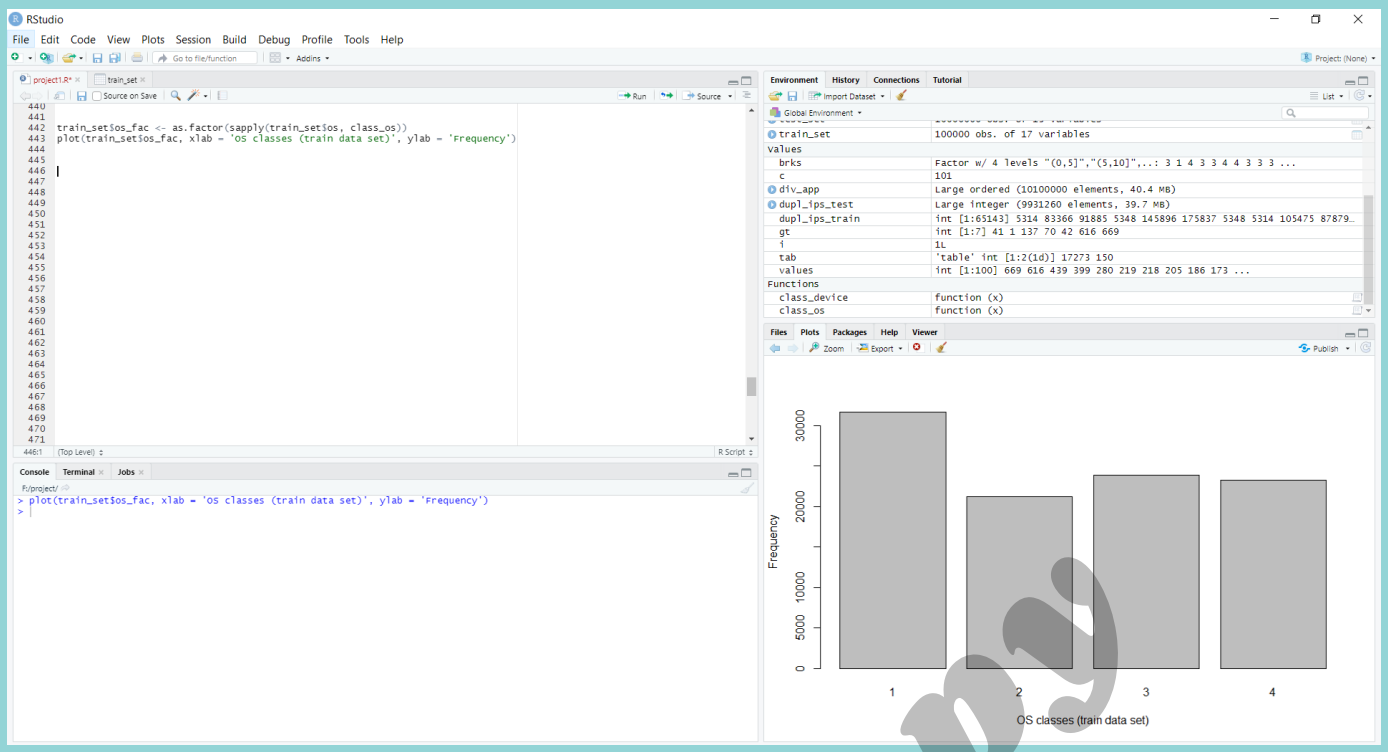
Code:



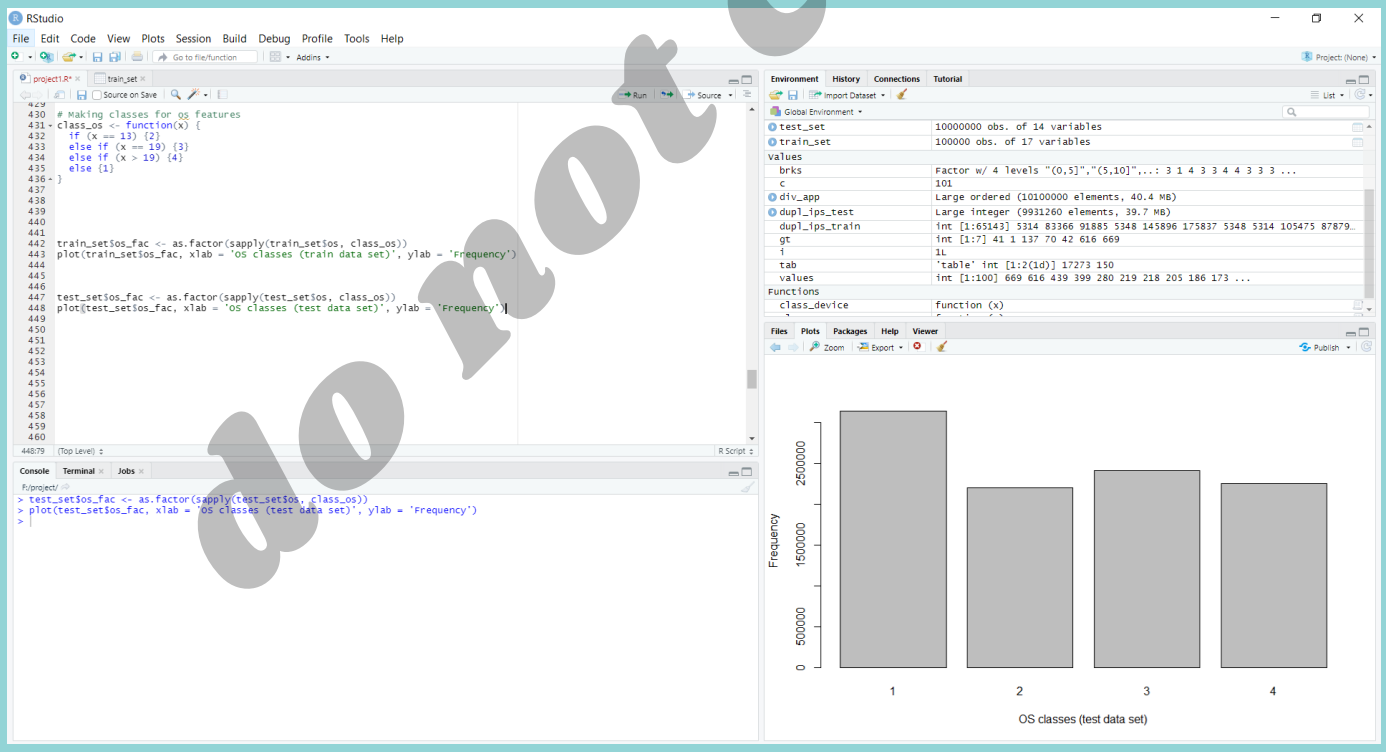
Code:



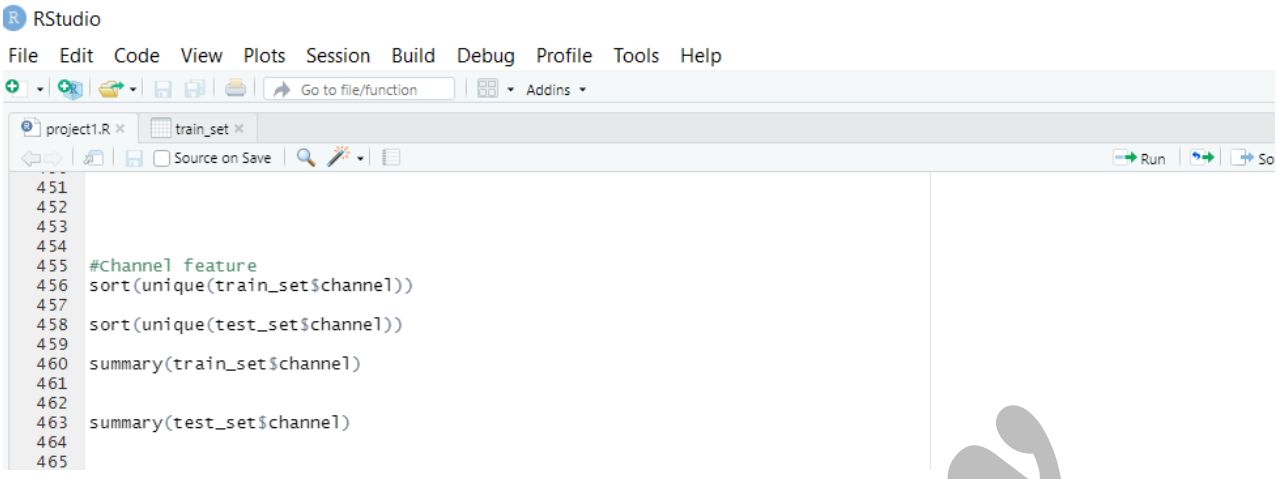
Code:



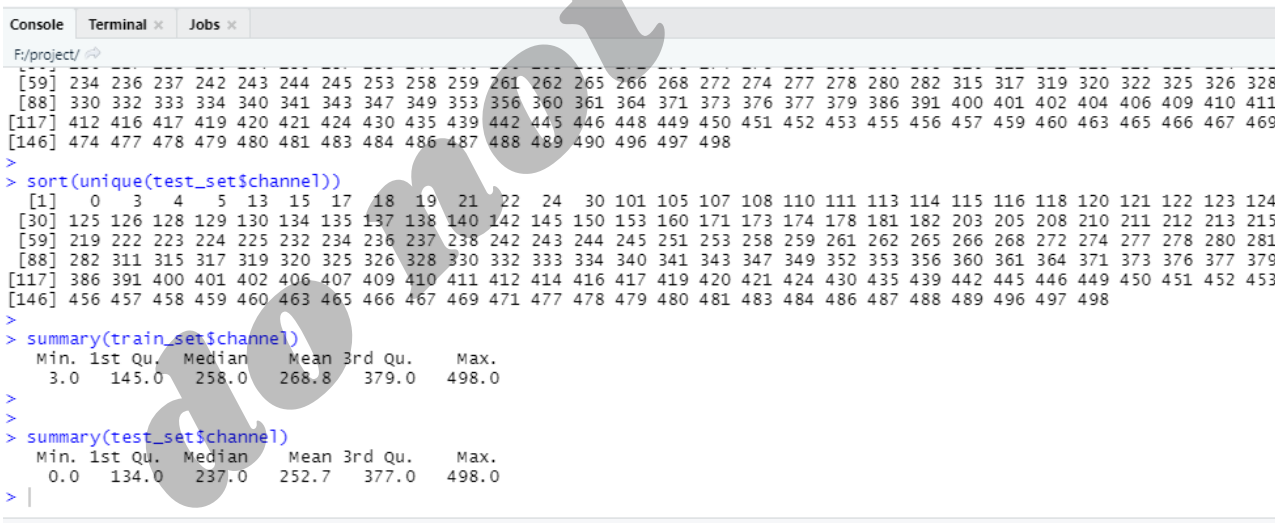
Code:



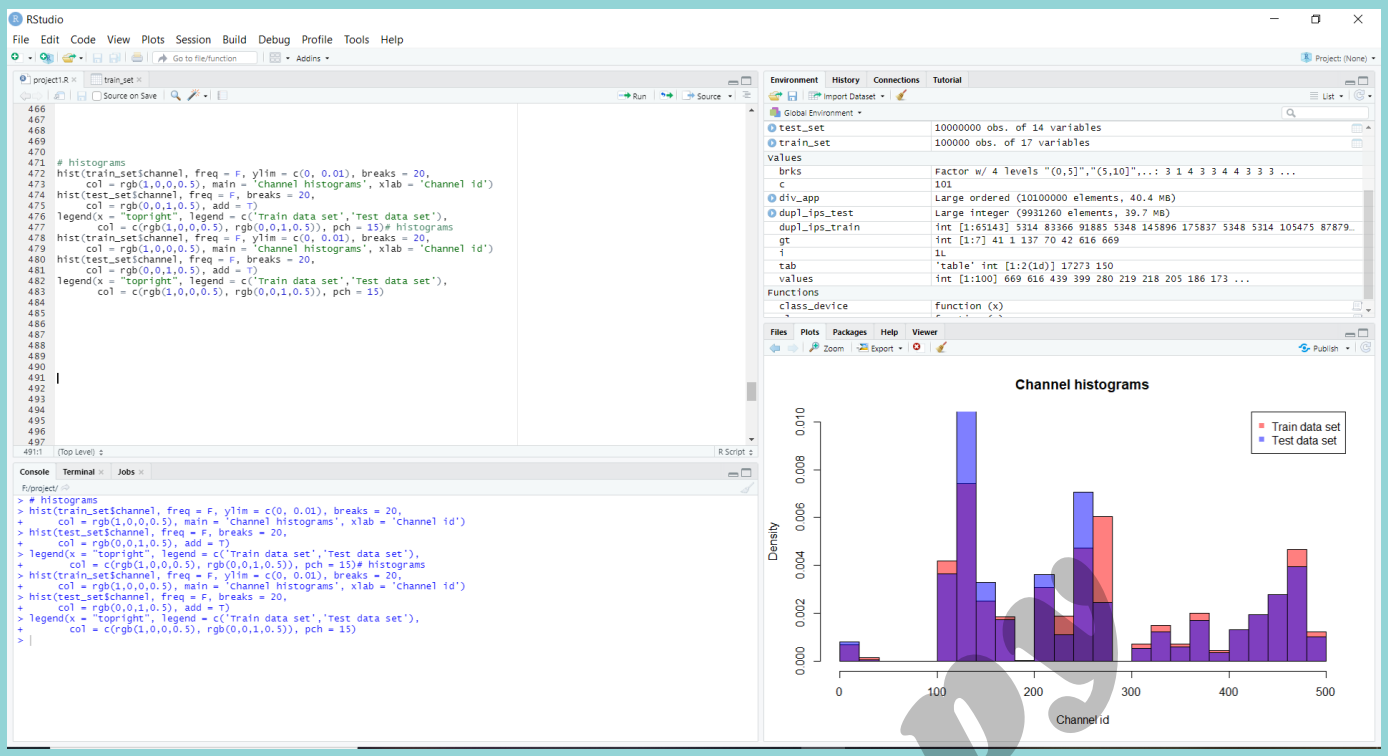
Code:



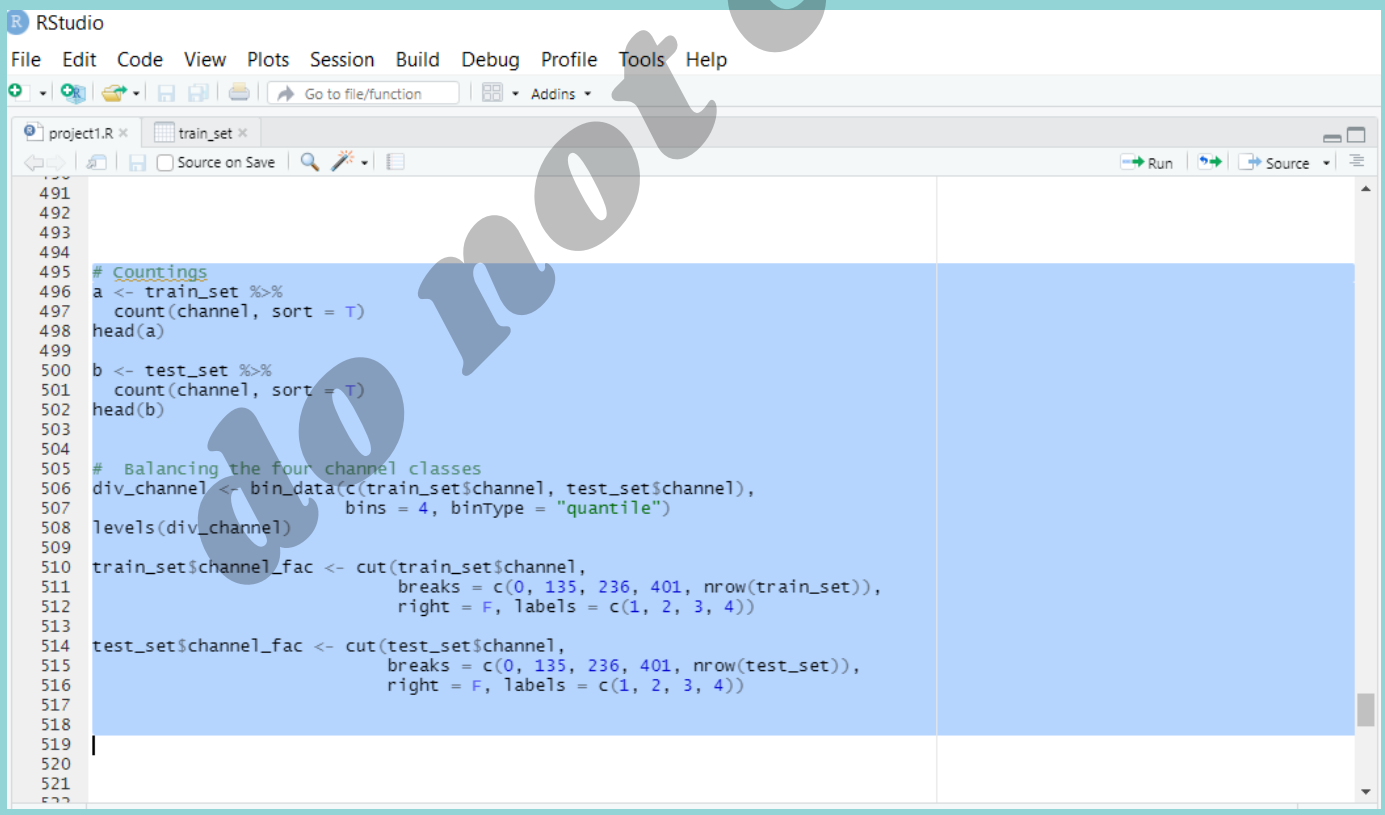
Output:



Code:



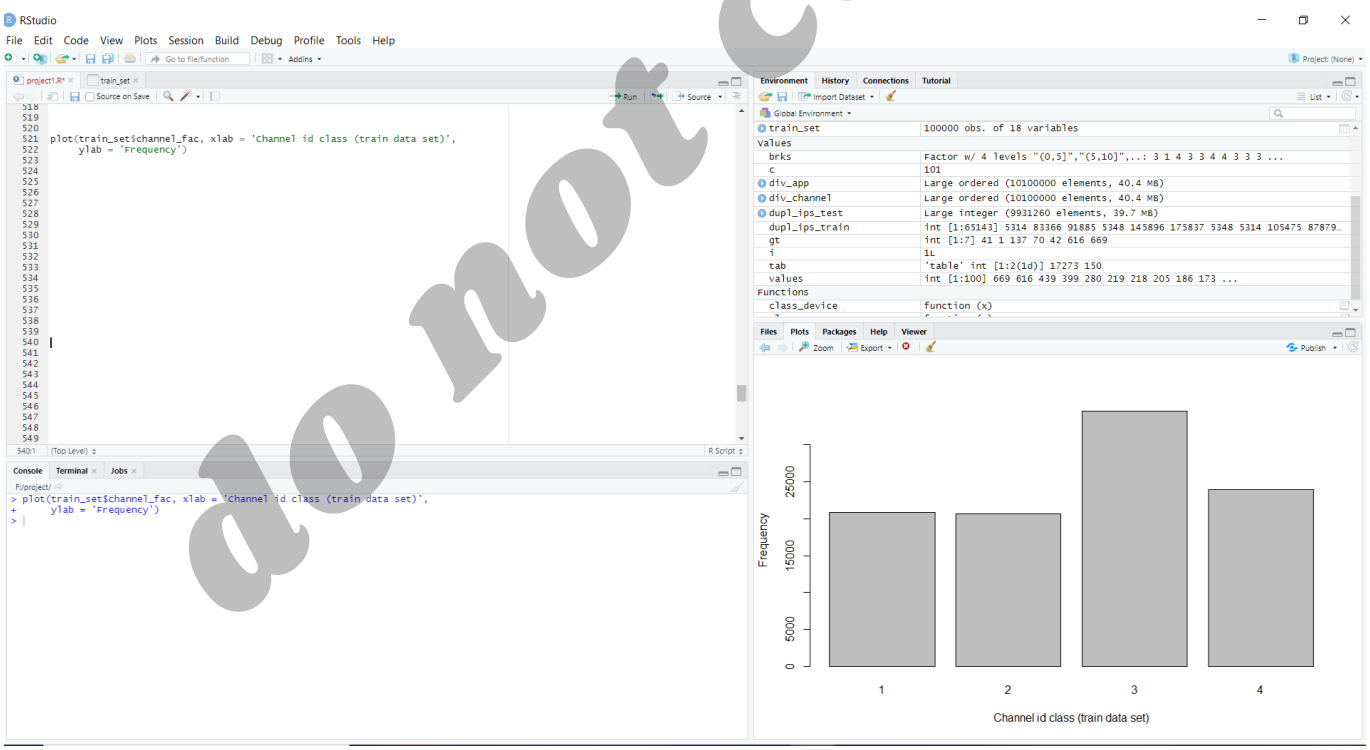
Code



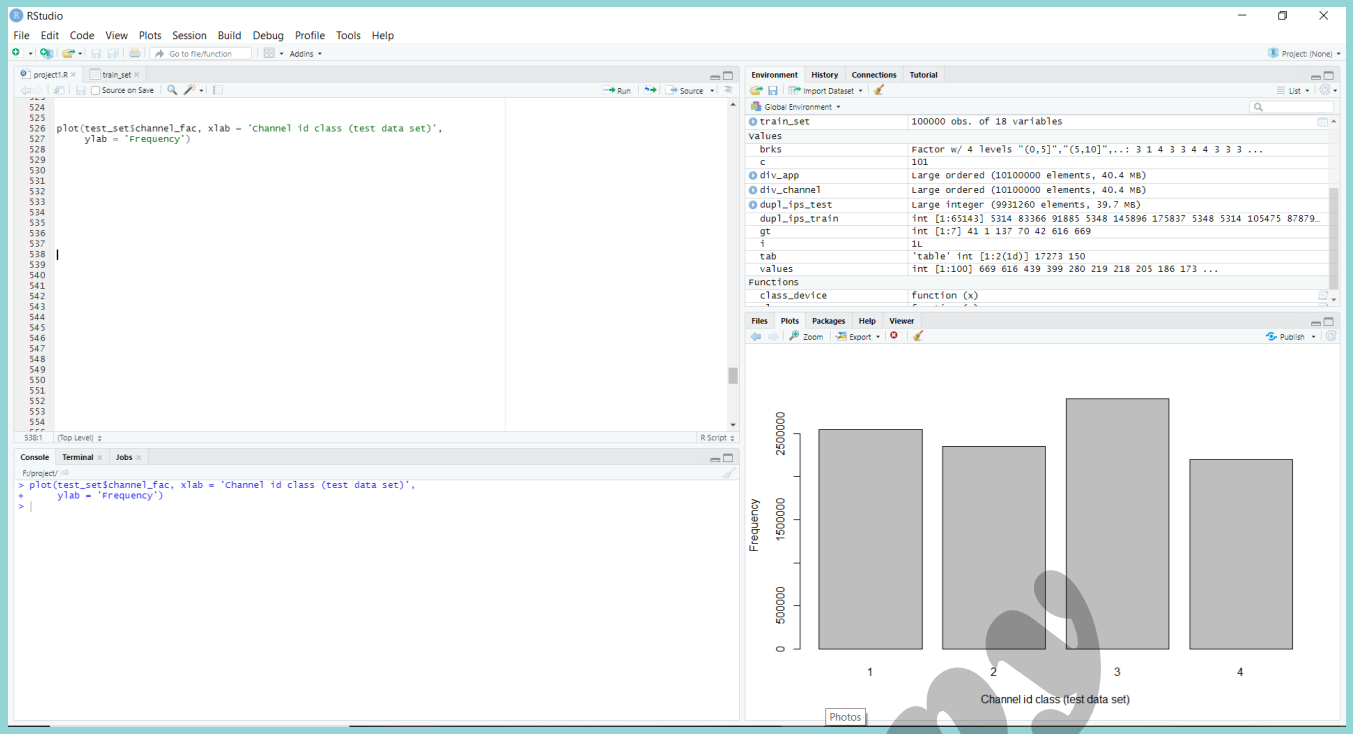
Output:

```
Console  Terminal  Jobs
F:/project/
1:      245 793105
2:      134 630888
3:      259 469845
4:      477 412559
5:      121 402226
6:      107 388035
>
>
> #  Balancing the four channel classes
> div_channel <- bin_data(c(train_set$channel, test_set$channel),
+                         bins = 4, binType = "quantile")
> levels(div_channel)
[1] "[0, 134)"   "[134, 242)" "[242, 377)" "[377, 498]"
>
> train_set$channel_fac <- cut(train_set$channel,
+                              breaks = c(0, 135, 236, 401, nrow(train_set)),
+                              right = F, labels = c(1, 2, 3, 4))
>
> test_set$channel_fac <- cut(test_set$channel,
+                             breaks = c(0, 135, 236, 401, nrow(test_set)),
+                             right = F, labels = c(1, 2, 3, 4))
>
```

Code:

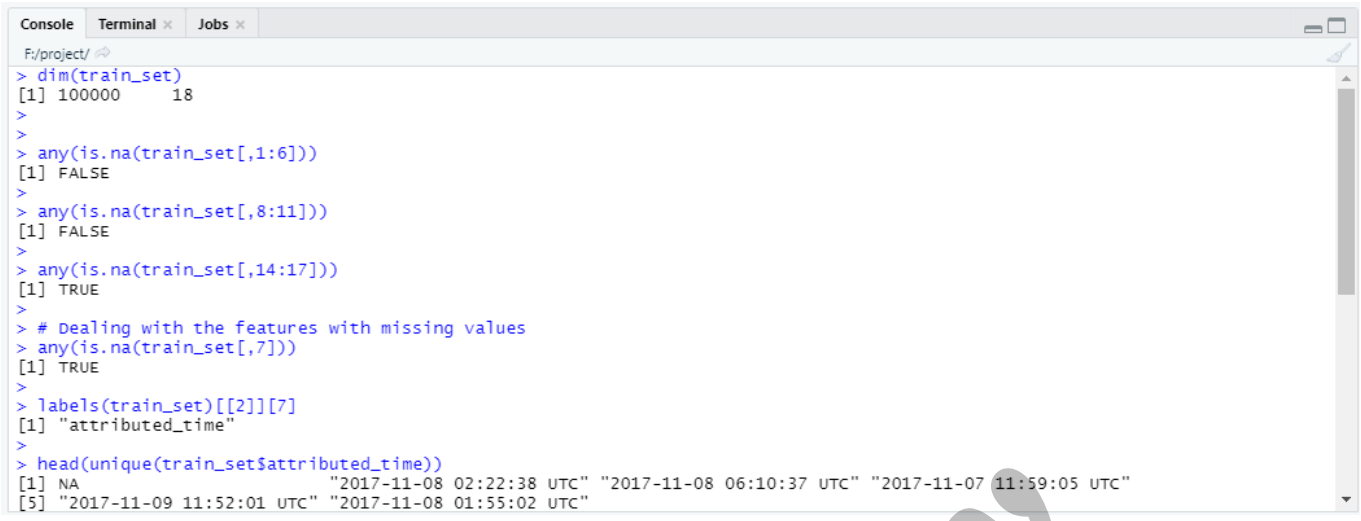


Code:

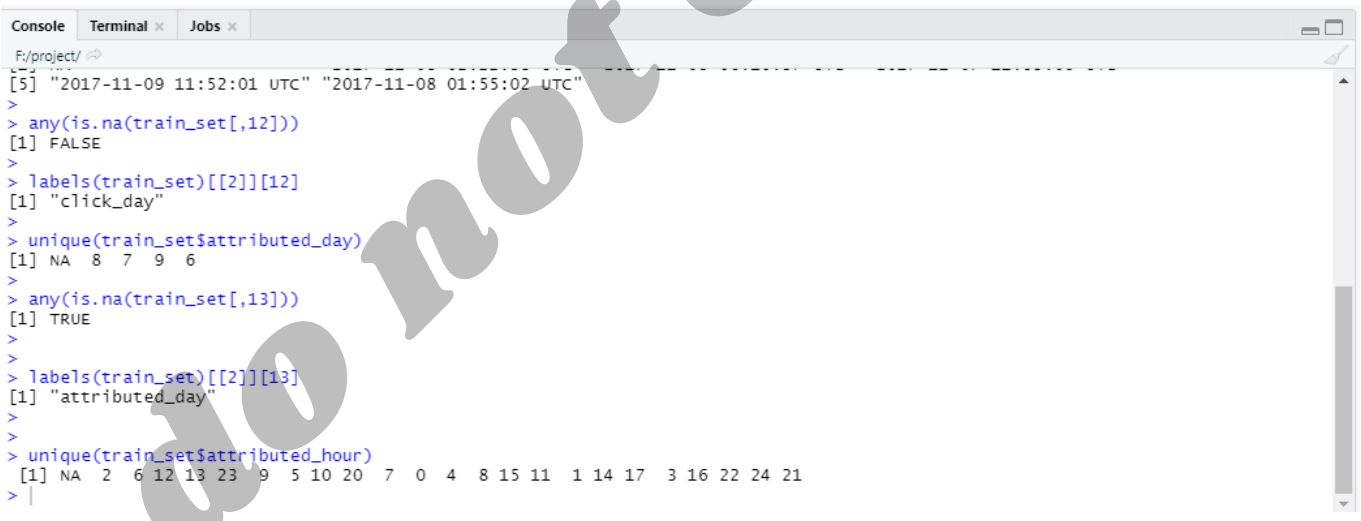


Code

Output:



Output:



Code:

```
#Reducing the quantity of not downloaded to balance the train target feature

n <- nrow(train_set[train_set$is_attributed == 1, ])
n

train_no <- train_set %>%
  filter(is_attributed == 0) %>%
  slice_sample(n = n, replace = F)
nrow(train_no)


train_yes <- train_set %>%
  filter(is_attributed == 1)
nrow(train_yes)

train_set1 <- rbind(train_no, train_yes)
train_set1 <- train_set1 %>%
  slice_sample(n = nrow(train_set1), replace = F)
nrow(train_set1)/2


# Cleaning the house
rm(list = setdiff(ls(), c('train_set', 'train_set1', 'test_set')))
ls()

gc()
```

Output:

```
> #Reducing the quantity of not downloaded to balance the train target feature
>
> n <- nrow(train_set[train_set$is_attributed == 1, ])
> n
[1] 227
>
> train_no <- train_set %>%
+   filter(is_attributed == 0) %>%
+   slice_sample(n = n, replace = F)
> nrow(train_no)
[1] 227
>
>
> train_yes <- train_set %>%
+   filter(is_attributed == 1)
> nrow(train_yes)
[1] 227
>
> train_set1 <- rbind(train_no, train_yes)
> train_set1 <- train_set1 %>%
+   slice_sample(n = nrow(train_set1), replace = F)
> nrow(train_set1)/2
[1] 227
>
> # Cleaning the house
> rm(list = setdiff(ls(), c('train_set', 'train_set1', 'test_set')))
> ls()
[1] "test_set"   "train_set"  "train_set1"
>
> gc()
          used  (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells  2576418 137.6   11441151  611.1  21052561 1124.4
Vcells 91036966 694.6  241397695 1841.8 241397695 1841.8
>
```

MODEL 1: LOGISTIC REGRESSION MODEL

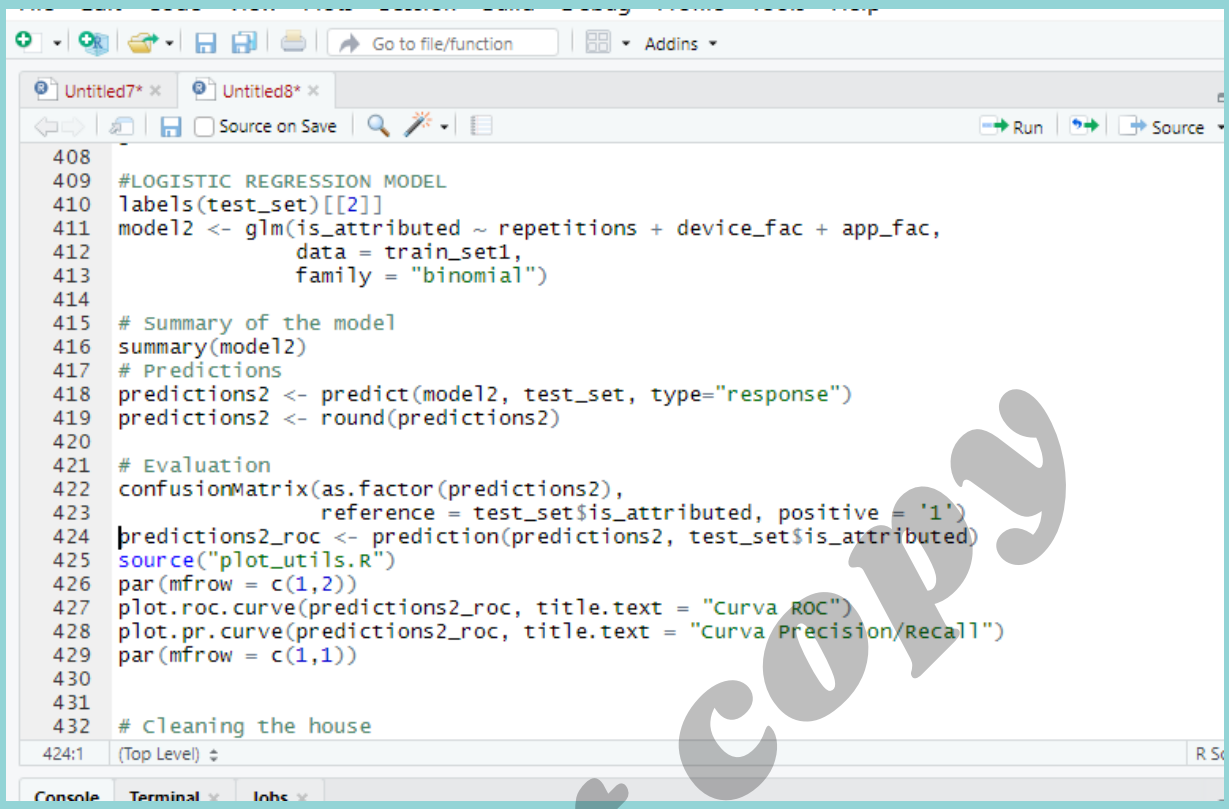
Code and Output:

```
#LOGISTIC REGRESSION MODEL
labels(test_set)[[2]]
model2 <- glm(is_attributed ~ repetitions + device_fac + app_fac,
              data = train_set1,
              family = "binomial")

# Summary of the model
summary(model2)
# Predictions
predictions2 <- predict(model2, test_set, type="response")
predictions2 <- round(predictions2)

# Evaluation
confusionMatrix(as.factor(predictions2),
                reference = test_set$is_attributed, positive = '1')
predictions2_roc <- prediction(predictions2, test_set$is_attributed)
source("plot_utils.R")
par(mfrow = c(1,2))
plot.roc.curve(predictions2_roc, title.text = "Curva ROC")
plot.pr.curve(predictions2_roc, title.text = "Curva Precision/Recall")
par(mfrow = c(1,1))


# Cleaning the house
```

```
C:/Users/hp/Desktop/talkingdata-adtracking-fraud-detection/
+                 reference = test_set$is_attributed, positive = '1')
Confusion Matrix and Statistics

          Reference
Prediction       0       1
         0 8275899    6164
         1 1705384   12553

               Accuracy : 0.8288
                 95% CI : (0.8286, 0.8291)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0108

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.670674
            Specificity : 0.829142
         Pos Pred Value : 0.007307
         Neg Pred Value : 0.999256
             Prevalence : 0.001872
         Detection Rate : 0.001255
   Detection Prevalence : 0.171794
      Balanced Accuracy : 0.749908

       'Positive' Class : 1

>
```

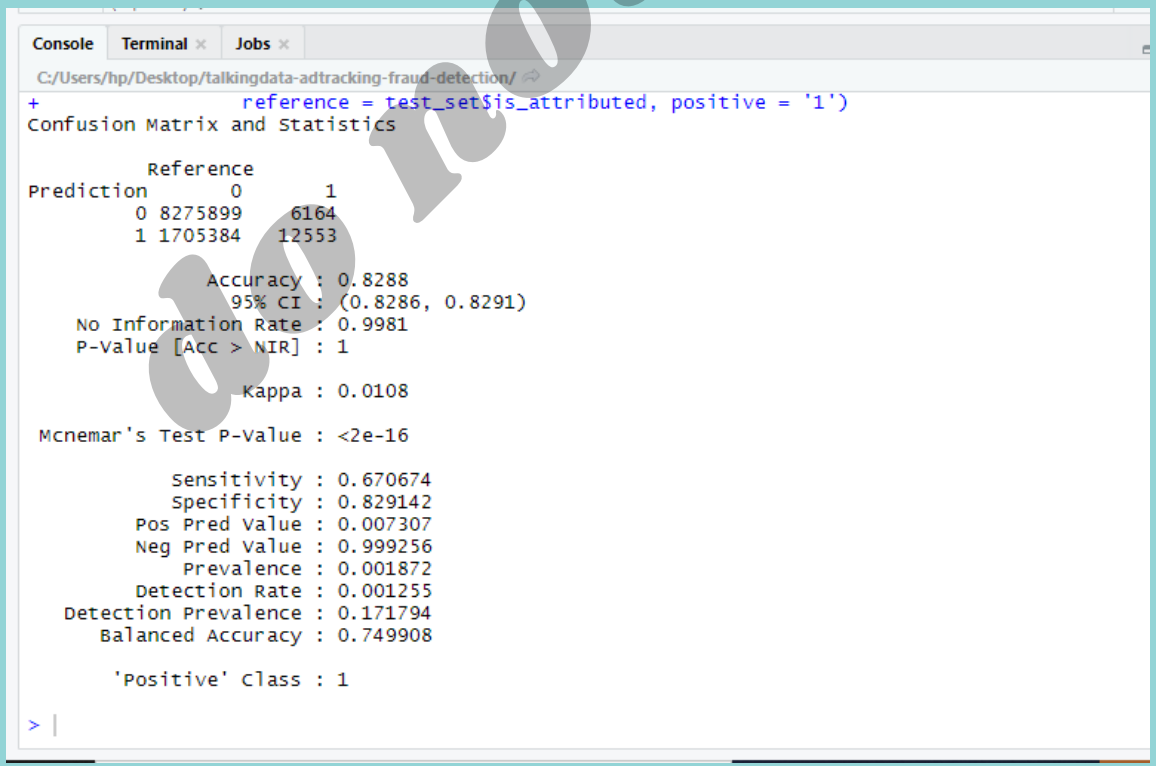
MODEL 1: LOGISTIC REGRESSION MODEL WITH MOST SIGNIFICANT VARIABLE

Code:



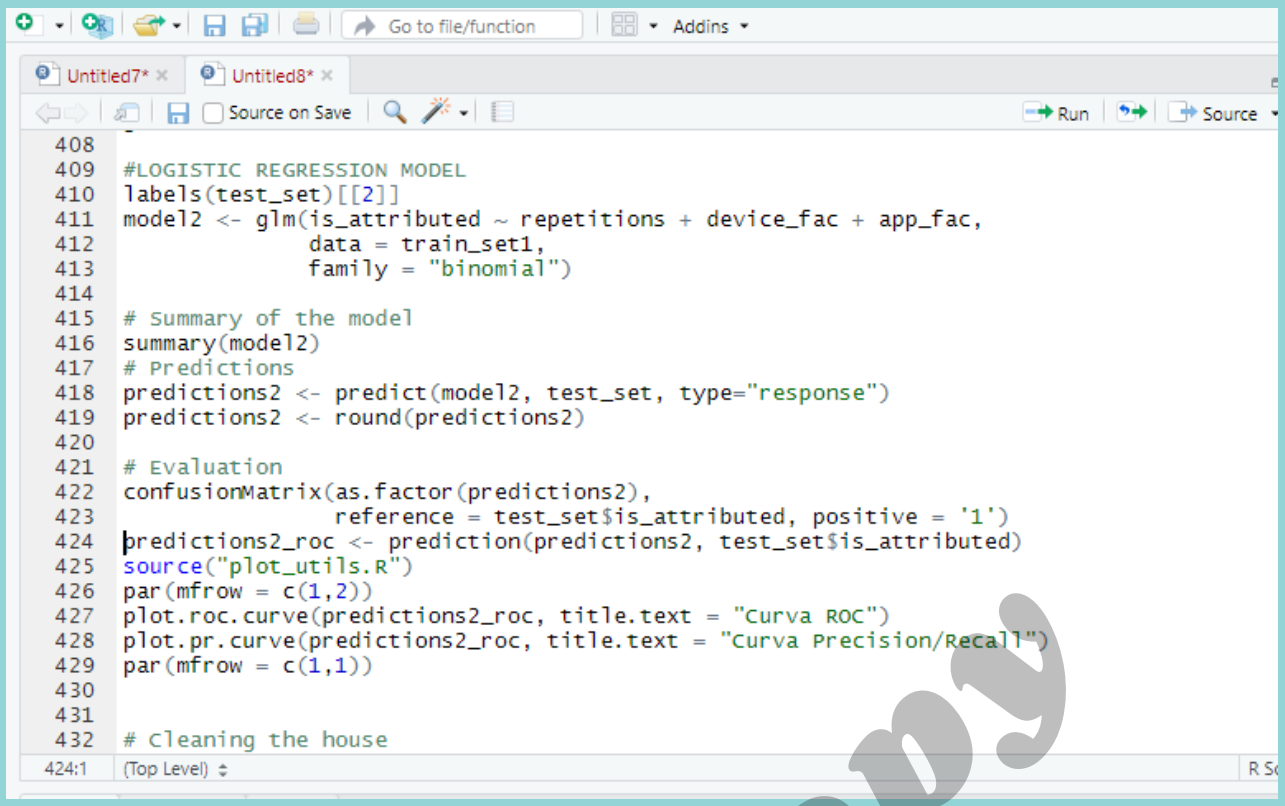
```
#LOGISTIC REGRESSION MODEL
labels(test_set)[[2]]
model2 <- glm(is_attributed ~ repetitions + device_fac + app_fac,
              data = train_set1,
              family = "binomial")

# Summary of the model
summary(model2)
# Predictions
predictions2 <- predict(model2, test_set, type="response")
predictions2 <- round(predictions2)

# Evaluation
confusionMatrix(as.factor(predictions2),
                reference = test_set$is_attributed, positive = '1')
predictions2_roc <- prediction(predictions2, test_set$is_attributed)
source("plot_utils.R")
par(mfrow = c(1,2))
plot.roc.curve(predictions2_roc, title.text = "Curva ROC")
plot.pr.curve(predictions2_roc, title.text = "Curva Precision/Recall")
par(mfrow = c(1,1))


# Cleaning the house
```

Output:



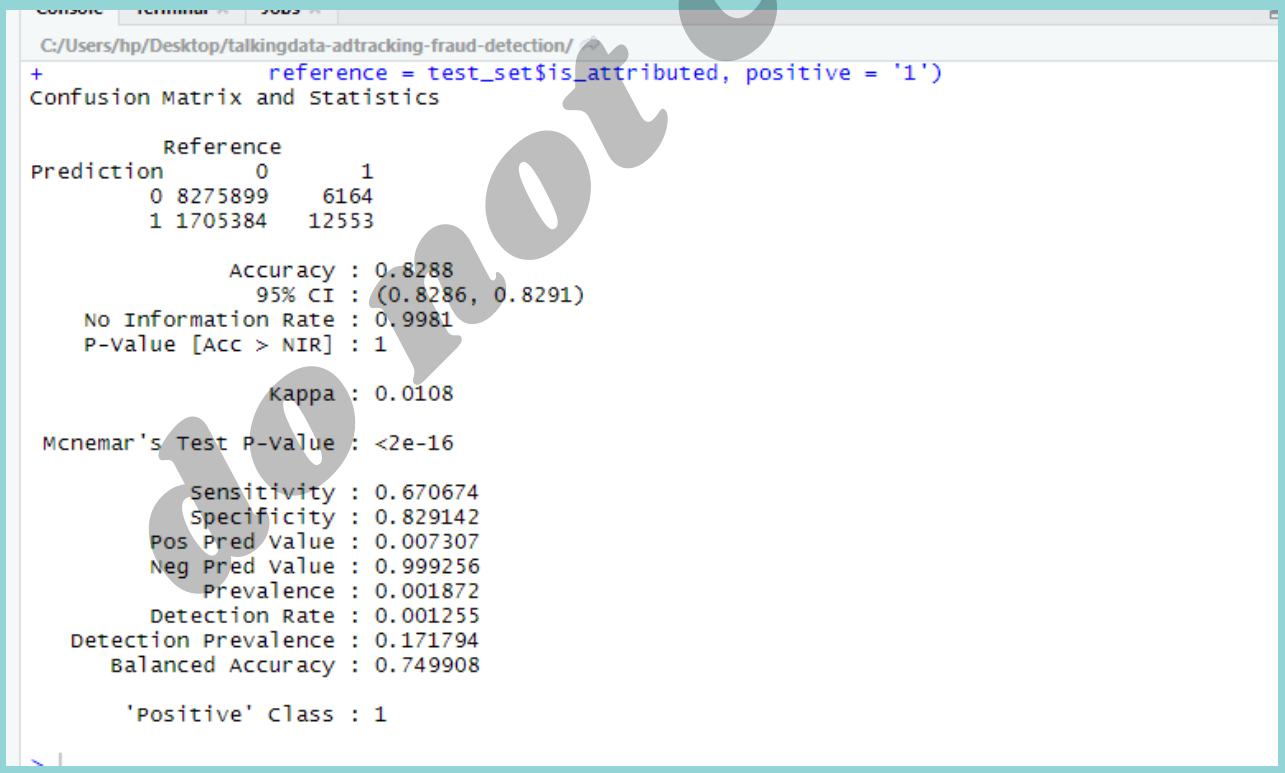
```
+                 reference = test_set$is_attributed, positive = '1')
Confusion Matrix and Statistics

          Reference
Prediction       0       1
         0 8275899    6164
         1 1705384   12553

               Accuracy : 0.8288
                 95% CI : (0.8286, 0.8291)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0108

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.670674
            Specificity : 0.829142
         Pos Pred Value : 0.007307
         Neg Pred Value : 0.999256
             Prevalence : 0.001872
         Detection Rate : 0.001255
   Detection Prevalence : 0.171794
      Balanced Accuracy : 0.749908

       'Positive' Class : 1

>
```

Code:



```
#LOGISTIC REGRESSION MODEL
labels(test_set)[[2]]
model2 <- glm(is_attributed ~ repetitions + device_fac + app_fac,
              data = train_set1,
              family = "binomial")

# Summary of the model
summary(model2)
# Predictions
predictions2 <- predict(model2, test_set, type="response")
predictions2 <- round(predictions2)

# Evaluation
confusionMatrix(as.factor(predictions2),
                reference = test_set$is_attributed, positive = '1')
predictions2_roc <- prediction(predictions2, test_set$is_attributed)
source("plot_utils.R")
par(mfrow = c(1,2))
plot.roc.curve(predictions2_roc, title.text = "Curva ROC")
plot.pr.curve(predictions2_roc, title.text = "Curva Precision/Recall")
par(mfrow = c(1,1))


# Cleaning the house
```

Output:



```
C:/Users/hp/Desktop/talkingdata-adtracking-fraud-detection/
+                reference = test_set$is_attributed, positive = '1')
Confusion Matrix and Statistics

          Reference
Prediction       0       1
         0 8275899    6164
         1 1705384   12553

               Accuracy : 0.8288
                 95% CI : (0.8286, 0.8291)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0108

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.670674
            Specificity : 0.829142
         Pos Pred Value : 0.007307
         Neg Pred Value : 0.999256
             Prevalence : 0.001872
         Detection Rate : 0.001255
   Detection Prevalence : 0.171794
      Balanced Accuracy : 0.749908

       'Positive' Class : 1
```

MODEL 2: REGRESSION TREE MODEL WITH MOST SIGNIFICANT VARIABLLE

Code:

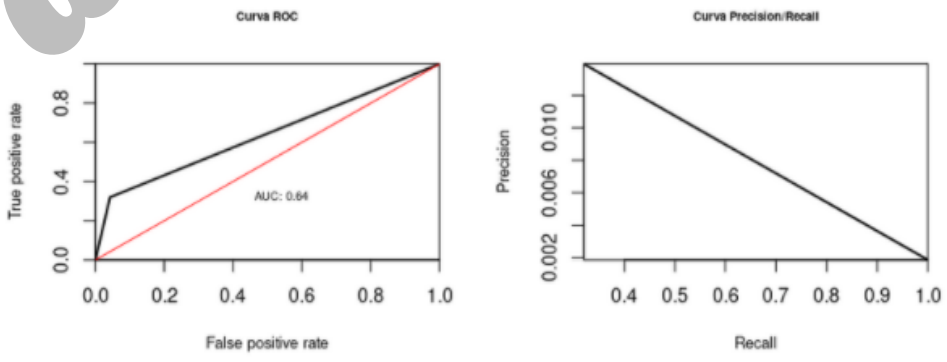
```
# Predictions
predictions2 <- predict(model2, test_set, type="response")
predictions2 <- round(predictions2)

# Evaluation
confusionMatrix(as.factor(predictions2),
                reference = test_set$is_attributed, positive = '1')
predictions2_roc <- prediction(predictions2, test_set$is_attributed)
install.packages("plot_utils.R")
library(plot_utils.R)
source("plot_utils.R")
install.packages("ROCR")
library(ROCR)

par(mfrow = c(1,2))
plot.roc.curve(predictions2_roc, title.text = "Curva ROC")
plot.pr.curve(predictions2_roc, title.text = "Curva Precision/Recall")
par(mfrow = c(1,1))


# Cleaning the house
rm(list = setdiff(ls(), c('train_set', 'train_set1', 'test_set')))
gc()
detach(package:ROCR)
```

Output:

Data	
model2	List of 30
test_set	10000000 obs. of 15 variables
train_set	100000 obs. of 18 variables
train_set1	454 obs. of 18 variables

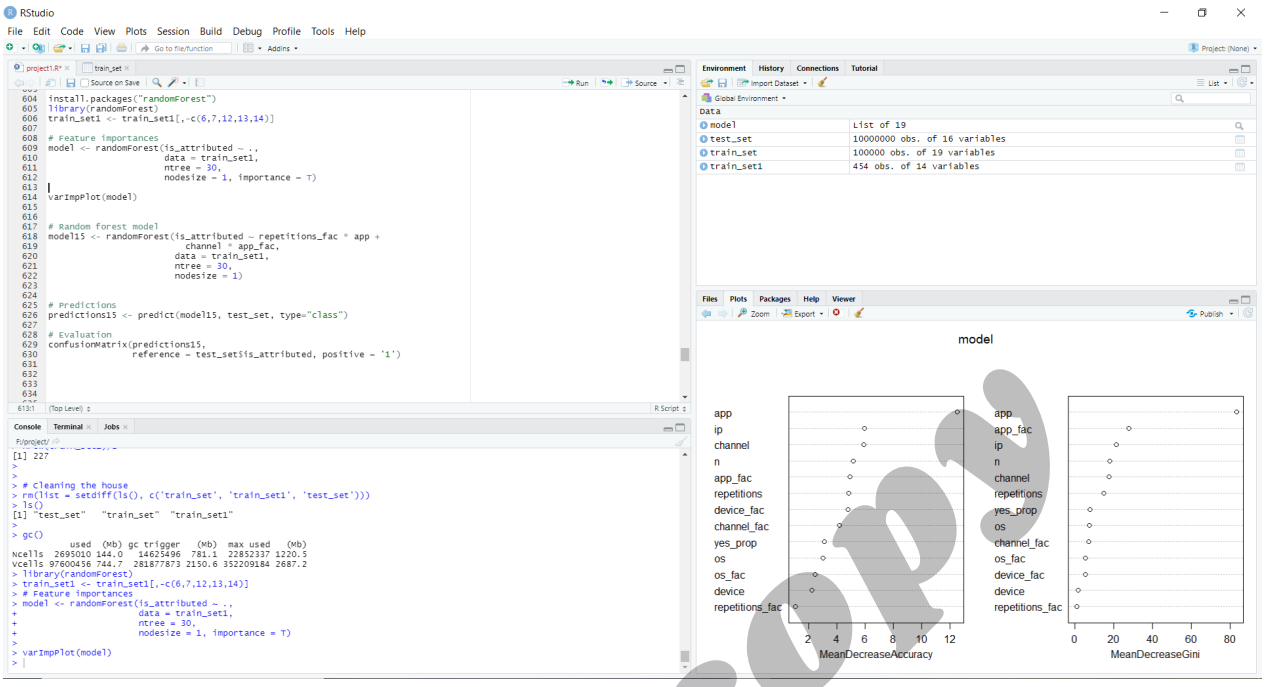
Files | Plots | Packages | Help | Viewer

Zoom | Export



MODEL 3: RANDOM FOREST BALANCED BY SMOTE

Code:



Code:

Output:

```
Console  Terminal  Jobs
F:/project/
          Reference
Prediction       0       1
         0 9720774    4940
         1  260509   13777

               Accuracy : 0.9735
                 95% CI : (0.9734, 0.9736)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0909

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.736069
            Specificity : 0.973900
         Pos Pred Value : 0.050229
         Neg Pred Value : 0.999492
             Prevalence : 0.001872
         Detection Rate : 0.001378
   Detection Prevalence : 0.027429
      Balanced Accuracy : 0.854985
```

```
Console  Terminal  Jobs
F:/project/
> model15 <- randomForest(is_attributed ~ repetitions_fac * app +
+                           channel * app_fac,
+                         data = train_set1,
+                         ntree = 30,
+                         nodesize = 1)
> # Predictions
> predictions15 <- predict(model15, test_set, type="class")
> # Evaluation
> confusionMatrix(predictions15,
+                 reference = test_set$is_attributed, positive = '1')
Confusion Matrix and Statistics

Confusion Matrix and Statistics

          Reference
Prediction       0       1
         0 9720774    4940
         1  260509   13777

               Accuracy : 0.9735
                 95% CI : (0.9734, 0.9736)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1
```

Code:

```
project1.R  train_set
Source on Save   Run   Sou
614 varImpPlot(model1)
615
616
617 library(DMwR)
618 train_set1 <- train_set %>%
619   select(is_attributed, repetitions_fac, app, channel, app_fac)
620
621 train_set1 <- SMOTE(is_attributed ~ ., data  = train_set1)
622 table(train_set1$is_attributed)
623
624 # Random forest model
625 model15 <- randomForest(is_attributed ~ repetitions_fac * app +
626                           channel * app_fac,
627                         data = train_set1,
628                         ntree = 30,
629                         nodesize = 1)
630
631 # Predictions
632 predictions15 <- predict(model15, test_set, type="class")
633
634 # Evaluation
635 confusionMatrix(predictions15,
636                 reference = test_set$is_attributed, positive = '1')
637
638
639
640
```

Output:

```
Console  Terminal  Jobs
F:/project/
> library(DMwR)
> train_set1 <- train_set %>%
+   select(is_attributed, repetitions_fac, app, channel, app_fac)
>
> train_set1 <- SMOTE(is_attributed ~ ., data  = train_set1)
> table(train_set1$is_attributed)

  0   1
908 681
> # Random forest model
> model15 <- randomForest(is_attributed ~ repetitions_fac * app +
+                           channel * app_fac,
+                         data = train_set1,
+                         ntree = 30,
+                         nodesize = 1)
> # Predictions
> predictions15 <- predict(model15, test_set, type="class")
> # Evaluation
> confusionMatrix(predictions15,
+                 reference = test_set$is_attributed, positive = '1')
Confusion Matrix and Statistics

          Reference
```

```
Confusion Matrix and Statistics

          Reference
Prediction       0       1
         0 9801351    4954
         1  179932   13763

               Accuracy : 0.9815
                 95% CI : (0.9814, 0.9816)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1266

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.735321
            Specificity : 0.981973
         Pos Pred Value : 0.071055
         Neg Pred Value : 0.999495
             Prevalence : 0.001872
         Detection Rate : 0.001376
   Detection Prevalence : 0.019370
```

```
               Accuracy : 0.9815
                 95% CI : (0.9814, 0.9816)
    No Information Rate : 0.9981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1266

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.735321
            Specificity : 0.981973
         Pos Pred Value : 0.071055
         Neg Pred Value : 0.999495
             Prevalence : 0.001872
         Detection Rate : 0.001376
   Detection Prevalence : 0.019370
      Balanced Accuracy : 0.858647

       'Positive' Class : 1
```

Literature Review

A company wants to know the CTR (Click Through Rate) in order to identify whether spending their money on digital advertising is worth it or not. A higher CTR represents more interest in that specific campaign, whereas a lower CTR can show that your ad may not be as relevant. High CTRs are important because they show that more people are clicking through to your website. Along with this high CTRs also help to get better ad positions for less money on online platforms like Google, Bing, etc.

A paper published in the Australian Journal of Information Systems in the year 2019, explores the various hypotheses surrounding the subject. The research confirms that the publishers-controlled factors: ad space duration, ad space size, ad space position, and ad space timing all have a significantly strong impact on the click-through rate of mobile in-app advertising.

The research has shown the main effects of ad space duration, ad space size, ad space position, and ad space timing, it has not found the optimal value for each. This leaves room for the practitioners to continue to test on each factor with more variants to determine the optimal value. Another research was done at the University of Science and Technology, Beijing, China; suggested a WELM-Adaboost algorithm-based approach for the CTR prediction of the RTB advertisement.

They applied the real advertisement dataset to implement the experiments by applying the AUC value as the measurement criteria. They compared both the ELM algorithm and the Weighted-ELM algorithm with the proposed approach.

Several research works to find solutions to the ad-click prediction have already been done using various machine learning techniques like Logistic Regression, Naive Bayes Classifier, Support Vector Machines, and Decision Trees.

Logistic Regression has a pivotal role in the early analysis in this domain. Due to the accelerated growth of online activity, analysts have been procuring different procedures to display relevant advertisements that suit the viewer's interest.

The recent advancement in social networking on the internet has formed some fabulous illustrations of ad serving machineries such as Twitter and Facebook. Twitter advertising prototype is also based on logistic regression.

Related Works

Exploring predictors' importance in binomial logistic regressions

This presents a real example where dominance analysis is used to determine predictors' importance in a binomial logistic regression model. More specifically, we model the distribution of a tropical native bird species, inhabiting a small oceanic island, using a binomial generalized linear model, and dominance analysis to identify the most important environmental variables.

RankEval: An Evaluation and Analysis Framework for Learning-to-Rank Solutions

RankEval is an open-source tool for the analysis and evaluation of Learning-to-Rank models based on ensembles of regression trees. The success of ensembles of regression trees fostered the development of several open-source libraries targeting efficiency of the learning phase and effectiveness of the resulting models. However, these libraries offer only very limited help for the tuning and evaluation of the trained models.

Speech Emotion Recognition

This repository handles building and training Speech Emotion Recognition System. The basic idea behind this tool is to build and train/test a suited machine learning (as well as deep learning) algorithm that could recognize and detect human emotions from speech. This is useful for many industry fields such as making product recommendations, affective computing, etc.

CAX-startup-prediction

CAX team will assist and provide adequate guidance wherever necessary. Solvers are encouraged to ask questions, discuss and share their codes and interesting materials they find to keep the forum active and engaged. CAX team will share the top submissions with the community.

Scope and Limitations

Scope

Marketing Plan

A marketing plan, that an app marketing agency team will devise for your app, will consist of – an app's marketing potential analysis, the right app marketing channels to use, possible adjustments for your app's marketing copy and creatives to produce. The app marketing best practices also demand an agency to carefully measure KPIs (stands for Key Performance Indicators) of such app marketing campaigns.

Growth Hacking

There is one set of apps promotion techniques that stands out the rest – Growth Hacking. Essentially it's about finding the ways for a particular mobile app to grow its number of users and revenue via applying non-standard app marketing techniques, preferably with low budget.

Limitations

- It doesn't ensure that the data captured is reliable because the user might click on the ad by mistake when it pops up.
- In the case of ensemble learning: it uses weakes models together which can induce a bias trade-off.
- **Dataset limitation:** More attributes can be added to make the prediction more accurate.The diversity of the instances could be taken for a wider range of population.
- Machine learning models have lesser computational units than Deep Learning.

Significance

Taking into consideration that only about 2 billion out of 7.3 billion people in the world have a smartphone, mobile advertising growth will continue despite the fact that this sector is already outpacing other ones. Therefore, we can expect new markets to emerge.

Accurate store visits enable you to build more customized audiences, which ensures you'll reach shoppers when they're active and engaged. The more accurate the location data, the better your ability to deliver offers and deals to customers when it matters most.

Mobile in-app advertising is a growing business. Optimizing mobile in-app advertising by itself is a new subject. For publishers, who have more than one app published, applying the new supply and delivery strategies could bring multiple benefits to them.

For agents, who publish the apps on the publishers' behalf, this strategy can bring even more values. Ad networks can integrate new strategies associated with these factors to increase the matching and relevance of the ads to their users.

New tendencies transpire with mobile advertising growth, such as protection from mobile ad fraud and mobile ad blocking. The latter represents a significant problem for publishers as the rate of ad blocking users on mobile skyrocketed.

Above all, users tend to get rid of mobile apps which abuse irrelevant in-app advertising. That's why publishers and developers should adopt less intrusive media within their mobile marketing strategies.

For all that, programmatic advertising still runs the show. It's crucial to identify what works for your mobile app and to global trends. The field offers dozens of various adjuvant tools and platforms, publishers should choose wisely picking the ones that will supercharge their mobile marketing strategies and boost their store performance.

References

- Hosmer, D. & Lemeshow, S. (2000). Applied Logistic Regression (Second Edition). New York: John Wiley & Sons, Inc. (Online Library Link: https://onlinelibrary.wiley.com/doi/book/10.1002/0471722146)
- CRAN Repository for R (https://cran.r-project.org/web/packages/kernlab/kernlab.pdf)
- https://www.rdocumentation.org/packages/kernlab/versions/0.9-29/topics/ksvm
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589
- https://scikit-learn.org/stable/modules/svm.html https://uc-r.github.io/regression_trees
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://www.kaggle.com/chtaret/fraud-detection-with-smote-and-randomforest H.-T. Lin, C.-J. Lin and R. C. Weng
- A note on Platt's probabilistic outputs for support vector machines(https://www.csie.ntu.edu.tw/~htlin/paper/doc/plattprob.pdf)

THANK YOU

Group Members

1.	Atharva Borkar	18BCG10026
2.	Arpit Kumar	18BCE10058
3.	Anindita Mishra	18BCE10038
4.	Niharika Singh	18BCE10171
5.	Ronak Jain	18BCE10222
6.	Shivani Sharma	18BCE10251
7.	Shreya Thakur	18BCE10256
8.	Vasudha Dwivedi	18BCE10288