Detecção de Fraudes no Tráfego de Cliques em Propagandas de Aplicações Mobile

Lucas Kitano

22/06/2021

Projeto com Feedback 1 do curso Big Data Analytics com R e Microsoft Azure Machine Learning

O objetivo deste projeto é criar um modelo que possa prever se um usuário fará o download de um aplicativo depois de clicar em um anúncio para dispositivos móveis. A importância disso se dá, pois um usuário pode clicar na propaganda apenas para gerar tráfego, mas não baixar o aplicativo em si, neste sentido, o objetivo deste projeto é identificar se um clique é fraudulento ou não.

Os dados utilizados são da pataforma TalkingData ("https://www.talkingdata.com"), cobrindo mais de 70% dos dispositvos móveis da China - o maior mercado móvel do mundo. Estima-se que eles lidam com 3 bilhões de cliques por dia, dos quais 90% são potencialemnte fraudulentos. Vamos criar um modelo de Machine Learning e obter algumas conclusões sobre os dados disponiveis.

Pacotes necessários

Aqui estão os pacotes necessários para a execução/reprodução deste script:

```
# Carrega Pacotes Necessários
# Use o comando "install.packages("nome_do_pacote") para instalar algum dos pacotes abaixo,
# caso não o possua.
library(sqldf)
```

```
## Loading required package: gsubfn
## Loading required package: proto
## Loading required package: RSQLite
```

```
library("ggplot2")
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(ROSE)
```

```
## Loaded ROSE 0.0-3
```

```
library(C50)
library(caret)
```

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

Carregando o dataset

Como o dataset original é grande (~7,4 GB), foi utilizado o comando "read.csv.sql" que permite carregar um arquivo CSV aplicando uma instrução da linguagem SQL, que no caso foi utilizada para trazer amostras 1.000.000 de amostras randômicas do dataset original.

```
# Define o diretório de datasets:
arquivo = "datasets/train.csv"

dados <- read.csv.sql(arquivo, header = TRUE, sep = ",",
                      sql = "select * from file order by random() limit 1000000", eol = "\n")


head(dados)
```

```
##       ip app device os channel         click_time attributed_time
## 1 117356   2      1  6     219 2017-11-08 00:58:32
## 2  35840  11      1 12     487 2017-11-07 04:07:19
## 3 133522   2      1 43     435 2017-11-07 13:42:45
## 4  39515  12      1 30     265 2017-11-08 10:39:37
## 5  32069   9      1 13     127 2017-11-09 04:36:06
## 6  59290   1      1 32     134 2017-11-07 09:02:40
##   is_attributed
## 1             0
## 2             0
## 3             0
## 4             0
## 5             0
## 6             0
```

Colunas do dataset:

ip: Endereço IP do clique;
app: ID do app (Marketing);
device: ID do celular do usuário (Exs. iphone 6 plus, iphone 7, huawei mate 7, etc.);
os: ID da versão do SO do celular do usuário;
channel: ID do editor de anúncios para celular;
click_time: Hora do clique;
attributed_time: Momento do download do aplicativo, caso o usuário tenha baixado o aplicativo;
is_attributed: A variável target, indicando se o app foi baixado.

As variáveis "ip", "app", "device", "os", e "channel" estão codificadas.

Para maiores detalhes sobre o dataset acesse: "https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data"

Pré-Processamento, Feature Engineering a Análise Exploratória

```
# Verificando o formato atribuído aos dados:
str(dados)
```

```
## 'data.frame':    1000000 obs. of  8 variables:
##  $ ip             : int  117356 35840 133522 39515 32069 59290 81799 15517 295568 27845 ...
##  $ app            : int  2 11 2 12 9 1 9 8 56 3 ...
##  $ device         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ os             : int  6 12 43 30 13 32 12 13 16 13 ...
##  $ channel        : int  219 487 435 265 127 134 215 145 406 115 ...
##  $ click_time     : chr  "2017-11-08 00:58:32" "2017-11-07 04:07:19" "2017-11-07 13:42:45" "2017-11-0
##  $ attributed_time: chr  "" "" "" "" ...
##  $ is_attributed  : int  0 0 0 0 0 0 0 0 0 0 ...
```

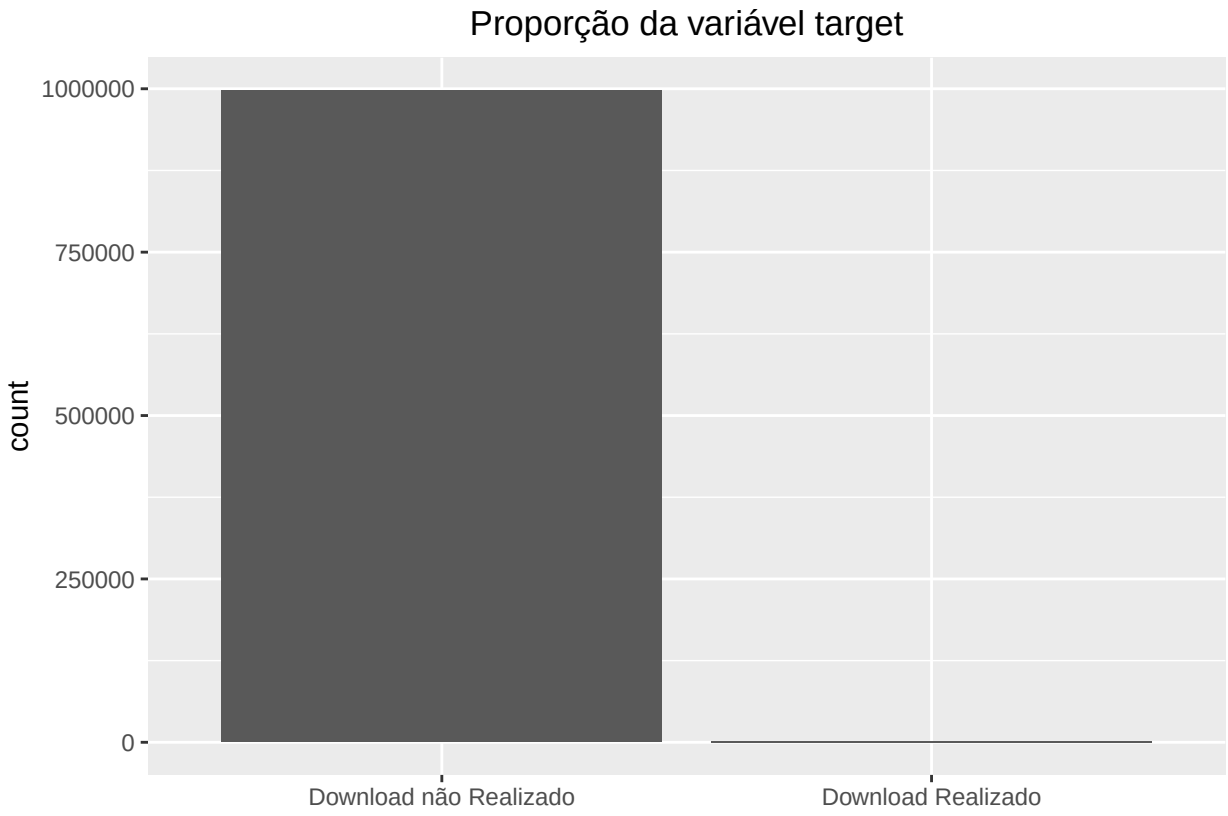
```
# Verificando se temos valores ausentes no dataset
sum(is.na(dados))
```

```
## [1] 0
```

```
# Transformando a variável alvo em fator:
dados$is_attributed <- factor(dados$is_attributed, levels = c(0, 1), labels = c("0", "1"))

# Verificando a distribuição de valores da variável alvo:
round(prop.table(table(dados$is_attributed))*100, digits = 1)
```

```
##
##    0    1
## 99.8  0.2
```

```
ggplot(data = dados, aes(x = factor(is_attributed, labels = c("Download não Realizado", "Download Reali
  geom_bar() +
  ggtitle("Proporção da variável target") + xlab("") + theme(plot.title = element_text(hjust = 0.5))
```



Normalmente não consideramos as variáveis de ID no modelo, no entanto, como o endereço IP identifica a localidade do usuário, vamos agrupar o endereço IP por faixas de valores.

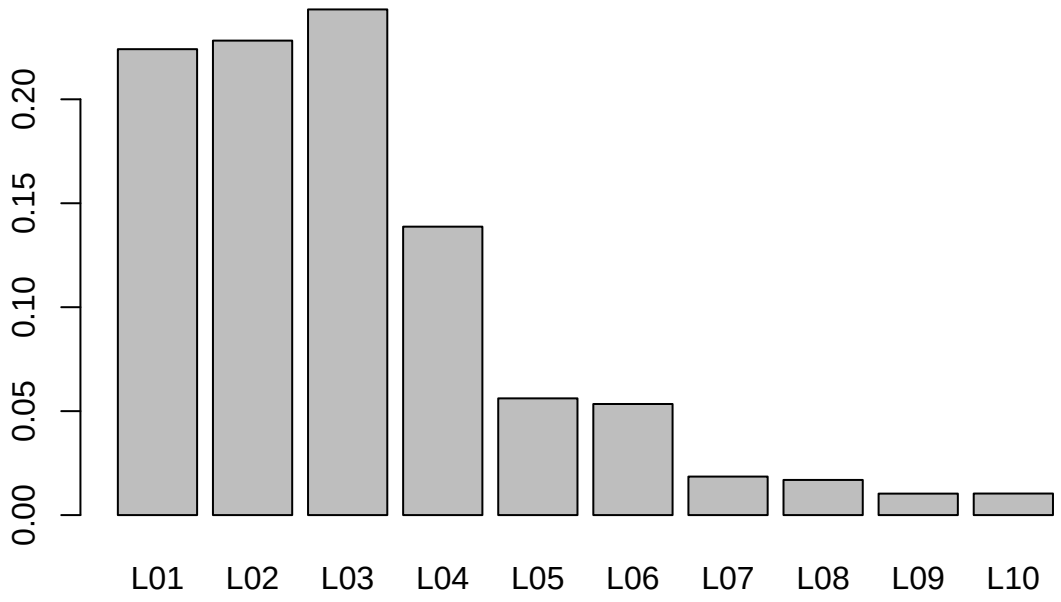
```
faixas_ip = 10

# Range do IP englobado em cada faixa:
(max(dados$ip) - min(dados$ip)) / faixas_ip
```

```
## [1] 36477.6
```

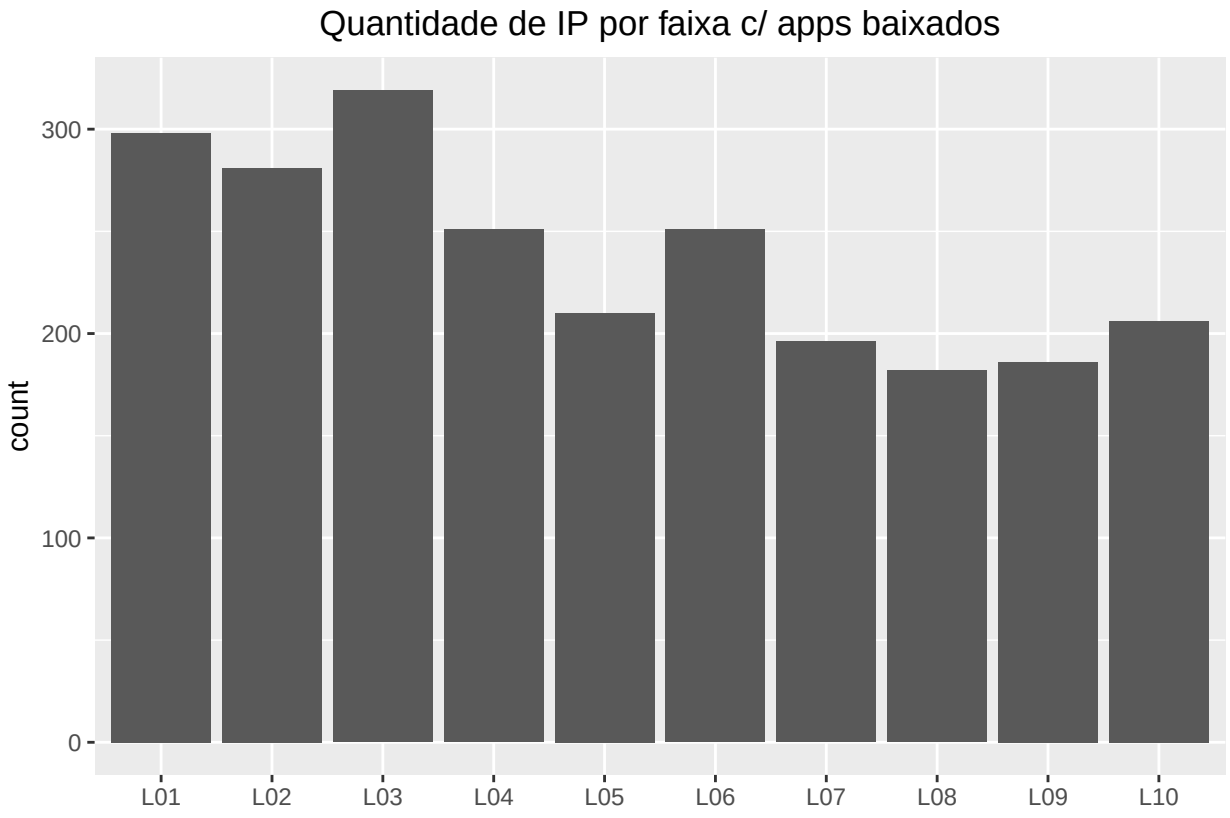
```
dados$ip_cut <- cut(dados$ip, faixas_ip, labels=c("L01", "L02", "L03", "L04", "L05", "L06", "L07", "L08

# Quantidades de IP por faixa:
table(dados$ip_cut)
```

```
##
##    L01    L02    L03    L04    L05    L06    L07    L08    L09    L10
## 224118 228230 243232 138732  56142  53441  18516  16900  10331  10358
```

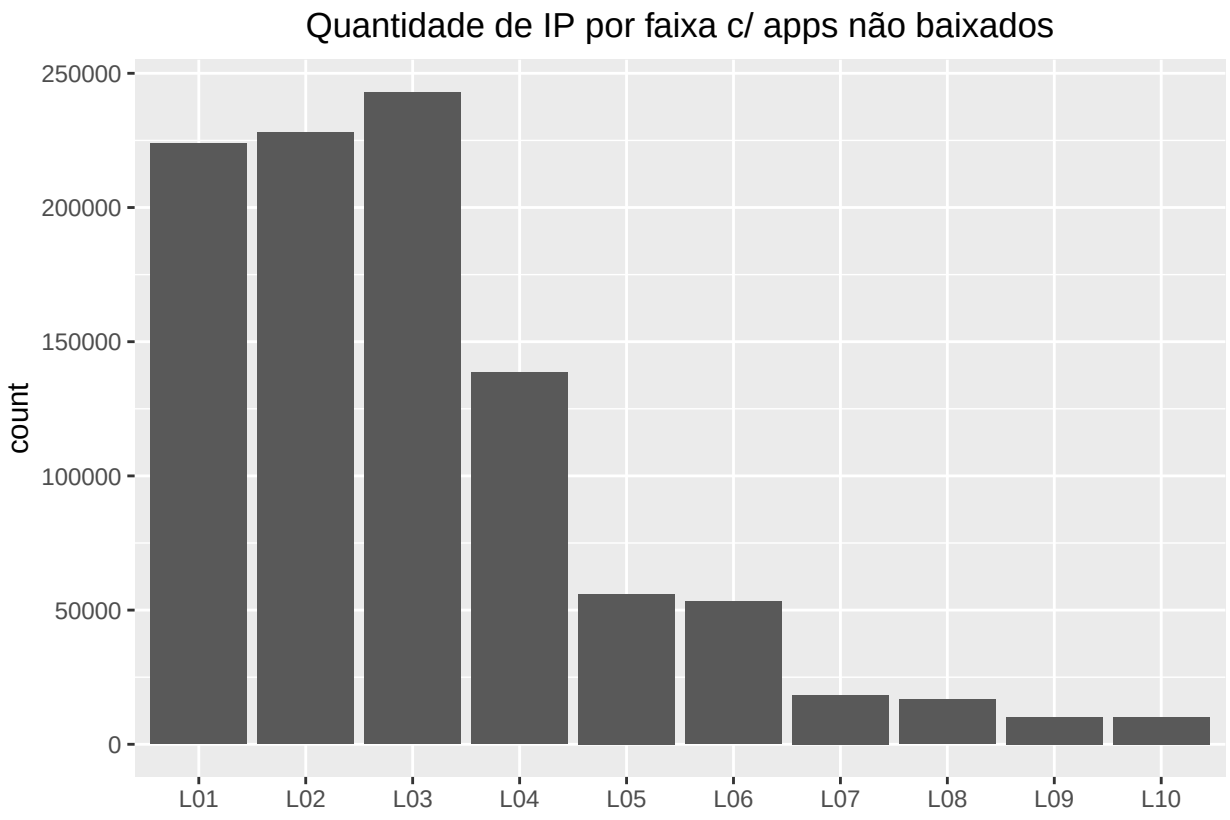
```
barplot(prop.table(table(dados$ip_cut)), main = "Quantidade de IP por faixa")
```




```
dados %>% filter(is_attributed=="1") %>%
  ggplot(., aes(ip_cut)) + geom_bar() + ggtitle("Quantidade de IP por faixa c/ apps baixados") + xlab("
  theme(plot.title = element_text(hjust = 0.5))
```




```
dados %>% filter(is_attributed=="0") %>%
  ggplot(., aes(ip_cut)) + geom_bar() + ggtitle("Quantidade de IP por faixa c/ apps não baixados") + xla
  theme(plot.title = element_text(hjust = 0.5))
```



Podemos observar que para os cliques que não geraram downloads, a maioria estão na localidade L01-L03, enquanto que para os que baixaram o app pós clique, a distribuição de IPs está mais uniforme, com a maioria dentro de L01. Conforme mencionado, vamos manter esta variável no modelo, com a ressalva de que possivelmente os algoritmos que gerem cliques fraudulentos possam estar mascarando o seu IP através de uma VPN, o que pode explicar em partes o por que da distribuição de IP's para quem baixou o APP estar com uma distribuição mais uniforme.

A variável "app" identifica um app pelo ID, assim como a variável "device" identifica o tipo de dispositvo do registro, e a variável "channel" identifica o ID do publisher, vamos converter estas variáveis para char, pois no modelo elas podem indicar que um app, ou dispositivo gerem mais ou menos downloads após o clique na propaganda.

```
dados$app_chr <- as.character(dados$app)
dados$device_chr <- as.character(dados$device)
dados$channel_chr <- as.character(dados$channel)
```

Não iremos considerar a variável "os", pois acreditamos que para o problema, ela é redundante com a variável "device".

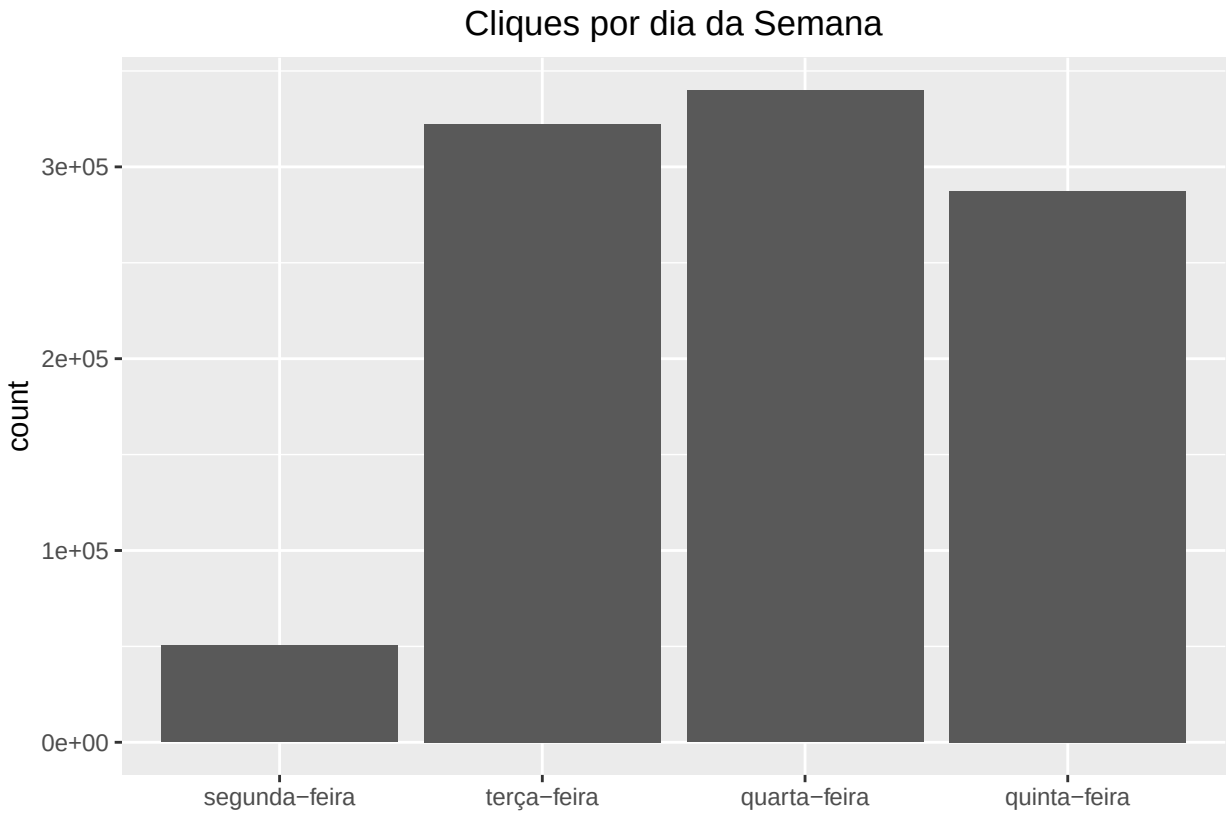
A variável "attributed_time" também não será considerada, pois ela só revela o momento do download caso o aplicativo tenha sido baixado.

Em relação a variável "click_time" vamos averiguar alguns pontos sobre ela:

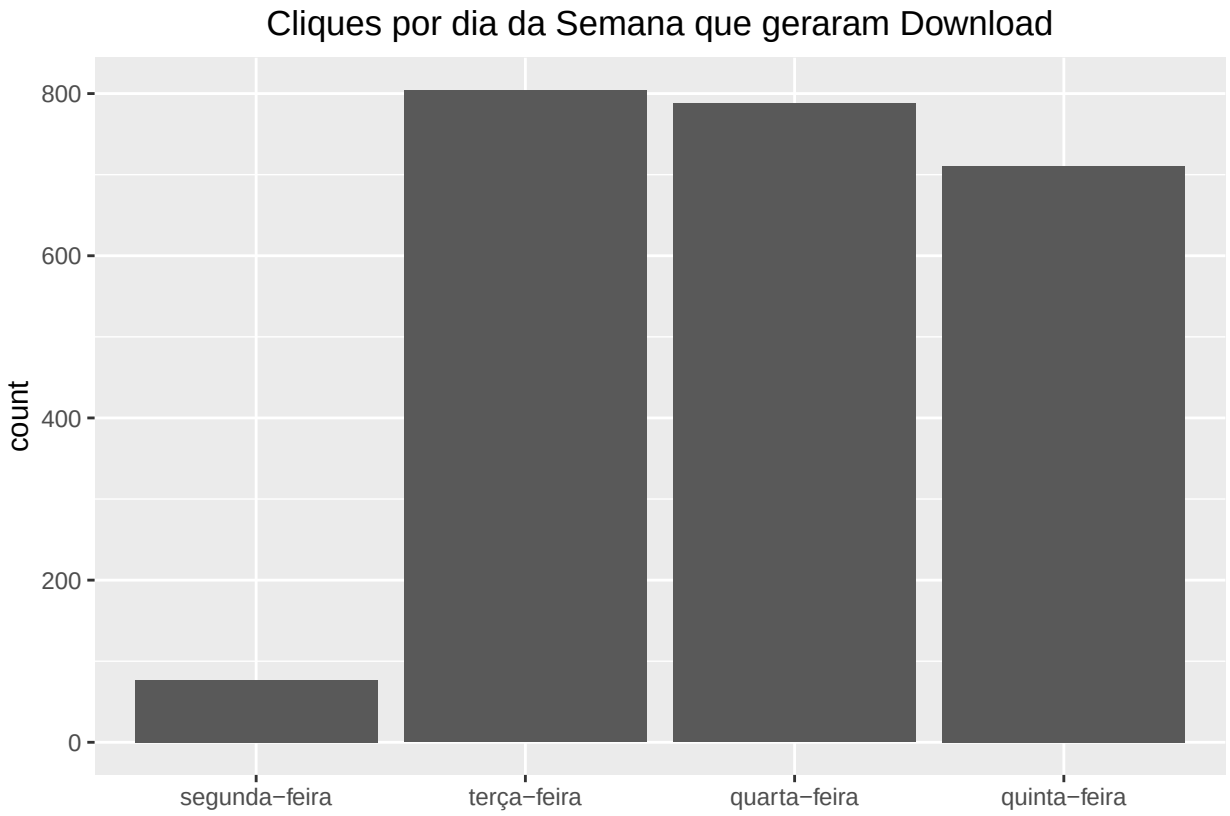
Vamos num primeiro momento extrair o dia da semana, e transformar esta variável em um fator:

```
dados$weekday <- weekdays(as.POSIXct(dados$click_time))

dados$weekday <- factor(dados$weekday,
```

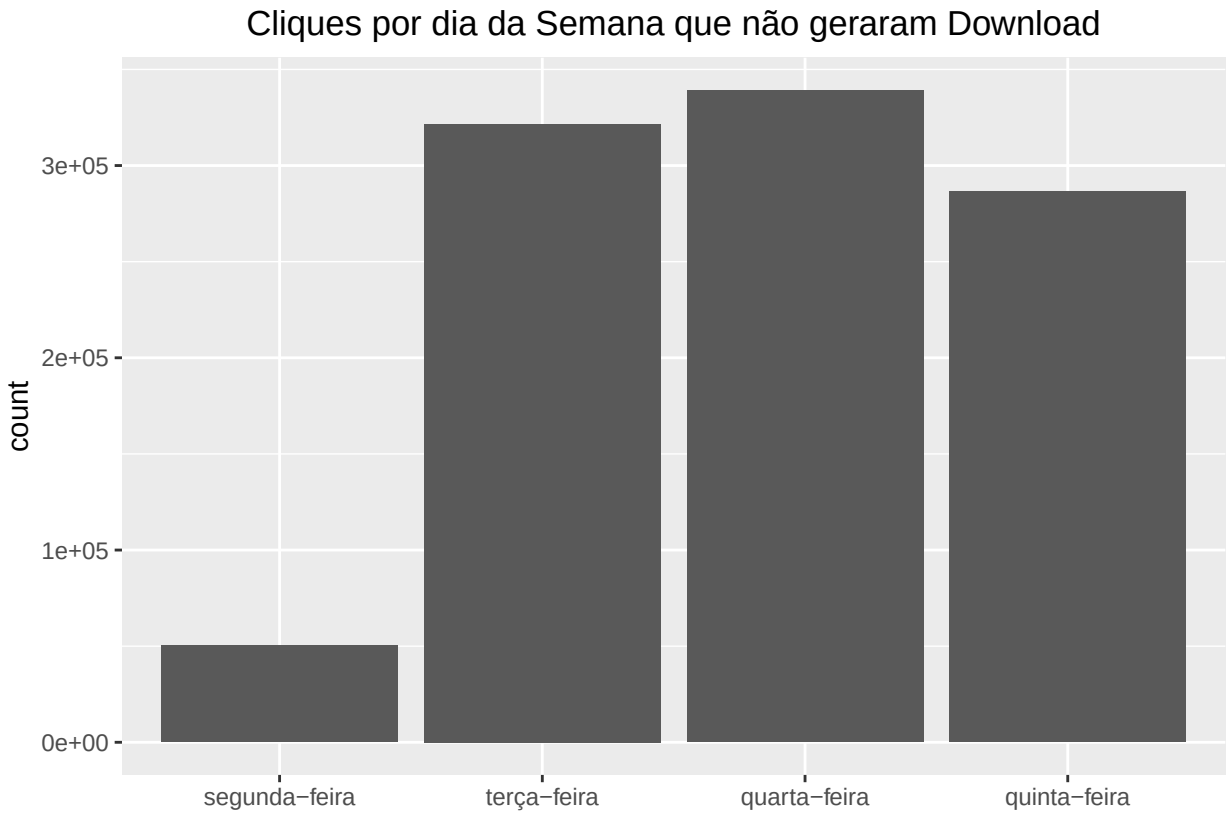
```
                            levels = c("domingo", "segunda-feira", "terça-feira", "quarta-feira", "quin
                                       "sexta-feira, sabado"))

# E então extrair a hora do evento:

dados$hourday <- as.numeric(strftime(as.POSIXct(dados$click_time), format="%H"))

# E com a hora do evento, separar em faixas do dia e também transformar em fator:

dados <- dados %>%
  mutate(period = case_when(
    .$hourday <= 6 ~ "Madrugada",
    .$hourday <= 12 ~ "Manha",
    .$hourday <= 18 ~ "Tarde",
    .$hourday <= 24 ~ "Noite"
    ))

dados$period <- factor(dados$period,
                       levels = c("Madrugada", "Manha", "Tarde", "Noite"))


# Vamos verificar a distribuição dos dados pelo dia da semana:

dados %>%
  ggplot(., aes(weekday)) + geom_bar() + ggtitle("Cliques por dia da Semana") + xlab("") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
dados %>% filter(is_attributed=="1") %>%
  ggplot(., aes(weekday)) + geom_bar() + ggtitle("Cliques por dia da Semana que geraram Download") + xla
  theme(plot.title = element_text(hjust = 0.5))
```



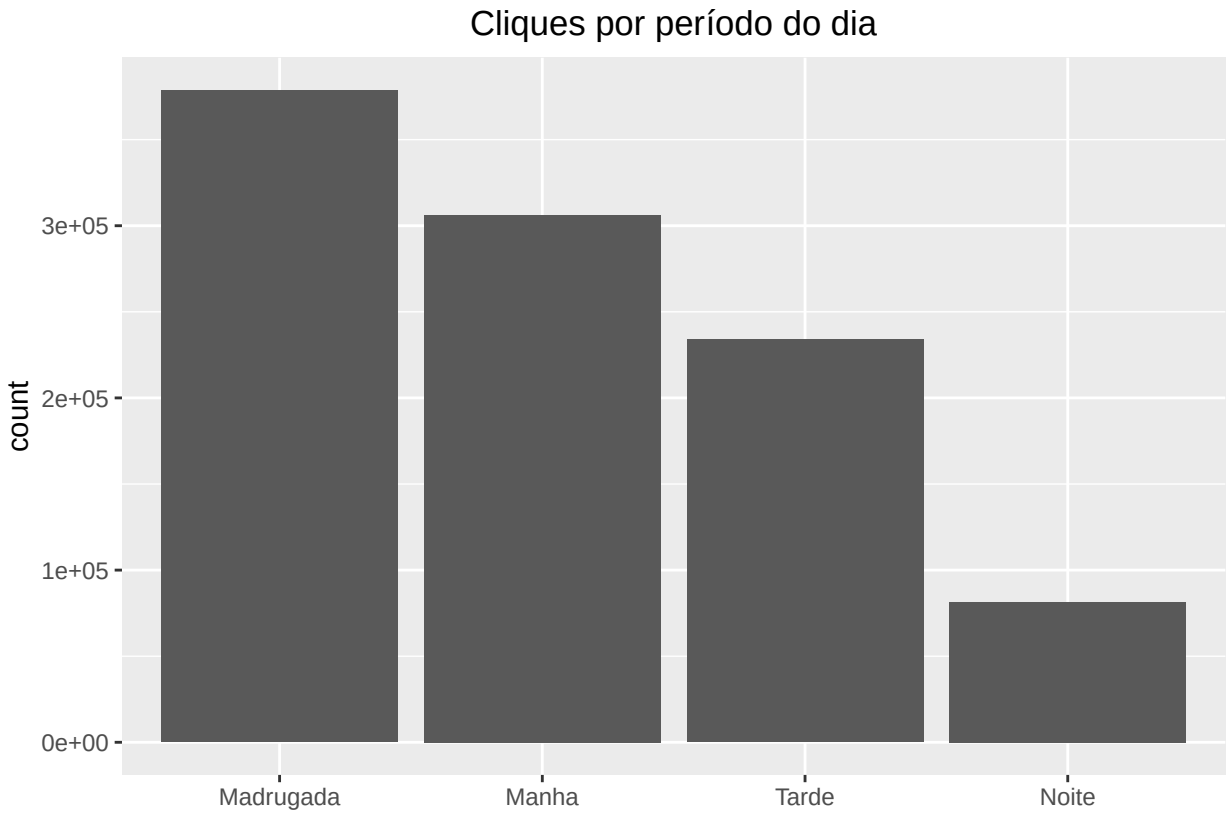

```
dados %>% filter(is_attributed=="0") %>%
  ggplot(., aes(weekday)) + geom_bar() + ggtitle("Cliques por dia da Semana que não geraram Download") +
  theme(plot.title = element_text(hjust = 0.5))
```



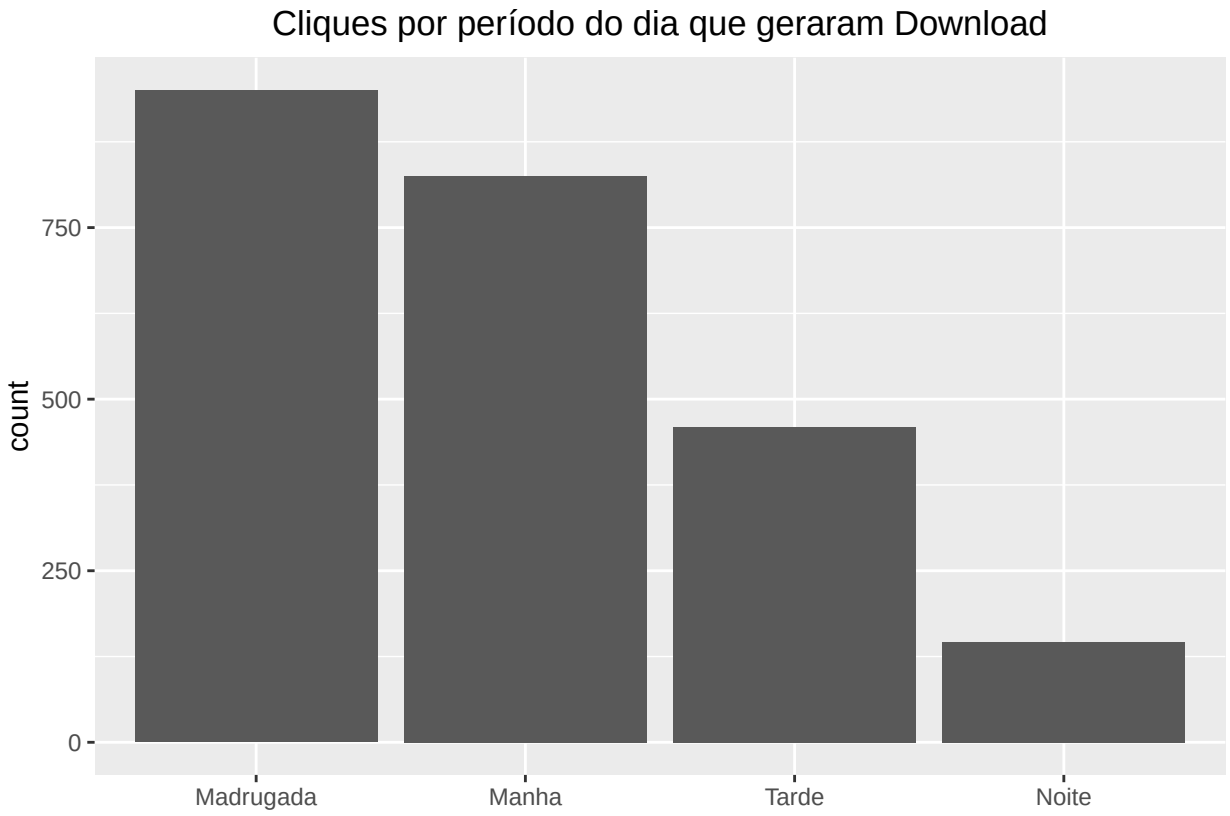
A distribuição de dados por dia da semana é semelhante tanto para todos os registros do dataset, quanto para os casos do qual o clique gerou download do app.

Vamos verificar a distribuição dos dados pelo período do dia:

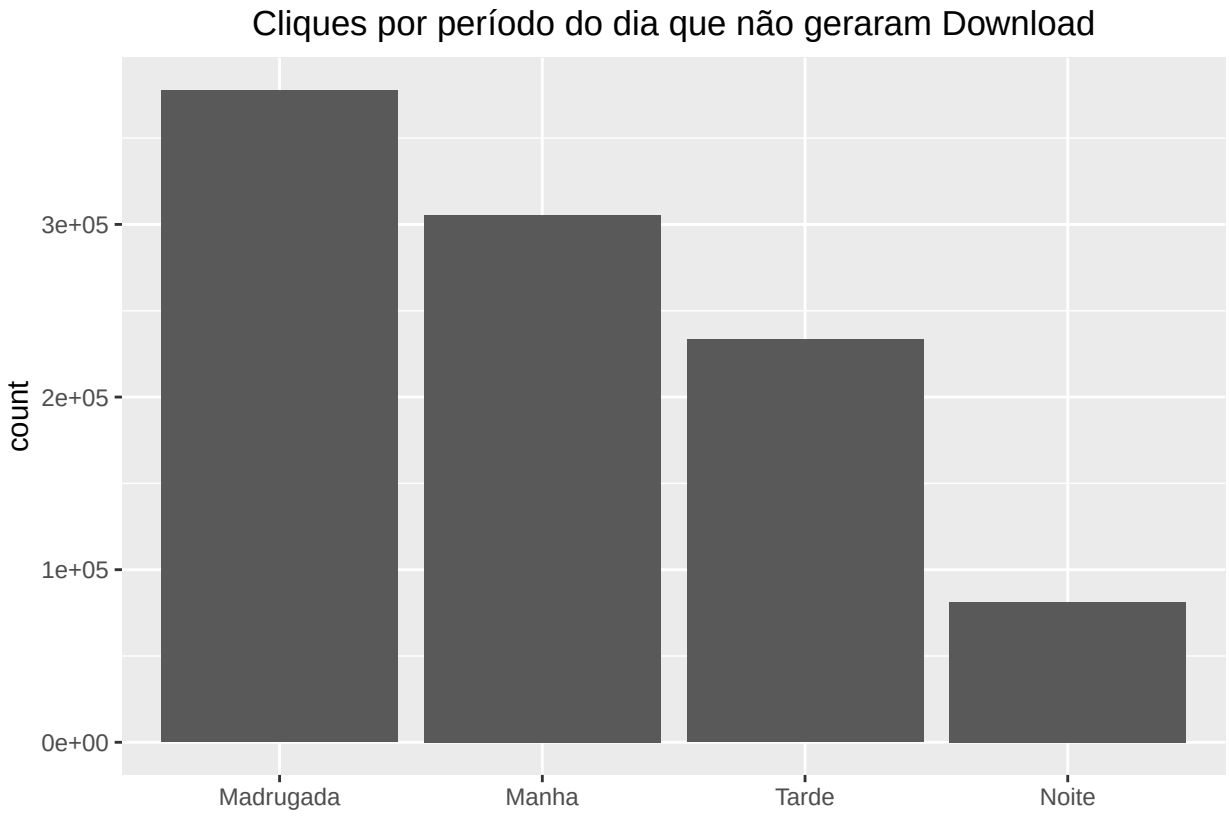
```
dados %>%
  ggplot(., aes(period)) + geom_bar() + ggtitle("Cliques por período do dia") + xlab("") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
dados %>% filter(is_attributed=="1") %>%
  ggplot(., aes(period)) + geom_bar() + ggtitle("Cliques por período do dia que geraram Download") + xla
  theme(plot.title = element_text(hjust = 0.5))
```




```
dados %>% filter(is_attributed=="0") %>%
  ggplot(., aes(period)) + geom_bar() + ggtitle("Cliques por período do dia que não geraram Download") +
  theme(plot.title = element_text(hjust = 0.5))
```



O mesmo pode se dizer olhando para o período do dia, com uma leve diferença no período da tarde e noite, para os casos em que o clique no link gerou download no app (houveram menos cliques em relação aos outros períodos do dia).

Feature Selection

Vamos criar um novo dataset apenas com as variáveis que a princípio iremos usar no modelo:

```
dados_mod <- dados %>%
  select(ip_cut, app_chr, device_chr, channel_chr, weekday, period, is_attributed)

# Precisamos transformar as variáveis do tipo character em fator para posterior balanceamento de
#classes.
dados_mod <- dados_mod %>% mutate_if(is.character, as.factor)
str(dados_mod)
```

```
## 'data.frame':    1000000 obs. of  7 variables:
##  $ ip_cut       : Factor w/ 10 levels "L01","L02","L03",..: 4 1 4 2 1 2 3 1 9 1 ...
##  $ app_chr      : Factor w/ 292 levels "0","1","10","100",..: 94 13 94 22 282 2 282 271 239 168 ...
##  $ device_chr   : Factor w/ 405 levels "0","1","100",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ channel_chr  : Factor w/ 172 levels "0","101","105",..: 53 165 135 73 20 25 52 30 121 10 ...
##  $ weekday      : Factor w/ 6 levels "domingo","segunda-feira",..: 4 3 3 4 5 3 3 2 5 3 ...
##  $ period       : Factor w/ 4 levels "Madrugada","Manha",..: 1 1 3 2 1 2 1 3 2 1 ...
##  $ is_attributed: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(dados_mod)
```

```
##   ip_cut app_chr device_chr channel_chr      weekday    period is_attributed
## 1    L04       2          1         219 quarta-feira Madrugada             0
## 2    L01      11          1         487  terça-feira Madrugada             0
## 3    L04       2          1         435  terça-feira     Tarde             0
## 4    L02      12          1         265 quarta-feira     Manha             0
## 5    L01       9          1         127 quinta-feira Madrugada             0
## 6    L02       1          1         134  terça-feira     Manha             0
```

```
# Vamos dividir os dados em treino e teste:

# Definindo 75% para os dados de treino:
prop_treino = 0.75
control = 1

# Função para gerar dados de treino e dados de teste:
splitData <- function(dataframe, seed = NULL, mult) {
  if (!is.null(seed)) set.seed(seed)
  index <- 1:nrow(dataframe)
  trainindex <- sample(index, trunc(length(index)*mult))
  trainset <- dataframe[trainindex, ]
  testset <- dataframe[-trainindex, ]
  list(trainset = trainset, testset = testset)
}

# Gerando dados de treino e de teste:
splits <- splitData(dados_mod, seed = control, mult = prop_treino)

# Separando os dados
dados_treino <- splits$trainset
dados_teste <- splits$testset

# Verificando o numero de linhas
nrow(dados_treino)
```

```
## [1] 750000
```

```
nrow(dados_teste)
```

```
## [1] 250000
```

Vamos aplicar o balanceamento de classes nos dados de treino e teste:

```
rose_treino <- ROSE(is_attributed ~ ., data = dados_treino, seed = control)$data

prop.table(table(rose_treino$is_attributed))
```

```
##
##         0         1
## 0.4997027 0.5002973
```

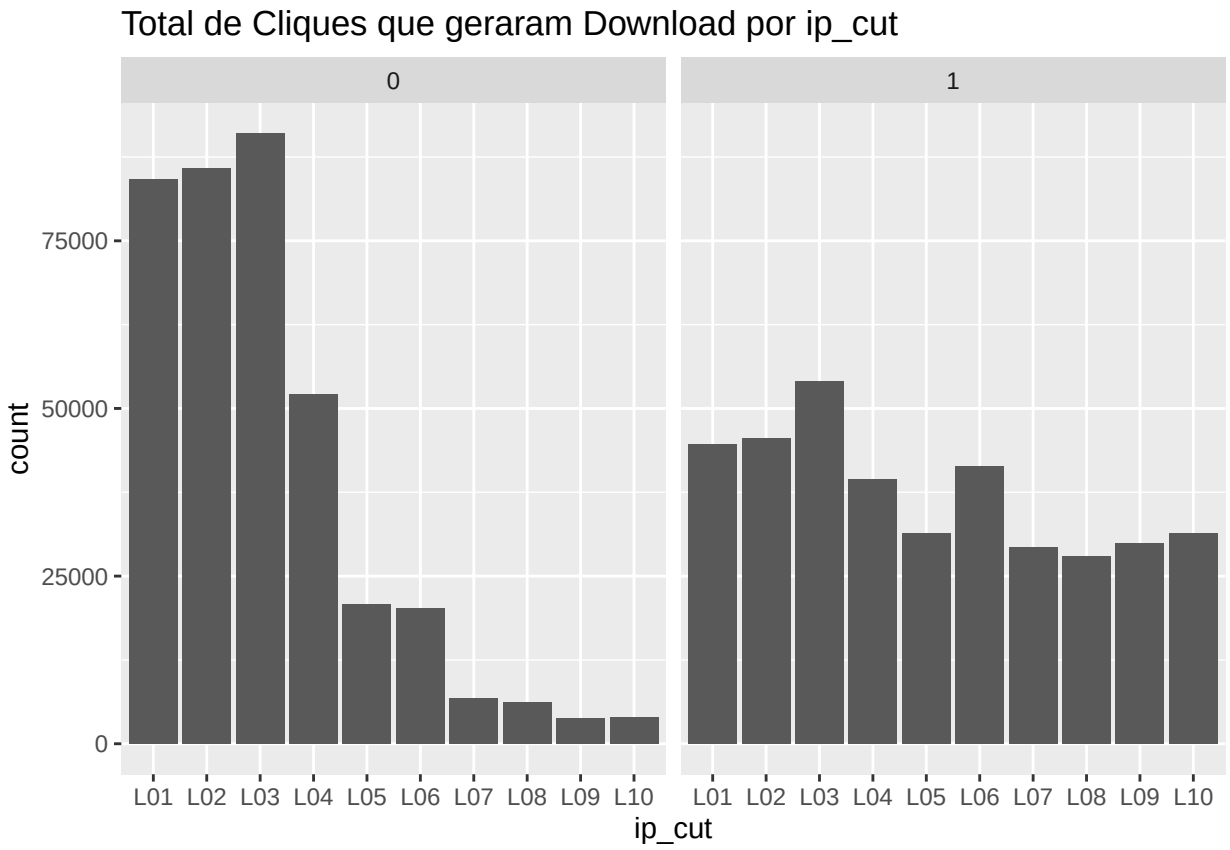
```
#A proporção entre as classes ficou de quase 50% para cada.

# Aplicando ROSE em dados de teste
rose_teste <- ROSE(is_attributed ~ ., data = dados_teste, seed = control)$data
prop.table(table(rose_teste$is_attributed))
```

```
##
```

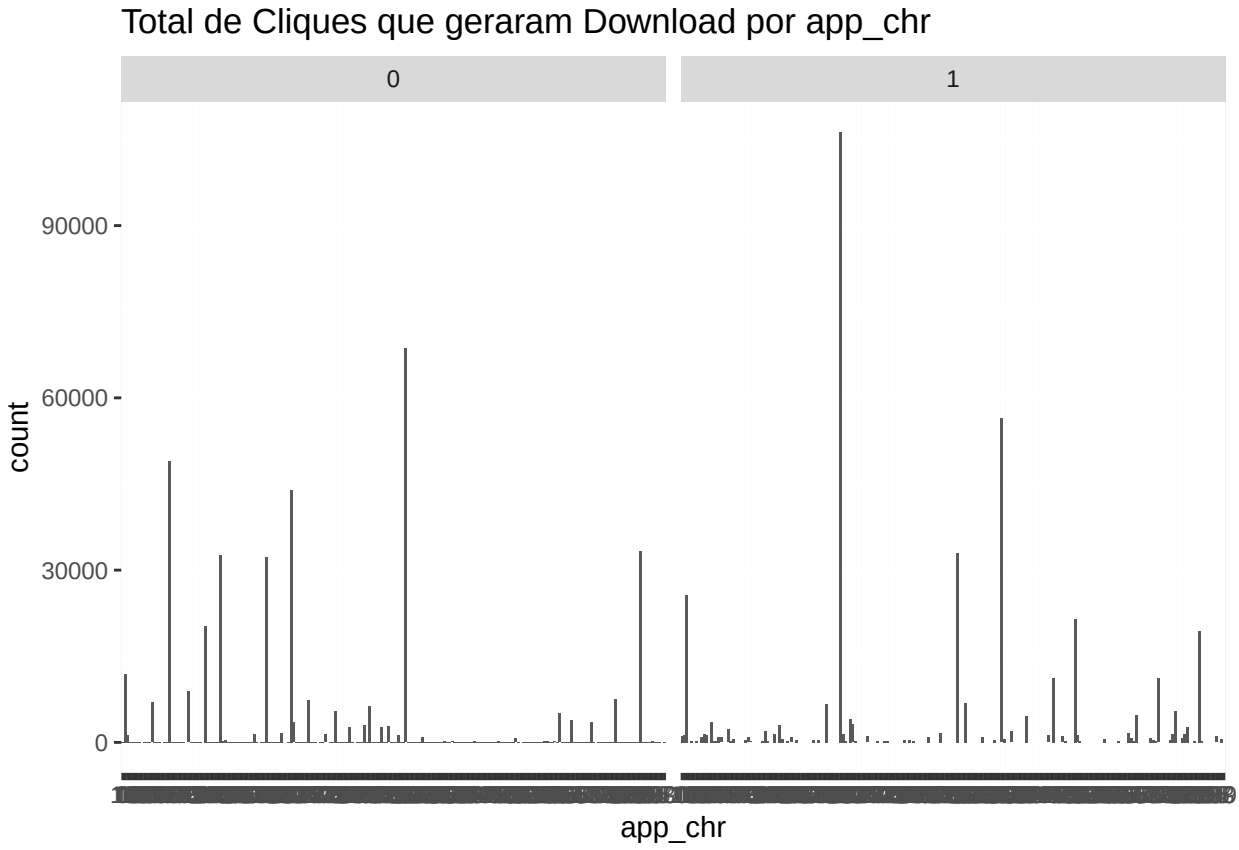
```
##      0      1
## 0.4996 0.5004
```

```
colNames <- (colnames(dados_mod[,1:6]))

# Plots usando ggplot2 nos dados pós balanceamento.
lapply(colNames, function(x){
  if(is.factor(rose_treino[,x])) {
    ggplot(rose_treino, aes_string(x)) +
      geom_bar() +
      facet_grid(. ~ is_attributed) +
      ggtitle(paste("Total de Cliques que geraram Download por",x))}})
```

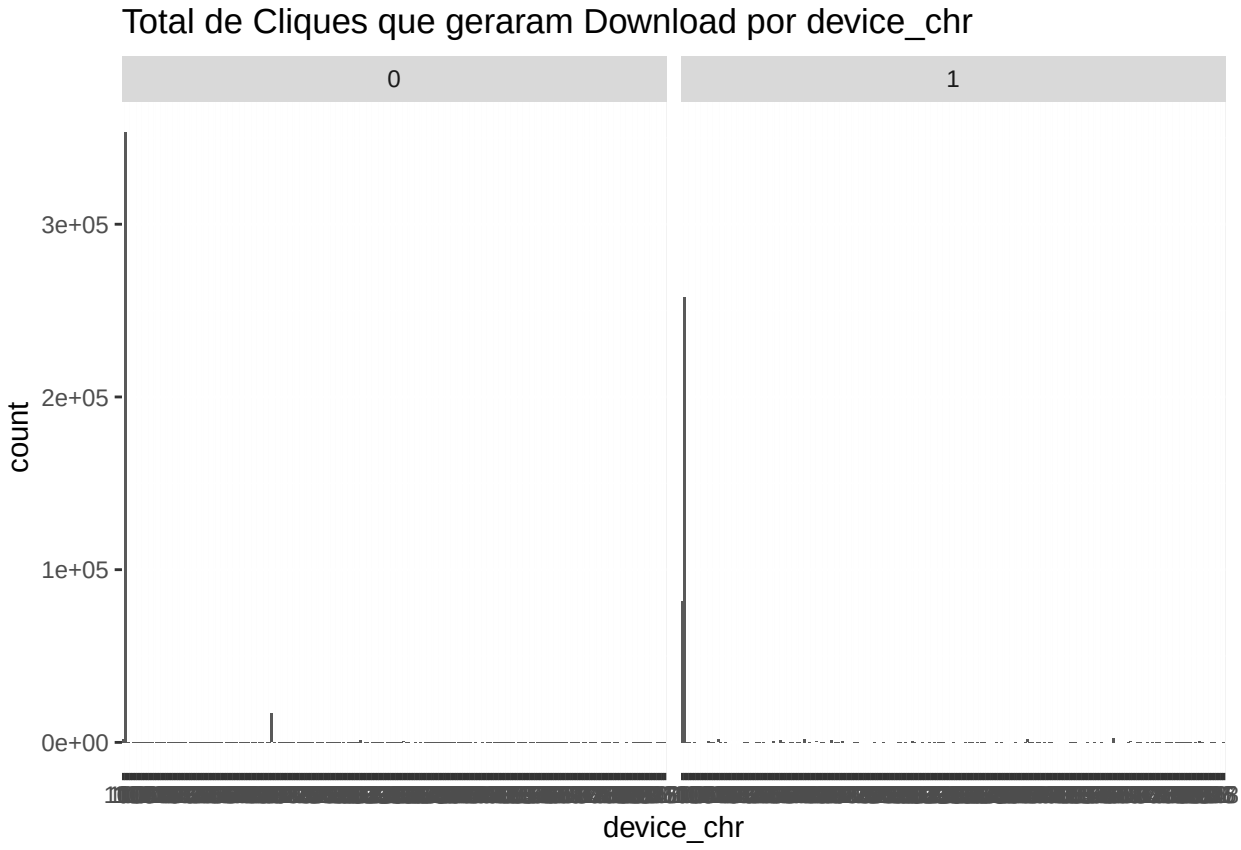
```
## [[1]]
```



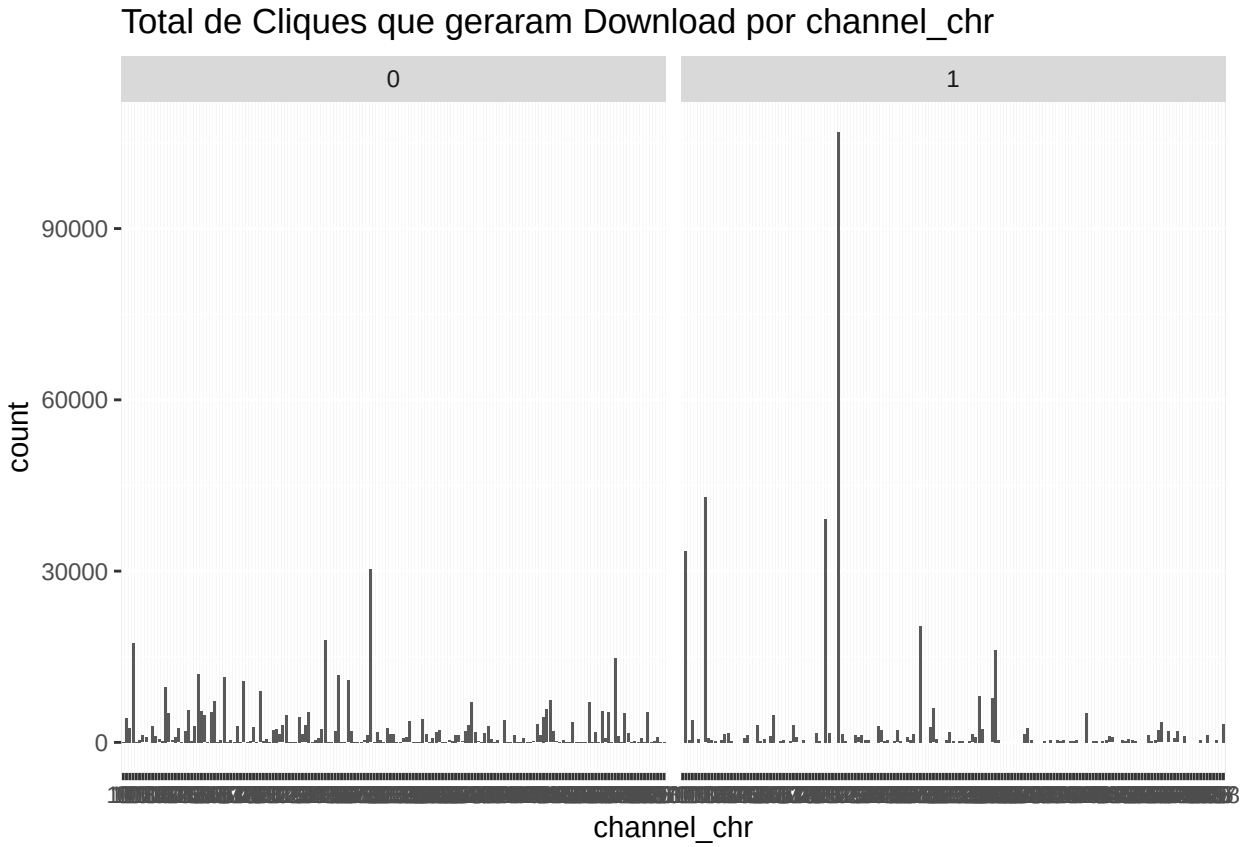

```
##
## [[2]]
```


Total de Cliques que geraram Download por app_chr
0
1
90000
60000
30000
0
count
app_chr

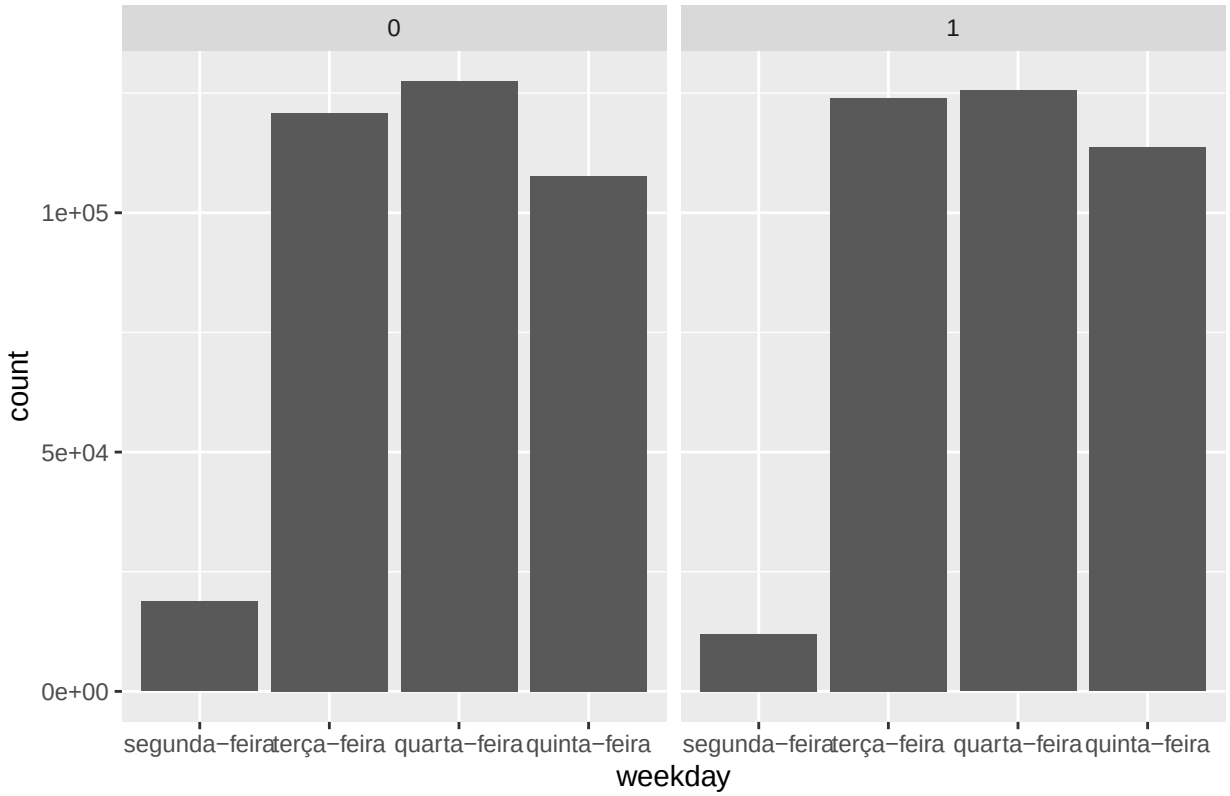
```
##
## [[3]]
```


Total de Cliques que geraram Download por device_chr
0
1
3e+05
2e+05
1e+05
0e+00
count
device_chr

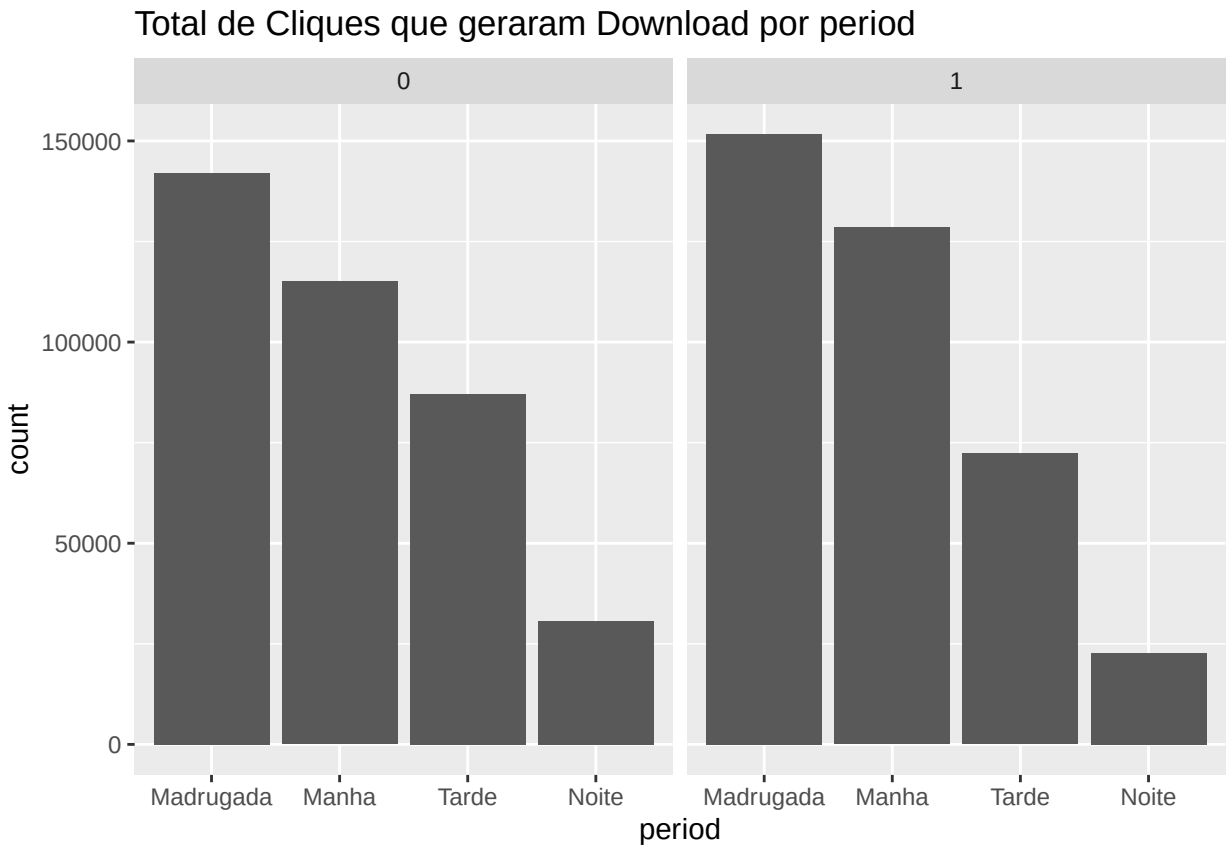
```
##
## [[4]]
```


Total de Cliques que geraram Download por channel_chr
0
1
90000
60000
30000
0
count
channel_chr

```
##
## [[5]]
```


Total de Cliques que geraram Download por weekday
0
1
1e+05
5e+04
0e+00
count
segunda−feira
terça−feira
quarta−feira
quinta−feira
weekday

```
##
## [[6]]
```



No geral, não houveram grandes modificações nas distribuições dos dados após o balanveamento de classes. Podemos verificar também que alguns apps estão mais propensos a serem baixados após o clique na propaganda, o mesmo vale para os dispositivos.

Criação e Avaliação do Modelo

Vamos criar uma primeira versão do modelo com Arvore de Decisão:

```
# Criando o modelo:
modelo1 <- C5.0(is_attributed ~ ., data = rose_treino, family = "binomial")

# Fazendo as previsões:
previsoes1 <- predict(modelo1, rose_teste)

# Verificando a matriz de confusão, e algumas métricas de performance nos dados de teste:
caret::confusionMatrix(rose_teste$is_attributed, previsoes1, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 120337   4563
##          1  33783  91317
##
##                Accuracy : 0.8466
##                  95% CI : (0.8452, 0.848)
```

```
##     No Information Rate : 0.6165
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6933
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9524
##             Specificity : 0.7808
##          Pos Pred Value : 0.7300
##          Neg Pred Value : 0.9635
##              Prevalence : 0.3835
##          Detection Rate : 0.3653
##    Detection Prevalence : 0.5004
##       Balanced Accuracy : 0.8666
##
##        'Positive' Class : 1
##
```

```
# Calculamos o Score AUC
roc.curve(rose_teste$is_attributed, previsoes1, plotit = F, col = "green", add.roc = F)
```

```
## Area under the curve (AUC): 0.847
```

Obtemos 84~85% de acurácia, com 94~95% de sensibilidade e 76~77% de especificidade. Já a área sob a curva (AUC) ficou em 84,5%. No geral, para uma primeira versão do modelo os resultados estão bons, entretanto podemos tentar otimizar o modelo para tentar equilibrar os valores de sensibildiade e especificidade, diminuindo assim a taxa de falsos positivos e aumentando o Score AUC.

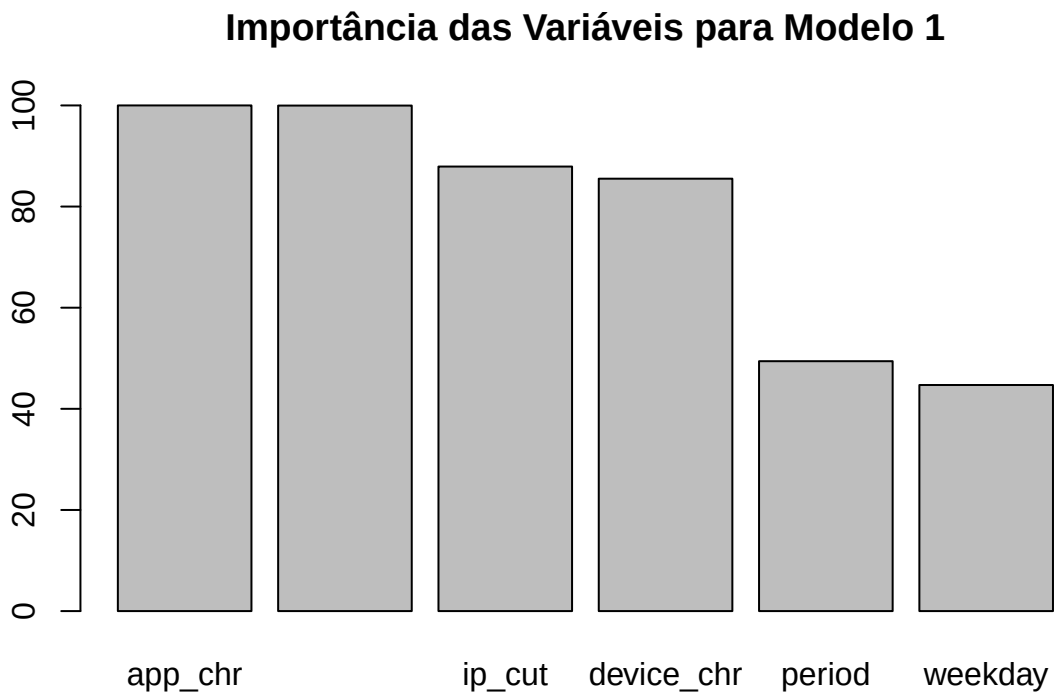
Otimização do Modelo

Vamos verificar a importancia de cada variável para o modelo criado, e assim criar um modelo apenas com as variáveis mais importantes.

```
importance1 <- varImp(modelo1)
importance1 <- cbind(variables = rownames(importance1), importance1)
rownames(importance1) <- 1:nrow(importance1)
importance1
```

```
##     variables Overall
## 1     app_chr  100.00
## 2 channel_chr   99.97
## 3      ip_cut   87.91
## 4  device_chr   85.51
## 5      period   49.42
## 6     weekday   44.71
```

```
barplot(importance1$Overall, names.arg = importance1$variables, main = "Importância das Variáveis para
```



Pela técnica utilizada, podemos notar que para o modelo criado, as variáveis "app_chr", "channel_chr", "ip_cut" e "device_chr" são as mais importantes, (com mais de 80% de Overall) vamos criar um modelo com apenas elas:

```
modelo2 <- C5.0(is_attributed ~ app_chr + channel_chr + device_chr + ip_cut, data = rose_treino, family
previsoes2 <- predict(modelo2, rose_teste)

caret::confusionMatrix(rose_teste$is_attributed, previsoes2, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 120535   4365
##          1  23984 101116
##
##                Accuracy : 0.8866
##                  95% CI : (0.8854, 0.8878)
##     No Information Rate : 0.5781
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7732
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9586
##             Specificity : 0.8340
```

```
##          Pos Pred Value : 0.8083
##          Neg Pred Value : 0.9651
##              Prevalence : 0.4219
##          Detection Rate : 0.4045
##    Detection Prevalence : 0.5004
##       Balanced Accuracy : 0.8963
##
##        'Positive' Class : 1
##
```

```
# Calculamos o Score AUC
roc.curve(rose_teste$is_attributed, previsoes2, plotit = F, col = "green", add.roc = F)
```

```
## Area under the curve (AUC): 0.887
```

Com a seleção de variáveis, conseguimos aumentar a acurácia do modelo de 88~90%, com uma sensibilidade de 94~96%, e especificidade de 83~86%, além do aumento do Score AUC de 88~90%.

Conclusão

Baseado no dataset utilizado, podemos concluir que o modelo 2 criado conseguiria prever com 90% de precisão se um app será baixado após o clique na propaganda ou não, os erros do modelo seriam em sua maioria falsos positivos (o modelo acusar que um app será baixado quando na verdade este não será), além disso, podemos concluir que alguns apps possuem mais chances de serem baixados do que outros (após o clique), assim como alguns apps possuem menos chances de serem baixados (possam estar sendo alvos de cliques fraudulentos). O mesmo vale para o ID do editor e a faixa de IP's. Com uma rápida consulta podemos descobrir quais são exatamente estes apps, dispositivos, etc.

```
dados_mod %>%
  filter(is_attributed == "0") %>%
  count(app_chr) %>%
  arrange(desc(n)) %>%
  head(10)
```

```
##    app_chr      n
## 1        3 183386
## 2       12 130485
## 3        2 116955
## 4        9  88251
## 5       15  86646
## 6       18  85367
## 7       14  54700
## 8        1  31254
## 9       13  23474
## 10       8  20106
```

Os apps de ID "3", "12" e "2" possuem alto número de ocorrências de cliques que não geraram download.

```
dados_mod %>%
  filter(is_attributed == "0") %>%
  count(device_chr) %>%
  arrange(desc(n)) %>%
  head(10)
```

```
##    device_chr      n
## 1           1 940960
## 2           2  44023
```

```
## 3           0   5093
## 4        3032   3806
## 5        3543   1496
## 6        3866    943
## 7          59    114
## 8           5     67
## 9          40     49
## 10          6     49
```

O dispositivo de ID “1” possui alto número de ocorrências de cliques que não geraram download.

```
dados_mod %>%
  filter(is_attributed == "0") %>%
  count(channel_chr) %>%
  arrange(desc(n)) %>%
  head(10)
```

```
##    channel_chr     n
## 1          280 81122
## 2          245 48076
## 3          107 45949
## 4          477 39080
## 5          134 31863
## 6          259 31284
## 7          153 30317
## 8          265 29397
## 9          178 28578
## 10         121 25375
```

O ID do publisher “280” possui alto número de ocorrências de cliques que não geraram download.