Google Analytics Customer Revenue Prediction EDA

Code

1 Introduction

Here is an Exploratory Data Analysis for the Google Analytics Customer Revenue Prediction competition within the R environment. For this EDA in the main we will use tidyverse (https://www.tidyverse.org/packages/) packages. Also for modelling we will use glmnet (https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html), xgboost (https://cran.r-project.org/web/packages/xgboost/vignettes/xgboostPresentation.html) and keras (https://keras.rstudio.com/) packages.

Our task is to build an algorithm that predicts the natural log of the sum of all transactions per user. Thus, for every user in the test set, the target is:

$$y_{user} = \sum_{i=1}^{n} transaction_{user_i}$$

$$target_{user} = ln(y_{user} + 1).$$

Submissions are scored on the root mean squared error, which is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2},$$

where $\hat{y}$ is the predicted revenue for a customer and y is the natural log of the actual revenue value.

Let’s prepare and have a look at the dataset.

2 Preparations

2.1 Load libraries 2.2 Load data

Here we load libraries for data wrangling and visualisation.

Hide

```
library(h2o)
library(caret)
library(lme4)
library(ggalluvial)
library(xgboost)
library(jsonlite)
library(lubridate)
library(knitr)
library(Rmisc)
library(scales)
library(countrycode)
library(highcharter)
library(glmnet)
library(keras)
library(forecast)
library(zoo)
library(magrittr)
library(tidyverse)
library(stringr)
library(forcats)
```

3 Peek at the dataset

3.1 General info

```
## Train set file size: 86330826 bytes
```

```
## Train set dimensions: 5694 13
```

```
## Observations: 5,694
## Variables: 13
## $ channelGrouping      <chr> "Organic Search", "Organic Search", "Orga...
## $ customDimensions     <chr> "[{'index': '4', 'value': 'North America'...
## $ date                 <int> 20171016, 20171016, 20171016, 20171016, 2...
## $ device               <chr> "{\"browser\": \"Safari\", \"browserVersi...
## $ fullVisitorId        <dbl> 5.200650e+18, 3.200002e+18, 9.066915e+18,...
## $ geoNetwork           <chr> "{\"continent\": \"Americas\", \"subConti...
## $ hits                 <chr> "[{'hitNumber': '1', 'time': '0', 'hour':...
## $ socialEngagementType <chr> "Not Socially Engaged", "Not Socially Eng...
## $ totals               <chr> "{\"visits\": \"1\", \"hits\": \"7\", \"p...
## $ trafficSource        <chr> "{\"campaign\": \"(not set)\", \"source\"...
## $ visitId              <int> 1508169977, 1508207702, 1508152302, 15081...
## $ visitNumber          <int> 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 3, 1, 5, 2,...
## $ visitStartTime       <int> 1508169977, 1508207702, 1508152302, 15081...
```

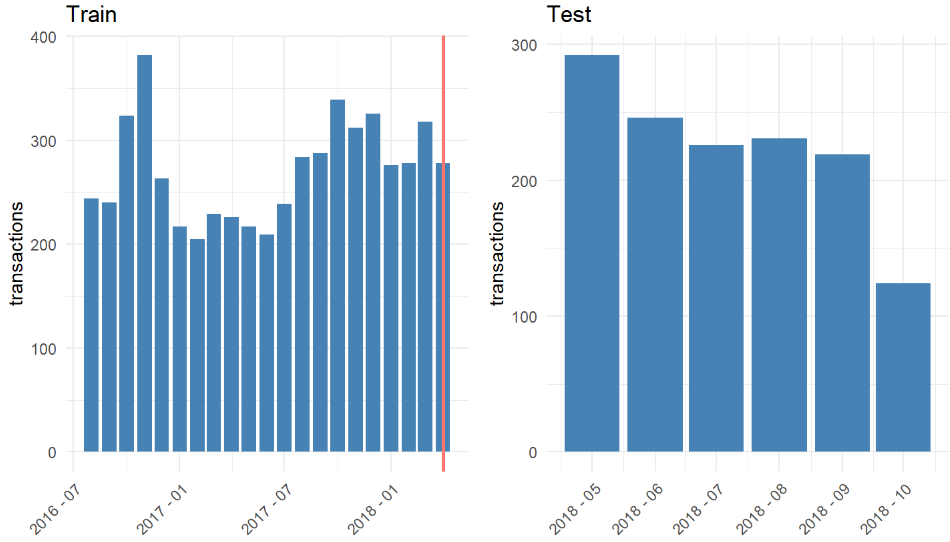
```
## Test set file size: 25175668 bytes
```

```
## Test set dimensions: 1338 13
```

```
## Observations: 1,338
## Variables: 13
## $ channelGrouping      <chr> "Organic Search", "Organic Search", "Orga...
## $ customDimensions     <chr> "[{'index': '4', 'value': 'South America'...
## $ date                 <int> 20180511, 20180511, 20180511, 20180511, 2...
## $ device               <chr> "{\"browser\": \"Chrome\", \"browserVersi...
## $ fullVisitorId        <dbl> 2.478621e+18, 7.711271e+17, 3.699492e+18,...
## $ geoNetwork           <chr> "{\"continent\": \"Americas\", \"subConti...
## $ hits                 <chr> "[{'hitNumber': '1', 'time': '0', 'hour':...
## $ socialEngagementType <chr> "Not Socially Engaged", "Not Socially Eng...
## $ totals               <chr> "{\"visits\": \"1\", \"hits\": \"13\", \"...
## $ trafficSource        <chr> "{\"referralPath\": \"(not set)\", \"camp...
## $ visitId              <int> 1526100413, 1526030584, 1526067726, 15260...
## $ visitNumber          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 4,...
## $ visitStartTime       <int> 1526100413, 1526030584, 1526067726, 15260...
```

3.2 Distribution of transaction dates

As shown in the figure, there are only a few of the transactions after Jan 2018 in the train set, because the rest is in the test set. It makes sense to create time-based splits for train/validation sets.

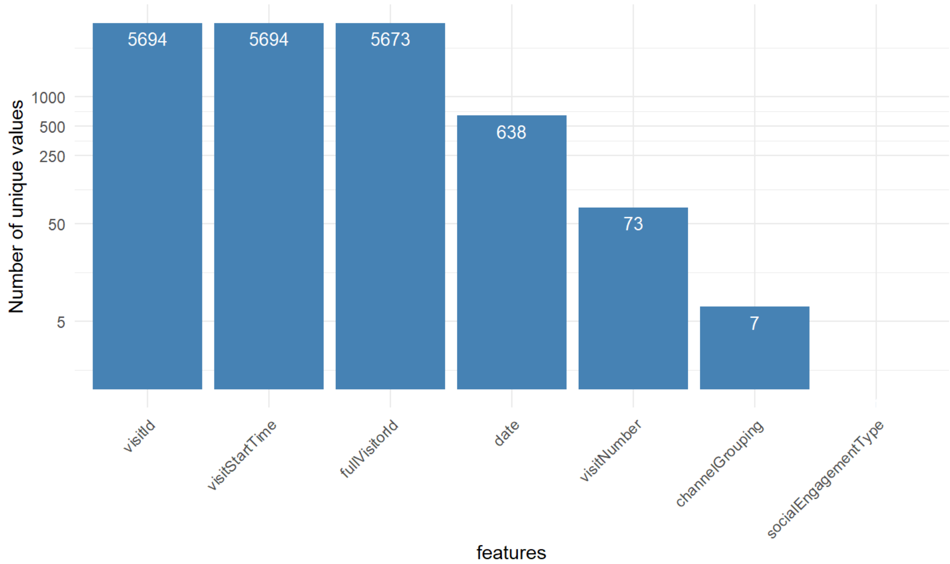


3.3 Dataset columns

There is a total of 13 features:

- **channelGrouping** - the channel via which the user came to the Store
- **customDimensions** - customer profile
- **date** - the date on which the user visited the Store
- **device** - the specifications for the device used to access the Store
- **fullVisitorId** - an unique identifier for each user of the Google Merchandise Store
- **geoNetwork** - this section contains information about the geography of the user
- **hits** - user actions during the session
- **socialEngagementType** - engagement type, either “Socially Engaged” or “Not Socially Engaged”
- **totals** - this section contains aggregate values across the session
- **trafficSource** - this section contains information about the Traffic Source from which the session originated
- **visitId** - an identifier for this session
- **visitNumber** - the session number for this user
- **visitStartTime** - the timestamp (POSIX).

Let’s have a look at counts of the simple features:



At least one column can be removed.

3.4 JSON data

Actually the columns **device**, **geoNetwork**, **trafficSource**, **totals**, **customDimensions**, **hits** contain data in JSON format. To parse it we can use jsonlite (https://cran.r-project.org/web/packages/jsonlite/) package but due to inconsistency of the data form the **trafficSource** column we need some addional tricks. I suggest to use the code, which is based on this idea (https://www.kaggle.com/mrlong/r-flatten-json-columns-to-make-single-data-frame):

Hide

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

parse <- . %>%
  bind_cols(flatten_json(.$customDimensions[0])) %>%
  bind_cols(flatten_json(.$hits[0])) %>%
  bind_cols(flatten_json(.$device)) %>%
  bind_cols(flatten_json(.$geoNetwork)) %>%
  bind_cols(flatten_json(.$trafficSource)) %>%
  bind_cols(flatten_json(.$totals)) %>%
  select(-customDimensions, -hits, -device, -geoNetwork, -trafficSource, -totals)
```

str_c function is a little faster than **paste()**.

Let's convert train and test sets to the tidy format:

Hide

```
tr <- parse(tr)
te <- parse(te)
```

3.5 Tidy datasets

3.5.1 Train 3.5.2 Test 3.5.3 Sample Submission

channelGrouping	date	fullVisitorId	socialEngagementType	visitId	visitNumber	visitStartTime	browser	browserVersion	browserSiz
Organic Search	20171016	5.200650e+18	Not Socially Engaged	1508169977	2	1508169977	Safari	not available in demo dataset	not availab in demo dataset
Organic Search	20171016	3.200002e+18	Not Socially Engaged	1508207702	1	1508207702	Chrome	not available in demo dataset	not availab in demo dataset

3.6 Train and test features sets intersection

Hide

```
# setdiff(names(tr), names(te))
tr %<>% select(-one_of("transactions"))
tr %<>% select(-one_of("totalTransactionRevenue"))
te %<>% select(-one_of("transactions"))
te %<>% select(-one_of("totalTransactionRevenue"))
te %<>% select(-one_of("transactionRevenue"))

setdiff(names(tr), names(te))
```

```
## [1] "transactionRevenue"
```

The test set lacks one columns, which is a target variable **transactionRevenue**.

3.7 Constant columns

Let's find constant columns:

Hide

```
fea_uniq_values <- sapply(tr, n_distinct)
(fea_del <- names(fea_uniq_values[fea_uniq_values == 1]))
```

```
##  [1] "socialEngagementType"
##  [2] "browserVersion"
##  [3] "browserSize"
##  [4] "operatingSystemVersion"
##  [5] "mobileDeviceBranding"
##  [6] "mobileDeviceModel"
##  [7] "mobileInputSelector"
##  [8] "mobileDeviceInfo"
##  [9] "mobileDeviceMarketingName"
## [10] "flashVersion"
## [11] "language"
## [12] "screenColors"
## [13] "screenResolution"
## [14] "cityId"
## [15] "latitude"
## [16] "longitude"
## [17] "networkLocation"
## [18] "adwordsClickInfo.criteriaParameters"
## [19] "visits"
```

Hide

```
tr %<>% select(-one_of(fea_del))
te %<>% select(-one_of(fea_del))
```

All these useless features we can safely remove.

3.8 Expanded train features

Hide

```
names(tr)
```

```
##  [1] "channelGrouping"                "date"
##  [3] "fullVisitorId"                  "visitId"
##  [5] "visitNumber"                    "visitStartTime"
##  [7] "browser"                        "operatingSystem"
##  [9] "isMobile"                       "deviceCategory"
## [11] "continent"                      "subContinent"
## [13] "country"                        "region"
## [15] "metro"                          "city"
## [17] "networkDomain"                  "campaign"
## [19] "source"                         "medium"
## [21] "keyword"                        "isTrueDirect"
## [23] "adContent"                      "referralPath"
## [25] "adwordsClickInfo.page"          "adwordsClickInfo.slot"
## [27] "adwordsClickInfo.gclId"         "adwordsClickInfo.adNetworkType"
## [29] "adwordsClickInfo.isVideoAd"     "hits1"
## [31] "pageviews"                      "timeOnSite"
## [33] "sessionQualityDim"              "newVisits"
## [35] "bounces"                        "transactionRevenue"
```

Hide

```
# names(te)
```

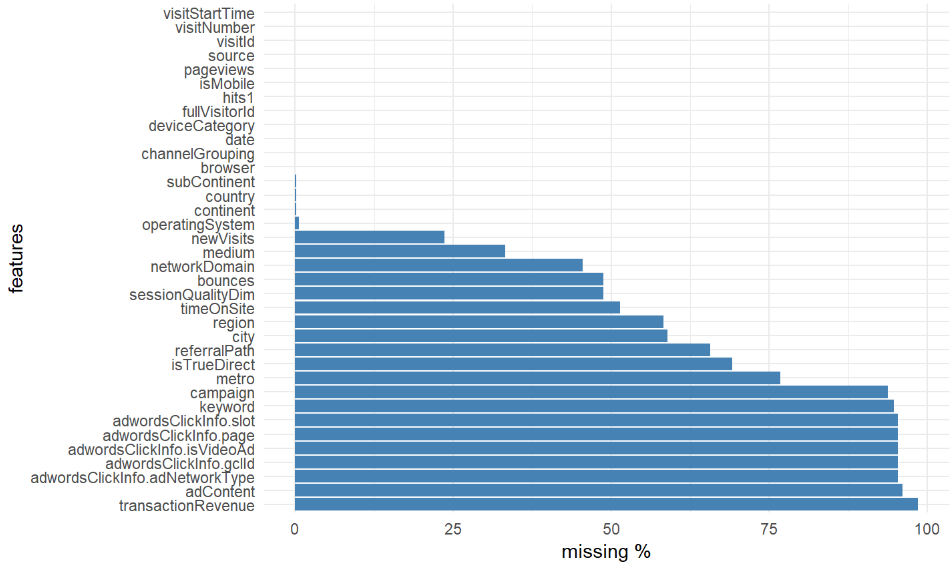
3.9 Missing values

After parsing of the JSON data we can observe many missing values in the data set. Let's find out how many missing values each feature has. We need to take into account that such values as "not available in demo dataset", "(not set)", "unknown.unknown", "(not provided)" can be treated as NA.

Hide

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

tr %<>% mutate_all(list(~ifelse(is_na_val(.), NA, .)))
te %<>% mutate_all(list(~ifelse(is_na_val(.), NA, .)))
```



There is a bunch of features missing nearly completely.

3.10 data transformations

We need to convert some features to their natural representation.

Hide

```
tr[0:3,]
```

```
## # A tibble: 3 x 36
##   channelGrouping   date fullVisitorId visitId visitNumber visitStartTime
##   <chr>            <int>         <dbl>   <int>       <int>          <int>
## 1 Organic Search  2.02e7       5.20e18  1.51e9           2     1508169977
## 2 Organic Search  2.02e7       3.20e18  1.51e9           1     1508207702
## 3 Organic Search  2.02e7       9.07e18  1.51e9           1     1508152302
## # ... with 30 more variables: browser <chr>, operatingSystem <chr>,
## #   isMobile <lgl>, deviceCategory <chr>, continent <chr>,
## #   subContinent <chr>, country <chr>, region <chr>, metro <chr>,
## #   city <chr>, networkDomain <chr>, campaign <chr>, source <chr>,
## #   medium <chr>, keyword <chr>, isTrueDirect <lgl>, adContent <chr>,
## #   referralPath <chr>, adwordsClickInfo.page <chr>,
## #   adwordsClickInfo.slot <chr>, adwordsClickInfo.gclId <chr>,
## #   adwordsClickInfo.adNetworkType <chr>,
## #   adwordsClickInfo.isVideoAd <lgl>, hits1 <chr>, pageviews <chr>,
## #   timeOnSite <chr>, sessionQualityDim <chr>, newVisits <chr>,
## #   bounces <chr>, transactionRevenue <chr>
```

Hide

```
te[0:3,]
```

```
## # A tibble: 3 x 35
##   channelGrouping   date fullVisitorId visitId visitNumber visitStartTime
##   <chr>            <int>         <dbl>   <int>       <int>          <int>
## 1 Organic Search  2.02e7       2.48e18  1.53e9           1     1526100413
## 2 Organic Search  2.02e7       7.71e17  1.53e9           1     1526030584
## 3 Organic Search  2.02e7       3.70e18  1.53e9           1     1526067726
## # ... with 29 more variables: browser <chr>, operatingSystem <chr>,
## #   isMobile <lgl>, deviceCategory <chr>, continent <chr>,
## #   subContinent <chr>, country <chr>, region <chr>, metro <chr>,
## #   city <chr>, networkDomain <chr>, referralPath <chr>, campaign <chr>,
## #   source <chr>, medium <chr>, keyword <chr>, adContent <chr>,
## #   isTrueDirect <lgl>, adwordsClickInfo.page <chr>,
## #   adwordsClickInfo.slot <chr>, adwordsClickInfo.gclId <chr>,
## #   adwordsClickInfo.adNetworkType <chr>,
## #   adwordsClickInfo.isVideoAd <lgl>, hits1 <chr>, pageviews <chr>,
## #   timeOnSite <chr>, newVisits <chr>, sessionQualityDim <chr>,
## #   bounces <chr>
```

Hide

```
tr %<>%
  mutate(date = ymd(date),
         hits = as.integer(hits1),
         pageviews = as.integer(pageviews),
         bounces = as.integer(bounces),
         newVisits = as.integer(newVisits),
         transactionRevenue = as.numeric(transactionRevenue))
tr %<>% select(-one_of("hits1"))

te %<>%
  mutate(date = ymd(date),
         hits = as.integer(hits1),
         pageviews = as.integer(pageviews),
         bounces = as.integer(bounces),
         newVisits = as.integer(newVisits))
te %<>% select(-one_of("hits1"))
```

3.11 Target variable

As a target variable we use **transactionRevenue** which is a sub-column of the **totals** JSON column. It looks like this variable is multiplied by 10^6.

Hide

```
y <- tr$transactionRevenue
tr$transactionRevenue <- NULL
summary(y)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
## 1.590e+06 2.888e+07 6.929e+07 1.413e+08 1.472e+08 2.548e+09      5611
```

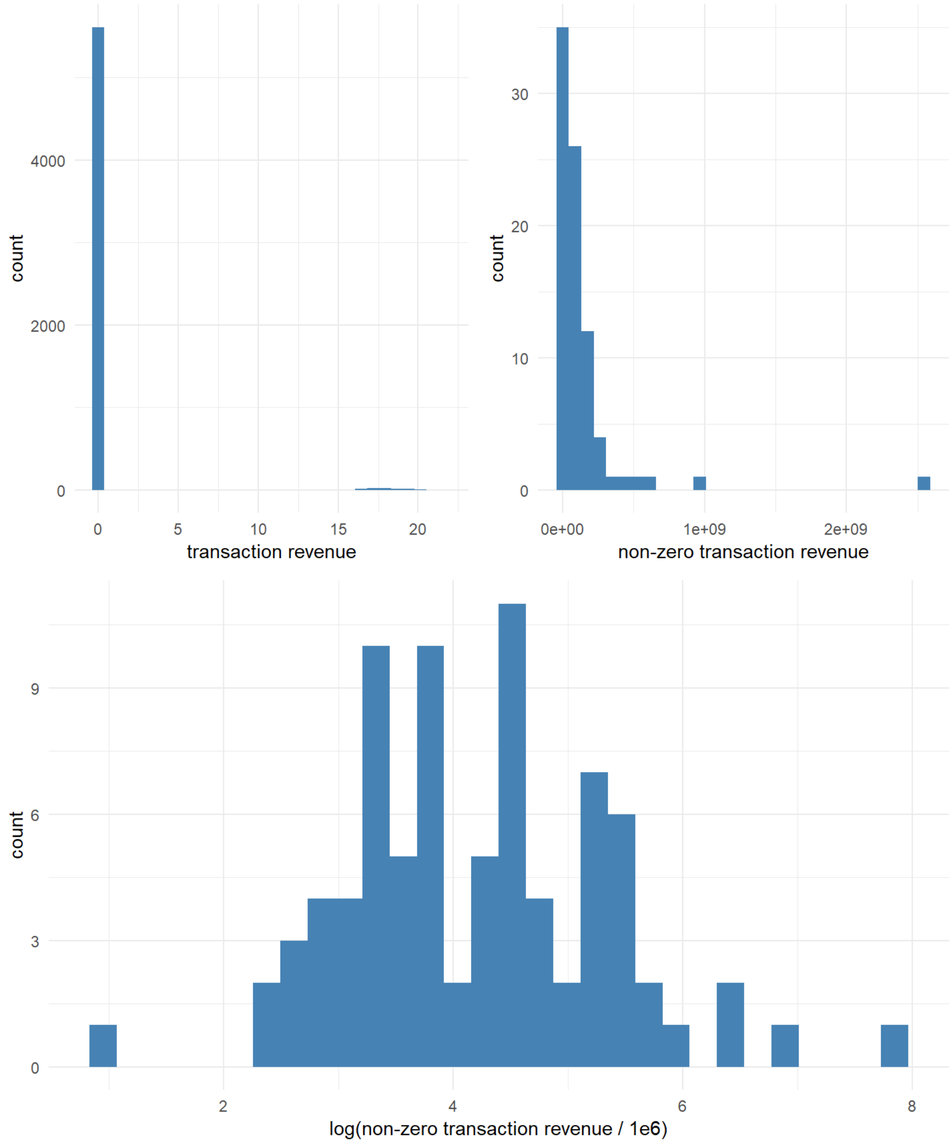
We can safely replace **NA** values with 0.

Hide

```
y[is.na(y)] <- 0
summary(y)
```

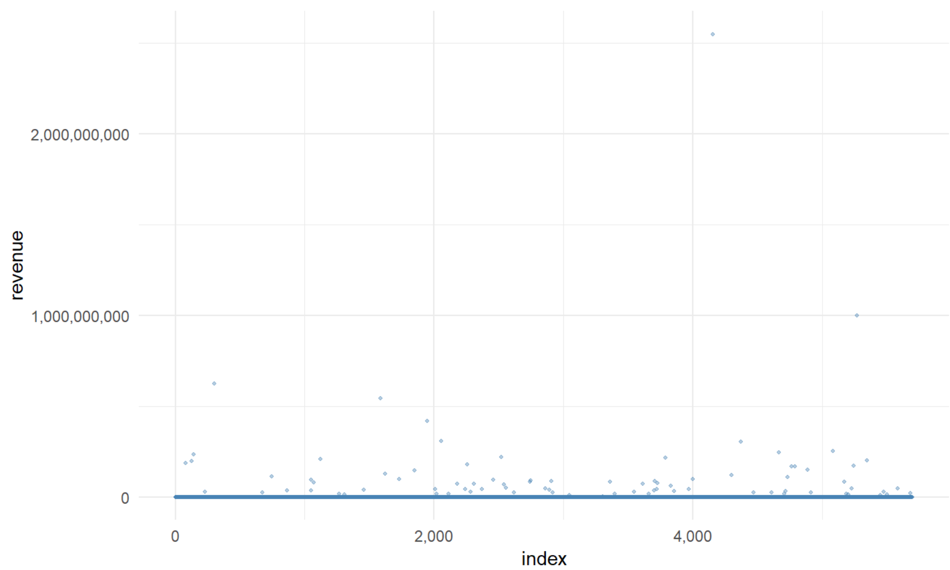
```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.000e+00 0.000e+00 0.000e+00 2.059e+06 0.000e+00 2.548e+09
```

```
## Warning: Calling `as_tibble()` on a vector is discouraged, because the behavior is likely to change in the futur
e. Use `tibble::enframe(name = NULL)` instead.
## This warning is displayed once per session.
```

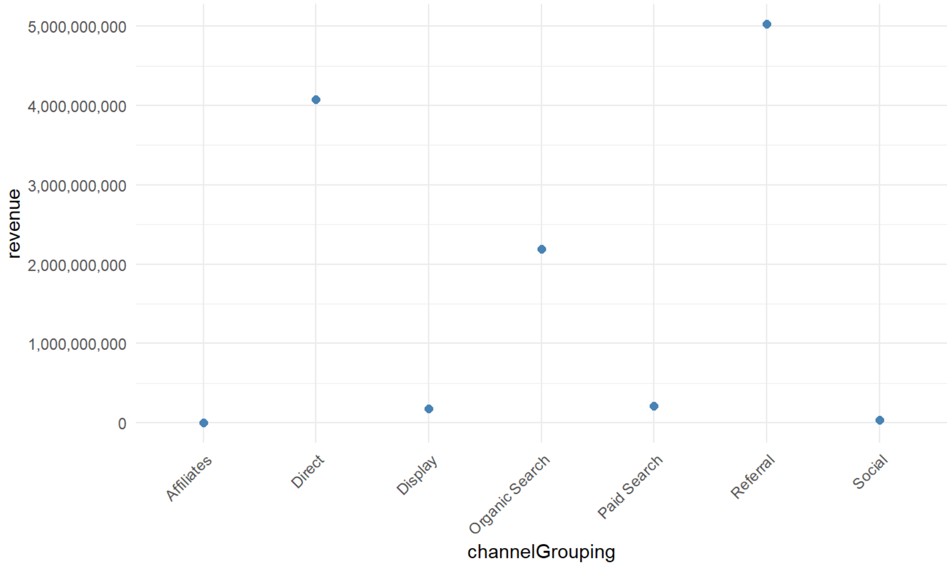


The target variable has a wide range of values. Its distribution is right-skewed. For modelling we will use log-transformed target.

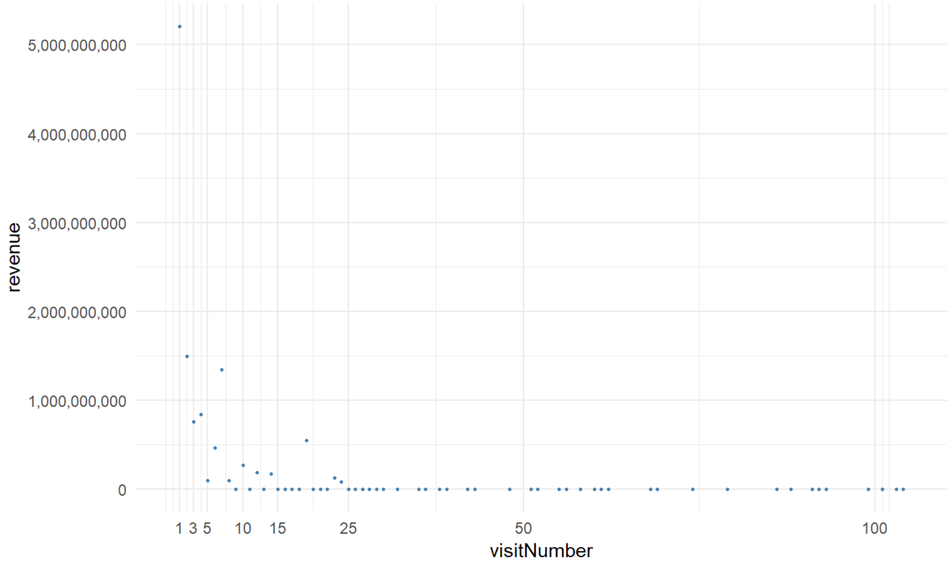
Only 1.46% of all transactions have non-zero revenue:



The next figure shows that users who came via **Affiliates** and **Social** channels do not generate revenue. The most profitable channel is **Referral**:

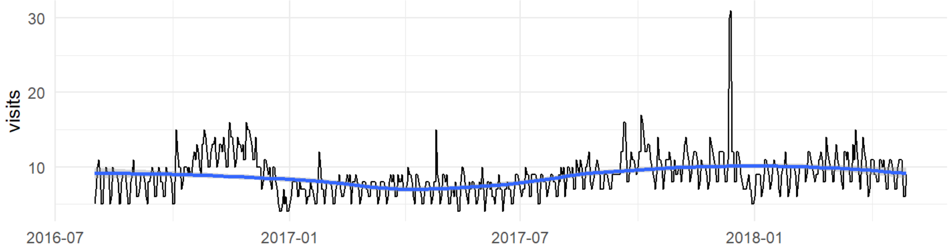


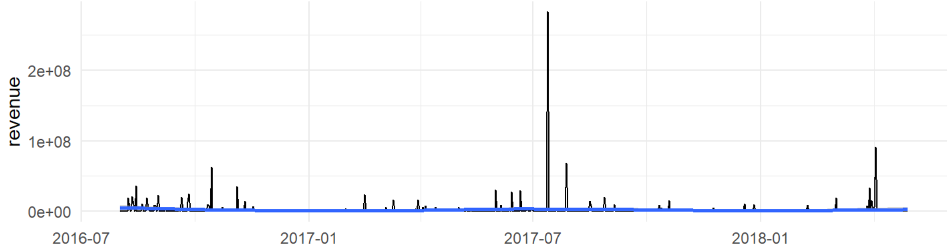
Also usually first/less visit users generate more total revenue:



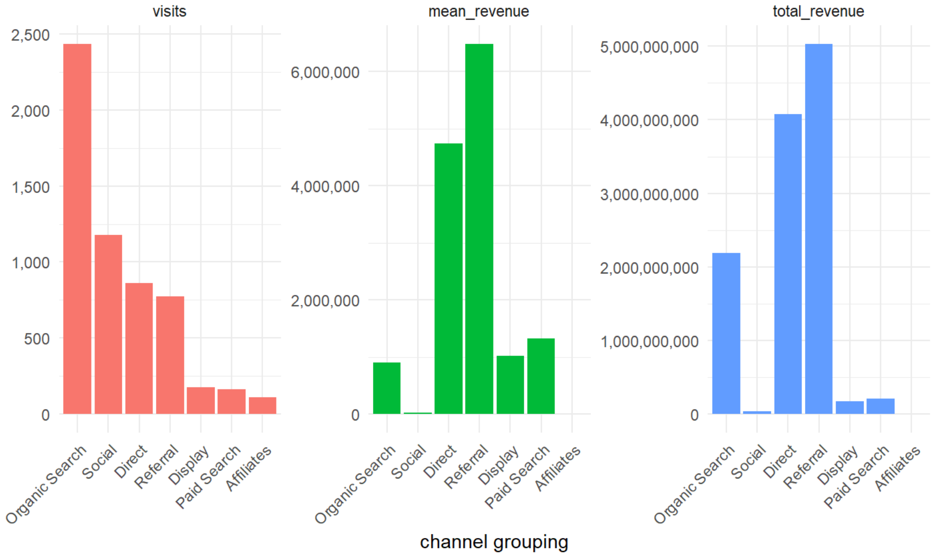
3.12 How target variable changes in time

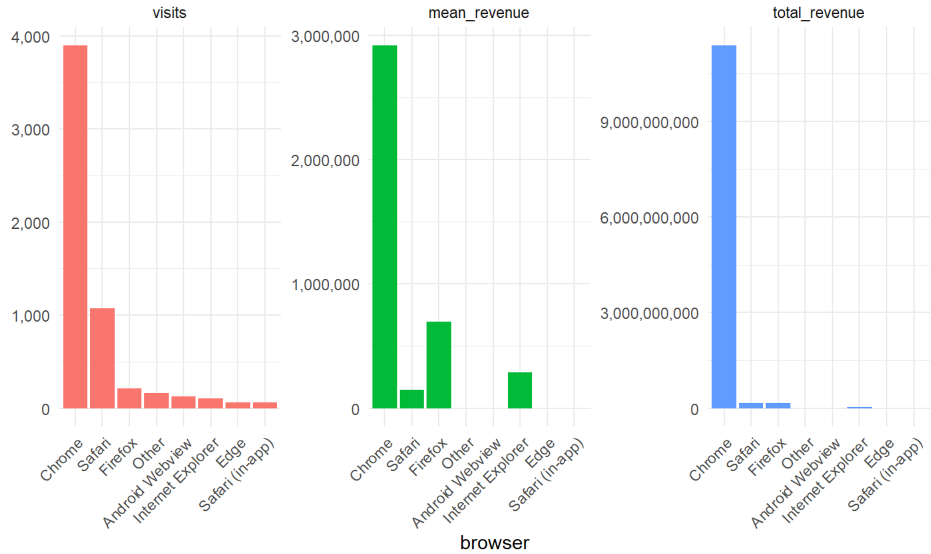
The revenue itself can be viewed as a timeseries. There seems to be a pattern of peaks.

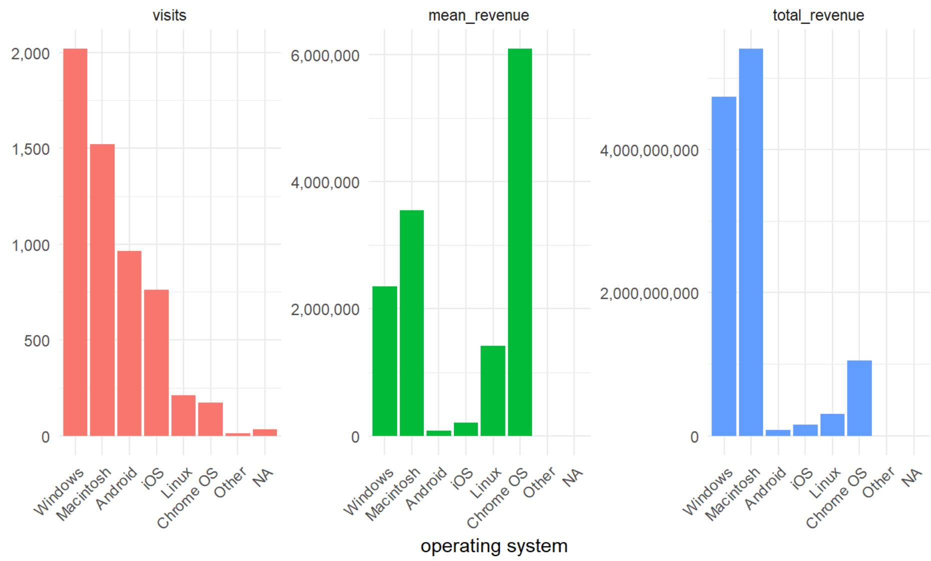


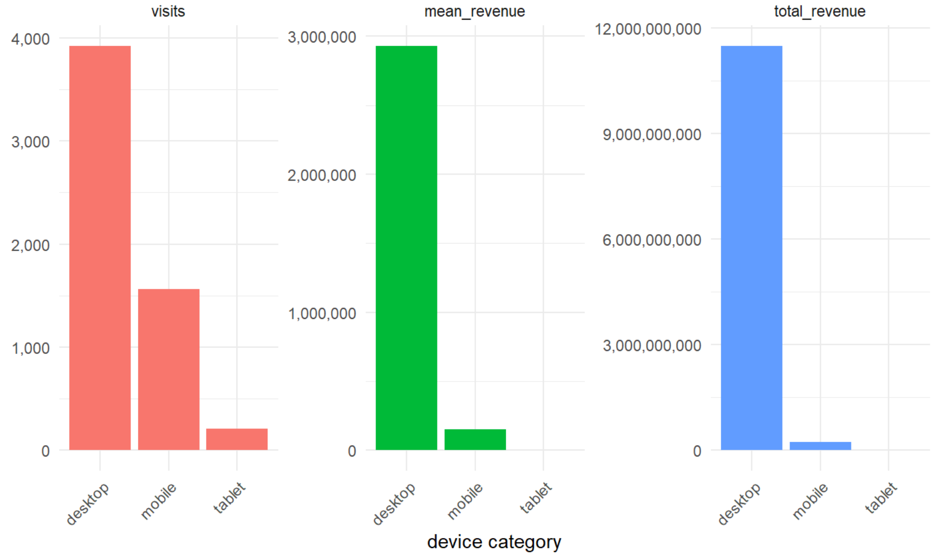


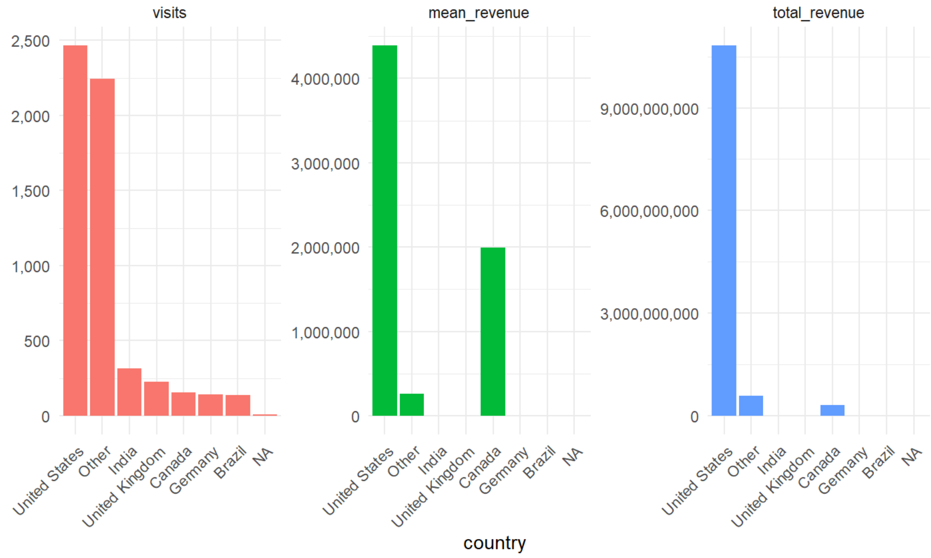
3.13 Distribution of visits and revenue by attributes

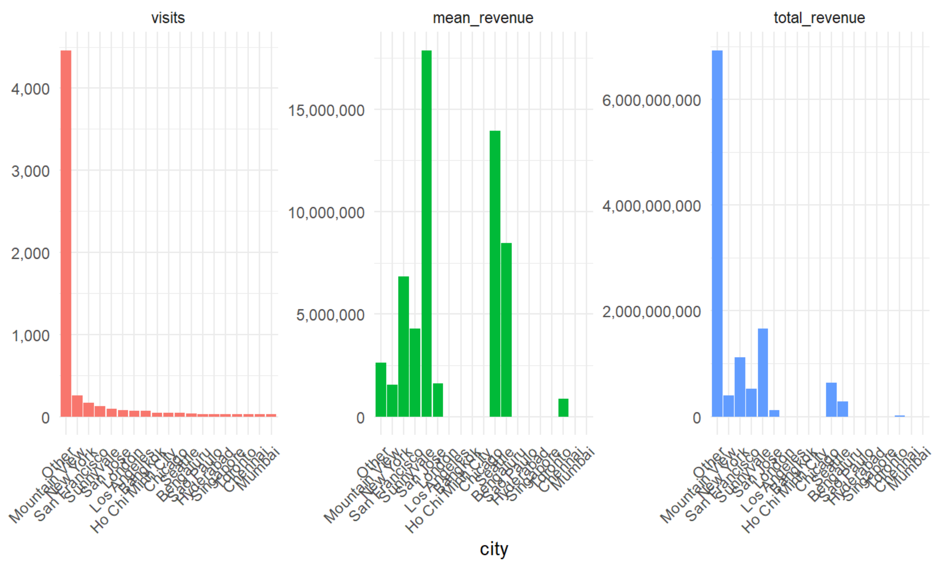

visits
mean_revenue
total_revenue
2,500
2,000
1,500
1,000
500
0
6,000,000
4,000,000
2,000,000
5,000,000,000
4,000,000,000
3,000,000,000
2,000,000,000
1,000,000,000
Organic Search
Social
Direct
Referral
Display
Paid Search
Affiliates
channel grouping

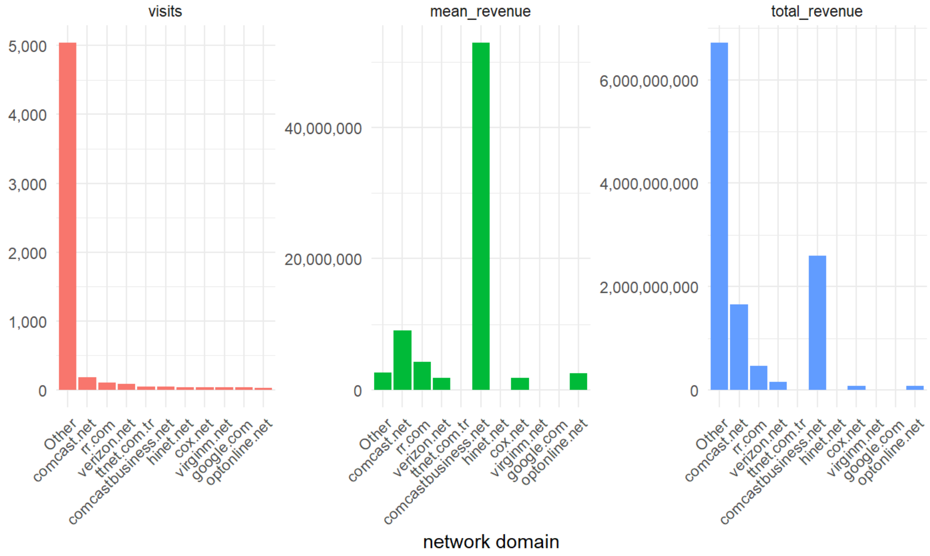

visits
mean_revenue
total_revenue
4,000
3,000
2,000
1,000
0
3,000,000
2,000,000
1,000,000
9,000,000,000
6,000,000,000
3,000,000,000
Chrome
Safari
Firefox
Other
Android Webview
Internet Explorer
Edge
Safari (in-app)
browser

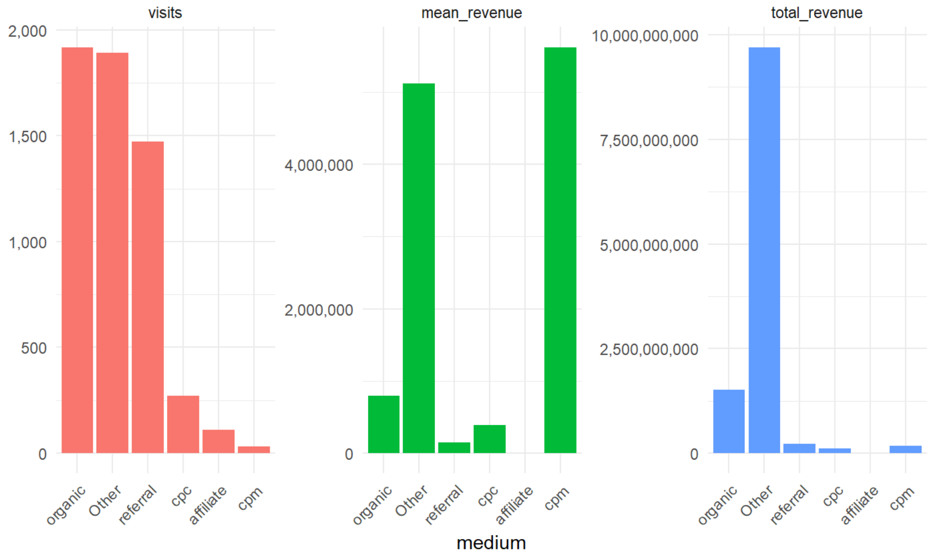

visits
mean_revenue
total_revenue
2,000
1,500
1,000
500
0
6,000,000
4,000,000
2,000,000
4,000,000,000
2,000,000,000
Windows
Macintosh
Android
iOS
Linux
Chrome OS
Other
NA
operating system


visits
mean_revenue
total_revenue
4,000
3,000
2,000
1,000
0
3,000,000
2,000,000
1,000,000
0
12,000,000,000
9,000,000,000
6,000,000,000
3,000,000,000
0
desktop
mobile
tablet
device category


visits
mean_revenue
total_revenue
2,500
2,000
1,500
1,000
500
0
4,000,000
3,000,000
2,000,000
1,000,000
0
9,000,000,000
6,000,000,000
3,000,000,000
0
United States
Other
India
United Kingdom
Canada
Germany
Brazil
NA
country


visits
mean_revenue
total_revenue
4,000
3,000
2,000
1,000
0
15,000,000
10,000,000
5,000,000
0
6,000,000,000
4,000,000,000
2,000,000,000
0
Other
Mumbai
city



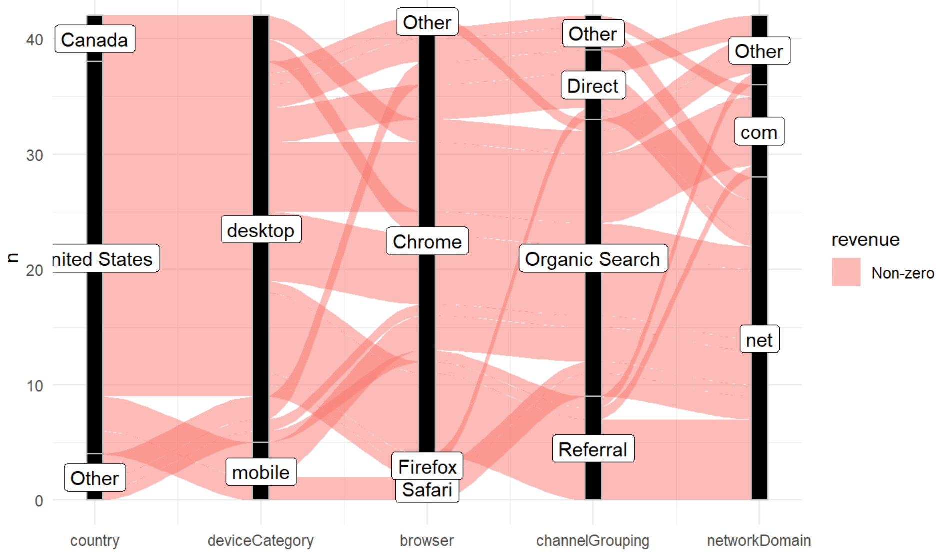


3.14 Findings:

- The most frequent channels are **OrganicSearch** and **Social**.
- Chrome is the most popular browser and its users produce the highest total revenue.
- Windows and MacOS are the most popular operating systems. It's interesting that ChromeOS users yield the highest mean revenue.
- Desktops are still in the ranks.
- The US users yield the most of the total revenue.
- Usually netwok domain is unknown.
- **organic** and **referral** are the most popular mediums.

4 Alluvial diagram

This kind of plot is useful for discovering of multi-feature interactions. The vertical size of each block is proportional to the frequency of the feature. The figure shows the flows for the case when revenue > 0:



Non-zero transaction revenue in the main is yielded by the flow US-desktop-Chrome-{OrganicSearch | Referral}-net.

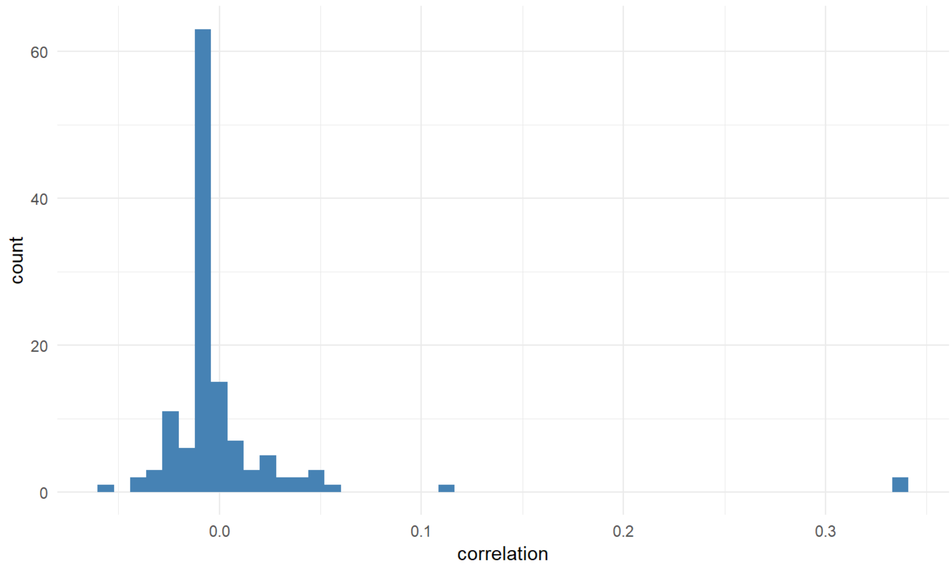
5 Correlations between revenue and features

Some features are categorical and we reencode them as OHE (with reduced set of levels). The ID columns are dropped.

Hide

```
m <- tr %>%
  mutate(year = year(date),
         month = month(date),
         day = day(date),
         isMobile = ifelse(isMobile, 1L, 0L),
         isTrueDirect = ifelse(isMobile, 1L, 0L)) %>%
  mutate_all(funs(ifelse(is.na(.), 0, .))) %>%
  select(-date, -fullVisitorId, -visitId) %>%
  mutate_if(is.character, factor) %>%
  mutate_if(is.factor, fct_lump, prop = 0.01) %>%
  model.matrix(~ . - 1, .) %>%
  cor(y) %>%
  data.table::as.data.table(keep.rownames=TRUE) %>%
  set_names("Feature", "rho") %>%
  arrange(-rho)

m %>%
  ggplot(aes(x = rho)) +
  geom_histogram(bins = 50, fill="steelblue") +
  labs(x = "correlation") +
  theme_minimal()
```



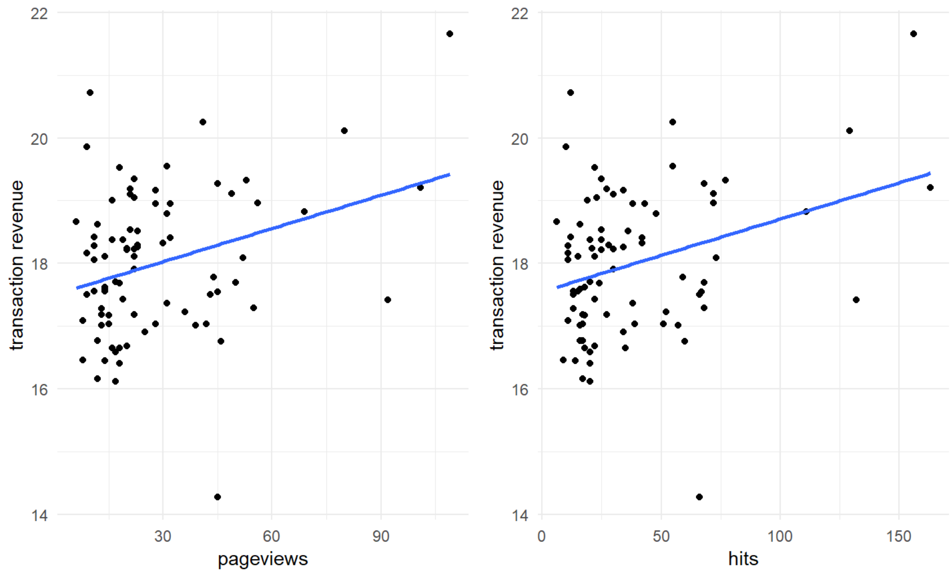
The values of the correlation coefficient are concentrated around zero, but there are several values bigger than 0.3:

Hide

```
m %>%
  filter(rho > 0.3) %>%
  kable()
```

Feature	rho
hits	0.3409752
pageviews	0.3370342

Let's visualize the relationship of the target variable with each of the correlated variables.



Here we observe weak positive relationship. Although, these features can play important role in a statistical model.

6 Revenue Predictive models

It is always useful to create several analtical models and compare them. Here for all models we use a preprocessed dataset:

Hide

```
grp_mean <- function(x, grp) ave(x, grp, FUN = function(x) mean(x, na.rm = TRUE))

idx <- tr$date < ymd("20171201")
id <- te[, "fullVisitorId"]
tri <- 1:nrow(tr)

tr_te <- tr %>%
  bind_rows(te) %>%
  mutate(year = year(date) %>% factor(),
         wday = wday(date) %>% factor(),
         hour = hour(as_datetime(visitStartTime)) %>% factor(),
         isMobile = ifelse(isMobile, 1L, 0L),
         isTrueDirect = ifelse(isTrueDirect, 1L, 0L),
         adwordsClickInfo.isVideoAd = ifelse(!adwordsClickInfo.isVideoAd, 0L, 1L)) %>%
  select(-date, -fullVisitorId, -visitId, -visitStartTime, -sessionQualityDim, -timeOnSite) %>%
  mutate_if(is.character, factor) %>%
  mutate(pageviews_mean_vn = grp_mean(pageviews, visitNumber),
         pageviews_mean_country = grp_mean(pageviews, country),
         pageviews_mean_city = grp_mean(pageviews, city),
         pageviews_mean_dom = grp_mean(pageviews, networkDomain),
         pageviews_mean_ref = grp_mean(pageviews, referralPath)) %T>%
  glimpse()
```

```
## Observations: 7,032
## Variables: 37
## $ channelGrouping              <fct> Organic Search, Organic Search,...
## $ visitNumber                  <int> 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 3...
## $ browser                      <fct> Safari, Chrome, Chrome, Chrome,...
## $ operatingSystem              <fct> iOS, Windows, Windows, Windows,...
## $ isMobile                     <int> 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0...
## $ deviceCategory               <fct> mobile, desktop, desktop, deskt...
## $ continent                    <fct> Americas, Americas, Europe, Asi...
## $ subContinent                 <fct> Northern America, Northern Amer...
## $ country                      <fct> Canada, Canada, Portugal, India...
## $ region                       <fct> NA, NA, NA, NA, NA, Riyadh Prov...
## $ metro                        <fct> NA, NA, NA, NA, NA, NA, New Yor...
## $ city                         <fct> NA, NA, NA, NA, NA, Riyadh, New...
## $ networkDomain                <fct> NA, NA, vodafone.pt, NA, as9105...
## $ campaign                     <fct> NA, NA, NA, NA, NA, 1000557 | G...
## $ source                       <fct> google, google, google, google,...
## $ medium                       <fct> organic, organic, organic, orga...
## $ keyword                      <fct> NA, NA, NA, NA, NA, (User verti...
## $ isTrueDirect                 <int> 1, NA, NA, NA, NA, NA, 1, NA, N...
## $ adContent                    <fct> NA, NA, NA, NA, NA, Google Merc...
## $ referralPath                 <fct> NA, NA, NA, NA, NA, NA, NA, NA,...
## $ adwordsClickInfo.page        <fct> NA, NA, NA, NA, NA, 1, NA, NA, ...
## $ adwordsClickInfo.slot        <fct> NA, NA, NA, NA, NA, RHS, NA, NA...
## $ adwordsClickInfo.gclId       <fct> NA, NA, NA, NA, NA, CL2-_8Pm9dY...
## $ adwordsClickInfo.adNetworkType <fct> NA, NA, NA, NA, NA, Content, NA...
## $ adwordsClickInfo.isVideoAd   <int> NA, NA, NA, NA, NA, 0, NA, NA, ...
## $ pageviews                    <int> 7, 14, 1, 3, 2, 1, 2, 1, 2, 1, ...
## $ newVisits                    <int> NA, 1, 1, 1, 1, NA, NA, 1, 1, 1...
## $ bounces                      <int> NA, NA, 1, NA, NA, 1, NA, 1, NA...
## $ hits                         <int> 7, 18, 1, 3, 2, 1, 2, 1, 2, 1, ...
## $ year                         <fct> 2017, 2017, 2017, 2017, 2017, 2...
## $ wday                         <fct> 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6...
## $ hour                         <fct> 16, 2, 11, 11, 21, 18, 21, 22, ...
## $ pageviews_mean_vn            <dbl> 4.493438, 3.476790, 3.476790, 3...
## $ pageviews_mean_country       <dbl> 5.619792, 5.619792, 3.370370, 2...
## $ pageviews_mean_city          <dbl> 7.000000, 14.000000, 1.000000, ...
## $ pageviews_mean_dom           <dbl> 7.000000, 14.000000, 5.500000, ...
## $ pageviews_mean_ref           <dbl> 7.000000, 14.000000, 1.000000, ...
```

Hide

```
# rm(tr, te, tr_ae, te_ae); invisible(gc())
```

6.1 GLMNET - Generalized linear model

For the **glmnet** model we need a model matrix. We replace **NA** values with zeros, rare factor levels are lumped:

Hide

```
tr_te_ohe <- tr_te %>%
  mutate_if(is.factor, fct_explicit_na) %>%
  mutate_if(is.numeric, funs(ifelse(is.na(.), 0L, .))) %>%
  mutate_if(is.factor, fct_lump, prop = 0.05) %>%
  select(-adwordsClickInfo.isVideoAd) %>%
  model.matrix(~.-1, .) %>%
  scale() %>%
  round(4)

X <- tr_te_ohe[tri, ]
X_test <- tr_te_ohe[-tri, ]
rm(tr_te_ohe); invisible(gc())
```

The next step is to create a cross-validated LASSO linear regression model:

Hide

```
m_glm <- cv.glmnet(X, log1p(y), alpha = 0, family="gaussian",
                   type.measure = "mse", nfolds = 5)
```

Finally, we create predictions of the LASSO model

Hide

```
pred_glm_tr <- predict(m_glm, X, s = "lambda.min") %>% c()
pred_glm <- predict(m_glm, X_test, s = "lambda.min") %>% c()
sub <- "glmnet_gs.csv"
# submit(pred_glm)
pred_glm[0:10]
```

```
##  [1]  0.74372371 -0.39662964 -0.12616842  0.06407258 -0.31940026
##  [6] -0.12545581 -0.13612285 -0.19296770  0.79437261 -0.10595989
```

Hide

```
rm(m_glm); invisible(gc())
```

6.2 XGB - Gradient boosting decision trees

At last, we are ready to create an XGB model. First, we need to preprocess the dataset. We don't care about **NA** values - XGB handles them by default:

Hide

```
tr_te_xgb <- tr_te %>%
  mutate_if(is.factor, as.integer) %>%
  glimpse()
```

```
## Observations: 7,032
## Variables: 37
## $ channelGrouping              <int> 4, 4, 4, 4, 4, 3, 2, 4, 4, 4, 1...
## $ visitNumber                  <int> 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 3...
## $ browser                      <int> 15, 5, 5, 5, 15, 5, 5, 15, 5, 9...
## $ operatingSystem              <int> 4, 10, 10, 10, 6, 1, 10, 4, 6, ...
## $ isMobile                     <int> 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0...
## $ deviceCategory               <int> 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1...
## $ continent                    <int> 2, 2, 4, 3, 4, 3, 2, 2, 2, 4, 4...
## $ subContinent                 <int> 11, 11, 17, 16, 12, 19, 11, 11,...
## $ country                      <int> 22, 22, 100, 52, 130, 107, 131,...
## $ region                       <int> NA, NA, NA, NA, NA, 147, 121, N...
## $ metro                        <int> NA, NA, NA, NA, NA, NA, 35, NA,...
## $ city                         <int> NA, NA, NA, NA, NA, 217, 176, N...
## $ networkDomain                <int> NA, NA, 1241, NA, 82, NA, NA, 1...
## $ campaign                     <int> NA, NA, NA, NA, NA, 4, NA, NA, ...
## $ source                       <int> 19, 19, 19, 19, 19, 19, 1, 19, ...
## $ medium                       <int> 4, 4, 4, 4, 4, 2, NA, 4, 4, 4, ...
## $ keyword                      <int> NA, NA, NA, NA, NA, 3, NA, NA, ...
## $ isTrueDirect                 <int> 1, NA, NA, NA, NA, NA, 1, NA, N...
## $ adContent                    <int> NA, NA, NA, NA, NA, 11, NA, NA,...
## $ referralPath                 <int> NA, NA, NA, NA, NA, NA, NA, NA,...
## $ adwordsClickInfo.page        <int> NA, NA, NA, NA, NA, 1, NA, NA, ...
## $ adwordsClickInfo.slot        <int> NA, NA, NA, NA, NA, 3, NA, NA, ...
## $ adwordsClickInfo.gclId       <int> NA, NA, NA, NA, NA, 156, NA, NA...
## $ adwordsClickInfo.adNetworkType <int> NA, NA, NA, NA, NA, 1, NA, NA, ...
## $ adwordsClickInfo.isVideoAd   <int> NA, NA, NA, NA, NA, 0, NA, NA, ...
## $ pageviews                    <int> 7, 14, 1, 3, 2, 1, 2, 1, 2, 1, ...
## $ newVisits                    <int> NA, 1, 1, 1, 1, NA, NA, 1, 1, 1...
## $ bounces                      <int> NA, NA, 1, NA, NA, 1, NA, 1, NA...
## $ hits                         <int> 7, 18, 1, 3, 2, 1, 2, 1, 2, 1, ...
## $ year                         <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1...
## $ wday                         <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6...
## $ hour                         <int> 17, 3, 12, 12, 22, 19, 22, 23, ...
## $ pageviews_mean_vn            <dbl> 4.493438, 3.476790, 3.476790, 3...
## $ pageviews_mean_country       <dbl> 5.619792, 5.619792, 3.370370, 2...
## $ pageviews_mean_city          <dbl> 7.000000, 14.000000, 1.000000, ...
## $ pageviews_mean_dom           <dbl> 7.000000, 14.000000, 5.500000, ...
## $ pageviews_mean_ref           <dbl> 7.000000, 14.000000, 1.000000, ...
```

Hide

```
rm(tr_te); invisible(gc())
```

Second, we create train, validation and test sets. We use time-based split:

Hide

```
dtest <- xgb.DMatrix(data = data.matrix(tr_te_xgb[-tri, ]))
tr_te_xgb <- tr_te_xgb[tri, ]
dtr <- xgb.DMatrix(data = data.matrix(tr_te_xgb[idx, ]), label = log1p(y[idx]))
dval <- xgb.DMatrix(data = data.matrix(tr_te_xgb[!idx, ]), label = log1p(y[!idx]))
dtrain <- xgb.DMatrix(data = data.matrix(tr_te_xgb), label = log1p(y))
cols <- colnames(tr_te_xgb)
rm(tr_te_xgb); invisible(gc)
```

The next step is to train the model:

Hide

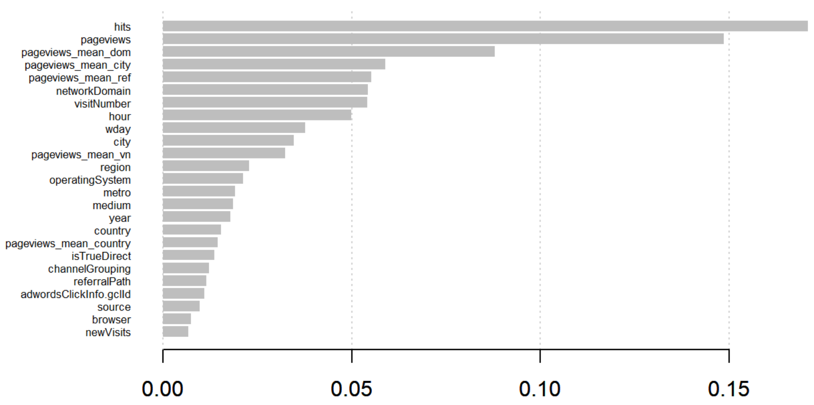
```
p <- list(objective = "reg:linear",
          booster = "gbtree",
          eval_metric = "rmse",
          nthread = 4,
          eta = 0.05,
          max_depth = 7,
          min_child_weight = 5,
          gamma = 0,
          subsample = 0.8,
          colsample_bytree = 0.7,
          colsample_bylevel = 0.6,
          nrounds = 2000)

set.seed(0)
m_xgb <- xgb.train(p, dtr, p$nrounds, list(val = dval), print_every_n = 100, early_stopping_rounds = 100)
```

```
## [1]  val-rmse:1.932156
## Will train until val_rmse hasn't improved in 100 rounds.
##
## [101]    val-rmse:1.770155
## Stopping. Best iteration:
## [55] val-rmse:1.731405
```

Hide

```
xgb.importance(cols, model = m_xgb) %>%
  xgb.plot.importance(top_n = 25)
```



Finally, we make predictions:

Hide

```
pred_xgb_tr <- predict(m_xgb, dtrain)
pred_xgb <- predict(m_xgb, dtest)
sub <- "xgb_gs.csv"
# submit(pred_xgb)
pred_xgb[0:10]
```

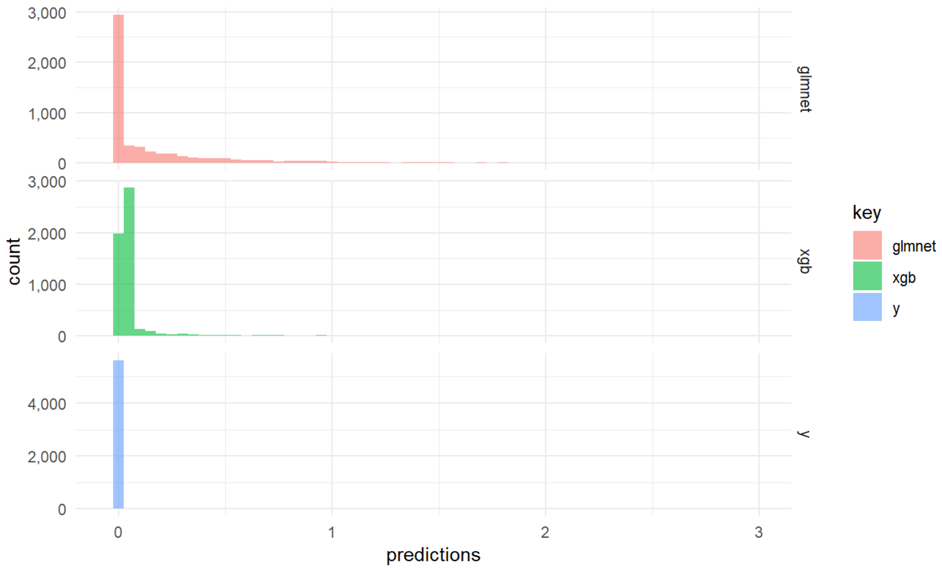
```
##  [1] 0.42952833 0.02526203 0.02526203 0.02508491 0.02526203 0.02575189
##  [7] 0.02151471 0.02526203 0.03711125 0.02526203
```

```
rm(dtr, dtrain, dval, dtest, m_xgb); invisible(gc)
```

As it was stated earlier, **hits** and **pageviews** plays important roles in the XGB model.

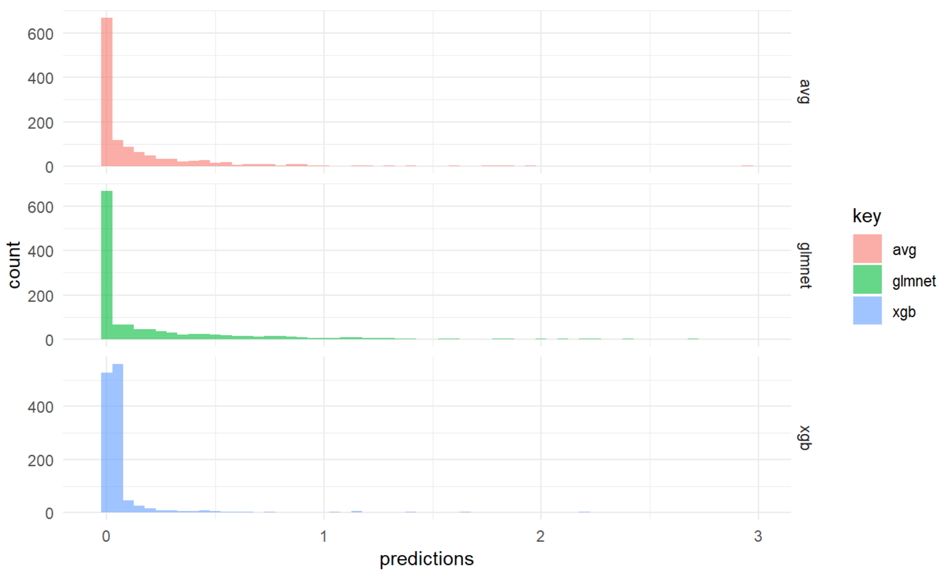
6.3 Distributions of predictions

Let's compare predictions for the train set:



As we can see the distributions of the predictions are quite different. The XGB model tends to produce more narrow interval - closer to the true distribution.

```
pred_avg <- log1p((expm1(pred_glm) + expm1(pred_xgb)) / 2)
sub <- "avg_gs.csv"
# submit(pred_xgb)
```



Distributions of the predictions for the test set differ much too, nevertheless after proper tuning of the models they can be useful for ensembling.

6.4 Output/Write results to csv:

```
tr_dollar <- y/(10^6)
te_dollar <- exp(pred_avg)

tr_actl <- cbind(tr, tr_dollar)
te_pred <- cbind(te, te_dollar)

write_csv(tr_actl, "../output/tr_actl.csv")
write_csv(te_pred, "../output/te_pred.csv")
```

The End…