Undersampling vs oversampling in a ML project with umbalanced target variable

Damiano Pincolini

2024-07-10

Table of contents

1. Preface

1.1. Project goal

This project aims to define a machine learning model able to solve a classification problem. Starting from a dataset containing features about a direct marketing campaigns (phone calls) of a Portuguese bank (see below for references), the goal is to predict if the client will subscribe a term deposit.

My specific interest will not be to find the best model based on a specific performance metric; instead, given a model, I want to tune its parameters in a proper way and, above all, handling the unbalanced distribution of the binary target variable ("yes", "no").

My attempt will be to proper evaluate how to fix the unbalanced distribution and, specifically, what between an undersampling or an oversampling strategy will fit the bill better.

I will download the dataset, wrangle it (for what necessary) and split into training and test set.

I will run explorative data analysis only on training set, so to avoid any data leakage and will take into account the results of this analysis to set a proprer pre-processing recipe.

During preprocessing, I will "re-balance" the target variable distribution via both undersampling and oversampling using different ratio between the two target classes and I will train a decision tree model (so to rank variable importance).

Finally, I will check what solution will produce the best result.

1.2. Loading packages

The packages I am going to use are tidyverse for manipulation and visualization, smartEDA for explorative analysis, tidymodels for modeling and some others in the command below.

```
pacman::p_load(tidyverse,
               SmartEDA,
               DescTools,
               factoextra,
               ggcorrplot,
               corrplot,
               tidymodels,
               themis,
               rpart.plot,
               vip,
               shapviz,
               knitr)
```

1.3. Loading data

The csv file contaning the dataset can be downladed at the https://archive.ics.uci.edu/dataset/222/bank+marketing page of the UC Irvine Machine Learning Repository.

I've named it "DataBank.csv" and saved in my desktop and then in R global environment as "DataOrigin". This file is available in this github page.

```
DataOrigin <- read_csv2("DataBank.csv")
```

1.4. Dataset content

Let's start with a first glance at DataOrigin structure and datatype.

```
ExpData(DataOrigin, type=1)
```

```
                                          Descriptions     Value
1                                   Sample size (nrow)     45211
2                              No. of variables (ncol)        17
3                      No. of numeric/interger variables         7
4                                No. of factor variables         0
5                                  No. of text variables        10
6                               No. of logical variables         0
7                            No. of identifier variables         0
8                                  No. of date variables         0
9               No. of zero variance variables (uniform)         0
10                 %. of variables having complete cases 100            (17)
```

```
11   %. of variables having >0% and <50% missing cases    0             (0)
12 %. of variables having >=50% and <90% missing cases    0
13          %. of variables having >=90% missing cases    0
```

The dataset has a 45211 rows and 17 columns (amongst which there's the target variable y).

There are 7 numeric and 10 categorical features and no missing case.

```
ExpData(DataOrigin, type=2)
```

```
   Index Variable_Name Variable_Type Sample_n Missing_Count Per_of_Missing
1      1           age       numeric    45211             0               0
2      2           job     character    45211             0               0
3      3       marital     character    45211             0               0
4      4     education     character    45211             0               0
5      5       default     character    45211             0               0
6      6       balance       numeric    45211             0               0
7      7       housing     character    45211             0               0
8      8          loan     character    45211             0               0
9      9       contact     character    45211             0               0
10    10           day       numeric    45211             0               0
11    11         month     character    45211             0               0
12    12      duration       numeric    45211             0               0
13    13      campaign       numeric    45211             0               0
14    14         pdays       numeric    45211             0               0
15    15      previous       numeric    45211             0               0
16    16      poutcome     character    45211             0               0
17    17             y     character    45211             0               0
   No_of_distinct_values
1                     77
2                     12
3                      3
4                      4
5                      2
6                   7168
7                      2
8                      2
9                      3
10                    31
11                    12
12                  1573
13                    48
```

```
14                   559
15                    41
16                     4
17                     2
```

```
str(DataOrigin)
```

```
spc_tbl_ [45,211 x 17] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ age       : num [1:45211] 58 44 33 47 33 35 28 42 58 43 ...
 $ job       : chr [1:45211] "management" "technician" "entrepreneur" "blue-collar" ...
 $ marital   : chr [1:45211] "married" "single" "married" "married" ...
 $ education: chr [1:45211] "tertiary" "secondary" "secondary" "unknown" ...
 $ default   : chr [1:45211] "no" "no" "no" "no" ...
 $ balance   : num [1:45211] 2143 29 2 1506 1 ...
 $ housing   : chr [1:45211] "yes" "yes" "yes" "yes" ...
 $ loan      : chr [1:45211] "no" "no" "yes" "no" ...
 $ contact   : chr [1:45211] "unknown" "unknown" "unknown" "unknown" ...
 $ day       : num [1:45211] 5 5 5 5 5 5 5 5 5 5 ...
 $ month     : chr [1:45211] "may" "may" "may" "may" ...
 $ duration : num [1:45211] 261 151 76 92 198 139 217 380 50 55 ...
 $ campaign : num [1:45211] 1 1 1 1 1 1 1 1 1 1 ...
 $ pdays     : num [1:45211] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
 $ previous : num [1:45211] 0 0 0 0 0 0 0 0 0 0 ...
 $ poutcome : chr [1:45211] "unknown" "unknown" "unknown" "unknown" ...
 $ y         : chr [1:45211] "no" "no" "no" "no" ...
 - attr(*, "spec")=
  .. cols(
  ..   age = col_double(),
  ..   job = col_character(),
  ..   marital = col_character(),
  ..   education = col_character(),
  ..   default = col_character(),
  ..   balance = col_double(),
  ..   housing = col_character(),
  ..   loan = col_character(),
  ..   contact = col_character(),
  ..   day = col_double(),
  ..   month = col_character(),
  ..   duration = col_double(),
  ..   campaign = col_double(),
  ..   pdays = col_double(),
  ..   previous = col_double(),
```

```
  ..   poutcome = col_character(),
  ..   y = col_character()
  .. )
 - attr(*, "problems")=<externalptr>
```

The dataset is composed of the following columns:

1. age
2. job: type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid', 'management','retired','self-employed','services','student','technician','unemployed', 'unknown')
3. marital: marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4. education: (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate', 'professional.course','university.degree','unknown')
5. default: has credit in default?
6. balance: average yearly balance
7. housing: has housing loan?
8. loan: has personal loan?
9. contact: contact communication type (categorical: 'cellular','telephone', 'unknown')
10. day: According to file documentation, this column is supposed to express the last contact day of the week, thus I would expect (since it's numeric) a range from 1 to 5 (monday to friday), while I see a range 1-31. I guess it represents the month's day.
11. month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
12. duration: last contact duration, in seconds (numeric). Please noote that dataset documentation states that this attribute highly affects the output target e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. According to authors' documentation, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
13. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

14. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; -1 means client was not previously contacted). Note 1: this feature seems to include two pieces of information. The first: whether there was a previous campaign (documentation lead me to the conclusion that it's a single campaign). The second: in the case of a previous campaign, the time passed after last contact. Note 2: I guess a new feature should be created (previous campaign: "yes" or "no").
15. previous: number of contacts performed before this campaign and for this client
16. poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent', 'success')
17. y (target variable): has the client subscribed a term deposit?

It may be useful to make the difference between previous campaign and current campaign clear.

Let's have a look to what follows.

```
DataOrigin|>
  group_by(pdays)|>
  summarize(count=n())|>
  mutate(perc=count/sum(count)*100)
```

```
# A tibble: 559 x 3
   pdays count     perc
   <dbl> <int>    <dbl>
 1    -1 36954 81.7
 2     1    15  0.0332
 3     2    37  0.0818
 4     3     1  0.00221
 5     4     2  0.00442
 6     5    11  0.0243
 7     6    10  0.0221
 8     7     7  0.0155
 9     8    25  0.0553
10     9    12  0.0265
# i 549 more rows
```

The 81,7% of customers was not involved in any previous campaign.

Then, specifically related to the remaining 18,3% it would be possible to do some binning in order to aggregate the number of days passed by after the client was last contacted from a previous campaign.

```
summary(DataOrigin)
```

```
      age            job              marital           education
 Min.   :18.00   Length:45211       Length:45211       Length:45211
 1st Qu.:33.00   Class :character   Class :character   Class :character
 Median :39.00   Mode  :character   Mode  :character   Mode  :character
 Mean   :40.94
 3rd Qu.:48.00
 Max.   :95.00
   default             balance         housing              loan
 Length:45211       Min.   : -8019   Length:45211       Length:45211
 Class :character   1st Qu.:    72   Class :character   Class :character
 Mode  :character   Median :   448   Mode  :character   Mode  :character
                    Mean   :  1362
                    3rd Qu.:  1428
                    Max.   :102127
   contact              day           month              duration
 Length:45211       Min.   : 1.00   Length:45211       Min.   :   0.0
 Class :character   1st Qu.: 8.00   Class :character   1st Qu.: 103.0
 Mode  :character   Median :16.00   Mode  :character   Median : 180.0
                    Mean   :15.81                      Mean   : 258.2
                    3rd Qu.:21.00                      3rd Qu.: 319.0
                    Max.   :31.00                      Max.   :4918.0
    campaign          pdays          previous          poutcome
 Min.   : 1.000   Min.   : -1.0   Min.   :  0.0000   Length:45211
 1st Qu.: 1.000   1st Qu.: -1.0   1st Qu.:  0.0000   Class :character
 Median : 2.000   Median : -1.0   Median :  0.0000   Mode  :character
 Mean   : 2.764   Mean   : 40.2   Mean   :  0.5803
 3rd Qu.: 3.000   3rd Qu.: -1.0   3rd Qu.:  0.0000
 Max.   :63.000   Max.   :871.0   Max.   :275.0000
      y
 Length:45211
 Class :character
 Mode  :character
```

Campaign column has a min value of 1 and a max value of 63. 63 contacts during one single campaign seem really too many! The third quantile is 3. I want to understand how frequent 63: could it be a mistake in data gathering?

Previous columns stands for the number of contacts performed before this campaign and for this client. With an average of 0.58 previous contact, max value is 275 contact.

How can this be possible?

Let's analyze campaign feature:

```
DataCampaignCount <- DataOrigin|>
  select(campaign)|>
  group_by(campaign)|>
  summarize(count=n())|>
  arrange(desc(campaign))|>
  mutate(countPerc=count/sum(count)*100,
         cum=cumsum(count),
         cumPerc=cum/sum(count)*100)
```

The relevance of number of contact bigger than 10 is pretty low. More than 97% of customers (1196 people) have been contacted 10 times or less during the campaign. All in all, I won't evaluate any transformation/elimination of this values.

To sum up, the to-do list for the data preparation/feature engineering is composed of the following actions to undertake:

1. The following features are categorical or numerical with defined values; thus they are supposed to be processed as factors: job, marital , education, default, housing, loan, contact, day, month, poutcome, y.

2. The current name of the feature "campaing" (which contains the number of contacts performed during this campaign) must be indicated with a more evocative name (I will pick "contCurrCamp").

3. The current name of the feature "previous" (which contains the number of contacts performed BEFORE this campaign) must be indicated with a more evocative name (I will pick "contPrevCamp").

4. I want to create a brand new feature that returns if this is the first campaign ever addressed to the customer could be useful ("firstCamp" with possible cases: "Y" or "N"). I will check if the information it contains is useful to the project purposes, or if it turns out to be redundant due to an execess of correlation with the predictor from which it is originated.

5. pdays: it is used -1 value to say that no contact has been made. -1 is thus a sort of categorical variable in a numerical column. A transformation into a categorical variable via binning (after isolating values = -1) should be useful for future uses. A name like "distDaysPrevCamp" should be slightly more intuitive.

6. "duration" variable is not supposed to be considered for ML classification model's purposes.

1.5. Feature engineering

```
Data <- DataOrigin|>
  mutate(job=as_factor(job),
         marital=as_factor(marital) ,
         education=as_factor(education),
         default=as_factor(default),
         housing=as_factor(housing),
         loan=as_factor(loan),
         contact=as_factor(contact),
         day=as_factor(day),
         month=as_factor(month),
         poutcome=as_factor(poutcome),
         y=as_factor(y),
         firstCamp=as_factor(ifelse(pdays==-1, "Y", "N")),
         distDaysPrevCamp=as_factor(case_when(pdays == -1 ~ "no prev campaign",
                                    pdays >30 & pdays <=60 ~ "30-60 dd",
                                    pdays >0 & pdays <=30 ~ "0-30 dd",
                                    pdays >60 & pdays <=90 ~ "60-90 dd",
                                    pdays > 90 ~ "over 90 dd")))|>
  select(-duration, -pdays)|>
  rename(contPrevCamp = previous,
         contCurrCamp = campaign)
```

1.6. Data partitioning

I'm now ready to split data set into train and test set. Of course, I want to keep the same proportion of target variable cases in both dataset, so I will use "strata" argument during the execution of initial_split() command.

```
set.seed(987, sample.kind = "Rounding")

DataSplit <- Data|>
  initial_split(prop = 0.80,
                strata = y)

DataTrain <- training(DataSplit)

DataTest <- testing(DataSplit)
```

I check whether the proportion of y cases are equal in train and test set as well as in the native dataset.

```
prop.table(table(Data$y))
```

```
       no       yes
0.8830152 0.1169848
```

```
prop.table(table(DataTrain$y))
```

```
       no       yes
0.8830181 0.1169819
```

```
prop.table(table(DataTest$y))
```

```
       no       yes
0.8830034 0.1169966
```

Everything looks fine, so dataset setup is completed. Since now and during the following EDA and model training phases, I won't use the test set at all, in order to avoid data leakage.

2. Explorative Data Analysis

2.1. Target feature analysis

First of all, it's worth recalling the frequencies of "yes" and "no" values which are assumed by the outcome variable "y".

```
prop.table(table(DataTrain$y))
```

```
       no       yes
0.8830181 0.1169819
```

As already seen during previous data quality step target value is pretty unbalance: almost 90% of cases belongs to "no" class.

2.2. Univariate analysis

For the extent of this paragraph, I will deeply rely on the smartEDA package.

2.2.1. Numerical features

Let's start with an overall summary of numerical features, after which I may want to get a closer look to single variable with graphics.

```
DataTrain|>
  ExpNumStat(by = "A",
             gp = "y",
             Qnt = c(0.25, 0.75),
             Nlim = 2,
             MesofShape = 2,
             Outlier = TRUE,
             round = 2)
```

```
         Vname Group    TN nNeg nZero  nPos NegInf PosInf NA_Value
1          age   All 36168    0     0 36168      0      0        0
2      balance   All 36168 2984  2843 30341      0      0        0
3 contCurrCamp   All 36168    0     0 36168      0      0        0
4 contPrevCamp   All 36168    0 29535  6633      0      0        0
  Per_of_Missing      sum   min    max    mean median      SD   CV  IQR
1              0  1480751    18     95   40.94     39   10.61 0.26   15
2              0 49765674 -8019 102127 1375.96    449 3110.14 2.26 1357
3              0    99977     1     63    2.76      2    3.11 1.12    2
4              0    21161     0    275    0.59      0    2.39 4.09    0
  Skewness Kurtosis 25%  75%  LB.25% UB.75% nOutliers
1     0.69     0.32  33   48    10.5   70.5       388
2     8.65   149.81  72 1429 -1963.5 3464.5      3808
3     4.97    40.66   1    3    -2.0    6.0      2438
4    45.57  4810.03   0    0     0.0    0.0      6633
```

Then, I want to detect possible outliers (for a better visualization I omit the sixth and the seventh row of the resulting dataframe which contain respectively every single lower and upper outlier cases).

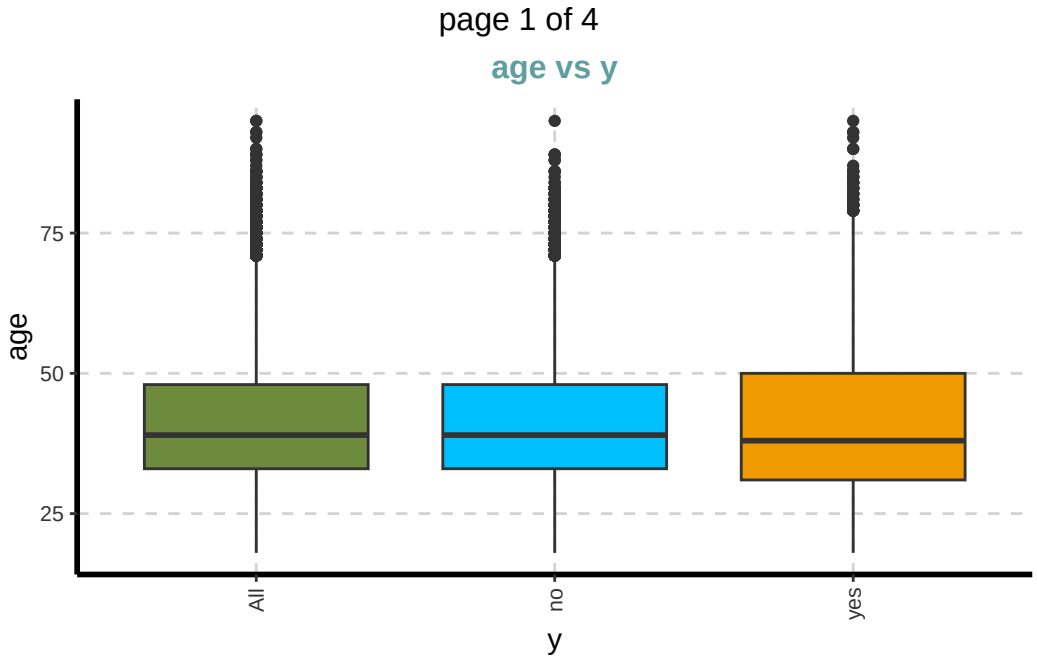
```
ExpOutliers(DataTrain,
            varlist = c("age", "balance", "contCurrCamp", "contPrevCamp"),
            method = "boxplot")$outlier_summary|>
  slice(-6,-7)
```

```
          Category   age balance contCurrCamp contPrevCamp
1 Lower cap : 0.05    27    -170            1            0
2 Upper cap : 0.95    59 5847.65            8            3
3      Lower bound  10.5 -1963.5           -2            0
4      Upper bound  70.5  3464.5            6            0
5  Num of outliers   388    3808         2438         6633
6      Mean before 40.94 1375.96         2.76         0.59
7       Mean after 40.55  640.67         2.13            0
8    Median before    39     449            2            0
9     Median after    39     348            2            0
```

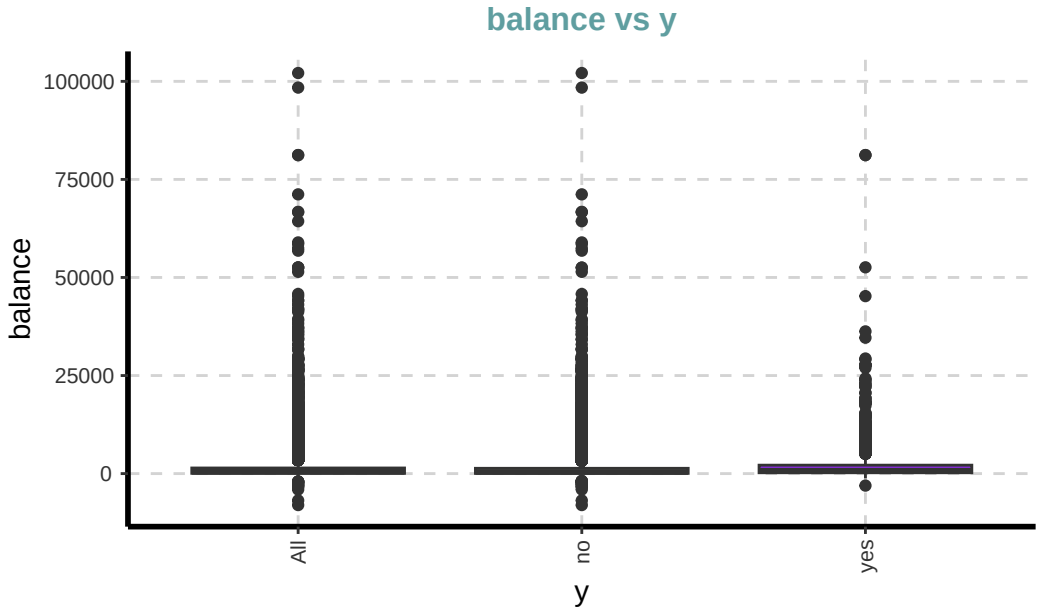
Finally, I analyze numerical features distribution

```
ExpNumViz(DataTrain,
          target="y",
          type=1,
          Page=c(1,1))
```

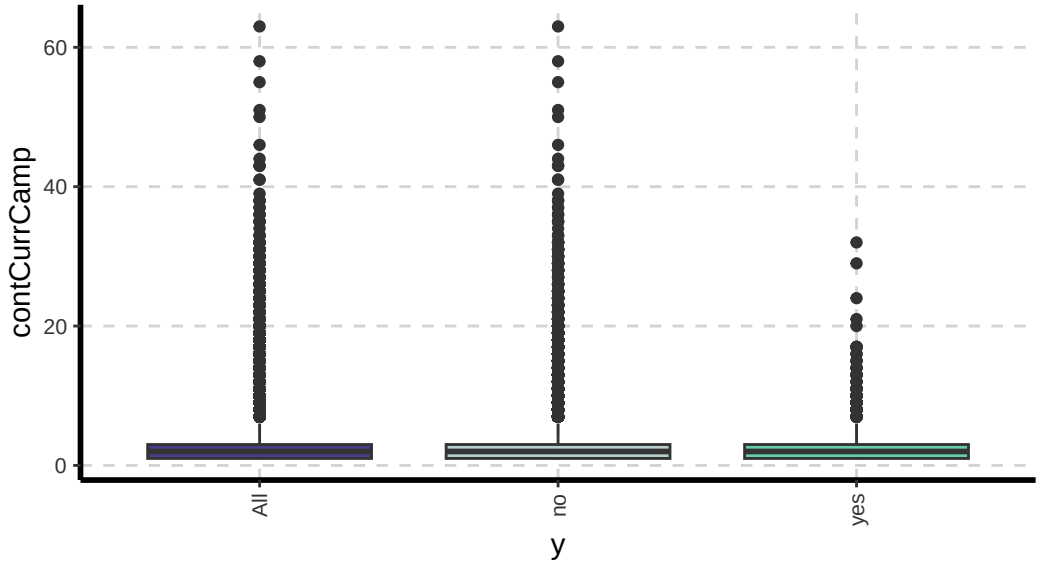
```
$`0`
```

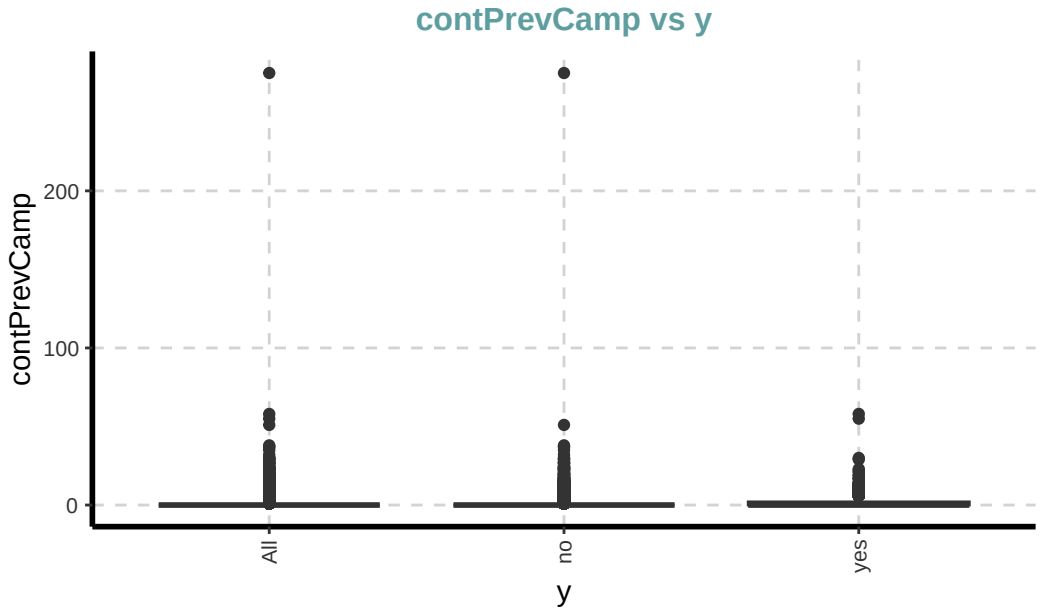


page 2 of 4



page 3 of 4





As far as numerical predictors are concerned, no specific issue seem to arise: outliers are definitely in there, but they appear to be considered acceptable and representative.

2.2.2. Categorical features

Again, I'll start with a summary of categorical features.

```
DataTrain|>
  ExpCatStat(Target = "y",
             result = "Stat",
             clim=15,
             nlim=5,
             bins=10,
             Pclass="yes",
             plot=FALSE,
             top=20,
             Round=2)
```

	Variable	Target	Unique	Chi-squared	p-value	df	IV	Value	Cramers V
1	job	y	12	637.624	0	NA	0.16		0.13
2	marital	y	3	155.188	0	NA	0.04		0.07
3	education	y	4	191.802	0	NA	0.06		0.07
4	default	y	2	21.182	0	NA	0.01		0.02

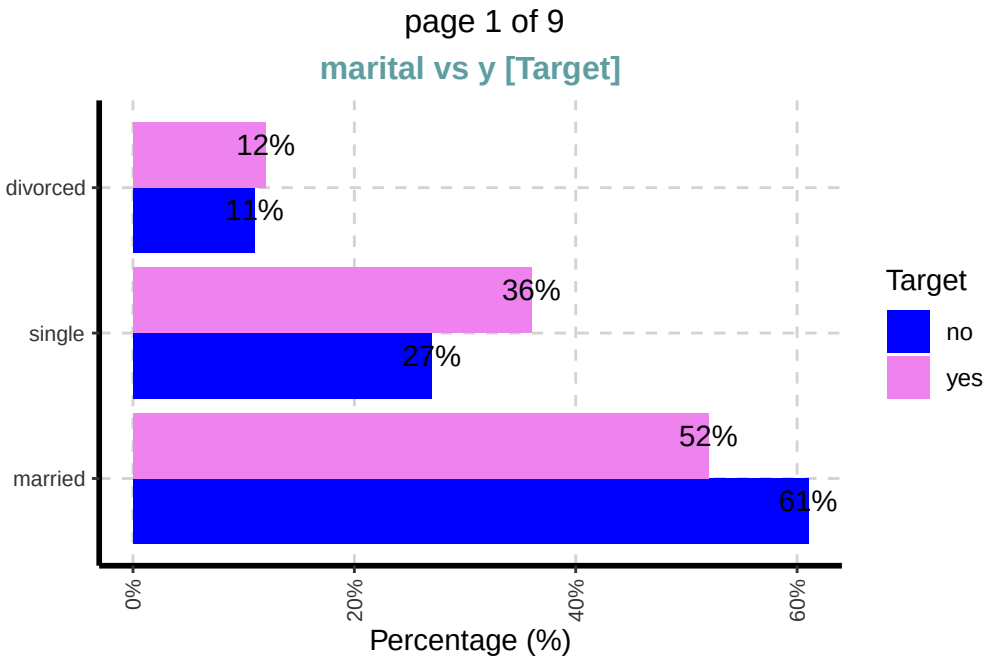
```
5           housing      y      2     716.087       0 NA     0.20      0.14
6              loan      y      2     163.462       0 NA     0.06      0.07
7           contact      y      3     854.425       0 NA     0.30      0.15
8             month      y     12    2516.955       0 NA     0.35      0.26
9         poutcome      y      4    3473.425       0 NA     0.56      0.31
10       firstCamp      y      2    1073.641       0 NA     0.24      0.17
11 distDaysPrevCamp      y      5    1249.406       0 NA     0.21      0.19
12              age      y     10     472.120       0 NA     0.11      0.11
13          balance      y     10     372.266       0 NA     0.09      0.10
14     contCurrCamp      y      5     214.015       0 NA     0.06      0.08
15     contPrevCamp      y      2     546.479       0 NA     0.11      0.12
   Degree of Association    Predictive Power
1                   Weak Somewhat Predictive
2              Very Weak      Not Predictive
3              Very Weak      Not Predictive
4              Very Weak      Not Predictive
5                   Weak   Medium Predictive
6              Very Weak      Not Predictive
7                   Weak   Highly Predictive
8               Moderate   Highly Predictive
9                 Strong   Highly Predictive
10                  Weak   Medium Predictive
11              Moderate   Medium Predictive
12                  Weak Somewhat Predictive
13                  Weak Somewhat Predictive
14             Very Weak      Not Predictive
15                  Weak Somewhat Predictive
```

Then, I move on to look at their distribution.

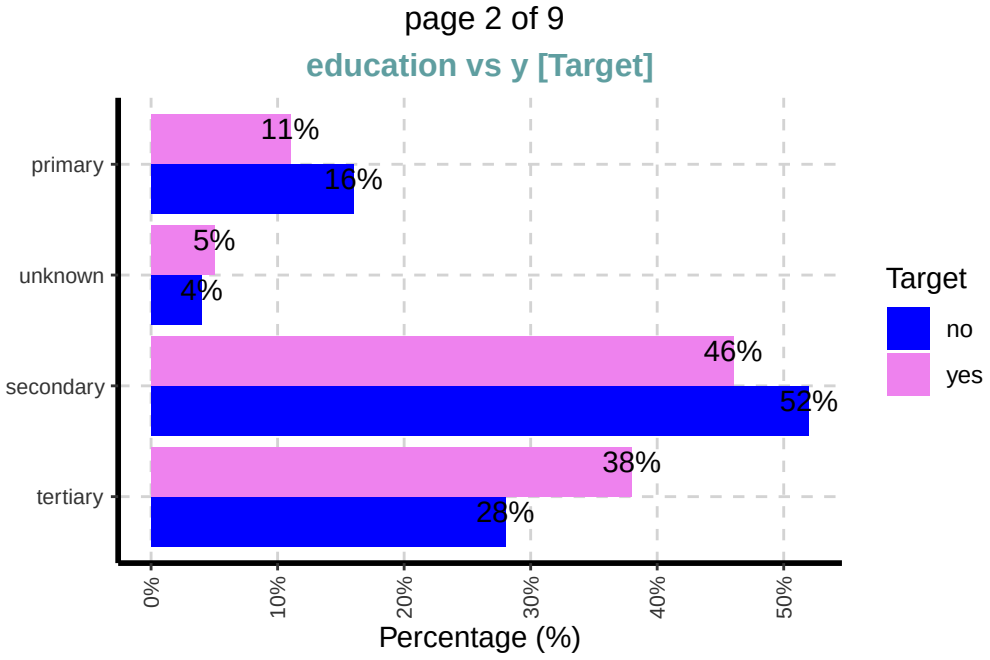
```
ExpCatViz(data=DataTrain,
          target="y",
          Page=c(1,1),
          Flip=TRUE,
          col=c("blue", "violet"))
```

```
$`0`
```

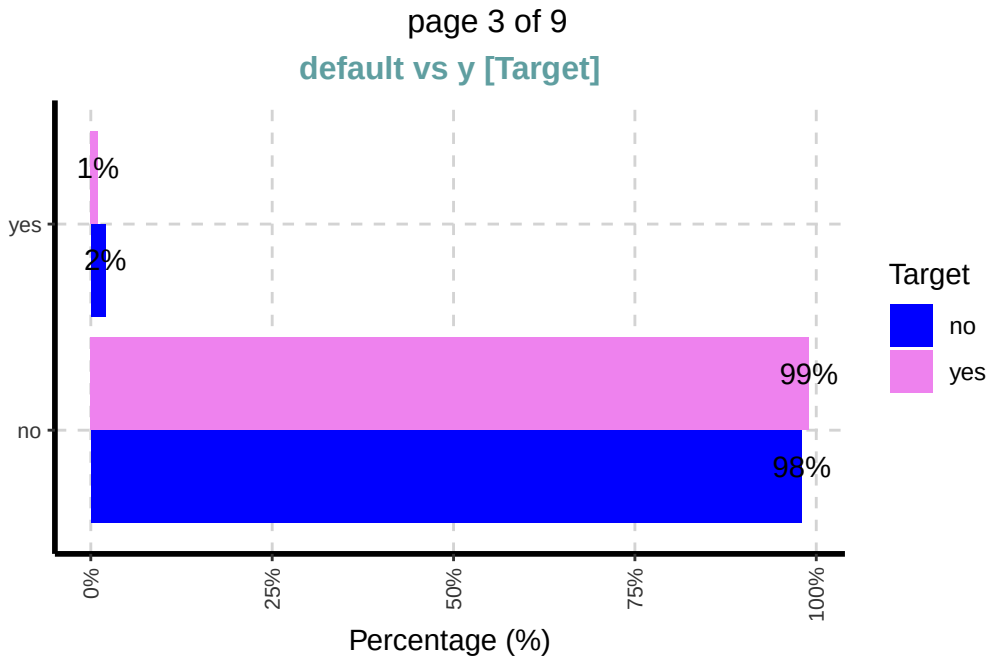
page 1 of 9



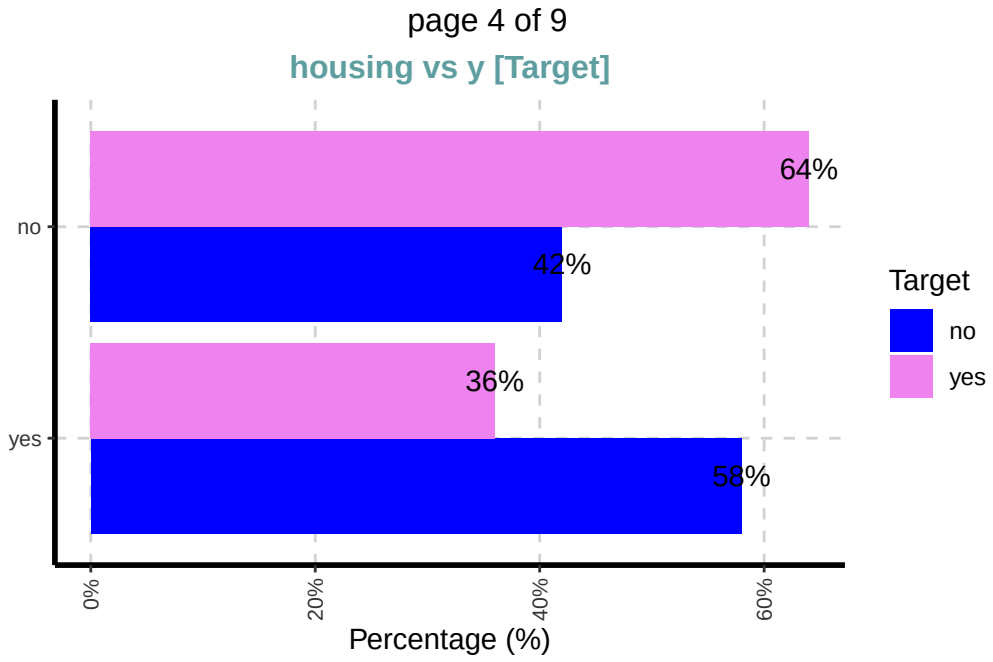
page 2 of 9



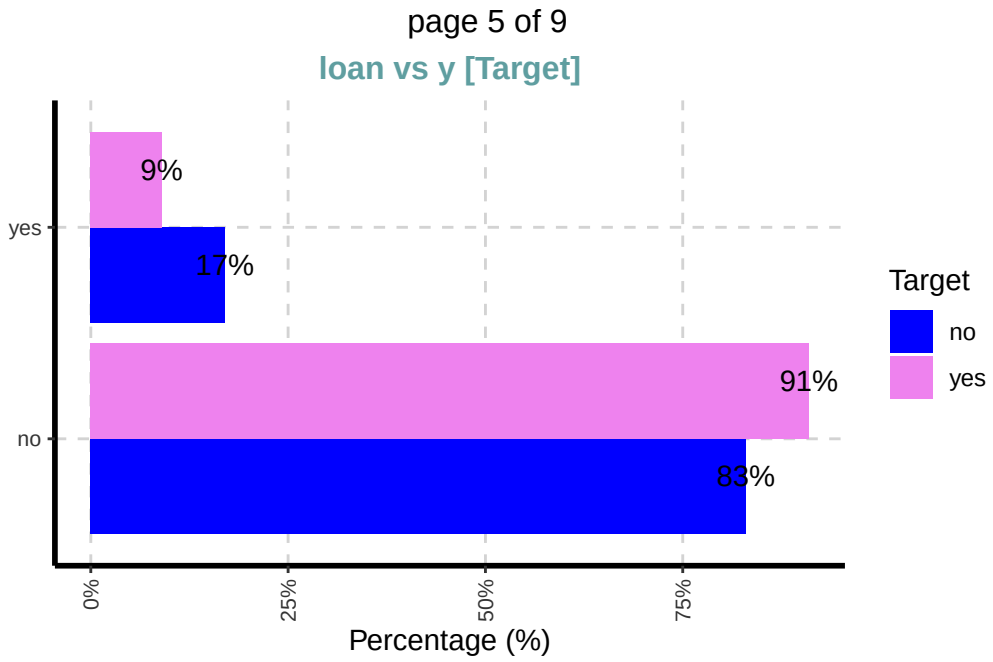
page 3 of 9



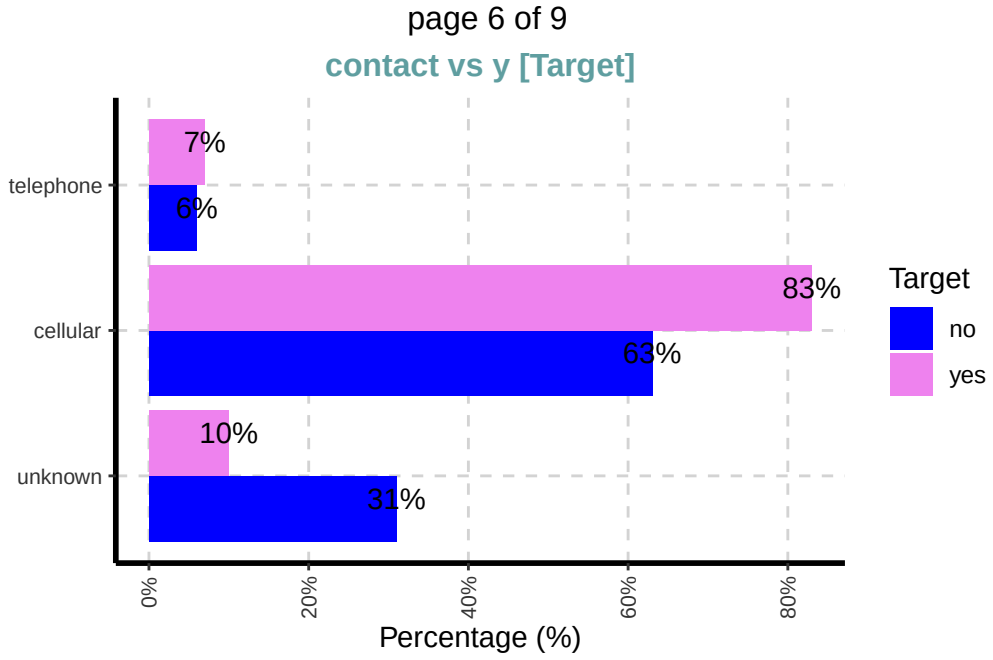
page 4 of 9



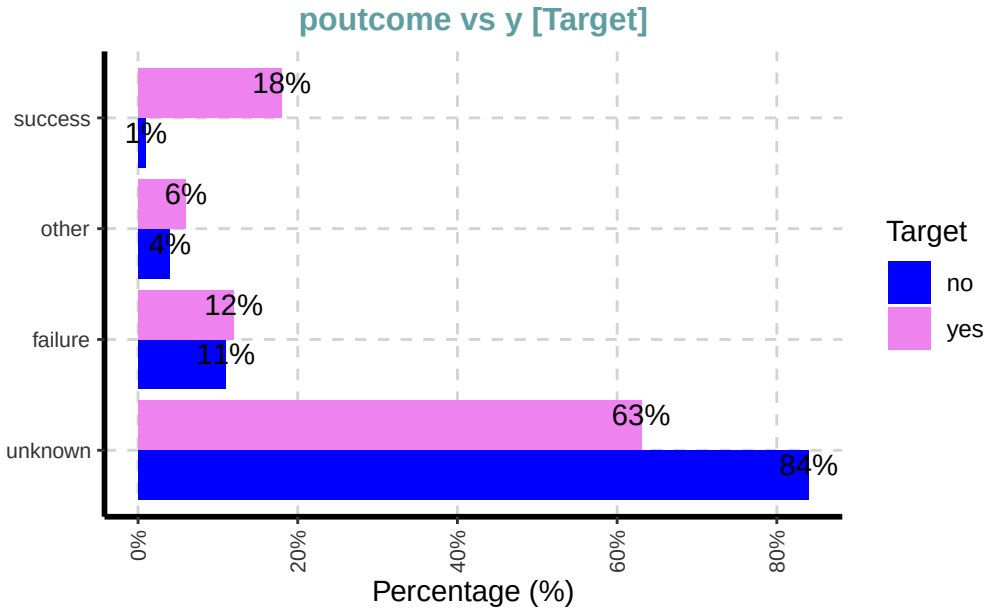
page 5 of 9



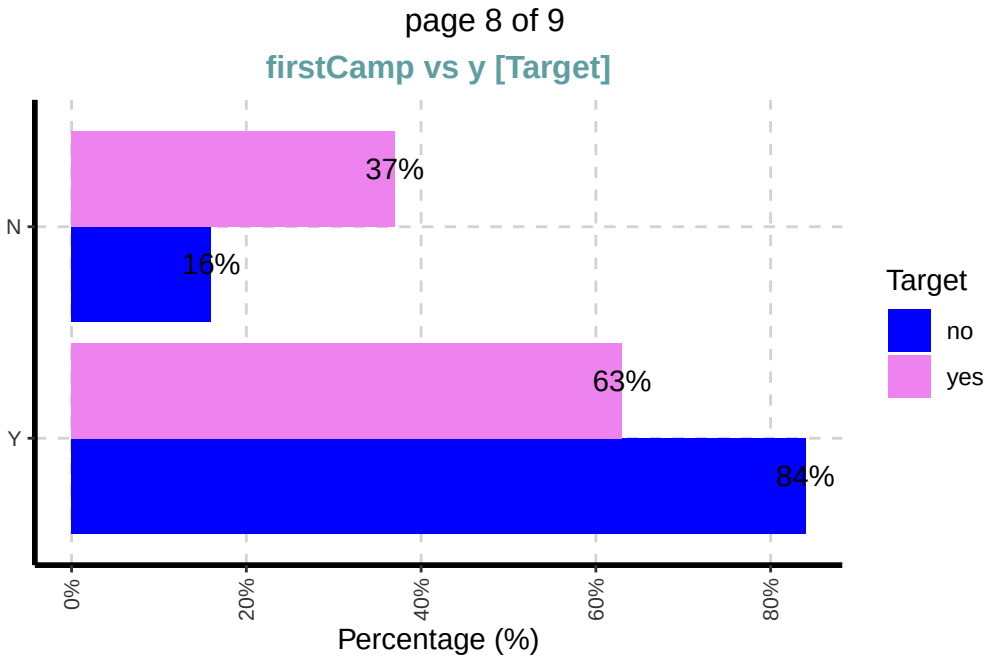
page 6 of 9

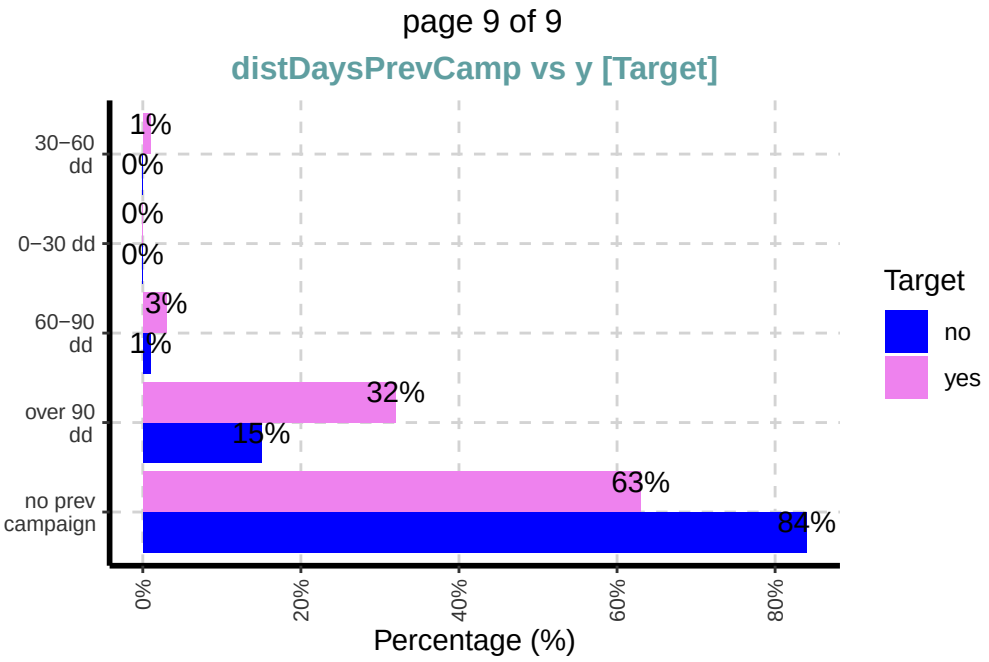


page 7 of 9



page 8 of 9





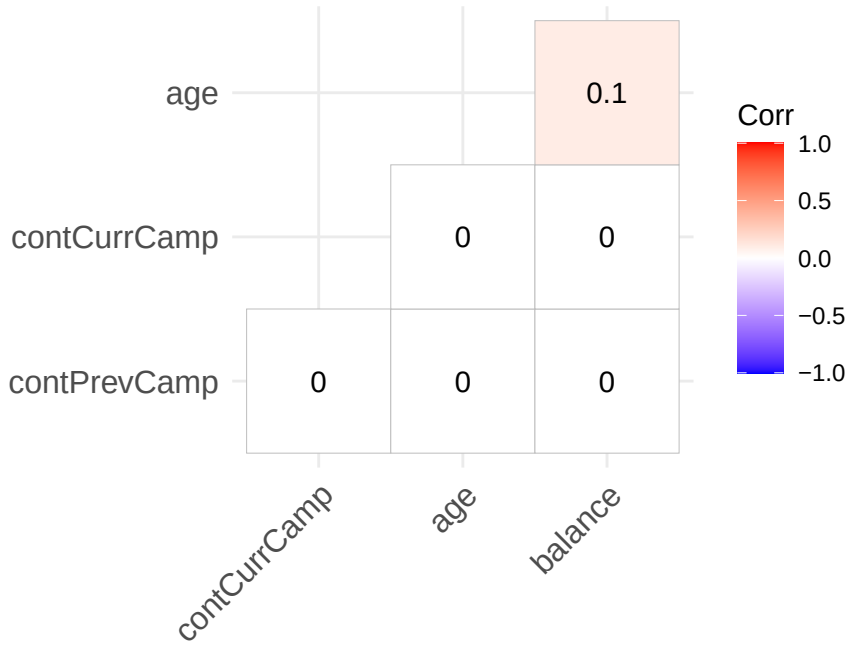
The relation between y (target variable) and other single categorical variables may be described as follows:

- generally, speaking there's an overall low-level of association.
- poutcome is the only predictor that show a somewhat relevant association with target variable (Cramer's V equal to 0.32).

2.3. Multivariate analysis

2.3.1 Correlation between numerical predictors

```
EdaCorMatr <- round(cor(DataTrain[,c(1,6,12,13)], use="complete.obs"), 1)

ggcorrplot(EdaCorMatr,
           hc.order = TRUE,
           type = "lower",
           lab = TRUE)
```



This analysis is probably quite poor with four numerical features. In any case, no correlation seems to exist between the variables taken into account.

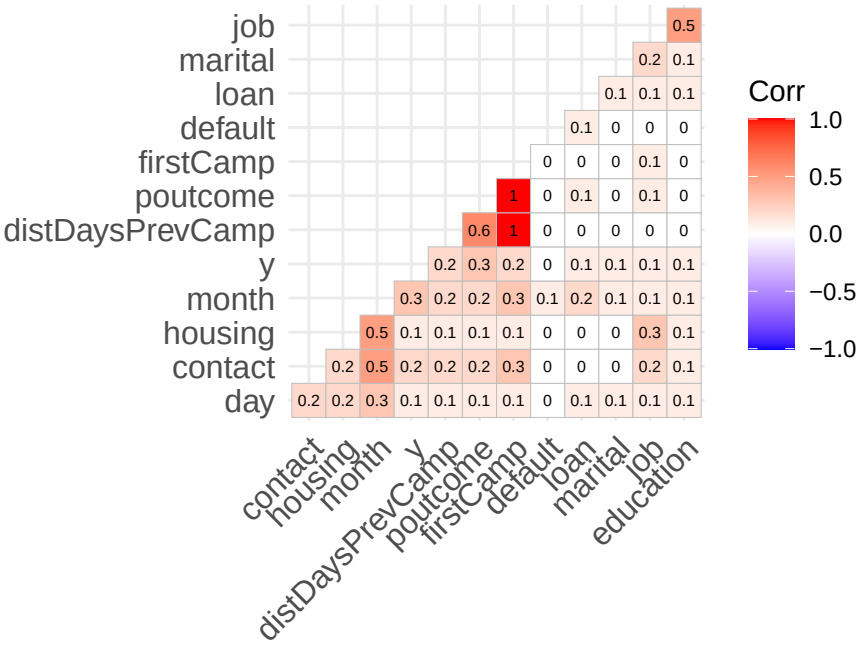
2.3.2. Association between categorical predictors

In order to analysize the association existing between every couple of features, I'm going to use Cramer's V and to apply this function to every pair of variables using the PairApply() command from the DescTools package.

```
EdaAssMatr <- DataTrain|>
  select(-age,
         -balance,
         -contCurrCamp,
         -contPrevCamp)|>
  PairApply(FUN=CramerV,
            symmetric=TRUE)|>
  round(digits = 1)
```

The result is a matrix that can be effectively exposed as correlation plot using the ggcorrplot() command.

```
ggcorrplot(EdaAssMatr,
           hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 2)
```



What comes from this plot?

1. "poutcome" and "firstCamp" are highly associated. That was quite easy to predict: the outcome of previous campaign is "unknown" as long as this is the first campaign. Since, according to Cramer's V, poutcome is more associated to target variable than firstCamp, I would pick the first.

2. Identically, the same thing happens between distDaysPrevCamp (distance in days from the previous campaign) and first campaign. Again: when this is the first campaign the distance in days from the previous campaign gets the categorical value "no previous campaign". Since, according to Cramer's V, distDaysPrevCamp is slightly more associated to y (the target variable) than firstCamp, I would pick the first.

2.4. Notes for further steps

1. The target variable is pretty unbalanced. This fact suggests an under/oversampling in the preprocessing step of the machine learning project phase.

2. The "firstCamp" feature that was not included in the original dataset and was created later, is probably useless because too correlated with other existing predictors.
3. Quite a lot of features seem to be slightly correlated with target variable. It could be wise to experiment a model (like decision tree) able to feed back a feature importance ranking.

3. ML model: decision tree

I have imported the dataset, cleaned and manipulated it; I have analysed numerical and categorical predictors in order to have a better knowledge of the "raw material" I'm about to use to train a ML model.

Due to the weak to medium correlation between every predictor and outcome, I will pick decision tree model: a simple option that will allow me to get feature importance which I am particularly interested in, especially after EDA's feedback.

The first thing is to create the baseline for any comparison, which is a decision tree whose hyperparameters will be tuned and that does not take any care of unbalanced data.

Next, I want to see if and how the choice of rebalancing target values proportion will bring some benefits.

In order to make a fair comparison, hyperparameter tuning will be carried on in the same way also for the training/testing phases on rebalanced dataset (both via undersampling and via oversampling). To keep things simple, optimize laptop resources and grant some readability of the model, I have defined a range of hyperparameters to choose that will be used for every following training session:

- cost_complexity from (-5) to (-3),
- tree_depth from 5 to 7.

Thus, I will tune three decision tree:

1. the first, without under/oversampling, with hyperparameters tuning,
2. the second which includes rebalancing via undersampling
3. the third which uses oversampling to solve unbalanced target values.

As stated before, my very first goal is not picking the most performing model at all, but understanding if and how both undersampling and oversampling affect the final result of a specific model, taking into account the ratio between the two possible classes of the target variable (in this project: "yes" or "no").

Thus, during under/oversampling, I will use a reiterate training and testing with different "yes/no" ratio in order to find out which ratio will produce more benefit. In details, I will use themis package which is specifically aimed to deal with unbalanced data.

3.1. Tuned decision tree without under/oversampling

3.1.1 Training and testing

I start by setting the preprocessing recipe.

```
dtRecipe <- DataTrain |>
  recipe(y ~.)|>
  step_rm(c("firstCamp"))
```

I select the model I'm interested to use (as said, decision tree) with hyperparameters tuning options.

```
dtTuneModel <- decision_tree(tree_depth = tune(),
                             cost_complexity = tune())|>
  set_engine("rpart")|>
  set_mode("classification")
```

I define:

1. tuning grid. I need to keep it simple to optimize my laptop computational resources. I will thus set 3 levels only (which means nine combination of the two hyperparameters).
2. cross validation folds. Again, I'm going to use 5 folds which is probably the lowest useful number of folds to make cross validation useful.
3. metric set. In order to evaluate performance, I will create an object (called "dtMultiMetric") composed of different index (accuracy, kappa, balanced accuracy, specificity, sentitivity and f-measure) for an overall evaluation; specifically I will pay particular attention to balanced accuracy which, in my opinion, should give fair relevance to both the need for catching true positive and of true negative predictions.

```
dtTuneGrid <- grid_regular(cost_complexity(c(-5, -3)),
                           tree_depth(c(5,7)),
                           levels = 3)
```

```r
set.seed(123, sample.kind = "Rounding")
dtTuneFold <- vfold_cv(DataTrain, v=5)

# set multi metrics
multi_metric <- metric_set(accuracy,
                           kap,
                           bal_accuracy,
                           spec,
                           sens,
                           f_meas)
```

I create the workflow creation, choose the best hyperparameters (here based on specificity metric), fit the model and plot it.

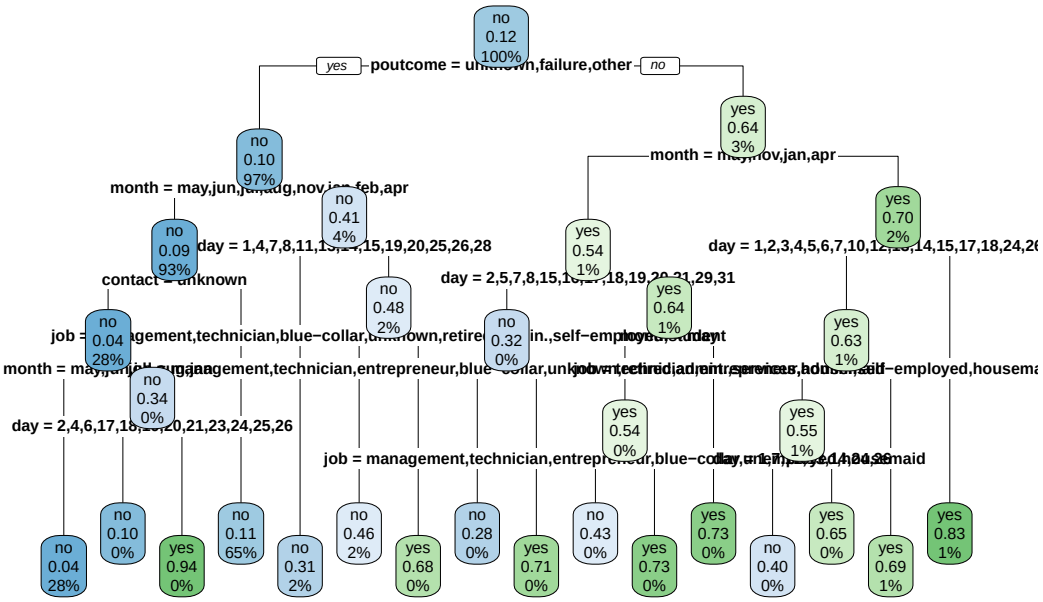
```r
dtTuneWorkflow <- workflow()|>
  add_recipe(dtRecipe)|>
  add_model(dtTuneModel)

dtTuneFit <- dtTuneWorkflow|>
  tune_grid(resamples = dtTuneFold,
            grid = dtTuneGrid,
            metrics = multi_metric)

# best hyperparameters selection
dtTunebest <- dtTuneFit|>
  select_best(metric="bal_accuracy")

# finalize
dtTuneWorkflowFinal <- dtTuneWorkflow|>
  finalize_workflow(dtTunebest)

dtTuneFinal <- dtTuneWorkflowFinal|>
  fit(DataTrain)
```

3.1.2. Main results

The tuned tree takes the following shape

```r
# tree plot
dtTuneFinal|>
```

```
  extract_fit_engine()|>
  rpart.plot(roundint = FALSE, cex=0.5)
```



shows these tuned hyperparameters

```
# tuning hyperparameters
kable(dtTunebest[1,1:2])
```

cost_complexity	tree_depth
0.001	5

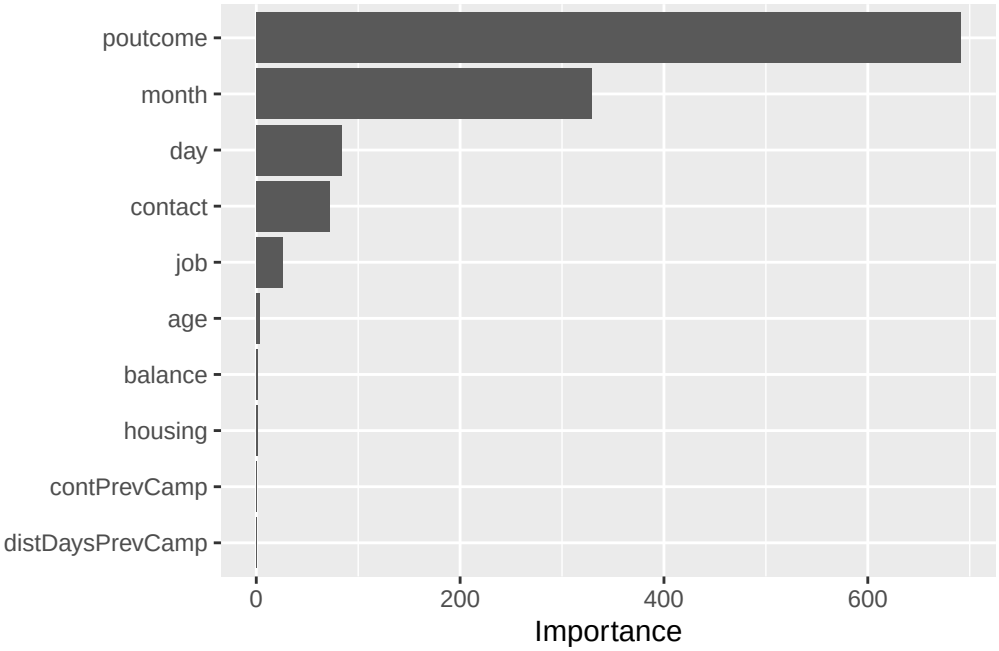
performs according to these metrics

```
# metric
dtMultiMetrics <- dtTuneFinal|>
  predict(DataTest)|>
  bind_cols(DataTest) |>
  multi_metric(truth = y, estimate = .pred_class)

dtMultiMetrics|>
  select(-.estimator)|>
  kable()
```

.metric	.estimate
accuracy	0.8921818
kap	0.2143912
bal_accuracy	0.5712028
spec	0.1521739
sens	0.9902317
f_meas	0.9419263

and reports this feature importance.

```
# feature importance plot
dtTuneFinal %>%
  extract_fit_parsnip()|>
  vip()
```



3.2. Tuned decision tree with undersampling

3.2.1. Training and testing

There is no need to create a model. I want to use the same model used previously (dtTuneModel).

The same goes for the tuning grid, the cross validation folds and the metric set: I've previously created dtTuneGrid, dtTuneFold and multi_metrics. They'll fit the bill.

Here it comes the crucial point.

I am interested in how the undersampling (and in the following step the oversampling) techique performs with different value of the argument called "under ratio" which is the ratio of the minority-to-majority frequencies (for example, a value of 2 would mean that the majority levels will have approximately twice as many rows than the minority level).

I want to reiterate the training cycle (preprocessing, hyperparameters tuning, and testing) with five different "under ratio" values to see which one returns the best performance of the tuned decision tree model.

```
under_ratios <- c(0.5, 0.75, 1, 1.25, 1.5)
```

```
dtUnderModFun <- function(under_ratios){

 dtRecipe <- DataTrain|>
    recipe(y ~ .)|>
    step_rm(c("firstCamp"))|>
    step_downsample(y, under_ratio = under_ratios, seed=523)

  dtWorkflow <- workflow()|>
    add_recipe(dtRecipe)|>
    add_model(dtTuneModel)

  dtFit <- dtWorkflow|>
    tune_grid(resamples = dtTuneFold,
              grid = dtTuneGrid,
              metrics = multi_metric)

  dtUnderTunebest <- dtFit|>
    select_best(metric="bal_accuracy")

  assign("dtUnderTunebestFun", dtUnderTunebest, envir = .GlobalEnv)

  dtTuneWorkflowFinal <-
    dtWorkflow|>
    finalize_workflow(dtUnderTunebest)

  dtTuneFinal <- dtTuneWorkflowFinal|>
    fit(DataTrain)
```

```
  dtMultiMetrics <- dtTuneFinal|>
    predict(DataTest)|>
    bind_cols(DataTest) |>
    multi_metric(truth = y, estimate = .pred_class)

  dtMultiMetrics[[3,3]]
}
```

After setting the seed, I reiterate the function I've created to keep all the necessary steps in a whole with map function from purr package.

```
set.seed(523, sample.kind = "Rounding")

dtUnderPar<- map_dbl(.x = under_ratios,
                     .f = dtUnderModFun)

dtUnderSampling <- bind_cols(ratio=under_ratios, bal_acc_under=dtUnderPar)

dtUnderSampling|>
  arrange(desc(bal_acc_under))|>
  kable()
```

ratio	bal_acc_under
1.00	0.7140660
1.50	0.7052510
1.25	0.7009457
0.50	0.6866842
0.75	0.6801416

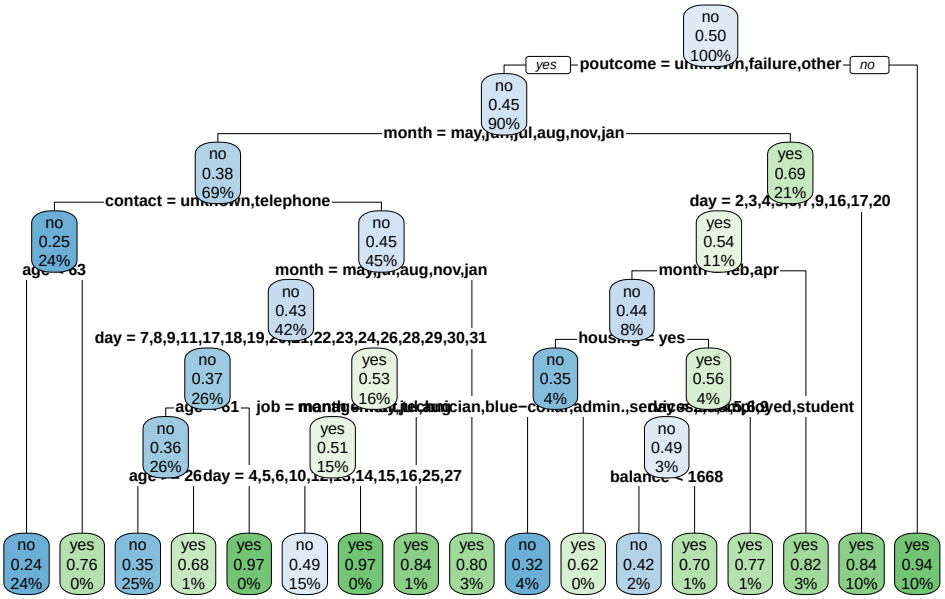
The optimal ratio for undersampling is 1.00. Let's use it to extract the final tree with undersampled data base.

3.2.2. Main results

The tuned tree takes the following shape

```
# tree plot

dtUnderTuneFinal|>
```

```
  extract_fit_engine()|>
  rpart.plot(roundint = FALSE, cex=0.5)
```



shows these tuned hyperparameters

```
kable(dtUnderTunebest[1,1:2])
```

cost_complexity	tree_depth
0.001	7

performs according to these metrics. Note that I don't want to apply preprocessing recipe on test dataset, thus I will use extract_fit_parsnip() command to obtain the parsnip model specification in order to use it without any preprocessing step.

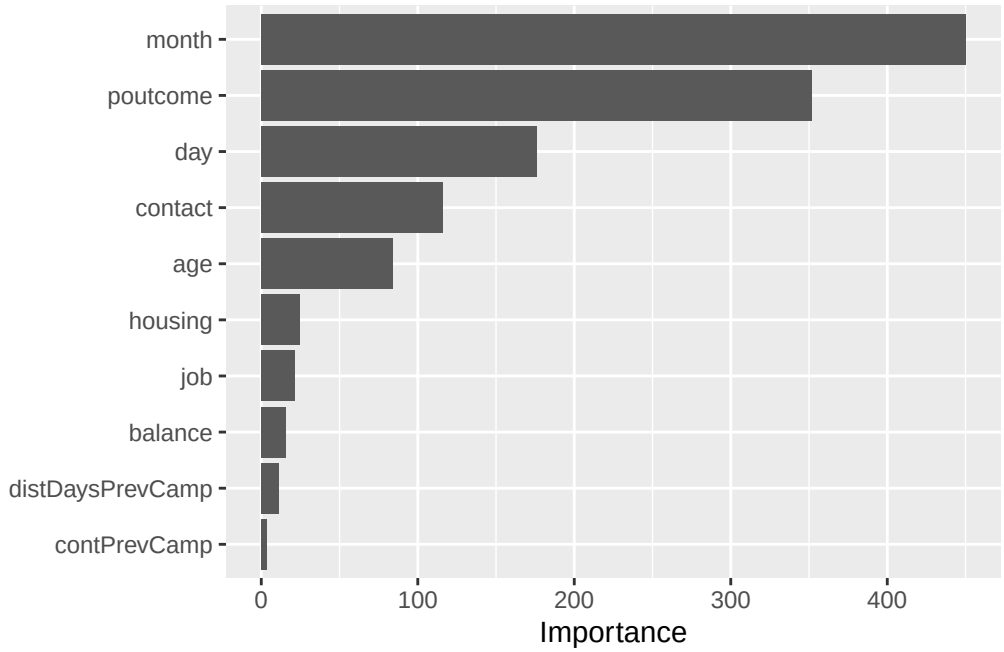
```
# prediction and evaluation

# Extract the fitted model
dtUnderFittedModel <- dtUnderTuneFinal|>
  extract_fit_parsnip()

# Make predictions without preprocessing
dtUnderMultiMetricsNoPrep <- dtUnderFittedModel|>
  predict(DataTest)|>
```

```
  bind_cols(DataTest) |>
  multi_metric(truth = y, estimate = .pred_class)|>
  select(-.estimator)|>
  kable()

dtUnderMultiMetricsNoPrep
```

.metric	.estimate
accuracy	0.8606657
kap	0.3883308
bal_accuracy	0.7140660
spec	0.5226843
sens	0.9054477
f_meas	0.9198473

and reports this feature importance.

```
# variable importance

dtUnderTuneFinal %>%
  extract_fit_parsnip()|>
  vip()
```



3.3. Tuned decision tree with oversampling

3.3.1. Training and testing

As for undersampling, I'll keep on using what has been previously created: the model "dtTuneModel", the tuning grid "dtTuneFold", the cross validation folds "dtTuneFold" and the metric set "multi_metrics".

What I need to do is to modify is the function that holds inside the oversampling preprocessing step and which is feed with "over ratio" arguments.

```
over_ratios <- c(0.5, 0.75, 1, 1.25, 1.5)


dtOverModFun <- function(over_ratios){

  dtRecipe <- DataTrain|>
    recipe(y ~ .)|>
    step_rm(c("firstCamp"))|>
    step_upsample(y, over_ratio = over_ratios, seed=523)

  dtWorkflow <- workflow()|>
    add_recipe(dtRecipe)|>
    add_model(dtTuneModel)

  dtFit <- dtWorkflow|>
    tune_grid(resamples = dtTuneFold,
              grid = dtTuneGrid,
              metrics = multi_metric)

  dtOverTunebest <- dtFit|>
    select_best(metric="bal_accuracy")

  assign("dtOverTunebestFun", dtUnderTunebest, envir = .GlobalEnv)

  dtTuneWorkflowFinal <-
    dtWorkflow|>
    finalize_workflow(dtOverTunebest)

  dtTuneFinal <- dtTuneWorkflowFinal|>
    fit(DataTrain)

  dtMultiMetrics <- dtTuneFinal|>
```

```
    predict(DataTest)|>
    bind_cols(DataTest) |>
    multi_metric(truth = y, estimate = .pred_class)

  dtMultiMetrics[[3,3]]
}
```

Time for applying the function and check results.

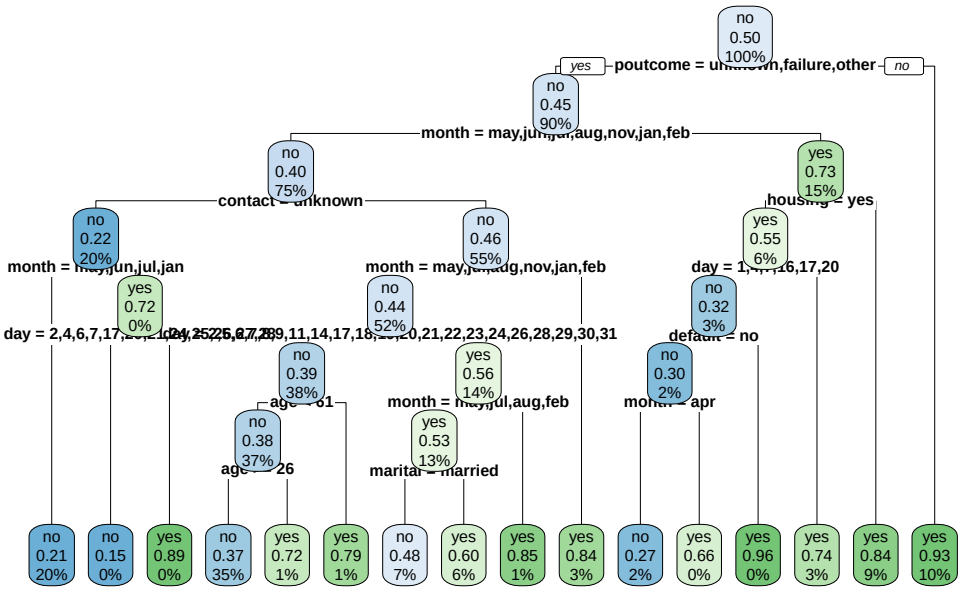
```
set.seed(523, sample.kind = "Rounding")

dtOverPar<- map_dbl(.x = over_ratios,
                    .f = dtOverModFun)

dtOverSampling <- bind_cols(ratio=over_ratios, bal_acc_over=dtOverPar)

dtOverSampling|>
  arrange(desc(bal_acc_over))|>
  kable()
```

ratio	bal_acc_over
1.50	0.7176579
0.75	0.7135286
1.00	0.7109728
0.50	0.7004033
1.25	0.6982459

3.3.2. Main results

The tuned tree takes the following shape

```
# tree plot

dtOverTuneFinal|>
  extract_fit_engine()|>
  rpart.plot(roundint = FALSE, cex=0.5)
```



shows these tuned hyperparameters

```
kable(dtOverTunebest[1,1:2])
```

cost_complexity	tree_depth
0.001	7

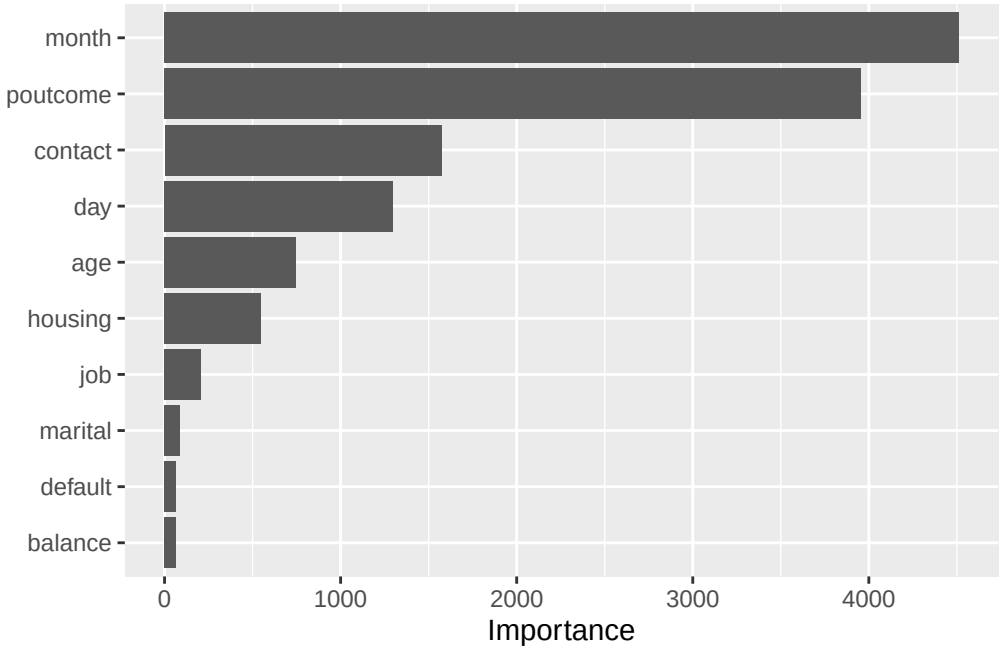
performs according to these metrics. Again, note that I'll just use the model wihout any preprocessing recipe.

```
# prediction and evaluation

# Extract the fitted model
dtOverFittedModel <- dtOverTuneFinal|>
  extract_fit_parsnip()

# Make predictions without preprocessing
dtOverMultiMetricsNoPrep <- dtOverFittedModel|>
  predict(DataTest)|>
  bind_cols(DataTest) |>
  multi_metric(truth = y, estimate = .pred_class)|>
  select(-.estimator)|>
  kable()

dtOverMultiMetricsNoPrep
```

.metric	.estimate
accuracy	0.8308084
kap	0.3470912
bal_accuracy	0.7176579
spec	0.5699433
sens	0.8653726
f_meas	0.9003257

and reports this feature importance.

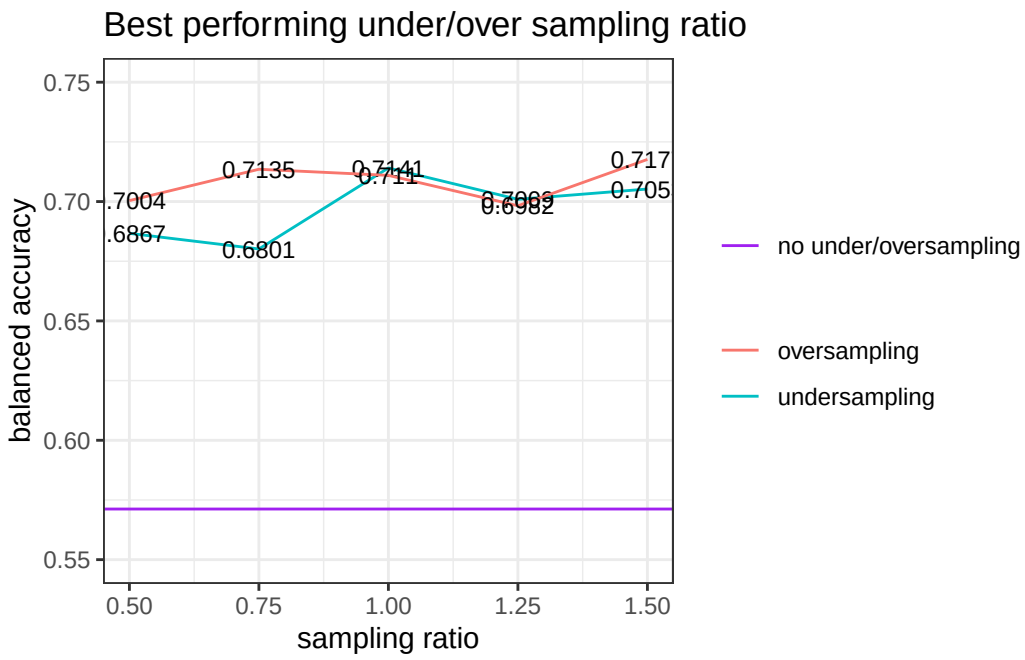
```
# variable importance

dtOverTuneFinal %>%
  extract_fit_parsnip()|>
  vip()
```



3.4. Which option performed better?

Let's try to put everything together and understand what is the best option in order to get the best model performance (in terms of balanced accuracy).

```
ggplot()+
  geom_line(data = dtUnderSampling,
            aes(x = ratio, y = bal_acc_under, colour = 'undersampling')) +
  geom_text(data = dtUnderSampling,
            aes(x = ratio, y = bal_acc_under, label = round(bal_acc_under, digits=4)),
            size = 3)+
  geom_line(data = dtOverSampling,
            aes(x = ratio, y = bal_acc_over, colour = 'oversampling')) +
  geom_text(data = dtOverSampling,
            aes(x = ratio, y = bal_acc_over, label = round(bal_acc_over, digits=4)),
            size = 3)+
  geom_hline(aes(yintercept = 0.5712028, linetype = "no under/oversampling"), colour="purple
  scale_y_continuous(limits=c(0.55, 0.75))+
  labs(x = "sampling ratio",
       y = "balanced accuracy",
       title = "Best performing under/over sampling ratio") +
  theme_bw()+
  theme(legend.title = element_blank())
```



The plot clearly show the significant effect of both under and over sampling on the model's performance.

4. Conclusions

About the model

- Rebalancing the training set pays off: without under/over sampling I have reached an about 0.57 balanced accuracy which has rocketed to about 0.70 after rebalancing.
- Other differences between training the model either on the original (unbalanced) or on a rebalanced dataset are:
- *feature importance changes. Specifically the first four variable are always the same (poutcome, month, day and contact), but their relavance (ranking) changes. Month is the most important variable to explain the model trained on an under/oversampled dataset, while poutcome is by far the most important when using the original dataset.*
- *tree depth changes: when working on the original dataset, the tree has 5 levels. After either under or over sampling, levels increase to 7. Please note that in the tune grid I set a range from 5 to 7, otherwise, the tree would have turned to be very deep, or long, and thus unreadable.*
- *cost complexity hasn't changed at all. In the range I've set in the tune grid (0.001 - 0.00001), 0.001 has always been returned.*
- Generally speaking, even the best performance does not seem very satisfying. A balanced accuracy of around 70% that may be acceptable (it's far better than tossing!), but mantains some room for uncertainty.
- Specific ranges have been set in the tuning grid in order to speed up the whole process and ensure model's readability.

About under vs over sampling

- Undersampling based decision tree has got its best result with a 1 minority-to majority ratio: 0.7141.
- Oversampling based decision tree "scored" a 0.717 balanced accuracy with, again, a 1.50 minority-to majority ratio.
- As said before, it is clearly important to fix the unbalanced target value issue in order to improve performance dramatically; it's definitely less important whether to follow undersampling or oversampling approach. The latter has led to a slightly better result which, in turn, has asked for more computation effort due to the increased dimension of the rebalanced dataset.

Lessons learned

1. It is useful to define *ex ante* tuning grid parameters to keep the process fast and "under control".
2. Chose a metric and stick to it throughout the whole project in order to ensure results comparability.
3. Set specific value ranges in the tuning grid (cp and tree depth) in order to speed up the whole process and ensure model's readability.
4. Test undersampling and oversampling with a pretty wide range of ratios in order to see which one brings to the best result: map function from purrr package has been very helpful in this stage.
5. Pay attention to set the seeds every time a sampling is made in order to ensure reproducibility.
6. When testing the model, after training, I have used the fitted model without preprocessing, in order to simulate a "real world" data to treat. But what if I had used the same whole workflow used for the training phase? I've separately (and not reported here) tried this solution and got the same exact metrics; I've done some other quick trial with both other dataset and other models and I've come to conclusion that while decision tree is totally insensitive to test preprocessing, other models (i.e. knn) bring to different performance if the test set is preprocessed or not (as expeted, in case of preprocessing metrics are higher, thus probably deceptive).

References

Moro,S., Rita,P., and Cortez,P.. (2012). Bank Marketing. UCI Machine Learning Repository. https://doi.org/10.24432/C5K306.