Caravan Insurance Challenge

Dataset source: (https://www.kaggle.com/datasets/uciml/caravan-insurance-challenge).

Goal: Identify potential purchasers of caravan insurance policies. The target variable is CARAVAN, which is a binary variable.

Each observation corresponds to data derived from a zip area code. Variables beginning with M refer to demographic statistics of the postal code, while variables beginning with P and A (as well as CARAVAN, the target variable) refer to product ownership and insurance statistics in the postal code. Most of the variables have values described by the tables L0, L1, L2, L3 and L4 which can are described in the dataset source linked above.

```
data <- read.csv('data/caravan-insurance-challenge.csv')  # kaggle dataset
# head(data, 3)
```

Exploratory Data Analysis

In the following chunks, after checking for missing values and splitting the dataset into training set and test set (using the "ORIGIN" variable), we will perform EDA on the portion of the dataset designed for training, thus isolating the test set (which will be only used for testing the models). In particular we will:

- analyze the target variable distribution (checking the balance of the dataset);
- analyze and understand the variable types according to the dataset description;
- check for multicollinearity (excluding the target variable).

Dataset shape

```
dim( data )
```

```
## [1] 9822   87
```

The dataset has 9822 rows and 86 columns, i.e. 9822 observations and 86 variables.

Checking for missing values

```
sum( is.na(data) ) # count missing values
```

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

No missing values are present in the data.

Train-Test Split using the 'ORIGIN' variable

The original dataset has a column called 'ORIGIN' which indicates whether the observation belongs to the train or test set; this column is used to split the data into two sets. The 'ORIGIN' column is then removed from the data after the split, since it is not necessary anymore.

```
# splitting the data into train and test sets (using 'ORIGIN' variable):
train <- data[data$ORIGIN == "train", ]
test <- data[data$ORIGIN == "test", ]

# remove 'ORIGIN' variable (not necessary anymore):
train <- train[, -which(names(train) == "ORIGIN")]
test <- test[, -which(names(test) == "ORIGIN")]

# dump the train and test sets to the disk:
```

```r
write.csv(train, "data/train_raw.csv", row.names = FALSE)
write.csv(test, "data/test_raw.csv", row.names = FALSE)

# check the shape of the train and test sets:
print( dim(train) )
```

```
## [1] 5822   86
```

```r
print( dim(test) )
```

```
## [1] 4000   86
```

```r
library(ggplot2)

train_test <- data.frame(
  set = c("Train", "Test"),
  count = c(nrow(train), nrow(test))
)

ggplot(train_test, aes(x = set, y = count, fill = set)) +
  geom_bar(stat = "identity", width = 0.7, show.legend = FALSE) +
  labs(title = "Train-test sets proportions", x = "Set", y = "Count") +
  theme_minimal() +
  geom_text(aes(label = count), vjust = -0.3)
```

The test set includes approximately the 40% of the original data. These datasets will be denoted as raw because they have not been preprocessed yet.

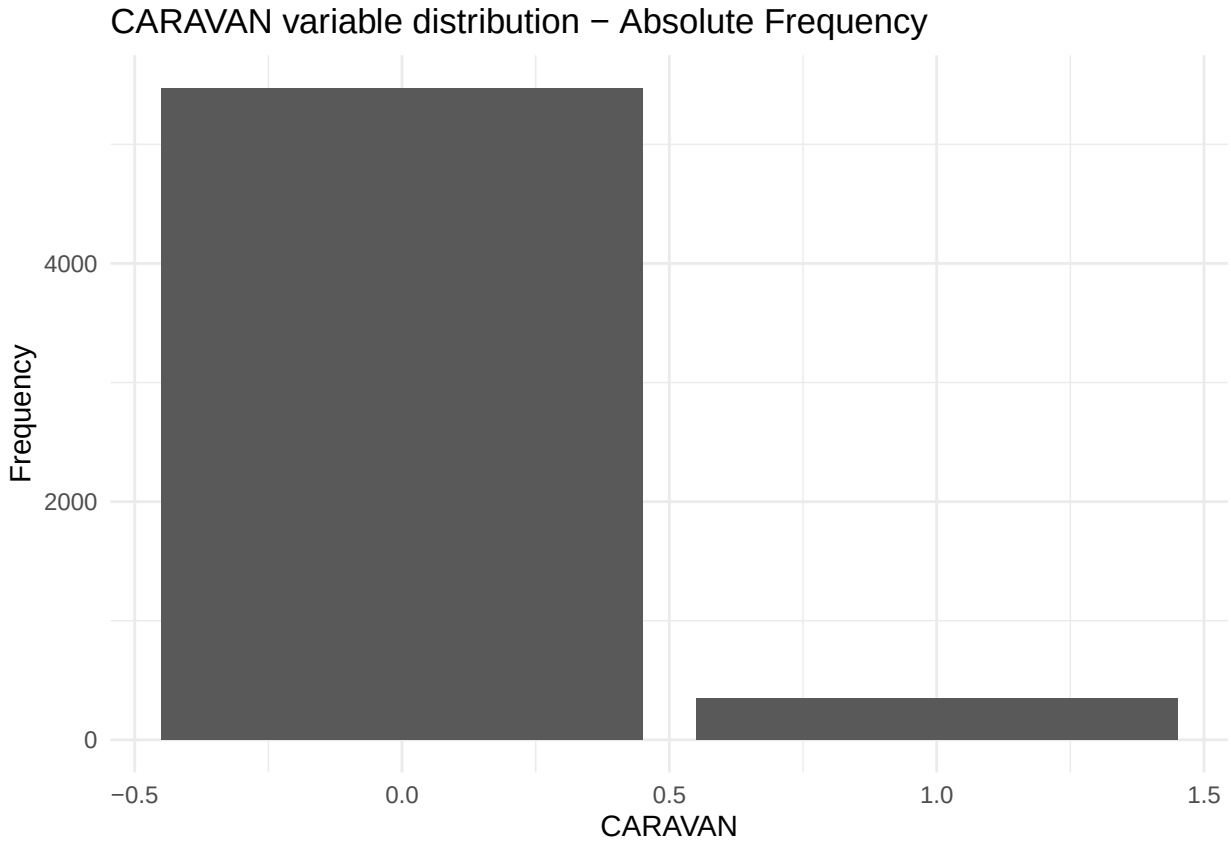
In the following chunks **we will perform exploratory data analysis only on the portion of the data**

that belongs to the training set, this because the test set is not supposed to be used for any kind of analysis, only for testing the model.

Analyzing target variable distribution

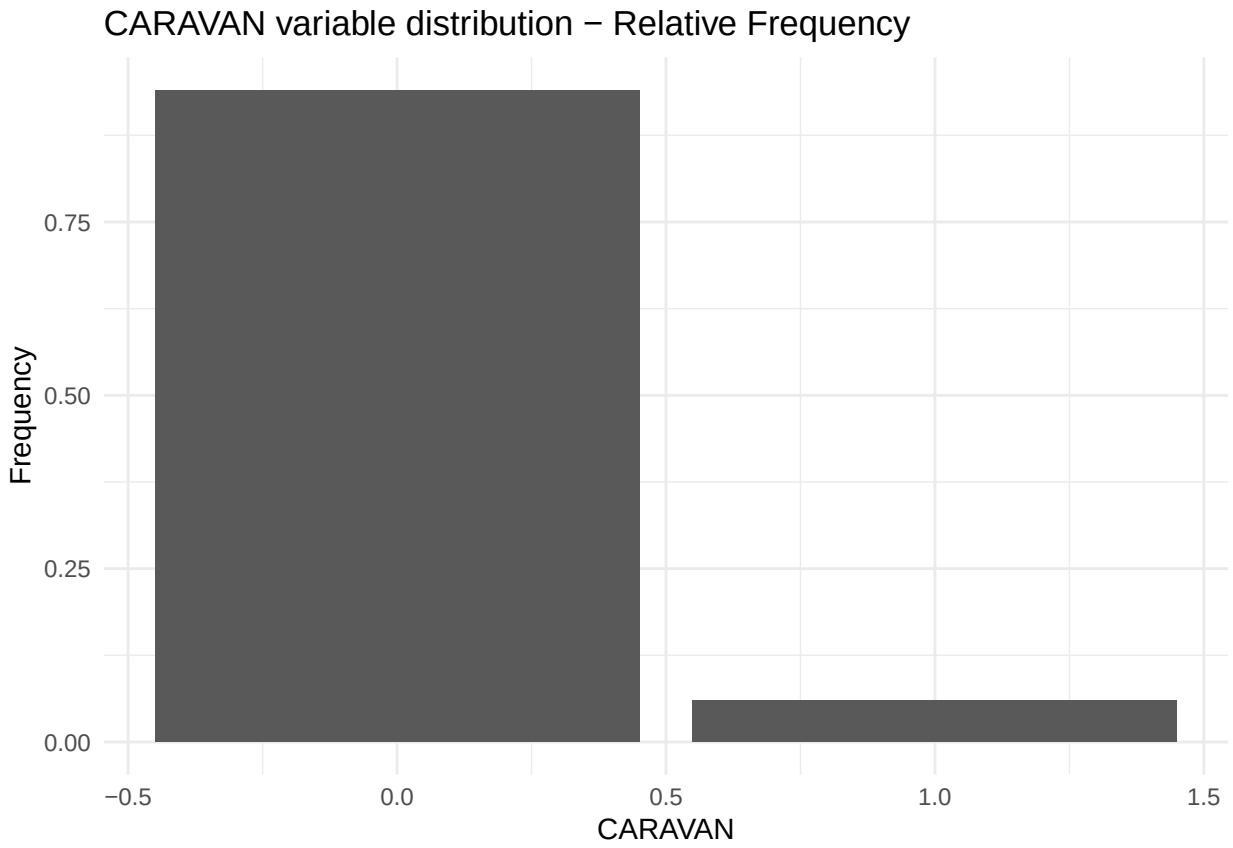
```r
library(ggplot2)

# absolute frequency plot:
ggplot(train, aes(x = CARAVAN)) +
  geom_bar() +
  labs(
    title = "CARAVAN variable distribution - Absolute Frequency",
    x = "CARAVAN", y = "Frequency"
  ) + theme_minimal()
```




```r
ggsave("graphs/abs_freq_CARAVAN.png", width = 10, height = 10)

# relative frequency plot:
ggplot(data, aes(x = CARAVAN)) +
  geom_bar(aes(y = (after_stat(count))/sum(after_stat(count)))) +
  labs(
    title = "CARAVAN variable distribution - Relative Frequency",
    x = "CARAVAN", y = "Frequency"
  ) + theme_minimal()
```




```
ggsave("graphs/rel_freq_CARAVAN.png", width = 10, height = 10)

# table with relative frequencies:
table(data$CARAVAN) / nrow(data)
```

```
##
##          0          1
## 0.94033802 0.05966198
```

The dataset is highly unbalanced: only 5.98% of the observations belonging to the positive class, this will be taken into account when building the models.

Variables type

- According to the data description, the only variables with a truly numeric interpretation are "MAANTHUI" and "MGEMOMV".
- the variables "MOSTYPE" and "MOSHOOFD" are the only variables that need to be one hot encoded because they represent concepts, such as religion, social class, etc... in particular their encoding is nominal, so it is not possible to consider them as numeric variables.
- all the other variables are encoded as numeric but they are actually categorical variables. These variables are encoded using an encoding that is ordinal, so considering them as numeric seems to be a suitable solution (L1, L3, L4 encodings).
- the "CARAVAN" variable (response variable) belongs to the L4 encoding. If we check the summary of the "CARAVAN" variable, we can see that the minimum value is 0 and the maximum value is 1: this means that the variable is binary, so we can consider it as a binary variable.

Correlation matrix

```
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
```

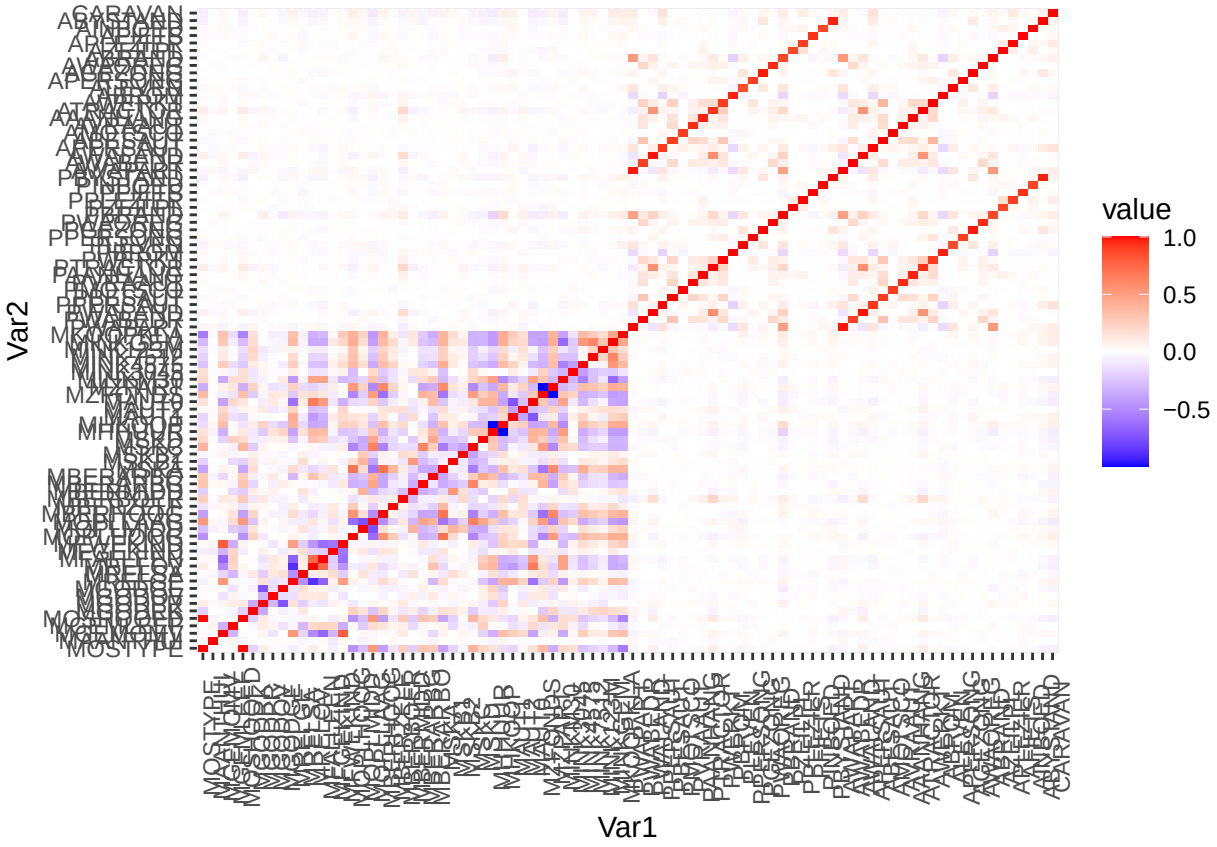
```
library(tidyr)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
cormat <- train  %>% cor()
mcor <- cormat %>% melt()

mcor %>% ggplot(
  aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  scale_fill_gradient2(low= "blue", high = "red", mid = "white") +
  theme(axis.text.x = element_text(angle = 90))
```




```
ggsave("graphs/correlation_matrix.png", width = 15, height = 15)
```

We can see that there are some variables that are highly correlated with each other: looking at the upper right quadrant, we can notice that each variable that starts with 'P', has a high correlation with a respective variable that start with 'A'... this means that having both in our data will likely provide little value to the analysis.

```
# Correlation between CARAVAN and other variables
cor_target <- cor(train$CARAVAN, train[, -which(names(train) == "CARAVAN")])

print(max(abs(cor_target)))
```

```
## [1] 0.1509097
```

Furthermore there does not seem to be any explanatory variable which is strictly correlated to our response variable.

Preprocessing

In the following chunks we will perform variables selection, one hot encode the variables that have a categorical nominal nature, perform feature scaling and, finally, address the dataset unbalance using a specific oversampling technique.

Variables selection

Examining the correlation plot reveals significant multicollinearity among certain variables. In this section, we will eliminate variables with high correlation coefficients (exceeding a threshold) with each other, excluding the target variable.

```r
library(caret)
```

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

```r
corr_matrix <- cor(train) # correlation matrix
multicorr_columns <- c()  # to store the names of the columns that are highly correlated with each othe
n_col <- ncol(corr_matrix) - 1 # excluding the target variable
corr_threshold <- 0.7 # threshold for which we consider two variables to be highly correlated

for (i in seq_len(n_col - 1)) {
  for (j in seq(i + 1, n_col)) {
    if (abs(corr_matrix[i, j]) > corr_threshold) { # is the correlation higher than the threshold?
      print(paste(colnames(corr_matrix)[i], colnames(corr_matrix)[j],
                  corr_matrix[i, j]))
      multicorr_columns <- c(multicorr_columns, colnames(corr_matrix)[j])
    }
  }
}
```

```
## [1] "MOSTYPE MOSHOOFD 0.992671873587132"
## [1] "MGEMOMV MFWEKIND 0.79401424585324"
## [1] "MGODPR MGODGE -0.741894510538976"
## [1] "MRELGE MRELOV -0.884361887633466"
## [1] "MRELOV MFALLEEN 0.745642268891389"
## [1] "MOPLMIDD MOPLLAAG -0.747581567682011"
## [1] "MHHUUR MHKOOP -0.999553947445255"
## [1] "MAUT1 MAUT0 -0.734563718602477"
## [1] "MZFONDS MZPART -0.999239298766863"
## [1] "PWAPART AWAPART 0.981369175933941"
## [1] "PWABEDR AWABEDR 0.895407235111021"
## [1] "PWALAND AWALAND 0.987578617328343"
## [1] "PPERSAUT APERSAUT 0.916154488158895"
## [1] "PBESAUT ABESAUT 0.902995584471846"
## [1] "PMOTSCO AMOTSCO 0.904855171005691"
## [1] "PVRAAUT AVRAAUT 0.948663332964804"
## [1] "PAANHANG AAANHANG 0.966080507278704"
## [1] "PTRACTOR ATRACTOR 0.929817837170216"
## [1] "PWERKT AWERKT 0.909670650335681"
## [1] "PBROM ABROM 0.969707583964885"
## [1] "PLEVEN ALEVEN 0.850171117578362"
## [1] "PPERSONG APERSONG 0.897561652966948"
## [1] "PGEZONG AGEZONG 0.979968511448332"
## [1] "PWAOREG AWAOREG 0.948429866997941"
## [1] "PBRAND ABRAND 0.865535920168006"
## [1] "PZEILPL AZEILPL 0.870333893257791"
## [1] "PPLEZIER APLEZIER 0.904436445408933"
## [1] "PFIETS AFIETS 0.935854262111417"
## [1] "PINBOED AINBOED 0.875256477164632"
## [1] "PBYSTAND ABYSTAND 0.966238760602872"
```

```r
# multicorr_columns --> contains the names of the columns that are highly correlated with each other
```

These correlation values seem intuitive when we analyse what each variable represents:MOSTYPE is composed by subtypes of MOSHOOFD (costumer types). MGEMOMV(average household size) is linked to MFWEKIND(household with children). MGODPR(Protestant) is negatively correlated with MGODGE(no

religion), and so on. Over half of the correlated couples are composed by which differ only by the first letter being either "A" or "P". This is because for a given type of policy the dataset contains both a variable regarding the contribution amount (variables starting with "P") and a variable regarding the number of policies (variables starting with "A").

```
# remove those columns from the train and test sets:
train <- train[, !colnames(train) %in% multicorr_columns]
test <- test[, !colnames(test) %in% multicorr_columns]
```

One hot encoding

The variables MOSTYPE and MOSHOOFD require one-hot encoding. MOSHOOFD was removed during correlation analysis as it serves as a generalization of MOSTYPE.

```
one_hot <- function(data, variable_name) {
  data[[ variable_name ]] <- as.factor(data[[ variable_name ]]) # convert to factor
  dummy <- dummyVars(" ~ .", data = data)
  data <- data.frame(predict(dummy, newdata = data))
  return(data)
}

train <- one_hot(train, "MOSTYPE")
test <- one_hot(test, "MOSTYPE")
```

Features scaling

We will conduct Min-Max Normalization on each variable, considering that each variable may belong to a different domain. The target variable CARAVAN is not normalized, since it is a binary variable.

```
minmax <- function(data) {

  if ("MAANTHUI" %in% colnames(data)) { # Domain MAANTHUI:  [1,10]
    data$MAANTHUI <- (data$MAANTHUI - 1) / (10 - 1)
  }

  if ("MGEMOMV" %in% colnames(data)) { # Domain MGEMOMV and MGEMLEEF:  [1,6]
    data$MGEMOMV <- (data$MGEMOMV - 1) / (6 - 1)
  }
  if ("MGEMLEEF" %in% colnames(data)) {
    data$MGEMLEEF <- (data$MGEMLEEF - 1) / (6 - 1)
  }

  for (i in colnames(data)) { # All the others have domain [0, 9]
    if (i != "CARAVAN" &&
        i != "MAANTHUI" &&
        i != "MGEMOMV" &&
        i != "MGEMLEEF") {
      data[[i]] <- (data[[i]] - 0) / (9 - 0)
    }
  }
  return(data)
}

# apply the minmax function to the train and test sets:
train <- minmax(train)
test <- minmax(test)
```

```r
write.csv(test, "data/test.csv", row.names = FALSE)
```

Addressing dataset unbalance

The dataset exhibits significant class imbalance, with only 5.68% of observations belonging to the positive class. To mitigate this issue, we will utilize the *ROSE* (Random Over-Sampling Examples) technique to generate synthetic samples of the minority class through oversampling.

```r
library(ROSE)
```

```
## Loaded ROSE 0.0-4
```

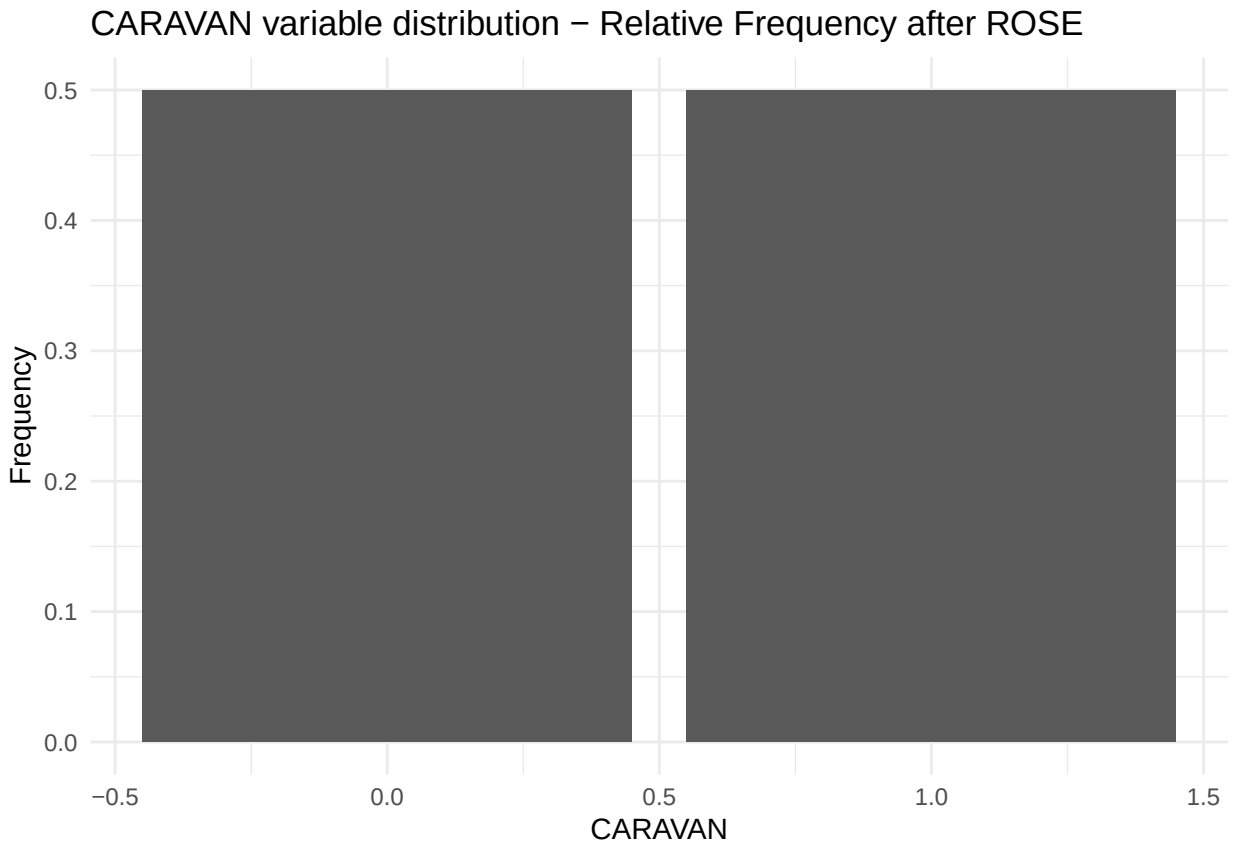
```r
n_0 <- nrow(train[train["CARAVAN"] == 0, ]) # number of observations in the majority class
rose <- ovun.sample(CARAVAN ~ .,
                    data = train,
                    method = "over",
                    N = 2 * n_0)

# write the new balanced training set:
write.csv(rose$data, "data/train_rose.csv", row.names = FALSE)
```

```r
# check the balance of the new training set:
prop.table(table(rose$data$CARAVAN))
```

```
##
##   0   1
## 0.5 0.5
```

```r
# relaltive frequency plot:
ggplot(rose$data, aes(x = CARAVAN)) +
  geom_bar(aes(y = (after_stat(count))/sum(after_stat(count)))) +
  labs(
    title = "CARAVAN variable distribution - Relative Frequency after ROSE",
    x = "CARAVAN", y = "Frequency"
  ) + theme_minimal()
```




```
ggsave("graphs/ROSE_rel_freq_CARAVAN.png", width = 10, height = 10)
```