# Credit Classification: Supervised Machine Learning

Gabriel D'assumpção de Carvalho

2025-07-14

# Libraries

```
# install.packages("ggplot2")
# install.packages("plotly")
# install.packages("caTools")
# install.packages("e1071")
# install.packages("class")

# Graphics
library(plotly)
library(ggplot2)
library(rpart.plot)

# Data Manipulation
library(caTools)
library(dplyr)

# Machine Learning
library(e1071) # For SVM
library(class) # For KNN
library(rpart) # For Decision Trees
library(caret) # For confusionMatrix

# Warning and message suppression
options(warn = -1)
suppressMessages(library(ggplot2))
suppressMessages(library(dplyr))
suppressMessages(library(caret))

set.seed(42)
```

# Introduction

# Exploratory Data Analysis (EDA)

## Data Import

```
# Load the dataset
df <- read.csv("/home/gabrieldadcarvalho/github/actuarial_seminar/data/original.csv")[, -1]
print(head(df))
```

```
##     income      age      loan default
## 1 66155.93 59.01702 8106.5321       0
## 2 34415.15 48.11715 6564.7450       0
## 3 57317.17 63.10805 8020.9533       0
## 4 42709.53 45.75197 6103.6423       0
## 5 66952.69 18.58434 8770.0992       1
## 6 24904.06 57.47161   15.4986       0
```

## Data Preprocessing

```
# Check statistics of the dataset
summary(df)
```

```
##      income           age              loan              default
##  Min.   :20014   Min.   :-52.42   Min.   :    1.378   Min.   :0.0000
##  1st Qu.:32796   1st Qu.: 28.99   1st Qu.: 1939.709   1st Qu.:0.0000
##  Median :45789   Median : 41.32   Median : 3974.719   Median :0.0000
##  Mean   :45332   Mean   : 40.81   Mean   : 4444.370   Mean   :0.1415
##  3rd Qu.:57791   3rd Qu.: 52.59   3rd Qu.: 6432.411   3rd Qu.:0.0000
##  Max.   :69996   Max.   : 63.97   Max.   :13766.051   Max.   :1.0000
##                  NA's   :3
```

```
print(df$age[df$age <= 0])
```

```
## [1] -28.21836 -52.42328 -36.49698        NA        NA        NA
```

Analyzing the statistics of the variables, we can see that the `age` variable has some tree NaN and negative values. Below i will converter the negative values to positive values, and for the NaN values, we discussed some imputation methods to handle these missing values.

```
# Convert age for positive values
df$age <- abs(df$age)
print(summary(df$age))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   18.06   29.03   41.35   40.92   52.59   63.97       3
```

We can see that the variable `age` has a tree NaN values. We can apply some imputation methods to handle these missing values. For example:

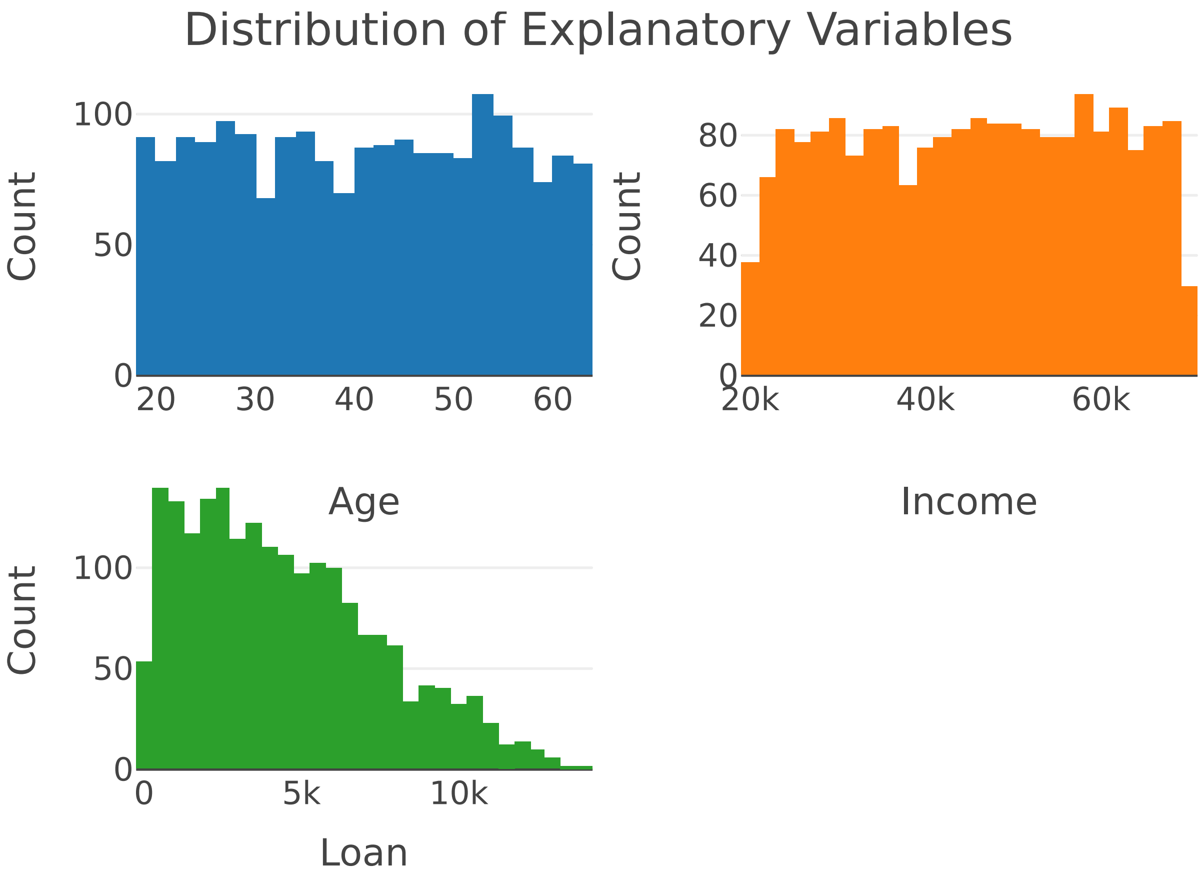
- Mean imputation: Replace NaN values with the mean of the column.
- Median imputation: Replace NaN values with the median of the column.
- Linear Regression imputation: Use linear regression to predict missing values based on other variables.
- Regression imputation: Use regression models to predict missing values based on other variables.
- Interpolation: Use interpolation methods to estimate missing values based on surrounding data points.
- Exploratory Data Analysis (EDA): Analyze the data to understand the distribution and value intervals of the variables, and then apply one statistics to replace the NaN values.

# Exploratory Data Analysis (EDA)

```
p1 <- plot_ly(df, x = ~age, type = "histogram", name = "Age") %>%
  layout(title = "Age", xaxis = list(title = "Age"), yaxis = list(title = "Count"), showlegend = FALSE)

p2 <- plot_ly(df, x = ~income, type = "histogram", name = "Income") %>%
  layout(title = "Income", xaxis = list(title = "Income"), yaxis = list(title = "Count"), showlegend = F

p3 <- plot_ly(df, x = ~loan, type = "histogram", name = "Loan") %>%
  layout(title = "Loan", xaxis = list(title = "Loan"), yaxis = list(title = "Count"), showlegend = FALSE

subplot(p1, p2, p3,
  nrows = 2, margin = 0.07,
  titleX = TRUE, titleY = TRUE,
  shareX = FALSE, shareY = FALSE
```

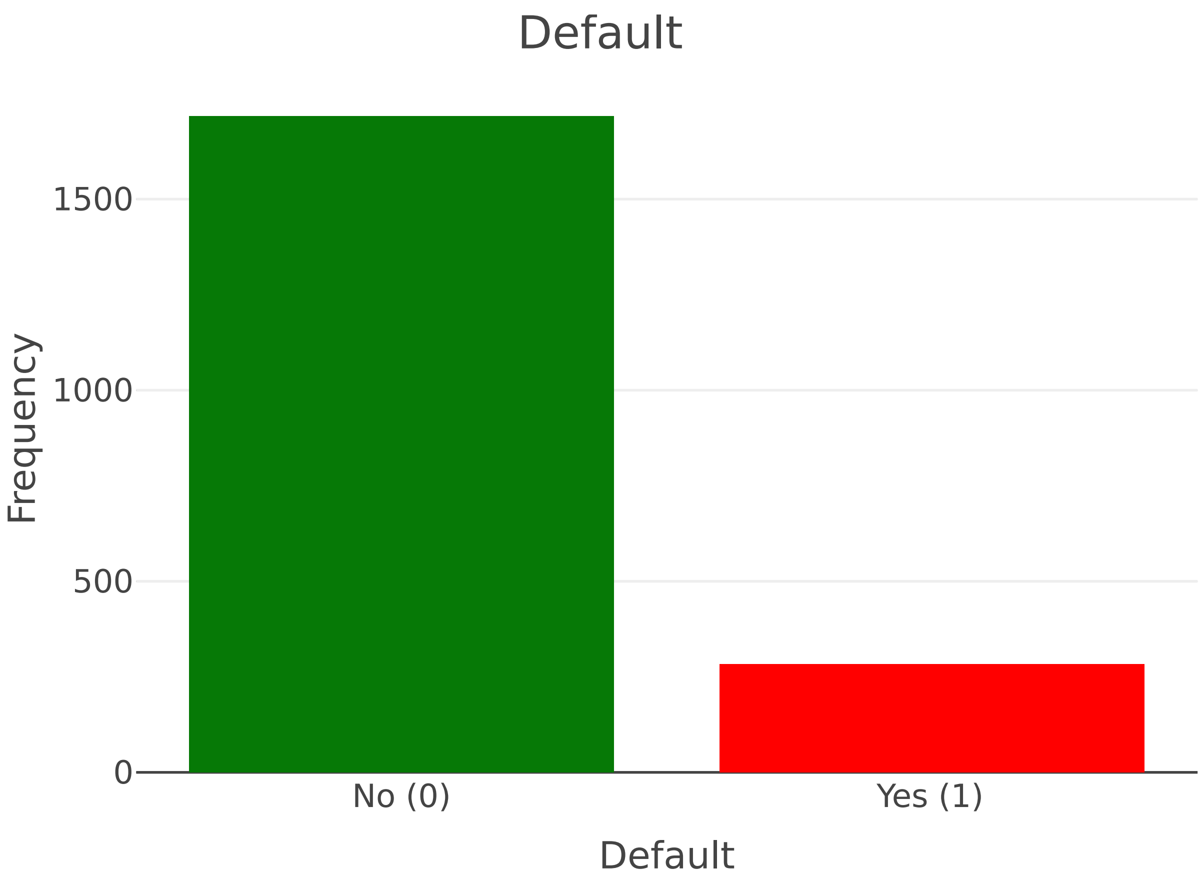
```
) %>%
  layout(title = "Distribution of Explanatory Variables")
```

```
## `google-chrome` and `chromium-browser` were not found. Try setting the `CHROMOTE_CHROME` environment
```




The `age` and `income` variables have a similar uniform distribution, while the `loan` variable has an asymmetric positive distribution. This affirmation is confirmed by the summary statistics of the dataset and the above plots.

```
df %>%
  count(default) %>%
  plot_ly(
    x = ~ factor(default, levels = c(0, 1), labels = c("No (0)", "Yes (1)")),
    y = ~n,
    type = "bar",
    color = ~ factor(default),
    colors = c("#067906", "red"),
    name = "Default Variable"
  ) %>%
  layout(
    title = "Default",
    xaxis = list(title = "Default", type = "category"),
    yaxis = list(title = "Frequency"),
    showlegend = FALSE
  )
```




```r
corrPearson <- cor(df[, !names(df) %in% "default"], method = "pearson", use = "pairwise.complete.obs")
corrSpearman <- cor(df[, !names(df) %in% "default"], method = "spearman", use = "pairwise.complete.obs"
print(corrPearson)
```

```
##              income          age        loan
## income  1.00000000 -0.034000535 0.441116504
## age    -0.03400054  1.000000000 0.006440323
## loan    0.44111650  0.006440323 1.000000000
```

```r
print(corrSpearman)
```

```
##              income          age        loan
## income  1.00000000 -0.034486684 0.401601061
## age    -0.03448668  1.000000000 0.009835956
## loan    0.40160106  0.009835956 1.000000000
```

### Linear Regression For Imputation data

For imputation, will be used the Linear Regression algorithm, which is a simple and effective methods for handling missing values. The Linear Regression algorithm works by finding the relationship between the target variable and the other features to predict the missing values.

```r
# Get the missing values in the 'age' column
ageNan <- df[is.na(df$age), ]
print(ageNan)
```

```
##      income age     loan default
## 29 59417.81  NA 2082.626       0
```

```
## 31 48528.85  NA 6155.785       0
## 32 23526.30  NA 2862.010       0
```

```
# Remove rows with NaN in 'age' for training and testing
dfNN <- df[!is.na(df$age), ]
print(summary(dfNN))
```

```
##      income           age             loan               default
##  Min.   :20014   Min.   :18.06   Min.   :    1.378   Min.   :0.0000
##  1st Qu.:32805   1st Qu.:29.03   1st Qu.: 1936.813   1st Qu.:0.0000
##  Median :45789   Median :41.35   Median : 3977.287   Median :0.0000
##  Mean   :45334   Mean   :40.92   Mean   : 4445.488   Mean   :0.1417
##  3rd Qu.:57788   3rd Qu.:52.59   3rd Qu.: 6440.861   3rd Qu.:0.0000
##  Max.   :69996   Max.   :63.97   Max.   :13766.051   Max.   :1.0000
```

### Split the dataset into training and testing sets

```
# Split the dataset into training and testing sets
split <- sample.split(dfNN$age, SplitRatio = 0.8)
train <- subset(dfNN, split == TRUE)
test <- subset(dfNN, split == FALSE)
```

### Normalize the numeric columns

```
# Select numeric columns, excluding 'default'
numeric_cols <- sapply(train, is.numeric)
cols_for_stats <- names(train)[numeric_cols & names(train) != "default"]

# Calculate statistics only for the selected columns
means <- colMeans(train[, cols_for_stats], na.rm = TRUE)
sds <- apply(train[, cols_for_stats], 2, sd, na.rm = TRUE)

# Z-score normalization
for (c in colnames(train[(numeric_cols & names(train) != "default")])) {
  if (is.numeric(train[[c]])) {
    train[[c]] <- (train[[c]] - means[c]) / sds[c]
    test[[c]] <- (test[[c]] - means[c]) / sds[c]
    ageNan[[c]] <- (ageNan[[c]] - means[c]) / sds[c]
    df[[c]] <- (df[[c]] - means[c]) / sds[c]
  }
}

print(summary(train))
```

```
##      income              age                 loan              default
##  Min.   :-1.76492   Min.   :-1.73218   Min.   :-1.4454   Min.   :0.0000
##  1st Qu.:-0.87437   1st Qu.:-0.89946   1st Qu.:-0.8240   1st Qu.:0.0000
##  Median : 0.03517   Median : 0.03068   Median :-0.1510   Median :0.0000
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   :0.1403
##  3rd Qu.: 0.87019   3rd Qu.: 0.89202   3rd Qu.: 0.6672   3rd Qu.:0.0000
##  Max.   : 1.72498   Max.   : 1.71814   Max.   : 3.0545   Max.   :1.0000
```

```
print(summary(test))
```

```
##      income               age                 loan               default
```

```
##  Min.   :-1.768317   Min.   :-1.71332   Min.   :-1.4234   Min.   :0.0000
##  1st Qu.:-0.869711   1st Qu.:-0.94137   1st Qu.:-0.7226   1st Qu.:0.0000
##  Median : 0.030767   Median :-0.03091   Median :-0.1195   Median :0.0000
##  Mean   : 0.006546   Mean   :-0.06830   Mean   : 0.0373   Mean   :0.1475
##  3rd Qu.: 0.877523   3rd Qu.: 0.74539   3rd Qu.: 0.6149   3rd Qu.:0.0000
##  Max.   : 1.703976   Max.   : 1.71162   Max.   : 2.9491   Max.   :1.0000
```

```
print(summary(ageNan))
```

```
##      income             age           loan            default
##  Min.   :-1.5229   Min.   : NA   Min.   :-0.7650   Min.   :0
##  1st Qu.:-0.6491   1st Qu.: NA   1st Qu.:-0.6376   1st Qu.:0
##  Median : 0.2246   Median : NA   Median :-0.5102   Median :0
##  Mean   :-0.1042   Mean   :NaN   Mean   :-0.2362   Mean   :0
##  3rd Qu.: 0.6051   3rd Qu.: NA   3rd Qu.: 0.0282   3rd Qu.:0
##  Max.   : 0.9857   Max.   : NA   Max.   : 0.5666   Max.   :0
##                    NA's   :3
```

### Train the Linear Regression model

```
# Train a linear regression model to predict 'age'
model <- lm(age ~ ., data = train)
```

```
print(summary(model))
```

```
##
## Call:
## lm(formula = age ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.02876 -0.62141  0.04569  0.63442  2.07822
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.22153    0.02372   9.338  < 2e-16 ***
## income      -0.18383    0.02448  -7.509 9.88e-14 ***
## loan         0.29675    0.02654  11.180  < 2e-16 ***
## default     -1.57939    0.06889 -22.927  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8658 on 1593 degrees of freedom
## Multiple R-squared:  0.2518, Adjusted R-squared:  0.2504
## F-statistic: 178.7 on 3 and 1593 DF,  p-value: < 2.2e-16
```

### Test the model

```
# Predict missing 'age' values in the test set
predictedAge <- predict(model, newdata = test)
```

```
# Mean Squared Error (MSE) for the predictions
mse <- 0
for (i in 1:length(predictedAge)) {
  mse = mse + (predictedAge[i] - test$age[i])^2
```

```
}

mse = mse / length(predictedAge)
print(mse)
```

```
##         1
## 0.7245957
```

**Impute missing values in the original dataset**

```
# Impute missing values in the original dataset
ageNan$age <- predict(model, newdata = ageNan)

# Replace NaN values in the original dataset with the predicted values
df$age[is.na(df$age)] <- ageNan$age

# Reverse Z-score normalization for the imputed values
for (c in colnames(df[(numeric_cols & names(df) != "default")])) {
  if (is.numeric(df[[c]])) {
    df[[c]] <- (df[[c]] * sds[c]) + means[c]
    ageNan[[c]] <- (ageNan[[c]] * sds[c]) + means[c]
  }
}

print(head(ageNan))
```

```
##      income      age     loan default
## 29 59417.81 38.62265 2082.626       0
## 31 48528.85 45.74307 6155.785       0
## 32 23526.30 45.76578 2862.010       0
```

```
print(summary(df))
```

```
##      income           age             loan              default
##  Min.   :20014   Min.   :18.06   Min.   :    1.378   Min.   :0.0000
##  1st Qu.:32796   1st Qu.:29.06   1st Qu.: 1939.709   1st Qu.:0.0000
##  Median :45789   Median :41.38   Median : 3974.719   Median :0.0000
##  Mean   :45332   Mean   :40.93   Mean   : 4444.370   Mean   :0.1415
##  3rd Qu.:57791   3rd Qu.:52.58   3rd Qu.: 6432.411   3rd Qu.:0.0000
##  Max.   :69996   Max.   :63.97   Max.   :13766.051   Max.   :1.0000
```

# Predict Default

## Data Preparation

```
split <- sample.split(df$default, SplitRatio = 0.8)
train <- subset(df, split == TRUE)
test <- subset(df, split == FALSE)

means <- colMeans(train[, cols_for_stats], na.rm = TRUE)
sds <- apply(train[, cols_for_stats], 2, sd, na.rm = TRUE)
```

```
# Z-score normalization
for (c in colnames(train[(numeric_cols & names(train) != "default")])) {
```

```
  if (is.numeric(train[[c]])) {
    train[[c]] <- (train[[c]] - means[c]) / sds[c]
    test[[c]] <- (test[[c]] - means[c]) / sds[c]
  }
}
```

### Logistic Regression

```
# Train a logistic regression model
logistic_model <- glm(default ~ ., data = train, family = binomial(link = "logit"))

print(summary(logistic_model))
```

```
##
## Call:
## glm(formula = default ~ ., family = binomial(link = "logit"),
##     data = train)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -8.1968     0.6598  -12.42   <2e-16 ***
## income       -3.6457     0.3643  -10.01   <2e-16 ***
## age          -5.0026     0.4248  -11.78   <2e-16 ***
## loan          5.6158     0.4907   11.45   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1303.12  on 1599  degrees of freedom
## Residual deviance:  323.71  on 1596  degrees of freedom
## AIC: 331.71
##
## Number of Fisher Scoring iterations: 9
```

```
# Predict on the test set
logistic_pred <- predict(logistic_model, newdata = test, type = "response")
logistic_pred_class <- ifelse(logistic_pred > 0.5, 1, 0)

head(data.frame(Prediction = logistic_pred_class, Probability = logistic_pred))
```

```
##    Prediction  Probability
## 3           0 2.330707e-06
## 4           0 1.948964e-03
## 5           1 9.395945e-01
## 9           0 7.029250e-06
## 10          0 2.947396e-04
## 13          0 1.853924e-02
```

```
# Confusion matrix for logistic regression
confusionMatrix(as.factor(test$default), as.factor(logistic_pred_class), dnn = c("Reference", "Predicti
```

```
## Confusion Matrix and Statistics
##
##          Prediction
```

```
## Reference   0   1
##          0 327  16
##          1  14  43
##
##                Accuracy : 0.925
##                  95% CI : (0.8947, 0.9488)
##     No Information Rate : 0.8525
##     P-Value [Acc > NIR] : 6.876e-06
##
##                   Kappa : 0.6975
##
##  Mcnemar's Test P-Value : 0.8551
##
##             Sensitivity : 0.9589
##             Specificity : 0.7288
##          Pos Pred Value : 0.9534
##          Neg Pred Value : 0.7544
##              Prevalence : 0.8525
##          Detection Rate : 0.8175
##    Detection Prevalence : 0.8575
##       Balanced Accuracy : 0.8439
##
##        'Positive' Class : 0
##
```

## Suport Vector Machine (SVM)

```
# Train a Support Vector Machine (SVM) model
svm_model <- svm(default ~ ., data = train, kernel = "radial", cost = 1, gamma = 3)

summary(svm_model)
```

```
##
## Call:
## svm(formula = default ~ ., data = train, kernel = "radial", cost = 1,
##     gamma = 3)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  3
##     epsilon:  0.1
##
##
## Number of Support Vectors:  524
```

```
# Predict on the test set using SVM
svm_pred <- predict(svm_model, newdata = test)
svm_pred_class <- ifelse(svm_pred >= 0.5, 1, 0)

head(data.frame(Previsto = svm_pred_class, Prob = svm_pred))
```

```
##    Previsto        Prob
```

```
## 3          0  0.03786639
## 4          0  0.03801636
## 5          1  0.85102751
## 9          0  0.03643876
## 10         0  0.02738613
## 13         0 -0.01139708
```

```
levels_ref <- c("0", "1")

reference <- factor(test$default, levels = levels_ref)
prediction <- factor(svm_pred_class, levels = levels_ref)

# Confusion matrix for SVM
confusionMatrix(reference, prediction, dnn = c("Reference", "Prediction"))
```

```
## Confusion Matrix and Statistics
##
##          Prediction
## Reference   0   1
##         0 342   1
##         1   9  48
##
##                Accuracy : 0.975
##                  95% CI : (0.9545, 0.9879)
##     No Information Rate : 0.8775
##     P-Value [Acc > NIR] : 1.758e-12
##
##                   Kappa : 0.8913
##
##  Mcnemar's Test P-Value : 0.02686
##
##             Sensitivity : 0.9744
##             Specificity : 0.9796
##          Pos Pred Value : 0.9971
##          Neg Pred Value : 0.8421
##              Prevalence : 0.8775
##          Detection Rate : 0.8550
##    Detection Prevalence : 0.8575
##       Balanced Accuracy : 0.9770
##
##        'Positive' Class : 0
##
```

## Decision tree

### Data Preparation

```
split <- sample.split(df$default, SplitRatio = 0.8)
train <- subset(df, split == TRUE)
test <- subset(df, split == FALSE)

means <- colMeans(train[, cols_for_stats], na.rm = TRUE)
sds <- apply(train[, cols_for_stats], 2, sd, na.rm = TRUE)

trainT <- train["default"]
```

```r
testT <- test["default"]

trainT$ageInterval <- cut(train$age, breaks = c(18, 40, 60, Inf), right = FALSE)
testT$ageInterval <- cut(test$age, breaks = c(18, 40, 60, Inf), right = FALSE)

# Menos intervalos para income
trainT$incomeInterval <- cut(train$income, breaks = c(20000, 50000, 100000, Inf), right = FALSE)
testT$incomeInterval <- cut(test$income, breaks = c(20000, 50000, 100000, Inf), right = FALSE)

# Menos intervalos para loan
trainT$loanInterval <- cut(train$loan, breaks = c(0, 5000, 10000, Inf), right = FALSE)
testT$loanInterval <- cut(test$loan, breaks = c(0, 5000, 10000, Inf), right = FALSE)
```

```r
ageClass <- table(trainT$ageInterval, trainT$default)
print(ageClass)
```

```
##
##              0   1
##   [18,40)  523 226
##   [40,60)  713   0
##   [60,Inf) 138   0
```

```r
incomeClass <- table(trainT$incomeInterval, trainT$default)
print(incomeClass)
```

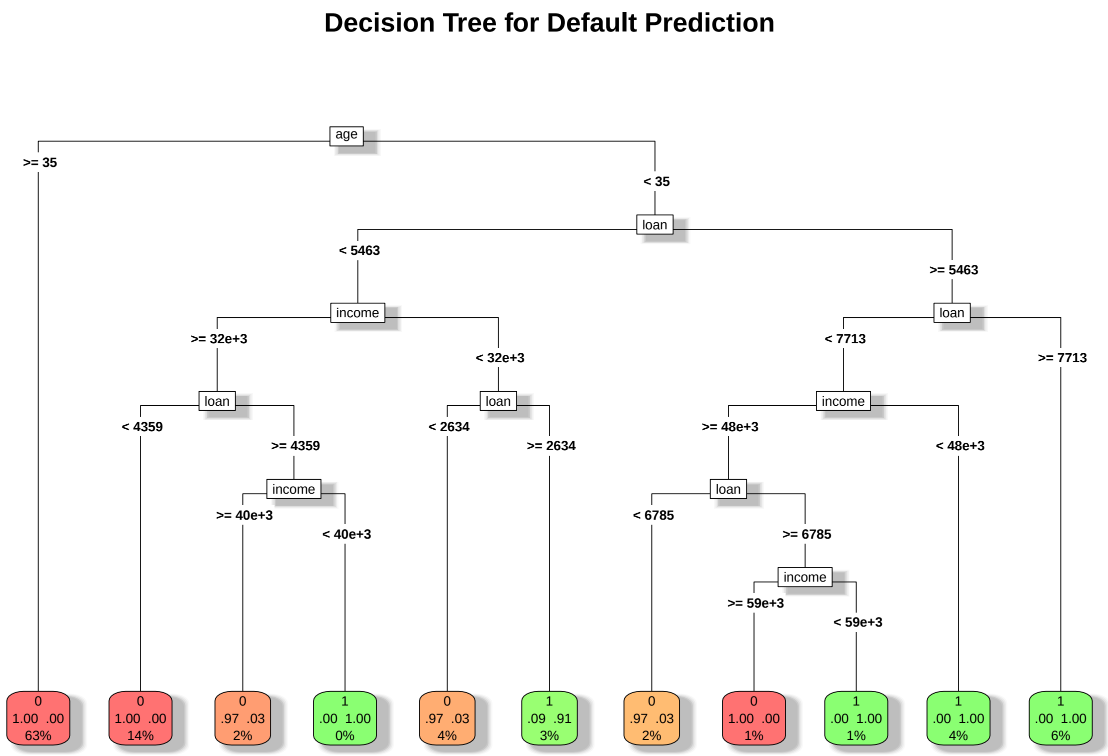
```
##
##                   0   1
##   [2e+04,5e+04) 810 135
##   [5e+04,1e+05) 564  91
##   [1e+05,Inf)     0   0
```

```r
loanClass <- table(trainT$loanInterval, trainT$default)
print(loanClass)
```

```
##
##                   0   1
##   [0,5e+03)     922  50
##   [5e+03,1e+04) 392 141
##   [1e+04,Inf)    60  35
```

```r
dt_model <- rpart(default ~., data=train, method = "class")
```

```r
rpart.plot(dt_model, type=5, extra=104, main="Decision Tree for Default Prediction", box.palette = "RdYl
```




### Predicting

```r
predictions <- predict(dt_model, newdata = test, type = "class")
```

```r
# Confusion matrix for Decision Tree
confusionMatrix(as.factor(test$default), as.factor(predictions), dnn = c("Reference", "Prediction"))
```

```
## Confusion Matrix and Statistics
##
##          Prediction
## Reference   0   1
##         0 340   3
##         1   5  52
##
##                Accuracy : 0.98
##                  95% CI : (0.961, 0.9913)
##     No Information Rate : 0.8625
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9169
##
##  Mcnemar's Test P-Value : 0.7237
##
##             Sensitivity : 0.9855
##             Specificity : 0.9455
##          Pos Pred Value : 0.9913
##          Neg Pred Value : 0.9123
##              Prevalence : 0.8625
##          Detection Rate : 0.8500
##    Detection Prevalence : 0.8575
```

```
##       Balanced Accuracy : 0.9655
##
##        'Positive' Class : 0
##
```