Failure Prediction in Air Pressure System of Scania Trucks

Souliotis Thomas

December 2020

Contents

1. Introduction

The purpose of this report is to present an approach for predicting failures in the Air Pressure System of Scania Trucks. The project is performed under the scope of HarvardX: PH125.9x Data Science, Capstone course which indicates to use a publicly available dataset to solve a problem of choice by applying data analysis and machine learning techniques. The target of the project is to develop and train machine learning algorithms in order to predict the failure in the air pressure system, thus improving maintenance quality and minimizing the subsequent cost. For this reason, a dedicated training set and a separate test set are provided. The presented work is based on the tools and knowledge that has been provided throughout the Data Science programm.

Having a degree in Mechanical Engineering and working background on powertrain engineering in the automotive sector, the choice of this case study was motivated by the fact that advancements utilizing digitalization are emerging in engineering sectors. In the respective field, Digitalization and IoT technologies are becoming key drivers for development of solutions for improvement and more cost-effective life design processes. Already, real time data analysis for various machinery is adopted and predictive maintenance strategies using machine learning have been implemented.

The respective publication for the examined problem, released by Scania CV AB, states that traditional maintenance plans for heavy trucks are static and shifts the interest towards solutions that utilize operating data to influence the maintenance intervals. Traditionally, for each application type and vehicle specification, a cyclic maintenance plan is given as the number of kilometers between maintenance occasions, with fixed maintenance protocols. As a result, maintenance plans are seldom updated even if the application of the vehicle changes. Furthermore, maintenance points are grouped together with the effect that some components are maintained more than necessary. The latter issue can not be easily addressed as it would require to create correct physical wear-models for several components. Currently, the advent of affordable telematics solutions, has facilitated the acquisition of data from on-board sensors from trucks in operation. In this respect, information of how the truck has been utilized can be taken taken into account in order to dynamically adjust maintenance program, to align with the truck's actual need.

The challenge of this project is to minimize the maintenance costs by effectively predicting failure events related to the APS. It is a classification problem for cost minimization and the cost metric for miss-classification is defined in the instructions. The cost of falsely predicting a failure, which is associated with an unnecessary check by a mechanic at a workshop, is 10, while the the cost of missing a faulty truck, which may cause a breakdown, is 500.

To achieve this, an initial exploratory analysis is performed in order to shape an understanding of the features provided. Two different training data sets are created by applying feature reduction and data balancing techniques. The first is a training dataset with a reduced amount of features while the other is extended, using more features. Several methods related to binary classification are applied, starting from a simplified prediction and extending to more complex methods with several tuning parameters, like Gradient Boosted Decision Trees. The results clearly show that the application of machine learning to accomplish predictive maintenance can be very beneficial.

The report is structured in 7 chapters. The 1st chapter is the introduction. In the 2nd chapter the dataset is presented, while in the 3d chapter the dataset analysis is performed in order to shape an understanding of the features provided and to generate the datasets to be used for the algorithms training. In the 4th chapter the modeling approach is presented and a detailed explanation is provided for all the models implemented. In the 5th chapter the results are discussed and in the 6th chapter the conclusions of the work performed are presented. In the 7th chapter the references used to perform this study are cited.

2. Dataset

The dataset is part of APS Failure and Operational Data for Scania Trucks and consists of data collected from heavy Scania trucks in everyday usage. The system in focus is the Air Pressure system (APS) which generates pressurized air that is utilized in various functions in a truck, such as braking and gear changes. There are only 2 classes in the data, positive and negative. The positive class consists of component failures for a specific component of the APS system, while the negative class consists of trucks with failures for components not related to the APS. The data consists of a subset of all available data, selected by experts.

The respective data and the corresponding information can be can be found in the following path:

- [APS Failures in Scania Trucks] https://www.kaggle.com/uciml/aps-failure-at-scania-trucks-data-set

In the repository there are separate training and test datasets. Moreover, there are two versions for each dataset, a version with the raw data that were provided and a version with pre-processed data that could be used as starting point. Since data wrangling and pre-processing is an important process of data analysis and the datasets that are fed on the machine learning models have a significant impact on the quality of the prediction, the raw datasets will be used. – and are loaded with the following code:

```
## Load training and test datasets from github repository
train_data<-read.csv("https://raw.githubusercontent.com/TomSouliot/Data_Science_Capstone/main/aps_failu
test_data<-read.csv("https://raw.githubusercontent.com/TomSouliot/Data_Science_Capstone/main/aps_failur

## Global setting
options(digits = 3)

## Create the Datasets
#-----------------------------------------------------
## Train Dataset
y_train<-as.factor(train_data$class)
x_train<-sapply(train_data, as.numeric)
x_train<-as.data.frame(x_train)%>%select(-class)

## Test Dataset
y_test<-as.factor(test_data$class)
x_test<-sapply(test_data, as.numeric)
x_test<-as.data.frame(x_test)%>%select(-class)
#-----------------------------------------------------
```

The **training set** will be used in order to train the machine learning models, while the **test set** will be used only to evaluate their performance. The **training set** contains **60000** examples in total, while the **test set** contains **16000** examples. There are **170** features per record that correspond to data that were collected from on-board sensors of trucks in operation. The features' names have been anonymized for proprietary reasons. The features are either single numerical counters or bins as part of histograms. For example, a histogram could be defined for the ambient temperature with 4 bins for defined temperature ranges:

- bin 1 collect values for temperature T < -20
- bin 2 collect values for temperature T >= -20 and T < 0
- bin 3 collect values for temperature T >= 0 and T < 20
- bin 4 collect values for temperature T > 20

A first overview of the train dataset and the features is presented in the table below:

Table 1: Dataset Structure

variable	class	First_Values
aa_000	double	76698, 33058, 41040, 12, 60874, 38312
ab_000	double	NA, NA, NA, 0, NA, NA
ac_000	double	2130706438, 0, 228, 70, 1368, 2130706432
ad_000	double	280, NA, 100, 66, 458, 218
ae_000	double	0, 0, 0, 0, 0, 0
af_000	double	0, 0, 0, 10, 0, 0
ag_000	double	0, 0, 0, 0, 0, 0
ag_001	double	0, 0, 0, 0, 0, 0
ag_002	double	0, 0, 0, 0, 0, 0
ag_003	double	0, 0, 0, 318, 0, 0
ag_004	double	37250, 18254, 1648, 2212, 43752, 9128
ag_005	double	1432864, 653294, 370592, 3232, 1966618, 701702
ag_006	double	3664156, 1720800, 1883374, 1872, 1800340, 1462836
ag_007	double	1007684, 516724, 292936, 0, 131646, 449716
ag_008	double	25896, 31642, 12016, 0, 4588, 39000
ag_009	double	0, 0, 0, 0, 0, 660
ah_000	double	2551696, 1393352, 1234132, 2668, 1974038, 1087760
ai_000	double	0, 0, 0, 0, 0, 0
aj_000	double	0, 68, 0, 0, 226, 0
ak_000	double	0, 0, 0, 0, 0, 0
al_000	double	0, 0, 0, 642, 0, 0
am_0	double	0, 0, 0, 3894, 0, 0
an_000	double	4933296, 2560898, 2371990, 10184, 3230626, 2283060
ao_000	double	3655166, 2127150, 2173634, 7554, 2618878, 1892752
ap_000	double	1766008, 1084598, 300796, 10764, 1058136, 469244
aq_000	double	1132040, 338544, 153698, 1014, 551022, 347054
ar_000	double	0, 0, 0, 0, 0, 0
as_000	double	0, 0, 0, 0, 0, 0
at_000	double	0, 0, 0, 0, 0, 0
au_000	double	0, 0, 0, 0, 0, 0
av_000	double	1012, 0, 358, 60, 1788, 1142
ax_000	double	268, 0, 110, 6, 642, 452
ay_000	double	0, 0, 0, 0, 0, 0
ay_001	double	0, 0, 0, 0, 0, 0
ay_002	double	0, 0, 0, 0, 0, 0
ay_003	double	0, 0, 0, 0, 0, 0
ay_004	double	0, 0, 0, 0, 42124, 0
ay_005	double	469014, 71510, 0, 0, 372236, 280112
ay_006	double	4239660, 772720, 870456, 0, 2128914, 1160742
ay_007	double	703300, 1996924, 239798, 2038, 819596, 774914
ay_008	double	755876, 99560, 1450312, 5596, 584074, 447274
ay_009	double	0, 0, 0, 0, 0, 0
az_000	double	5374, 7336, 1620, 64, 1644, 1580
az_001	double	2108, 7808, 1156, 6, 362, 680
az_002	double	4114, 13776, 1228, 6, 562, 886
az_003	double	12348, 13086, 34250, 914, 842, 1270
az_004	double	615248, 1010074, 1811606, 76, 30194, 42450
az_005	double	5526276, 1873902, 710672, 2478, 3911734, 2615652
az_006	double	2378, 14726, 34, 2398, 1606, 524
az_007	double	4, 6, 0, 1692, 0, 0

3. Data Analysis

Exploratory analysis is a very important process that allows us to shape a better understanding of the data provided and their potential effect in developing the prediction algorithm. To achieve this, the structure of the dataset and respective statistical indicators are examined, while graphs that summarize important aspects of the dataset are created. Through this process, the necessary modifications are performed in the dataset, in order to generate an equivalent dataset that can be effectively used in order to train the machine learning algorithms. The purpose of the modifications performed is discussed and is supported by dedicated graphs.

Regarding the approach followed, in the beginning, features that belong to respective histograms are substituted by a single feature that represents the total operating time of the truck. Afterwards, features with zero variance are excluded and the missing values in the dataset are imputed. The data matrix is scaled and statistical indicators of the variables, their distribution and their correlation are examined. To decouple from correlated features and from associated multicollinearity, but most important to optimize the number of features, PCA is implemented. Finally, a data balancing technique is applied in order to reduce the imbalance between the negative and the positive class in the dataset.

Feature selection is very important, as it directly affects the computational complexity and the quality of the prediction algorithm. To quantify this trade off, 2 respective train datasets will be generated. A dataset that consists of a reduced amount of features and an extended.

To perform the required calculations to develop the failure prediction algorithm and to present the results, the following libraries were used:

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(knitr)
library(gridExtra)
library(kableExtra)
library(funModeling)
library(corrplot)
library(matrixStats)
library(DMwR)
library(doParallel)
library(caTools)
library(randomForest)
library(xgboost)
library(psych)
library(magrittr)
```

I. Histogram Features

As mentioned in the dataset description, there are features that correspond to bins of histograms. Each histogram refers to a different physical value or condition, for example ambient Temperature or Altitude. The bins of each histogram reflect the amount of time that the truck operated under the conditions characterizing the particular bin. For example how much time the truck operated in the range of 0-20°C ambient temperature, 20-40°C and so on.

By looking at the structure of the dataset, 70 features are identified that belong to 7 histograms. The sum across the bins of each histogram indicates the total operating time of the truck. In order to reduce the dimension of the dataset and to reduce the amount of features available, the 70 features will be substituted by an equivalent feature with the total operating time of the truck. This is the first step towards generating the *reduced* training dataset.

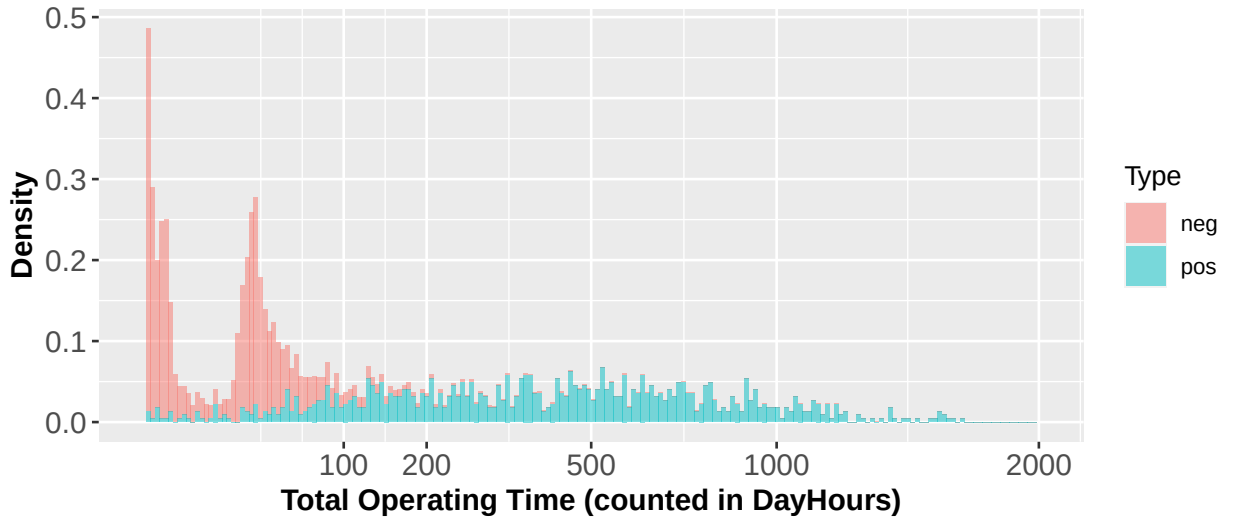
However, there is a high possibility that some of the failures of the APS, are more related to the operating time under specific conditions, hence specific bins. For example, the operating time in high altitude, or very cold and very hot ambient temperature could be strongly correlated with the deterioration and failure of the APS system. By replacing all histogram features With a single feature corresponding to the total operating time, this information and the correlation with the APS failure will be lost and can have an impact on the prediction quality. For that reason, for the *extended* training dataset we will not proceed to this modification and the features of the histogram bins will be included.

As a naming convention we will refer to the generated dataset with the histogram bins modification and the reduced amount of features as **reduced**, and to the generated dataset that includes the histograms bins and will use an extended amount of features, as **extended**. In the same respect, in order to avoid big names in the code, a naming convention is followed for the generated datasets. Using as example the dataset *x_train_m11*, it refers to a training dataset. The first counter, **1** refers to the **reduced** dataset and **2** to the **extended**. The second counter refers to the modification step during the post processing. Following this convention, *x_train_m23* would refer to the 3rd modification step of the **extended** training dataset. It is also important to mention that the modifications are applied to the respective test datasets as well for compatibility.

```
## The 7 histogram bins are ag, ay, az, ba, cn, cs, ee
#-------------------------------------------------------
# Histogram Bins
ag_hist= x_train[,c(7:16)]
ay_hist= x_train[,c(33:42)]
az_hist= x_train[,c(43:52)]
ba_hist= x_train[,c(53:62)]
cn_hist= x_train[,c(100:109)]
cs_hist= x_train[,c(114:123)]
ee_hist= x_train[,c(159:168)]
# Sum across bins
ag_cml = rowSums(ag_hist)
ay_cml = rowSums(ay_hist)
az_cml = rowSums(az_hist)
ba_cml = rowSums(ba_hist)
cn_cml = rowSums(cn_hist)
cs_cml = rowSums(cs_hist)
ee_cml = rowSums(ee_hist)

# Sum is the same for all bins
h=as.data.frame(cbind(ag_cml, ay_cml, az_cml, ba_cml, cn_cml, cs_cml, ee_cml))
h1<- rowSds(as.matrix(h))
h2<- rowMeans(as.matrix(h))
```

```
## Remove all the histogram bins features and replace by the Operational Time
x_train_m11<- x_train[, -c(7:16, 33:42, 43:52, 53:62, 100:109, 114:123, 159:168 )]
x_train_m11<- cbind(x_train_m11, Ttot = h2)

#-------------------------------------------------
## Perform the same on test set
agt_hist= x_test[,c(7:16)]
ayt_hist= x_test[,c(33:42)]
azt_hist= x_test[,c(43:52)]
bat_hist= x_test[,c(53:62)]
cnt_hist= x_test[,c(100:109)]
cst_hist= x_test[,c(114:123)]
eet_hist= x_test[,c(159:168)]
# Sum across bins
agt_cml = rowSums(agt_hist)
ayt_cml = rowSums(ayt_hist)
azt_cml = rowSums(azt_hist)
bat_cml = rowSums(bat_hist)
cnt_cml = rowSums(cnt_hist)
cst_cml = rowSums(cst_hist)
eet_cml = rowSums(eet_hist)

ht=as.data.frame(cbind(agt_cml, ayt_cml, azt_cml, bat_cml, cnt_cml, cst_cml, eet_cml))
ht1<- rowSds(as.matrix(ht))
ht2<- rowMeans(as.matrix(ht))

x_test_m11<- x_test[, -c(7:16, 33:42, 43:52, 53:62, 100:109, 114:123, 159:168 )]
x_test_m11<- cbind(x_test_m11, Ttot = ht2)
#-----------------------------------------------
```

From the distribution of the positive and negative classes over the total operating time, we observe that the APS tends to fail more in trucks with high operating time and consequently usage. (Since the timescale is not known, a logical assumption is made from the order of magnitude and transformed into DayHours-24)



II. Zero Variability Features

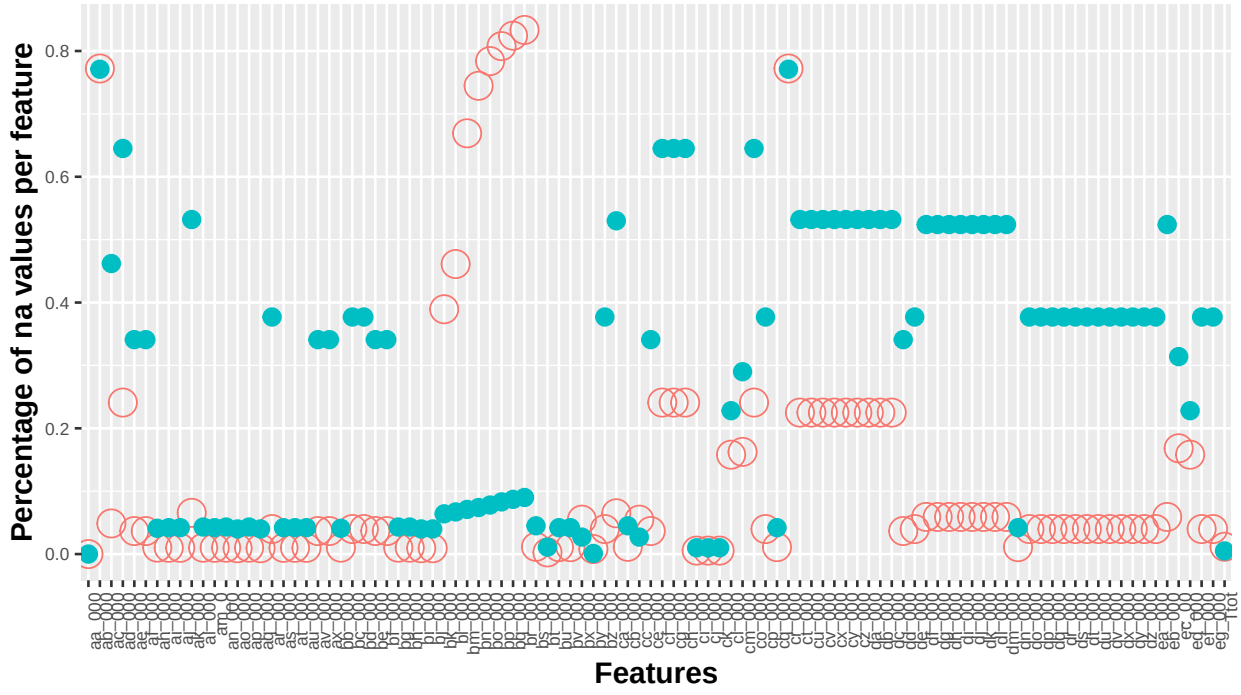
The standard deviation that is calculated for every feature, indicates that there is a feature with 0 variability. Since, this feature can not have any impact in the prediction, it is excluded from the dataset.

```
## Check for features with 0 variance
#-----------------------------------------------
clsds1<-colSds(as.matrix(x_train_m11), na.rm= TRUE)
clsds2<-colSds(as.matrix(x_train), na.rm= TRUE)

# Columns with 0 variance
#sum(clsds1==0)
#sum(clsds2==0)

# Remove features with 0 variance |Reduced
x_train_m12<-x_train_m11[!clsds1==0]
x_test_m12<-x_test_m11[!clsds1==0]

# Remove features with 0 variance |Extended
x_train_m22<-x_train[!clsds2==0]
x_test_m22<-x_test[!clsds2==0]
#-----------------------------------------------
```

III. Missing Values

Missing values in the dataset are denoted by *"na"*. The percentage of missing values in the dataset is **9.09**, which is not negligible.

By calculating the percentage of missing values for every feature, we observe that the percentage in some features is up to 80%. The percentage for every feature for the positive and negative class is presented below.



We can see that there are features with high percentage of missing values both for negative and positive classes. Those features will be excluded from the datasets, as it is considered that they can not be very helpful. On the contrary, features with high percentage of missing values in either classes will not be excluded as there is associated risk of losing important information.

```
## Exclude features with high percentage of nas both for pos and neg
#-------------------------------------------------------------------------
# Reduced
x_train_m13<-x_train_m12[!(k12$p_tr>0.75 & k12$n_tr>0.75)]
x_test_m13<-x_test_m12[!(k12$p_tr>0.75 & k12$n_tr>0.75)]

# Extended
x_train_m23<-x_train_m22[!(k22$p_tr>0.75 & k22$n_tr>0.75)]
x_test_m23<-x_test_m22[!(k22$p_tr>0.75 & k22$n_tr>0.75)]
#-------------------------------------------------------------------------
```

The remaining missing values in the dataset will be imputed with the median of every feature.

```
#-------------------------------------------------------------------------
# Function to fill na with column median
fna=function(x){x[is.na(x)]= median(x, na.rm= TRUE)
  x}
# Fill nas of Reduced Dataset
x_train_m14=data.frame(apply(x_train_m13,2,fna))
x_test_m14=data.frame(apply(x_test_m13,2,fna))

# Fill nas of Extended Dataset
x_train_m24=data.frame(apply(x_train_m23,2,fna))
x_test_m24=data.frame(apply(x_test_m23,2,fna))

# Confirm that there are no missing values
#sum(is.na(x_train_m14))
#sum(is.na(x_test_m14))

#sum(is.na(x_train_m24))
#sum(is.na(x_test_m24))
#-------------------------------------------------------------------------
```

IV. Scaling

Scaling of the datasets is performed in order to facilitate the analysis of the distribution of the features and their correlation.

```
#-------------------------------------------------------------------------
# Scale Reduced Dataset
x1i <- as.matrix(x_train_m14)
x1c <- sweep(x1i, 2, colMeans(x1i))
x1s <- sweep(x1c, 2, colSds(x1i), FUN = "/")
# Scale Extended Dataset
x2i <- as.matrix(x_train_m24)
x2c <- sweep(x2i, 2, colMeans(x2i))
x2s <- sweep(x2c, 2, colSds(x2i), FUN = "/")
# Scale Extended Test set -> required for simplified model
```

```
x2i_ts <- as.matrix(x_test_m24)
x2c_ts <- sweep(x2i_ts, 2, colMeans(x2i))
x2s_ts <- sweep(x2c_ts, 2, colSds(x2i), FUN = "/")
#-------------------------------------------------------------------------
```

V. Features Distribution

Indicative histograms of the first 25 features of the dataset are provided. Moreover, a table with descriptive statistics for the first 70 features is presented below.

Both the features' histograms and the descriptive statistics indicate that the features do not follow the normal distribution.

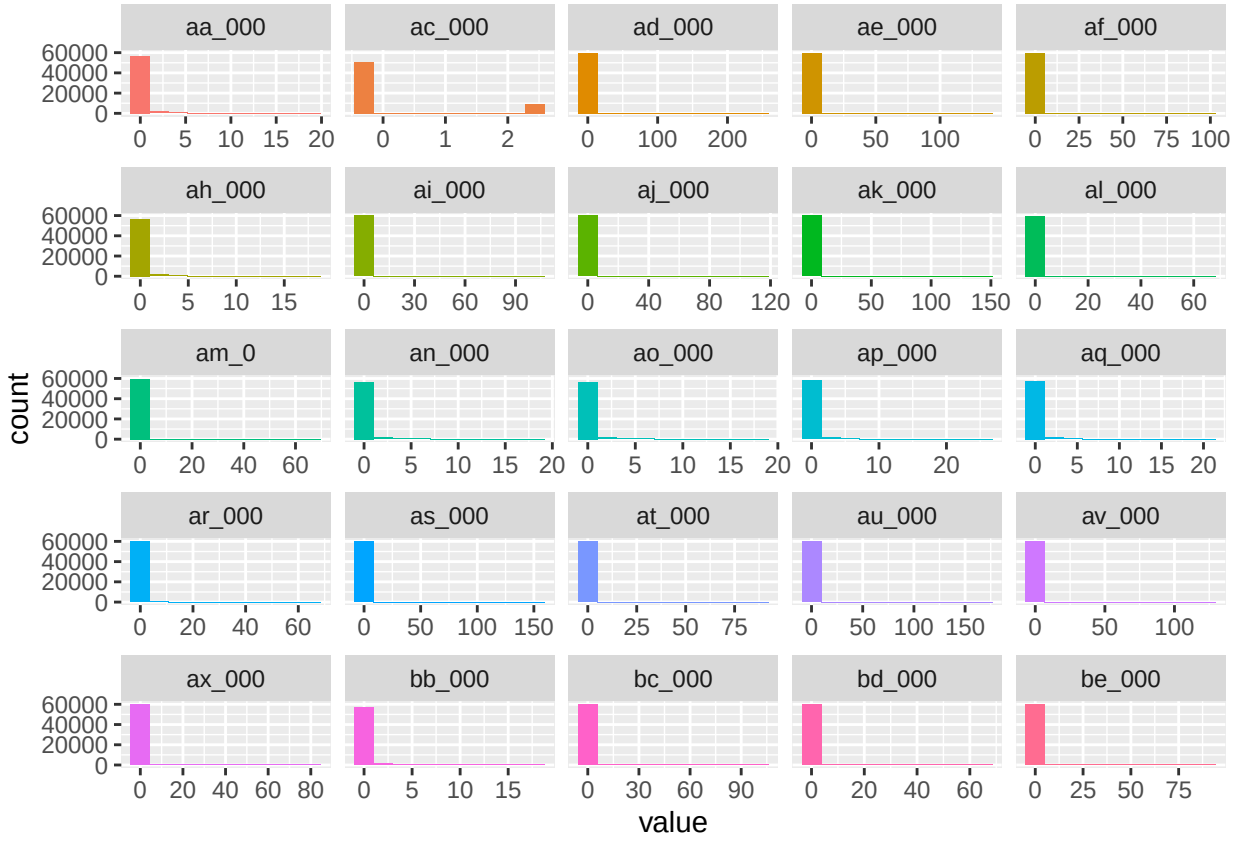


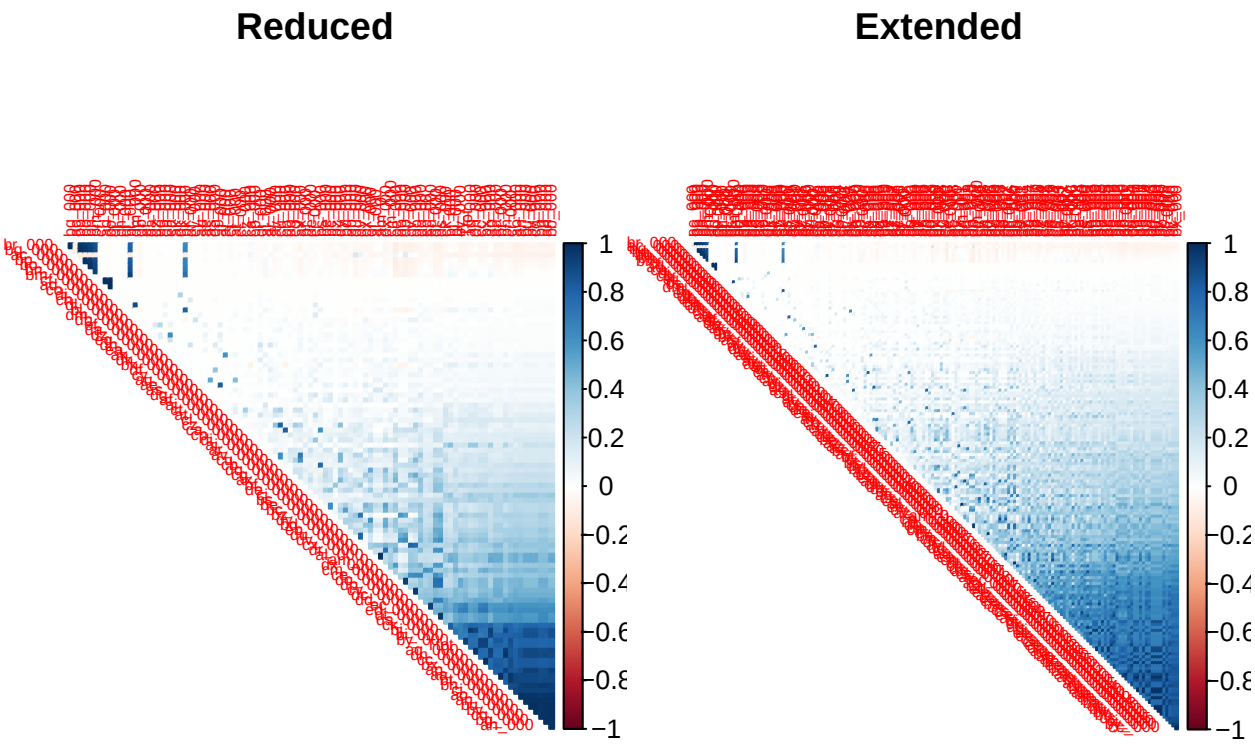
Table 2: Dataset Summary

aa_000	Min. :-0.41	1st Qu.:-0.40	Median :-0.20	Mean : 0.00	3rd Qu.:-0.07	Max. :18.48
ac_000	Min. :-0.433	1st Qu.:-0.433	Median :-0.433	Mean : 0.000	3rd Qu.:-0.433	Max. : 2.310
ad_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :245
ae_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :133
af_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :97.7
ah_000	Min. :-0.43	1st Qu.:-0.43	Median :-0.19	Mean : 0.00	3rd Qu.:-0.05	Max. :17.40
ai_000	Min. : -0.1	1st Qu.: -0.1	Median : -0.1	Mean : 0.0	3rd Qu.: -0.1	Max. :101.6
aj_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :112
ak_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :143
al_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :64.7
am_0	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :66.0
an_000	Min. :-0.44	1st Qu.:-0.43	Median :-0.20	Mean : 0.00	3rd Qu.:-0.04	Max. :17.73
ao_000	Min. :-0.44	1st Qu.:-0.43	Median :-0.20	Mean : 0.00	3rd Qu.:-0.05	Max. :17.56
ap_000	Min. :-0.32	1st Qu.:-0.32	Median :-0.21	Mean : 0.00	3rd Qu.:-0.09	Max. :25.04
aq_000	Min. :-0.35	1st Qu.:-0.35	Median :-0.21	Mean : 0.00	3rd Qu.:-0.05	Max. :19.99
ar_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :64.9
as_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :151
at_000	Min. : 0.0	1st Qu.: 0.0	Median : 0.0	Mean : 0.0	3rd Qu.: 0.0	Max. :87.4
au_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :167
av_000	Min. : -0.2	1st Qu.: -0.2	Median : -0.1	Mean : 0.0	3rd Qu.: -0.1	Max. :122.8
ax_000	Min. :-0.2	1st Qu.:-0.2	Median :-0.2	Mean : 0.0	3rd Qu.:-0.1	Max. :80.0
bb_000	Min. :-0.42	1st Qu.:-0.41	Median :-0.20	Mean : 0.00	3rd Qu.:-0.06	Max. :17.39
bc_000	Min. : -0.1	1st Qu.: -0.1	Median : -0.1	Mean : 0.0	3rd Qu.: -0.1	Max. :100.2
bd_000	Min. :-0.2	1st Qu.:-0.2	Median :-0.2	Mean : 0.0	3rd Qu.:-0.1	Max. :64.7
be_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :89.3
bf_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :95.2
bg_000	Min. :-0.43	1st Qu.:-0.43	Median :-0.19	Mean : 0.00	3rd Qu.:-0.05	Max. :17.42
bh_000	Min. :-0.38	1st Qu.:-0.37	Median :-0.21	Mean : 0.00	3rd Qu.:-0.06	Max. :20.76
bi_000	Min. :-0.33	1st Qu.:-0.32	Median :-0.21	Mean : 0.00	3rd Qu.:-0.08	Max. :30.07
bj_000	Min. :-0.28	1st Qu.:-0.27	Median :-0.19	Mean : 0.00	3rd Qu.:-0.10	Max. :24.97
bk_000	Min. :-1.22	1st Qu.:-0.30	Median :-0.21	Mean : 0.00	3rd Qu.:-0.10	Max. : 5.08
bl_000	Min. :-1.15	1st Qu.:-0.26	Median :-0.22	Mean : 0.00	3rd Qu.:-0.18	Max. : 4.29
bm_000	Min. :-1.18	1st Qu.:-0.22	Median :-0.22	Mean : 0.00	3rd Qu.:-0.22	Max. : 4.08
bn_000	Min. :-1.20	1st Qu.:-0.22	Median :-0.22	Mean : 0.00	3rd Qu.:-0.22	Max. : 3.89
bo_000	Min. :-1.26	1st Qu.:-0.21	Median :-0.21	Mean : 0.00	3rd Qu.:-0.21	Max. : 3.81
bp_000	Min. :-1.33	1st Qu.:-0.21	Median :-0.21	Mean : 0.00	3rd Qu.:-0.21	Max. : 3.76
bq_000	Min. :-1.39	1st Qu.:-0.20	Median :-0.20	Mean : 0.00	3rd Qu.:-0.20	Max. : 3.71
br_000	Min. :-1.45	1st Qu.:-0.20	Median :-0.20	Mean : 0.00	3rd Qu.:-0.20	Max. : 3.67
bs_000	Min. :-0.95	1st Qu.:-0.74	Median :-0.35	Mean : 0.00	3rd Qu.: 0.45	Max. :11.39
bt_000	Min. :-0.41	1st Qu.:-0.40	Median :-0.20	Mean : 0.00	3rd Qu.:-0.07	Max. :18.50
bu_000	Min. :-0.42	1st Qu.:-0.41	Median :-0.20	Mean : 0.00	3rd Qu.:-0.06	Max. :17.44
bv_000	Min. :-0.42	1st Qu.:-0.41	Median :-0.20	Mean : 0.00	3rd Qu.:-0.06	Max. :17.44
bx_000	Min. :-0.40	1st Qu.:-0.39	Median :-0.17	Mean : 0.00	3rd Qu.:-0.05	Max. :18.10
by_000	Min. :-0.41	1st Qu.:-0.40	Median :-0.17	Mean : 0.00	3rd Qu.:-0.03	Max. :18.22
bz_000	Min. :-0.2	1st Qu.:-0.2	Median :-0.2	Mean : 0.0	3rd Qu.:-0.1	Max. :65.8
ca_000	Min. :-1.073	1st Qu.:-0.851	Median :-0.358	Mean : 0.000	3rd Qu.: 0.731	Max. : 2.327
cb_000	Min. :-1.100	1st Qu.:-0.886	Median :-0.341	Mean : 0.000	3rd Qu.: 0.803	Max. : 2.192
cc_000	Min. :-0.40	1st Qu.:-0.39	Median :-0.17	Mean : 0.00	3rd Qu.:-0.05	Max. :15.47
ce_000	Min. :-0.4	1st Qu.:-0.4	Median :-0.4	Mean : 0.0	3rd Qu.: 0.2	Max. :34.5
cf_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :245
cg_000	Min. :-0.2	1st Qu.:-0.2	Median :-0.1	Mean : 0.0	3rd Qu.: 0.0	Max. :66.0
ch_000	Min. : 0.0	1st Qu.: 0.0	Median : 0.0	Mean : 0.0	3rd Qu.: 0.0	Max. :77.5
ci_000	Min. :-0.42	1st Qu.:-0.41	Median :-0.19	Mean : 0.00	3rd Qu.:-0.06	Max. :16.50
cj_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :53.8
ck_000	Min. :-0.33	1st Qu.:-0.32	Median :-0.21	Mean : 0.00	3rd Qu.:-0.08	Max. :25.16
cl_000	Min. :-0.07	1st Qu.:-0.07	Median :-0.07	Mean : 0.00	3rd Qu.:-0.07	Max. :30.17
cm_000	Min. :-0.2	1st Qu.:-0.2	Median :-0.2	Mean : 0.0	3rd Qu.:-0.2	Max. :47.1
co_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :245
cp_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :65.3
cq_000	Min. :-0.42	1st Qu.:-0.41	Median :-0.20	Mean : 0.00	3rd Qu.:-0.06	Max. :17.44
ct_000	Min. : -0.1	1st Qu.: -0.1	Median : -0.1	Mean : 0.0	3rd Qu.: 0.0	Max. :185.4
cu_000	Min. : -0.1	1st Qu.: -0.1	Median : -0.1	Mean : 0.0	3rd Qu.: -0.1	Max. :109.2
cv_000	Min. :-0.54	1st Qu.:-0.52	Median :-0.18	Mean : 0.00	3rd Qu.: 0.13	Max. :24.74
cx_000	Min. :-0.2	1st Qu.:-0.2	Median :-0.2	Mean : 0.0	3rd Qu.:-0.1	Max. :32.9
cy_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :120
cz_000	Min. :-0.1	1st Qu.:-0.1	Median :-0.1	Mean : 0.0	3rd Qu.:-0.1	Max. :88.2
da_000	Min. : 0	1st Qu.: 0	Median : 0	Mean : 0	3rd Qu.: 0	Max. :126
db_000	Min. : -0.2	1st Qu.: -0.2	Median : -0.2	Mean : 0.0	3rd Qu.: 0.0	Max. :148.9
dc_000	Min. :-0.6	1st Qu.:-0.6	Median :-0.1	Mean : 0.0	3rd Qu.: 0.1	Max. :32.9
dd_000	Min. :-0.3	1st Qu.:-0.3	Median :-0.2	Mean : 0.0	3rd Qu.: 0.0	Max. :47.4

VI. Features Correlation

In order to investigate the dependence between the features of the datasets, the correlation matrix is calculated. The respective correlation plots for the Reduced and Extended datasets are provided below.

```
#-------------------------------------------------------
# Correlation matrix
train_cor1 <- cor(x1s)
train_cor2 <- cor(x2s)

# Find features with high correlation >0.9
hcor1 <- findCorrelation(train_cor1, cutoff=0.9,names = TRUE)
hcor2 <- findCorrelation(train_cor2, cutoff=0.9,names = TRUE)
#-------------------------------------------------------
```



The plots demonstrate several highly correlated features. Having highly correlated features is generally not desirable as it creates Multicollinearity. Multicollinearity can have a severe impact on the performance of machine learning models and can lead to skewed or misleading results. For example logistic regression and random forest models can become very unstable in the presence of high feature correlations.

We will not proceed to manually excluding highly correlated features. To effectively deal with this problem, Principle Component Analysis will be applied.

VII. Principal Component Analysis PCA

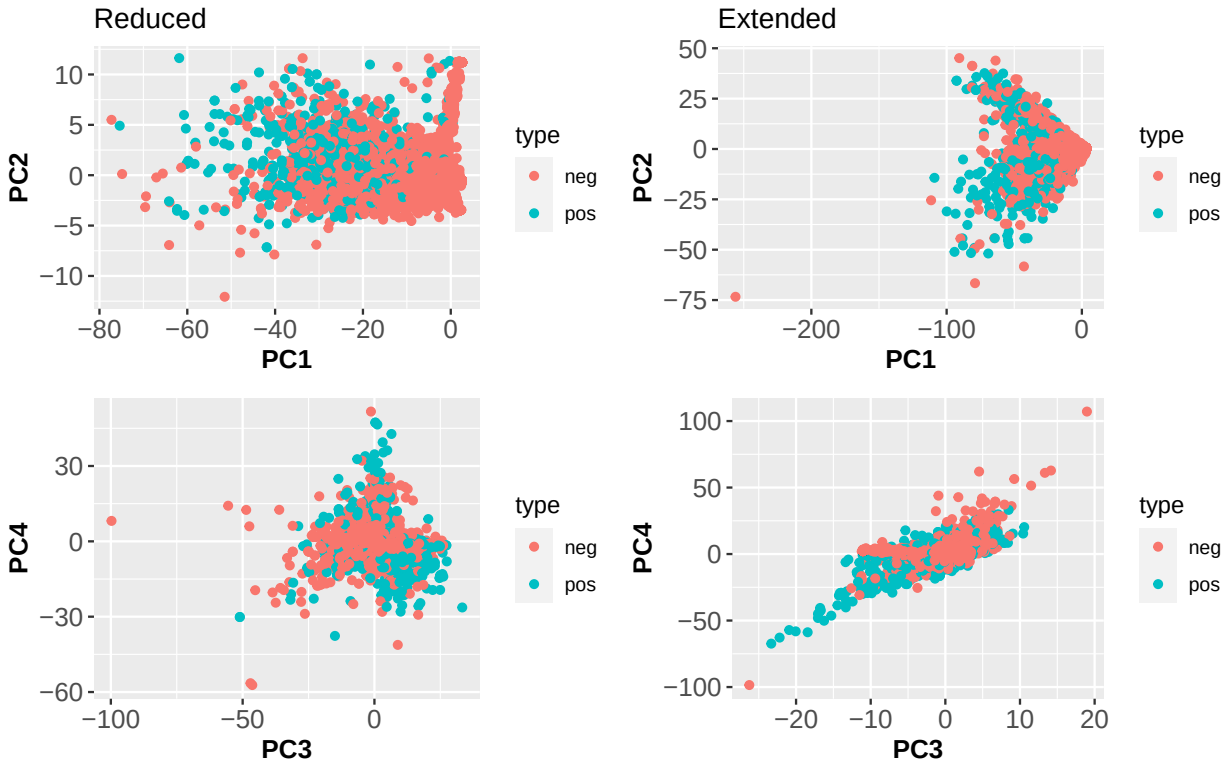
As mentioned, in order to effectively deal with the high correlation among features but furthermore and most importantly to optimize the amount of features to be included in the training set, the process of Principal Component Analysis (PCA) will be applied.

PCA produces components, which consist of a normalized linear combination of the original features and aim to capture as much information as possible with high explained variance. What is extremely significant, is that their variability is decreasing and they are not correlated to each other. This practically means that the first component captures the maximum variance in the dataset, is not correlated to the second which captures less variance and so on. The algorithm also computes these variabilities so that we can know how much of the dataset's total variability is explained as we add components. This allows us to select a reduced amount of components-features that captures the desired variability of the dataset.

PCA is always performed in a matrix with numeric and standardized data. Performing PCA on un-normalized variables will lead to dependence of a principal component on the variable with high variance, which is undesirable. For that reason, we enable the *scale* and *center* options.

```
#-----------------------------------------------------
# PCA on datasets
pca_train1 <- prcomp(x_train_m14, scale. = TRUE, center = TRUE)
pca_train2 <- prcomp(x_train_m24, scale. = TRUE, center = TRUE)
# Check cumulative proportion of variance explained
#summary(pca_train1)
#summary(pca_train2)
#-----------------------------------------------------
```

The first 4 principal components which capture a significant amount of the dataset's variance are presented below for the Reduced and the Extended dataset. The plots demonstrate that the positive and negative classes are overlapping, which indicates that the failure prediction is not so evident.

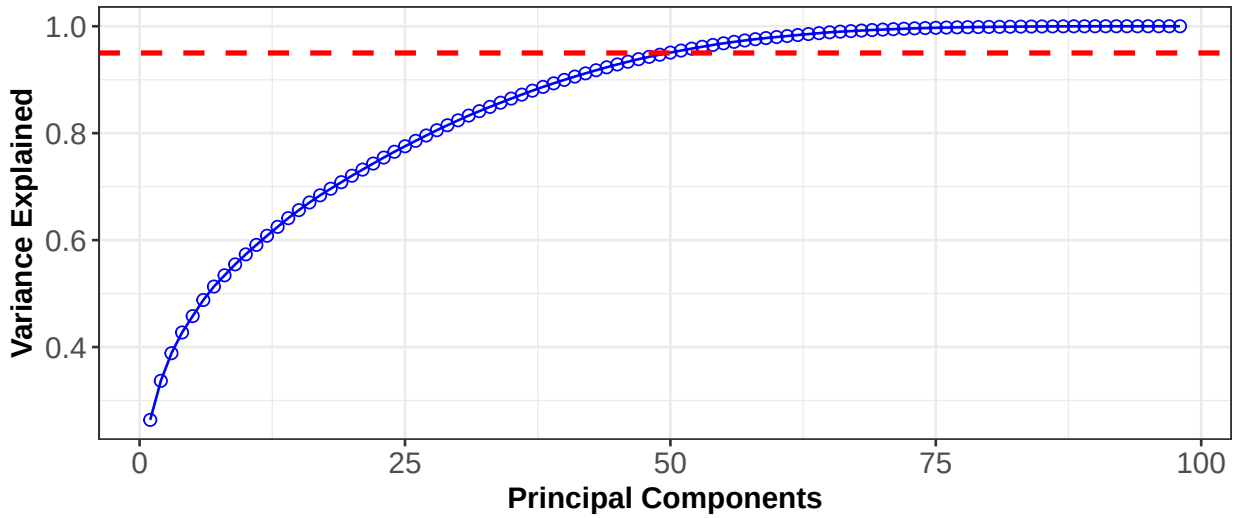


As mentioned, PCA is very useful in optimizing the amount of features to be included in the training set. To achieve this, a desired percentage of explained variance is defined and the number of principal components that satisfy this criteria is selected. The cumulative variance explained for the principal components is calculated below:

```
#-----------------------------------------------------
# Calculate Variance Explained |Reduced
cml_var_explained1 <- cumsum(pca_train1$sdev^2 / sum(pca_train1$sdev^2))
cml_var_explained1 <- data.frame(c(1:length(pca_train1$sdev)),cml_var_explained1)
names(cml_var_explained1)[1] = 'PCs'
names(cml_var_explained1)[2] = 'Percentage'
#-----------------------------------------------------
# Calculate features required for 95% Var |Reduced
features_needed_Red<-min(which(cml_var_explained1>0.95))
#-----------------------------------------------------
# For simplicity not calculated for Extended
# -> obvious from summary of PCA
#-----------------------------------------------------
```

The desired explained variance is set to 95%. From the calculated cumulative variance explained for the *Reduced* dataset, we can see that *1* principal components are required to satisfy the desired criteria. By performing the same process for the *Extended* dataset, 80 principal components are required respectively to capture 95% of the dataset's variance.

An indicative plot for the cumulative proportional variance explained per component, for the Reduced dataset is presented below:



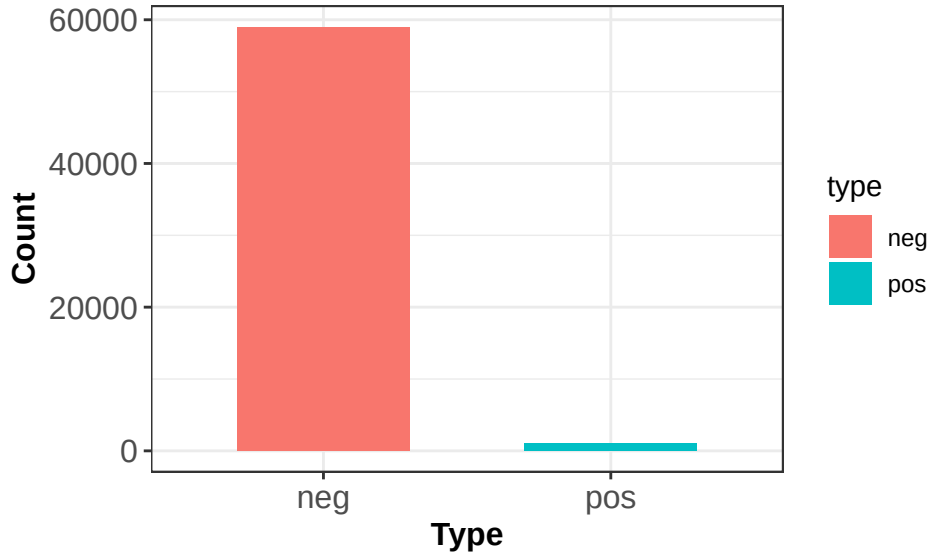
The *Reduced* dataset with the 50 first principal components and the *Extended* with the 80 first principal components respectively are generated with the code below. The same transformations are also performed for the test datasets.

```
#-----------------------------------------------------
# Select first 50 PCAS for 95% var |Reduced
pca_train_m1<- as.data.frame(pca_train1$x)
pca_train_m1<- pca_train_m1[,c(1:50)]

# Select first 80 PCAS for 95% var |Extended
```

```r
pca_train_m2<- as.data.frame(pca_train2$x)
pca_train_m2<- pca_train_m2[,c(1:80)]
#-----------------------------------------------------

## PCA transformation on test sets
#-----------------------------------------------------
pca_test1<- predict(pca_train1, x_test_m14)
pca_test2<- predict(pca_train2, x_test_m24)

pca_test_m1<- as.data.frame(pca_test1)
pca_test_m1<- pca_test_m1[,c(1:50)]

pca_test_m2<- as.data.frame(pca_test2)
pca_test_m2<- pca_test_m2[,c(1:80)]
#-----------------------------------------------------
```

VIII. Data Class Balancing

By plotting the proportion of the classes in the training datasets, we observe that the negative class, significantly outnumbers the positive class (failure events). The negative class represents the **0.983%** of the training dataset, while the positive class only the **0.017%**.



Imbalancing in the dataset can be observed in several real life cases. For example, a manufacturing operating under six sigma principle may encounter 10 in a million defected products. However, in terms of training machine learning algorithms it is highly undesirable as they result in biased predictions and misleading accuracies.

Machine learning algorithms are accuracy driven and aim to minimize the overall error, to which the minority class contributes very little in an imbalanced dataset. This causes the performance of existing classifiers to get biased towards majority class. Moreover, they assume that errors obtained from different classes have the same cost, which does not apply in the examined case.

The problem of imbalancing can be solved by applying mechanisms that modify the original dataset and provide a balanced distribution of classes. Well known methods for treating imbalanced datasets are *Undersampling*, *Oversampling*, *Synthetic Data Generation* and *Cost Sensitive Learning*.

In this case *Synthetic Minority Oversampling Technique* (SMOTE) will be used which is a synthetic data generation method. SMOTE is a powerful method that overcomes imbalance by generating artificial data. It down samples the majority class and generates a random set of minority class observations based on feature space similarities from minority samples.

SMOTE is applied on the Reduced and Extended datasets respectively with the following code:

```
## Balancing SMOTE
#-----------------------------------------
# SMOTE balancing train |REduced
set.seed(1989, sample.kind = "Rounding")
pca_train_m1bal <- as.data.frame(cbind(pca_train_m1, Type=y_train))
pca_train_m1bal = SMOTE(Type~., pca_train_m1bal, perc.over = 1900, perc.under = 157.895, k=5)

# SMOTE balancing train |Extended
set.seed(1989, sample.kind = "Rounding")
pca_train_m2bal <- as.data.frame(cbind(pca_train_m2, Type=y_train))
pca_train_m2bal = SMOTE(Type~., pca_train_m2bal, perc.over = 1900, perc.under = 157.895, k=5)
#-----------------------------------------
```

Having performed all the necessary modifications, the final datasets to be used in the training and testing of the machine learning algorithms are generated below:

```
## Final Training-Test sets
#-----------------------------------------
# Reduced
y_train_f1<-as.factor(pca_train_m1bal$Type)
x_train_f1<-as.data.frame(pca_train_m1bal)%>%select(-Type)
x_test_f1<- pca_test_m1

# Extended
y_train_f2<-as.factor(pca_train_m2bal$Type)
x_train_f2<-as.data.frame(pca_train_m2bal)%>%select(-Type)
x_test_f2<- pca_test_m2
#-----------------------------------------
```

4. Modeling Approach

The challenge of this project is to minimize the maintenance costs by effectively predicting failure events related to the APS. It is a classification problem for cost minimization. In this chapter the models that were developed in order to effectively predict the failure events will be presented, starting from simplified methods and extending to more advanced methods, in terms of computational complexity and tuning parameters that need to be optimized. The machine learning models that are applied are based on the insights of the Machine Learning Course of the respective Data Science Program and on dedicated literature review that was performed regarding binary classification.

Initially a simplified method will be developed that is based on an engineering perspective that missing values are associated with a failure in a sensor or a system malfunction. The performance of this method will be used as reference for the machine learning methods Afterwards, kmeans-clustering, a simplified machine learning method will be used. Then, logistic regression and boosted logistic regression will be applied. Extending to more advanced methods, Random Forest and Extreme Gradient Boosting will be applied and their tuning parameters will be optimized. Finally, an ensemble method will be presented.

Most classification algorithms calculate accuracy based on the percentage of observations correctly classified. However, for the examined case, choosing a performance metric is a critical aspect, because according to the instructions there are different costs for miss-classification. The cost of falsely predicting a failure, which is associated with an unnecessary check by a mechanic at a workshop, is 10, while the the cost of missing a faulty truck, which may cause a breakdown, is 500. Since the cost of a false negative is 50 times higher from a false positive, Specificity will be used as performance metric to optimize the models. (It should be noted that, the way classes are handled in the developed models, the ability to predict a positive outcome is defined by specificity, while in theory it is specified by sensitivity.)

As mentioned, in the exploratory analysis chapter, 2 separate training datasets have been developed. The *Reduced* has been generated by removing all histogram bins features while an equivalent feature for the total operating time has been added and consists of 50 principal components that capture 95% of the variance. The *Extended* corresponds to all features available in the raw dataset and consists of 80 principal components that capture 95% of the variance as well. Logistic Regression, Boosted Logistic Regression, Random Forest and Extreme Gradient Boosting will be applied on both training datasets and their performance will be compared, in order to assess the impact of the dataset on the prediction quality.

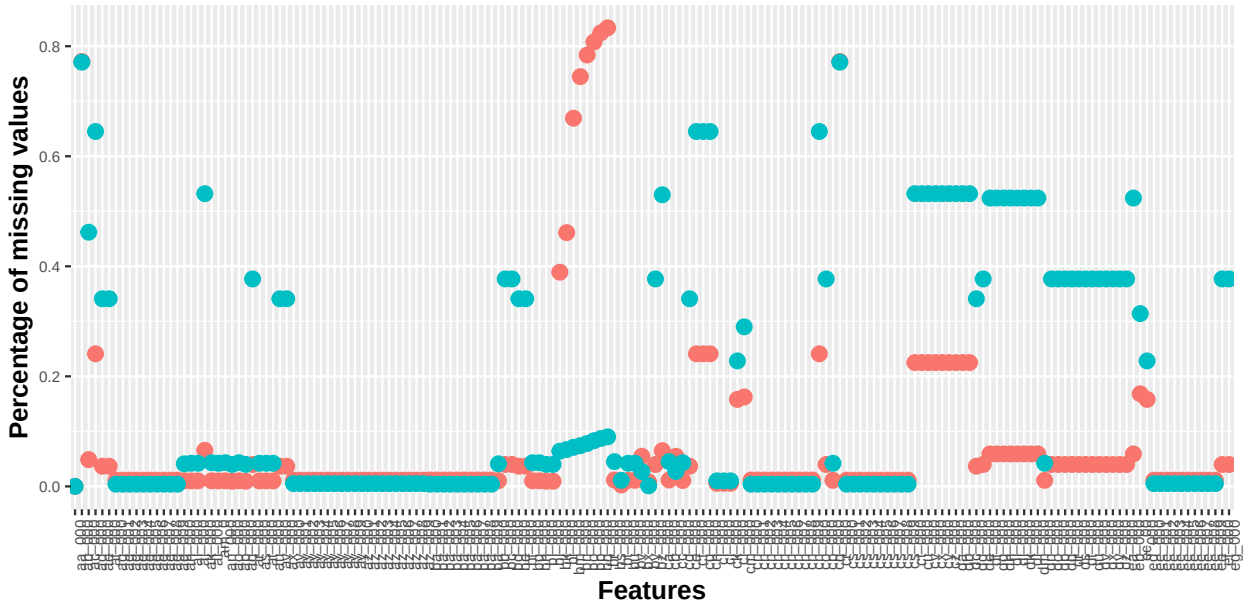
The training datasets will be used to train the machine learning models and their tuning parameters will be optimized by means of cross-validation. All models will be applied independently on the test set to evaluate their performance.

I. Simplified Engineering Method

As mentioned in the exploratory analysis part, there are many missing values in the dataset. In terms of data analysis, missing values do not contain any useful information and are imputed in order to facilitate the analysis and data processing methods. However, from an engineering perspective, missing values convey important information as they are related to failure. In the examined case, many features correspond to measurements from on-board sensors. A missing value is often an outcome of either a faulty sensor, which can lead to a system malfunction, or a failure in a system which results in an error in the controllers communication line (CAN-bus).

Based on this consideration, a simplified engineering approach to predict failure events from the missing values will be performed. The raw dataset without any modifications will be used for this approach.

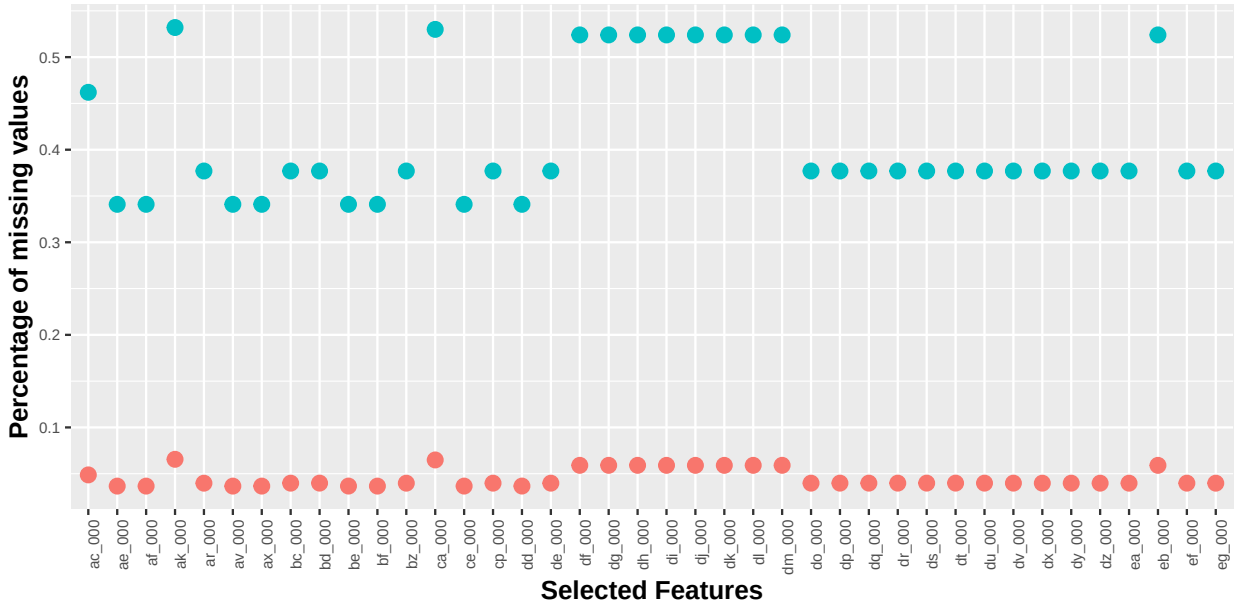
```
#---------------------------------------------
# na per feature |Negative Class
n_train<-x_train[y_train=="neg",]
nacl_n_train <-colSums(is.na(n_train))

# na per feature |Positive Class
p_train<-x_train[y_train=="pos",]
nacl_p_train <-colSums(is.na(p_train))

# data.frame with combined pos and neg na info
nacl <-data.frame(features, p_train=nacl_p_train/dim(p_train)[1],
                  n_train = nacl_n_train/dim(n_train)[1])
#---------------------------------------------
```



By plotting the percentage of missing values per feature for positive and negative class we observe that most of the features have low percentage but there are also features with high percentage in either of the classes. Based on the plot, we will create a rule to select the features that can help us distinguish the positive from the negative class. In this respect, we will select features that have a low percentage of missing values in the negative class and on the same time, a high percentage on the positive class, which would indicate failure.

```
#-----------------------------------------------
# Select features with high na perc in pos class
nacl <-nacl[nacl$p_train>0.3,]
# Keep features with low na perc in neg class
nacl <-nacl[nacl$n_train<0.1,]
#-----------------------------------------------
```

The selected features that follow the rule are presented below:



The selected features will be used in order to determine if we have a failure event. With this simplified approach we consider that a missing value in any of the selected features is an indicator of a failure event.

```
#-----------------------------------------------
# predict failure if there is na in selected features
index<-features%in%nacl$features
x_test_na<-x_test[,index]
train_na<-rowSums(is.na(x_test_na))
pred_na<-ifelse(train_na>0,"pos","neg")
pred_na<-factor(pred_na, levels= levels(y_test))
cm_na <- confusionMatrix(pred_na, y_test)
#-----------------------------------------------

# Store results for Simplified NA method
results<-tibble(Method="Simplified NA method",
                Dataset = "Raw",
                Specificity = cm_na$byClass["Specificity"],
                Sensitivity = cm_na$byClass["Sensitivity"],
                FN = cm_na$table[1,2],
                FP = cm_na$table[2,1],
                Cost = FN*500 +FP*10)
#-----------------------------------------------
```

The results for the simplified method based on the missing values are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590

Despite the simplicity of the method and the assumptions that is based on, the method yields a satisfying accuracy and subsequent cost. This method represents the performance that can be achieved without using machine learning and will be used as reference point for the machine learning methods that will be applied. However, this method has considerable drawbacks, as it is not predictive and can not provide any input on the importance of the features in identifying failure events.

II. k-means Clustering

k-means clustering is the first machine learning method that will be applied. It is a simplified method that uses the features and the distances of the observations to define clusters. The number of centers-clusters needs to be defined and then each observation is assigned to the cluster with the closest center to that observation.

For the examined case, 2 clusters are required, for the negative and positive class respectively. The scaled matrix of the extended dataset will be used to define the centers of the clusters. Afterwards, a function is defined that uses the calculated centers of the kmeans clustering process to assign observations to a cluster. The function will be applied on the test set to make a prediction.

```
#-----------------------------------------------------------------------
# Perform k-means clustering on scaled train set with 2 clusters
set.seed(1989, sample.kind = "Rounding")
k <- kmeans(x2s, centers = 2)
#-----------------------------------------------------------------------
# The defined predict_kmeans() function takes two arguments
# 1.a matrix of observations x
# 2.a k-means object k
# Assigns each row of x to a cluster from k ->Prediction
predict_kmeans <- function(x, k) {
  centers <- k$centers    # extract cluster centers
  # calculate distance to cluster centers
  distances <- sapply(1:nrow(x), function(i){
  apply(centers, 1, function(y) dist(rbind(x[i,], y)))
  })
  max.col(-t(distances))  # select cluster with min distance to center
}
#-----------------------------------------------------------------------
# Use the predict_kmeans() function to make predictions on the test set
pred_kmeans <- ifelse(predict_kmeans(x2s_ts, k) == 2, "neg", "pos")
pred_kmeans <- factor(pred_kmeans, levels= levels(y_test))
cm_kmeans <- confusionMatrix(pred_kmeans, y_test)
#-----------------------------------------------------------------------

# Store results for kmeans clustering method
results<-bind_rows(results,tibble(Method="k-means Clustering",
                                  Dataset = "Scaled",
                                  Specificity = cm_kmeans$byClass["Specificity"],
                                  Sensitivity = cm_kmeans$byClass["Sensitivity"],
                                  FN = cm_kmeans$table[1,2],
                                  FP = cm_kmeans$table[2,1],
```

```
                                    Cost = FN*500 +FP*10))
#-------------------------------------------------------------------------
```

The results for the k-means clustering method are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580

The results demonstrate that a significant improvement is achieved already by applying a simplified machine learning algorithm.

III. Logistic Regression

Logistic regression is a basic machine learning method used for binary classification problems, like the failure prediction. Based on the field of statistics, logistic regression is an extension of linear regression that uses the logistic function, which can take any real-valued number and map it into a value between 0 and 1. The coefficients for the regression are estimated using the maximum-likelihood estimation and a prediction is made based on the estimated probability for a default class.

The logistic regression is applied both for the *Reduced* training dataset, as well as the *Extended*.

```
## Logistic Regression Reduced
#-------------------------------------------------------------------------
set.seed(1989, sample.kind = "Rounding")
train_glm1 <- train(x_train_f1, y_train_f1,
                    method = "glm",
                    family = "binomial")


pred_glm1 <- predict(train_glm1, x_test_f1)
cm_glm1 <- confusionMatrix(pred_glm1, y_test)
#-------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method="Logistic Regression",
                                  Dataset = "Reduced",
                                  Specificity = cm_glm1$byClass["Specificity"],
                                  Sensitivity = cm_glm1$byClass["Sensitivity"],
                                  FN = cm_glm1$table[1,2],
                                  FP = cm_glm1$table[2,1],
                                  Cost = FN*500 +FP*10))
#-------------------------------------------------------------------------
```

```
## Logistic Regression Extended
#-------------------------------------------------------------------------
set.seed(1989, sample.kind = "Rounding")
train_glm2 <- train(x_train_f2, y_train_f2,
                    method = "glm",
                    family = "binomial")
```

```
pred_glm2 <- predict(train_glm2, x_test_f2)
cm_glm2 <- confusionMatrix(pred_glm2, y_test)
#------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method="Logistic Regression",
                                  Dataset = "Extended",
                                  Specificity = cm_glm2$byClass["Specificity"],
                                  Sensitivity = cm_glm2$byClass["Sensitivity"],
                                  FN = cm_glm2$table[1,2],
                                  FP = cm_glm2$table[2,1],
                                  Cost = FN*500 +FP*10))
#------------------------------------------------------------------------
```

The results for the Logistic Regression for both training datasets are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580
Logistic Regression	Reduced	0.928	0.972	27	444	17940
Logistic Regression	Extended	0.963	0.837	14	2550	32500

The results demonstrate that by applying logistic regression, the specificity significantly improves and leads to a subsequent decrease in the cost. We can see that a better overall cost is achieved by training the logistic regression model with the Reduced dataset. Although the Extended dataset achieves very high specificity and effectively predicts the failure events, it also predicts many false failure events(FP) which leads to an increased cost. This can probably be attributed to noise because of the extended number of features used.

IV. Boosted Logistic Regression

Boosting is a development introduced in classification methodology that often leads to significant improvement in the performance of classification algorithms. Boosted logistic regression uses a generalized additive model and then applies the cost function of logistic regression. The algorithm relies on a voting scheme to make classifications. Many week classifiers (nIter) are applied to each sample and their findings are used as votes to make the final classification

The *nIter* tuning parameter of the algorithm will be optimized using 6-fold cross-validation. Only odd numbers will be used for the nIter in order to avoid a possible "voting tie", which would produce an NA result. Parallel processing is also activated to reduce the computational time.

Two Class Summary is selected as summary function in order to be able to use *Specificity* as performance metric for the model optimization. Choosing *specificity* allows as to focus more on the positive events (failure) prediction which come with a higher cost.

The Boosted logistic regression is applied both for the *Reduced* training dataset, as well as the *Extended*.

```
## Boosted Logistic Regression Reduced
#------------------------------------------------------------------------
# Parallel Processing settings
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
#getDoParWorkers()
#------------------------------------------------------------------------
```

```r
set.seed(1989, sample.kind = "Rounding")

# Initial Tuning Grid for optimization
#tuning_lgb1 <- data.frame(nIter = c(15, 21, 25 ))

# Optimized parameter for Specificity metric
tuning_lgb1 <- data.frame(nIter = c( 25 ))

control_lgb1 <- trainControl(method= "cv", number = 6, p= 0.8,
                             #verboseIter = TRUE,
                             classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             allowParallel = TRUE)
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

train_lgb1 <- train(x_train_f1, y_train_f1,
                    method = "LogitBoost",
                    trControl = control_lgb1,
                    tuneGrid = tuning_lgb1,
                    metric = "Spec",
                    allowParallel = TRUE)
                    #verbose = TRUE)

stopCluster(cl)
registerDoSEQ()

pred_lgb1<- predict(train_lgb1, x_test_f1)
cm_lgb1<- confusionMatrix(pred_lgb1, y_test)
#------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method=" Boosted Logistic Regression",
                                  Dataset = "Reduced",
                                  Specificity = cm_lgb1$byClass["Specificity"],
                                  Sensitivity = cm_lgb1$byClass["Sensitivity"],
                                  FN = cm_lgb1$table[1,2],
                                  FP = cm_lgb1$table[2,1],
                                  Cost = FN*500 +FP*10))
#------------------------------------------------------------------------
```

```r
## Boosted Logistic Regression Extended
#------------------------------------------------------------------------
# Parallel Processing settings
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
#getDoParWorkers()
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

# Initial Tuning Grid for optimization
#tuning_lgb2 <- data.frame(nIter = c(15, 21, 25 ))
```

```
# Optimized parameter for Specificity metric
tuning_lgb2 <- data.frame(nIter = c( 25 ))

control_lgb2 <- trainControl(method= "cv", number = 6, p= 0.8,
                             #verboseIter = TRUE,
                             classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             allowParallel = TRUE)
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

train_lgb2 <- train(x_train_f2, y_train_f2,
                    method = "LogitBoost",
                    trControl = control_lgb2,
                    tuneGrid = tuning_lgb2,
                    metric = "Spec",
                    allowParallel = TRUE)
                    #verbose = TRUE)

stopCluster(cl)
registerDoSEQ()

pred_lgb2 <- predict(train_lgb2, x_test_f2)
cm_lgb2 <- confusionMatrix(pred_lgb2, y_test)
#------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method=" Boosted Logistic Regression",
                                  Dataset = "Extended",
                                  Specificity = cm_lgb2$byClass["Specificity"],
                                  Sensitivity = cm_lgb2$byClass["Sensitivity"],
                                  FN = cm_lgb2$table[1,2],
                                  FP = cm_lgb2$table[2,1],
                                  Cost = FN*500 +FP*10))
#------------------------------------------------------------------------
```

The results for the Boosted Logistic Regression for both training datasets are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580
Logistic Regression	Reduced	0.928	0.972	27	444	17940
Logistic Regression	Extended	0.963	0.837	14	2550	32500
Boosted Logistic Regression	Reduced	0.928	0.948	27	816	21660
Boosted Logistic Regression	Extended	0.941	0.957	22	676	17760

By applying Boosted Logistic Regression, we observe the opposite trend compared to Logistic Regression. The cost of the Extended training dataset is lower than the Reduced dataset. The specificity for the Extended dataset reduced but the sensitivity significantly improved. On the contrary, the sensitivity for the Reduced dataset decreased

V. Random Forest

Random Forest is a machine learning algorithm that builds multiple decision trees and merges them together to get a more accurate and stable prediction, using the *bagging* method. The general idea of the bagging method is that a combination of learning models increases the overall result. Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity and less overfitting, that generally results in a better model.

The *mtry* tuning parameter of the algorithm will be optimized using 6-fold cross-validation. This tuning parameter represents the number of random features available for splitting at each tree node. Parallel processing is also activated to reduce the computational time. Two Class Summary is selected as summary function in order to be able to use *Specificity* as performance metric for the model optimization.

The Random Forest algorithm is applied both for the *Reduced* training dataset, as well as the *Extended*.

```
## Random Forest Reduced
#------------------------------------------------------------------------
# Parallel Processing settings
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
#getDoParWorkers()
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

control_rf1<- trainControl(method= "cv", number = 6, p= 0.8,
                           #verboseIter = TRUE,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary,
                           allowParallel = TRUE)

# Initial Tuning Grid for optimization
#tuning_rf1 <- data.frame(mtry = c( 5, 7, 9))

# Optimized parameter for Specificity metric
tuning_rf1 <- data.frame(mtry = 9)
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

train_rf1 <- train(x_train_f1, y_train_f1,
                   method = "rf",
                   trControl = control_rf1,
                   tuneGrid = tuning_rf1,
                   metric = "Spec",
                   importance = TRUE,
                   allowParallel = TRUE)
                   #verbose = TRUE)
# default ntrees=500
# default p=0.75

stopCluster(cl)
registerDoSEQ()
```

```r
pred_rf1<- predict(train_rf1, x_test_f1)
cm_rf1 <- confusionMatrix(pred_rf1, y_test)
#------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method="Random Forest",
                                  Dataset = "Reduced",
                                  Specificity = cm_rf1$byClass["Specificity"],
                                  Sensitivity = cm_rf1$byClass["Sensitivity"],
                                  FN = cm_rf1$table[1,2],
                                  FP = cm_rf1$table[2,1],
                                  Cost = FN*500 +FP*10))
#------------------------------------------------------------------------
```

```r
## Random Forest Extended
#------------------------------------------------------------------------
# Parallel Processing settings
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
#getDoParWorkers()
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

control_rf2<- trainControl(method= "cv", number = 6, p= 0.8,
                           #verboseIter = TRUE,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary,
                           allowParallel = TRUE)

# Initial Tuning Grid for optimization
#tuning_rf2 <- data.frame(mtry = c( 5, 7, 9))

# Optimized parameter for Specificity metric
tuning_rf2 <- data.frame(mtry = 9)
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

train_rf2 <- train(x_train_f2, y_train_f2,
                   method = "rf",
                   trControl = control_rf2,
                   tuneGrid = tuning_rf2,
                   metric = "Spec",
                   importance = TRUE,
                   allowParallel = TRUE)
                   #verbose = TRUE)

stopCluster(cl)
registerDoSEQ()

pred_rf2<- predict(train_rf2, x_test_f2)
cm_rf2 <- confusionMatrix(pred_rf2, y_test)
#------------------------------------------------------------------------
```

```
results<-bind_rows(results,tibble(Method="Random Forest",
                                  Dataset = "Extended",
                                  Specificity = cm_rf2$byClass["Specificity"],
                                  Sensitivity = cm_rf2$byClass["Sensitivity"],
                                  FN = cm_rf2$table[1,2],
                                  FP = cm_rf2$table[2,1],
                                  Cost = FN*500 +FP*10))
#------------------------------------------------------------------------
```

The results for the Random Forest for both training datasets are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580
Logistic Regression	Reduced	0.928	0.972	27	444	17940
Logistic Regression	Extended	0.963	0.837	14	2550	32500
Boosted Logistic Regression	Reduced	0.928	0.948	27	816	21660
Boosted Logistic Regression	Extended	0.941	0.957	22	676	17760
Random Forest	Reduced	0.912	0.982	33	288	19380
Random Forest	Extended	0.917	0.984	31	248	17980

The results demonstrate that the performance of the Random Forest models is similar to Logistic Regression models. Compared to Logistic Regression models, the Specificity decreases but is counter balanced from the increased sensitivity, to produce similar cost results. The Extended training dataset yields a marginally better overall performance from the Reduced dataset.

VI. Extreme Gradient Boosting

Extreme Gradient Boosting algorithm is an implementation of Gradient Boosting concept that uses a more regularized model formalization to control over-fitting, which gives better performance. It is characterized by very high execution speed and performance.

Extreme Gradient Boosting has various tuning parameters that can be optimized. The tree specific parameters will be optimized by using 4-fold cross validation. Parallel processing is also activated to reduce the computational time. Two Class Summary is selected as summary function in order to be able to use *Specificity* as performance metric for the model optimization.

The algorithm is applied both for the *Reduced* training dataset, as well as the *Extended*.

```
## XGBoost Reduced
#------------------------------------------------------------------------
# Parallel Processing settings
cl <- makePSOCKcluster(10)
registerDoParallel(cl)
#getDoParWorkers()
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

control_xgb1 <- trainControl(method= "cv", number = 4, p= 0.8,
```

```
                           #verboseIter = TRUE,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary,
                           allowParallel = TRUE)

# Initial Tuning Grid for optimization
#tuning_xgb1 <- expand.grid(nrounds = c(150, 200, 250),
#                           max_depth =  c(10, 15, 20),
#                           eta = c(0.2, 0.3, 0.4),
#                           gamma = c(5 , 20 ),
#                           colsample_bytree = c(0.3, 0.5, 0.7, 0.9),
#                           min_child_weight = 2,
#                           subsample = c(0.7, 0.8, 0.9))

# Optimized parameters for Specificity metric
tuning_xgb1 <- expand.grid(nrounds = c(250),
                           max_depth =  15,
                           eta = c(0.3),
                           gamma = c(5),
                           colsample_bytree = c(0.5),
                           min_child_weight = 2,
                           subsample = c(0.8))
#------------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

train_xgb1 <- train(x_train_f1, y_train_f1,
                    method = "xgbTree",
                    trControl = control_xgb1,
                    tuneGrid = tuning_xgb1,
                    metric = "Spec",
                    importance = TRUE,
                    allowParallel = TRUE)
```

```
## [19:22:35] WARNING: amalgamation/../src/learner.cc:516:
## Parameters: { allowParallel, importance } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
```

```
                    #verbose = TRUE)

stopCluster(cl)
registerDoSEQ()

pred_xgb1<- predict(train_xgb1, x_test_f1)
cm_xgb1 <- confusionMatrix(pred_xgb1, y_test)
#------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method="Extreme Gradient Boosting",
                                  Dataset = "Reduced",
                                  Specificity = cm_xgb1$byClass["Specificity"],
```

```
                                    Sensitivity = cm_xgb1$byClass["Sensitivity"],
                                    FN = cm_xgb1$table[1,2],
                                    FP = cm_xgb1$table[2,1],
                                    Cost = FN*500 +FP*10))
#-----------------------------------------------------------------------
```

```
## XGBoost Extended
#-----------------------------------------------------------------------
# Parallel Processing settings
cl <- makePSOCKcluster(10)
registerDoParallel(cl)
#getDoParWorkers()
#-----------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

control_xgb2 <- trainControl(method= "cv", number = 4, p= 0.8,
                             #verboseIter = TRUE,
                             classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             allowParallel = TRUE)

# Initial Tuning Grid for optimization
#tuning_xgb2 <- expand.grid(nrounds = c(150, 200),
#                           max_depth = c(10, 15, 20),
#                           eta = c(0.2, 0.3, 0.4),
#                           gamma = 5,
#                           colsample_bytree = c(0.5, 0.7, 0.9),
#                           min_child_weight = c( 2),
#                           subsample = c( 0.7, 0.8))

# Optimized parameters for Specificity metric
tuning_xgb2 <- expand.grid(nrounds = c(200),
                           max_depth = c(10),
                           eta = c(0.3),
                           gamma = 5,
                           colsample_bytree = c(0.9),
                           min_child_weight =  2,
                           subsample = c(0.8))
#-----------------------------------------------------------------------

set.seed(1989, sample.kind = "Rounding")

train_xgb2 <- train(x_train_f2, y_train_f2,
                    method = "xgbTree",
                    trControl = control_xgb2,
                    tuneGrid = tuning_xgb2,
                    metric = "Spec",
                    importance = TRUE,
                    allowParallel = TRUE)
```

```
## [19:23:39] WARNING: amalgamation/../src/learner.cc:516:
## Parameters: { allowParallel, importance } might not be used.
```

```
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
```

```
                    #verbose = TRUE)

stopCluster(cl)
registerDoSEQ()

pred_xgb2 <- predict(train_xgb2, x_test_f2)
cm_xgb2 <- confusionMatrix(pred_xgb2, y_test)
#------------------------------------------------------------------------

results<-bind_rows(results,tibble(Method="Extreme Gradient Boosting",
                                  Dataset = "Extended",
                                  Specificity = cm_xgb2$byClass["Specificity"],
                                  Sensitivity = cm_xgb2$byClass["Sensitivity"],
                                  FN = cm_xgb2$table[1,2],
                                  FP = cm_xgb2$table[2,1],
                                  Cost = FN*500 +FP*10))
#------------------------------------------------------------------------
```

The results for the Extreme Gradient Boosting model for both training datasets are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580
Logistic Regression	Reduced	0.928	0.972	27	444	17940
Logistic Regression	Extended	0.963	0.837	14	2550	32500
Boosted Logistic Regression	Reduced	0.928	0.948	27	816	21660
Boosted Logistic Regression	Extended	0.941	0.957	22	676	17760
Random Forest	Reduced	0.912	0.982	33	288	19380
Random Forest	Extended	0.917	0.984	31	248	17980
Extreme Gradient Boosting	Reduced	0.896	0.983	39	266	22160
Extreme Gradient Boosting	Extended	0.904	0.985	36	228	20280

Although in general, Gradient Boosted Trees outperform Random Forest, the results demonstrate that the performance of the Extreme Gradient Boosted model in the examined case is marginally lower. Overall, they achieve similar sensitivity with the Random Forest models but lower specificity, resulting in a higher cost.

VII. Ensemble

Finally, an ensemble will be created by combining the results of the machine learning models presented. The ensemble will consist of the logistic regression, the boosted logistic regression, the random forest and the extreme gradient boosted models since they achieve similar performance. The simplified NA method and the kmeans clustering method will not be used for the ensemble since they are simplified approaches that are indicatively applied and have a much lower performance compared to the other machine learning models.

The ensemble consists of the predicted results of 4 models. Consequently, there is a high possibility to have a "tie" event. However,since the cost of missing a failure event (positive class) has a cost of 500, it is more important to predict a positive class, even falsely, than to miss it. For that reason, for the ensemble method, we will predict a positive class when 2 out of 4 models predict positive.

An ensemble with the aforementioned rule is created both for the results of the *Reduced* training dataset, as well as the *Extended*.

```
## Ensemble Reduced
#----------------------------------------

ensemble1 <- cbind(glm = pred_glm1,
                   lgb = pred_lgb1,
                   rf = pred_rf1,
                   xgb2 = pred_xgb1)

pred_en1 <- ifelse(rowMeans(ensemble1) >= 1.5, "pos", "neg")

pred_en1<-factor(pred_en1, levels= levels(y_test))
cm_en1 <- confusionMatrix(pred_en1, y_test)

results<-bind_rows(results,tibble(Method="Ensemble",
                                  Dataset = "Reduced",
                                  Specificity = cm_en1$byClass["Specificity"],
                                  Sensitivity = cm_en1$byClass["Sensitivity"],
                                  FN = cm_en1$table[1,2],
                                  FP = cm_en1$table[2,1],
                                  Cost = FN*500 +FP*10))

#----------------------------------------
```

```
## Ensemble Extended
#----------------------------------------

ensemble2 <- cbind(glm = pred_glm2,
                   lgb = pred_lgb2,
                   rf = pred_rf2,
                   xgb2 = pred_xgb2)

pred_en2 <- ifelse(rowMeans(ensemble2) >= 1.5, "pos", "neg")

pred_en2<-factor(pred_en2, levels= levels(y_test))
cm_en2 <- confusionMatrix(pred_en2, y_test)

results<-bind_rows(results,tibble(Method="Ensemble",
                                  Dataset = "Extended",
                                  Specificity = cm_en2$byClass["Specificity"],
```

```
                                    Sensitivity = cm_en2$byClass["Sensitivity"],
                                    FN = cm_en2$table[1,2],
                                    FP = cm_en2$table[2,1],
                                    Cost = FN*500 +FP*10))

#---------------------------------------
```

The results of the Ensembles for both training datasets are presented below:

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580
Logistic Regression	Reduced	0.928	0.972	27	444	17940
Logistic Regression	Extended	0.963	0.837	14	2550	32500
Boosted Logistic Regression	Reduced	0.928	0.948	27	816	21660
Boosted Logistic Regression	Extended	0.941	0.957	22	676	17760
Random Forest	Reduced	0.912	0.982	33	288	19380
Random Forest	Extended	0.917	0.984	31	248	17980
Extreme Gradient Boosting	Reduced	0.896	0.983	39	266	22160
Extreme Gradient Boosting	Extended	0.904	0.985	36	228	20280
Ensemble	Reduced	0.949	0.971	19	453	14030
Ensemble	Extended	0.973	0.964	10	566	10660

The results demonstrate that, although the 4 machine learning models used have similar performance, the ensemble achieves a significantly improved performance and subsequent cost reduction.

5. Results

Several approaches have been presented for the failure prediction in the Air Pressure System of Scania Trucks, starting from simplified methods and extending to more advanced methods. The first method applied, represents the performance that can be achieved purely, with an engineering approach into the problem. Comparing it with the performance of the kmeans-clustering, a simplified machine learning method, we realize that the application of machine learning techniques can prove beneficial in terms of improving the maintenance quality and reducing the associated costs.

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Simplified NA method	Raw	0.573	0.932	160	1059	90590
k-means Clustering	Scaled	0.712	0.983	108	258	56580

The results demonstrate, that by applying more advanced machine learning algorithms, the performance regarding failure prediction can be significantly improved. Overall, the applied methods achieve similar results, with the Random Forest achieving marginally better performance. Moreover, we observe that the models that utilize the Extended training dataset, that consists of more features, generally achieve better performance compared to the models of the Reduced dataset. However, the observed improvement, involves also increased computational complexity.

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Logistic Regression	Reduced	0.928	0.972	27	444	17940
Logistic Regression	Extended	0.963	0.837	14	2550	32500
Boosted Logistic Regression	Reduced	0.928	0.948	27	816	21660
Boosted Logistic Regression	Extended	0.941	0.957	22	676	17760
Random Forest	Reduced	0.912	0.982	33	288	19380
Random Forest	Extended	0.917	0.984	31	248	17980
Extreme Gradient Boosting	Reduced	0.896	0.983	39	266	22160
Extreme Gradient Boosting	Extended	0.904	0.985	36	228	20280

Finally, by creating an ensemble method from the 4 advanced machine learning models presented, the performance is significantly improved and the cost substantially minimized, with the ensemble for the *Extended* dataset achieving a cost of **10660**.

Method	Dataset	Specificity	Sensitivity	FN	FP	Cost
Ensemble	Reduced	0.949	0.971	19	453	14030
Ensemble	Extended	0.973	0.964	10	566	10660

6. Conclusions

Under the scope of this project, machine learning models that can effectively predict failures in the air pressure system have been developed. To achieve this, an exploratory analysis and respective preprocessing of the dataset were performed in the beginning. Two training datasets were generated, a Reduced and an Extended, referring to the amount of features that they consist of. Several models with different approaches were developed, starting from a simplified engineering method and kmeans-clustering, then logistic regression and boosted logistic regression, and extending to the advanced methods of random forest and extreme gradient boosting with tuning parameters that needed to be optimized. Finally, the performance of the models was evaluated with the test sets.

Overall, the machine learning models achieve similar performance, while the performance of the Random Forest models are marginally better. Moreover, training of the machine learning models with the Extended dataset leads to better results in general, but involves higher computational complexity. The ensemble of the models corresponding to the Extended training dataset achieved the best performance, with a minimum cost of **10660**.

The challenge of this project is to minimize the maintenance costs by effectively predicting failure events related to the APS. The results demonstrate that the application of machine learning to accomplish predictive maintenance can prove very beneficial in terms of improving the maintenance quality and reducing the associated costs.

The presented work is based on the insights of the Machine Learning Course of the respective Data Science Program and on dedicated literature review that was performed regarding binary classification. The completion of this project was a very productive and educative process. Following the principle of Genchi Genbutsu, it allowed me to acquire significant knowledge on data pre-processing and deepen on techniques regarding dataset dimension reduction, features' selection, principal component analysis and data balancing, as well as, build a solid background on machine learning models and approaches related to binary classification. The examined case was the Industrial Challenge at the 15th International Symposium on Intelligent Data Analysis, 2016. The achieved costs of the top three contestants were *9920*, *10900* and *11480*. The minimum achieved cost of the presented work, **10660**, is certainly rewarding and reflects the effort that was put through and the level of comprehension towards effectively dealing with the examined problem.

The respective publication released by Scania CV AB on the examined case, explains that one of the main motivations was to express how operational data influence maintenance. To this end, data have to be categorized according to their importance and a respective influence value needs to be set for every feature, with a scalar from 1 to 9 for example. In this respect, a set of optimized rules can be created that allows experts to improve the maintenance quality. This engineering specific problem has not been addressed in this work, as it is not related with the requirements of the Capstone but is intended as future work. The categorization of the features can be achieved by combining the indicated variable importance from the Random Forest and Extreme Gradient Boosting and the rotation matrix of the principal component analysis that represents the weighting of every feature of the data.

7. References

1. Introduction to Data Science - Data Analysis and Prediction Algorithms with R
2. Lindgren T., Biteus J. Expert Guided Adaptive Maintenance
3. Kaggle Notebooks
4. Classification with correlated features: unreliability of feature ranking and solutions
5. Why Feature Correlation Matters... A Lot!
6. An Introduction to corrplot Package
7. Deep Learning Book Series · 2.8 Singular Value Decomposition
8. PCA: A Practical Guide to Principal Component Analysis in R & Python
9. PRINCIPAL COMPONENT ANALYSIS IN R
10. Practical Guide to deal with Imbalanced Classification Problems in R
11. Learning from Imbalanced Classes
12. SMOTE Algorithm For Unbalanced Classification Problems
13. Why is accuracy not the best measure for assessing classification models?
14. The caret Package
15. Top 10 Binary Classification Algorithms
16. Logistic Regression for Machine Learning.&text=Techniques%20used%20to%20learn%20the,logistic%20regression%20m
17. Prediction Based On LogitBoost Classification Algorithm
18. Friedman J., Hastie T., Tibshirani R. (2000) Additive Logistic Regression: A Statistical View of Boosting, The Annual of Statistics, Vol.28, No.2, 337-407
19. A COMPLETE GUIDE TO THE RANDOM FOREST ALGORITHM
20. Random Forest, Wikipedia
21. How to use XGBoost algorithm in R in easy steps
22. Tuning xgboost in R: Part I
23. Gradient Boosting, Wikipedia