Modelo de Classificação em R

Carlos Augusto Schneider

19/10/2022

ETAPA 1 - Carregamento e preparação do dataframe

Definindo o diretório de trabalho:

```
setwd("/home/carlos/FCD/BigDataRAzure/projetos/projeto2")
getwd()
```

```
## [1] "/home/carlos/FCD/BigDataRAzure/projetos/projeto2"
```

Carregando os pacotes exigidos

```
library(RWeka)
library(tidyverse)
library(corrgram)
library(caret)
library(ROSE)
library(corrgram)
library(naivebayes)
library(kernlab)
library(h2o)
library(ggbeeswarm)
library(kableExtra)
library(tinytex)
```

Carregando o dataset

Descrição das variáveis

Size: Tamanho da lata de combustível a incendiar em cm. Apesar de numérica, também é uma variável fator; 1 = 7cm, 2 = 12cm, 3 = 14cm, 4 = 16cm, 5 = 20c, 6 = Metade cheia GLP, 7 = Totalmente Cheia GLP.

Fuel: Tipo de combustível testado para extinção do incêncio;

Distance:: Distância do combustível incendiado para o mecanismo extintor em cm;

Desibel: Volume Sonoro do mecanismo extintor em db;

Airflow: Fluxo de ar resultante das ondas sonoras (m/s);

Frequency: Frequência da onda sonora (Hz);

Class: Status (1 = incêndio extinto ou 0 = não extinto).

Verificando a existência de valores não disponíveis (NA)

```
any(is.na(df))
```

```
## [1] FALSE
```

Como a variável Size tem especificações diferentes em relação ao tipo de combustível (conforme descrição), decidimos convertê-la para o tipo fator. Vamos também colocar a variável target na 1ª posição do dataframe.

```
colnames(df)
```

```
## [1] "SIZE"      "FUEL"      "DISTANCE"  "DESIBEL"   "AIRFLOW"   "FREQUENCY"
## [7] "CLASS"
```

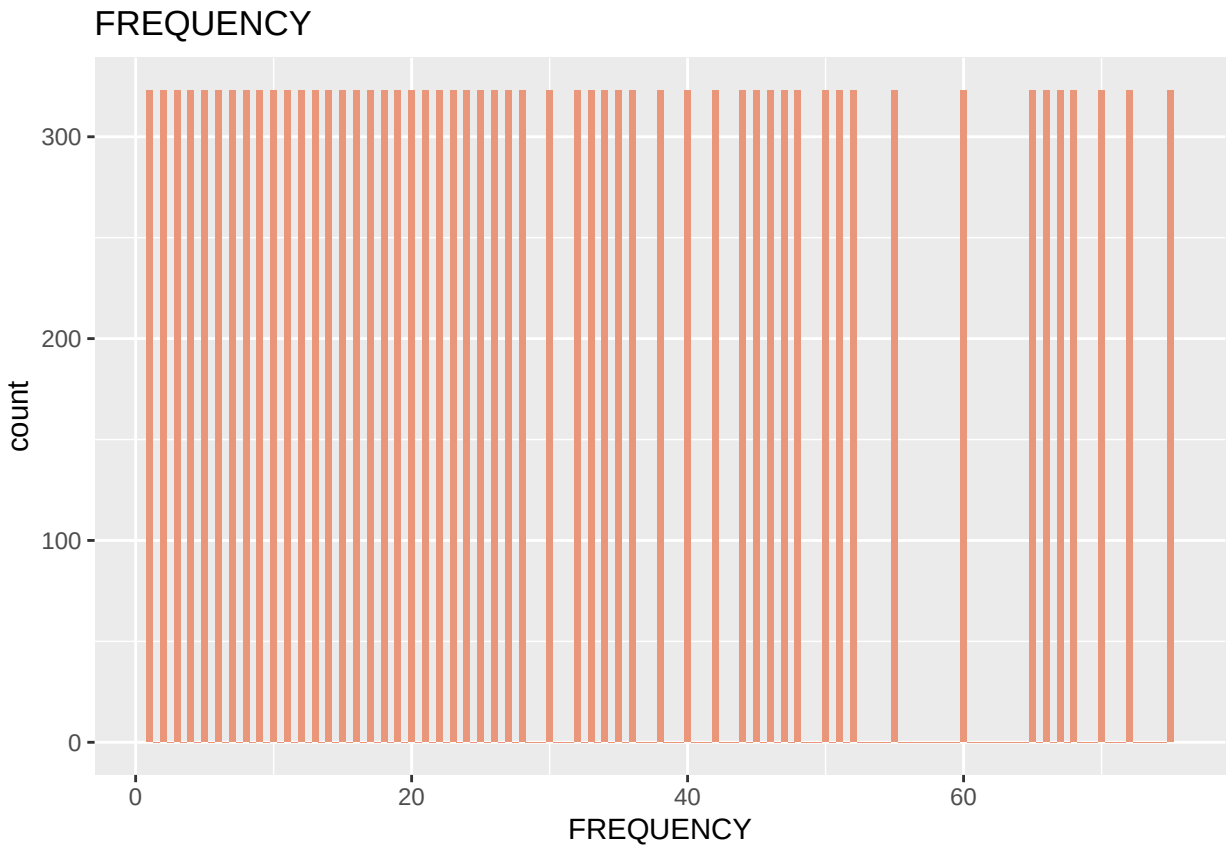
```
df1 <- df %>%
  select(CLASS,1,2,3,4,5,6)
view(df1)

df1$SIZE <- as.factor(df1$SIZE)
str(df1)
```

```
## 'data.frame':    17442 obs. of  7 variables:
##  $ CLASS    : Factor w/ 2 levels "0","1": 1 2 2 2 2 2 2 2 2 2 ...
##  $ SIZE     : Factor w/ 7 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ FUEL     : Factor w/ 4 levels "gasoline","thinner",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ DISTANCE : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ DESIBEL  : num  96 96 96 96 109 109 103 95 102 93 ...
##  $ AIRFLOW  : num  0 0 2.6 3.2 4.5 7.8 9.7 12 13.3 15.4 ...
##  $ FREQUENCY: num  75 72 70 68 67 66 65 60 55 52 ...
```

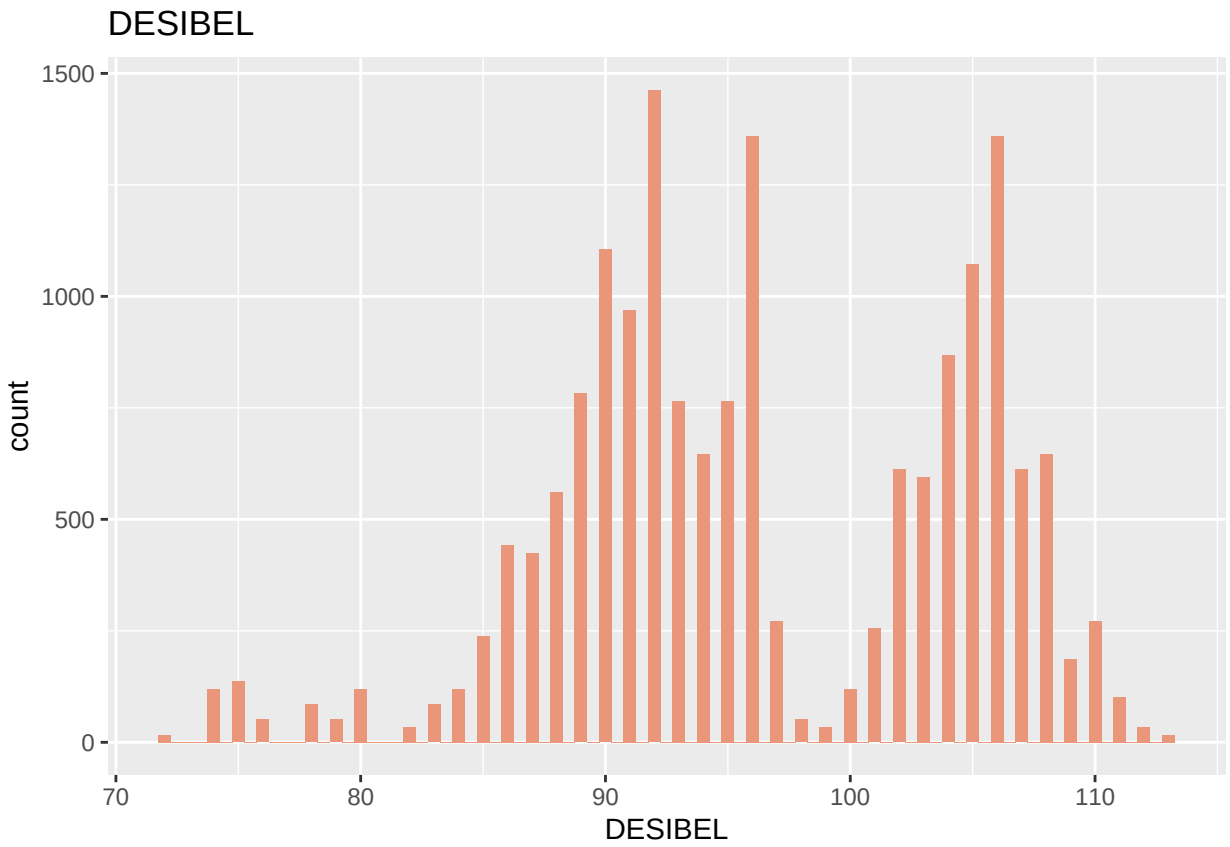
ETAPA 2 - Análise Exploratória

Histogramas das variáveis numéricas

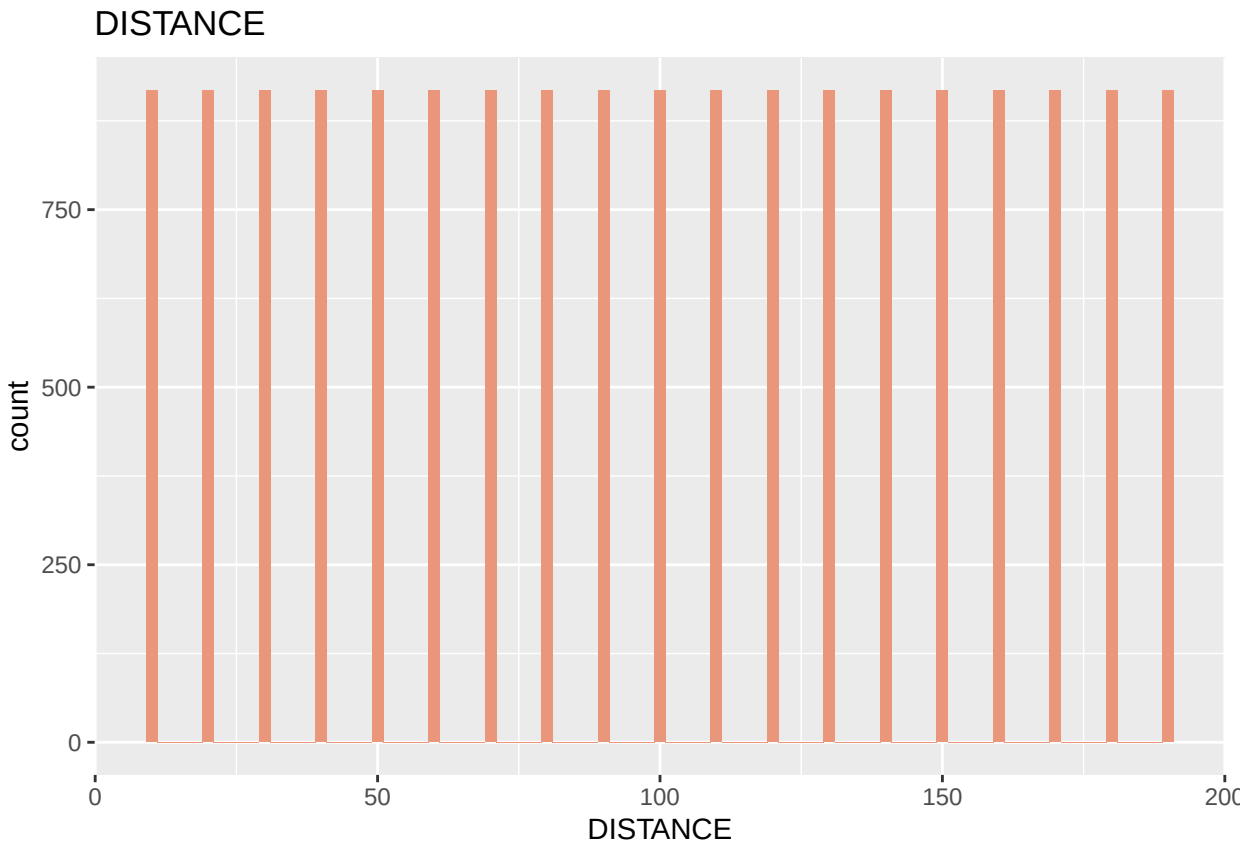
```
ggplot(df1, aes(x = FREQUENCY)) +
  geom_histogram(binwidth = 0.5, fill = "darksalmon") +
  ggtitle("FREQUENCY")
```



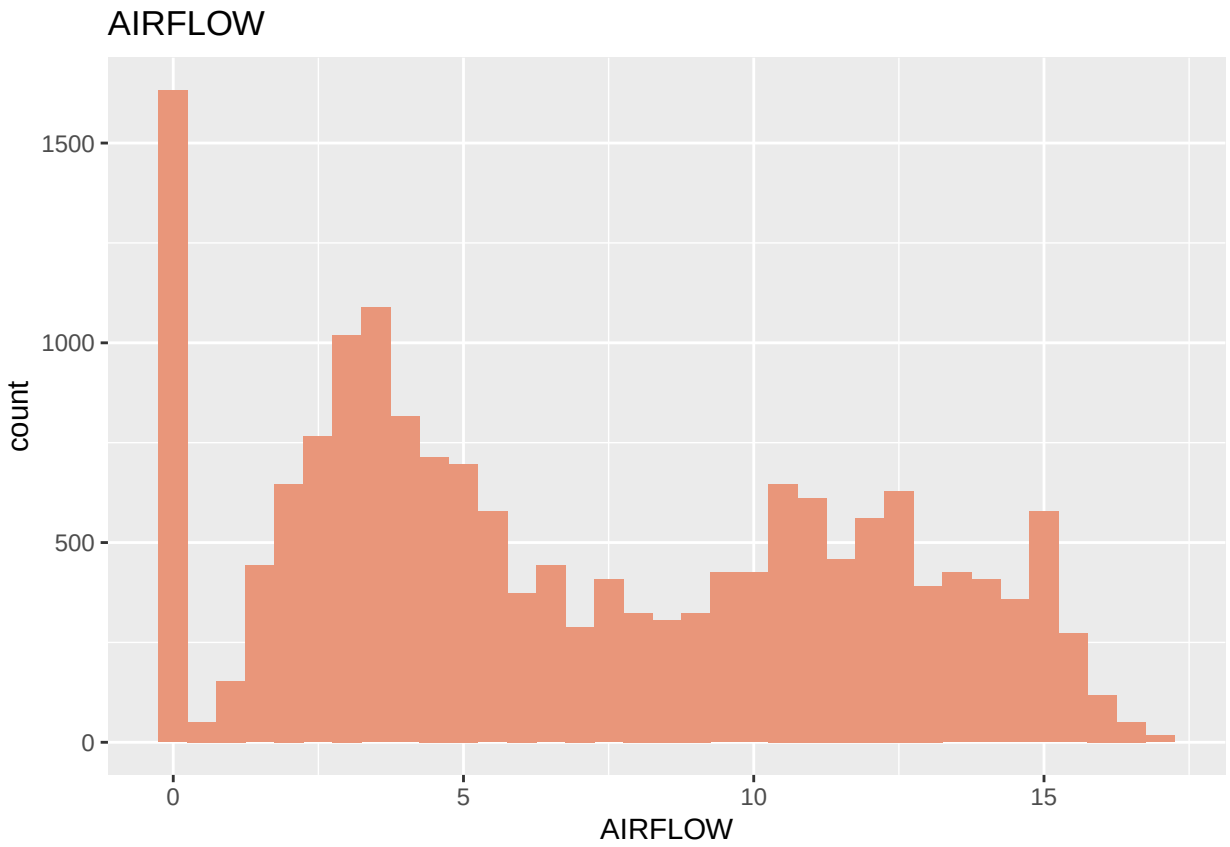

```
ggplot(df1, aes(x = DESIBEL)) +
  geom_histogram(binwidth = 0.5, fill = "darksalmon") +
  ggtitle("DESIBEL")
```




```
ggplot(df1, aes(x = DISTANCE)) +
  geom_histogram(binwidth = 2.0, fill = "darksalmon") +
  ggtitle("DISTANCE")
```




```
ggplot(df1, aes(x = AIRFLOW)) +
  geom_histogram(binwidth = 0.5, fill = "darksalmon") +
  ggtitle("AIRFLOW")
```



Chama a atenção o grande número de observações iguais a zero da variável AIRFLOW.

```
nrow(filter(df1,AIRFLOW == 0))
```

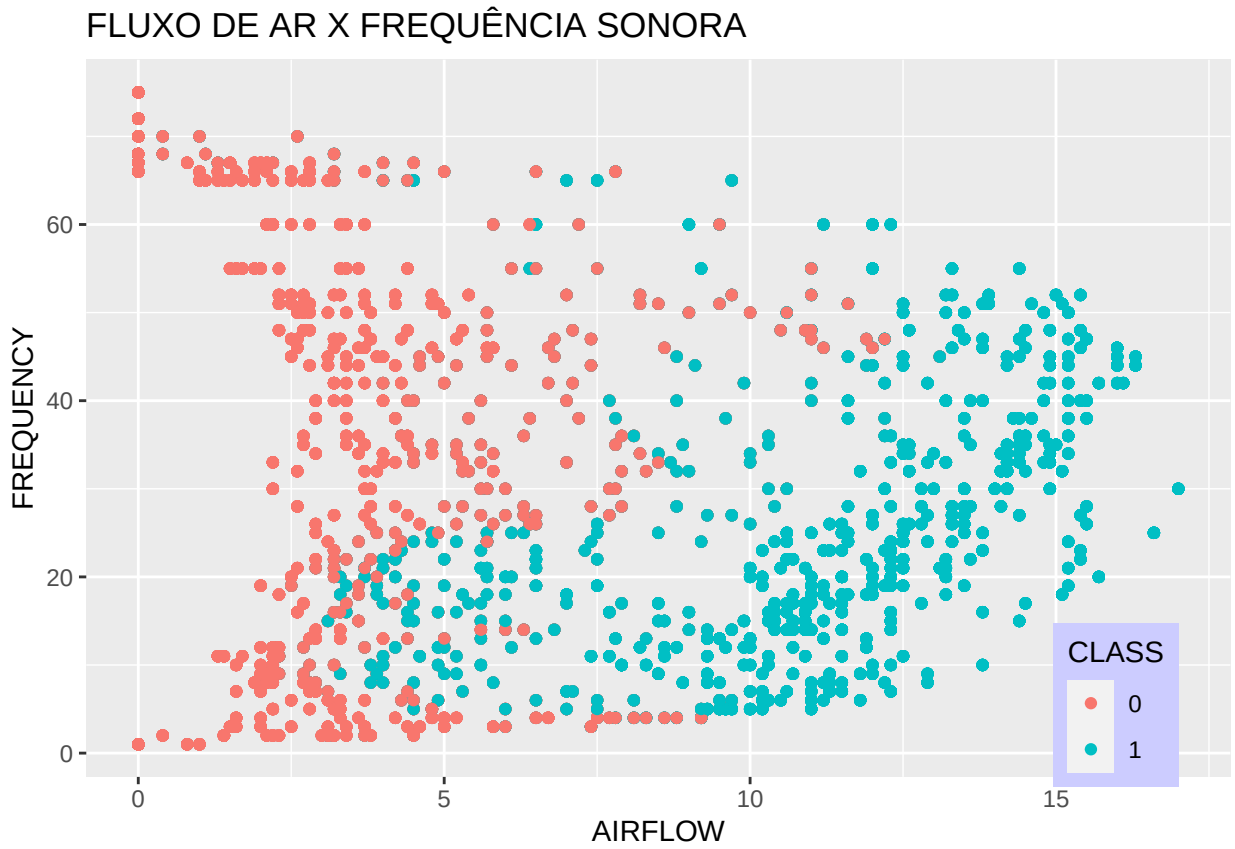
```
## [1] 1632
```

Vamos criar uma coluna do tipo fator, indicando "sim" para AIRFLOW = 0 e "nao" para AIRFLOW diferente de 0 e relacionar às outras variáveis do dataframe. Será uma variável do tipo fator.

```
indice <- which(df1$AIRFLOW == 0)
df1$AFzero <- "nao"
df1[indice,which(colnames(df1)=="AFzero")] = "sim"
df1$AFzero <- as.factor(df1$AFzero)

rm(indice)
```

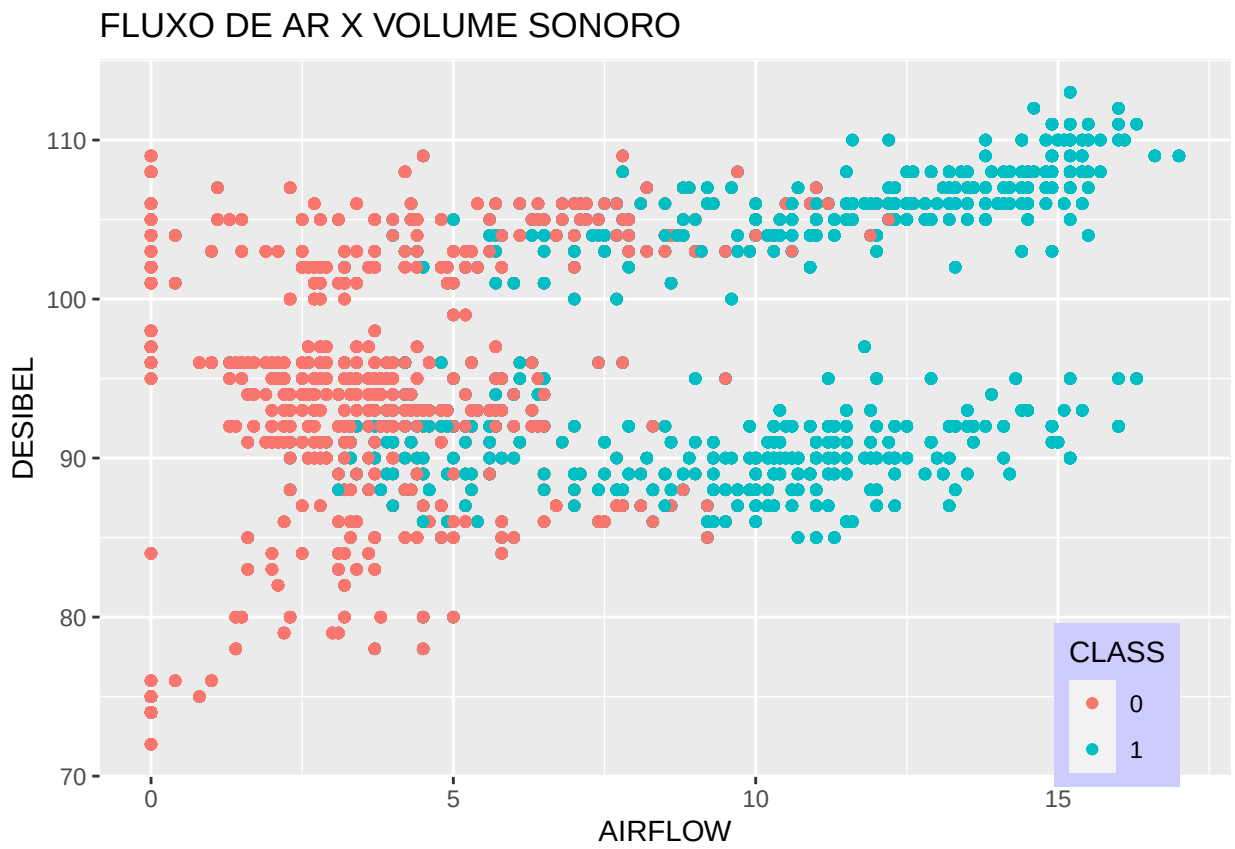
Analisando as variáveis numéricas

```
ggplot(df1, aes(x = AIRFLOW, y = FREQUENCY, color = CLASS)) +
  geom_point() +
  ggtitle("FLUXO DE AR X FREQUÊNCIA SONORA")+
  theme(legend.position = c(0.9,0.1), legend.background = element_rect(fill = "#ccccff"))
```



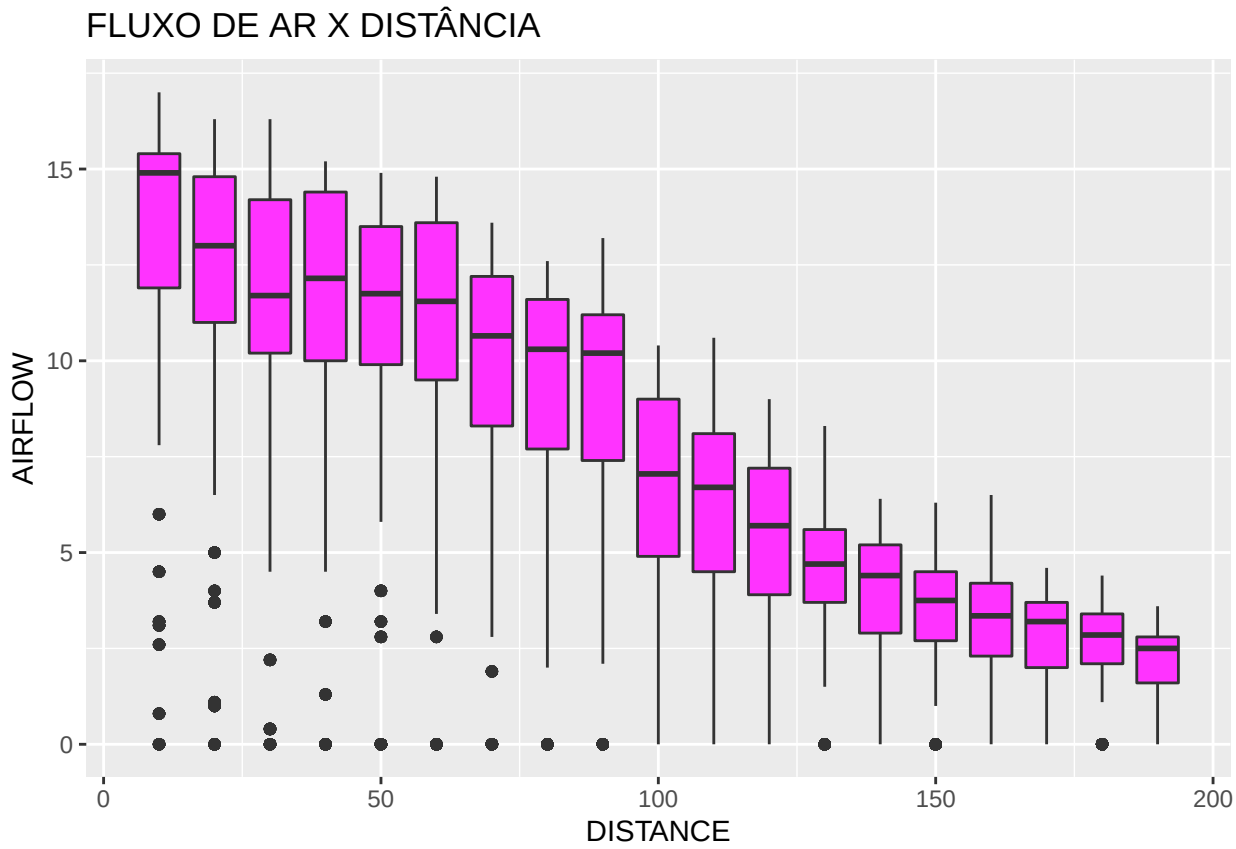
A extinção do incêndio está fortemente associada a níveis maiores de fluxo de ar. Quanto à frequência, percebe-se que frequências próximas de zero ou superiores a 60 hz geram menos corrente de ar.

```
ggplot(df1, aes(x = AIRFLOW, y = DESIBEL, color = CLASS)) +
  geom_point() +
  ggtitle("FLUXO DE AR X VOLUME SONORO")+
  theme(legend.position = c(0.9,0.1), legend.background = element_rect(fill = "#ccccff"))
```



Já em relação ao volume, percebe-se claramente que volumes maiores geram maior fluxo de ar, não havendo incêndios extintos com volumes abaixo de 80db.

```
ggplot(df1, aes(x = DISTANCE, y = AIRFLOW, group = DISTANCE)) +
  geom_boxplot(fill = "#ff33ff") +
  ggtitle("FLUXO DE AR X DISTÂNCIA")
```

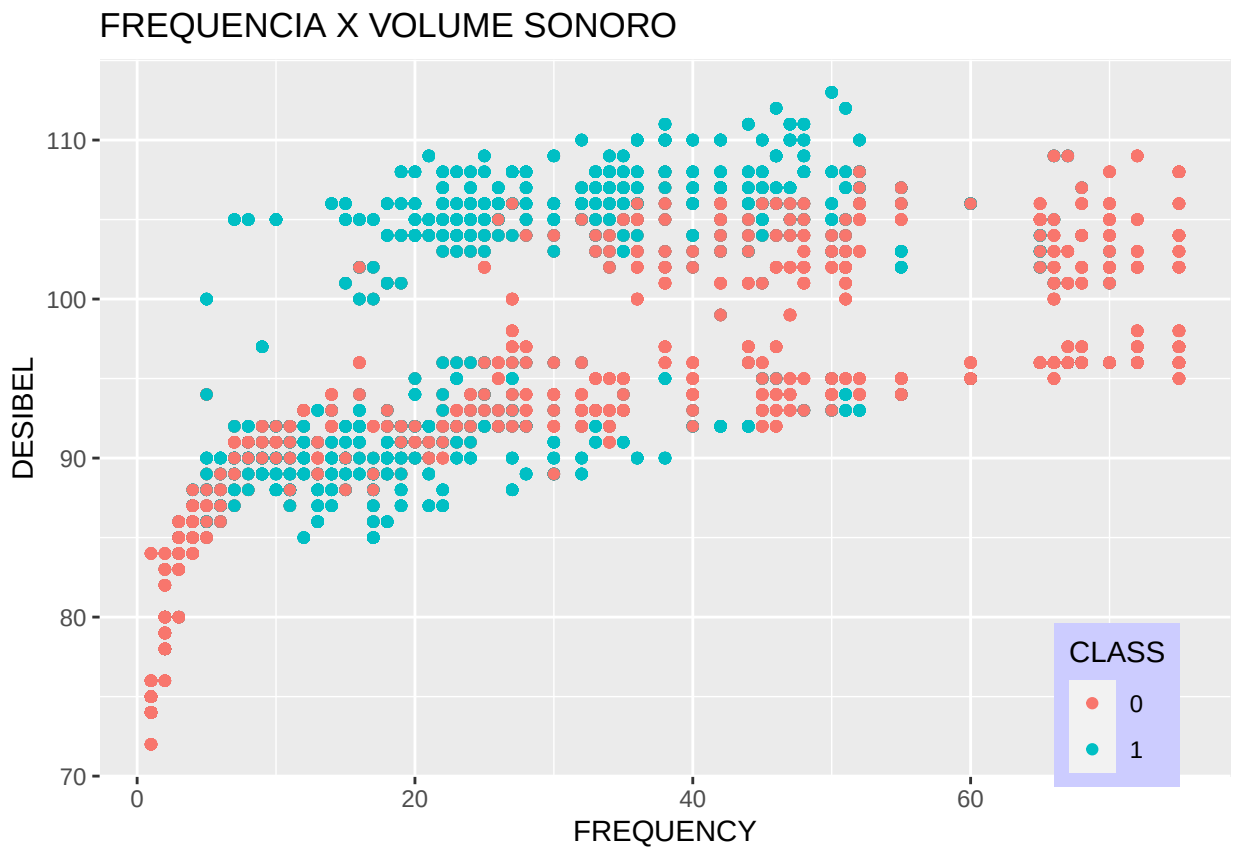


Em relação ao aumento da distância, percebe-se que o fluxo de ar claramente diminui.

```
corrgram(df1[4:7],cor.method = "pearson", panel = panel.cor)
```

A correlação do volume sonoro com a fluxo de ar é positiva (0,38) e a correlação da frequência com o fluxo de ar é negativa (-0,21). A correlação mais importante é do fluxo de ar com a distância (correlação negativa de -0,71).

```
ggplot(df1, aes(x = FREQUENCY, y = DESIBEL, color = CLASS)) +
  geom_point() +
  ggtitle("FREQUENCIA X VOLUME SONORO")+
  theme(legend.position = c(0.9,0.1), legend.background = element_rect(fill = "#ccccff"))
```



Aqui fica bem claro que a maior parte dos sucessos para extinção do incêndio situam-se em frequências entre 20 e 50 Hz com volume alto (acima de 90 db).

```
tabela <- prop.table(table(df1$CLASS,df1$AFzero))*100
kable(tabela,caption = "Resultados - Modelo AutoML",format = "pipe")
```

Table 1: Resultados - Modelo AutoML

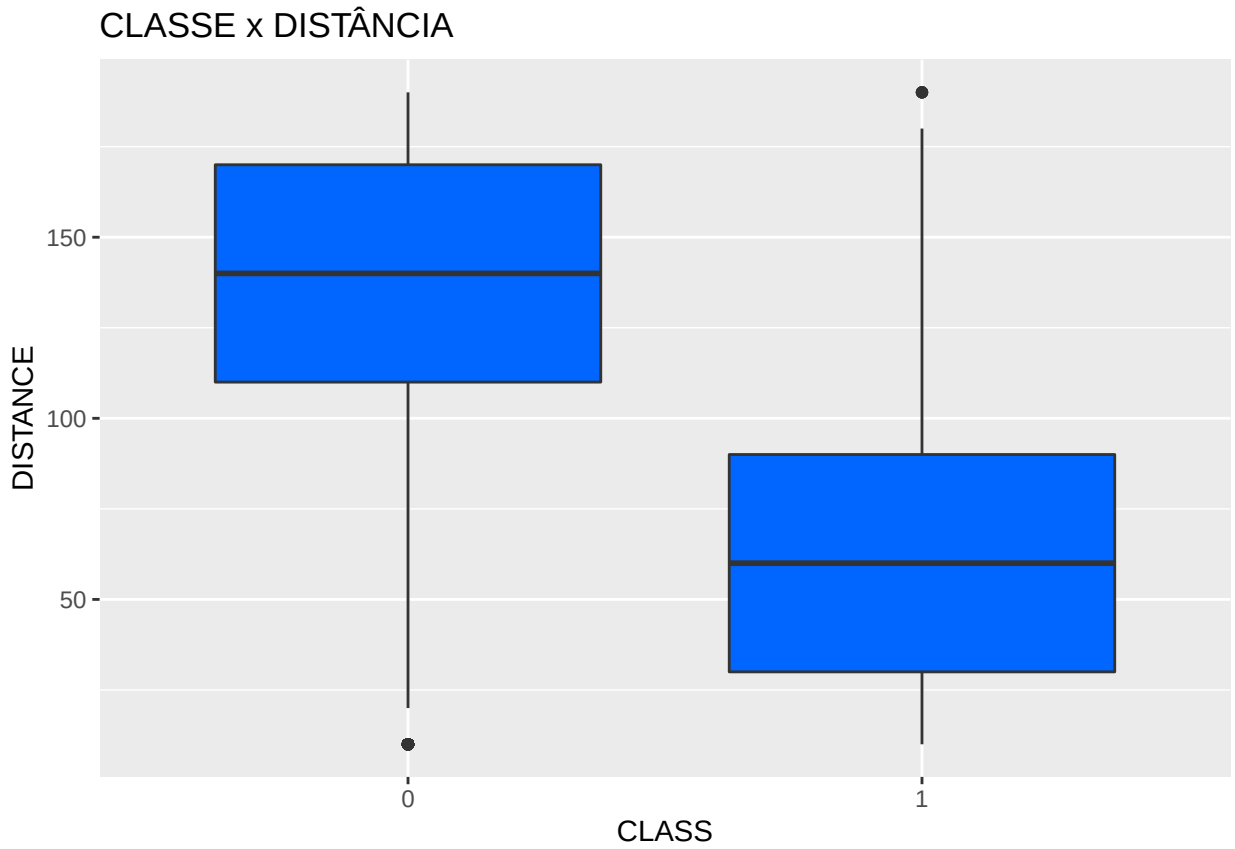
	nao	sim
0	40.96434	9.2535260
1	49.67894	0.1031992

```
rm(tabela)
```

Percebemos que em uma pequena parcela das observações, houve extinção do incêndio mesmo com fluxo de ar igual a zero (0,10% das observações do dataset, aproximadamente).

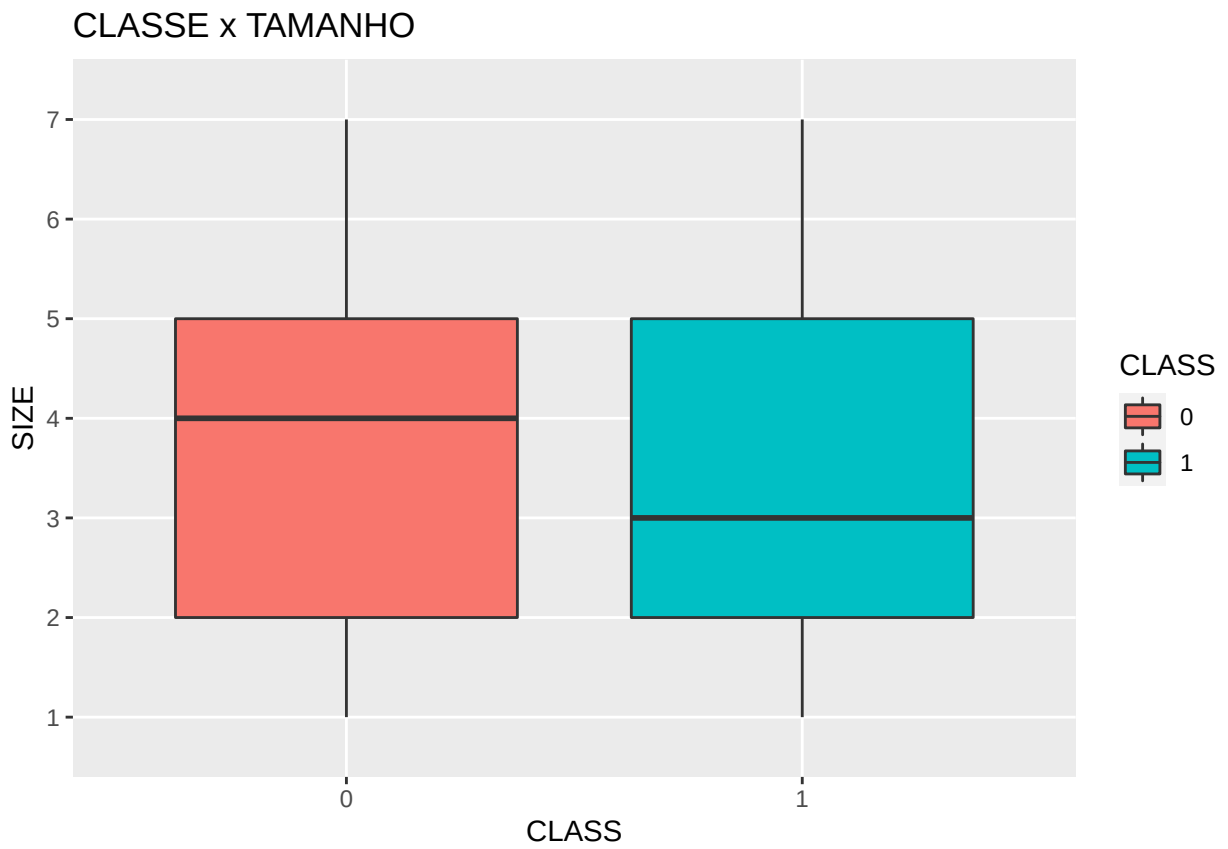
Gerando boxplots para análise das variáveis fator:

```
ggplot(df1, aes(x = CLASS, y = DISTANCE))+
  geom_boxplot(fill = "#0066FF") +
  ggtitle("CLASSE x DISTÂNCIA")
```



Percebe-se aqui que há impacto claro da distância do aparato ao incêndio quanto ao sucesso na extinção.

```
ggplot(df1, aes(x = CLASS, y = SIZE, group = CLASS, fill = CLASS))+
  geom_boxplot()+
  ggtitle("CLASSE x TAMANHO")
```



Percebe-se que incêndios de tamanhos menores tem maior sucesso de extinção, mas o impacto dessa variável parece ser menor que o da variável DISTANCE.

```
tabela <- prop.table(table("Combustível" =  df1$FUEL,"Resultado" = df1$CLASS))*100
kable(tabela,caption = "Resultados - Modelo AutoML",format = "pipe")
```

Table 2: Resultados - Modelo AutoML

	0	1
gasoline	13.650958	15.760807
thinner	15.147346	14.264419
kerosene	16.230937	13.180828
lpg	5.188625	6.576081

```
rm(tabela)
```

Percebe-se que alguns combustíveis são menos suscetíveis ao sistema extintor (tiner e querosene), pois têm maior percentual de fracassos que sucessos.

ETAPA 3 - Treinamento e validação do modelo

Gerando os dados de treino e de teste do modelo

```
set.seed(123)
training.samples <- df1$CLASS %>%
  createDataPartition(p = 0.7, list = FALSE)
train.data  <- df1[training.samples, ]
```

```
test.data <- df1[-training.samples, ]
```

Os datasets estão balanceados.

```
tabela1 <- prop.table(table(train.data$CLASS))*100
kable(tabela1,caption = "Resultados - Modelo AutoML",format = "pipe")
```

Table 3: Resultados - Modelo AutoML

Var1	Freq
0	50.21702
1	49.78298

```
tabela2 <- prop.table(table(test.data$CLASS))*100
kable(tabela2,caption = "Resultados - Modelo AutoML",format = "pipe")
```

Table 4: Resultados - Modelo AutoML

Var1	Freq
0	50.21984
1	49.78016

```
rm(tabela1,tabela2)
```

Usando "cross validation" - validação cruzada, neste caso com 5 combinações diferentes para seleção do melhor modelo.

```
control1 <- trainControl(method = "cv", number = 5)
```

Modelo Random Forest

```
model_rf1 <- train(CLASS ~ .,
                   seed = 123,
                   data = train.data,
                   method = "rf",
                   trControl = control1,
                   metric = "Accuracy")
```

Modelo Naive_Bayes

```
model_nb1 <- train(CLASS ~ .,
                   seed = 123,
                   data = train.data,
                   method = "naive_bayes",
                   trControl = control1,
                   metric = "Accuracy")
```

Modelo SVM (Support Vector Machine)

```
model_svm1 <- train(CLASS ~ .,
                  seed = 123,
                  data = train.data,
                  method = "svmLinear",
                  trControl = control1,
                  metric = "Accuracy")
```

Avaliando os modelos

```
model_rf1$results #R2 (max Accuracy 0,96)
```

```
##   mtry  Accuracy     Kappa  AccuracySD    KappaSD
## 1    2 0.9216272 0.8432374 0.006453184 0.01290244
## 2    8 0.9583165 0.9166285 0.007032922 0.01406495
## 3   14 0.9588900 0.9177773 0.007654260 0.01530897
```

```
model_nb1$results #R2 (max Accuracy 0,86)
```

```
##   usekernel laplace adjust  Accuracy     Kappa  AccuracySD    KappaSD
## 1     FALSE       0      1 0.8561126 0.7124024 0.010418136 0.02083984
## 2      TRUE       0      1 0.8380143 0.6763737 0.006470448 0.01292992
```

```
model_svm1$results # (Accuracy 0,90)
```

```
##   C  Accuracy     Kappa  AccuracySD    KappaSD
## 1 1 0.9009906 0.8019673 0.005073965 0.01015711
```

Modelo Random Forest teve a melhor acurácia (0,96), seguido pelo modelo SVM (0,90) e pelo Naive Bayes (0,86). Vamos utilizar o modelo Random Forest para fazer as previsões:

```
previsoes <- predict(model_rf1, test.data)
```

Vamos verificar a acurácia

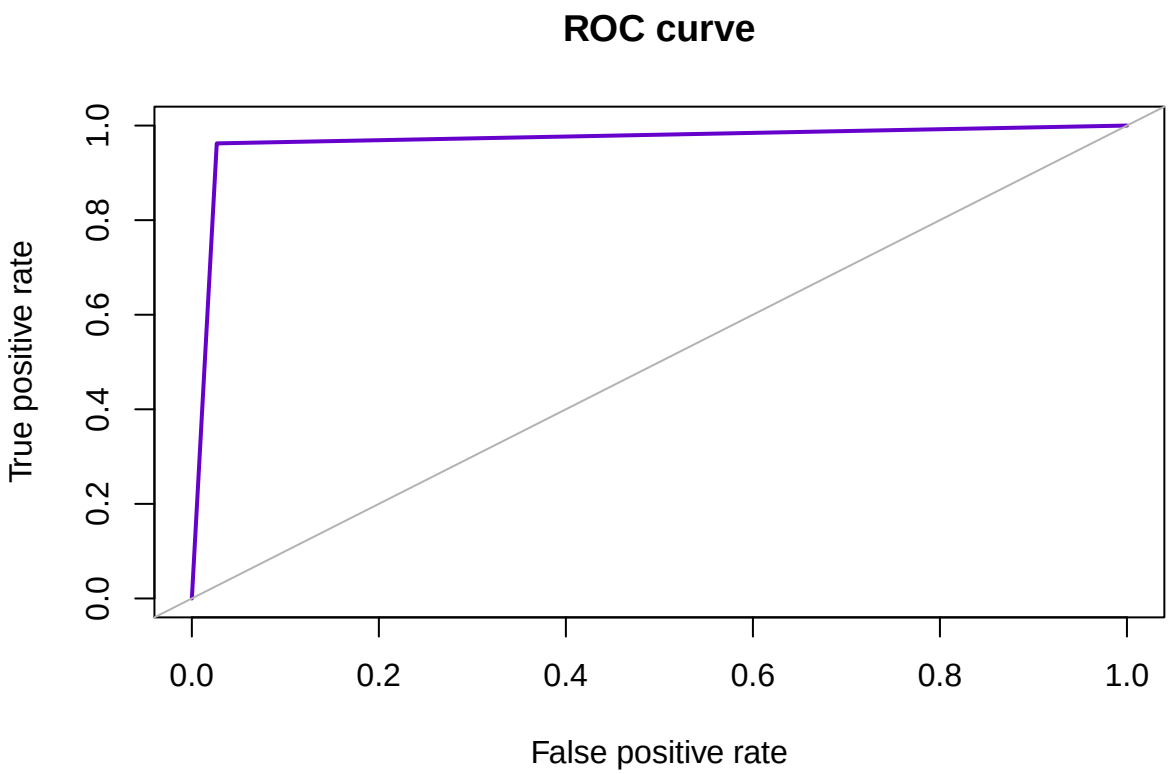
```
caret::confusionMatrix(test.data$CLASS, previsoes, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2557   70
##          1   98 2506
##
##                Accuracy : 0.9679
##                  95% CI : (0.9627, 0.9725)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9358
##
##  Mcnemar's Test P-Value : 0.03724
##
##             Sensitivity : 0.9728
##             Specificity : 0.9631
##          Pos Pred Value : 0.9624
##          Neg Pred Value : 0.9734
##              Prevalence : 0.4924
##          Detection Rate : 0.4791
##    Detection Prevalence : 0.4978
##       Balanced Accuracy : 0.9680
##
##        'Positive' Class : 1
##
```

Accuracy de 0,9677

Calculando o Score AUC

```
roc.curve(test.data$CLASS, previsoes, plotit = T, col = "#6600cc")
```




```
## Area under the curve (AUC): 0.968
```

ETAPA 4 - Modelo Auto ML

Vamos elaborar um modelo AutoML com o pacote H20: Iniciando o H2O

Convertendo os dados para o formato H2O

```
h2o_frame <- as.h2o(df1)
```

```
##   |                                                                      |
```

```
tabela <- head(h2o_frame)
kable(tabela,caption = "Resultados - Modelo AutoML",format = "pipe")
```

Table 5: Resultados - Modelo AutoML

CLASS	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	AFzero
0	1	gasoline	10	96	0.0	75	sim
1	1	gasoline	10	96	0.0	72	sim
1	1	gasoline	10	96	2.6	70	nao
1	1	gasoline	10	96	3.2	68	nao
1	1	gasoline	10	109	4.5	67	nao

CLASS	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	AFzero
1	1	gasoline	10	109	7.8	66	nao

```
rm(tabela)
```

Split dos dados em treino e teste

```
h2o_frame_split <- h2o.splitFrame(h2o_frame, ratios = 0.70, seed = 123)
```

Modelo AutoML Não vamos fazer balanceamento de classes (não será necessário)

```
modelo_automl <- h2o.automl(y = 'CLASS',
                            seed = 123,
                            balance_classes = FALSE,
                            training_frame = h2o_frame_split[[1]],
                            leaderboard_frame = h2o_frame_split[[2]],
                            max_runtime_secs = 60 * 10,
                            include_algos = c('XGBoost', 'DRF', 'DeepLearning'),
                            sort_metric = "auc")
```

```
##   |                                                                              |
```

```
leaderboard_automl <- as.data.frame(modelo_automl@leaderboard)
leaderboard_automl
```

```
##                                                model_id       auc    logloss
## 1       XGBoost_grid_1_AutoML_3_20221023_100746_model_4 0.9980303 0.05659203
## 2       XGBoost_grid_1_AutoML_3_20221023_100746_model_6 0.9977436 0.06243445
## 3       XGBoost_grid_1_AutoML_3_20221023_100746_model_8 0.9977120 0.06458639
## 4                    XGBoost_3_AutoML_3_20221023_100746 0.9975320 0.06465715
## 5                    XGBoost_2_AutoML_3_20221023_100746 0.9974615 0.06542672
## 6      XGBoost_grid_1_AutoML_3_20221023_100746_model_10 0.9974290 0.06727234
## 7       XGBoost_grid_1_AutoML_3_20221023_100746_model_5 0.9974256 0.06436371
## 8       XGBoost_grid_1_AutoML_3_20221023_100746_model_3 0.9974191 0.06586742
## 9       XGBoost_grid_1_AutoML_3_20221023_100746_model_1 0.9974021 0.06756636
## 10      XGBoost_grid_1_AutoML_3_20221023_100746_model_9 0.9972798 0.07426946
## 11      XGBoost_grid_1_AutoML_3_20221023_100746_model_2 0.9971409 0.07529752
## 12                   XGBoost_1_AutoML_3_20221023_100746 0.9966334 0.07633966
## 13                       DRF_1_AutoML_3_20221023_100746 0.9964895 0.09213585
## 14      XGBoost_grid_1_AutoML_3_20221023_100746_model_7 0.9964776 0.07903766
## 15 DeepLearning_grid_1_AutoML_3_20221023_100746_model_1 0.9941930 0.10216675
## 16     XGBoost_grid_1_AutoML_3_20221023_100746_model_11 0.9926755 0.12035363
## 17                       XRT_1_AutoML_3_20221023_100746 0.9913751 0.13598869
## 18              DeepLearning_1_AutoML_3_20221023_100746 0.9889232 0.13823426
## 19 DeepLearning_grid_1_AutoML_3_20221023_100746_model_2 0.9842373 0.18862917
## 20 DeepLearning_grid_1_AutoML_3_20221023_100746_model_3 0.9831711 0.17447598
## 21 DeepLearning_grid_1_AutoML_3_20221023_100746_model_4 0.9722906 0.22167615
## 22     XGBoost_grid_1_AutoML_3_20221023_100746_model_12 0.9703336 0.26178108
##        aucpr mean_per_class_error      rmse        mse
## 1  0.9980480           0.02201880 0.1287204 0.01656894
## 2  0.9977877           0.02452659 0.1348777 0.01819200
## 3  0.9977431           0.02451928 0.1366351 0.01866915
## 4  0.9975519           0.02510350 0.1373707 0.01887072
## 5  0.9975100           0.02586698 0.1394583 0.01944863
```

```
## 6  0.9974490           0.02451928 0.1395862 0.01948432
## 7  0.9974755           0.02356663 0.1360688 0.01851471
## 8  0.9974850           0.02510664 0.1390371 0.01933131
## 9  0.9974291           0.02297771 0.1391147 0.01935289
## 10 0.9973268           0.02742787 0.1455256 0.02117771
## 11 0.9971884           0.02876565 0.1470760 0.02163135
## 12 0.9967309           0.02856969 0.1500923 0.02252771
## 13 0.9965175           0.03031870 0.1598982 0.02556743
## 14 0.9965609           0.03128075 0.1526094 0.02328962
## 15 0.9940558           0.03591120 0.1682762 0.02831687
## 16 0.9927571           0.04462077 0.1867630 0.03488042
## 17 0.9916760           0.04885774 0.1976702 0.03907351
## 18 0.9889717           0.05678609 0.2041400 0.04167315
## 19 0.9840029           0.06489473 0.2349323 0.05519319
## 20 0.9832211           0.06779601 0.2301812 0.05298338
## 21 0.9729295           0.09326220 0.2633894 0.06937397
## 22 0.9712038           0.09173681 0.2738030 0.07496806
```

Extraindo o líder (modelo com melhor performance - XGBoost)

```
lider_automl <- modelo_automl@leader
```

Verificando o desempenho do modelo na previsão dos dados de teste.

```
predicted <- h2o.predict(lider_automl,h2o_frame_split[[2]])
```

```
##   |                                                                              |
```

```
performanceH20 <- h2o.performance(lider_automl,h2o_frame_split[[2]])
performanceH20
```

```
## H2OBinomialMetrics: xgboost
##
## MSE:  0.01656894
## RMSE:  0.1287204
## LogLoss:  0.05659203
## Mean Per-Class Error:  0.0220188
## AUC:  0.9980303
## AUCPR:  0.998048
## Gini:  0.9960607
## R^2:  0.9337241
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      2516   70 0.027069   =70/2586
## 1        44 2549 0.016969   =44/2593
## Totals 2560 2619 0.022012  =114/5179
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                       max f1  0.424948    0.978127 218
## 2                       max f2  0.139899    0.984606 295
## 3                 max f0point5  0.830393    0.982724 115
## 4                 max accuracy  0.508264    0.977988 200
## 5                max precision  0.999990    1.000000   0
## 6                   max recall  0.005010    1.000000 382
## 7             max specificity  0.999990    1.000000   0
```

```
## 8             max absolute_mcc  0.424948    0.956024 218
## 9   max min_per_class_accuracy  0.487855    0.977958 203
## 10 max mean_per_class_accuracy  0.508264    0.977989 200
## 11                     max tns  0.999990 2586.000000   0
## 12                     max fns  0.999990 1295.000000   0
## 13                     max fps  0.000015 2586.000000 399
## 14                     max tps  0.005010 2593.000000 382
## 15                     max tnr  0.999990    1.000000   0
## 16                     max fnr  0.999990    0.499422   0
## 17                     max fpr  0.000015    1.000000 399
## 18                     max tpr  0.005010    1.000000 382
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```
tabela <- h2o.confusionMatrix(performanceH20)
kable(tabela,caption = "Resultados - Modelo AutoML",format = "pipe")
```

Table 6: Resultados - Modelo AutoML

	0	1	Error	Rate
0	2516	70	0.0270688	=70/2586
1	44	2549	0.0169688	=44/2593
Totals	2560	2619	0.0220120	=114/5179

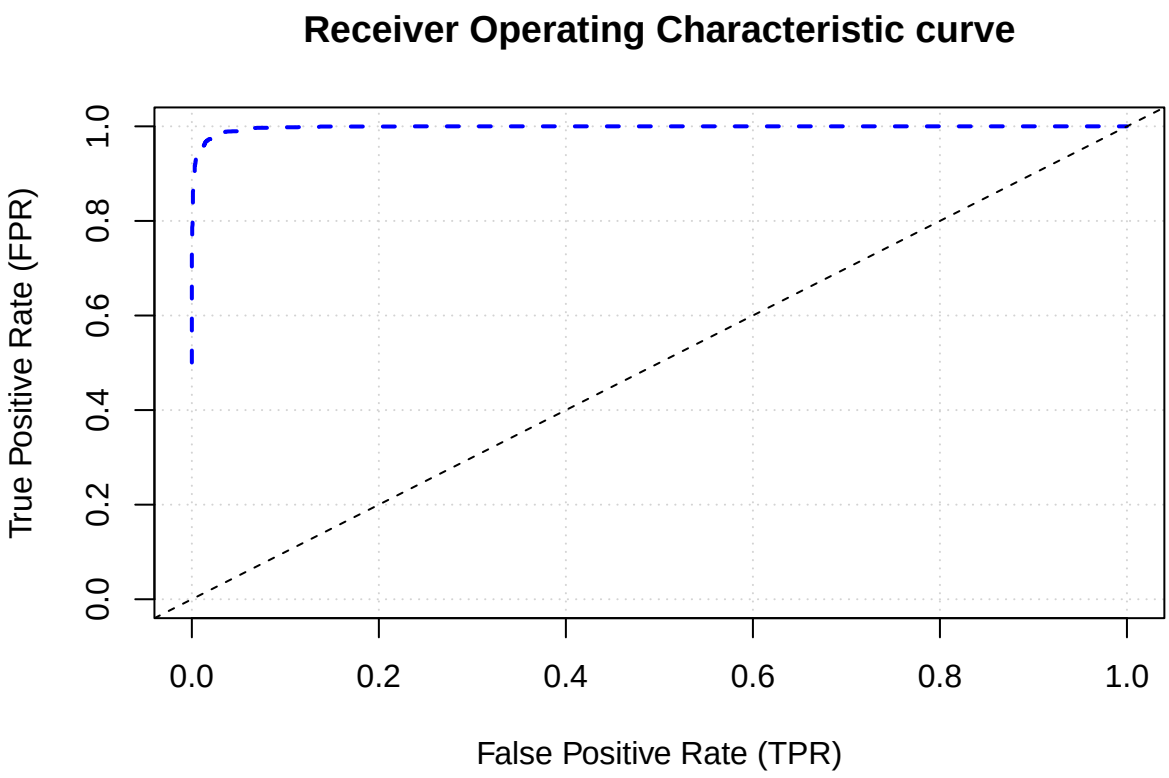
```
h2o.accuracy(performanceH20,thresholds = "max")
```

```
## [[1]]
## [1] 0.977988
```

```
rm(tabela)

#Métrica AUC
h2o.auc(performanceH20)
```

```
## [1] 0.9980303
```

```
plot(performanceH20,type='roc')
```



O desempenho do modelo AutoML foi superior ao modelo manual (0,98 contra 0,96 de acurácia). No entanto os dois modelos foram excelentes.

Para o melhor modelo extraímos a contribuição de cada variável para as previsões. Os valores extraídos são chamados de valores SHAP. Usamos os dados de teste.

```
var_contrib <- predict_contributions.H2OModel(lider_automl, h2o_frame_split[[2]])
```

```
##   |                                                                      |
```

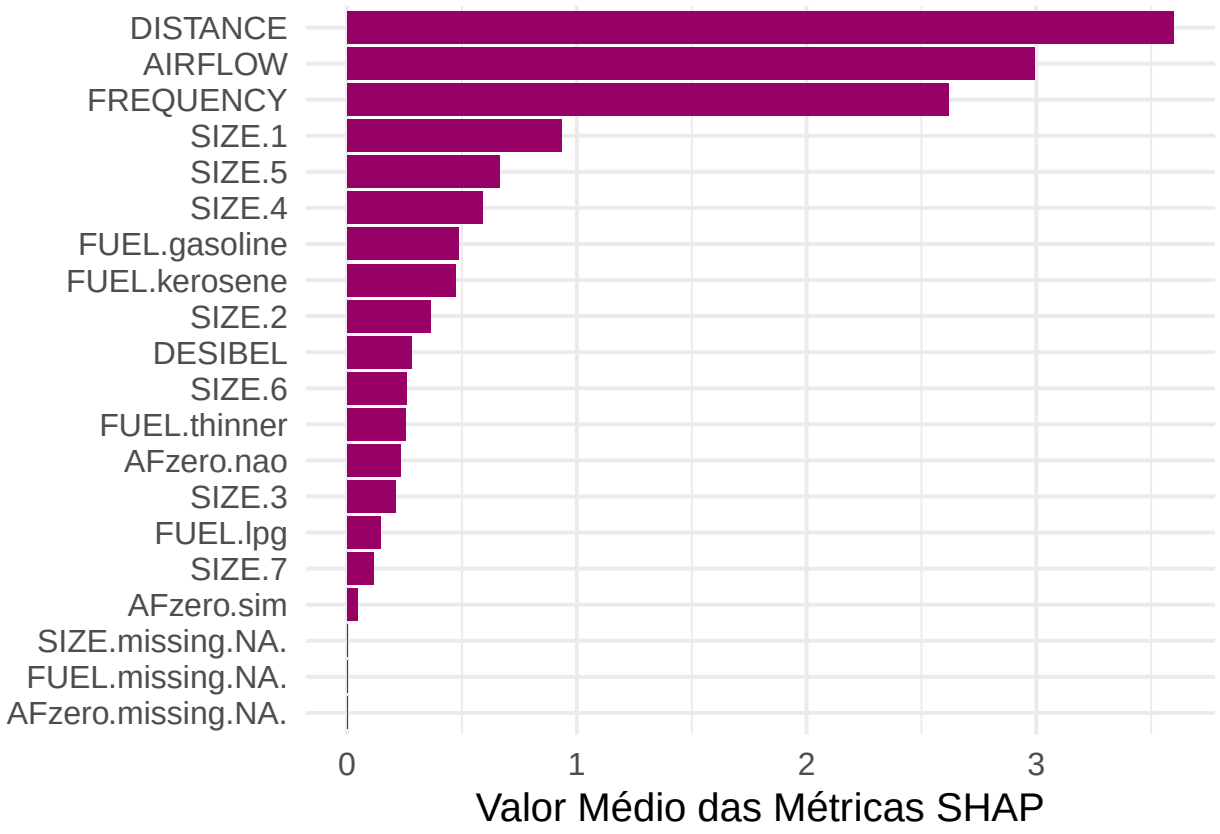
Preparando o dataframe

```
df_var_contrib <- var_contrib %>%
  as.data.frame() %>%
  select(-BiasTerm) %>%
  gather(feature, shap_value) %>%
  group_by(feature) %>%
  mutate(shap_importance = mean(abs(shap_value)), shap_force = mean(shap_value)) %>%
  ungroup()
```

Plot da importância de cada variável para prever a variável alvo

```
df_var_contrib %>%
  select(feature, shap_importance) %>%
  distinct() %>%
  ggplot(aes(x = reorder(feature,shap_importance), y = shap_importance)) +
  geom_col(fill = '#990066') +
  coord_flip() +
  xlab(NULL) +
```

```
  ylab("Valor Médio das Métricas SHAP") +
  theme_minimal(base_size = 15)
```



Verificamos assim a importância de cada variável para o modelo AutoML. Desligando o H2O