Work Sample

J. Antonio García

09/11/2020

Reduce maintenance cost through predictive techniques

Preamble

```
# html_notebook

packs <- c('lubridate', 'dplyr', 'ggplot2', 'caret', 'MLmetrics', 'gbm', 'e1071', 'LiblineaR',
           'xgboost', 'randomForest', 'doParallel')
index <- packs %in% row.names(installed.packages())
if (any(!index)){
  sapply(packs[!index], FUN=install.packages )
}
```

```
require(lubridate) # easy handle datetimes
require(dplyr) # like SQL in R, and also load pipe operator
require(ggplot2) # easy, fast  and nice plots
require(caret) # a toolbox
require(MLmetrics) # metric in one line
require(gbm) # boosting alg.
require(e1071) # numerical rutines for svm implementation
require(LiblineaR) # another implementation for RLogReg
require(xgboost) # The implementation
require(randomForest) # nice RF implementation
require(doParallel) #ugly parallel in R but useful
```

```

cl <- makePSOCKcluster(2)
registerDoParallel(cl)
```

Set up parallel enviroment

EDA

```
rm(list=ls())  # clean env.
t1 <- Sys.time()
data.raw <- read.csv(file='device_failure.csv') # few records kernels function works fine
print(sum(is.na(data.raw))) # NOT NULLS! THANKS A LOT :D
```

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

```
data.raw %>% arrange(device, date) %>% mutate(date = ymd(date) ) -> data.raw
data.raw %>% group_by(device) %>% arrange(device, date) %>%
```

```r
  mutate( l.attribute1 = lag(attribute1),
          l.attribute2 = lag(attribute2),
          l.attribute3 = lag(attribute3),
          l.attribute4 = lag(attribute4),
          l.attribute5 = lag(attribute5),
          l.attribute6 = lag(attribute6),
          l.attribute7 = lag(attribute7),
          l.attribute8 = lag(attribute8),
          l.attribute9 = lag(attribute9)) -> data.raw
data.raw <- na.omit(data.raw)
```

Each device has 0 or 1 failure, and if has a failure it's the last row

```r
n.fails.index <- which(data.raw$failure==1) #only 106 failures
nn.fails <- data.raw[ rep(n.fails.index, each=9) + -4:4, ]
head(nn.fails, 50 )
```

```
## # A tibble: 50 x 21
## # Groups:   device [11]
##    date       device failure attribute1 attribute2 attribute3 attribute4
##    <date>     <chr>    <int>      <int>      <int>      <int>      <int>
##  1 2015-01-15 S1F02~       0  222474632          0          0          1
##  2 2015-01-16 S1F02~       0  243825496          0          0          1
##  3 2015-01-17 S1F02~       0   20761856          0          0          1
##  4 2015-01-18 S1F02~       0   41291000          0          0          1
##  5 2015-01-19 S1F02~       1   64499464          0          0          1
##  6 2015-01-02 S1F02~       0   63705712          0          1          0
##  7 2015-01-03 S1F02~       0   53868456          0          1          0
##  8 2015-01-04 S1F02~       0    4263992          0          1          0
##  9 2015-01-05 S1F02~       0   37773128          0          1          0
## 10 2015-07-30 S1F03~       0    3869656        232          0          0
## # ... with 40 more rows, and 14 more variables: attribute5 <int>,
## #   attribute6 <int>, attribute7 <int>, attribute8 <int>, attribute9 <int>,
## #   l.attribute1 <int>, l.attribute2 <int>, l.attribute3 <int>,
## #   l.attribute4 <int>, l.attribute5 <int>, l.attribute6 <int>,
## #   l.attribute7 <int>, l.attribute8 <int>, l.attribute9 <int>
```

And in general, the devices present at most one fault and from which no information is recorded about them. Above all we are in a case where the variable to predict has a **strong positive bias**, more than 99% of the records are not failures, a very common case in practice …

```r
data.raw %>% filter(failure==1) %>% group_by(device) %>% summarise(n=n()) -> t
summary(t$n)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       1       1       1       1       1
```

```r
table(data.raw$failure) / dim(data.raw)[1]
```

```
##
##            0            1
## 0.9991404825 0.0008595175
```

```r
summary(data.raw)
```

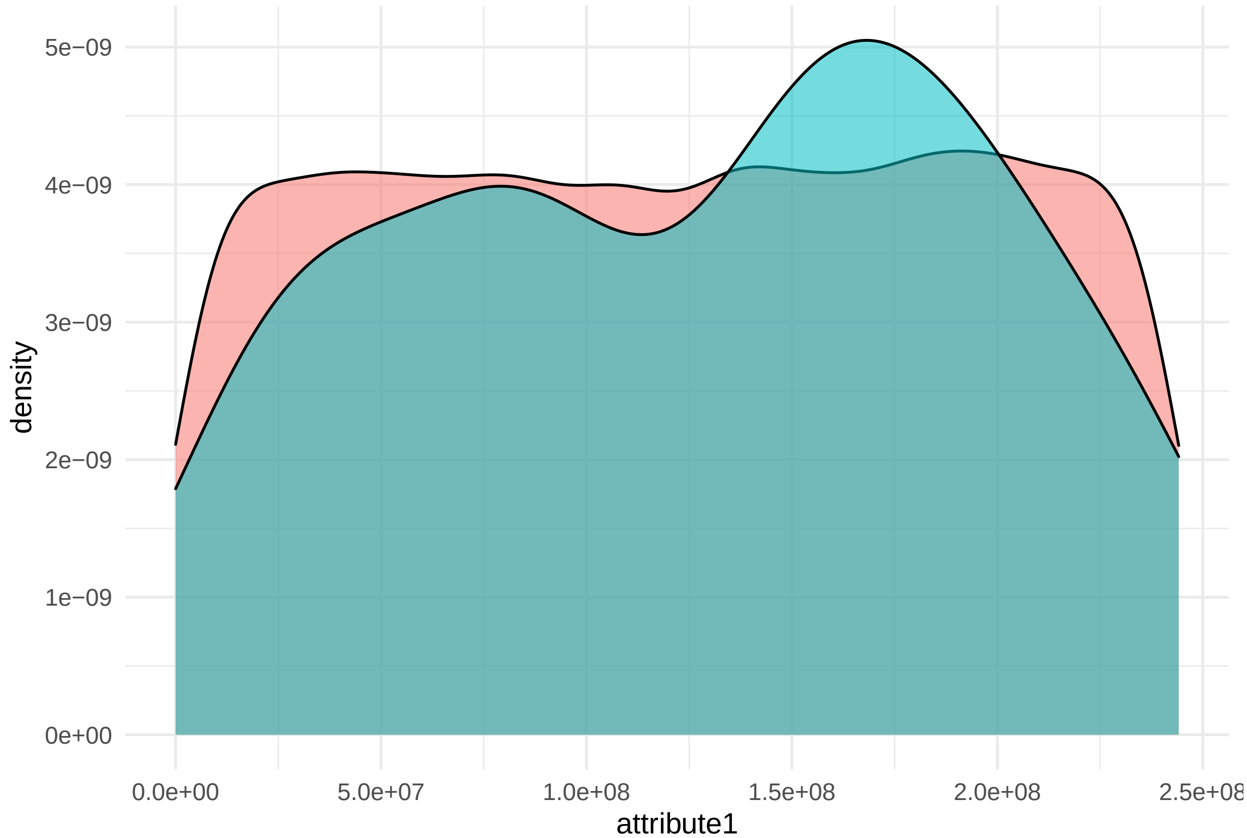
```
##       date                device             failure
##  Min.   :2015-01-02   Length:123325      Min.   :0.0000000
```

```
##  1st Qu.:2015-02-10   Class :character   1st Qu.:0.0000000
##  Median :2015-03-28   Mode  :character   Median :0.0000000
##  Mean   :2015-04-17                      Mean   :0.0008595
##  3rd Qu.:2015-06-18                      3rd Qu.:0.0000000
##  Max.   :2015-11-02                      Max.   :1.0000000
##    attribute1          attribute2        attribute3         attribute4
##  Min.   :        0   Min.   :    0.0   Min.   :    0.00   Min.   :   0.000
##  1st Qu.: 61311432   1st Qu.:    0.0   1st Qu.:    0.00   1st Qu.:   0.000
##  Median :122785544   Median :    0.0   Median :    0.00   Median :   0.000
##  Mean   :122391362   Mean   :  157.7   Mean   :    9.76   Mean   :   1.723
##  3rd Qu.:183330360   3rd Qu.:    0.0   3rd Qu.:    0.00   3rd Qu.:   0.000
##  Max.   :244140480   Max.   :64968.0   Max.   :24929.00   Max.   :1666.000
##    attribute5      attribute6       attribute7         attribute8
##  Min.   : 1.00   Min.   :     8   Min.   :  0.0000   Min.   :  0.0000
##  1st Qu.: 8.00   1st Qu.:221528   1st Qu.:  0.0000   1st Qu.:  0.0000
##  Median :10.00   Median :250060   Median :  0.0000   Median :  0.0000
##  Mean   :14.24   Mean   :260377   Mean   :  0.2892   Mean   :  0.2892
##  3rd Qu.:12.00   3rd Qu.:310396   3rd Qu.:  0.0000   3rd Qu.:  0.0000
##  Max.   :98.00   Max.   :689161   Max.   :832.0000   Max.   :832.0000
##    attribute9        l.attribute1        l.attribute2      l.attribute3
##  Min.   :    0.00   Min.   :        0   Min.   :    0.0   Min.   :    0.000
##  1st Qu.:    0.00   1st Qu.: 61298592   1st Qu.:    0.0   1st Qu.:    0.000
##  Median :    0.00   Median :122798936   Median :    0.0   Median :    0.000
##  Mean   :   12.11   Mean   :122390254   Mean   :  152.7   Mean   :    9.739
##  3rd Qu.:    0.00   3rd Qu.:183314520   3rd Qu.:    0.0   3rd Qu.:    0.000
##  Max.   :18701.00   Max.   :244140480   Max.   :64968.0   Max.   :24929.000
##   l.attribute4       l.attribute5    l.attribute6     l.attribute7
##  Min.   :   0.000   Min.   : 1.00   Min.   :     8   Min.   :  0.0000
##  1st Qu.:   0.000   1st Qu.: 8.00   1st Qu.:221462   1st Qu.:  0.0000
##  Median :   0.000   Median :10.00   Median :249721   Median :  0.0000
##  Mean   :   1.665   Mean   :14.24   Mean   :260080   Mean   :  0.2532
##  3rd Qu.:   0.000   3rd Qu.:12.00   3rd Qu.:310207   3rd Qu.:  0.0000
##  Max.   :1666.000   Max.   :98.00   Max.   :689062   Max.   :832.0000
##   l.attribute8       l.attribute9
##  Min.   :  0.0000   Min.   :    0.0
##  1st Qu.:  0.0000   1st Qu.:    0.0
##  Median :  0.0000   Median :    0.0
##  Mean   :  0.2532   Mean   :   12.1
##  3rd Qu.:  0.0000   3rd Qu.:    0.0
##  Max.   :832.0000   Max.   :18701.0
```

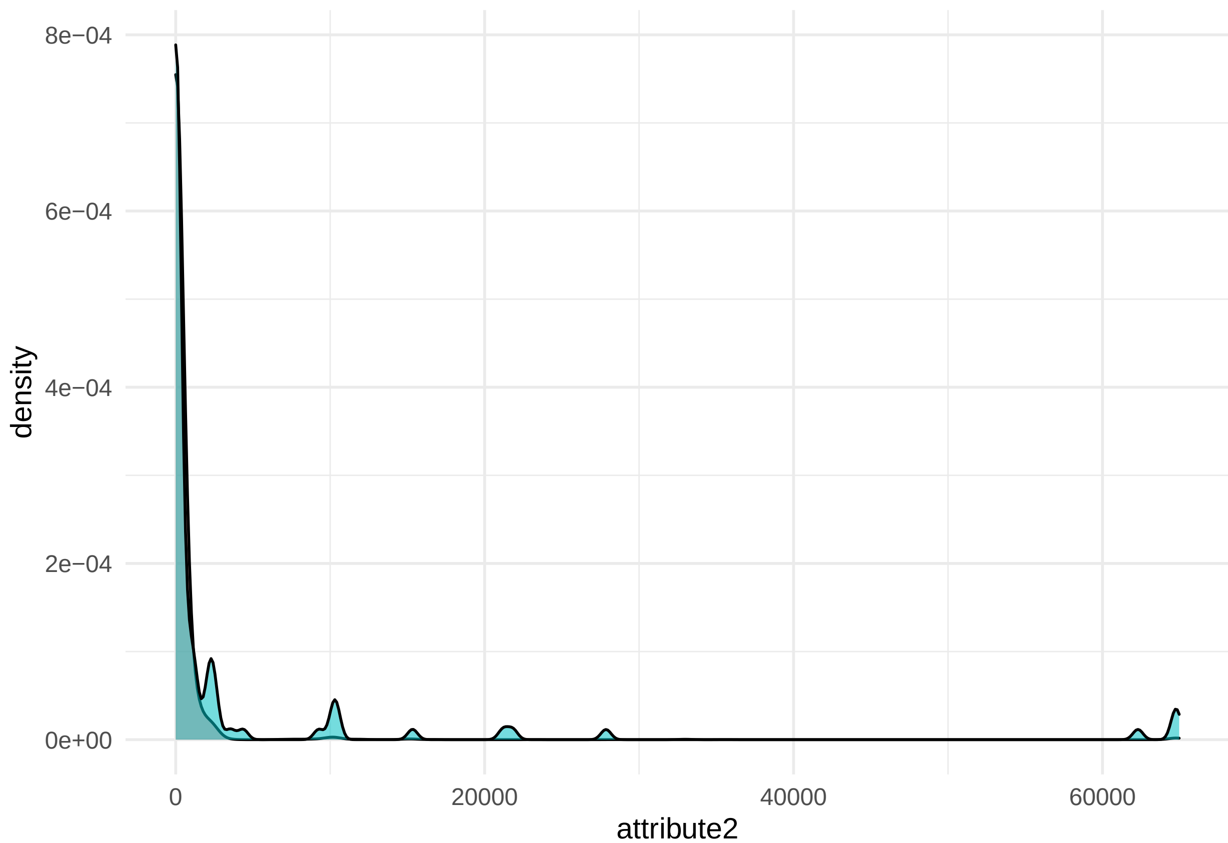
Focus on success stories

So we decided to focus on success stories to infer from them insights that allow us to carry out the task.

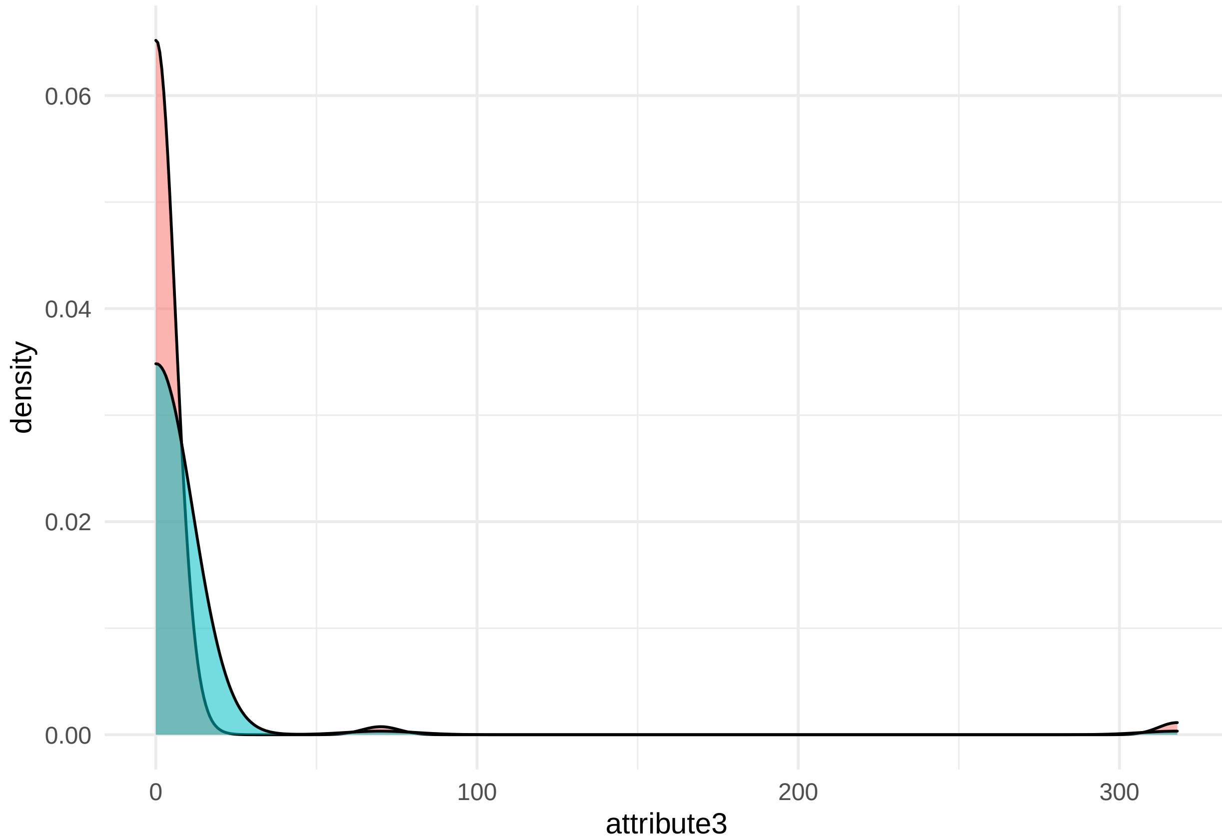
```
device.with.failures <- unique(data.raw$device[n.fails.index] )
data.sample <- data.raw[ data.raw$device %in% device.with.failures, ]
ggplot(data.sample, aes(attribute1, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() + theme(legend.position="none")
```



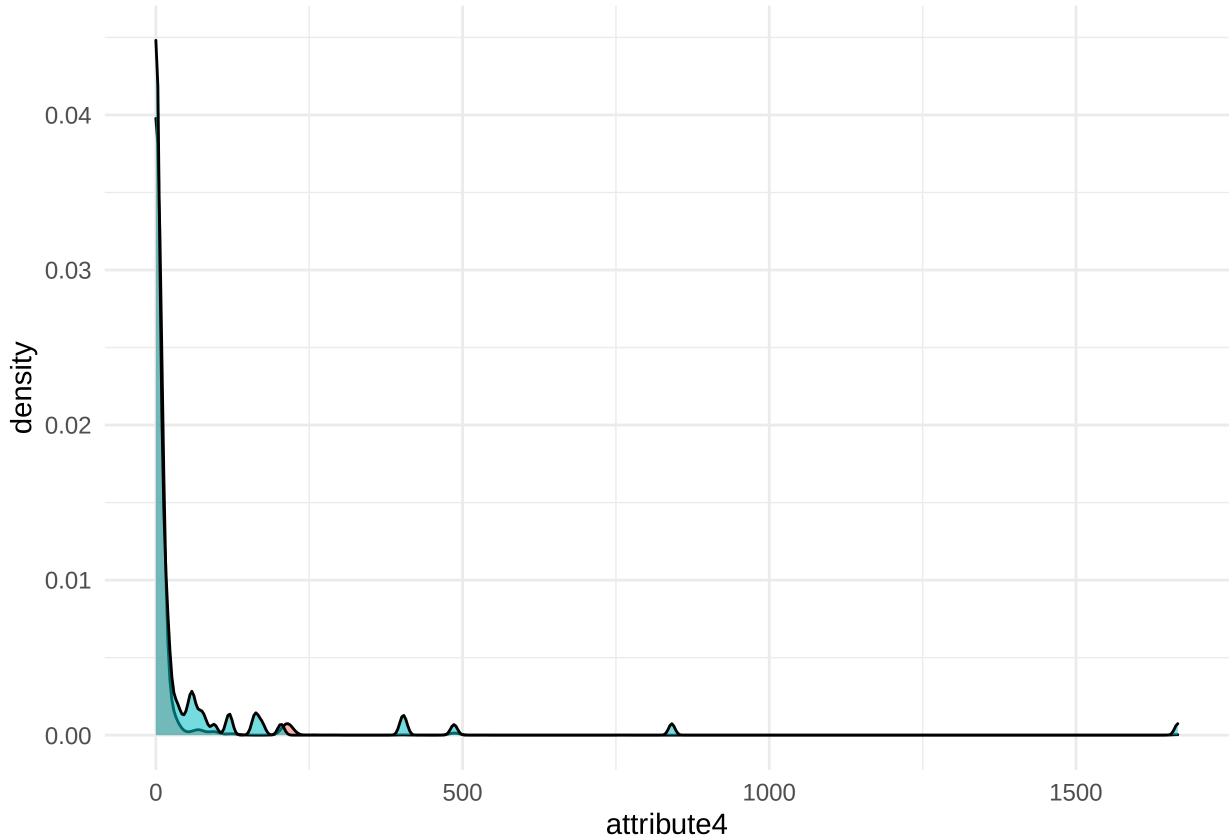

```
ggplot(data.sample, aes(attribute2, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



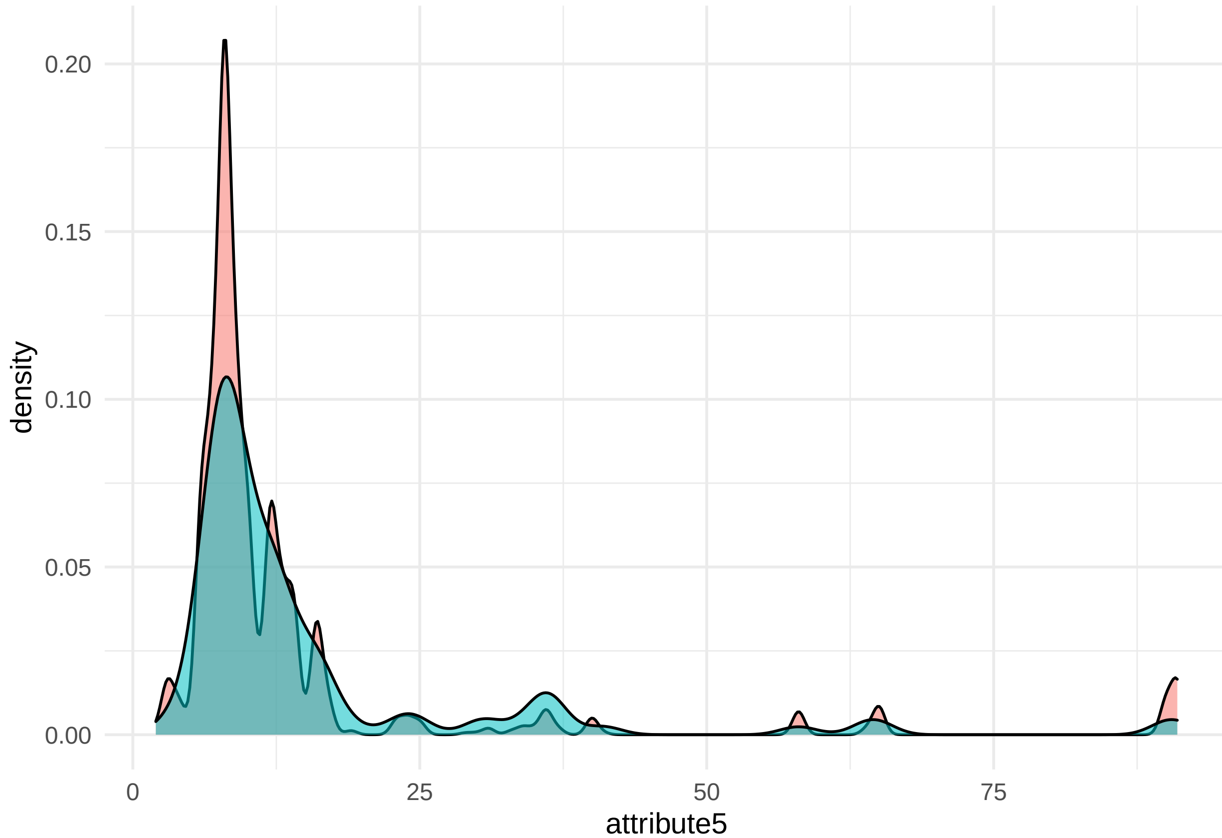

```
ggplot(data.sample, aes(attribute3, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



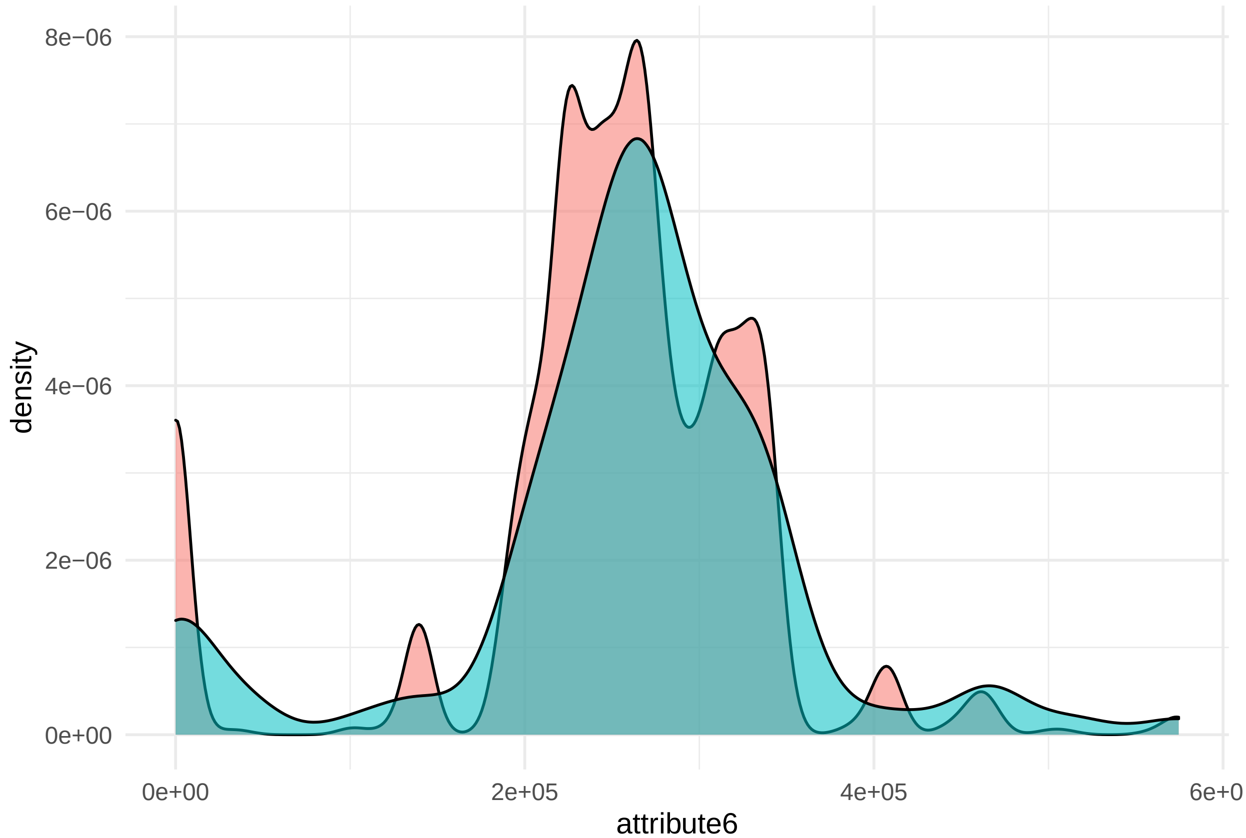

```
ggplot(data.sample, aes(attribute4, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



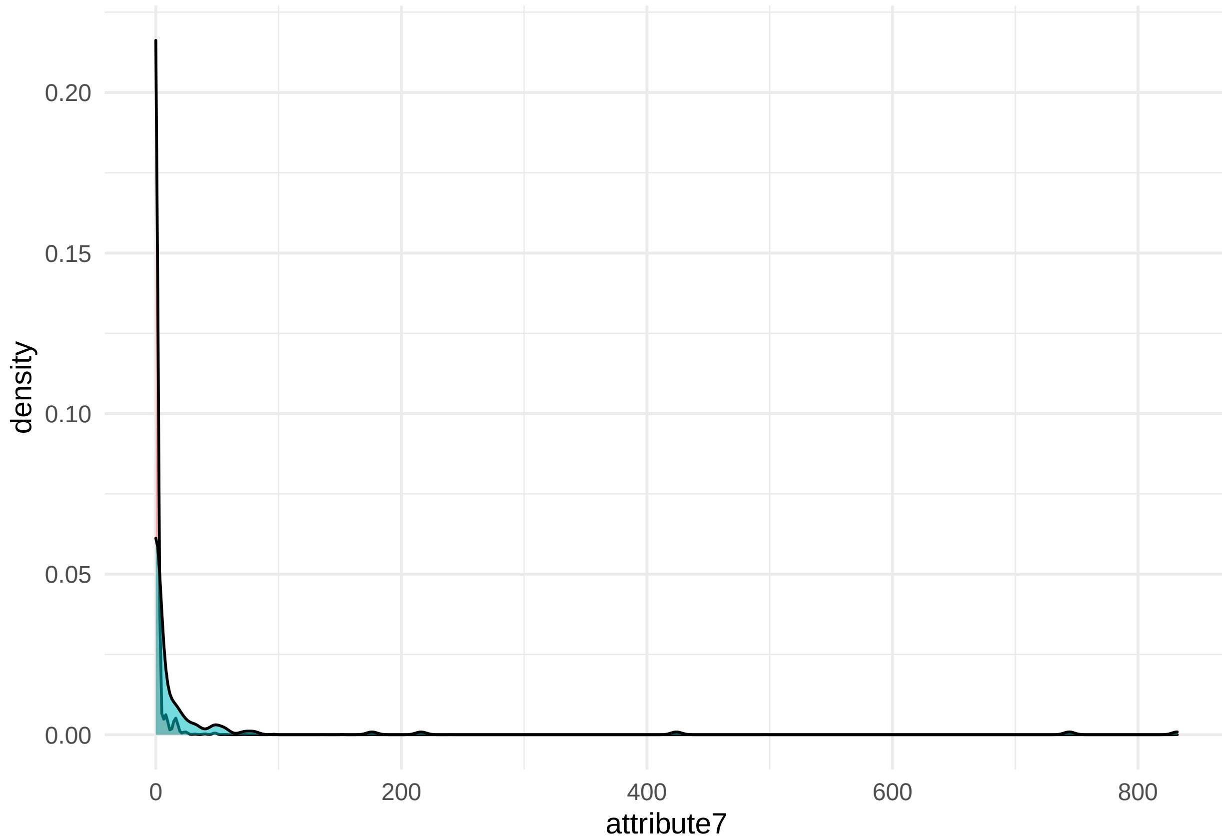

```
ggplot(data.sample, aes(attribute5, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



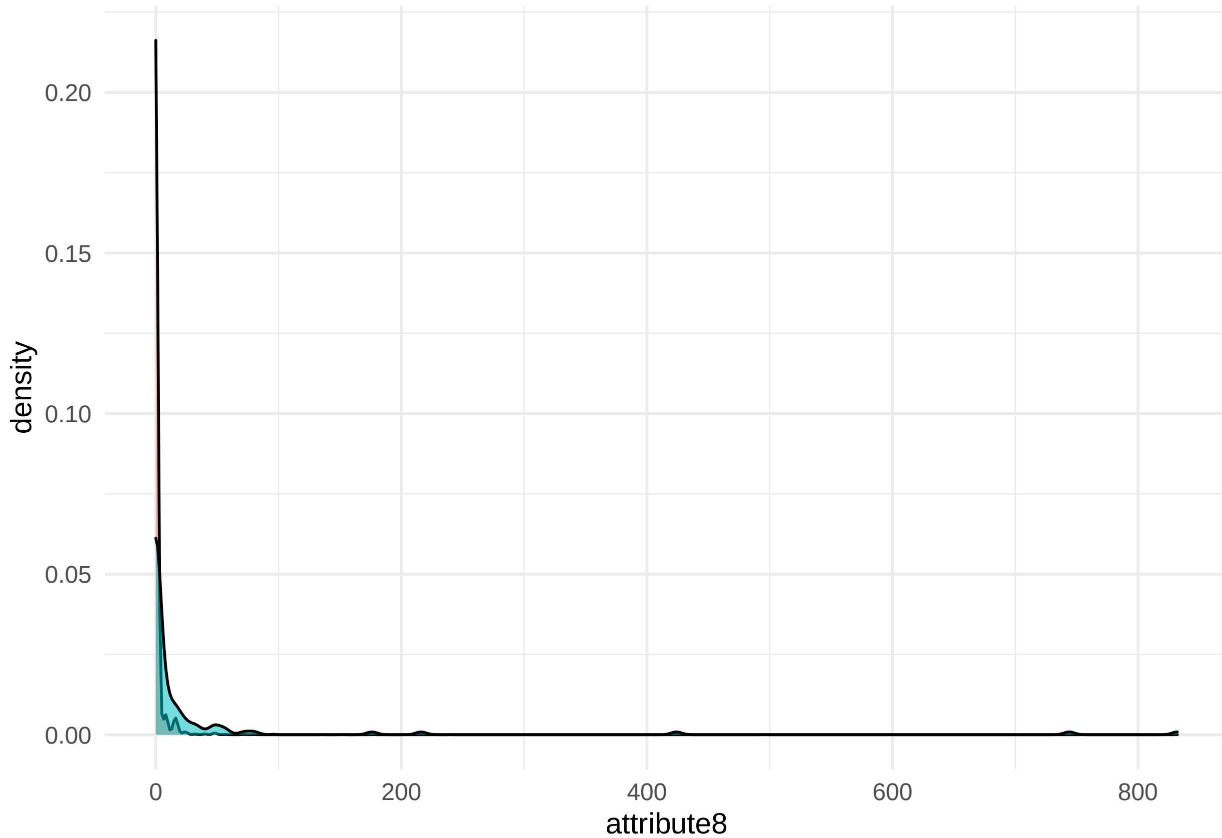

```
ggplot(data.sample, aes(attribute6, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



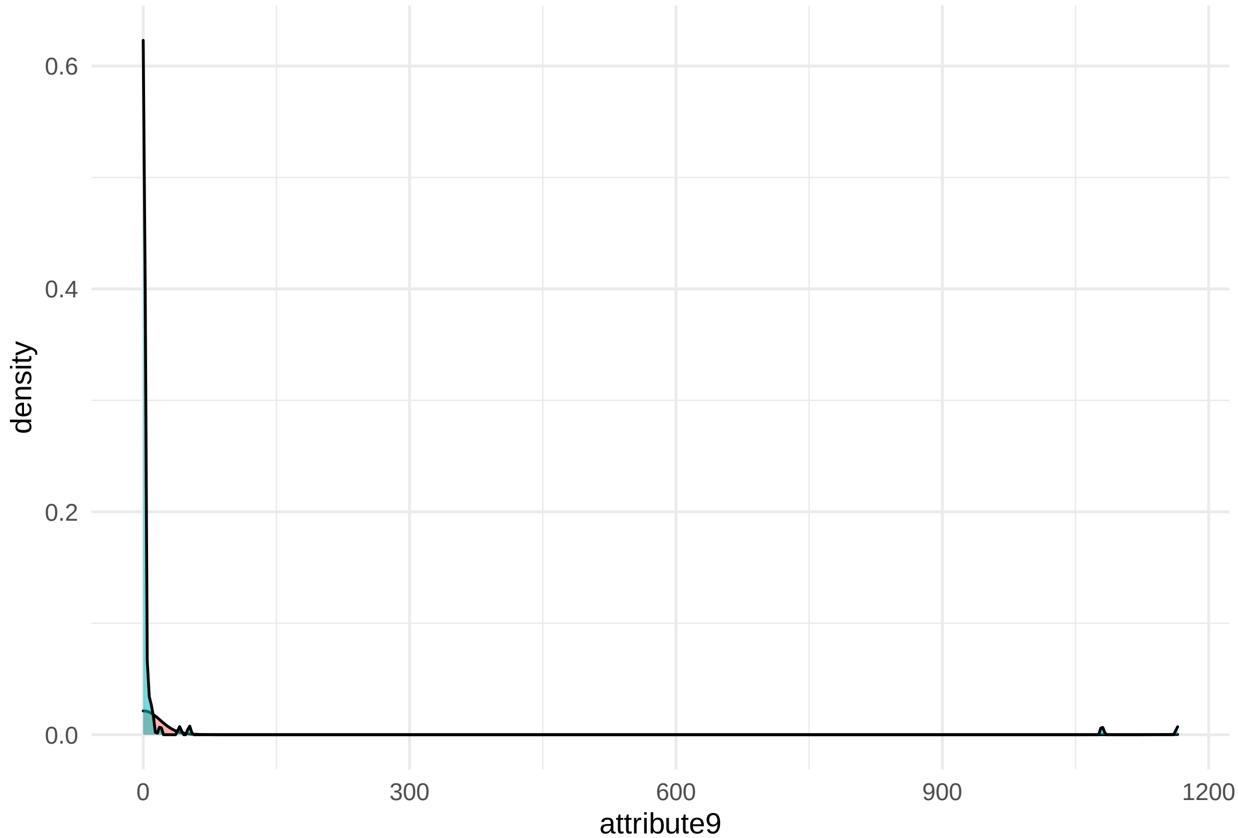

```
ggplot(data.sample, aes(attribute7, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```




```
ggplot(data.sample, aes(attribute8, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```




```r
ggplot(data.sample, aes(attribute9, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



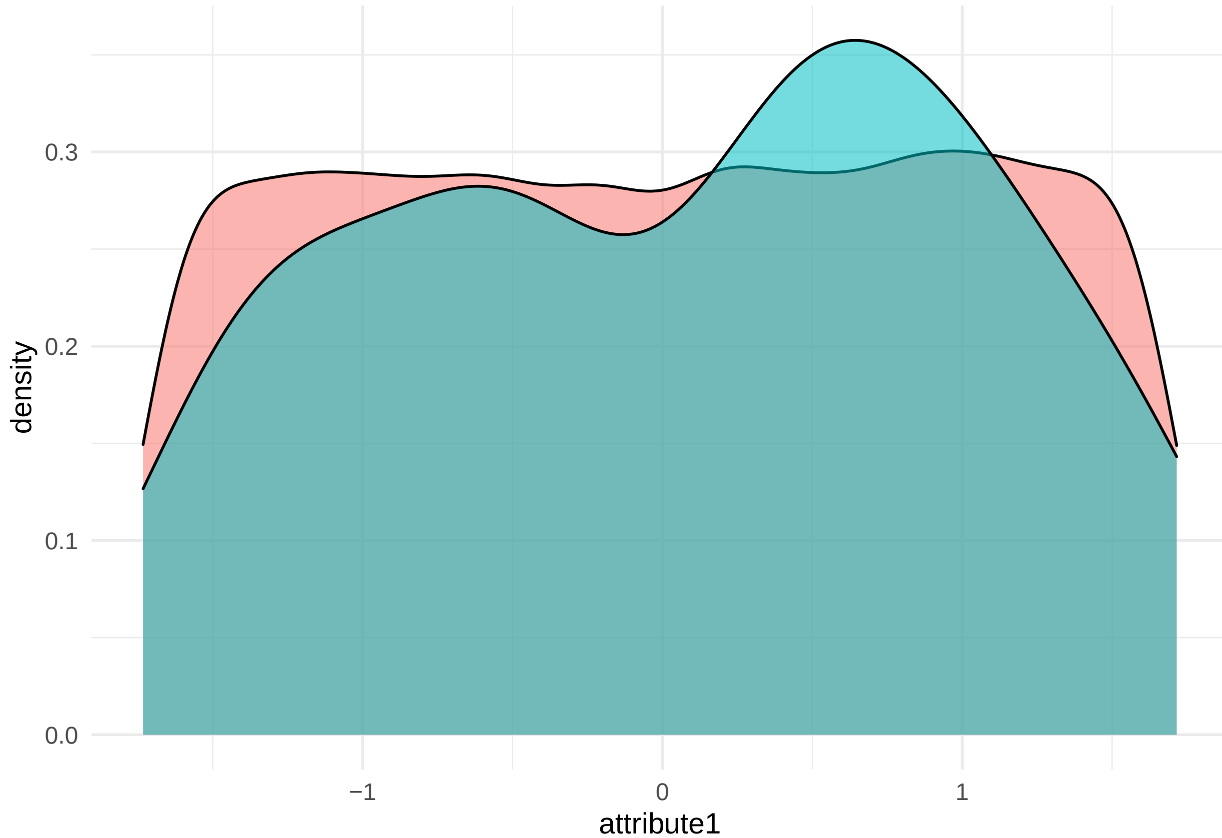
Due to the distribution of some of the variables, we apply a non-linear transformation that allows us to more easily discriminate between failures and non-failures.

```
index.columns <- c(2, 3, 4, 7, 8, 9:18) + 3
# log features selected
data.sample[, names(data.sample)[index.columns]] <-
  log(data.sample[, names(data.sample)[index.columns]] + 1 )
# standar features
index.columns <- grep('attr', names(data.sample))
summary(data.sample)
```

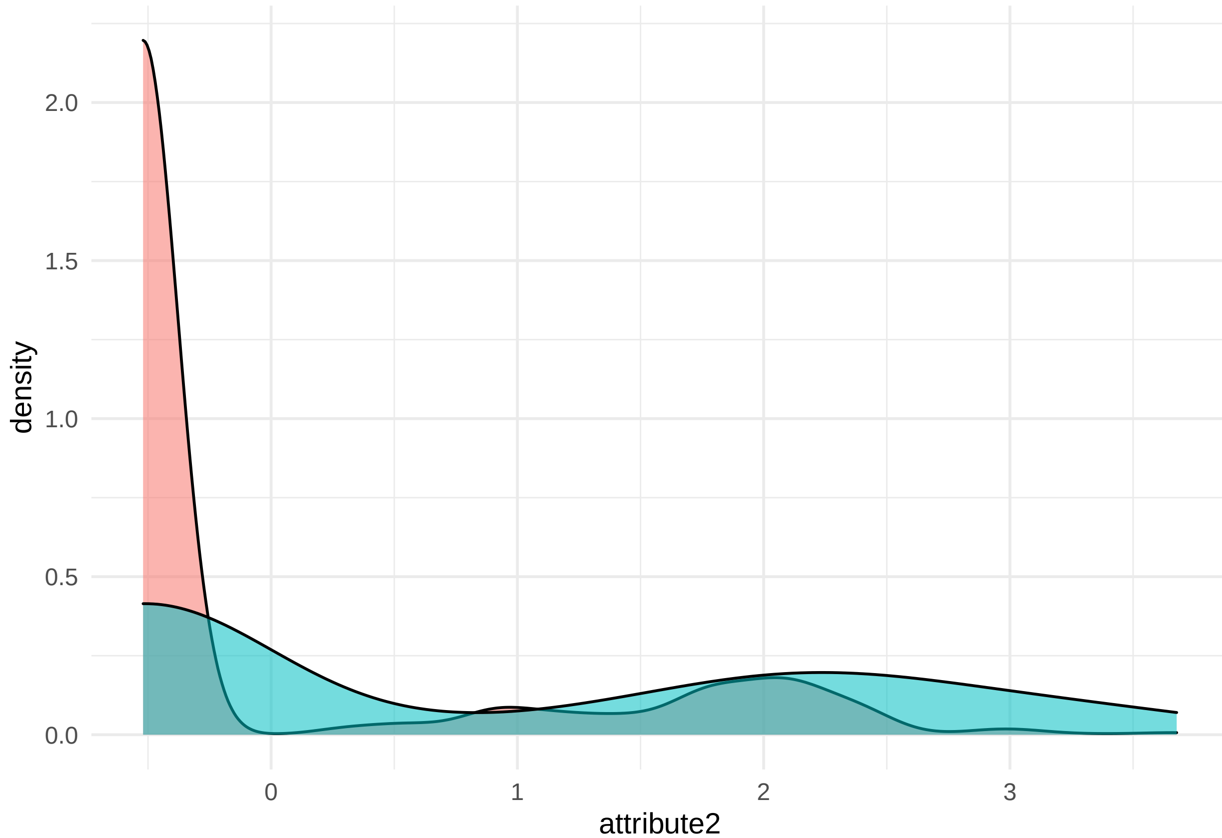
```
##       date               device             failure           attribute1
##  Min.   :2015-01-02   Length:10607       Min.   :0.000000   Min.   :     4224
##  1st Qu.:2015-01-31   Class :character   1st Qu.:0.000000   1st Qu.: 61045068
##  Median :2015-03-12   Mode  :character   Median :0.000000   Median :123450384
##  Mean   :2015-03-22                      Mean   :0.009993   Mean   :122672510
##  3rd Qu.:2015-05-02                      3rd Qu.:0.000000   3rd Qu.:184160428
##  Max.   :2015-10-26                      Max.   :1.000000   Max.   :244135688
##    attribute2        attribute3        attribute4        attribute5
##  Min.   : 0.000   Min.   :0.0000   Min.   :0.0000   Min.   : 2.0
##  1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 8.0
##  Median : 0.000   Median :0.0000   Median :0.0000   Median : 9.0
##  Mean   : 1.373   Mean   :0.2365   Mean   :0.5074   Mean   :14.2
##  3rd Qu.: 0.000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:13.0
##  Max.   :11.082   Max.   :5.7652   Max.   :7.4188   Max.   :91.0
##    attribute6        attribute7        attribute8        attribute9
##  Min.   :    19   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
```

```
##  1st Qu.:222560   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :256236   Median :0.0000   Median :0.0000   Median :0.0000  
##  Mean   :248158   Mean   :0.1895   Mean   :0.1895   Mean   :0.4359  
##  3rd Qu.:299202   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
##  Max.   :574599   Max.   :6.7250   Max.   :6.7250   Max.   :7.0613  
##   l.attribute1     l.attribute2     l.attribute3     l.attribute4   
##  Min.   : 8.349   Min.   : 0.000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:17.925   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :18.629   Median : 0.000   Median :0.0000   Median :0.0000  
##  Mean   :18.314   Mean   : 1.353   Mean   :0.2365   Mean   :0.4941  
##  3rd Qu.:19.031   3rd Qu.: 0.000   3rd Qu.:0.0000   3rd Qu.:0.0000  
##  Max.   :19.313   Max.   :11.082   Max.   :5.7652   Max.   :7.4188  
##   l.attribute5    l.attribute6     l.attribute7     l.attribute8   
##  Min.   :1.099   Min.   : 2.996   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:2.197   1st Qu.:12.312   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :2.303   Median :12.453   Median :0.0000   Median :0.0000  
##  Mean   :2.458   Mean   :11.833   Mean   :0.1794   Mean   :0.1794  
##  3rd Qu.:2.639   3rd Qu.:12.609   3rd Qu.:0.0000   3rd Qu.:0.0000  
##  Max.   :4.522   Max.   :13.261   Max.   :6.7250   Max.   :6.7250  
##   l.attribute9  
##  Min.   :0.000  
##  1st Qu.:0.000  
##  Median :0.000  
##  Mean   :0.435  
##  3rd Qu.:0.000  
##  Max.   :7.061
```

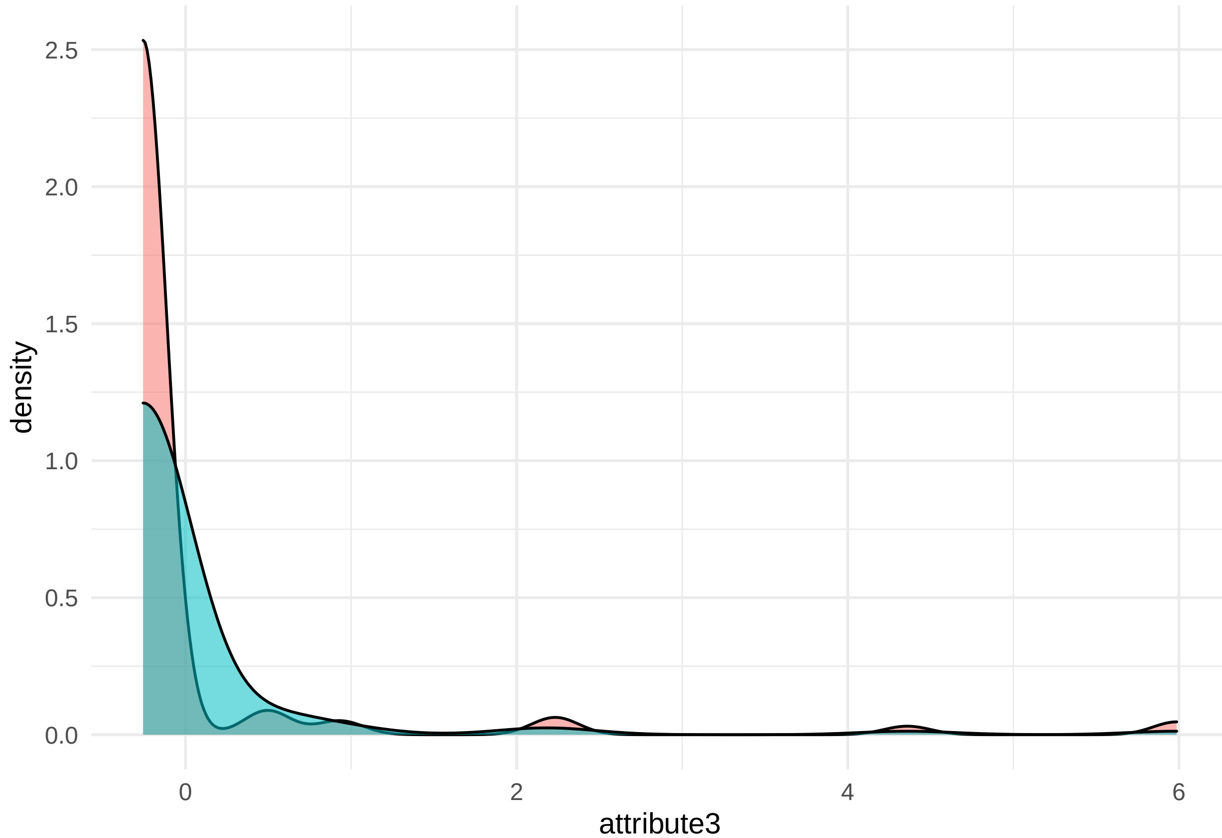
```r
for ( i in index.columns ){
  temp <- data.sample[, names(data.sample)[i]]
  data.sample[, names(data.sample)[i]] <- scale(temp)
}
ggplot(data.sample, aes(attribute1, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() + theme(legend.position="none")
```



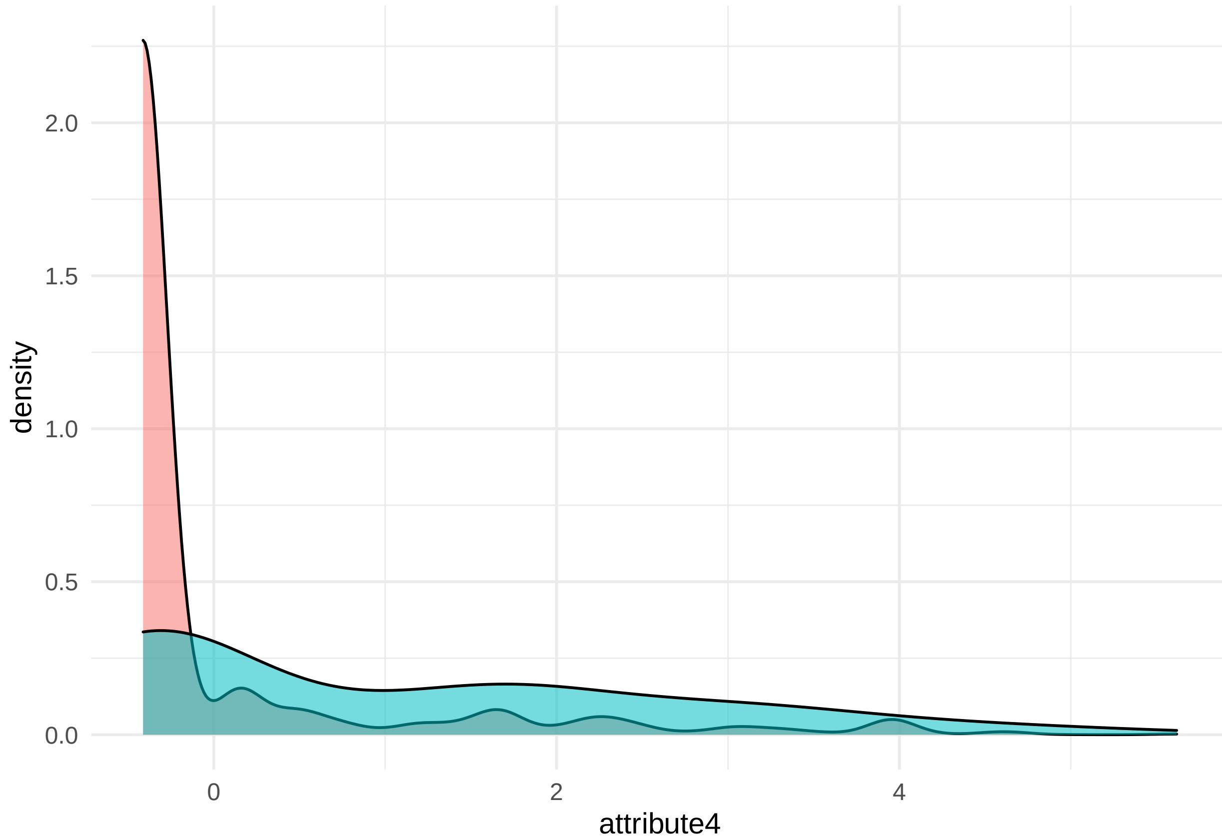

```
ggplot(data.sample, aes(attribute2, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



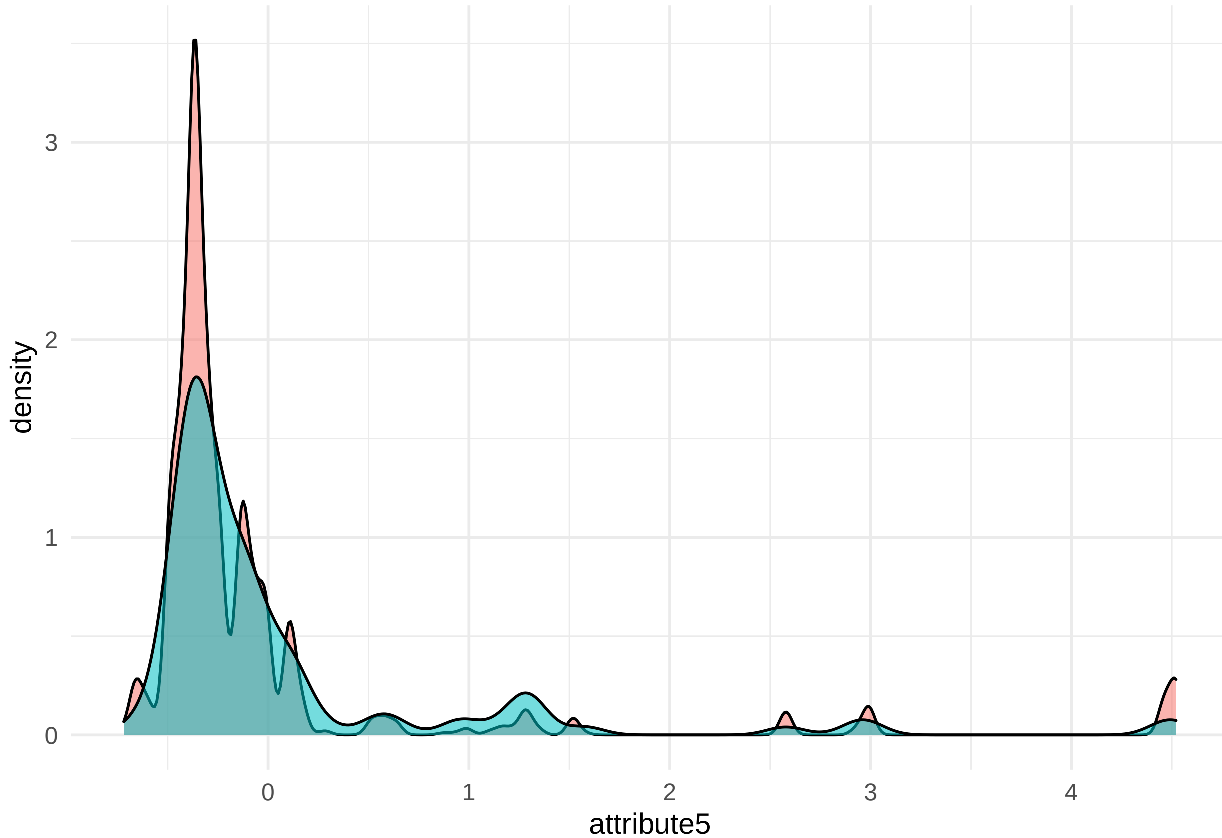

```
ggplot(data.sample, aes(attribute3, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



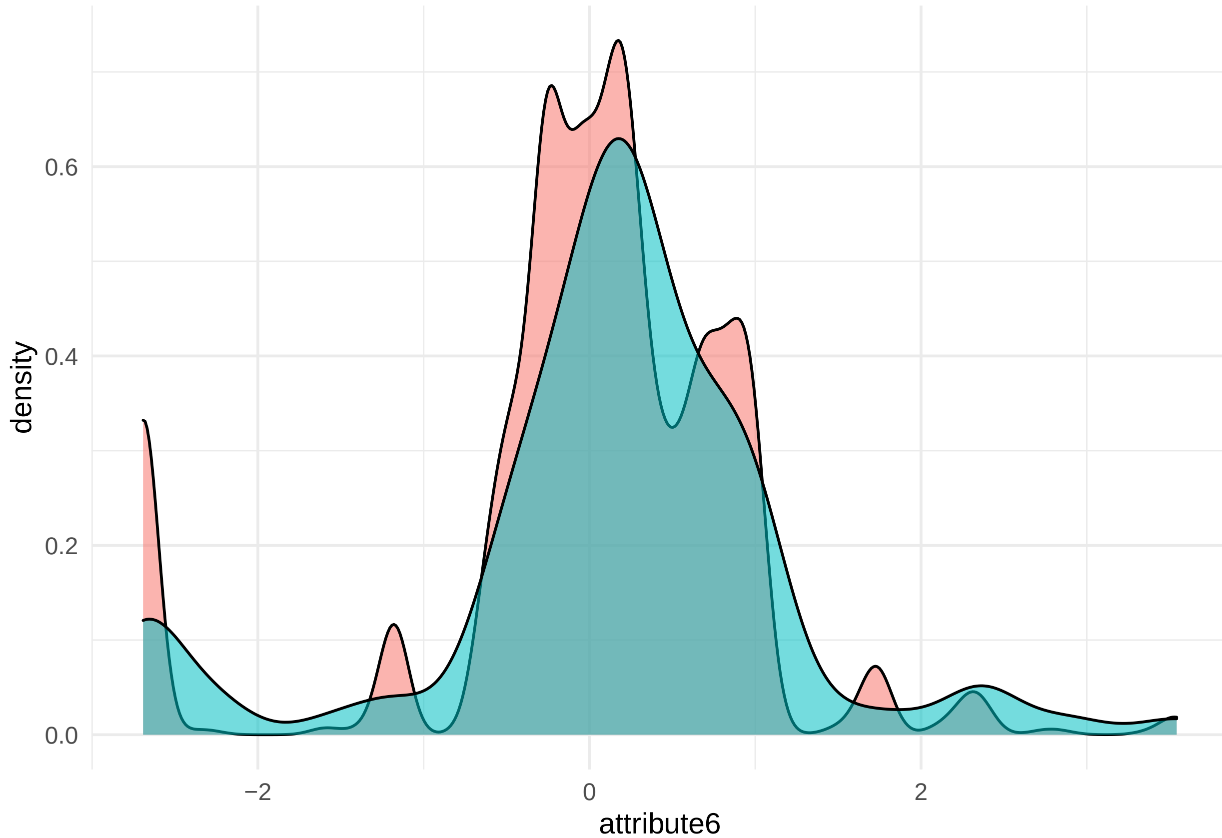

```r
ggplot(data.sample, aes(attribute4, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



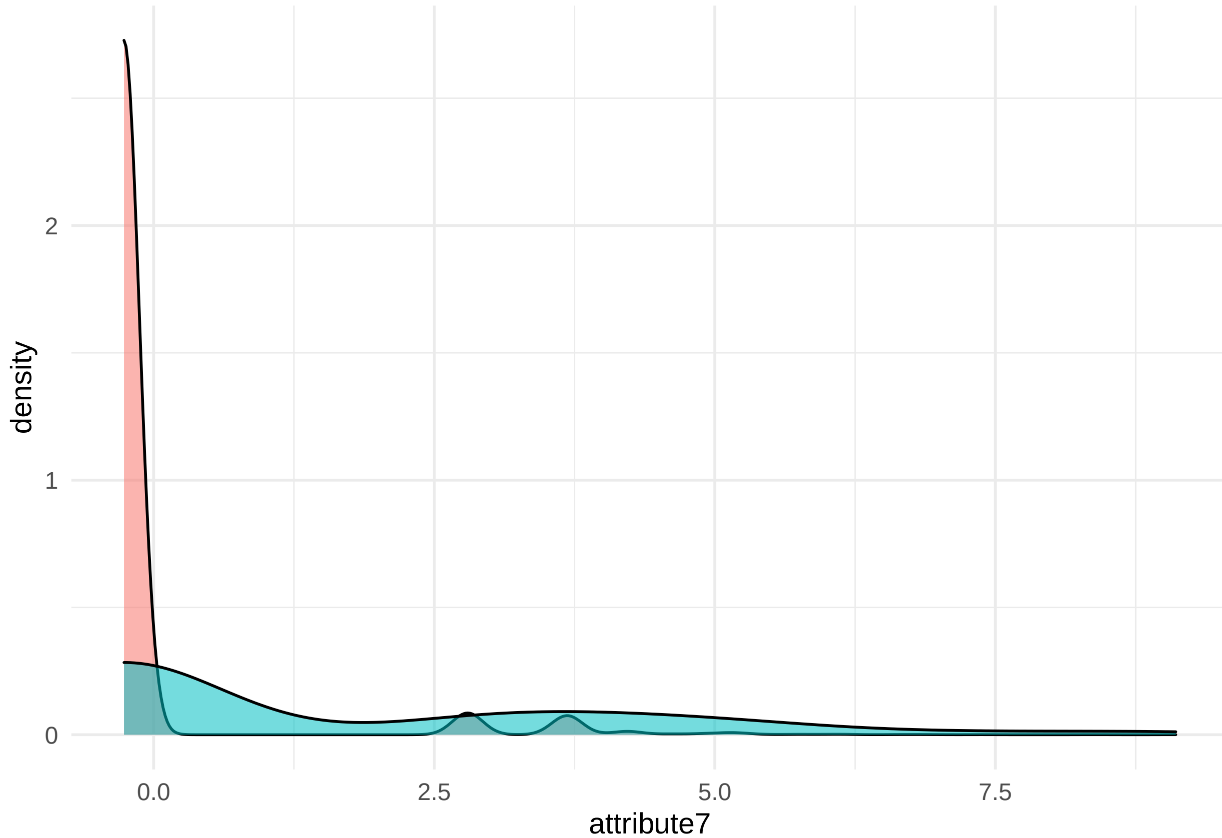

```r
ggplot(data.sample, aes(attribute5, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



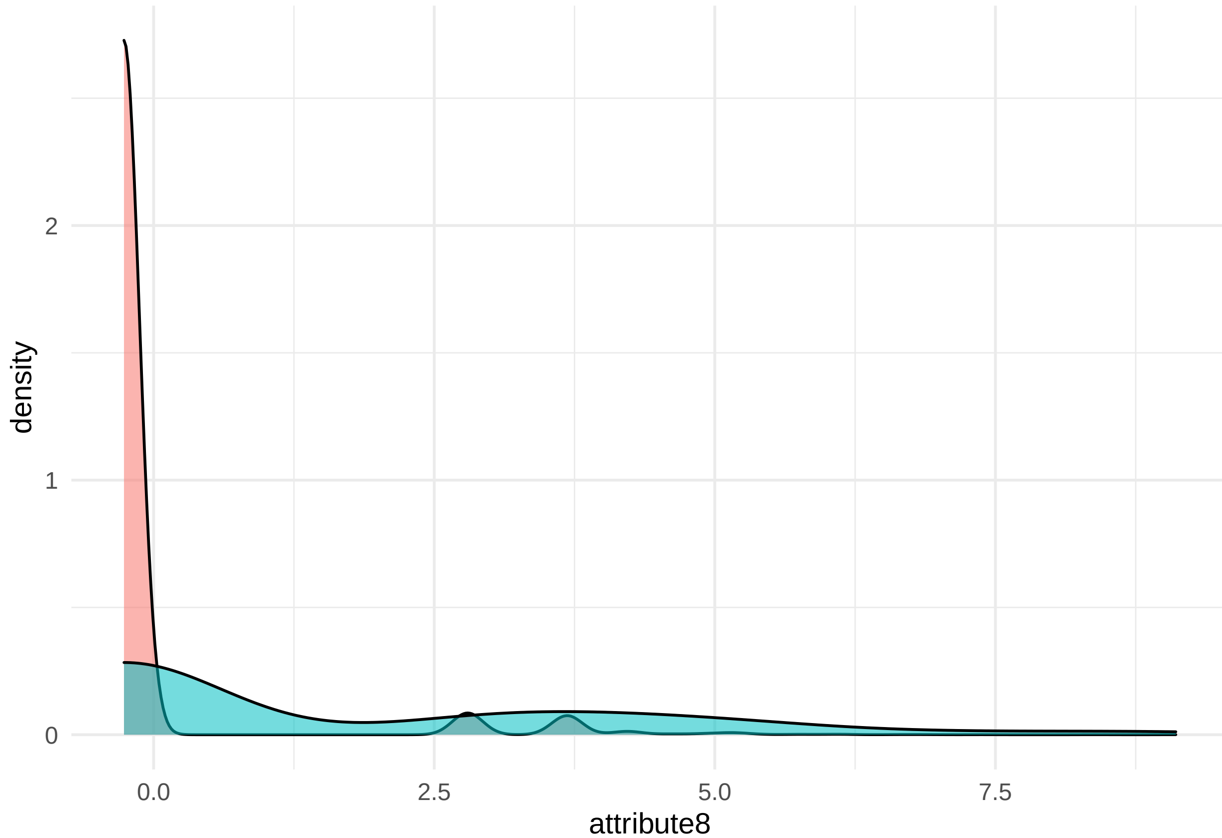

```r
ggplot(data.sample, aes(attribute6, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



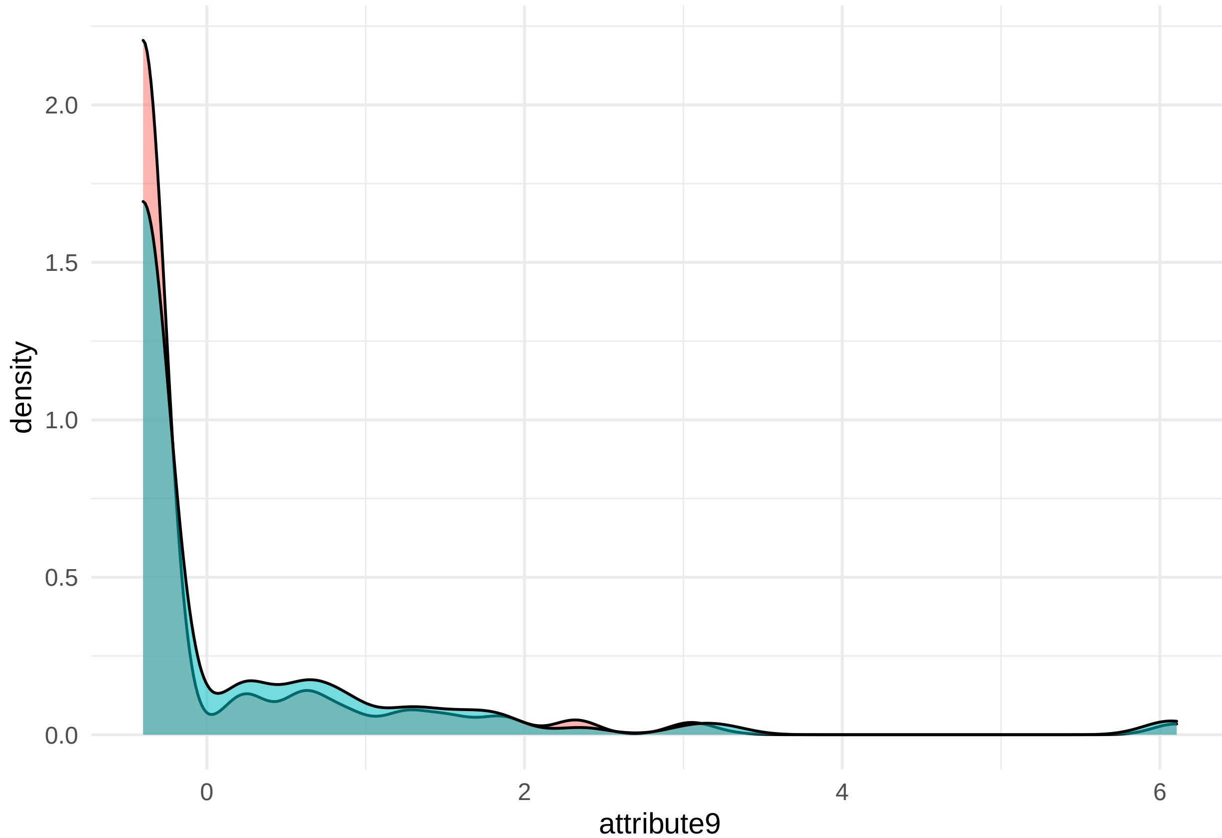

```
ggplot(data.sample, aes(attribute7, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```




```
ggplot(data.sample, aes(attribute8, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```




```
ggplot(data.sample, aes(attribute9, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



After we divided our sample to continue with a preselection of models, among the enormous variety of algorithms and implementations that exist, we decided to report 4, because they are interpretable models and easy to explain to non-specialized people.

Since we are interested in keeping the number of false negatives and false positives low, we opted for the F_1 metric to measure the performance of the algorithms.

```
createPartition <- function(data_, p=0.7){
  # Inputs: data_ (data.frame) to split
  #         p (numeric): dataframe's proportion for train sample
  t <- unique(data_$device)
  n <- length(t)
  n.p <- round(n*p, 0)
  t.sample <- sample(t, n.p)
  train.index <- which( data_$device %in% t.sample)
  return(train.index)
}

f1 <- function (data, lev = NULL, model = NULL) {
  # Function requiere to calculate F1 score within caret::train , see doc.
  precision <- posPredValue(data$pred, data$obs, positive = "Failure")
  recall  <- sensitivity(data$pred, data$obs, positive = "Failure")
  f1_val <- (2 * precision * recall) / (precision + recall)
  names(f1_val) <- c("F1")
  return(f1_val)
}
```

```r
set.seed(0)
data.sample$failure <- factor(data.sample$failure)
levels(data.sample$failure) <- c('NoFailure', 'Failure')
train.index <- createPartition(data.sample)
data.sample$date <- data.sample$device <- NULL
train <- data.sample[train.index, ]
test <- data.sample[-train.index, ]
fit.control <- trainControl(  method = 'repeatedcv', number = 10, repeats = 3,
                             allowParallel = TRUE, classProbs = TRUE,
                             summaryFunction = f1,  sampling =  "down")

set.seed(0)
gbmFit1 <- train(failure ~ ., data = train, method = "gbm", trControl = fit.control,
                 verbose = FALSE)
xgb.Fit1 <- train(failure ~ ., data = train, method = "xgbTree", #tuneLength = 5, search= 'random',
                  trControl = fit.control,
                  verbose = FALSE)
rf.Fit1 <- train(failure ~ ., data = train, method = "rf", trControl = fit.control,
                 verbose = FALSE)
rlg.Fit1 <- train(failure ~ ., data = train, method = "regLogistic",
                  trControl = fit.control, verbose = FALSE)
```

The test based on the Bonferroni intervals strongly suggests that XGB and RF perform better than the other methods, however, when evaluating it on the data test , we opted to only report RF's tunning results:

```r
resamps <- resamples(list(GBM = gbmFit1, XGB = xgb.Fit1,
                          RF = rf.Fit1, RLG=rlg.Fit1  ))
summary(resamps)
```

```
##
## Call:
## summary.resamples(object = resamps)
##
## Models: GBM, XGB, RF, RLG
## Number of resamples: 30
##
## F1
##           Min.    1st Qu.     Median       Mean    3rd Qu.       Max. NA's
## GBM 0.01652893 0.03516260 0.04480287 0.04568529 0.05143872 0.07526882    0
## XGB 0.02419355 0.03330070 0.04371180 0.04614946 0.05542813 0.10687023    0
## RF  0.01183432 0.04102891 0.05163225 0.05286921 0.06431066 0.09600000    0
## RLG 0.02564103 0.04301242 0.05266805 0.05399095 0.05780347 0.09756098    0
```

```r
summary(diff(resamps))
```

```
##
## Call:
## summary.diff.resamples(object = diff(resamps))
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## F1
##     GBM    XGB        RF         RLG
```

```
## GBM        -0.0004642 -0.0071839 -0.0083057
## XGB 1.0000            -0.0067198 -0.0078415
## RF  0.6181 0.6889                -0.0011217
## RLG 0.1565 0.8533     1.0000
```

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 2.016175 mins
```

```r
confusionMatrix(predict(rf.Fit1$finalModel,test), test$failure)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2349      10
##   Failure         876      22
##
##                Accuracy : 0.728
##                  95% CI : (0.7123, 0.7432)
##     No Information Rate : 0.9902
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0289
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7284
##             Specificity : 0.6875
##          Pos Pred Value : 0.9958
##          Neg Pred Value : 0.0245
##              Prevalence : 0.9902
##          Detection Rate : 0.7212
##    Detection Prevalence : 0.7243
##       Balanced Accuracy : 0.7079
##
##        'Positive' Class : NoFailure
##
```

```r
confusionMatrix(predict(xgb.Fit1,test), test$failure)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2623      11
##   Failure         602      21
##
##                Accuracy : 0.8118
##                  95% CI : (0.7979, 0.8251)
##     No Information Rate : 0.9902
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0463
##
```

```
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.81333
##             Specificity : 0.65625
##          Pos Pred Value : 0.99582
##          Neg Pred Value : 0.03371
##              Prevalence : 0.99018
##          Detection Rate : 0.80534
##    Detection Prevalence : 0.80872
##       Balanced Accuracy : 0.73479
##
##        'Positive' Class : NoFailure
##
```

```
t3 <- Sys.time()
set.seed(0)
tune_grid <- expand.grid(nrounds=c(100,300), max_depth = c(4:7), eta = c(0.05, 1),   gamma = c(0.01),
                         colsample_bytree = c(0.75), subsample = c(0.50),  min_child_weight = c(0))

xgb_fit <- train(failure ~., data = train, method = "xgbTree",
                trControl= fit.control,
                tuneGrid = tune_grid,
                tuneLength = 10)
tune_grid <- expand.grid(.mtry = (1:16))
rf_fit <- train(failure ~., data = train, method = "rf",
                trControl= fit.control,
                tuneGrid = tune_grid,
                tuneLength = 10)

t4 <- Sys.time()
t4 - t1
```

```
## Time difference of 3.655641 mins
```

```
confusionMatrix(predict(rf_fit$finalModel, test), test$failure)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2534      14
##   Failure         691      18
##
##                Accuracy : 0.7835
##                  95% CI : (0.769, 0.7976)
##     No Information Rate : 0.9902
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0304
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.78574
##             Specificity : 0.56250
##          Pos Pred Value : 0.99451
##          Neg Pred Value : 0.02539
```

```
##               Prevalence : 0.99018
##           Detection Rate : 0.77802
##     Detection Prevalence : 0.78232
##        Balanced Accuracy : 0.67412
##
##         'Positive' Class : NoFailure
##
```

```r
confusionMatrix(predict(xgb_fit, test), test$failure)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  NoFailure Failure
##   NoFailure      2048       5
##   Failure        1177      27
##
##                Accuracy : 0.6371
##                  95% CI : (0.6203, 0.6536)
##     No Information Rate : 0.9902
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.025
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.63504
##             Specificity : 0.84375
##          Pos Pred Value : 0.99756
##          Neg Pred Value : 0.02243
##              Prevalence : 0.99018
##          Detection Rate : 0.62880
##    Detection Prevalence : 0.63033
##       Balanced Accuracy : 0.73939
##
##        'Positive' Class : NoFailure
##
```