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
```

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

```
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 )
```

```
##            date   device failure attribute1 attribute2 attribute3 attribute4
## 422  2015-01-15 S1F023H2       0  222474632          0          0          1
## 423  2015-01-16 S1F023H2       0  243825496          0          0          1
## 424  2015-01-17 S1F023H2       0   20761856          0          0          1
## 425  2015-01-18 S1F023H2       0   41291000          0          0          1
## 426  2015-01-19 S1F023H2       1   64499464          0          0          1
## 427  2015-01-01 S1F02A0J       0    8217840          0          1          0
## 428  2015-01-02 S1F02A0J       0   63705712          0          1          0
## 429  2015-01-03 S1F02A0J       0   53868456          0          1          0
## 430  2015-01-04 S1F02A0J       0    4263992          0          1          0
## 1045 2015-07-30 S1F03YZM       0    3869656        232          0          0
## 1046 2015-07-31 S1F03YZM       0   29417000        240          0          0
## 1047 2015-08-01 S1F03YZM       0   56266776        240          0          0
## 1048 2015-08-02 S1F03YZM       0   85363816        240          0          0
## 1049 2015-08-03 S1F03YZM       1  110199904        240          0          0
## 1050 2015-01-01 S1F044ET       0  161730848          0          0          0
## 1051 2015-01-02 S1F044ET       0  181377384          0          0          0
## 1052 2015-01-03 S1F044ET       0  196733608          0          0          0
## 1053 2015-01-04 S1F044ET       0  211125168          0          0          0
## 1712 2015-07-14 S1F09DZQ       0     226184       2016          0          3
## 1713 2015-07-15 S1F09DZQ       0   19472584       2024          0          3
## 1714 2015-07-16 S1F09DZQ       0   39413344       2048          0          3
## 1715 2015-07-17 S1F09DZQ       0   57504848       2144          0          3
## 1716 2015-07-18 S1F09DZQ       1   77351504       2304          0          3
## 1717 2015-01-01 S1F09MAK       0    9461552       7928          0          7
## 1718 2015-01-02 S1F09MAK       0   31592888       7944          0          7
## 1719 2015-01-03 S1F09MAK       0   52282144       8392          0          7
## 1720 2015-01-04 S1F09MAK       0   74134944       8392          0          7
## 3196 2015-01-03 S1F0CTDN       0   91492168        528          0          4
## 3197 2015-01-04 S1F0CTDN       0  112311608        528          0          4
## 3198 2015-01-05 S1F0CTDN       0  134261688        528          0          4
## 3199 2015-01-06 S1F0CTDN       0  159974064        528          0          4
## 3200 2015-01-07 S1F0CTDN       1  184069720        528          0          4
## 3201 2015-01-01 S1F0CTDY       0   40804784        192          0          0
## 3202 2015-01-02 S1F0CTDY       0   56611416        192          0          0
## 3203 2015-01-03 S1F0CTDY       0   74367848        192          0          0
## 3204 2015-01-04 S1F0CTDY       0   91399392        192          0          0
## 3706 2015-02-10 S1F0DSTY       0    9962632          0          0          0
## 3707 2015-02-11 S1F0DSTY       0   78874408          0          0          0
## 3708 2015-02-12 S1F0DSTY       0  204249320          0          0         39
## 3709 2015-02-13 S1F0DSTY       0  239198208       1440          0         41
## 3710 2015-02-14 S1F0DSTY       1   97170872       2576          0         60
## 3711 2015-01-01 S1F0E9EP       0  106791400          0          0          0
## 3712 2015-01-02 S1F0E9EP       0  126870472          0          0          0
## 3713 2015-01-03 S1F0E9EP       0  147004000          0          0          0
## 3714 2015-01-04 S1F0E9EP       0  169708424          0          0          0
## 4656 2015-05-03 S1F0F4EB       0  210648704          0          0          0
## 4657 2015-05-04 S1F0F4EB       0  232409568          0          0          0
## 4658 2015-05-05 S1F0F4EB       0   53270184          0          0          0
```

```
## 4659 2015-05-06 S1F0F4EB       0   33624256          0          0          0
## 4660 2015-05-07 S1F0F4EB       1  243261216          0          0          0
##      attribute5 attribute6 attribute7 attribute8 attribute9
## 422          19     510519         16         16          3
## 423          19     511783         16         16          3
## 424          19     513110         16         16          3
## 425          19     513722         16         16          3
## 426          19     514661         16         16          3
## 427          14     311869          0          0          0
## 428          14     311869          0          0          0
## 429          14     311869          0          0          0
## 430          14     312779          0          0          0
## 1045          8     289753          0          0          0
## 1046          8     291025          0          0          0
## 1047          8     292335          0          0          0
## 1048          8     293573          0          0          0
## 1049          8     294852          0          0          0
## 1050          5     226578          0          0          0
## 1051          5     227942          0          0          0
## 1052          5     229360          0          0          0
## 1053          5     230748          0          0          0
## 1712          7     414833          0          0          2
## 1713          7     416005          0          0          2
## 1714          7     416919          0          0          2
## 1715          7     417826          0          0          2
## 1716          7     418563          0          0          2
## 1717          3     306534          0          0          4
## 1718          3     306534          0          0          4
## 1719          3     306534          0          0          4
## 1720          3     306534          0          0          4
## 3196          9     383713         32         32          3
## 3197          9     384948         32         32          3
## 3198          9     386214         32         32          3
## 3199          9     387343         32         32          3
## 3200          9     387871         32         32          3
## 3201          4     337518          0          0          0
## 3202          4     338339          0          0          0
## 3203          4     339425          0          0          0
## 3204          4     340527          0          0          0
## 3706         12     462170          0          0          0
## 3707         12     462170          0          0          0
## 3708         12     462175          0          0          0
## 3709         12     462175          0          0          0
## 3710         12     462175          0          0          0
## 3711          8     196552          0          0          0
## 3712          8     197887          0          0          0
## 3713          8     199248          0          0          0
## 3714          8     200566          0          0          0
## 4656         10     255731          0          0          3
## 4657         10     255731          0          0          3
## 4658         10     255731          0          0          3
## 4659         10     255731          0          0          3
## 4660         10     255731          0          0          3
```

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 ...

```
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
```

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

```
##
##            0            1
## 0.9991485533 0.0008514467
```

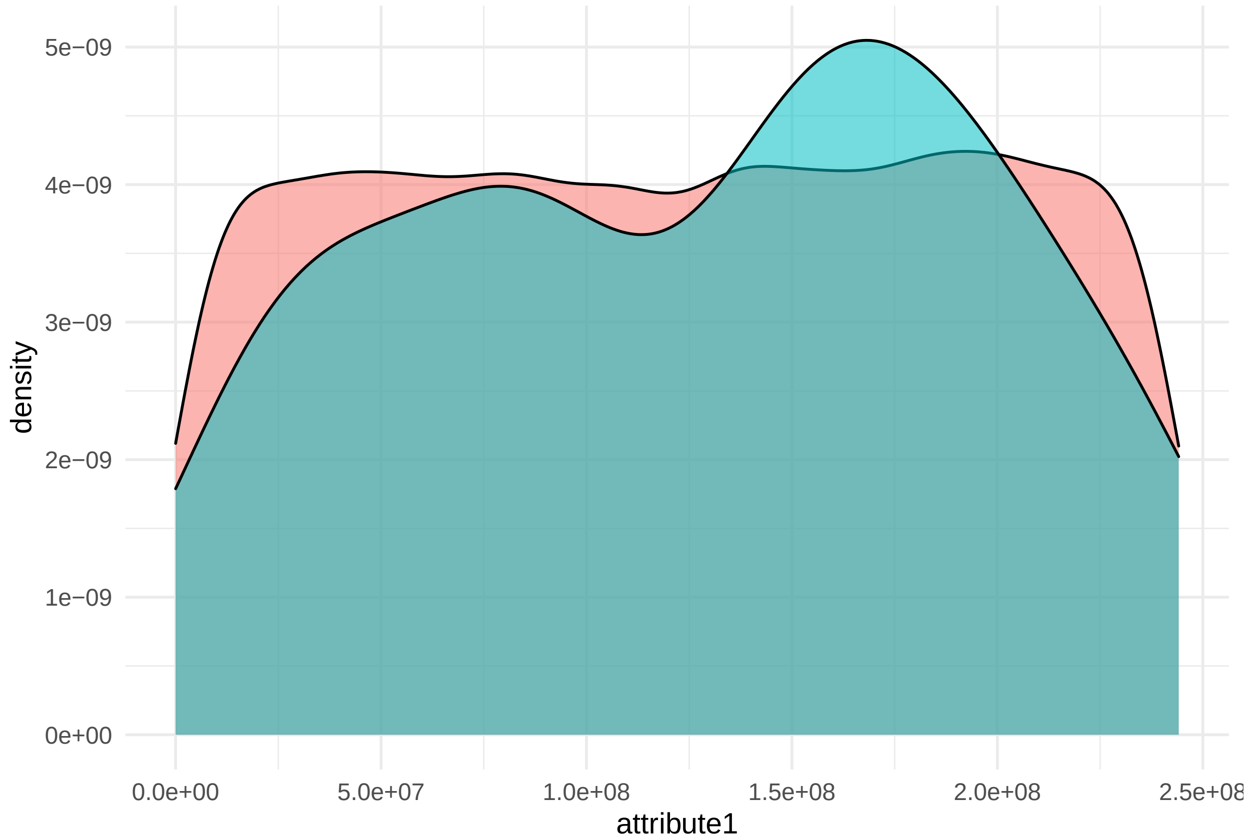
```
summary(data.raw)
```

```
##       date                device             failure
##  Min.   :2015-01-01   Length:124494      Min.   :0.0000000
##  1st Qu.:2015-02-09   Class :character   1st Qu.:0.0000000
##  Median :2015-03-27   Mode  :character   Median :0.0000000
##  Mean   :2015-04-16                      Mean   :0.0008514
##  3rd Qu.:2015-06-17                      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.: 61284762   1st Qu.:    0.0   1st Qu.:    0.00   1st Qu.:   0.000
##  Median :122797388   Median :    0.0   Median :    0.00   Median :   0.000
##  Mean   :122388103   Mean   :  159.5   Mean   :    9.94   Mean   :   1.741
##  3rd Qu.:183309640   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.:221452   1st Qu.:  0.0000   1st Qu.:  0.0000
##  Median :10.00   Median :249800   Median :  0.0000   Median :  0.0000
##  Mean   :14.22   Mean   :260173   Mean   :  0.2925   Mean   :  0.2925
##  3rd Qu.:12.00   3rd Qu.:310266   3rd Qu.:  0.0000   3rd Qu.:  0.0000
##  Max.   :98.00   Max.   :689161   Max.   :832.0000   Max.   :832.0000
##    attribute9
##  Min.   :    0.00
##  1st Qu.:    0.00
##  Median :    0.00
##  Mean   :   12.45
##  3rd Qu.:    0.00
##  Max.   :18701.00
```

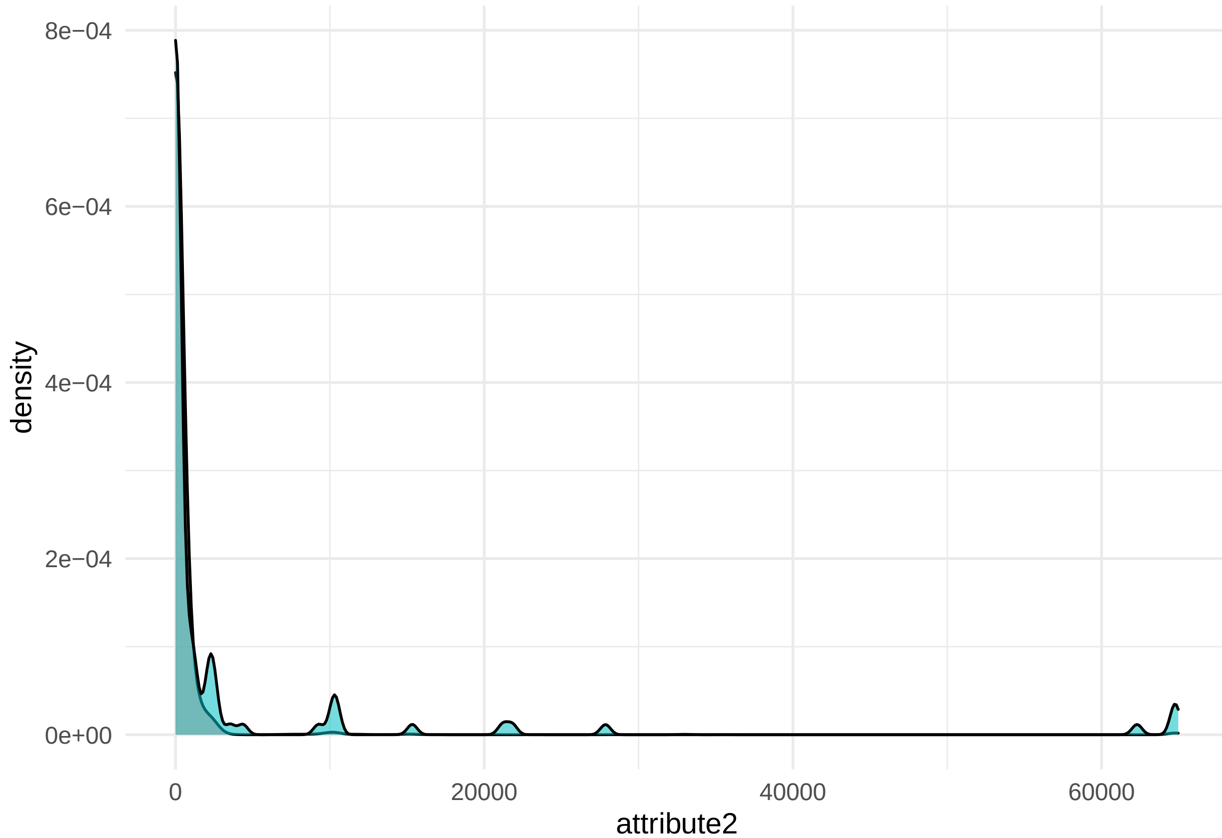
Focus on success stories

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

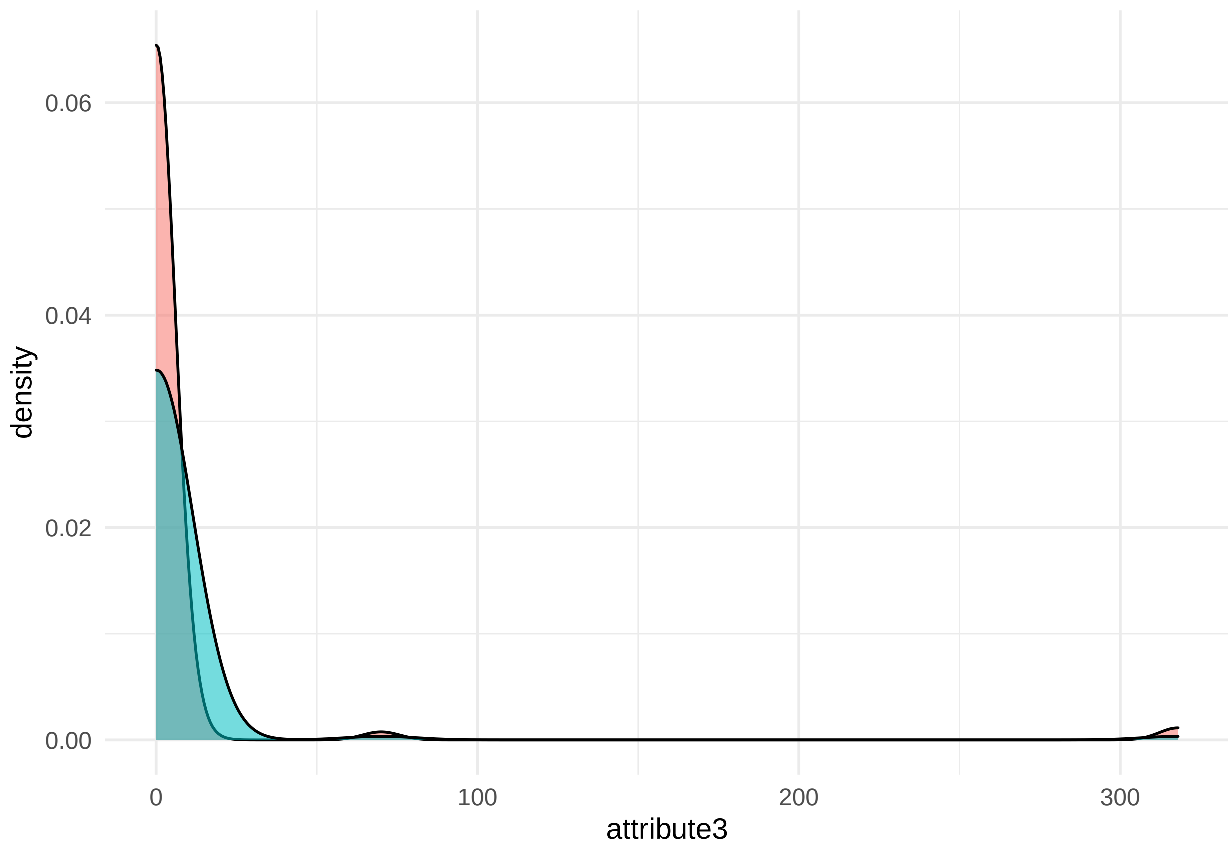
```
device.with.failures <- unique(data.raw$device[n.fails.index] )
data.sample <- data.raw[ data.raw$device %in% device.with.failures, ]
ggplot(data.sample, aes(attribute1, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() + theme(legend.position="none")
```



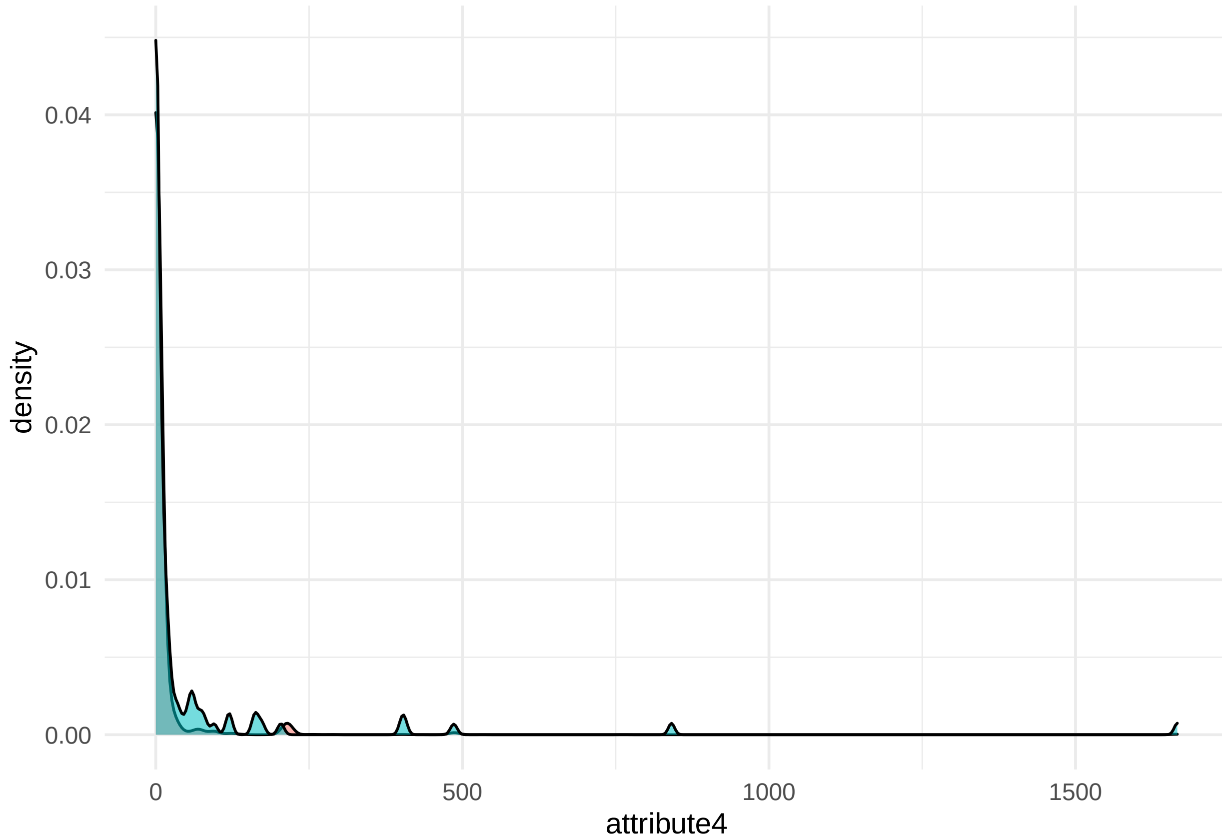

```
ggplot(data.sample, aes(attribute2, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



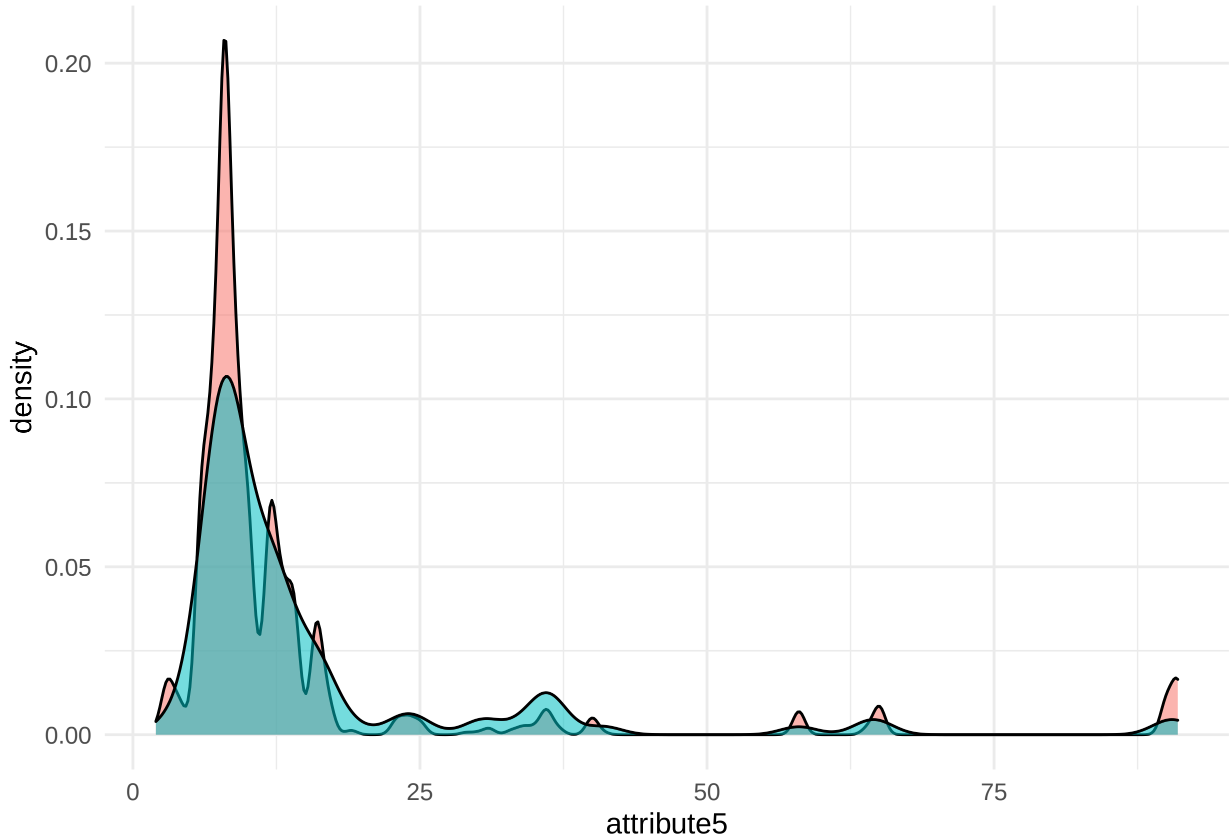

```
ggplot(data.sample, aes(attribute3, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



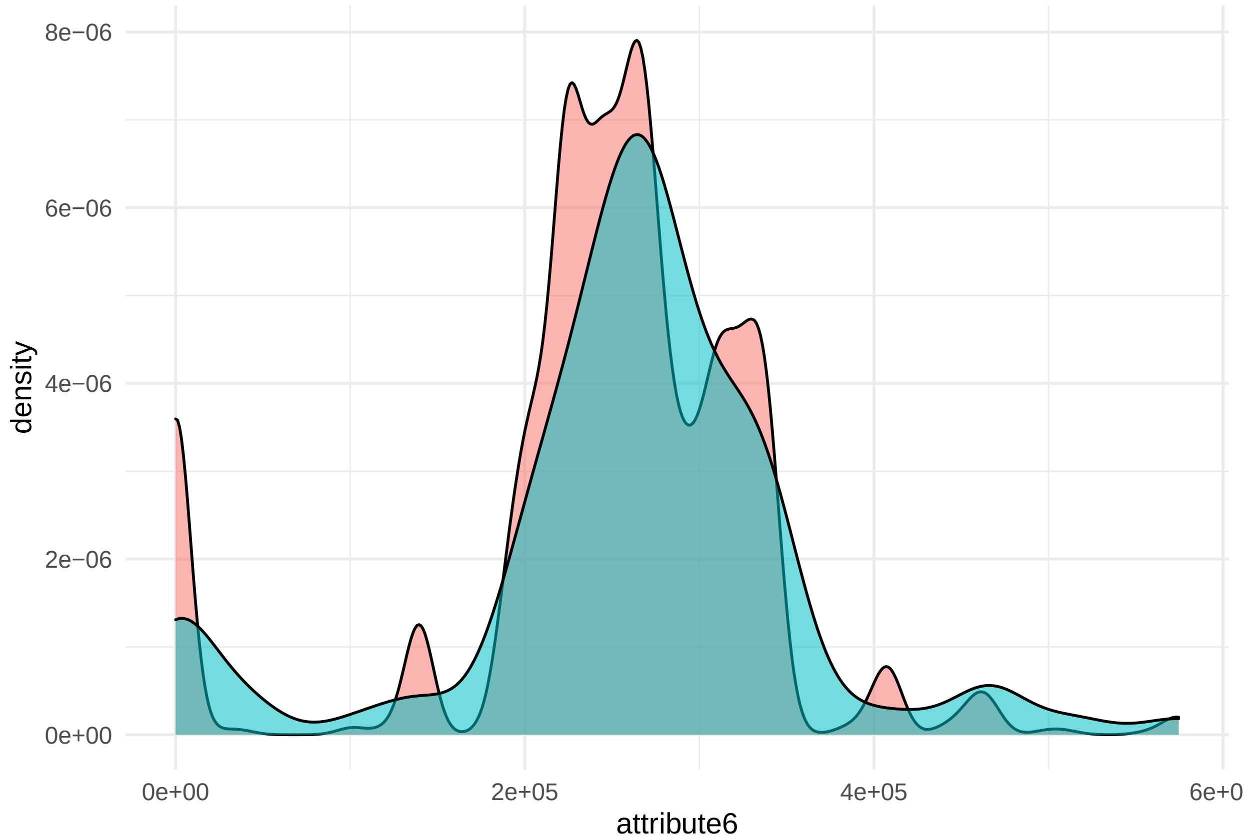

```r
ggplot(data.sample, aes(attribute4, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



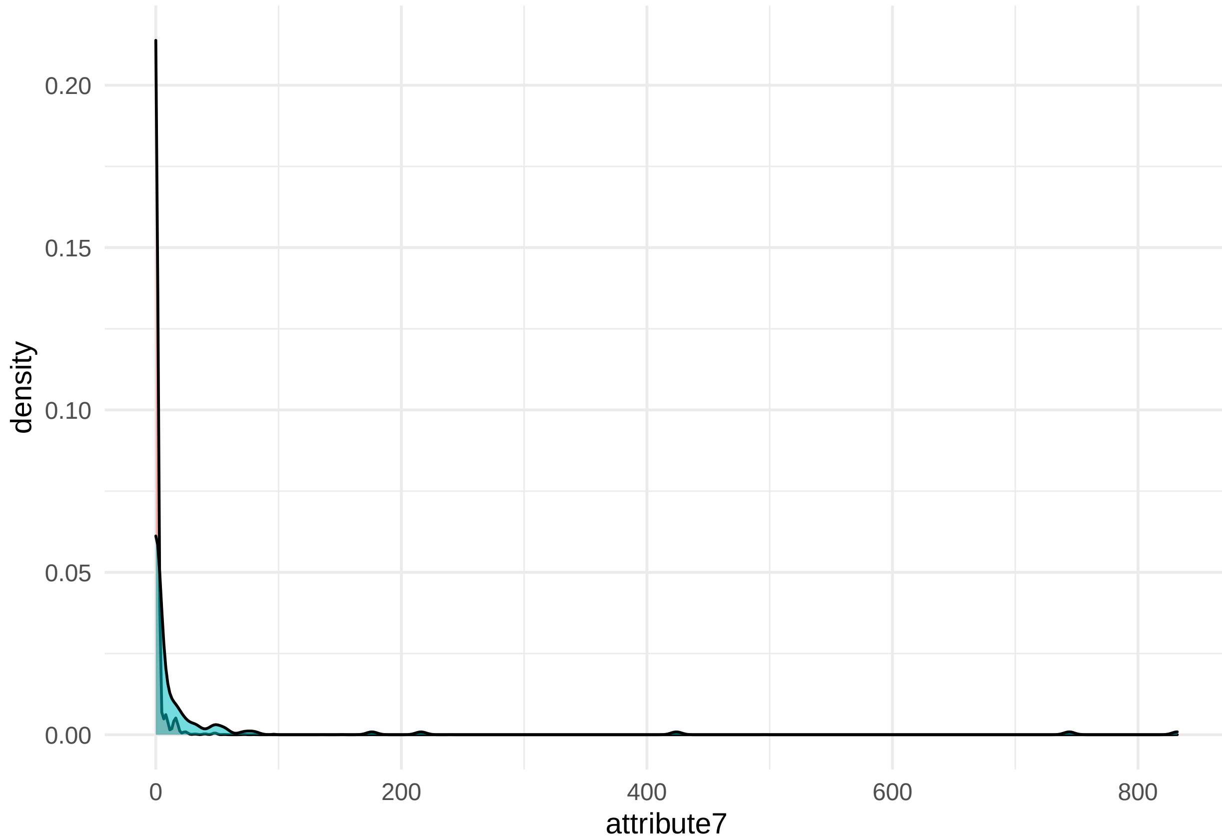

```
ggplot(data.sample, aes(attribute5, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



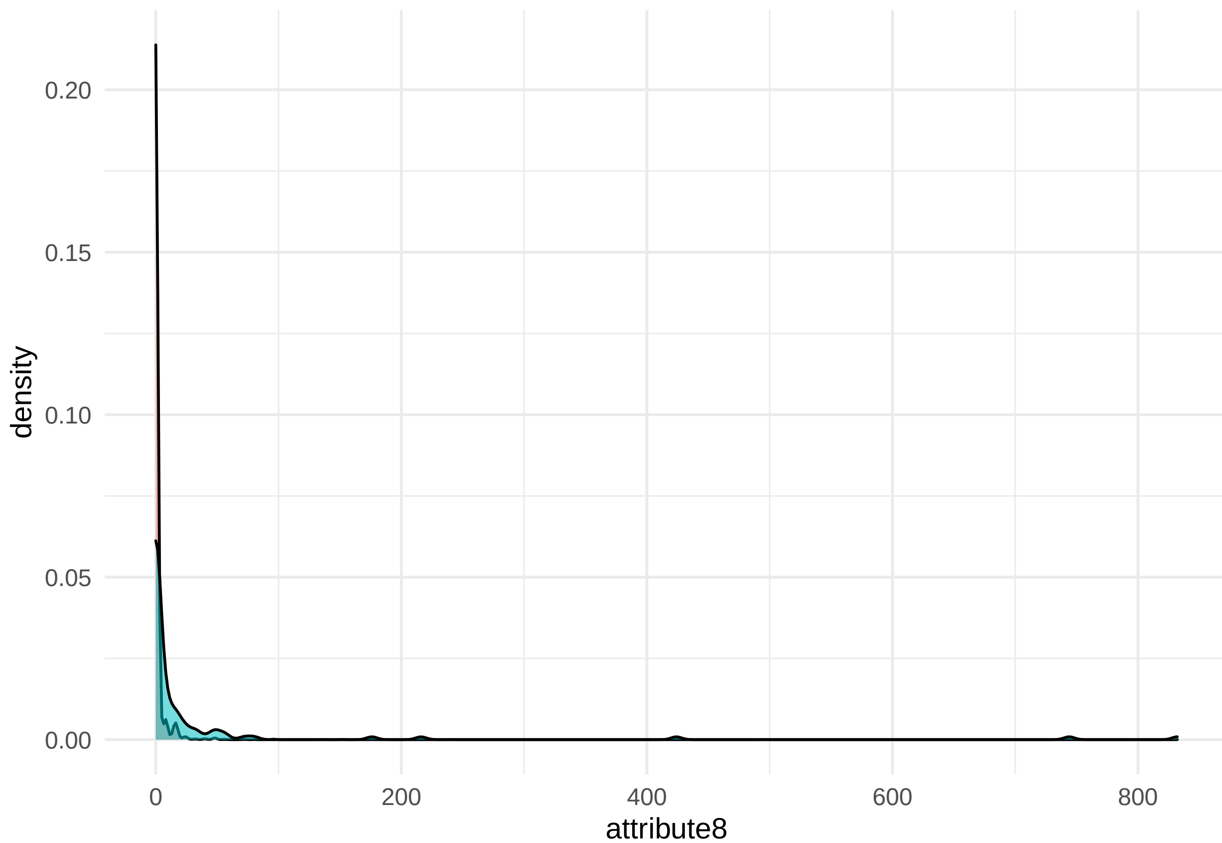

```r
ggplot(data.sample, aes(attribute6, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



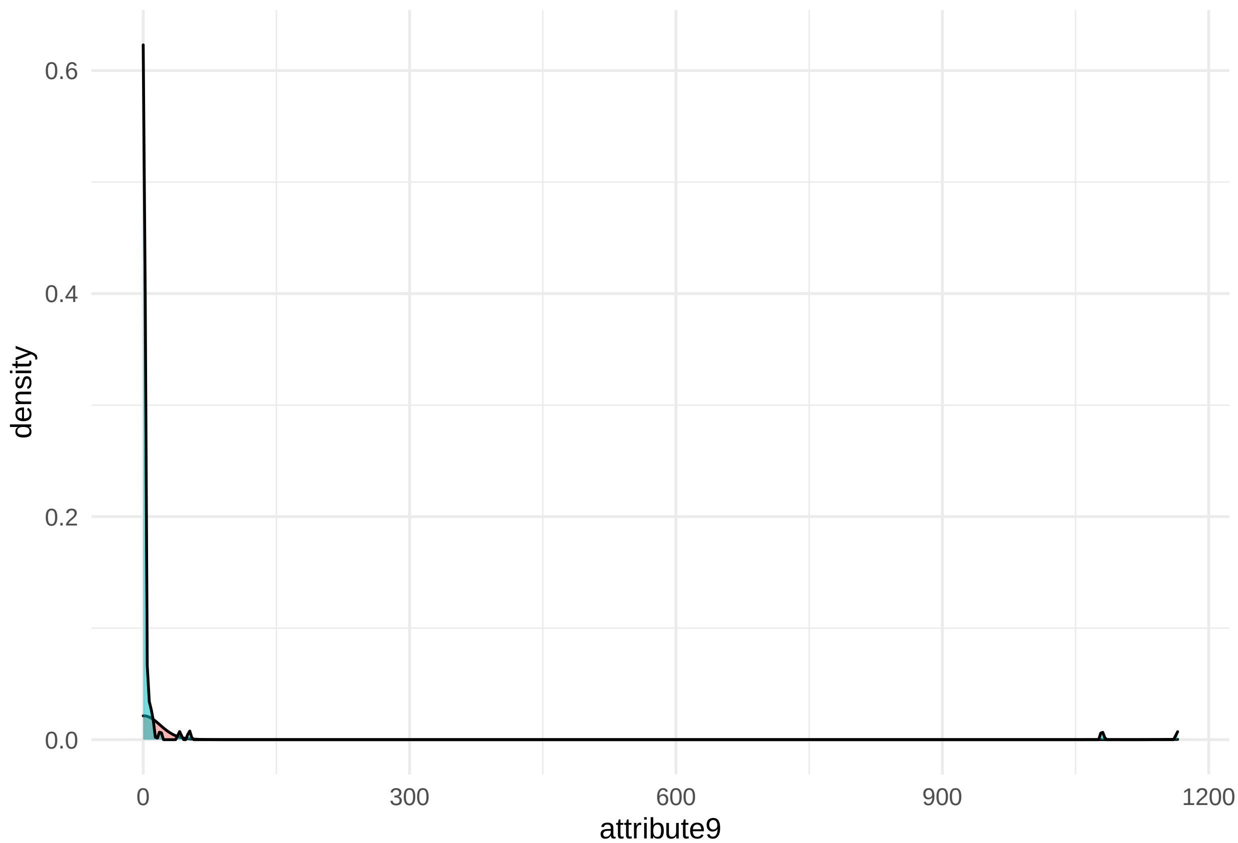

```
ggplot(data.sample, aes(attribute7, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```




```
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")
```



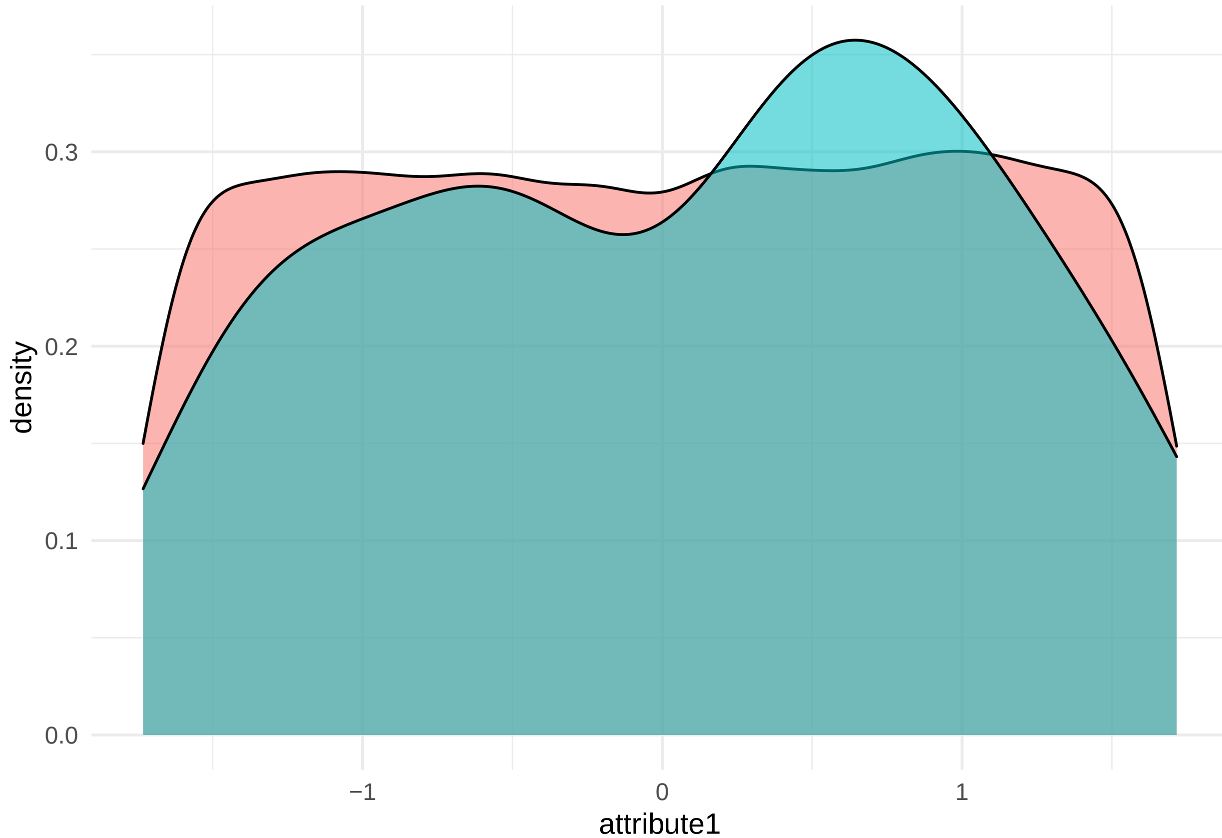
Due to the distribution of some of the variables, we apply a non-linear transformation that allows us to more easily discriminate between failures and non-failures.

```
index.columns <- c(2, 3, 4, 7, 8, 9) + 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)
```

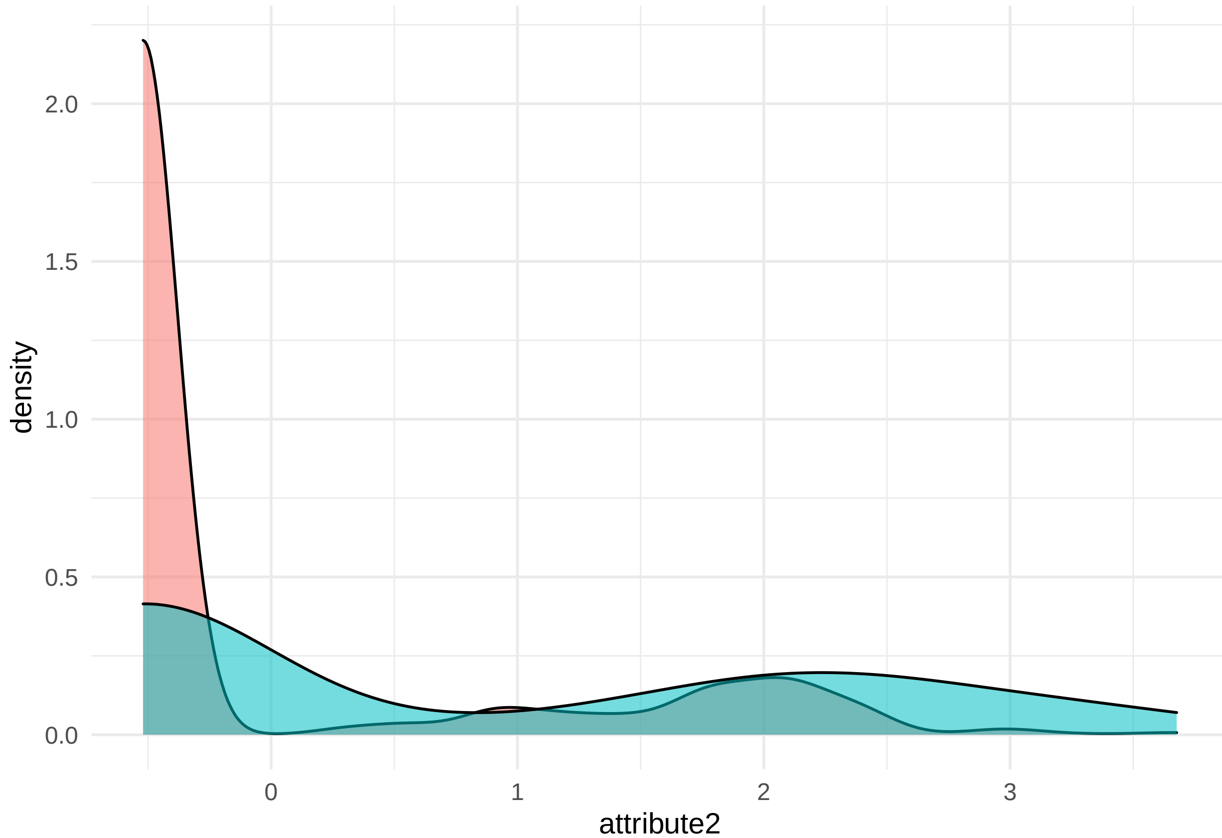
```
##       date               device             failure            attribute1
##  Min.   :2015-01-01   Length:10713       Min.   :0.000000   Min.   :     4224
##  1st Qu.:2015-01-30   Class :character   1st Qu.:0.000000   1st Qu.: 61036016
##  Median :2015-03-11   Mode  :character   Median :0.000000   Median :123326224
##  Mean   :2015-03-22                      Mean   :0.009895   Mean   :122631486
##  3rd Qu.:2015-05-01                      3rd Qu.:0.000000   3rd Qu.:184069720
##  Max.   :2015-10-26                      Max.   :1.000000   Max.   :244135688
##    attribute2        attribute3       attribute4        attribute5
##  Min.   : 0.000   Min.   :0.000   Min.   :0.0000   Min.   : 2.00
##  1st Qu.: 0.000   1st Qu.:0.000   1st Qu.:0.0000   1st Qu.: 8.00
##  Median : 0.000   Median :0.000   Median :0.0000   Median : 9.00
##  Mean   : 1.373   Mean   :0.236   Mean   :0.5068   Mean   :14.21
##  3rd Qu.: 0.000   3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.:13.00
##  Max.   :11.082   Max.   :5.765   Max.   :7.4188   Max.   :91.00
##    attribute6        attribute7       attribute8       attribute9
##  Min.   :    19   Min.   :0.000   Min.   :0.000   Min.   :0.0000
```

```
##  1st Qu.:222242   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.0000
##  Median :256129   Median :0.000   Median :0.000   Median :0.0000
##  Mean   :247970   Mean   :0.190   Mean   :0.190   Mean   :0.4361
##  3rd Qu.:299197   3rd Qu.:0.000   3rd Qu.:0.000   3rd Qu.:0.0000
##  Max.   :574599   Max.   :6.725   Max.   :6.725   Max.   :7.0613
```

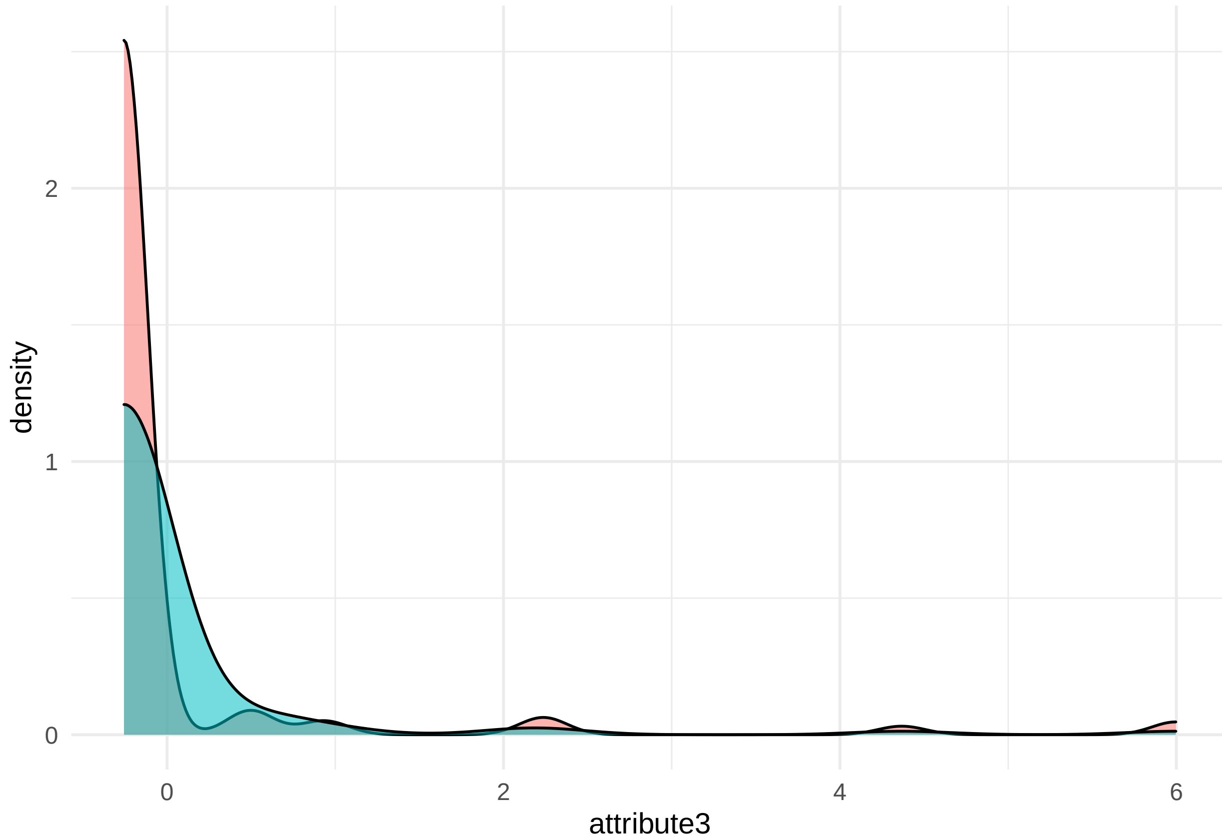
```r
for ( i in index.columns ){
  temp <- data.sample[, names(data.sample)[i]]
  data.sample[, names(data.sample)[i]] <- scale(temp)
}
ggplot(data.sample, aes(attribute1, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() + theme(legend.position="none")
```



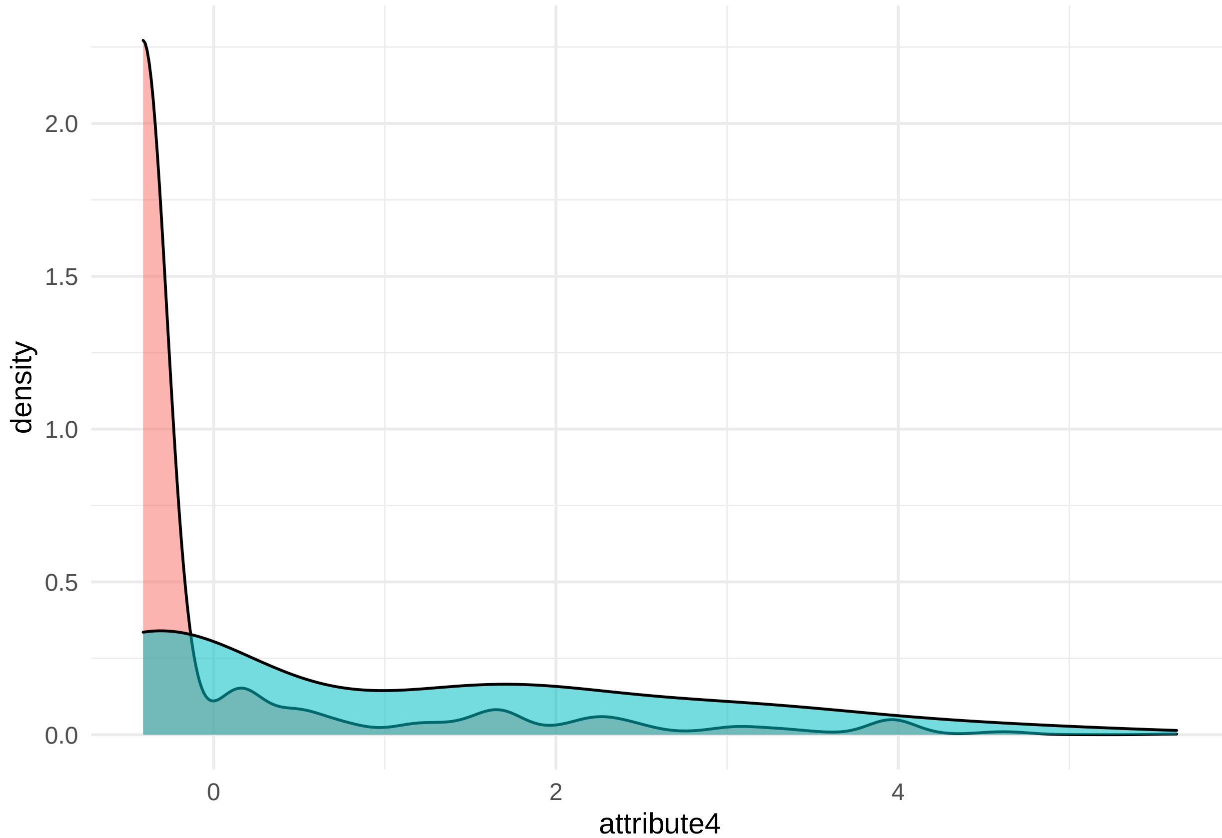

```r
ggplot(data.sample, aes(attribute2, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



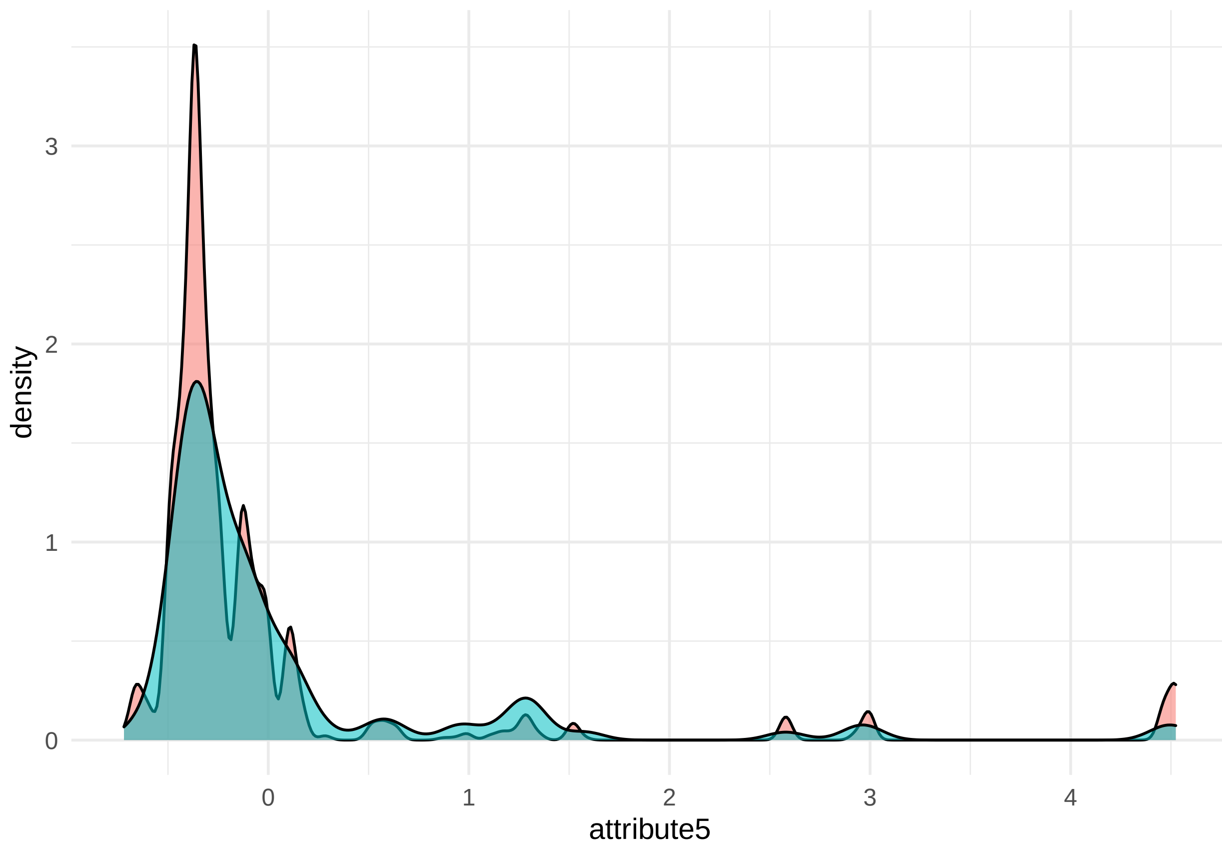

```
ggplot(data.sample, aes(attribute3, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal() +theme(legend.position="none")
```



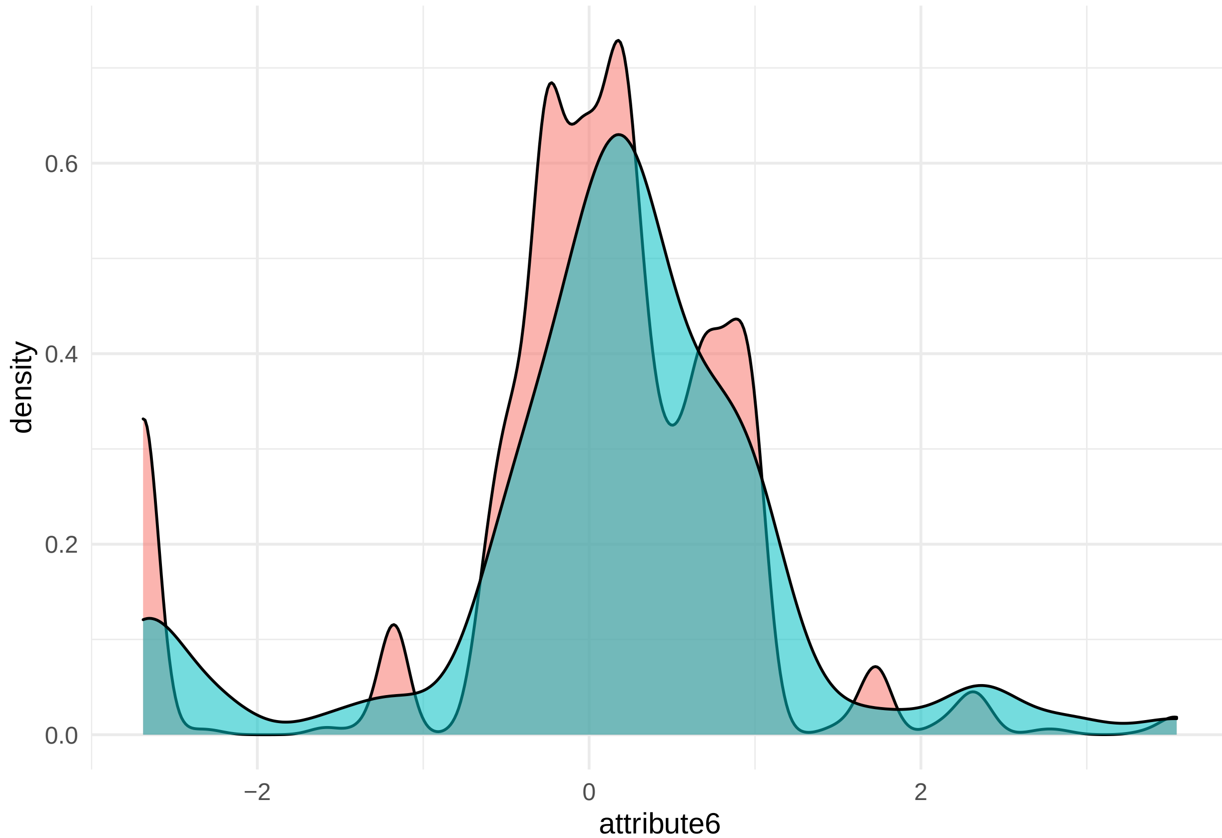

```
ggplot(data.sample, aes(attribute4, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



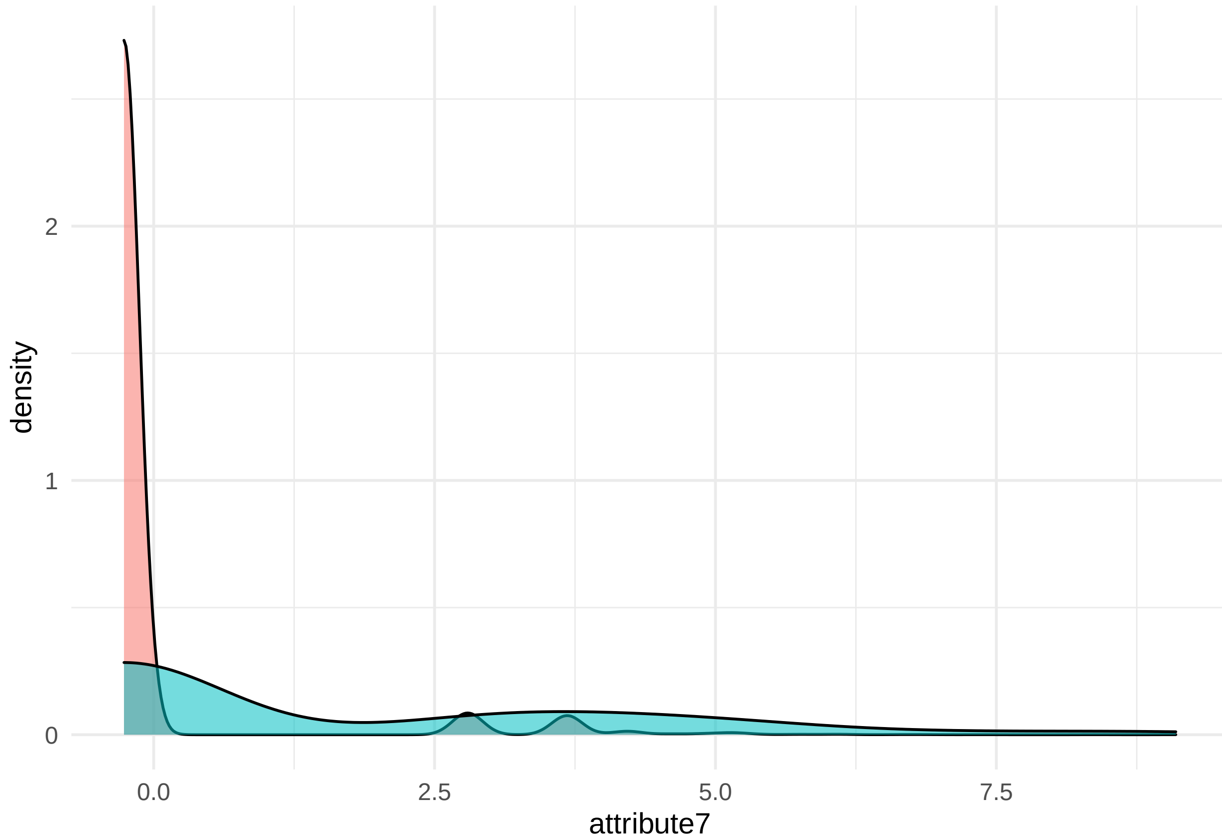

```r
ggplot(data.sample, aes(attribute5, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



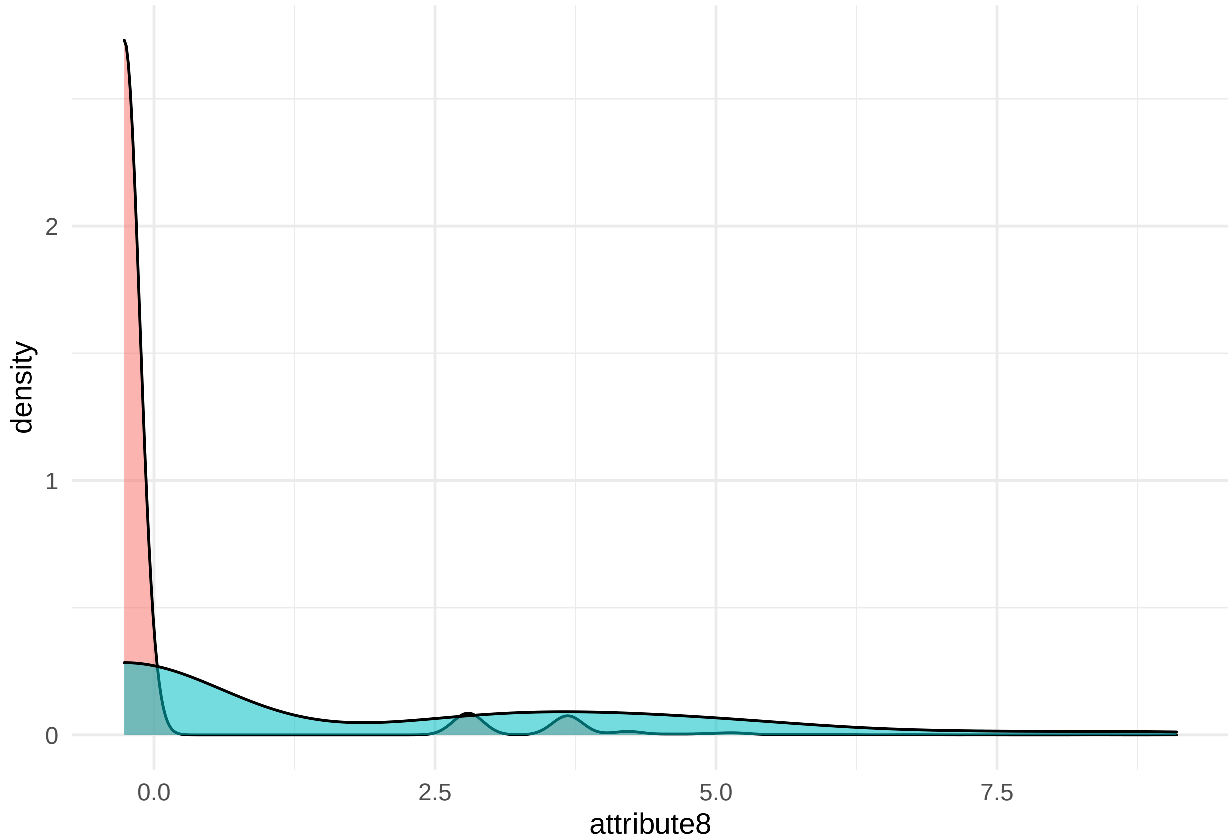

```
ggplot(data.sample, aes(attribute6, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```



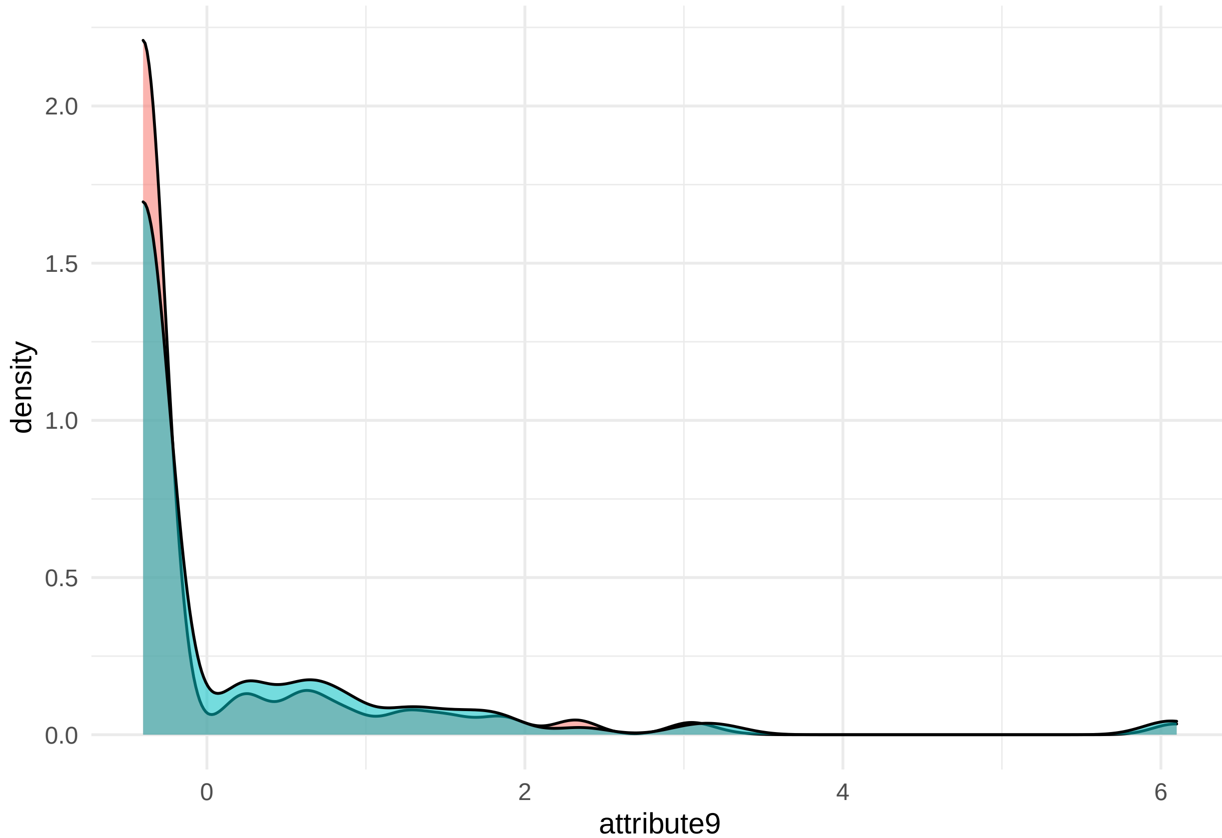

```
ggplot(data.sample, aes(attribute7, fill =  as.character(failure), alpha=.01)) +
  geom_density()  + theme_minimal()+theme(legend.position="none")
```




```r
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)
}
```

```
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:

```
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 0.01219512 0.03410688 0.04296460 0.04550566 0.05845819 0.07608696    0
## XGB 0.01324503 0.03552263 0.04706534 0.04605392 0.05585896 0.07253886    0
## RF  0.02395210 0.04024590 0.04866864 0.05001125 0.05809740 0.08383234    0
## RLG 0.02564103 0.04318703 0.05063494 0.05385692 0.06554422 0.10084034    0
```

```
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        -0.0005483 -0.0045056 -0.0083513
## XGB 1.0000            -0.0039573 -0.0078030
## RF  1.0000 1.0000                -0.0038457
## RLG 0.3446 0.3562     1.0000
```

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

```
## Time difference of 1.858871 mins
```

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

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2210       7
##   Failure        1047      25
##
##                Accuracy : 0.6795
##                  95% CI : (0.6633, 0.6955)
##     No Information Rate : 0.9903
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0269
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.67854
##             Specificity : 0.78125
##          Pos Pred Value : 0.99684
##          Neg Pred Value : 0.02332
##              Prevalence : 0.99027
##          Detection Rate : 0.67194
##    Detection Prevalence : 0.67407
##       Balanced Accuracy : 0.72989
##
##        'Positive' Class : NoFailure
##
```

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

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2606       8
##   Failure         651      24
##
##                Accuracy : 0.7996
##                  95% CI : (0.7855, 0.8132)
##     No Information Rate : 0.9903
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0502
##
```

```
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.80012
##             Specificity : 0.75000
##          Pos Pred Value : 0.99694
##          Neg Pred Value : 0.03556
##              Prevalence : 0.99027
##          Detection Rate : 0.79234
##    Detection Prevalence : 0.79477
##       Balanced Accuracy : 0.77506
##
##        'Positive' Class : NoFailure
##
```

```r
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 3.331422 mins
```

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

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2610      10
##   Failure         647      22
##
##                Accuracy : 0.8002
##                  95% CI : (0.7862, 0.8138)
##     No Information Rate : 0.9903
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.045
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.80135
##             Specificity : 0.68750
##          Pos Pred Value : 0.99618
##          Neg Pred Value : 0.03288
```

```
##             Prevalence : 0.99027
##         Detection Rate : 0.79355
##   Detection Prevalence : 0.79659
##      Balanced Accuracy : 0.74443
##
##       'Positive' Class : NoFailure
##
```

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

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NoFailure Failure
##   NoFailure      2166       6
##   Failure        1091      26
##
##                Accuracy : 0.6665
##                  95% CI : (0.6501, 0.6826)
##     No Information Rate : 0.9903
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0268
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.66503
##             Specificity : 0.81250
##          Pos Pred Value : 0.99724
##          Neg Pred Value : 0.02328
##              Prevalence : 0.99027
##          Detection Rate : 0.65856
##    Detection Prevalence : 0.66038
##       Balanced Accuracy : 0.73876
##
##        'Positive' Class : NoFailure
##
```