In [41]:
```
#loading library

library(boot)
library(caret)
library(tidyverse)
library(ISLR2)
library(glmnet)
library(tidyverse)
library(ISLR2)
library(caret)
library(e1071)
library(neuralnet)
library(dplyr)
library(ggplot2)
library(Hmisc)
library(corrplot)
library(DescTools)
library(kernlab)
library(xgboost)
```

In [42]:
```
#controlling the size of plot outputs
fig <- function(width, heigth){
     options(repr.plot.width = width, repr.plot.height = heigth)
}
fig(18,10)
```

In [43]:
```
#loading data and naming default set
defaults <-  read.csv("data.csv")

#checking for nas
sum(is.na(defaults))
```

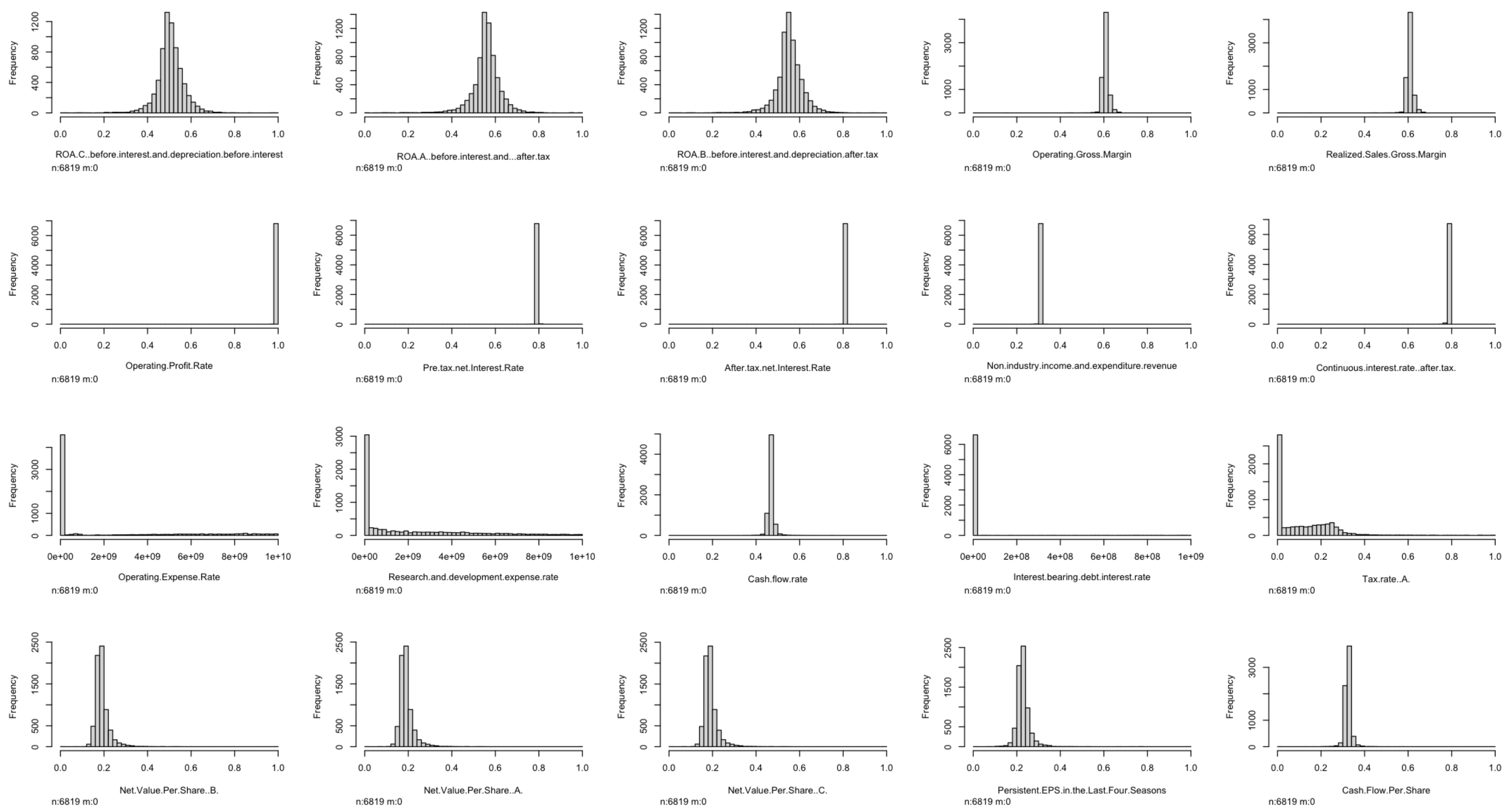
```
0
```

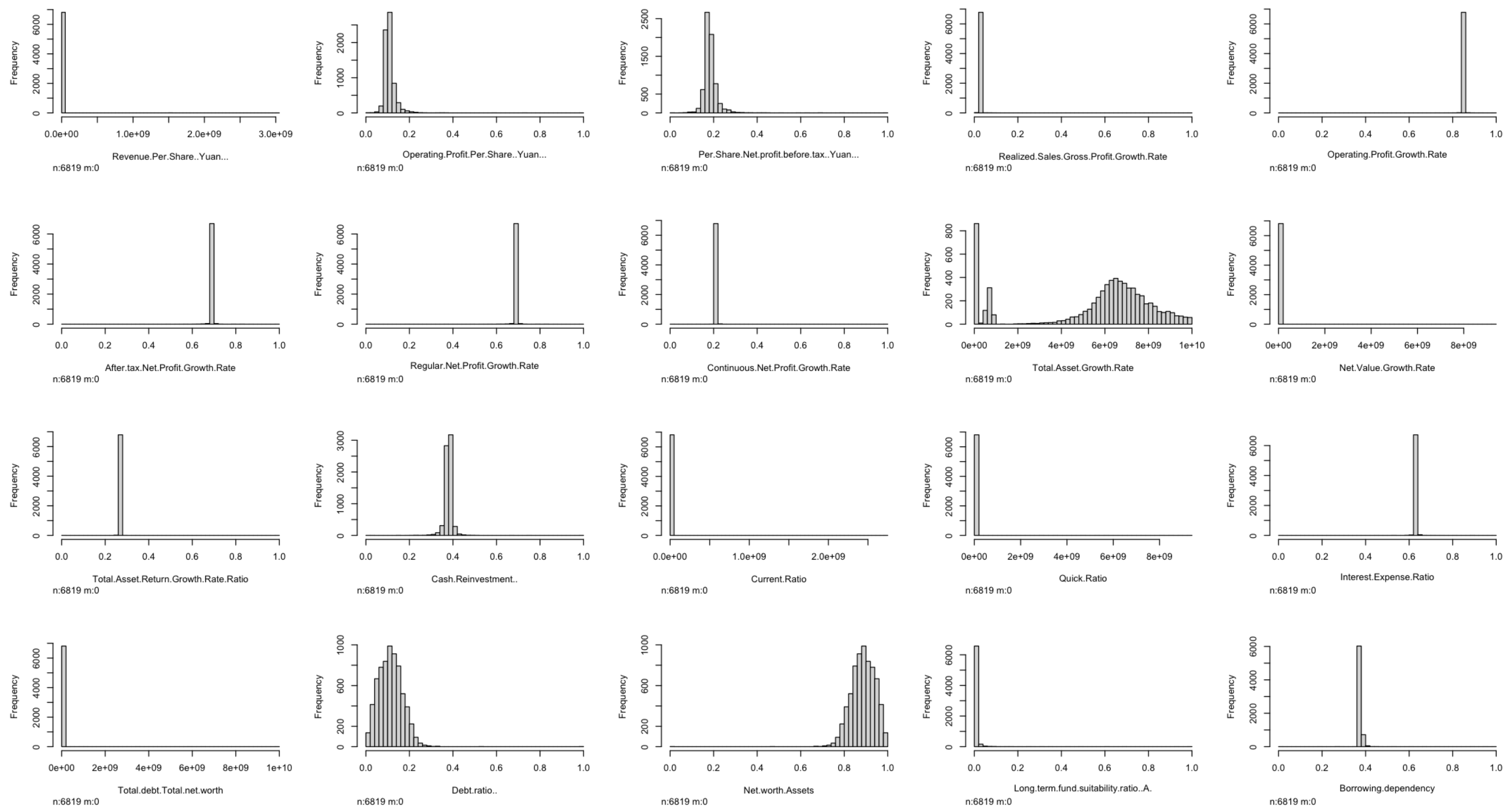
In [44]:
```
#in this block we look at the types of data in the our data set, there are only 2 factor variables, one integer a

table(sapply(defaults,class))
```

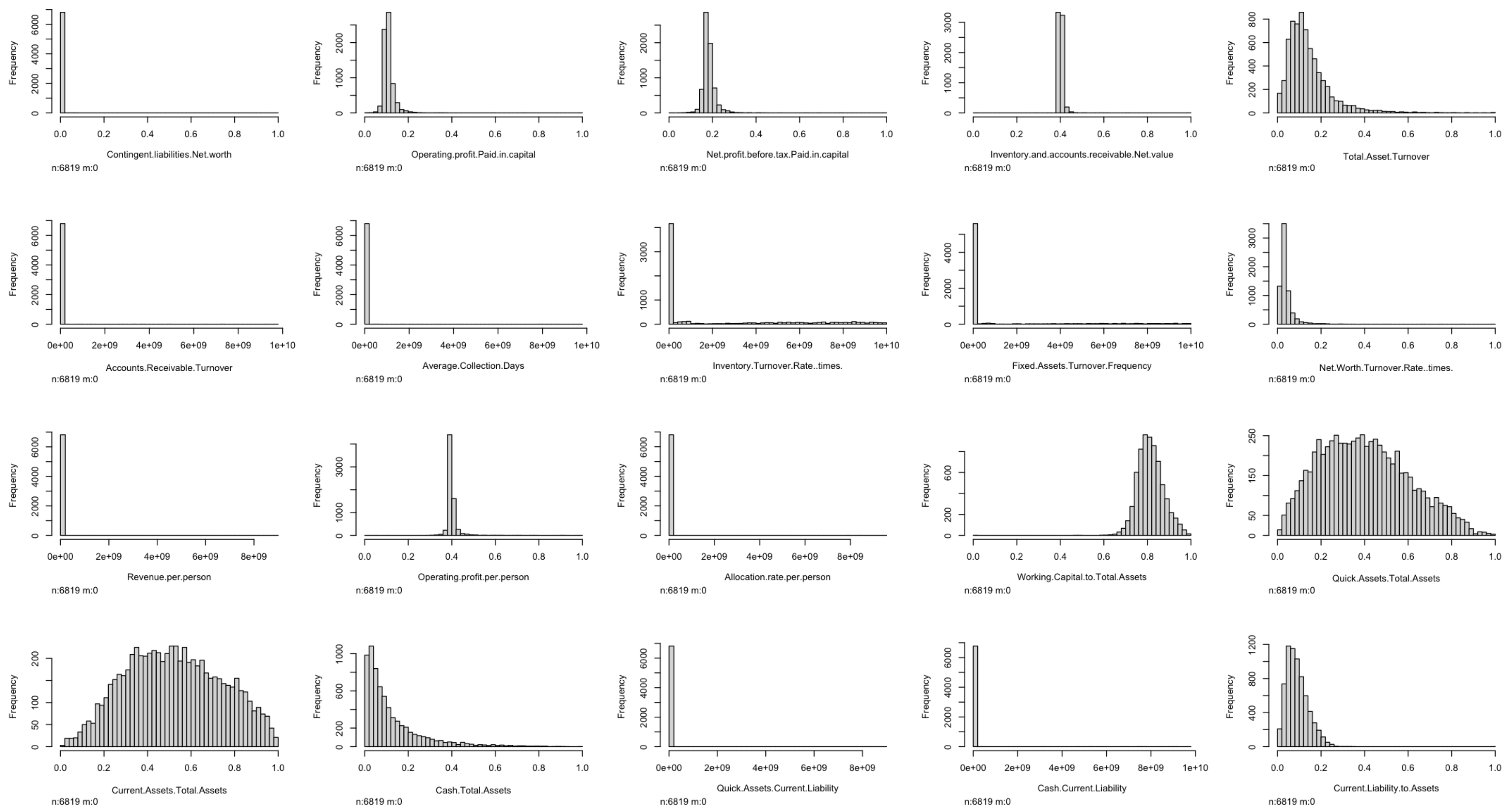
```
integer numeric 
      3      93 
```

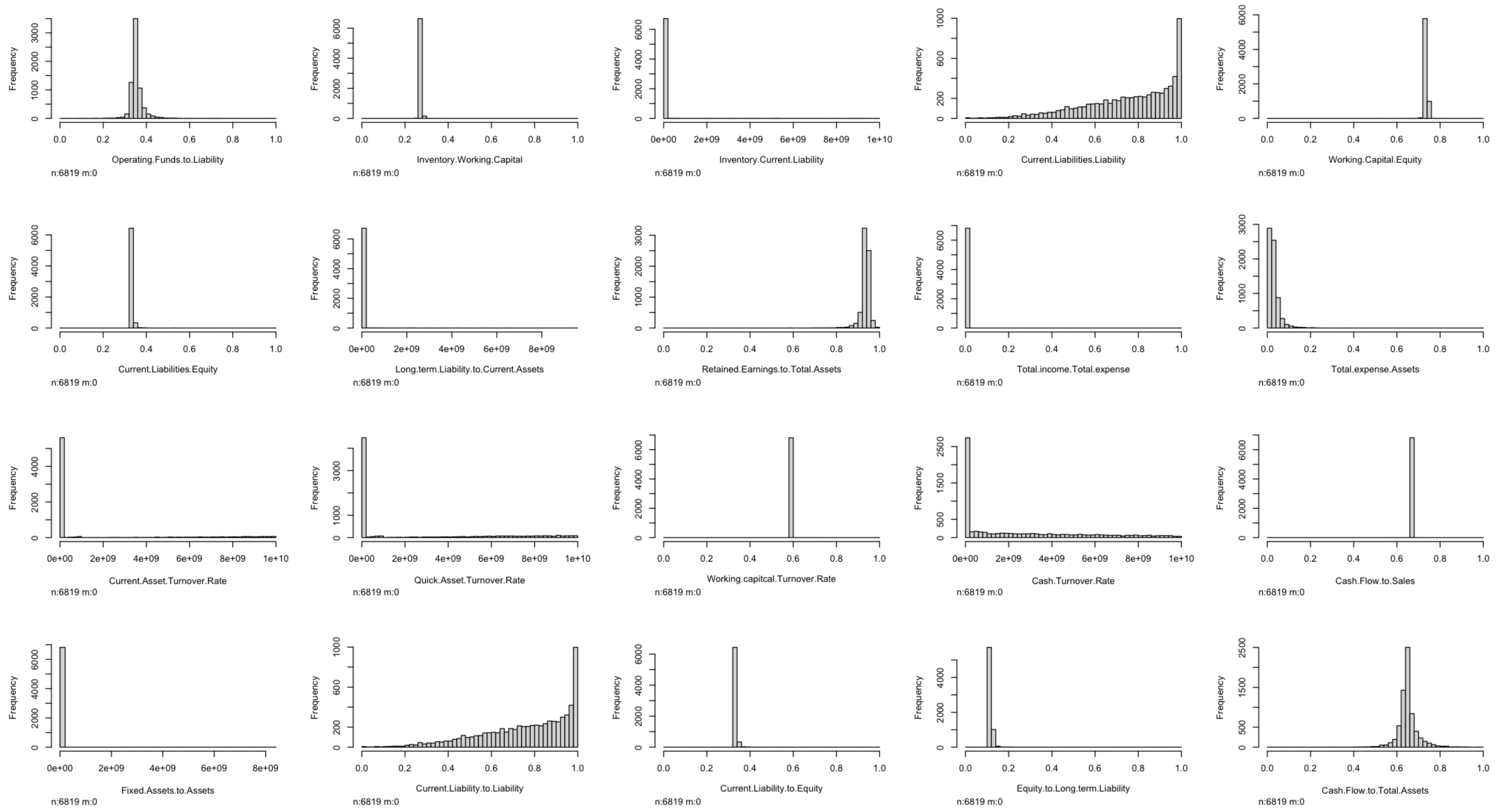
In [45]:
```
#redefining factor variables
defaults$Bankrupt. <- as.factor(defaults$Bankrupt.)
defaults$Liability.Assets.Flag <- as.factor(defaults$Liability.Assets.Flag)

#removing this column because it has zero variance i.e all the values are the same. this means no new informatio
defaults$Net.Income.Flag <- NULL
```

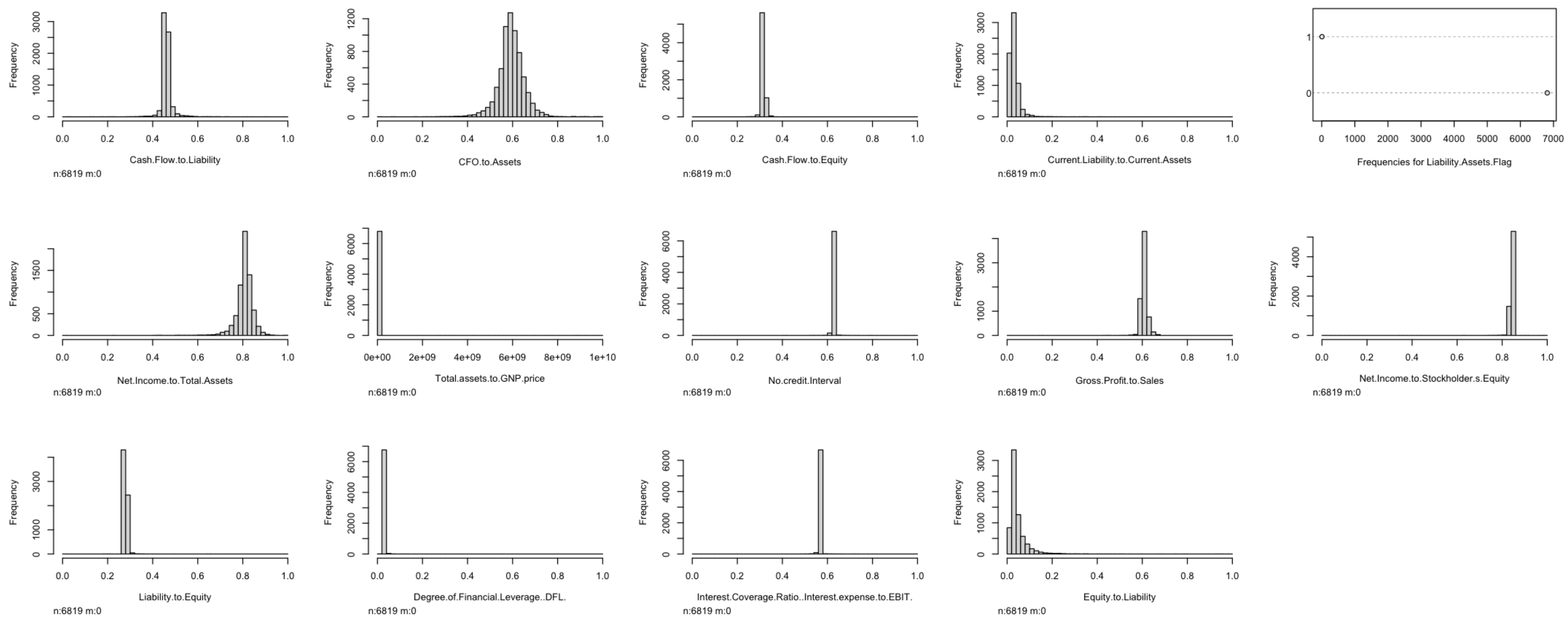
In [48]:
```
#examining the distribution of each feature in the dataset
hist.data.frame(defaults[-1]) #generates histogram of each feature in model to examine their distributions
```


ROA.C..before.interest.and.depreciation.before.interest
n:6819 m:0
ROA.A..before.interest.and...after.tax
n:6819 m:0
ROA.B..before.interest.and.depreciation.after.tax
n:6819 m:0
Operating.Gross.Margin
n:6819 m:0
Realized.Sales.Gross.Margin
n:6819 m:0
Operating.Profit.Rate
n:6819 m:0
Pre.tax.net.Interest.Rate
n:6819 m:0
After.tax.net.Interest.Rate
n:6819 m:0
Non.industry.income.and.expenditure.revenue
n:6819 m:0
Continuous.interest.rate..after.tax.
n:6819 m:0
Operating.Expense.Rate
n:6819 m:0
Research.and.development.expense.rate
n:6819 m:0
Cash.flow.rate
n:6819 m:0
Interest.bearing.debt.interest.rate
n:6819 m:0
Tax.rate..A.
n:6819 m:0
Net.Value.Per.Share..B.
n:6819 m:0
Net.Value.Per.Share..A.
n:6819 m:0
Net.Value.Per.Share..C.
n:6819 m:0
Persistent.EPS.in.the.Last.Four.Seasons
n:6819 m:0
Cash.Flow.Per.Share
n:6819 m:0
Frequency

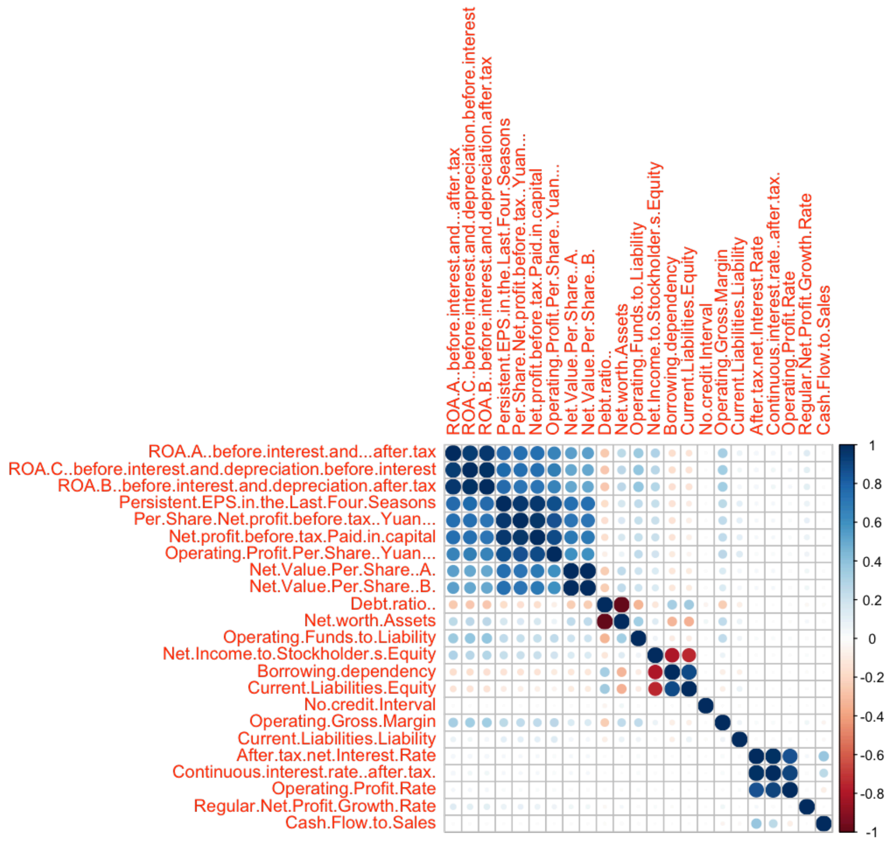

Frequency
Revenue.Per.Share..Yuan...
n:6819 m:0
Operating.Profit.Per.Share..Yuan...
n:6819 m:0
Per.Share.Net.profit.before.tax..Yuan...
n:6819 m:0
Realized.Sales.Gross.Profit.Growth.Rate
n:6819 m:0
Operating.Profit.Growth.Rate
n:6819 m:0
After.tax.Net.Profit.Growth.Rate
n:6819 m:0
Regular.Net.Profit.Growth.Rate
n:6819 m:0
Continuous.Net.Profit.Growth.Rate
n:6819 m:0
Total.Asset.Growth.Rate
n:6819 m:0
Net.Value.Growth.Rate
n:6819 m:0
Total.Asset.Return.Growth.Rate.Ratio
n:6819 m:0
Cash.Reinvestment..
n:6819 m:0
Current.Ratio
n:6819 m:0
Quick.Ratio
n:6819 m:0
Interest.Expense.Ratio
n:6819 m:0
Total.debt.Total.net.worth
n:6819 m:0
Debt.ratio..
n:6819 m:0
Net.worth.Assets
n:6819 m:0
Long.term.fund.suitability.ratio..A.
n:6819 m:0
Borrowing.dependency
n:6819 m:0


Contingent.liabilities.Net.worth
n:6819 m:0
Operating.profit.Paid.in.capital
n:6819 m:0
Net.profit.before.tax.Paid.in.capital
n:6819 m:0
Inventory.and.accounts.receivable.Net.value
n:6819 m:0
Total.Asset.Turnover
n:6819 m:0
Accounts.Receivable.Turnover
n:6819 m:0
Average.Collection.Days
n:6819 m:0
Inventory.Turnover.Rate..times.
n:6819 m:0
Fixed.Assets.Turnover.Frequency
n:6819 m:0
Net.Worth.Turnover.Rate..times.
n:6819 m:0
Revenue.per.person
n:6819 m:0
Operating.profit.per.person
n:6819 m:0
Allocation.rate.per.person
n:6819 m:0
Working.Capital.to.Total.Assets
n:6819 m:0
Quick.Assets.Total.Assets
n:6819 m:0
Current.Assets.Total.Assets
n:6819 m:0
Cash.Total.Assets
n:6819 m:0
Quick.Assets.Current.Liability
n:6819 m:0
Cash.Current.Liability
n:6819 m:0
Current.Liability.to.Assets
n:6819 m:0
Frequency


Frequency
Operating.Funds.to.Liability
n:6819 m:0
Inventory.Working.Capital
n:6819 m:0
Inventory.Current.Liability
n:6819 m:0
Current.Liabilities.Liability
n:6819 m:0
Working.Capital.Equity
n:6819 m:0
Current.Liabilities.Equity
n:6819 m:0
Long.term.Liability.to.Current.Assets
n:6819 m:0
Retained.Earnings.to.Total.Assets
n:6819 m:0
Total.income.Total.expense
n:6819 m:0
Total.expense.Assets
n:6819 m:0
Current.Asset.Turnover.Rate
n:6819 m:0
Quick.Asset.Turnover.Rate
n:6819 m:0
Working.capitcal.Turnover.Rate
n:6819 m:0
Cash.Turnover.Rate
n:6819 m:0
Cash.Flow.to.Sales
n:6819 m:0
Fixed.Assets.to.Assets
n:6819 m:0
Current.Liability.to.Liability
n:6819 m:0
Current.Liability.to.Equity
n:6819 m:0
Equity.to.Long.term.Liability
n:6819 m:0
Cash.Flow.to.Total.Assets
n:6819 m:0

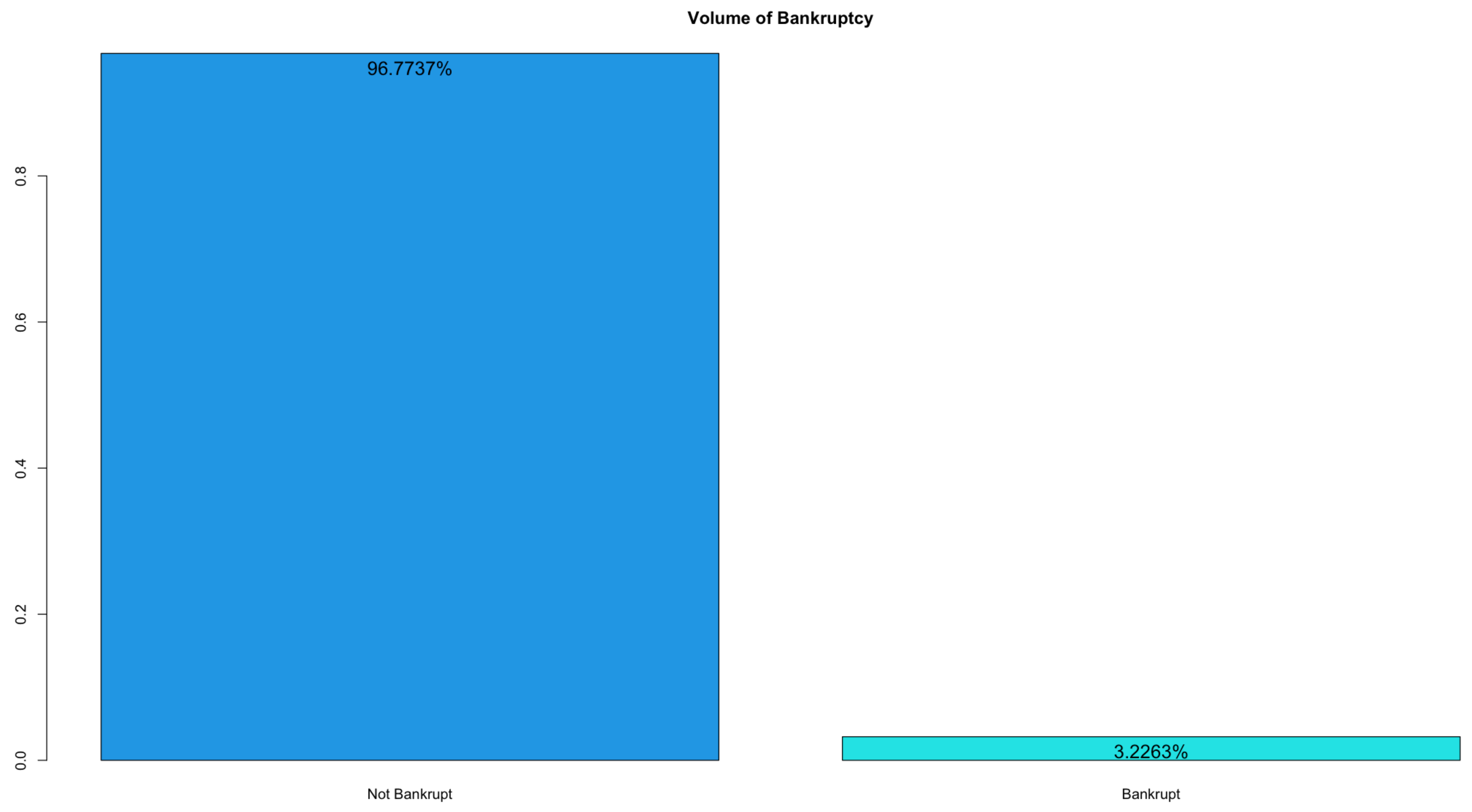


In [47]:
```
#finding highly correlated variables
de.cor <- cor(select_if(defaults,is.numeric)) #selects only numeric columns since cor object doesn't like factors

#finding highly correlated variables at specified threshold and then getting index of those variables
cor.indx <- findCorrelation(de.cor, cutoff = 0.8, verbose = FALSE )

#visualising on the combinations with correlations above 0.8 due to the high number of features
corrplot(cor(defaults[-1][cor.indx]), method = 'circle')
```


ROA.A..before.interest.and...after.tax
ROA.C..before.interest.and.depreciation.before.interest
ROA.B..before.interest.and.depreciation.after.tax
Persistent.EPS.in.the.Last.Four.Seasons
Per.Share.Net.profit.before.tax..Yuan...
Net.profit.before.tax.Paid.in.capital
Operating.Profit.Per.Share..Yuan...
Net.Value.Per.Share..A.
Net.Value.Per.Share..B.
Debt.ratio..
Net.worth.Assets
Operating.Funds.to.Liability
Net.Income.to.Stockholder.s.Equity
Borrowing.dependency
Current.Liabilities.Equity
No.credit.Interval
Operating.Gross.Margin
Current.Liabilities.Liability
After.tax.net.Interest.Rate
Continuous.interest.rate..after.tax.
Operating.Profit.Rate
Regular.Net.Profit.Growth.Rate
Cash.Flow.to.Sales
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

```
In [ ]: # creating new data set without highly correlated features
        defaults.clean <- defaults[, !names(defaults) %in% #removes all column names in below list from dataframe
                  c("ROA.B..before.interest.and.depreciation.after.tax",
                   "ROA.C..before.interest.and.depreciation.before.interest",
                   "Persistent.EPS.in.the.Last.Four.Seasons",
                   "Operating.Profit.Per.Share..Yuan...",
                   "Current.Liabilities.Equity",
                   "Current.Liability.to.Equity",
                   "Operating.Gross.Margin",
                   "Pre.tax.net.Interest.Rate",
                   "Operating.Profit.Rate",
                   "Net.Value.Per.Share..A.",
                   "Net.Value.Per.Share..B.",
                   "Net.Value.Per.Share..C.")
                                ]
```

```
In [49]: #visualizing bankruptcy classes
         bankrupt <- sum(defaults$Bankrupt.== 1)/length(defaults$Bankrupt.) #counting number of bankrupt companies as frac
         not.bankrupt <- 1-bankrupt #counting no bankrupt cases

         # bankrupt #percent of defaulters
         # not.bankrupt*100 #percent of non-defaulters

         #visualizing data as a bar plot
         bar <- barplot( c(not.bankrupt,bankrupt),
                names.arg = c("Not Bankrupt", "Bankrupt"), #naming bars
                 col = 4:5, #assigning color to each bar
                 main = "Volume of Bankruptcy", #title of plot
                 )

         text(x=bar, #since by default, barplot gets midpoint of each bar, so using this sets the text placement
              y = c(not.bankrupt,bankrupt)-.02, # sets vertical height of text. i adjusted it downwards
              labels=c("96.7737%","3.2263%"), #sets lables
              cex = 1.3 #sets font size
             )
```



Methodology

Data Preparation - Splitting Train and Validation Sets and scaling data

In [50]:
```
set.seed(1000)

splitSample <- sample(1:2, size=nrow(defaults.clean), prob=c(0.9,0.1), replace = TRUE)

train_set <- defaults.clean[splitSample==1,]
valid_set <- defaults.clean[splitSample==2,]
```

In [51]:
```
#Standardize sets
count = 2
check = 0
while(count < length(defaults.clean[3,]) && check == 0){
  count = count + 1
  if(colnames(defaults.clean[1,])[count] == "Liability.Assets.Flag"){
    check = 1
  }
}
```

In [52]:
```
for (i in 2:length(defaults.clean[3,])){
  if(i != count){
    means = mean(train_set[,i])
    sds = sd(train_set[,i])
    valid_set[,i] = (valid_set[,i]-means)/sds
  }
}
```

In [53]:
```
train_liability <- train_set[,count]

total_col <- length(defaults.clean[3,])-1

train_set[,count] <- NULL

train_set[,2:total_col] <- scale(train_set[,2:total_col])

train_set[,total_col+1] <- train_liability

valid_liability <- valid_set[,count]
valid_set[,count] <- NULL
valid_set[,total_col+1] <- valid_liability

colnames(train_set)[total_col+1] <- "Liability.Assets.Flag"
colnames(valid_set)[total_col+1] <- "Liability.Assets.Flag"
```

Model Cross Validation

In [ ]:
```
#defining cross entropy loss function for random forest

crossEntropyLoss <- function(data, lev = NULL, model = NULL) {
  predictions <- predict(model, data, type = "prob")
  actual <- as.numeric(data$target) - 1
  loss <- -sum(actual * log(predictions[, 2]) + (1 - actual) * log(predictions[, 1])) / nrow(data)
  return(list(metric = loss, classProbs = predictions))
}
```

In [62]:
```
#setting parameters for k-fold cross validation
control.base <- trainControl(method="cv", number=10)
```

In [56]:
```
#Logistic Regression
fit.glm <- caret::train(Bankrupt. ~ .,
                  data = train_set,
                  method = "glm",
                  family = "binomial",
                  trControl = control.base)
```

```
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
```

```
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
```

In [57]:
```
#Random Forest
#defining grid for cross validation, due to size of feature space, we use the root of number of features. standa

#setting parameters for cross validation
ctrl.rf <- trainControl(method="cv", number=10)

tunegrid <- expand.grid(.mtry=sqrt(dim(train_set)[2])) #changes loss function to cross entropy

fit.rf<- caret::train(Bankrupt.~.,
                      data=train_set,
                      method = 'rf',
                      metric = 'Accuracy',
                      type = "class",
                      tuneGrid = tunegrid,
                      trControl = ctrl.rf)
```

In [59]:

```r
#xgboosted tree

#setting parameters for cross validation
ctrl.xgboost <- trainControl(method="cv", number=10)

# Define grid of parameter combinations to tune over. Here everything is at default value except for the number c
xgboost.grid <- expand.grid(
      nrounds = c(100, 500,1000),
      max_depth = 6,
      eta = 0.3,
      gamma = 0.1,
      colsample_bytree = 1,
      min_child_weight = 1,
      subsample = 1
)

# Define cross-validation scheme
ctrl <- trainControl(
      method = "repeatedcv",
      number = 10,
      repeats = 1,
      verboseIter = FALSE,
      allowParallel = TRUE
  # numberParallel = 4
)

# Train the model using cross-validation
xgb_model <- train(
      Bankrupt. ~ .,
      data = train_set,
      method = "xgbTree",
      trControl = ctrl,
      tuneGrid = xgboost.grid,
      nthread = 5
)
```

```
[12:34:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:34:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:35:07] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:35:07] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:35:43] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:35:43] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:36:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:36:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:37:09] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:37:09] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:37:49] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:37:49] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:38:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:38:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:39:08] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:39:08] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:39:53] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:39:53] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:40:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
[12:40:35] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
```

In [64]:
```
#SVM

#creating grid of parameters for to search through during tuning
tunegrid.linear <- expand.grid(C = c(0.001,0.1,1,10,100))

#we benchmark the less computationally intense SVM w/linear kernel against the more expensive radial kernel

cv.svm.linear <- train(Bankrupt.~.,
                       data = train_set,
                       method="svmLinear", #defining kernel
                       tuneGrid = tunegrid.linear,
                       trControl=control.base)

cv.svm.radial <- train(Bankrupt.~.,
                      data = train_set,
                      method="svmRadial",
                      trControl=control.base)

linear.svm.param <- cv.svm.linear$bestTune #best tuning for linear svm
# linear.svm.param
radial.svm.param <- cv.svm.radial$bestTune #best tuning for radial svm
# radial.svm.param
```

In [65]:
```
#SVM Sigmoid
cost <- c(0.001, 0.01, 0.1, 1, 10, 1/dim(train_set))
gamma <- c(0.01, 0.1, 0.25, 0.5, 1)
sigmoid_svms_accu_mean <- data.frame(
  cost = 0,
  gamma = 0,
  accuracy = 0
)

j = 1
for(c in cost){
    for (g in gamma){
      sigmoid_svms_accu <- c()
      for (i in length(10)){

        splitSample <- sample(1:2, size=nrow(train_set), prob=c(0.9,0.1), replace = TRUE)
        train <- train_set[splitSample==1,]
```

```r
        test <- train_set[splitSample==2,]
        sigmoid_svms <- svm(Bankrupt. ~., data = train, kernel = "sigmoid", cost = c, gamma = g)
        predict.svms <- predict(sigmoid_svms, test)
        cm.svms <- confusionMatrix(predict.svms, test$Bankrupt.)
        sigmoid_svms_accu[i] <- cm.svms$overall['Accuracy']
      }
      new_row = list(cost = c, gamma= g, accuracy = mean(sigmoid_svms_accu))
      sigmoid_svms_accu_mean = rbind(sigmoid_svms_accu_mean,new_row)
      j = j +1
  }
}

max_accuracy = 0
index = 1
j = 1
for(i in sigmoid_svms_accu_mean$accuracy){
  if(i > max_accuracy){
    max_accuracy = i
    index = j
  }
  j = j+1
}

#sigmoid_svms_accu_mean[which.max(sigmoid_svms_accu_mean$accuracy),]
#which.max(sigmoid_svms_accu_mean$accuracy)
```

In [66]:
```r
cv.svm.sigmoid <- svm(Bankrupt. ~., data = train_set,
                    kernel = "sigmoid",
                    cost = sigmoid_svms_accu_mean[index,1],
                    gamma = sigmoid_svms_accu_mean[index,2],
                    classProbs = TRUE)
```

In [68]:
```
#kNN

#tunning to find optimal number of neighbors for this dataset
cv.knn <- train(Bankrupt.~.,
                      data=train_set,
                      method="knn",
                      trControl=control.base)

#optimal number of neighbors
# knn.param <- cv.knn$bestTune
```

Cross Validation Summary

In [84]:
```
data.frame(Models = c("Logistic Regression", "Random Forest", "XGBoosted Tree",
                      "Linear SVM", "Radial SVM","Sigmoidal SVM", "kNN"),
          Cross.Validated.Accuracy = c(fit.glm$results$Accuracy,
          fit.rf$results$Accuracy,
          xgb_model$results$Accuracy[1],
          cv.svm.linear$results[1,2],
          cv.svm.radial$results[1,3],
          sigmoid_svms_accu_mean[which.max(sigmoid_svms_accu_mean$accuracy),3],
          cv.knn$results[2,2]
                      )
           )
```

A data.frame: 7 × 2

Models	Cross.Validated.Accuracy
<chr>	<dbl>
Logistic Regression	0.9539564
Random Forest	0.9706550
XGBoosted Tree	0.9685478
Linear SVM	0.9683852
Radial SVM	0.9678992
Sigmoidal SVM	0.9819079
kNN	0.9698462

Model Validation

In [85]:

```
Results <- data.frame(Metric = c("Accuracy","Sensitivity","Specifity")) #generating dataframe to view collective

Models <- list(fit.glm, fit.rf, xgb_model,cv.svm.linear, cv.svm.radial,cv.svm.sigmoid, cv.knn) #storing models ir

Model.Names <- c("Logistic Regression", "Random Forest", "XGBoosted Tree", "Linear SVM", "Radial SVM","Sigmoidal

#the intuition behind the loop is to pass through the list of models, generate predictions and obtain the test st

for (i in 1:length(Models)){ #iterating through the list of models to calculate and visualize their performance
        temp <- confusionMatrix(    #creating confusion matrix
            predict(Models[[i]],valid_set), #nesting prediction function inside confusion matrix
            valid_set$Bankrupt.
        )
    # print(i)
    Results[Model.Names[[i]]] <- c(temp$overall['Accuracy'],temp$byClass['Sensitivity'],temp$byClass['Specificity
    temp <- NULL #clearing temp variable
    }

Results
```

```
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
“prediction from a rank-deficient fit may be misleading”
```

A data.frame: 3 × 8

Metric	Logistic Regression	Random Forest	XGBoosted Tree	Linear SVM	Radial SVM	Sigmoidal SVM	kNN
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Accuracy	0.96774194	0.9708141	0.9723502	0.9662058	0.9662058	0.9662058	0.9723502
Sensitivity	0.99841017	0.9984102	0.9952305	1.0000000	1.0000000	1.0000000	0.9984102
Specifity	0.09090909	0.1818182	0.3181818	0.0000000	0.0000000	0.0000000	0.2272727