Company Bankruptcy Prediction

Matthew Zhang

Spring 2022

Introduction

PSTAT 131 Final Project:

Oftentimes it is difficult to recognize or even understand internal factors of why a certain company has gone bankrupt outside of typical zero profits and negative returns. In the current economy, companies are at the brink of debt as a result of inflation and a variety of other factors. Thus, there has been an increasing demand of business consultation in how to avoid bankruptcy and work around a volatile economy. In this project, I have tasked myself to be a business analyst aiming to help clients determine if their company/business is at risk of bankruptcy based on a variety of given statistics and variables in classification. I will be providing recommendations, conclusions and speculation for future work in this notebook.

Objective:

It will be a binary classification denoted by the classes 0 and 1 for surviving bankruptcy and going bankrupt, respectively. Further, determining which aspects that businesses can improve to avoid bankruptcy.

About the data:

I am using the "Company Bankruptcy Prediction" dataset taken from Kaggle.com that consists of thousands of companies that have either gone bankrupt or have not gone bankrupt. The data itself was obtained from the Taiwan Economic Journal from 1999-2009. The ability to predict company bankruptcy assesses fundamental financial concepts in insurance, stakeholders and investors. The dataset that will be loaded into a dataframe is composed of 6819 observations and 96 features. https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction

Loading the data

```
#Import necessary packages
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.2.0 --
```

```
## v broom        0.7.12     v rsample      0.1.1
## v dials        0.1.0      v tune         0.2.0
## v infer        1.0.0      v workflows    0.2.6
## v modeldata    0.1.1      v workflowsets 0.2.1
## v parsnip      0.2.1      v yardstick    0.0.9
## v recipes      0.2.0
```

```
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
library(dplyr)
library(corrr)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(discrim)
```

```
##
## Attaching package: 'discrim'

## The following object is masked from 'package:dials':
##
##     smoothness
```

```
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-4
```

```
library(ggplot2)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(ggpubr)
library(janitor)
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(rpart.plot)
```

```
## Loading required package: rpart

##
## Attaching package: 'rpart'

## The following object is masked from 'package:dials':
##
##     prune
```

```
library(randomForest)
```

```
## randomForest 4.7-1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ranger)
```

```
##
## Attaching package: 'ranger'
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(vip)
```

```
##
## Attaching package: 'vip'
## The following object is masked from 'package:utils':
##
##     vi
```

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
## The following object is masked from 'package:dplyr':
##
##     slice
```

Start by loading in the data and changing the variable names for easier analysis.

```
#Import relevant data set
df <- read_csv('data/data.csv', col_names = FALSE, skip = 1)
```

```
## Rows: 6819 Columns: 96
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (96): X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(df, 5)
```

```
## # A tibble: 5 x 96
##      X1    X2    X3    X4    X5    X6    X7    X8    X9   X10   X11     X12
```

```
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1     1 0.371 0.424 0.406 0.601 0.601 0.999 0.797 0.809 0.303 0.781 1.26e-4
## 2     1 0.464 0.538 0.517 0.610 0.610 0.999 0.797 0.809 0.304 0.782 2.90e-4
## 3     1 0.426 0.499 0.472 0.601 0.601 0.999 0.796 0.808 0.302 0.780 2.36e-4
## 4     1 0.400 0.451 0.458 0.584 0.584 0.999 0.797 0.809 0.303 0.781 1.08e-4
## 5     1 0.465 0.538 0.522 0.599 0.599 0.999 0.797 0.809 0.303 0.782 7.89e+9
## # ... with 84 more variables: X13 <dbl>, X14 <dbl>, X15 <dbl>, X16 <dbl>,
## #   X17 <dbl>, X18 <dbl>, X19 <dbl>, X20 <dbl>, X21 <dbl>, X22 <dbl>,
## #   X23 <dbl>, X24 <dbl>, X25 <dbl>, X26 <dbl>, X27 <dbl>, X28 <dbl>,
## #   X29 <dbl>, X30 <dbl>, X31 <dbl>, X32 <dbl>, X33 <dbl>, X34 <dbl>,
## #   X35 <dbl>, X36 <dbl>, X37 <dbl>, X38 <dbl>, X39 <dbl>, X40 <dbl>,
## #   X41 <dbl>, X42 <dbl>, X43 <dbl>, X44 <dbl>, X45 <dbl>, X46 <dbl>,
## #   X47 <dbl>, X48 <dbl>, X49 <dbl>, X50 <dbl>, X51 <dbl>, X52 <dbl>, ...
```

```
tail(df, 2)
```

```
## # A tibble: 2 x 96
##      X1    X2    X3    X4    X5    X6    X7    X8    X9   X10   X11      X12
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1     0 0.506 0.560 0.554 0.608 0.608 0.999 0.797 0.809 0.303 0.782 0.000124
## 2     0 0.493 0.570 0.550 0.627 0.627 0.998 0.802 0.814 0.313 0.786 0.00143
## # ... with 84 more variables: X13 <dbl>, X14 <dbl>, X15 <dbl>, X16 <dbl>,
## #   X17 <dbl>, X18 <dbl>, X19 <dbl>, X20 <dbl>, X21 <dbl>, X22 <dbl>,
## #   X23 <dbl>, X24 <dbl>, X25 <dbl>, X26 <dbl>, X27 <dbl>, X28 <dbl>,
## #   X29 <dbl>, X30 <dbl>, X31 <dbl>, X32 <dbl>, X33 <dbl>, X34 <dbl>,
## #   X35 <dbl>, X36 <dbl>, X37 <dbl>, X38 <dbl>, X39 <dbl>, X40 <dbl>,
## #   X41 <dbl>, X42 <dbl>, X43 <dbl>, X44 <dbl>, X45 <dbl>, X46 <dbl>,
## #   X47 <dbl>, X48 <dbl>, X49 <dbl>, X50 <dbl>, X51 <dbl>, X52 <dbl>, ...
```

```
dim(df)
```

```
## [1] 6819   96
```

Data Cleaning

Rename outcome variable from "Bankrupt." to "bankrupt" and others into variable names. This will help for fluency and to avoid potential difficulties later on in EDA and modeling.
Decided that just having no columns names is better because the variable names are so long such as "RoaCBeforeInterestAndDepreciationBeforeInterest". Reference the codebook to interpret what the variables actually mean represent.

```
#df <- clean_names(df, case='upper_camel')
colnames(df)[1] <- "bankrupt"
head(df, 3)
```

```
## # A tibble: 3 x 96
##   bankrupt    X2    X3    X4    X5    X6    X7    X8    X9   X10   X11      X12
##      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1        1 0.371 0.424 0.406 0.601 0.601 0.999 0.797 0.809 0.303 0.781 0.000126
## 2        1 0.464 0.538 0.517 0.610 0.610 0.999 0.797 0.809 0.304 0.782 0.000290
## 3        1 0.426 0.499 0.472 0.601 0.601 0.999 0.796 0.808 0.302 0.780 0.000236
## # ... with 84 more variables: X13 <dbl>, X14 <dbl>, X15 <dbl>, X16 <dbl>,
## #   X17 <dbl>, X18 <dbl>, X19 <dbl>, X20 <dbl>, X21 <dbl>, X22 <dbl>,
## #   X23 <dbl>, X24 <dbl>, X25 <dbl>, X26 <dbl>, X27 <dbl>, X28 <dbl>,
## #   X29 <dbl>, X30 <dbl>, X31 <dbl>, X32 <dbl>, X33 <dbl>, X34 <dbl>,
## #   X35 <dbl>, X36 <dbl>, X37 <dbl>, X38 <dbl>, X39 <dbl>, X40 <dbl>,
## #   X41 <dbl>, X42 <dbl>, X43 <dbl>, X44 <dbl>, X45 <dbl>, X46 <dbl>,
```

```
## #   X47 <dbl>, X48 <dbl>, X49 <dbl>, X50 <dbl>, X51 <dbl>, X52 <dbl>, ...
```

Observing variables in the data set and summary statistics

```
str(df)
```

```
## spec_tbl_df [6,819 x 96] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ bankrupt: num [1:6819] 1 1 1 1 1 1 0 0 0 0 ...
##  $ X2      : num [1:6819] 0.371 0.464 0.426 0.4 0.465 ...
##  $ X3      : num [1:6819] 0.424 0.538 0.499 0.451 0.538 ...
##  $ X4      : num [1:6819] 0.406 0.517 0.472 0.458 0.522 ...
##  $ X5      : num [1:6819] 0.601 0.61 0.601 0.584 0.599 ...
##  $ X6      : num [1:6819] 0.601 0.61 0.601 0.584 0.599 ...
##  $ X7      : num [1:6819] 0.999 0.999 0.999 0.999 0.999 ...
##  $ X8      : num [1:6819] 0.797 0.797 0.796 0.797 0.797 ...
##  $ X9      : num [1:6819] 0.809 0.809 0.808 0.809 0.809 ...
##  $ X10     : num [1:6819] 0.303 0.304 0.302 0.303 0.303 ...
##  $ X11     : num [1:6819] 0.781 0.782 0.78 0.781 0.782 ...
##  $ X12     : num [1:6819] 1.26e-04 2.90e-04 2.36e-04 1.08e-04 7.89e+09 ...
##  $ X13     : num [1:6819] 0.00 0.00 2.55e+07 0.00 0.00 0.00 7.30e+08 5.09e+07 0.00 0.00 ...
##  $ X14     : num [1:6819] 0.458 0.462 0.459 0.466 0.463 ...
##  $ X15     : num [1:6819] 0.000725 0.000647 0.00079 0.000449 0.000686 ...
##  $ X16     : num [1:6819] 0 0 0 0 0 ...
##  $ X17     : num [1:6819] 0.148 0.182 0.178 0.154 0.168 ...
##  $ X18     : num [1:6819] 0.148 0.182 0.178 0.154 0.168 ...
##  $ X19     : num [1:6819] 0.148 0.182 0.194 0.154 0.168 ...
##  $ X20     : num [1:6819] 0.169 0.209 0.181 0.194 0.213 ...
##  $ X21     : num [1:6819] 0.312 0.318 0.307 0.322 0.319 ...
##  $ X22     : num [1:6819] 0.01756 0.02114 0.00594 0.01437 0.02969 ...
##  $ X23     : num [1:6819] 0.0959 0.0937 0.0923 0.0778 0.0969 ...
##  $ X24     : num [1:6819] 0.139 0.17 0.143 0.149 0.168 ...
##  $ X25     : num [1:6819] 0.0221 0.0221 0.0228 0.022 0.0221 ...
##  $ X26     : num [1:6819] 0.848 0.848 0.848 0.848 0.848 ...
##  $ X27     : num [1:6819] 0.689 0.69 0.689 0.689 0.69 ...
##  $ X28     : num [1:6819] 0.689 0.69 0.689 0.689 0.69 ...
##  $ X29     : num [1:6819] 0.218 0.218 0.218 0.218 0.218 ...
##  $ X30     : num [1:6819] 4.98e+09 6.11e+09 7.28e+09 4.88e+09 5.51e+09 6.08e+08 5.72e+09 6.63e+09 6.8
##  $ X31     : num [1:6819] 0.000327 0.000443 0.000396 0.000382 0.000439 ...
##  $ X32     : num [1:6819] 0.263 0.265 0.264 0.263 0.265 ...
##  $ X33     : num [1:6819] 0.364 0.377 0.369 0.384 0.38 ...
##  $ X34     : num [1:6819] 0.00226 0.00602 0.01154 0.00419 0.00602 ...
##  $ X35     : num [1:6819] 0.00121 0.00404 0.00535 0.0029 0.00373 ...
##  $ X36     : num [1:6819] 0.63 0.635 0.63 0.63 0.636 ...
##  $ X37     : num [1:6819] 0.02127 0.0125 0.02125 0.00957 0.00515 ...
##  $ X38     : num [1:6819] 0.208 0.171 0.208 0.151 0.107 ...
##  $ X39     : num [1:6819] 0.792 0.829 0.792 0.849 0.893 ...
##  $ X40     : num [1:6819] 0.00502 0.00506 0.0051 0.00505 0.0053 ...
##  $ X41     : num [1:6819] 0.39 0.377 0.379 0.38 0.375 ...
##  $ X42     : num [1:6819] 0.00648 0.00584 0.00656 0.00537 0.00662 ...
##  $ X43     : num [1:6819] 0.0959 0.0937 0.0923 0.0777 0.0969 ...
##  $ X44     : num [1:6819] 0.138 0.169 0.148 0.148 0.167 ...
##  $ X45     : num [1:6819] 0.398 0.398 0.407 0.398 0.4 ...
##  $ X46     : num [1:6819] 0.087 0.0645 0.015 0.09 0.1754 ...
##  $ X47     : num [1:6819] 0.00181 0.00129 0.0015 0.00197 0.00145 ...
##  $ X48     : num [1:6819] 0.00349 0.00492 0.00423 0.00321 0.00437 ...
```

```
##  $ X49     : num [1:6819] 1.82e-04 9.36e+09 6.50e+07 7.13e+09 1.63e-04 ...
##  $ X50     : num [1:6819] 1.17e-04 7.19e+08 2.65e+09 9.15e+09 2.94e-04 ...
##  $ X51     : num [1:6819] 0.0329 0.0255 0.0134 0.0281 0.0402 ...
##  $ X52     : num [1:6819] 0.03416 0.00689 0.029 0.01546 0.05811 ...
##  $ X53     : num [1:6819] 0.393 0.392 0.382 0.378 0.394 ...
##  $ X54     : num [1:6819] 0.0371 0.0123 0.141 0.0213 0.024 ...
##  $ X55     : num [1:6819] 0.673 0.751 0.83 0.726 0.752 ...
##  $ X56     : num [1:6819] 0.167 0.127 0.34 0.162 0.26 ...
##  $ X57     : num [1:6819] 0.191 0.182 0.603 0.226 0.358 ...
##  $ X58     : num [1:6819] 0.004094 0.014948 0.000991 0.018851 0.014161 ...
##  $ X59     : num [1:6819] 0.002 0.00414 0.0063 0.00296 0.00427 ...
##  $ X60     : num [1:6819] 1.47e-04 1.38e-03 5.34e+09 1.01e-03 6.80e-04 ...
##  $ X61     : num [1:6819] 0.1473 0.057 0.0982 0.0987 0.1102 ...
##  $ X62     : num [1:6819] 0.334 0.341 0.337 0.349 0.345 ...
##  $ X63     : num [1:6819] 0.277 0.29 0.277 0.277 0.288 ...
##  $ X64     : num [1:6819] 0.00104 0.00521 0.01388 0.00354 0.00487 ...
##  $ X65     : num [1:6819] 0.676 0.309 0.446 0.616 0.975 ...
##  $ X66     : num [1:6819] 0.721 0.732 0.743 0.73 0.732 ...
##  $ X67     : num [1:6819] 0.339 0.33 0.335 0.332 0.331 ...
##  $ X68     : num [1:6819] 0.02559 0.02395 0.00372 0.02217 0 ...
##  $ X69     : num [1:6819] 0.903 0.931 0.91 0.907 0.914 ...
##  $ X70     : num [1:6819] 0.00202 0.00223 0.00206 0.00183 0.00222 ...
##  $ X71     : num [1:6819] 0.0649 0.0255 0.0214 0.0242 0.0264 ...
##  $ X72     : num [1:6819] 7.01e+08 1.07e-04 1.79e-03 8.14e+09 6.68e+09 ...
##  $ X73     : num [1:6819] 6.55e+09 7.70e+09 1.02e-03 6.05e+09 5.05e+09 ...
##  $ X74     : num [1:6819] 0.594 0.594 0.595 0.594 0.594 ...
##  $ X75     : num [1:6819] 4.58e+08 2.49e+09 7.61e+08 2.03e+09 8.24e+08 ...
##  $ X76     : num [1:6819] 0.672 0.672 0.672 0.672 0.672 ...
##  $ X77     : num [1:6819] 0.424 0.469 0.276 0.559 0.31 ...
##  $ X78     : num [1:6819] 0.676 0.309 0.446 0.616 0.975 ...
##  $ X79     : num [1:6819] 0.339 0.33 0.335 0.332 0.331 ...
##  $ X80     : num [1:6819] 0.127 0.121 0.118 0.121 0.111 ...
##  $ X81     : num [1:6819] 0.638 0.641 0.643 0.579 0.622 ...
##  $ X82     : num [1:6819] 0.459 0.459 0.459 0.449 0.454 ...
##  $ X83     : num [1:6819] 0.52 0.567 0.538 0.604 0.578 ...
##  $ X84     : num [1:6819] 0.313 0.314 0.315 0.302 0.312 ...
##  $ X85     : num [1:6819] 0.1183 0.0478 0.0253 0.0672 0.0477 ...
##  $ X86     : num [1:6819] 0 0 0 0 0 0 0 0 0 0 ...
##  $ X87     : num [1:6819] 0.717 0.795 0.775 0.74 0.795 ...
##  $ X88     : num [1:6819] 0.00922 0.00832 0.04 0.00325 0.00388 ...
##  $ X89     : num [1:6819] 0.623 0.624 0.624 0.623 0.624 ...
##  $ X90     : num [1:6819] 0.601 0.61 0.601 0.584 0.599 ...
##  $ X91     : num [1:6819] 0.828 0.84 0.837 0.835 0.84 ...
##  $ X92     : num [1:6819] 0.29 0.284 0.29 0.282 0.279 ...
##  $ X93     : num [1:6819] 0.0266 0.2646 0.0266 0.0267 0.0248 ...
##  $ X94     : num [1:6819] 0.564 0.57 0.564 0.565 0.576 ...
##  $ X95     : num [1:6819] 1 1 1 1 1 1 1 1 1 1 ...
##  $ X96     : num [1:6819] 0.0165 0.0208 0.0165 0.024 0.0355 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   X1 = col_double(),
##   ..   X2 = col_double(),
##   ..   X3 = col_double(),
##   ..   X4 = col_double(),
```

```
##   ..   X5 = col_double(),
##   ..   X6 = col_double(),
##   ..   X7 = col_double(),
##   ..   X8 = col_double(),
##   ..   X9 = col_double(),
##   ..   X10 = col_double(),
##   ..   X11 = col_double(),
##   ..   X12 = col_double(),
##   ..   X13 = col_double(),
##   ..   X14 = col_double(),
##   ..   X15 = col_double(),
##   ..   X16 = col_double(),
##   ..   X17 = col_double(),
##   ..   X18 = col_double(),
##   ..   X19 = col_double(),
##   ..   X20 = col_double(),
##   ..   X21 = col_double(),
##   ..   X22 = col_double(),
##   ..   X23 = col_double(),
##   ..   X24 = col_double(),
##   ..   X25 = col_double(),
##   ..   X26 = col_double(),
##   ..   X27 = col_double(),
##   ..   X28 = col_double(),
##   ..   X29 = col_double(),
##   ..   X30 = col_double(),
##   ..   X31 = col_double(),
##   ..   X32 = col_double(),
##   ..   X33 = col_double(),
##   ..   X34 = col_double(),
##   ..   X35 = col_double(),
##   ..   X36 = col_double(),
##   ..   X37 = col_double(),
##   ..   X38 = col_double(),
##   ..   X39 = col_double(),
##   ..   X40 = col_double(),
##   ..   X41 = col_double(),
##   ..   X42 = col_double(),
##   ..   X43 = col_double(),
##   ..   X44 = col_double(),
##   ..   X45 = col_double(),
##   ..   X46 = col_double(),
##   ..   X47 = col_double(),
##   ..   X48 = col_double(),
##   ..   X49 = col_double(),
##   ..   X50 = col_double(),
##   ..   X51 = col_double(),
##   ..   X52 = col_double(),
##   ..   X53 = col_double(),
##   ..   X54 = col_double(),
##   ..   X55 = col_double(),
##   ..   X56 = col_double(),
##   ..   X57 = col_double(),
##   ..   X58 = col_double(),
```

```
##   ..   X59 = col_double(),
##   ..   X60 = col_double(),
##   ..   X61 = col_double(),
##   ..   X62 = col_double(),
##   ..   X63 = col_double(),
##   ..   X64 = col_double(),
##   ..   X65 = col_double(),
##   ..   X66 = col_double(),
##   ..   X67 = col_double(),
##   ..   X68 = col_double(),
##   ..   X69 = col_double(),
##   ..   X70 = col_double(),
##   ..   X71 = col_double(),
##   ..   X72 = col_double(),
##   ..   X73 = col_double(),
##   ..   X74 = col_double(),
##   ..   X75 = col_double(),
##   ..   X76 = col_double(),
##   ..   X77 = col_double(),
##   ..   X78 = col_double(),
##   ..   X79 = col_double(),
##   ..   X80 = col_double(),
##   ..   X81 = col_double(),
##   ..   X82 = col_double(),
##   ..   X83 = col_double(),
##   ..   X84 = col_double(),
##   ..   X85 = col_double(),
##   ..   X86 = col_double(),
##   ..   X87 = col_double(),
##   ..   X88 = col_double(),
##   ..   X89 = col_double(),
##   ..   X90 = col_double(),
##   ..   X91 = col_double(),
##   ..   X92 = col_double(),
##   ..   X93 = col_double(),
##   ..   X94 = col_double(),
##   ..   X95 = col_double(),
##   ..   X96 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
summary(df)
```

```
##     bankrupt             X2               X3               X4
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.4765   1st Qu.:0.5355   1st Qu.:0.5273
##  Median :0.00000   Median :0.5027   Median :0.5598   Median :0.5523
##  Mean   :0.03226   Mean   :0.5052   Mean   :0.5586   Mean   :0.5536
##  3rd Qu.:0.00000   3rd Qu.:0.5356   3rd Qu.:0.5892   3rd Qu.:0.5841
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##        X5               X6               X7               X8
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.6004   1st Qu.:0.6004   1st Qu.:0.9990   1st Qu.:0.7974
##  Median :0.6060   Median :0.6060   Median :0.9990   Median :0.7975
##  Mean   :0.6079   Mean   :0.6079   Mean   :0.9988   Mean   :0.7972
```

```
##  3rd Qu.:0.6139   3rd Qu.:0.6138   3rd Qu.:0.9991   3rd Qu.:0.7976
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##        X9              X10              X11              X12
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000e+00
##  1st Qu.:0.8093   1st Qu.:0.3035   1st Qu.:0.7816   1st Qu.:0.000e+00
##  Median :0.8094   Median :0.3035   Median :0.7816   Median :0.000e+00
##  Mean   :0.8091   Mean   :0.3036   Mean   :0.7814   Mean   :1.995e+09
##  3rd Qu.:0.8095   3rd Qu.:0.3036   3rd Qu.:0.7817   3rd Qu.:4.145e+09
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :9.990e+09
##       X13               X14              X15                X16
##  Min.   :0.00e+00   Min.   :0.0000   Min.   :        0   Min.   :0.00000
##  1st Qu.:0.00e+00   1st Qu.:0.4616   1st Qu.:        0   1st Qu.:0.00000
##  Median :5.09e+08   Median :0.4651   Median :        0   Median :0.07349
##  Mean   :1.95e+09   Mean   :0.4674   Mean   : 16448013   Mean   :0.11500
##  3rd Qu.:3.45e+09   3rd Qu.:0.4710   3rd Qu.:        0   3rd Qu.:0.20584
##  Max.   :9.98e+09   Max.   :1.0000   Max.   :990000000   Max.   :1.00000
##       X17              X18              X19              X20
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1736   1st Qu.:0.1736   1st Qu.:0.1737   1st Qu.:0.2147
##  Median :0.1844   Median :0.1844   Median :0.1844   Median :0.2245
##  Mean   :0.1907   Mean   :0.1906   Mean   :0.1907   Mean   :0.2288
##  3rd Qu.:0.1996   3rd Qu.:0.1996   3rd Qu.:0.1996   3rd Qu.:0.2388
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       X21              X22                X23              X24
##  Min.   :0.0000   Min.   :0.000e+00   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.3177   1st Qu.:0.000e+00   1st Qu.:0.09608   1st Qu.:0.1704
##  Median :0.3225   Median :0.000e+00   Median :0.10423   Median :0.1797
##  Mean   :0.3235   Mean   :1.329e+06   Mean   :0.10909   Mean   :0.1844
##  3rd Qu.:0.3286   3rd Qu.:0.000e+00   3rd Qu.:0.11615   3rd Qu.:0.1935
##  Max.   :1.0000   Max.   :3.020e+09   Max.   :1.00000   Max.   :1.0000
##       X25              X26              X27              X28
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.02206   1st Qu.:0.8480   1st Qu.:0.6893   1st Qu.:0.6893
##  Median :0.02210   Median :0.8480   Median :0.6894   Median :0.6894
##  Mean   :0.02241   Mean   :0.8480   Mean   :0.6891   Mean   :0.6892
##  3rd Qu.:0.02215   3rd Qu.:0.8481   3rd Qu.:0.6896   3rd Qu.:0.6896
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       X29              X30                X31              X32
##  Min.   :0.0000   Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.0000
##  1st Qu.:0.2176   1st Qu.:4.860e+09   1st Qu.:0.000e+00   1st Qu.:0.2638
##  Median :0.2176   Median :6.400e+09   Median :0.000e+00   Median :0.2640
##  Mean   :0.2176   Mean   :5.508e+09   Mean   :1.566e+06   Mean   :0.2642
##  3rd Qu.:0.2176   3rd Qu.:7.390e+09   3rd Qu.:0.000e+00   3rd Qu.:0.2644
##  Max.   :1.0000   Max.   :9.990e+09   Max.   :9.330e+09   Max.   :1.0000
##       X33              X34                X35              X36
##  Min.   :0.0000   Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.0000
##  1st Qu.:0.3747   1st Qu.:0.000e+00   1st Qu.:0.000e+00   1st Qu.:0.6306
##  Median :0.3804   Median :0.000e+00   Median :0.000e+00   Median :0.6307
##  Mean   :0.3797   Mean   :4.033e+05   Mean   :8.377e+06   Mean   :0.6310
##  3rd Qu.:0.3867   3rd Qu.:0.000e+00   3rd Qu.:0.000e+00   3rd Qu.:0.6311
##  Max.   :1.0000   Max.   :2.750e+09   Max.   :9.230e+09   Max.   :1.0000
##       X37                X38              X39              X40
##  Min.   :0.000e+00   Min.   :0.00000   Min.   :0.0000   Min.   :0.000000
##  1st Qu.:0.000e+00   1st Qu.:0.07289   1st Qu.:0.8512   1st Qu.:0.005244
```

```
##  Median :0.000e+00   Median :0.11141   Median :0.8886   Median :0.005665
##  Mean   :4.416e+06   Mean   :0.11318   Mean   :0.8868   Mean   :0.008783
##  3rd Qu.:0.000e+00   3rd Qu.:0.14880   3rd Qu.:0.9271   3rd Qu.:0.006847
##  Max.   :9.940e+09   Max.   :1.00000   Max.   :1.0000   Max.   :1.000000
##       X41               X42               X43               X44
##  Min.   :0.0000   Min.   :0.000000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3702   1st Qu.:0.005366   1st Qu.:0.0961   1st Qu.:0.1694
##  Median :0.3726   Median :0.005366   Median :0.1041   Median :0.1785
##  Mean   :0.3747   Mean   :0.005968   Mean   :0.1090   Mean   :0.1827
##  3rd Qu.:0.3763   3rd Qu.:0.005764   3rd Qu.:0.1159   3rd Qu.:0.1916
##  Max.   :1.0000   Max.   :1.000000   Max.   :1.0000   Max.   :1.0000
##       X45               X46               X47                X48
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.000e+00   Min.   :0.000e+00
##  1st Qu.:0.3974   1st Qu.:0.07646   1st Qu.:0.000e+00   1st Qu.:0.000e+00
##  Median :0.4001   Median :0.11844   Median :0.000e+00   Median :0.000e+00
##  Mean   :0.4025   Mean   :0.14161   Mean   :1.279e+07   Mean   :9.826e+06
##  3rd Qu.:0.4046   3rd Qu.:0.17691   3rd Qu.:0.000e+00   3rd Qu.:0.000e+00
##  Max.   :1.0000   Max.   :1.00000   Max.   :9.740e+09   Max.   :9.730e+09
##       X49                X50                X51               X52
##  Min.   :0.000e+00   Min.   :0.000e+00   Min.   :0.00000   Min.   :0.000e+00
##  1st Qu.:0.000e+00   1st Qu.:0.000e+00   1st Qu.:0.02177   1st Qu.:0.000e+00
##  Median :0.000e+00   Median :0.000e+00   Median :0.02952   Median :0.000e+00
##  Mean   :2.149e+09   Mean   :1.009e+09   Mean   :0.03860   Mean   :2.326e+06
##  3rd Qu.:4.620e+09   3rd Qu.:0.000e+00   3rd Qu.:0.04290   3rd Qu.:0.000e+00
##  Max.   :9.990e+09   Max.   :9.990e+09   Max.   :1.00000   Max.   :8.810e+09
##       X53               X54                X55               X56
##  Min.   :0.0000   Min.   :0.000e+00   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3924   1st Qu.:0.000e+00   1st Qu.:0.7743   1st Qu.:0.2420
##  Median :0.3959   Median :0.000e+00   Median :0.8103   Median :0.3865
##  Mean   :0.4007   Mean   :1.126e+07   Mean   :0.8141   Mean   :0.4001
##  3rd Qu.:0.4019   3rd Qu.:0.000e+00   3rd Qu.:0.8504   3rd Qu.:0.5406
##  Max.   :1.0000   Max.   :9.570e+09   Max.   :1.0000   Max.   :1.0000
##       X57               X58                X59                X60
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.000e+00   Min.   :0.000e+00
##  1st Qu.:0.3528   1st Qu.:0.03354   1st Qu.:0.000e+00   1st Qu.:0.000e+00
##  Median :0.5148   Median :0.07489   Median :0.000e+00   Median :0.000e+00
##  Mean   :0.5223   Mean   :0.12409   Mean   :3.593e+06   Mean   :3.716e+07
##  3rd Qu.:0.6891   3rd Qu.:0.16107   3rd Qu.:0.000e+00   3rd Qu.:0.000e+00
##  Max.   :1.0000   Max.   :1.00000   Max.   :8.820e+09   Max.   :9.650e+09
##       X61               X62               X63               X64
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000e+00
##  1st Qu.:0.05330   1st Qu.:0.3410   1st Qu.:0.2770   1st Qu.:0.000e+00
##  Median :0.08270   Median :0.3486   Median :0.2772   Median :0.000e+00
##  Mean   :0.09067   Mean   :0.3538   Mean   :0.2774   Mean   :5.581e+07
##  3rd Qu.:0.11952   3rd Qu.:0.3609   3rd Qu.:0.2774   3rd Qu.:0.000e+00
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :9.910e+09
##       X65               X66               X67               X68
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000e+00
##  1st Qu.:0.6270   1st Qu.:0.7336   1st Qu.:0.3281   1st Qu.:0.000e+00
##  Median :0.8069   Median :0.7360   Median :0.3297   Median :0.000e+00
##  Mean   :0.7616   Mean   :0.7358   Mean   :0.3314   Mean   :5.416e+07
##  3rd Qu.:0.9420   3rd Qu.:0.7386   3rd Qu.:0.3323   3rd Qu.:0.000e+00
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :9.540e+09
##       X69               X70                X71               X72
```

```
##  Min.   :0.0000   Min.   :0.000000   Min.   :0.00000   Min.   :0.000e+00
##  1st Qu.:0.9311   1st Qu.:0.002236   1st Qu.:0.01457   1st Qu.:0.000e+00
##  Median :0.9377   Median :0.002336   Median :0.02267   Median :0.000e+00
##  Mean   :0.9347   Mean   :0.002549   Mean   :0.02918   Mean   :1.196e+09
##  3rd Qu.:0.9448   3rd Qu.:0.002492   3rd Qu.:0.03593   3rd Qu.:0.000e+00
##  Max.   :1.0000   Max.   :1.000000   Max.   :1.00000   Max.   :1.000e+10
##       X73                X74             X75                X76
##  Min.   :0.000e+00   Min.   :0.0000   Min.   :0.000e+00   Min.   :0.0000
##  1st Qu.:0.000e+00   1st Qu.:0.5939   1st Qu.:0.000e+00   1st Qu.:0.6716
##  Median :0.000e+00   Median :0.5940   Median :1.080e+09   Median :0.6716
##  Mean   :2.164e+09   Mean   :0.5940   Mean   :2.472e+09   Mean   :0.6715
##  3rd Qu.:4.900e+09   3rd Qu.:0.5940   3rd Qu.:4.510e+09   3rd Qu.:0.6716
##  Max.   :1.000e+10   Max.   :1.0000   Max.   :1.000e+10   Max.   :1.0000
##       X77               X78              X79              X80
##  Min.   :0.00e+00   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00e+00   1st Qu.:0.6270   1st Qu.:0.3281   1st Qu.:0.1109
##  Median :0.00e+00   Median :0.8069   Median :0.3297   Median :0.1123
##  Mean   :1.22e+06   Mean   :0.7616   Mean   :0.3314   Mean   :0.1156
##  3rd Qu.:0.00e+00   3rd Qu.:0.9420   3rd Qu.:0.3323   3rd Qu.:0.1171
##  Max.   :8.32e+09   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       X81              X82              X83              X84
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.6333   1st Qu.:0.4571   1st Qu.:0.5660   1st Qu.:0.3130
##  Median :0.6454   Median :0.4598   Median :0.5933   Median :0.3150
##  Mean   :0.6497   Mean   :0.4618   Mean   :0.5934   Mean   :0.3156
##  3rd Qu.:0.6631   3rd Qu.:0.4642   3rd Qu.:0.6248   3rd Qu.:0.3177
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       X85               X86               X87              X88
##  Min.   :0.00000   Min.   :0.000000   Min.   :0.0000   Min.   :0.000e+00
##  1st Qu.:0.01803   1st Qu.:0.000000   1st Qu.:0.7967   1st Qu.:0.000e+00
##  Median :0.02760   Median :0.000000   Median :0.8106   Median :0.000e+00
##  Mean   :0.03151   Mean   :0.001173   Mean   :0.8078   Mean   :1.863e+07
##  3rd Qu.:0.03837   3rd Qu.:0.000000   3rd Qu.:0.8265   3rd Qu.:0.000e+00
##  Max.   :1.00000   Max.   :1.000000   Max.   :1.0000   Max.   :9.820e+09
##       X89              X90              X91              X92
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.6236   1st Qu.:0.6004   1st Qu.:0.8401   1st Qu.:0.2769
##  Median :0.6239   Median :0.6060   Median :0.8412   Median :0.2788
##  Mean   :0.6239   Mean   :0.6079   Mean   :0.8404   Mean   :0.2804
##  3rd Qu.:0.6242   3rd Qu.:0.6139   3rd Qu.:0.8424   3rd Qu.:0.2814
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       X93               X94              X95          X96
##  Min.   :0.00000   Min.   :0.0000   Min.   :1   Min.   :0.00000
##  1st Qu.:0.02679   1st Qu.:0.5652   1st Qu.:1   1st Qu.:0.02448
##  Median :0.02681   Median :0.5653   Median :1   Median :0.03380
##  Mean   :0.02754   Mean   :0.5654   Mean   :1   Mean   :0.04758
##  3rd Qu.:0.02691   3rd Qu.:0.5657   3rd Qu.:1   3rd Qu.:0.05284
##  Max.   :1.00000   Max.   :1.0000   Max.   :1   Max.   :1.00000
```

```
#describe(df)
```

Looking deeper into the columns, it is evident that all columns are numerical values and thus do not need to convert any categorical variables into dummy variables. Thus, we move forward and observe the presence of any null values.

```
dim(df)
```

```
## [1] 6819   96
```

```
sum(is.na(df))
```

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

```
sum(rowSums(is.na(df) | df == ""))
```

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

Evidently, there are no null values. Otherwise, I would consider dropping rows given enough data, or filling in values based on a measure of central tendency in the columns. We can move forward with checking duplicate values as well.

```
#No duplicate values
sum(duplicated(df))
```

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

Looking at the number of unique values in the data frame. Noticeably, Net.Income.Flag only has 1 unique value which will not help for distinguishing classes. Thus, I chose to remove Net.Income.Flag (X95) as a predictor.

```
sapply(df, function(x) n_distinct(x))
```

```
## bankrupt       X2       X3       X4       X5       X6       X7       X8
##        2     3333     3151     3160     3781     3788     3376     3789
##       X9      X10      X11      X12      X13      X14      X15      X16
##     3604     2551     3617     2966     1536     5557     1080     2488
##      X17      X18      X19      X20      X21      X22      X23      X24
##     2278     2285     2284     1358     1545     3807     1236     1522
##      X25      X26      X27      X28      X29      X30      X31      X32
##     5583     6249     6246     6253     6270     1751     4502     2903
##      X33      X34      X35      X36      X37      X38      X39      X40
##     3599     6132     6094     3794     5518     4208     4208     6523
##      X41      X42      X43      X44      X45      X46      X47      X48
##     4338     1855     4423     4785     5289      381     1593     5451
##      X49      X50      X51      X52      X53      X54      X55      X56
##     2397     2451      741     5667     3023     6768     6819     6819
##      X57      X58      X59      X60      X61      X62      X63      X64
##     6819     6819     6819     6816     6819     6819     6593     6590
##      X65      X66      X67      X68      X69      X70      X71      X72
##     6627     6819     6819     4249     6819     6819     6819     6261
##      X73      X74      X75      X76      X77      X78      X79      X80
##     5377     6819     4023     6819     6814     6627     6819     4251
##      X81      X82      X83      X84      X85      X86      X87      X88
##     6819     6819     6819     6819     6819        2     6819     6818
##      X89      X90      X91      X92      X93      X94      X95      X96
##     6819     6816     6819     6819     6240     6240        1     6819
```

Building off the previous logic, I remove all data with little to no variation which is analogous to high correlation between variables.

```
no_var <- nearZeroVar(df)
no_var[!no_var %in% 1]
```

```
## [1] 86 95
```

```
df <- df[ , -no_var[!no_var %in% 1]]
dim(df)
```

```
## [1] 6819   94
```

When doing a classification problem, it is always important to look at the distribution of classes and see how that may influence our model. It seems there is a class imbalance and will look deeper during EDA phase.

```
dplyr::count(df, bankrupt, sort = TRUE)
```

```
## # A tibble: 2 x 2
##   bankrupt     n
##      <dbl> <int>
## 1        0  6599
## 2        1   220
```

EDA

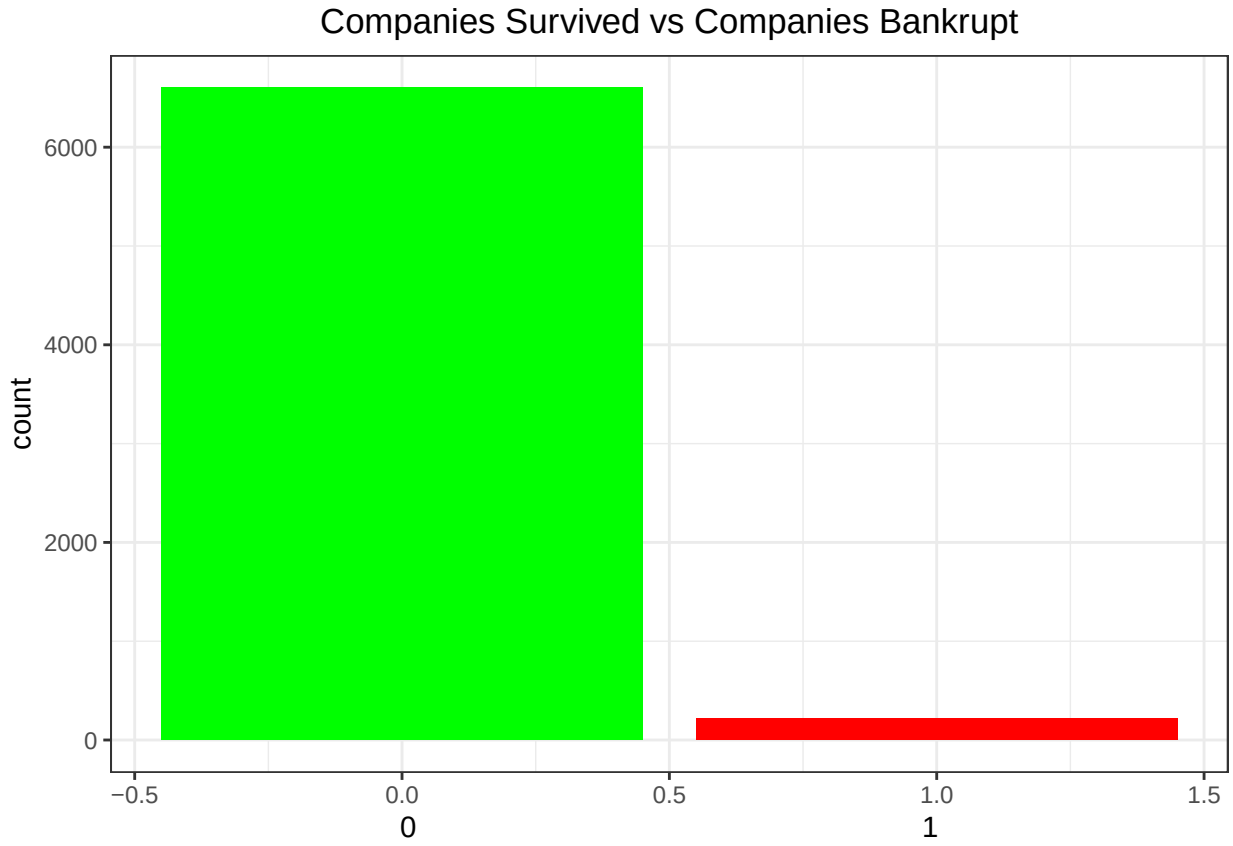
My exploratory data analysis will serve to not only understand the predictors better, but help visualize the distribution of variables that may not seem transparent initially.
More specifically, searching for variation and relationships between variables to tell a story.
I hope to dive deeper into the data to help maximize efficiency when running my models. First, visualize the distribution of classes as left off in the last phase.

```
colnames(df)
```

```
##  [1] "bankrupt" "X2"       "X3"       "X4"       "X5"       "X6"
##  [7] "X7"       "X8"       "X9"       "X10"      "X11"      "X12"
## [13] "X13"      "X14"      "X15"      "X16"      "X17"      "X18"
## [19] "X19"      "X20"      "X21"      "X22"      "X23"      "X24"
## [25] "X25"      "X26"      "X27"      "X28"      "X29"      "X30"
## [31] "X31"      "X32"      "X33"      "X34"      "X35"      "X36"
## [37] "X37"      "X38"      "X39"      "X40"      "X41"      "X42"
## [43] "X43"      "X44"      "X45"      "X46"      "X47"      "X48"
## [49] "X49"      "X50"      "X51"      "X52"      "X53"      "X54"
## [55] "X55"      "X56"      "X57"      "X58"      "X59"      "X60"
## [61] "X61"      "X62"      "X63"      "X64"      "X65"      "X66"
## [67] "X67"      "X68"      "X69"      "X70"      "X71"      "X72"
## [73] "X73"      "X74"      "X75"      "X76"      "X77"      "X78"
## [79] "X79"      "X80"      "X81"      "X82"      "X83"      "X84"
## [85] "X85"      "X87"      "X88"      "X89"      "X90"      "X91"
## [91] "X92"      "X93"      "X94"      "X96"
```

Visualize the distribution of the outcome variable and examine the level of class imbalance. Exploring the 'bankrupt' count and percentage distributions.

```
ggplot(df, aes(bankrupt)) +
  geom_bar(fill = c("Green","Red")) +
  theme_bw() +
  ggtitle("Companies Survived vs Companies Bankrupt") +
  theme(plot.title = element_text(hjust = 0.5)) +
  xlab("0                                                                    1")
```




```
#Show relevant counts and percentage of classes
table(df$bankrupt)
```

```
##
##    0    1
## 6599  220
```

```
prop.table(table(df$bankrupt))
```

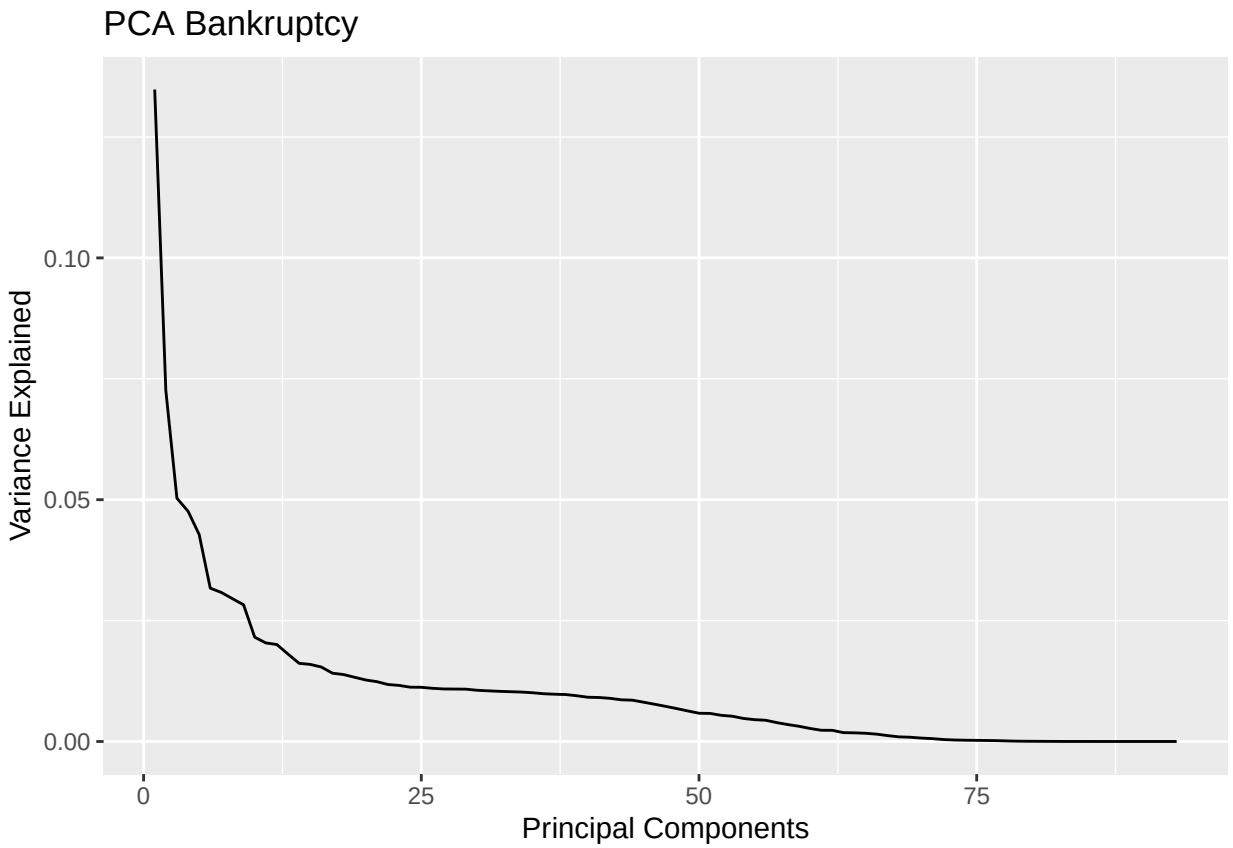
```
##
##         0         1
## 0.9677372 0.0322628
```

From the results, class 0 consists of approximately 96.7% of the observations while class 1 consists of approximately 3.3% of the observations. There is a class imbalance which produces a general risk model predictability, but also of over-fitting and inaccuracy. Intuitively, it makes sense that more companies survive bankruptcy as opposed to going bankrupt.

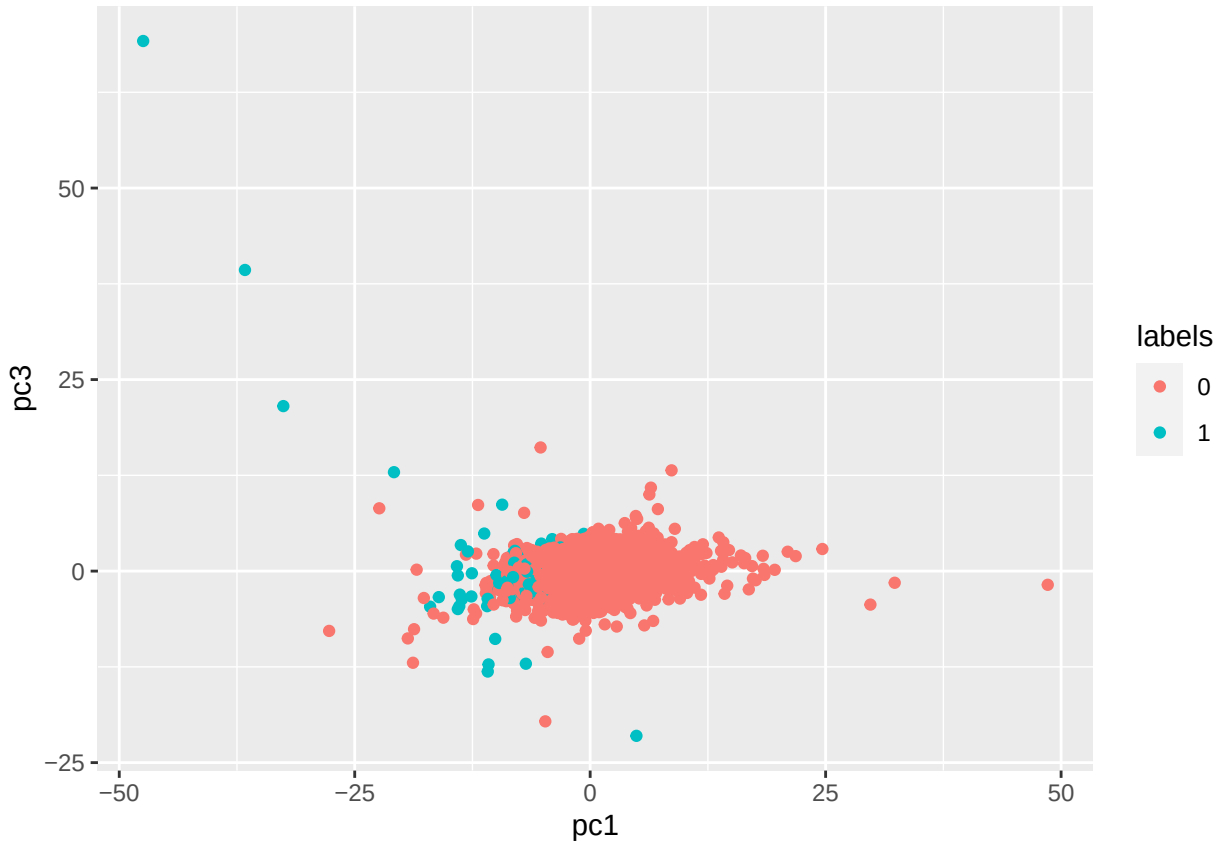
Visualization in Principal Component Analysis

```
pca1 <- prcomp(df[, 2:94], center=T, scale = TRUE)

total_var <- sum(pca1$sdev^2)
var_explained <- data.frame(pc = seq(1:93), var_explained  = pca1$sdev^2 / total_var )

ggplot(var_explained, aes(pc, var_explained)) +
  geom_line() +
  xlab("Principal Components") +
  ylab("Variance Explained") +
```

```
  ggtitle("PCA Bankruptcy")
```



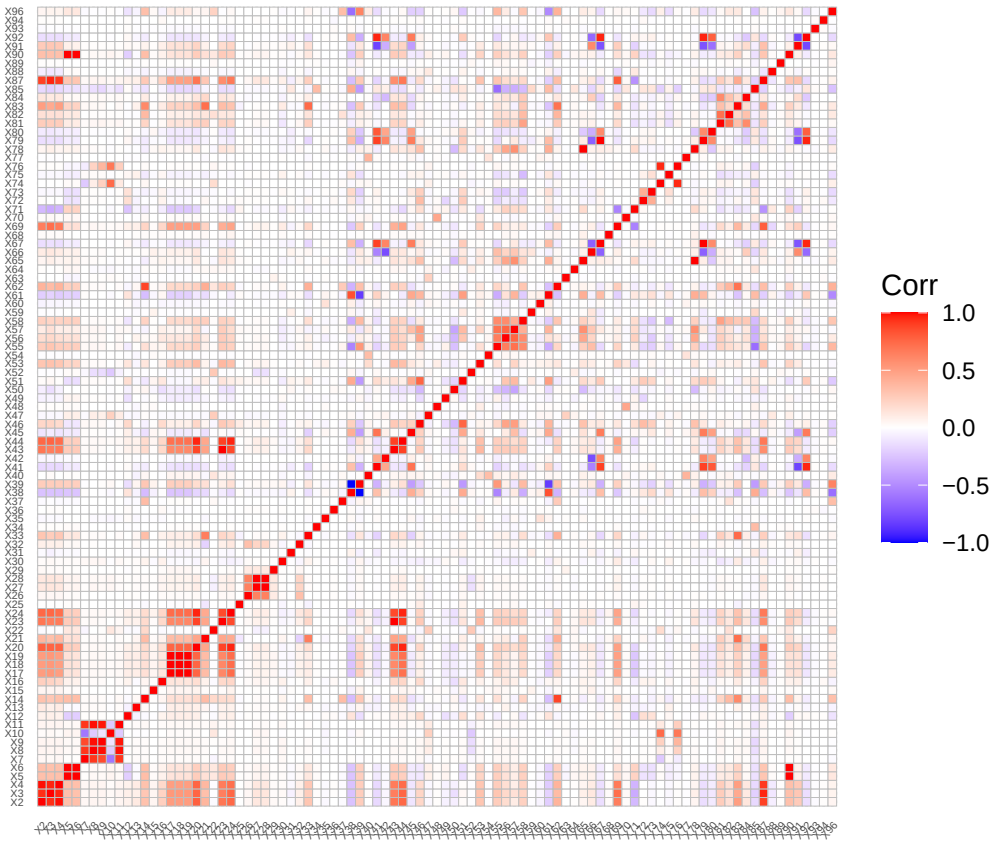

```
data_df <- data.frame(pc1 = pca1$x[, 1], pc2 = pca1$x[, 2], pc3 = pca1$x[, 3], labels = as.factor(df$ba

ggplot(data_df, aes(pc1, pc3, col = labels)) +
  geom_point()
```



These PCA plots simply help visualize the amount of explained variance in principal components and if it is required. First plot communicates the number of explained variances by principal components. The second plot compares principal component 1 and principal component 3.

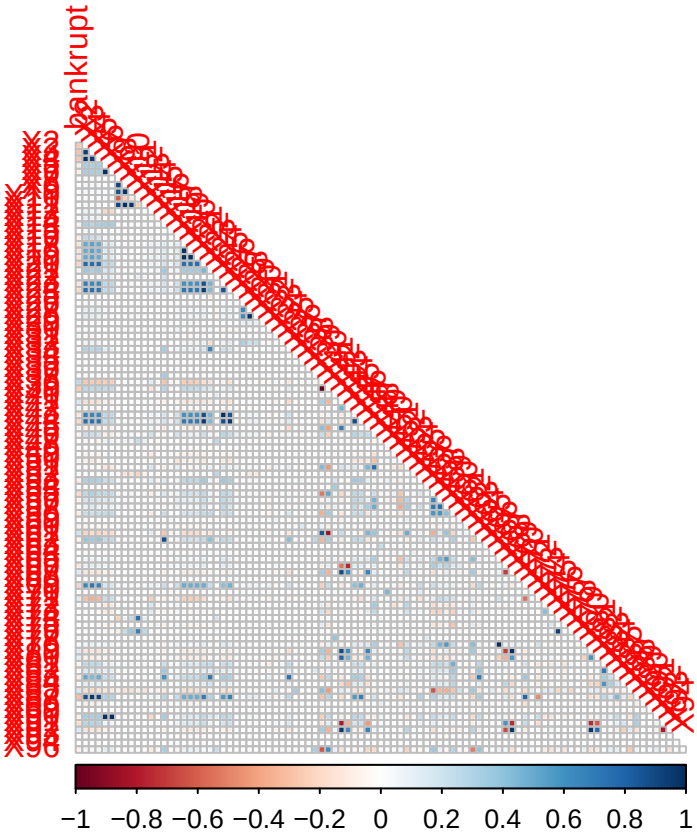
Correlation Matrix

```
library(ggcorrplot)
bank_corr <- cor(df[,-1])
ggcorrplot(bank_corr, tl.cex = 4, tl.srt = 50)
```



Visibly, X44 and X23, X24 have high positive correlation which indicates potential to drop due to similar variables. Moreover, X17 and X18, X19, X20.
Alternatively,

```
df %>%
  select(where(is.numeric)) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



Looking at the matrices, we can try to remove a lot of the highly correlated features that essentially provide the same information to the model's learning and will only introduce more noise. Thus, we remove those predictors with a correlation of higher than 75%. Although this is arbitrarily chosen, it communicates the number of predictors with high correlation; typically a threshold of above 50%.

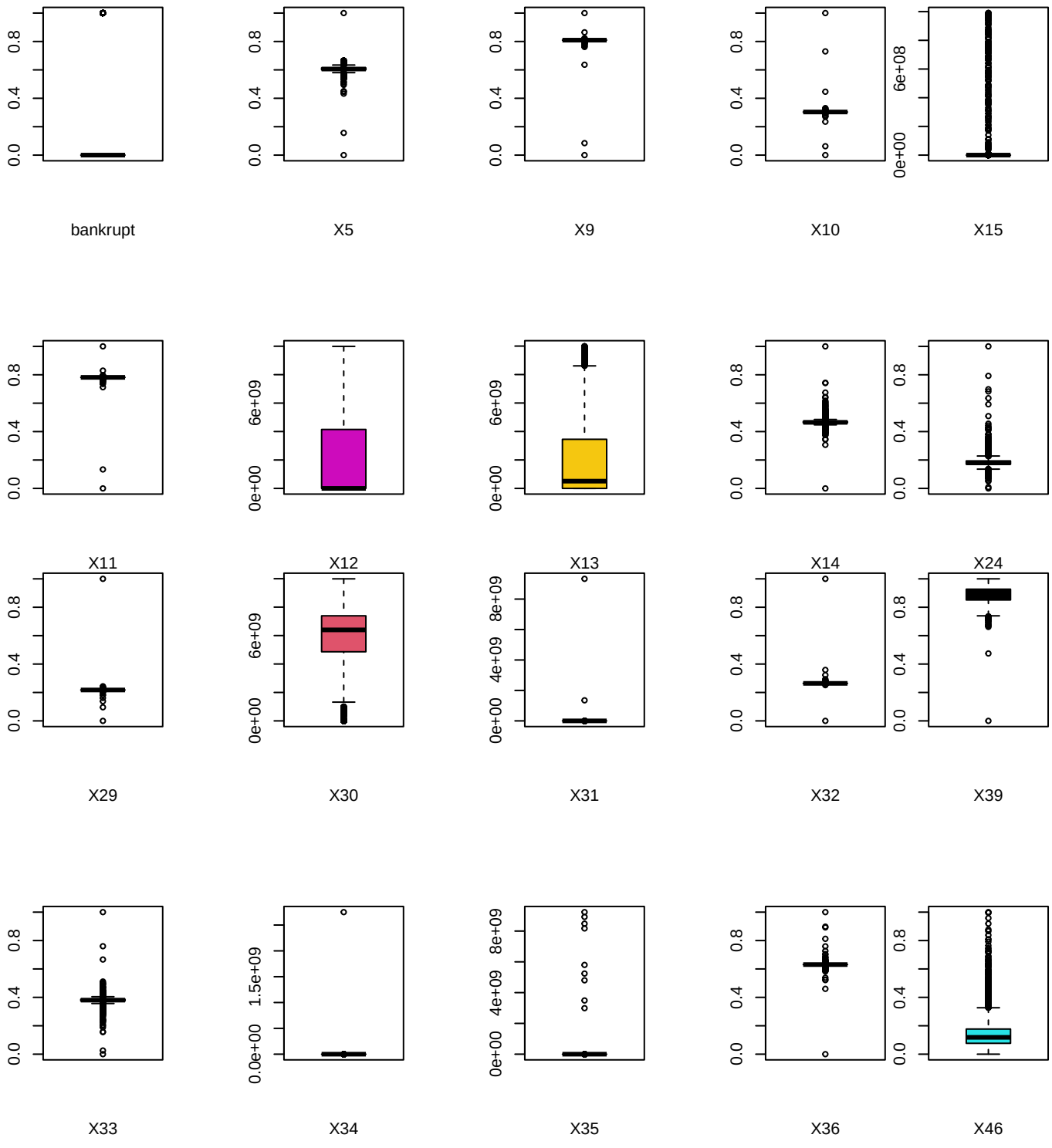
```
cor_df <- cor(df[,-1])

high_corr <- findCorrelation(cor_df, cutoff = .75)
high_corr[!high_corr %in% 1]
```

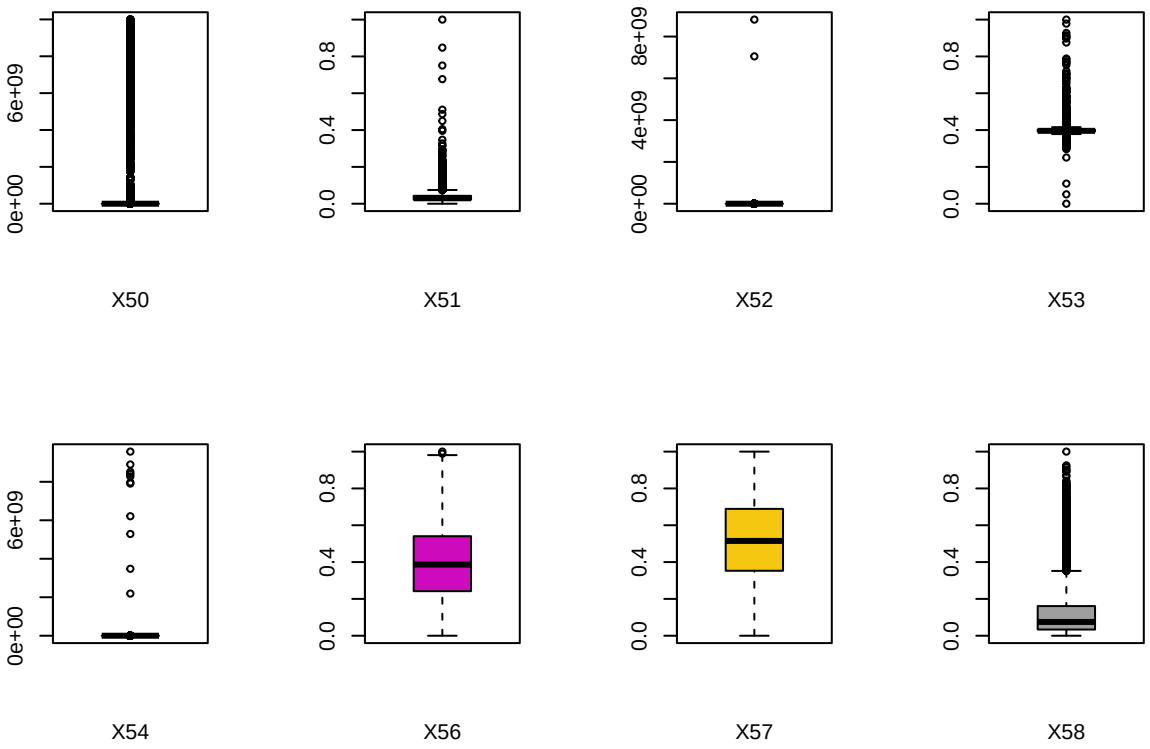
```
##  [1]  2  3 85 19 23 43 22 17 16 37 38 61 90 40 66 88  4 55 45 65 64  8  7  6 27
## [26] 75
```

```
df <- df[,-high_corr[!high_corr %in% 1]]
```

Explore the statistics of some given variables

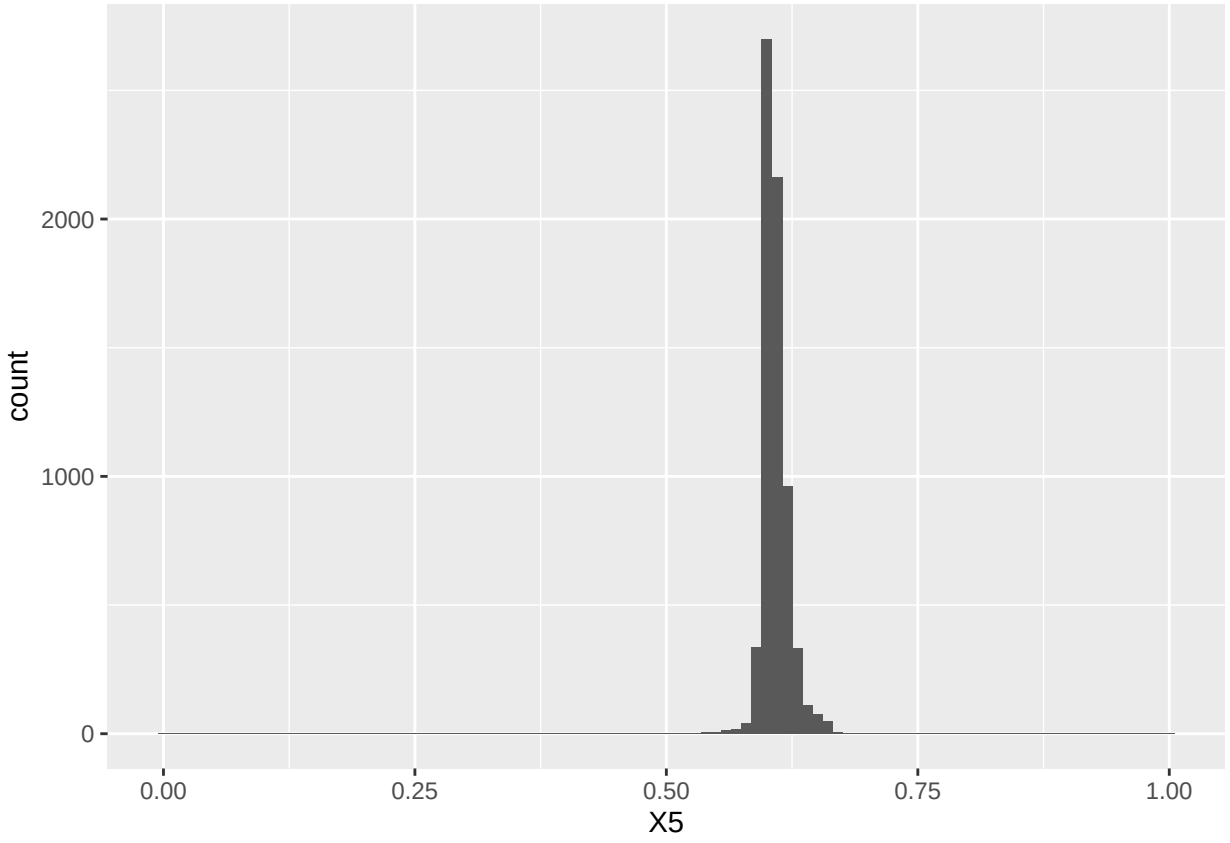
```
#Boxplots
par(mfrow = c(2,4))
for (i in seq(10:49))
  boxplot(df[,i],xlab=colnames(df)[i],col=i)
```


bankrupt
X5
X9
X10
X15
X11
X12
X13
X14
X24
X29
X30
X31
X32
X39
X33
X34
X35
X36
X46

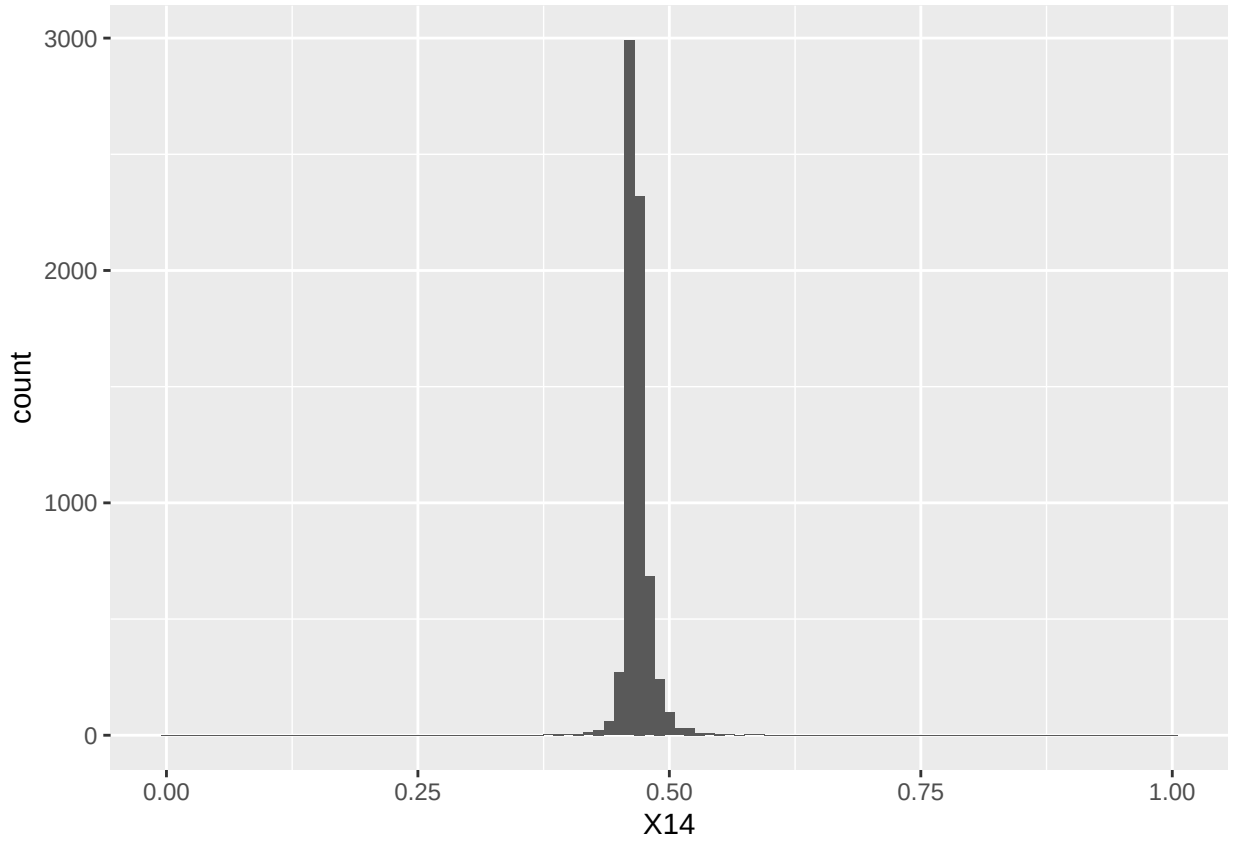


Looking at the distrbution of observations for some variables.

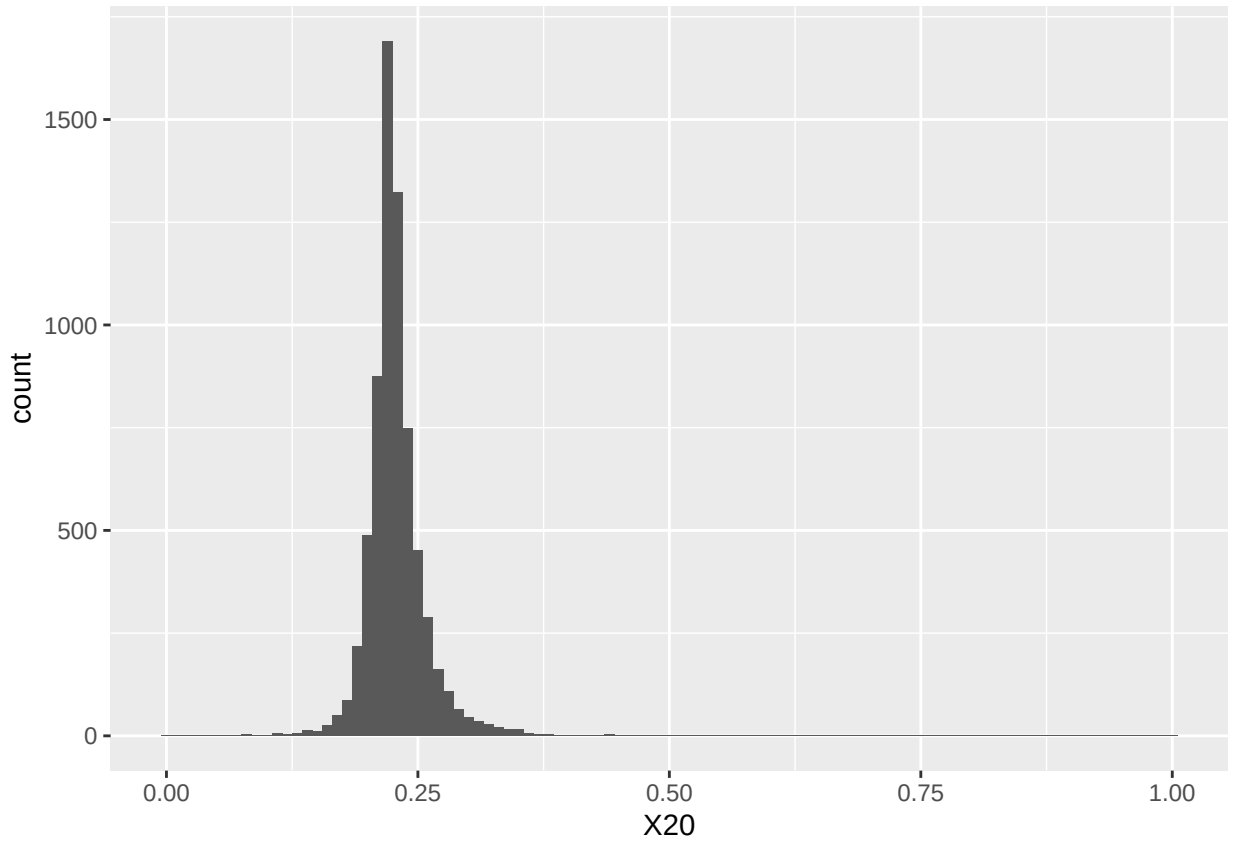
```
ggplot(df, aes(x = X5)) +
  geom_histogram(binwidth = 0.01)
```



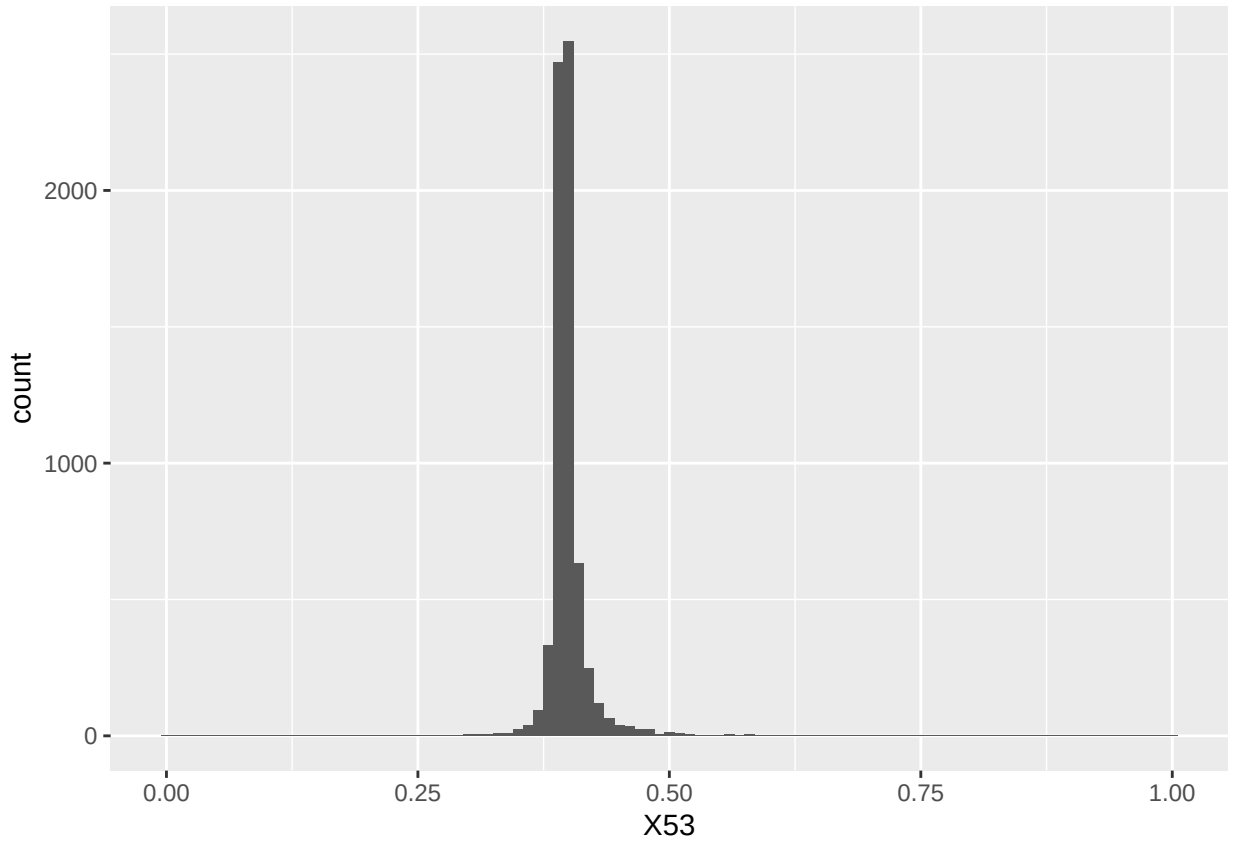
```
ggplot(df, aes(x = X14)) +
  geom_histogram(binwidth = 0.01)
```



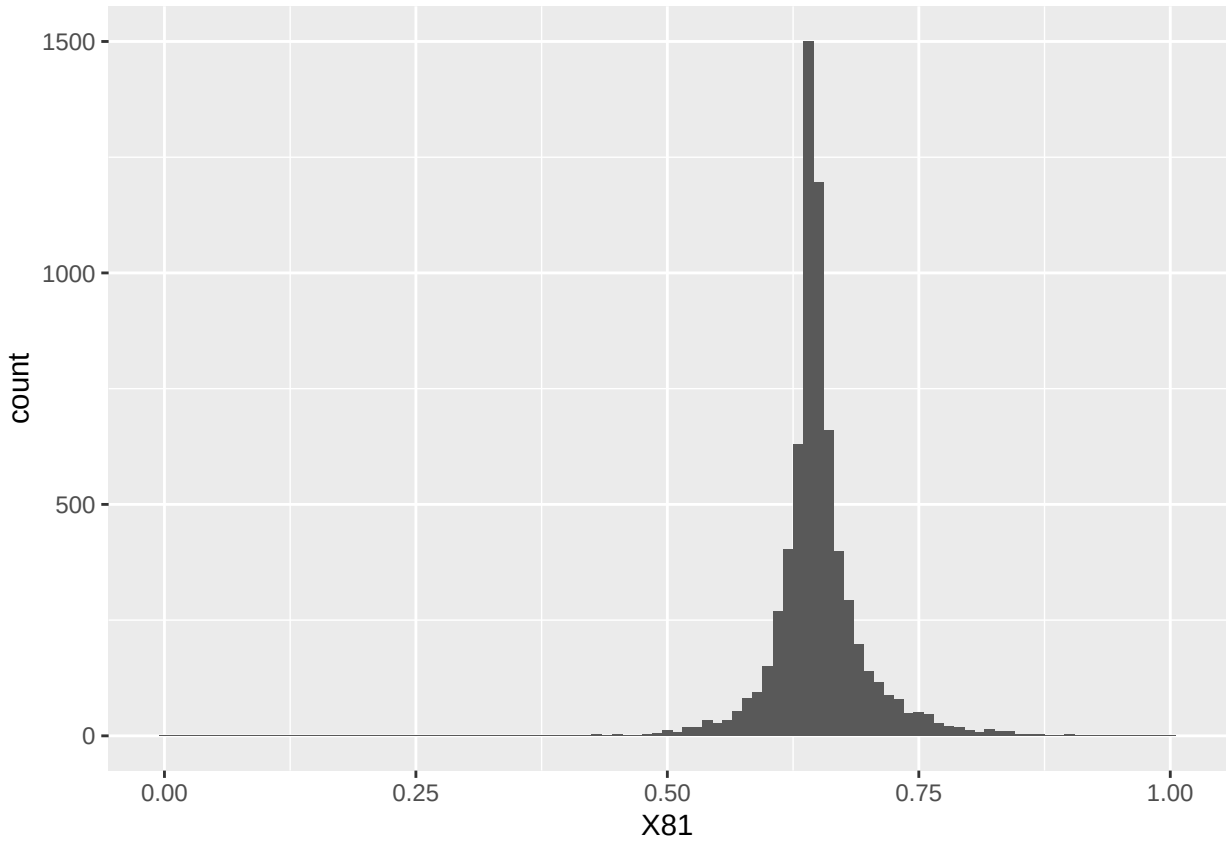

```
ggplot(df, aes(x = X20)) +
  geom_histogram(binwidth = 0.01)
```




```
ggplot(df, aes(x = X53)) +
  geom_histogram(binwidth = 0.01)
```

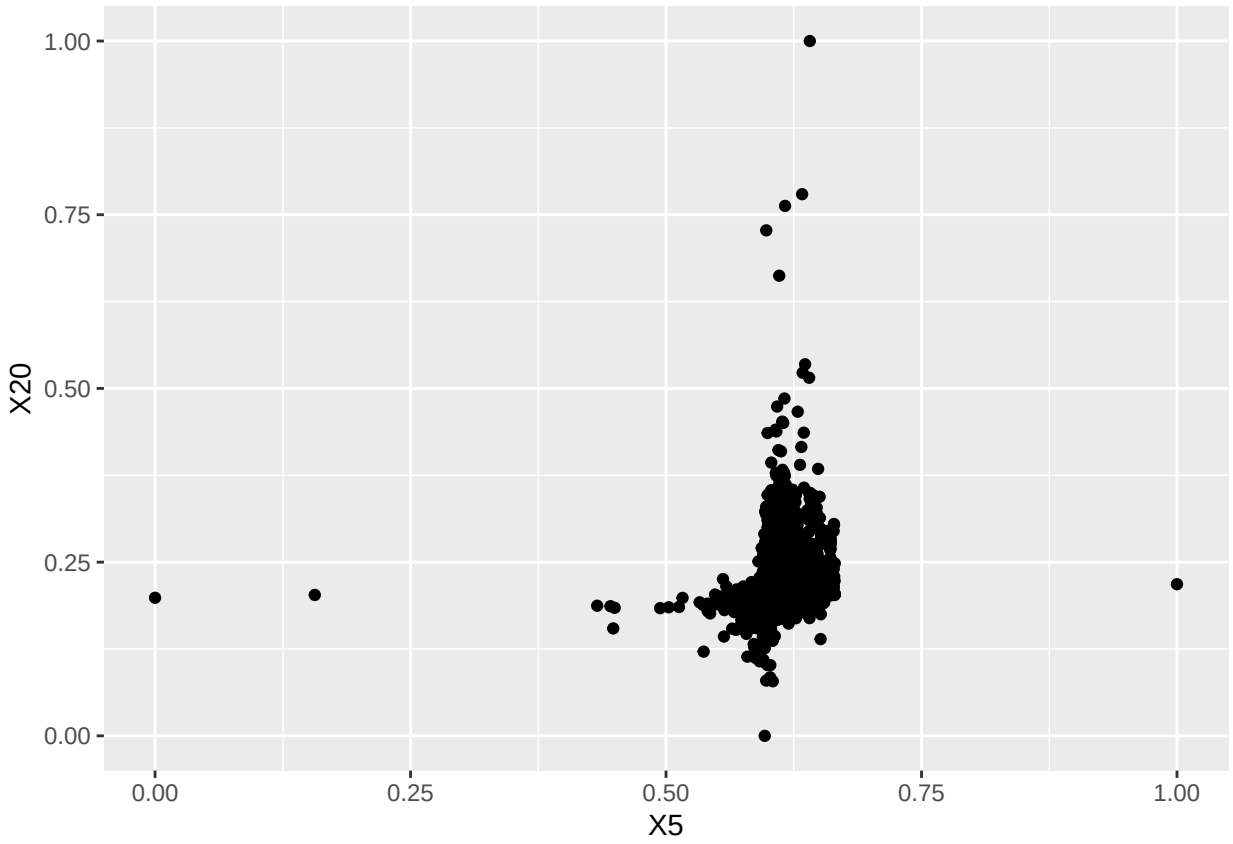



```
ggplot(df, aes(x = X81)) +
  geom_histogram(binwidth = 0.01)
```

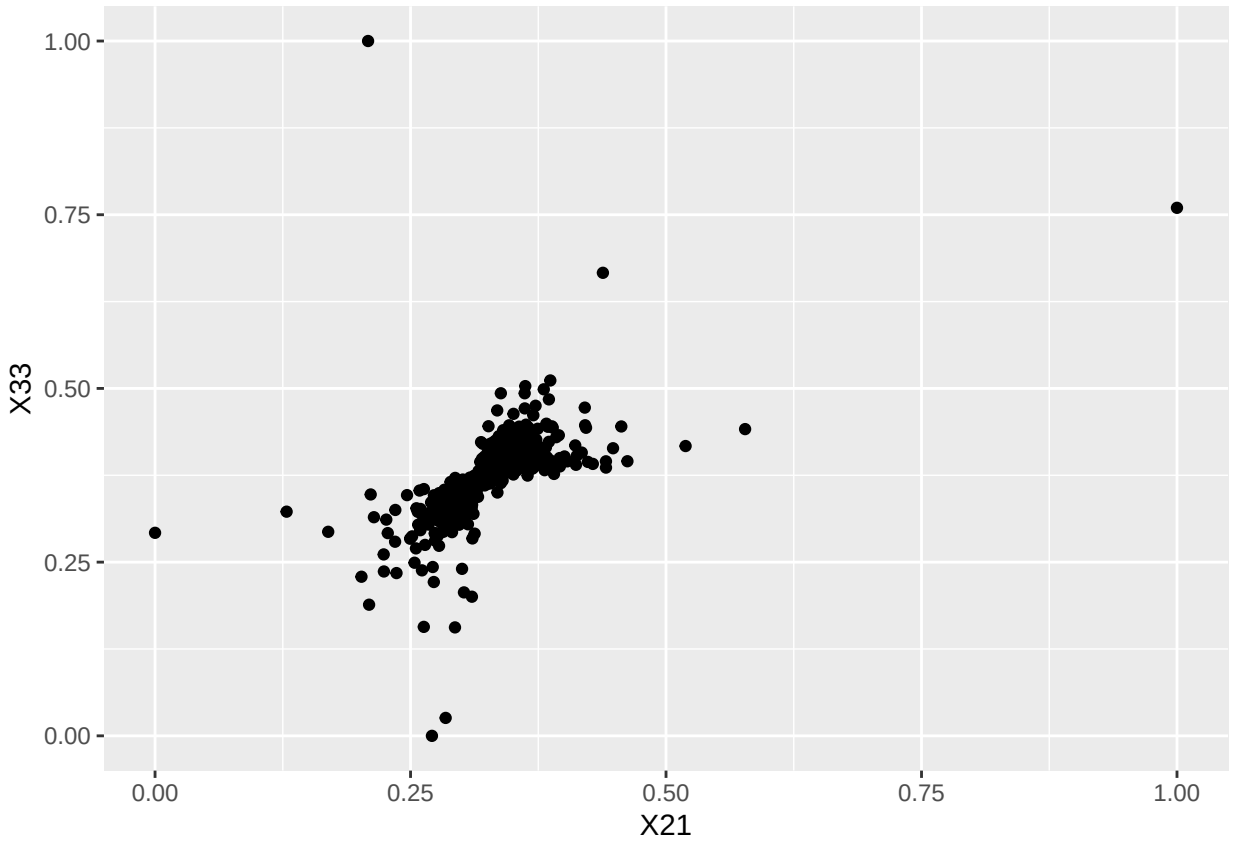


Most of the variables are or near normally distributed which is a good sign for general assumptions. Let's observe a few relationships between the variables.

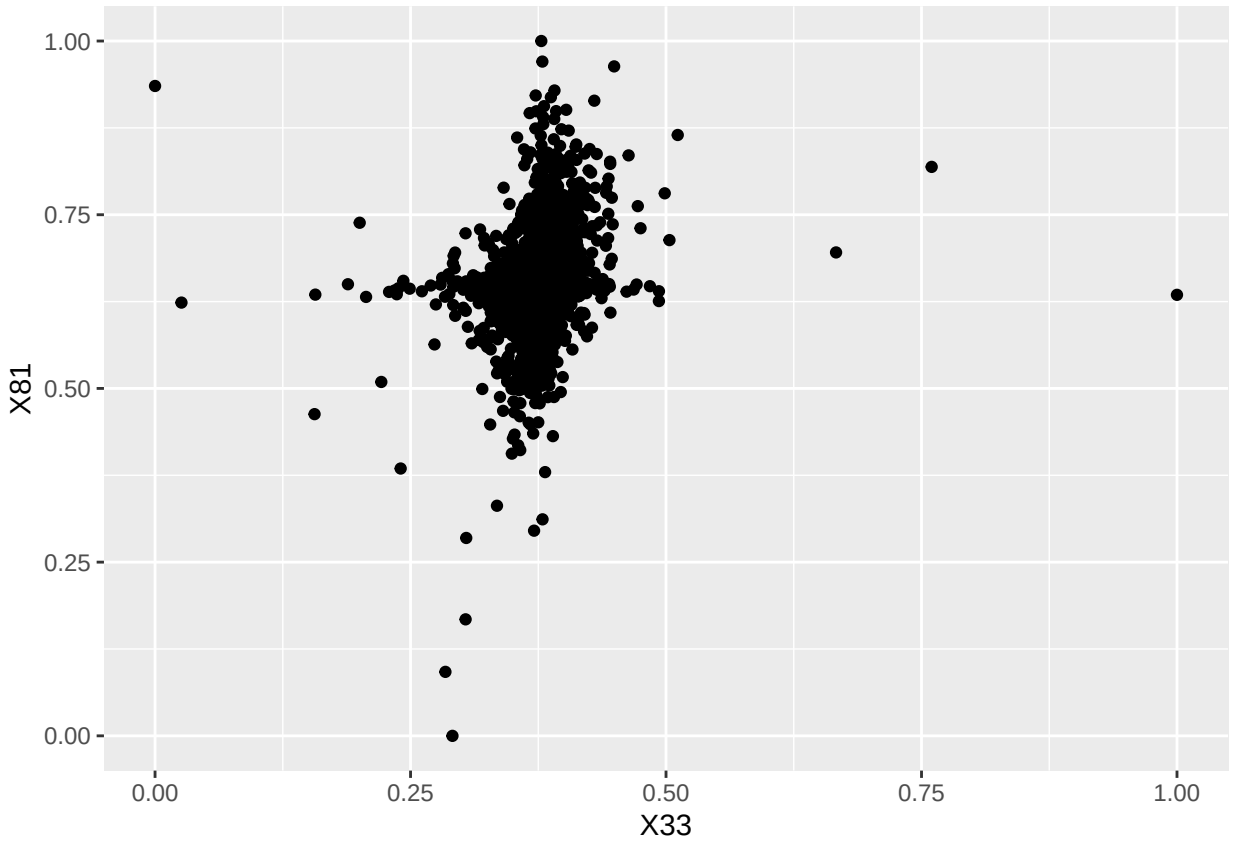
```
ggplot(df) +
  geom_point(mapping = aes(x = X5, y = X20))
```




```
ggplot(df) +
  geom_point(mapping = aes(x = X21, y = X33))
```




```
ggplot(df) +
  geom_point(mapping = aes(x = X33, y = X81))
```

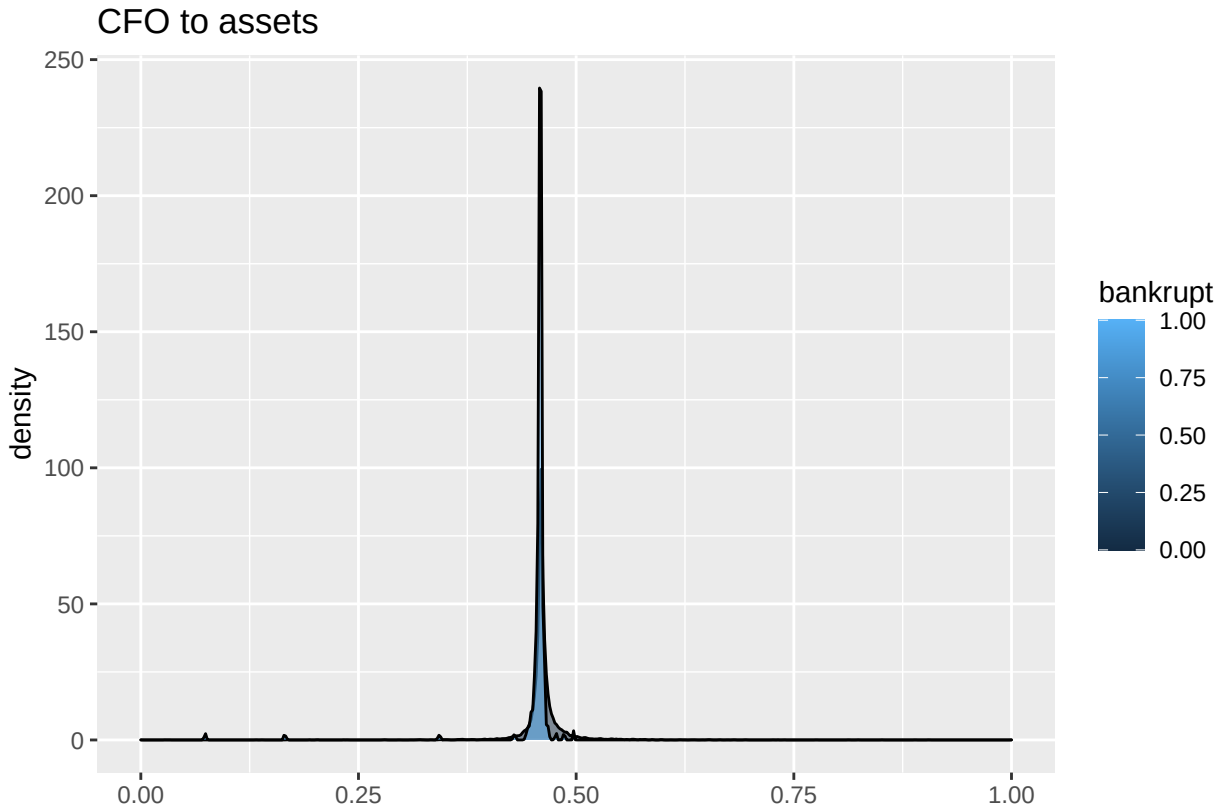


An interesting near heteroscedastic relationship between X21 and X33

How does current liability look across current assets?

```
df %>%
    group_by(bankrupt)%>%
    dplyr::summarize(TAGR = X82)%>%
    ggplot()+
    geom_density(aes(TAGR, group = bankrupt, fill = bankrupt), alpha = .5)+
    xlab("")+
    ggtitle("CFO to assets")
```

```
## `summarise()` has grouped output by 'bankrupt'. You can override using the
## `.groups` argument.
```

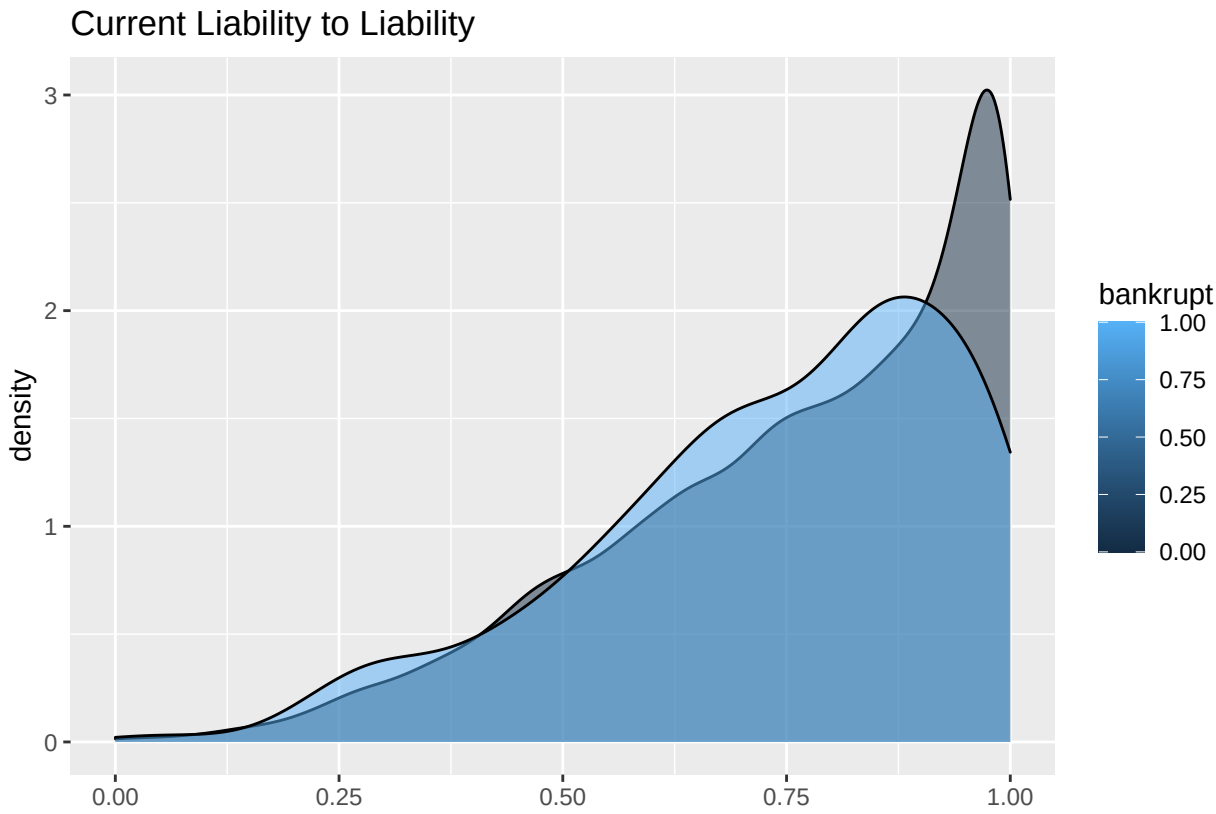


Survived and bankrupt have similar trends.

Current Liability to Equity

```
df %>%
    group_by(bankrupt) %>%
    dplyr::summarise(TAGR = X78) %>%
    ggplot() +
    geom_density(aes(TAGR, group = bankrupt, fill = bankrupt), alpha = .5) +
    xlab("") +
    ggtitle("Current Liability to Liability")
```

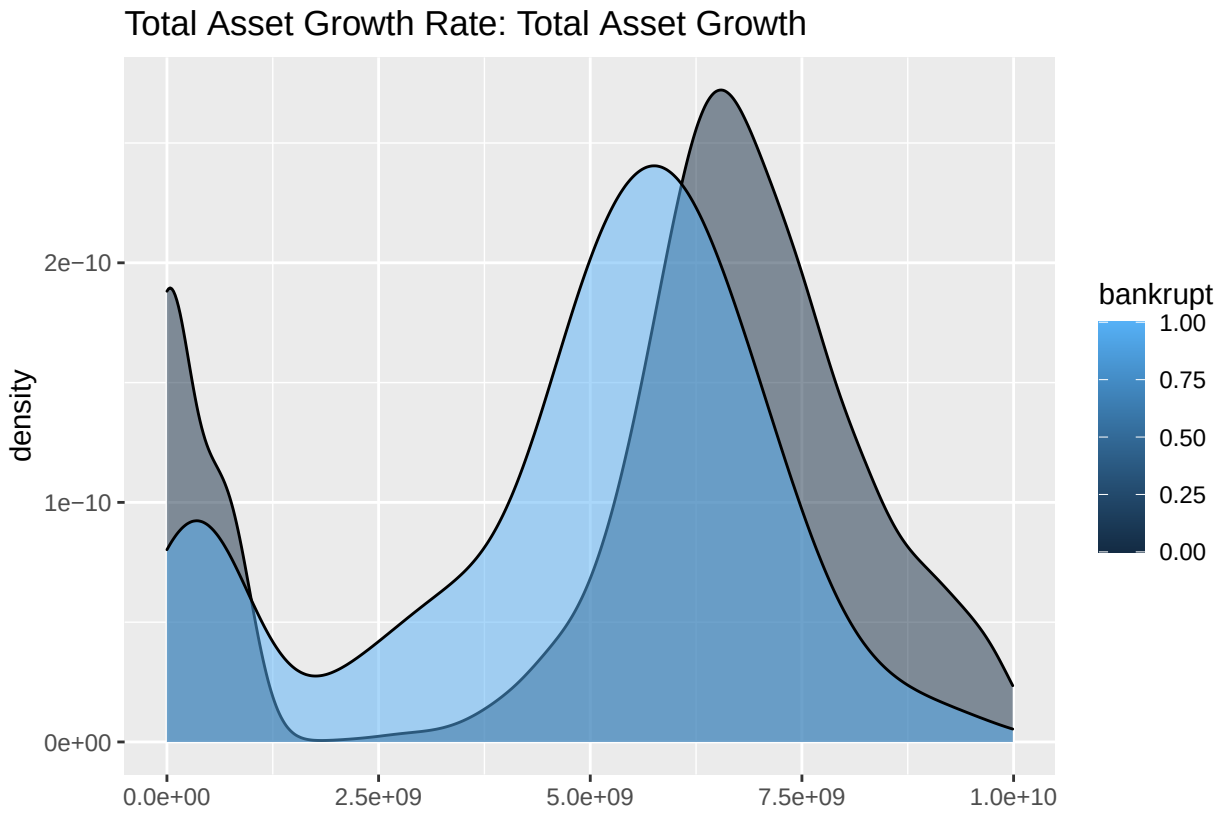
```
## `summarise()` has grouped output by 'bankrupt'. You can override using the
## `.groups` argument.
```



Survived seems to break off for current liability to increased liability.

```
df %>%
    group_by(bankrupt) %>%
    dplyr::summarise(TAGR = X30) %>%
    ggplot() +
    geom_density(aes(TAGR, group = bankrupt, fill = bankrupt), alpha = .5) +
    xlab("") +
    ggtitle("Total Asset Growth Rate: Total Asset Growth")
```

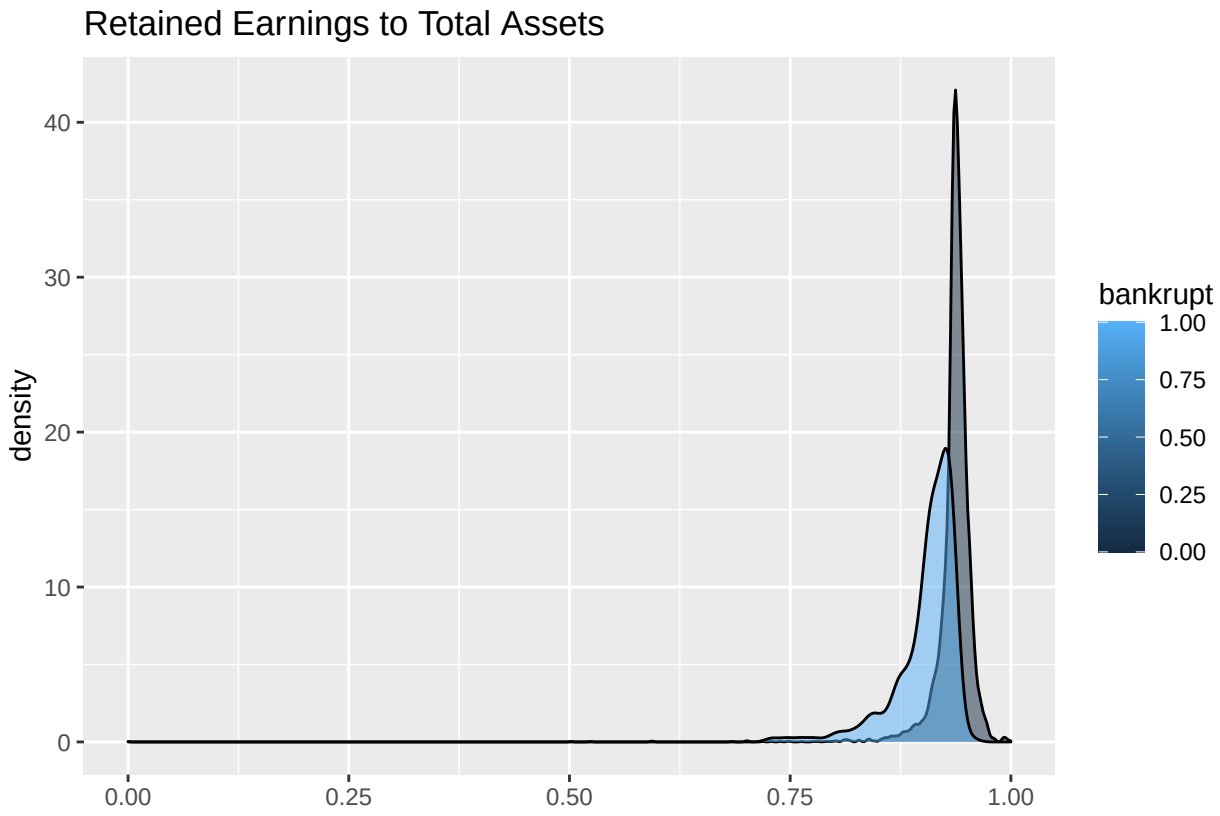
```
## `summarise()` has grouped output by 'bankrupt'. You can override using the
## `.groups` argument.
```



Survived has a big higher proportional total asset growth rate.

```
df %>%
    group_by(bankrupt) %>%
    dplyr::summarise(TAGR = X69) %>%
    ggplot() +
    geom_density(aes(TAGR, group = bankrupt, fill = bankrupt), alpha = .5) +
    xlab("") +
    ggtitle("Retained Earnings to Total Assets")
```

```
## `summarise()` has grouped output by 'bankrupt'. You can override using the
## `.groups` argument.
```



Modeling

Set Seed and Splitting the Data

We look towards model fitting and will start with a baseline for each model. The data split will be 75% training and 25% testing sets with stratified sampling upon the outcome 'bankrupt'.

```
df$bankrupt <-factor(df$bankrupt, levels=c(0, 1))
#Initial Split

train_test_split <- initial_split(df, strata = 'bankrupt', prop = 0.8)
train_df <- training(train_test_split)
test_df <- testing(train_test_split)

#Double check number of observations
dim(train_df)
```

```
## [1] 5455   68
```

```
dim(test_df)
```

```
## [1] 1364   68
```

```
#Convert outcome to factor
#df$bankrupt <- as.factor(df$bankrupt)
head(df)
```

```
## # A tibble: 6 x 68
##   bankrupt    X5    X9   X10   X11      X12      X13   X14      X15   X18   X20
##   <fct>    <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1 1        0.601 0.809 0.303 0.781 1.26e-4        0 0.458 0.000725 0.148 0.169
```

```
## 2 1        0.610 0.809 0.304 0.782 2.90e-4        0 0.462 0.000647 0.182 0.209
## 3 1        0.601 0.808 0.302 0.780 2.36e-4 25500000 0.459 0.000790 0.178 0.181
## 4 1        0.584 0.809 0.303 0.781 1.08e-4        0 0.466 0.000449 0.154 0.194
## 5 1        0.599 0.809 0.303 0.782 7.89e+9        0 0.463 0.000686 0.168 0.213
## 6 1        0.590 0.809 0.303 0.781 1.57e-4        0 0.466 0.000716 0.156 0.174
## # ... with 57 more variables: X21 <dbl>, X24 <dbl>, X25 <dbl>, X26 <dbl>,
## #   X28 <dbl>, X29 <dbl>, X30 <dbl>, X31 <dbl>, X32 <dbl>, X33 <dbl>,
## #   X34 <dbl>, X35 <dbl>, X36 <dbl>, X39 <dbl>, X41 <dbl>, X42 <dbl>,
## #   X44 <dbl>, X46 <dbl>, X47 <dbl>, X48 <dbl>, X49 <dbl>, X50 <dbl>,
## #   X51 <dbl>, X52 <dbl>, X53 <dbl>, X54 <dbl>, X56 <dbl>, X57 <dbl>,
## #   X58 <dbl>, X59 <dbl>, X60 <dbl>, X62 <dbl>, X63 <dbl>, X67 <dbl>,
## #   X68 <dbl>, X69 <dbl>, X70 <dbl>, X71 <dbl>, X72 <dbl>, X73 <dbl>, ...
```

Now, use K-fold cross-validation setting k = 5.

```
bank_folds <- vfold_cv(train_df, v = 5, strata = 'bankrupt')
bank_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits              id
##   <list>              <chr>
## 1 <split [4364/1091]> Fold1
## 2 <split [4364/1091]> Fold2
## 3 <split [4364/1091]> Fold3
## 4 <split [4364/1091]> Fold4
## 5 <split [4364/1091]> Fold5
```

We are performing k-folds cross validation which is a form of cross validation that takes multiple subsets of the training data to fit the model on. This is effective because it allows all observations to be input into the model which reduces bias. In essence, there are multiple iterations of validation where taking a certain fold assesses the model while the remaining are used to fit the model and thus, re-sampling.

Start by modeling with Logistic Regression, typically a great model for binary classification as it predicts the percentage of being in one or two classes at a given threshold. We can avoid step_impute_linear() because we have no missing values and hence, do not require the imputation.

Establishing a baseline model and analyzing the results. ### Baseline Model Untuned Logistic Regression.

```
bank_recipe <- recipe(bankrupt ~ ., data = train_df) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_numeric()) %>%
  step_scale(all_numeric())
```

```
log_reg <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

log_wf <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(bank_recipe)

log_fit <- fit_resamples(log_wf, bank_folds)
```

```
## ! Fold1: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
```

```
## ! Fold1: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
```

```
## ! Fold2: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...
```

```
## ! Fold2: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold3: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
## ! Fold3: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold4: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...
## ! Fold4: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold5: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
## ! Fold5: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
```

```
collect_metrics(log_fit)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.968     5 0.00589 Preprocessor1_Model1
## 2 roc_auc  binary     0.790     5 0.0785  Preprocessor1_Model1
```

```
log_test <- fit(log_wf, test_df)
```

```
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
predict(log_test, new_data = test_df, type = "class") %>%
  bind_cols(test_df) %>%
  accuracy(truth = bankrupt, estimate = .pred_class)
```

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

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.934
```

We can see that the model is performing very well; which is unsual and might need some further exploration. This is as a result of the class imbalance where it is predicting Survived/Not Bankrupt most of the time because ~97% of the observations consist of this. The immediate solution to this imbalance is upsampling as mentioned in lecture. The idea is that it will increase the minority class in this case Survived/Not Bankrupt and sample with replacement, balancing out the classes for better training.

```
train_up <- upSample(y = train_df$bankrupt,
                     x = train_df[,-1],
                     yname = "bankrupt")
table(train_up$bankrupt)
```

```
##
##    0    1
## 5289 5289
```

```
bank_folds1 <- vfold_cv(train_up, v = 5, strata = 'bankrupt')
bank_folds1
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits              id
##   <list>              <chr>
```

```
## 1 <split [8462/2116]> Fold1
## 2 <split [8462/2116]> Fold2
## 3 <split [8462/2116]> Fold3
## 4 <split [8462/2116]> Fold4
## 5 <split [8464/2114]> Fold5
```

```
bank_recipe1 <- recipe(bankrupt ~ ., data = train_up) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_numeric()) %>%
  step_scale(all_numeric())
```

```
log_reg1 <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

log_wf1 <- workflow() %>%
  add_model(log_reg1) %>%
  add_recipe(bank_recipe1)

log_fit1 <- fit_resamples(log_wf1, bank_folds1)
```

```
## ! Fold1: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...
## ! Fold1: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold2: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
## ! Fold2: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold3: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...
## ! Fold3: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold4: preprocessor 1/1, model 1/1: glm.fit: algorithm did not converge, glm.fi...
## ! Fold4: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
## ! Fold5: preprocessor 1/1, model 1/1: glm.fit: fitted probabilities numerically 0...
## ! Fold5: preprocessor 1/1, model 1/1 (predictions): prediction from a rank-defici...
```

```
collect_metrics(log_fit1)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.866     5  0.0163 Preprocessor1_Model1
## 2 roc_auc  binary     0.902     5  0.0310 Preprocessor1_Model1
```

Although the accuracy has decreased, our model has a lot more predictive power rather than simply predicting 0 or "Not Bankrupt" for most predictions. More samples allows the model to have more data to work with for both classes.

Let's move forward with other models to see how they perform. #### Linear Discriminant Analysis Simply testing performance.

```
control <- control_resamples(save_pred = TRUE)
lda_mod <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")
```

```
lda_wf <- workflow() %>%
  add_recipe(bank_recipe1) %>%
  add_model(lda_mod)

lda_fit <- fit_resamples (resamples = bank_folds1,
                          lda_wf,
                          control = control)
```

```
## ! Fold1: preprocessor 1/1, model 1/1: variables are collinear
## ! Fold2: preprocessor 1/1, model 1/1: variables are collinear
## ! Fold3: preprocessor 1/1, model 1/1: variables are collinear
## ! Fold4: preprocessor 1/1, model 1/1: variables are collinear
## ! Fold5: preprocessor 1/1, model 1/1: variables are collinear
```

Look at metrics.

```
collect_metrics(lda_fit)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.871     5 0.00536 Preprocessor1_Model1
## 2 roc_auc  binary     0.942     5 0.00343 Preprocessor1_Model1
```

KNN

K-Nearest Neighbors is a form of supervised learning for classification in this case. KNN is distance-based and implicitly assumes the smaller the distance between two points, the more similar they are. Will be fitting a KNN model and as usual, setting up model and workflow.

```
#Set up model and workflow
knn_model <- nearest_neighbor(neighbors = tune(), mode = "classification") %>%
  set_engine("kknn")

knn_wf <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(bank_recipe1)
```

Create a tuning grid by defining it.

```
#We need to determine best K
knn_params <- extract_parameter_set_dials(knn_model)

knn_grid <- grid_regular(knn_params, levels = 2)
```

Fit the resampled k-fold cross validation.

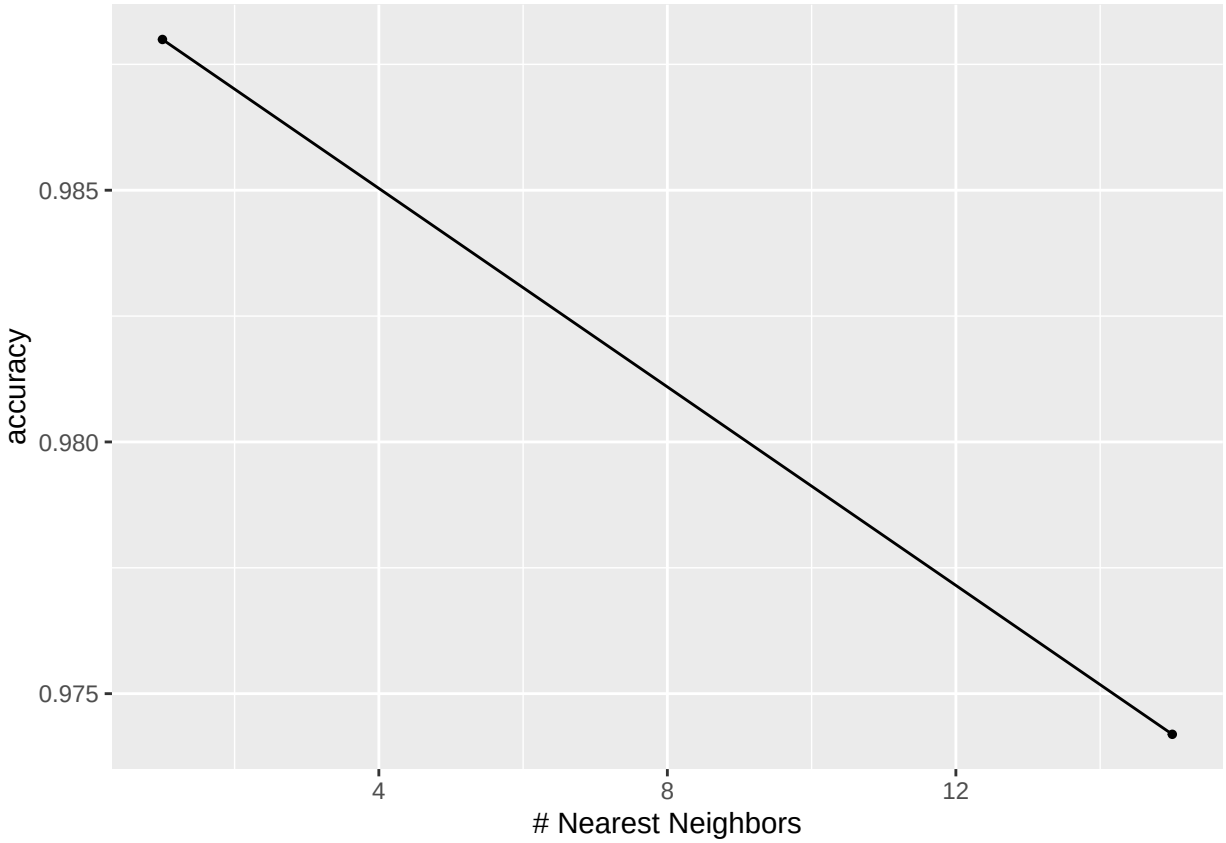
```
library(kknn)
```

```
##
## Attaching package: 'kknn'
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
knn_tune <- knn_wf %>%
  tune_grid(resamples = bank_folds1, grid = knn_grid)
```

Visualize the behavior of K.

```
autoplot(knn_tune, metric = "accuracy")
```



Display best K based on accuracy metric for binary classification.

```
show_best(knn_tune, metric = "accuracy")
```

```
## # A tibble: 2 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1         1 accuracy binary     0.988     5 0.00104 Preprocessor1_Model1
## 2        15 accuracy binary     0.974     5 0.00244 Preprocessor1_Model2
```

Decision Trees

Decision trees are typically used to classify or estimate continuous values by partitioning the sample space efficiently into sets with similar data points until one gets closer to a homogenous set and can reasonably predict the value for new data points.
Define model and workflow.

```
tree_model <- decision_tree(
  mode = "classification") %>%
  set_engine("rpart")

tree_wf <- workflow() %>%
```

```
  add_model(tree_model %>%
              set_args(cost_complexity = tune())) %>%
  add_recipe(bank_recipe1)
```

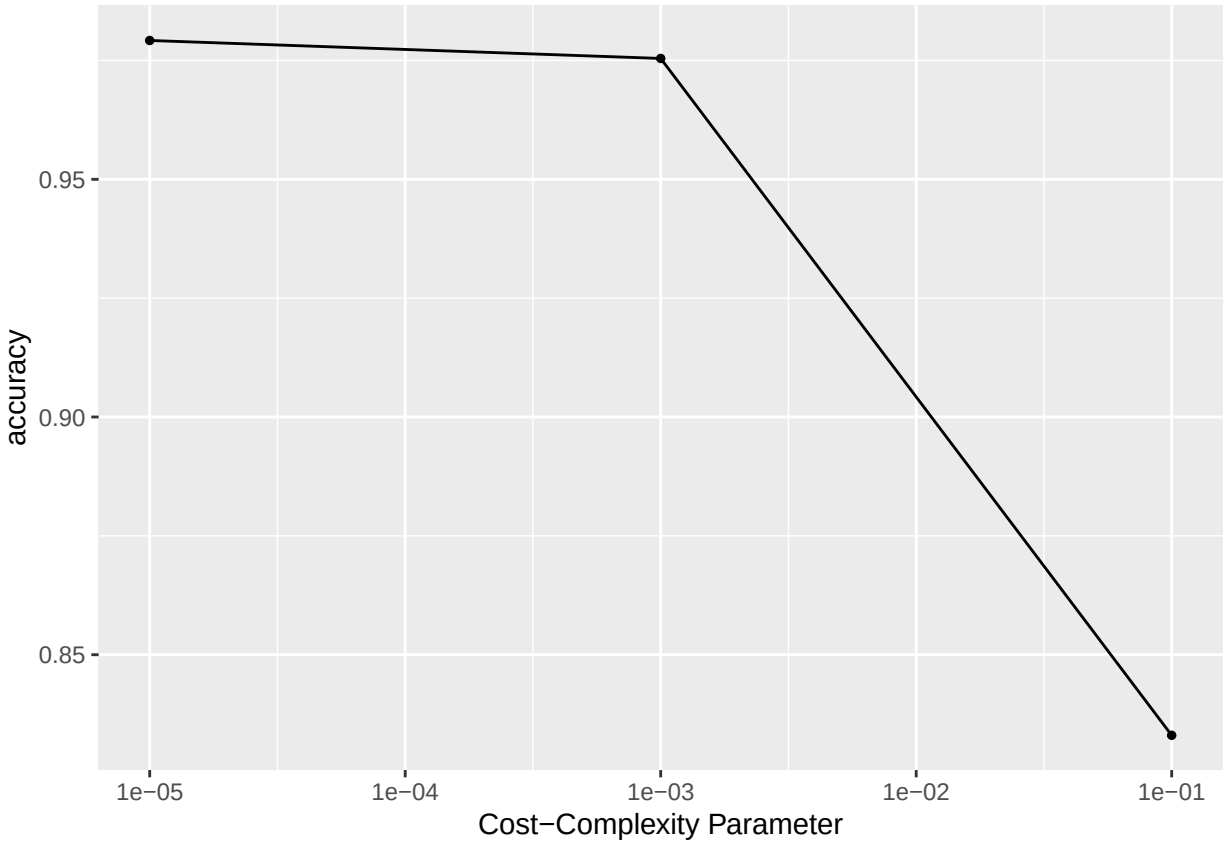
Create a tuning grid by defining it.

```
tree_grid <- grid_regular(cost_complexity(range = c(-5, -1)), levels = 3)
```

```
tree_tune <- tune_grid(
  tree_wf,
  resamples = bank_folds1,
  grid = tree_grid
)
```

Visualize using a plot the accuracy with cost complexity parameter.

```
autoplot(tree_tune, metric = "accuracy")
```



To confirm our results, let's take a look at the best tree and complexity.

```
show_best(tree_tune, metric = "accuracy")
```

```
## # A tibble: 3 x 7
##   cost_complexity .metric  .estimator  mean     n std_err .config
##             <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1         0.00001 accuracy binary     0.979     5 0.00211 Preprocessor1_Model1
## 2         0.001   accuracy binary     0.975     5 0.00238 Preprocessor1_Model2
## 3         0.1     accuracy binary     0.833     5 0.00402 Preprocessor1_Model3
```

Get best pruned tree.

```
best_pruned <- select_best(tree_tune, metric = "accuracy")
best_pruned
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <chr>
## 1         0.00001 Preprocessor1_Model1
```

Finalize the workflow for decision trees.

```
best_comp <- select_best(tree_tune)
```

```
## Warning: No value of `metric` was given; metric 'roc_auc' will be used.
```

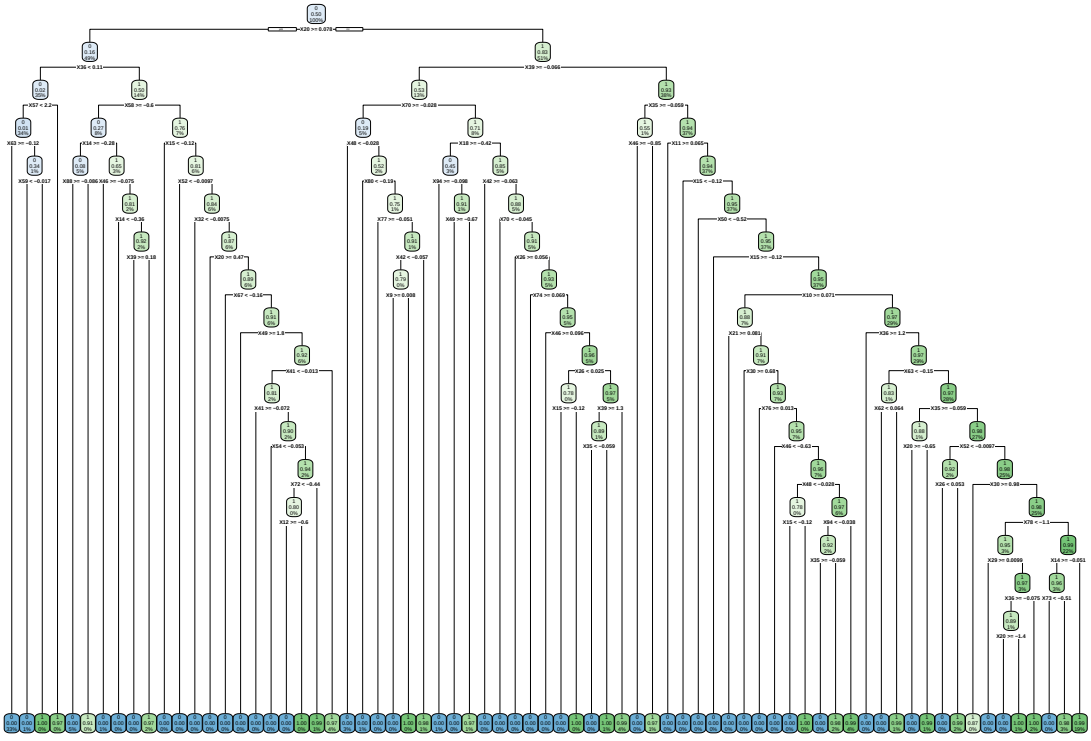
```
tree_final <- finalize_workflow(tree_wf, best_comp)
tree_final_fit <- fit(tree_final, data = train_up)
```

Final decision tree visualized.

```
tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```

```
## Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and is.binary
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Random Forests

Random forests are an ensemble method of decision trees used for both classification and regression. Although decision trees use a greedy algorithm, by maximizing data; we use the idea of "wisdom of the crowds" to generate an efficient model and collection of results and buffering performance.
Define model and workflow.

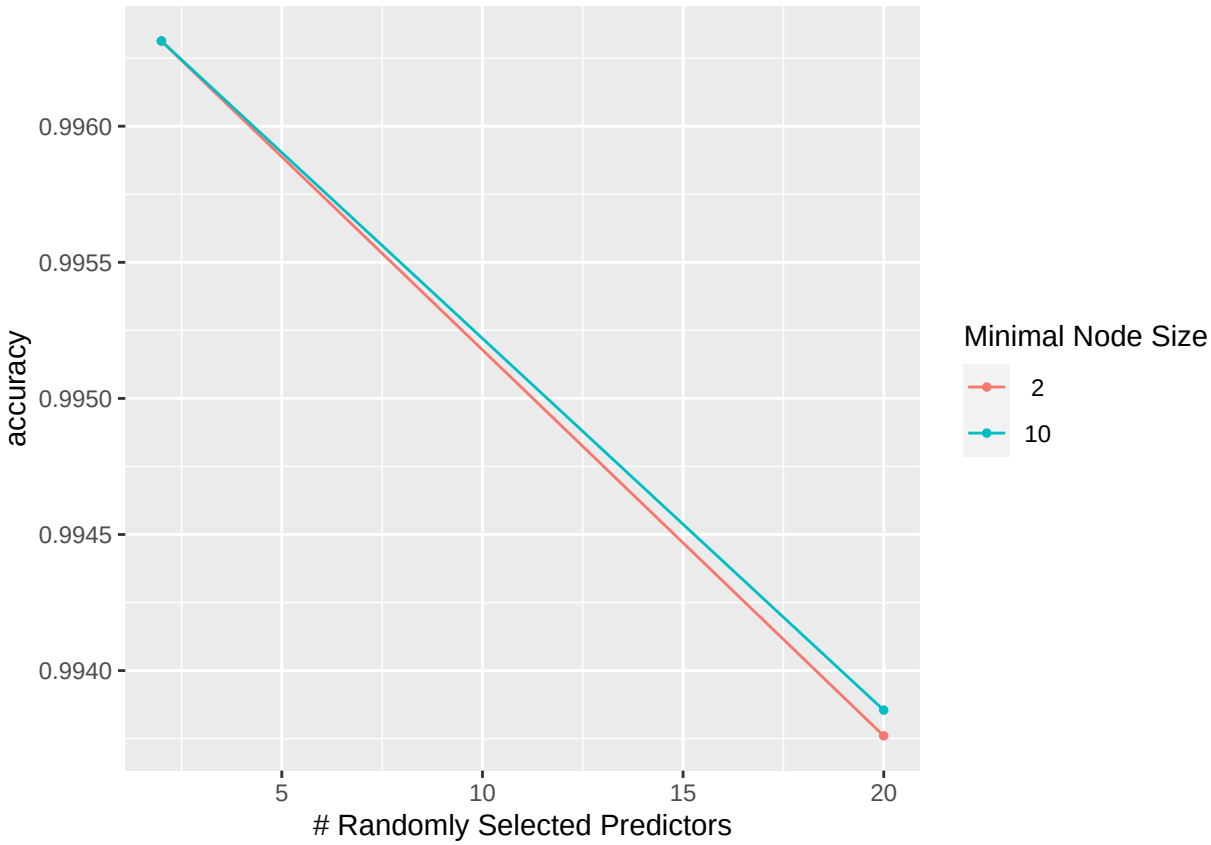
```
rf_model <- rand_forest(
              min_n = tune(),
              mtry = tune(),
              mode = "classification") %>%
  set_engine("ranger")

rf_wf <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(bank_recipe1)
```

```
rf_params <- extract_parameter_set_dials(rf_model) %>%
  update(mtry = mtry(range = c(2, 20)),
         min_n = min_n(range = c(2, 10)))

rf_grid <- grid_regular(rf_params, levels = 2)
```

```
rf_tune <- rf_wf %>%
  tune_grid(
    resamples = bank_folds1,
    grid = rf_grid)
```

Visualize using a plot the accuracy with randomly selected predictors.

```
autoplot(rf_tune, metric = "accuracy")
```



To confirm our results, let's take a look at the best rf.

```
show_best(rf_tune, metric = "accuracy")
```

```
## # A tibble: 4 x 8
##    mtry min_n .metric  .estimator  mean     n  std_err .config
##   <int> <int> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
## 1     2    10 accuracy binary     0.996     5 0.000378 Preprocessor1_Model3
## 2     2     2 accuracy binary     0.996     5 0.000434 Preprocessor1_Model1
## 3    20    10 accuracy binary     0.994     5 0.000748 Preprocessor1_Model4
## 4    20     2 accuracy binary     0.994     5 0.000783 Preprocessor1_Model2
```

Boosted Trees

Boosted trees are a form of gradient boosting and forms a class of algorithms rather than a single one. Define model and workflow.

```
bt_model <- boost_tree(mode = "classification",
                       min_n = tune(),
                       mtry = tune(),
                       learn_rate = tune()) %>%
  set_engine("xgboost")

bt_wf <- workflow() %>%
  add_model(bt_model) %>%
  add_recipe(bank_recipe1)
```

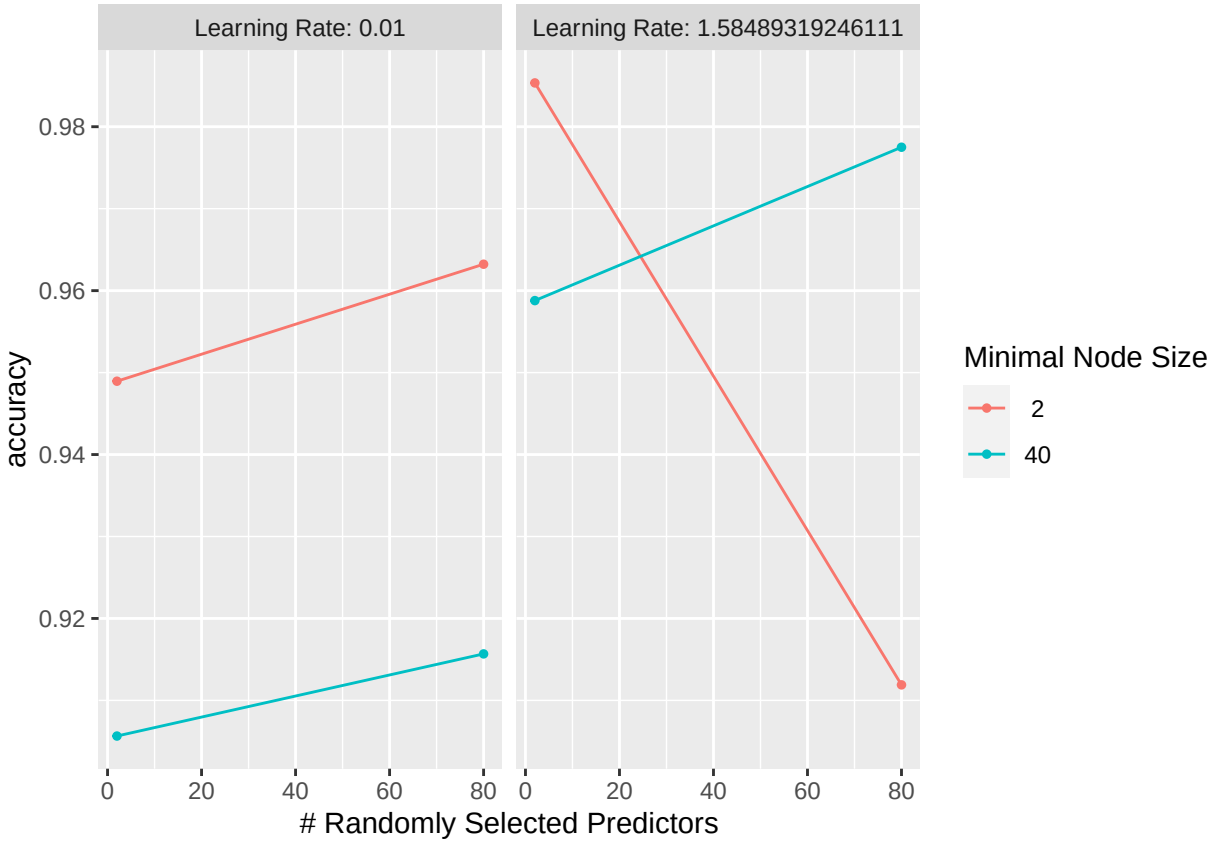
```
bt_params <- extract_parameter_set_dials(bt_model) %>%
  update(mtry = mtry(range= c(2, 80)),
```

```
        learn_rate = learn_rate(range = c(-2, 0.2))
  )

# define grid
bt_grid <- grid_regular(bt_params, levels = 2)
```

```
bt_tune <- bt_wf %>%
  tune_grid(resamples = bank_folds1,
    grid = bt_grid)
```

Visualize using a plot the accuracy with randomly selected predictors.

```
autoplot(bt_tune, metric = "accuracy")
```



To confirm our results, let's take a look at the best boosted tree.

```
show_best(bt_tune, metric = "accuracy")
```

```
## # A tibble: 5 x 9
##    mtry min_n learn_rate .metric  .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     2     2       1.58 accuracy binary     0.985     5 0.00121 Preprocessor1_~
## 2    80    40       1.58 accuracy binary     0.977     5 0.00274 Preprocessor1_~
## 3    80     2       0.01 accuracy binary     0.963     5 0.00228 Preprocessor1_~
## 4     2    40       1.58 accuracy binary     0.959     5 0.00294 Preprocessor1_~
## 5     2     2       0.01 accuracy binary     0.949     5 0.00667 Preprocessor1_~
```

Optimal parameters of learn_rate = 1.58, min_n = 2, mtry = 80 with a higher mean than our logistic regression model.

Let's fit our final model and generate predictions

```
rf_final <- rf_wf %>%
  finalize_workflow(select_best(rf_tune, metric = "accuracy"))
```

```
rf_final_fit <- fit(rf_final, train_up)
rf_final_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_dummy()
## * step_center()
## * step_scale()
##
## -- Model -----------------------------------------------------------------------
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~2L,      x), min.node.size = min_ro
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      10578
## Number of independent variables:  67
## Mtry:                             2
## Target node size:                 10
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.003088018
```

```
rf_final_fit %>% extract_fit_engine()
```

```
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~2L,      x), min.node.size = min_ro
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      10578
## Number of independent variables:  67
## Mtry:                             2
## Target node size:                 10
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.003088018
```

Obtain the final accuracy for the testing set.

```
rf_final_acc <- augment(rf_final_fit, new_data = test_df) %>%
  accuracy(truth = bankrupt, estimate = .pred_class)
rf_final_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.962
```

Conclusions

By way of conclusion, after training and tuning many models including Logistic Regression, Linear Discriminant Analysis, Decision Trees, Random Forests and Boosted Trees, it is evident that our final random forests model performed the best with a .96 accuracy that can be interpreted as 96% of observations were correctly predicted as opposed to incorrectly predicted in a binary classification. This can be deducted by the fact that it is an ensemble method and is built upon multiple decision trees and a bit less prone to error from a single tree. For this case, it was interesting to note that given the bankruptcy data to make predictions, the algorithm provided only the appropriate features to each tree in the forest, getting that tree's individual prediction, and then aggregates all predictions together to determine the overall prediction that the algorithm will make for said data.

Furthermore, it may be a bit illusive that our model has a ridiculously high accuracy. We can acknowledgeable that this is too high and we know that the model is simply predicting a majority to be 0 or not bankrupt, and thus a higher accuracy. In simple use cases, it might be applicable to say that the ones that the model does detect means the companies are noticeably more at risk of bankruptcy than the ones that were not detected by the model.

Overall, we can conclude that there a variety of contributing factors that denote bankruptcy within a company more so than others. This data set provided a lot of inside into the vulnerability of businesses and helps others learn from previous companies gone bankrupt and others that have survived. In a way, they have set a precedent with this data and analysis for future businesses.

Future Work

For future work, I would conduct a fully researched PCA within my modeling considering the large number of variables. Yet, through feature engineering and without categorical dummy variables denoting 0 and 1 it was avoidable in computing. Furthermore, I want to address the over-fitting and default prediction of 0 saving the model accuracy; considering L1 and L2 regression to combat this issue.