FFA project group 7

Loading essential packages

```r
library(tidyverse)
```

```
Warning: package 'stringr' was built under R version 4.3.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.3     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(readxl)
library(DataExplorer)
```

```
Warning: package 'DataExplorer' was built under R version 4.3.3
```

```r
library(plotly)
```

```
Attaching package: 'plotly'

The following object is masked from 'package:ggplot2':

    last_plot

The following object is masked from 'package:stats':

    filter

The following object is masked from 'package:graphics':

    layout
```

```r
library(zoo)
```

```
Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

```r
library(moments)
library(yardstick)
```

```
Attaching package: 'yardstick'

The following object is masked from 'package:readr':

    spec
```

```r
library(recipes)
```

```
Attaching package: 'recipes'
```

```
The following object is masked from 'package:stringr':

    fixed

The following object is masked from 'package:stats':

    step
```

```r
library(parsnip)
library(glmnet)
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-8
```

```r
library(coefplot)
library(tidyr)
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.1.1 --
v broom        1.0.5     v rsample      1.2.0
v dials        1.2.0     v tune         1.1.2
v infer        1.0.5     v workflows    1.1.3
v modeldata    1.2.0     v workflowsets 1.0.1
-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x Matrix::expand()  masks tidyr::expand()
x plotly::filter()  masks dplyr::filter(), stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x Matrix::pack()    masks tidyr::pack()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
```

```
x Matrix::unpack()  masks tidyr::unpack()
x Matrix::update()  masks recipes::update(), stats::update()
* Learn how to get started at https://www.tidymodels.org/start/
```

```r
library(xgboost)
```

```
Attaching package: 'xgboost'

The following object is masked from 'package:plotly':

    slice

The following object is masked from 'package:dplyr':

    slice
```

```r
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
```

```r
library(magrittr)
```

```
Attaching package: 'magrittr'

The following object is masked from 'package:purrr':
    set_names

The following object is masked from 'package:tidyr':

    extract
```

1. Exploratory Data Analysis

1.1 Original data

Load dataset

```
df <- read_csv("C:/Users/rayne/Documents/y4s2/ACCT420_forensic_forecasting/Bank Data 2000-
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 162408 Columns: 525
-- Column specification --------------------------------------------------------
Delimiter: ","
chr   (43): freq, Ticker Symbol_05601, Cusip_06004, Sedol_06006, Isin_06008,...
dbl  (461): code, year_, Net Sales Or Revenues_01001, Costs Of Goods Sold_01...
lgl   (16): Indicator  Restatement Type Key Items In Us Dollars_11557, Indic...
dttm   (1): Date Of Incorporation_18273
date   (4): Fiscal Period End Date_05350, Inactive Date_07015, Date Added To...

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.
```

162408 observations of 525 variables

Taking a look at the columns (525 columns too long so I wont run)

```
#names(df)
```

Firstly, filter companies whose General Code is 4 for banks.

```r
df <- df |>
  filter(`General Industry Classification_06010`==4)

count(df)
```

```
# A tibble: 1 x 1
      n
  <int>
1 46181
```

```r
df |>
  select(code)|>
  unique()|>
  count()
```

```
# A tibble: 1 x 1
      n
  <int>
1  3053
```

From 162k observations to 46k observations once we filter banks only, and 3053 unique companies

Getting a sense of how many countries in the data

```r
df |>
  group_by(Nation_06026)|>
  summarize(country_counts=n())|>
  arrange(desc(country_counts))
```

```
# A tibble: 125 x 2
   Nation_06026   country_counts
   <chr>                   <int>
 1 UNITED STATES           16839
 2 JAPAN                    2413
 3 INDIA                    1022
 4 INDONESIA                 938
 5 CHINA                     851
 6 NORWAY                    740
 7 BRAZIL                    681
 8 DENMARK                     677
 9 FRANCE                      676
10 UNITED KINGDOM              660
# i 115 more rows
```

125 countries

1.2 AAER dataset AAER is a binary, 1 for higher likelihood of fraud and 0 otherwise

AAER inner joined onto full data (AAER only includes US firms?? Since it is published by the SEC)

```
df_aaer <- readRDS("C:/Users/rayne/Documents/y4s2/ACCT420_forensic_forecasting/banks_final
```

AAER has 42k observations, now we filter out banking industry to see how many observations are there from banking.

```
df_aaer <- df_aaer |>
  filter(ITEM6010 == 4)

count(df_aaer)
```

```
# A tibble: 1 x 1
      n
  <int>
1 18941
```

From 42k to 18941 observations

Seeing how many unique companies are in banking industry

```
df_aaer |>
  filter(ITEM6010 == 4)|>
  select(code)|>
  unique()|>
  count()
```

```
# A tibble: 1 x 1
      n
```

```
  <int>
1  1262
```

From 3053 banking companies (full dataset) to 1262 banking companies (AAER data)

Getting the countries that AAER data includes

```
df_aaer |>
  group_by(ITEM6026)|>
  summarize(n=n())|>
  arrange(desc(n))
```

```
# A tibble: 39 x 2
   ITEM6026           n
   <chr>          <int>
 1 UNITED STATES  16839
 2 TURKEY           165
 3 AUSTRALIA        144
 4 ARGENTINA        135
 5 UNITED KINGDOM   106
 6 SOUTH AFRICA      94
 7 BRAZIL            93
 8 HONG KONG         88
 9 PUERTO RICO       88
10 SPAIN             85
# i 29 more rows
```

39 countries, most observations from the USA as expected

How many occurrences of AAER in relation to total observations per year?

```
df_aaer |>
  group_by(year)|>
  mutate(total_AAERS = sum(AAER==1), total_observations=n(), percent_AAERS = total_AAERS/t
  select(year, total_AAERS, total_observations, percent_AAERS)|>
  arrange(year)|>
  unique()
```

```
# A tibble: 24 x 4
# Groups:   year [24]
    year total_AAERS total_observations percent_AAERS
```

```
   <dbl>       <int>                <int>         <dbl>
 1  2000          12                  766        0.0157
 2  2001          11                  822        0.0134
 3  2002          12                  849        0.0141
 4  2003          12                  877        0.0137
 5  2004          11                  899        0.0122
 6  2005          17                  919        0.0185
 7  2006          16                  918        0.0174
 8  2007          19                  910        0.0209
 9  2008          19                  904        0.0210
10  2009          19                  901        0.0211
# i 14 more rows
```

215 occurrences of AAER in total out of 18941 total observations from 2000 to 2023 (2023 has 0 cases).

Takeaway from EDA:

1. The AAER dataset filters the original dataset from 162k rows to 18941 rows.
2. From 125 countries to 39 countries, mostly USA which makes sense since the AAER is published by the SEC.
3. From 3053 banking companies to 1262 banking companies.
4. Percent of AAER is very low each year, which makes sense since frauds are far and few between and companies which commit fraud tend to keep it well hidden.
5. As the years go by, the percent of AAER decreases, suggesting (just from this data) that there could be stricter regulations over time or companies got better at hiding.
6. 0 cases in 2023 -> suggestive that we have no data on it as it is still early 2024. We should filter out 2023.

```r
df_aaer <- df_aaer |>
  filter(year != 2023)
```

2. Feature engineering (and more EDA)

2 papers have been referenced to inspire our model building process. The first one enlightens us on the raw accounting line items and financial ratios to use to predict AAER, second one breaks down the formulas behind each financial ratio, with which we could use raw accounting line items to to manually derive.

1. From the journal of accounting research: Detecting Accounting Fraud in Publicly Traded U.S. Firms Using a Machine Learning Approach (Bao at el, 2015)
2. Predicting Material Accounting Misstatements (Dechow et al, 2011)

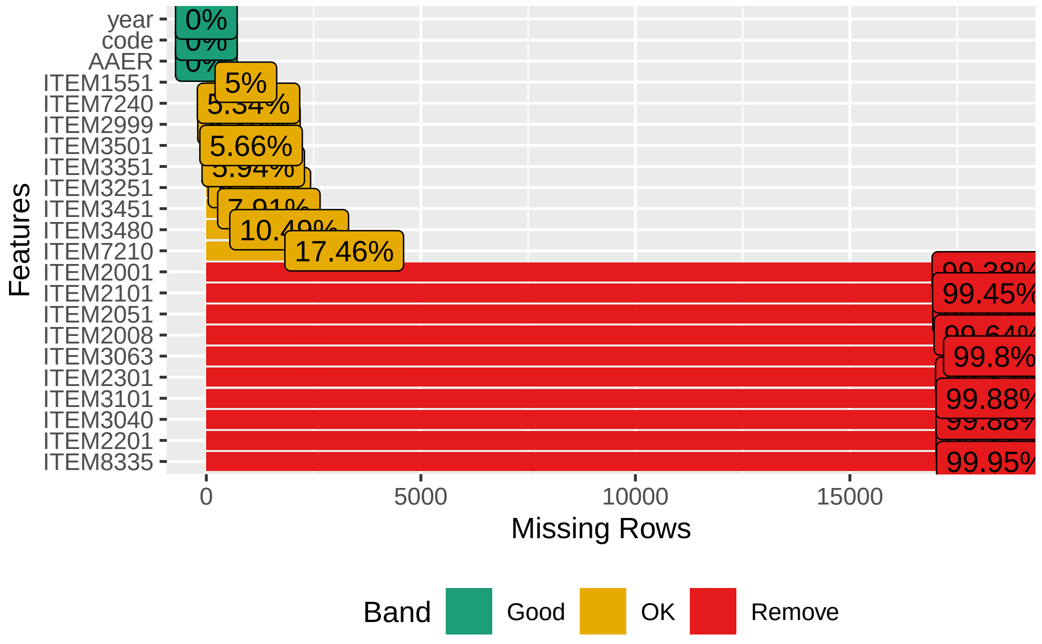
Additionally, based on our own text analysis and topic modelling, we realised that auditors play a huge role. Hence, aside from these well-researched accounting line items and financial ratios, we also have indicators for auditors.

2.1 EDA on 18 raw accounting line variables model

Selecting the 18 raw accounting line variables and convert to numeric

```
df_aaer1 <- df_aaer|>
  # ITEM7210 is the market cap, not part of the accounting line
  select(AAER, code, year, ITEM2001, ITEM2051, ITEM2101, ITEM2008, ITEM2201, ITEM2301, ITE
  mutate(ITEM2999=as.numeric(ITEM2999), ITEM3351=as.numeric(ITEM3351))
```

```
plot_missing(df_aaer1)
```



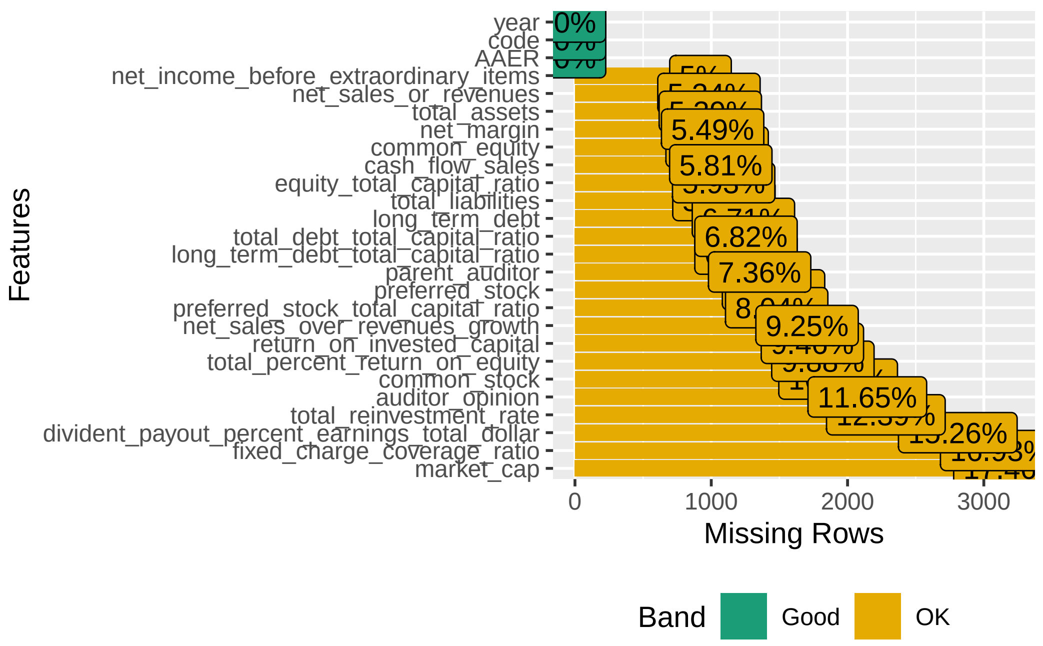
We keep only the raw accounting line items that have more than 80% filled data (the yellow ITEMS). Those with too many NA values are not useful for our model. Hence, we are left with 8 accounting line items.

2.2 EDA on 22 variable model (8 accounting line + 12 financial ratio + 2 indicators for big4 auditor and auditor's opinion)

Then, we add in 12 relevant financial ratios, auditor's opinion and parent auditor (Big 4 indicator) variables to the 8 raw accounting line items filtered. Note: an extra variable is market cap (ITEM7210) not used in modelling but used in data imputation.

```
df_aaer1 <- df_aaer |>
  select(AAER, code, year, ITEM3480, ITEM1551, ITEM7240, ITEM3501, ITEM3351, ITEM3251, ITE
  mutate(ITEM2999=as.numeric(ITEM2999), ITEM3351=as.numeric(ITEM3351))|>
  rename(common_stock=ITEM3480, net_income_before_extraordinary_items=ITEM1551, net_sales_
         )
```

```
plot_missing(df_aaer1)
```



All variables in our 22 variable model passes the 80% filled data test. Now it's time to explore on their correlation in order to prevent multicollinearity in our modelling process.

Firstly, we need to convert auditor's opinion and parent auditor to factor.

```
df_aaer1 <- df_aaer1|>
  mutate(parent_auditor = case_when(
    grepl("KPMG|Deloitte|PricewaterhouseCoopers|Ernst & Young",
          parent_auditor, ignore.case = TRUE) ~ "Big 4",
    TRUE ~ "Other"
  ))|>
  # convert Big 4 into factor, 1 if "Big 4", 0 if "Other"
  mutate(parent_auditor=factor(parent_auditor,levels = c("Other", "Big 4"), labels = c(0,
  mutate(auditor_opinion=as.factor(auditor_opinion))
```

Now, we take a look at the correlation matrix. We are trying to prevent multicollinearity in our model. If variables are correlated with one another, we will drop one of the two correlated variables, as part of pre-processing.

```
#ggplotly(df_aaer1 |>
          # plot_correlation(maxcat=16))
```

The red correlation shows the variable correlated to itself (which is obviously 1), the blue correlation are the same factor variable's factor levels (eg. AAER=0 and AAER=1) being correlated by -1, which is expected. There is no significant correlation between 2 different variables. As such, we conclude that all X variables are not correlated to one another and can be used in the logistic modelling process.

Since we have decided not to drop any variables, we now have to deal with all the NA values. Firstly, we group by company code and backfill the values based on the last recorded value in a company. This is better than merely imputing the mean of company code, as this is able to capture trends over time while a mean is unable to do so.

Function to backfill missing values within each group

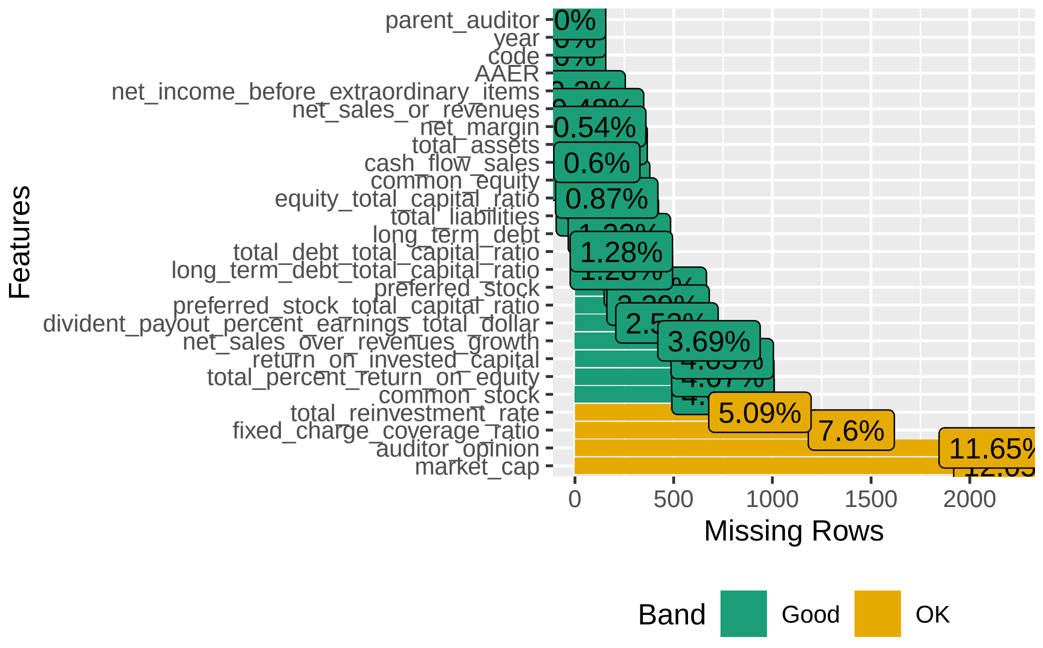
```
backfill_group <- function(x) {
  if (any(!is.na(x))) {
    # If there are non-NA values, fill missing values with last non-NA value
    na.locf(x, na.rm = FALSE)
  } else {
    # If all values are NA, return as is
    x
  }
}
```

I decided to define this object as df_aaer2

```
df_aaer2 <- df_aaer1|>
  arrange(code,year)|>
  group_by(code)|>
  mutate(across(-c(year, auditor_opinion, parent_auditor), backfill_group))|>
  ungroup()
```

Next, we re-look at our NA after this first step of imputation, it looks way better. It is worth noting at this point that parent_auditor has no NA values because when I factorised it to "Big 4" and "Other" (non-big 4), the small amount of NA values (6%) were characterised as the level of "Other".

```
plot_missing(df_aaer2)
```

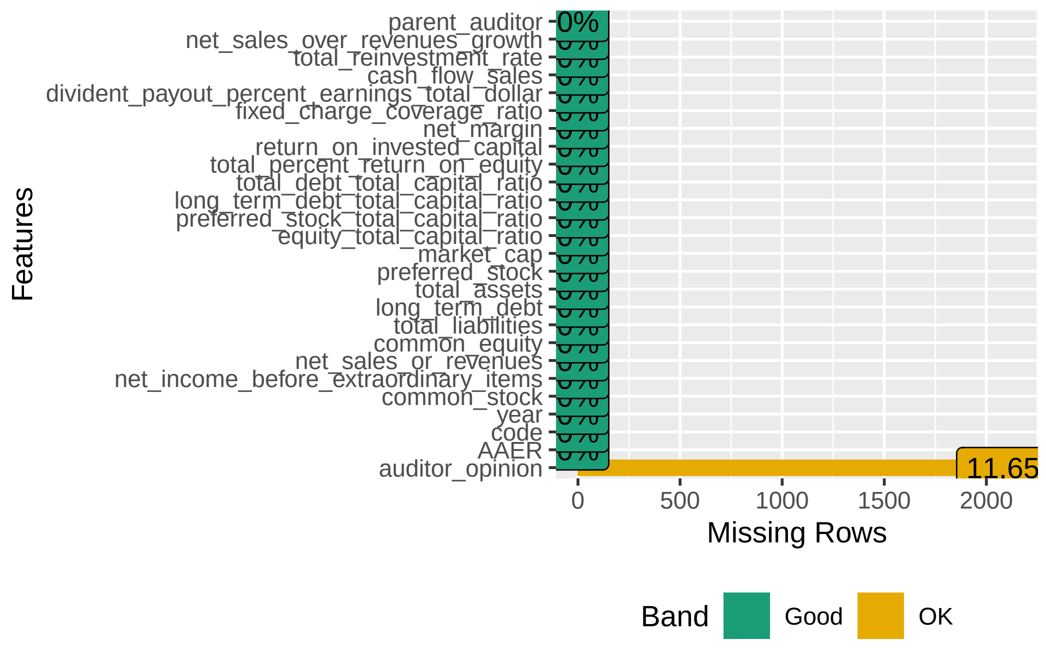


For the variables there were not able to be filled by backfilling, this suggests that these are companies with NA data in every record. In this case, it is unwise to fill it by imputing the mean of the entire column as one would usually do, and I decided to group by market cap then take the mean.

This means that if the company is large, then it will be imputed by a large cap mean values, and small companies will be imputed by small cap mean values. Hence, this allows us to impute data as accurately as possible by assigning small companies and large companies different means in accordance to their market cap.

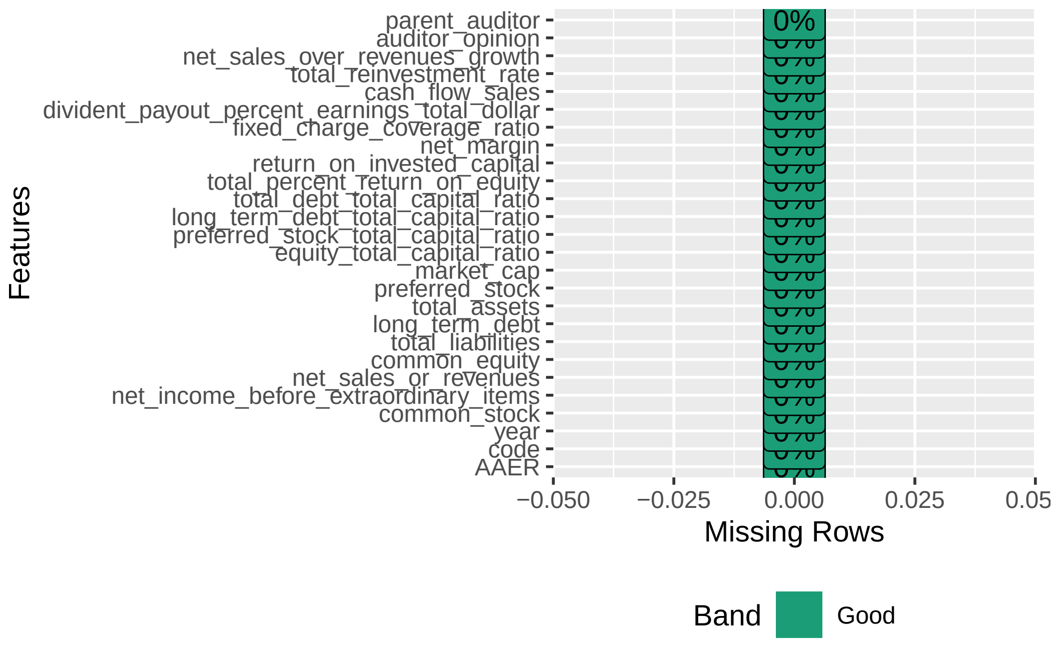
Firstly, we have to convert market cap to factor, and levels are sorted ordinally.

```
df_aaer2 = df_aaer2 %>%   mutate(market_cap = case_when(     market_cap > 200000000000 ~ "

mktcap_order = c("Unknown", "Micro-cap", "Small-cap",                    "Mid-cap","Large-c

df_aaer2$market_cap = na.fill(df_aaer2$market_cap, fill = "Unknown")

df_aaer2$market_cap <- factor(df_aaer2$market_cap,levels = mktcap_order,

df_aaer2 <- df_aaer2|>
  group_by(market_cap)|>
  # For all variables except AAER (y variable), if na, take mean, else take value in the r
  mutate(across(-c(AAER,auditor_opinion, parent_auditor), ~ifelse(is.na(.), mean(., na.rm=
  ungroup()



plot_missing(df_aaer2)
```



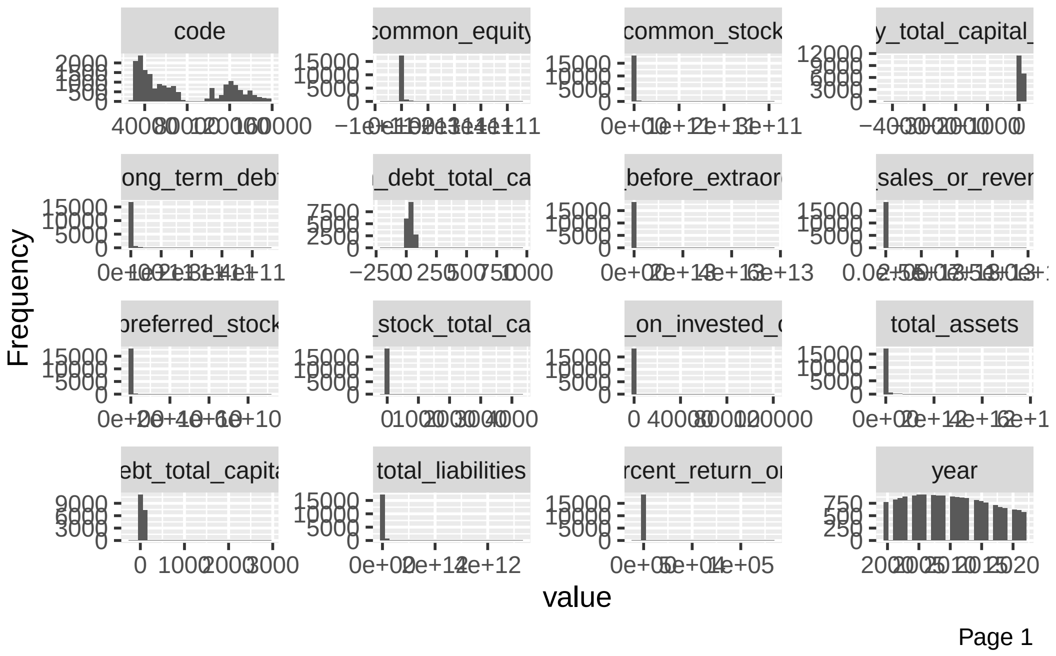
Now all that's left is to fill auditor_opinion NA values. This is tricky as auditor_opinion is not numeric, but categorical, so we cannot impute based on mean or backfilling, etc. And each level means a different thing. Therefore, we have to use accountancy logic to impute NA values. Ultimately, we decided to take a conservative approach to impute NA with 1, "not audited".

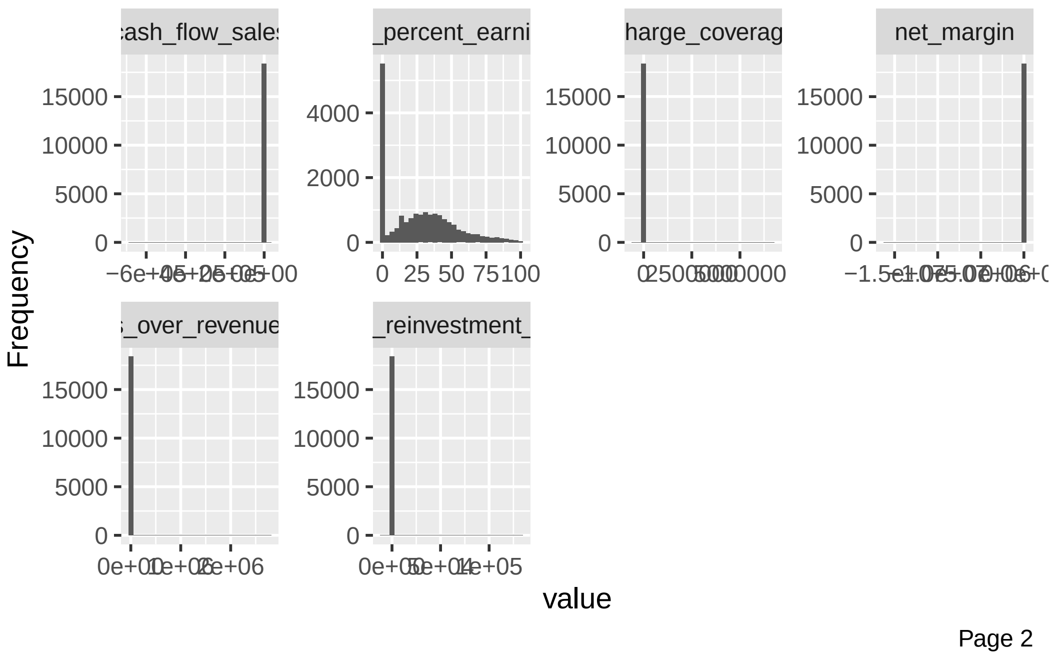
```r
df_aaer2 <- df_aaer2 |>
  mutate(auditor_opinion = if_else(is.na(auditor_opinion),as.factor(1),auditor_opinion))

df_aaer2 |>
  plot_missing()
```



Lastly, we have to look at how skewed the data is. For skewed X variables, it is imperative that we perform log transformation such that data is normalized. For this, I tried to use a histogram to check skewness.

```r
df_aaer2|>
  plot_histogram()
```





Imo, it is hard to visually evaluate which variables are skewed based on the histogram. Hence, we can use a fxn.

```r
log_and_drop_skewed_numeric_columns <- function(df, threshold, small_number) {
    for (col in names(df)) {
        if (is.numeric(df[[col]])) {
            skew <- skewness(df[[col]])
            if (abs(skew) > threshold) {
                log_col_name <- paste0("log_", col)
                # Log-transform positive values with the addition of a small number
                df[[log_col_name]] <- ifelse(df[[col]] > 0, log(df[[col]] + small_number),
                df[[col]] <- NULL
            }
        }
    }
    return(df)
}

# Set the skewness threshold (absolute value)
skewness_threshold = 10
small_number = 0.01
# Call the function and get the list of skewed columns
df_aaer3 = log_and_drop_skewed_numeric_columns(df_aaer2, skewness_threshold, small_number)
```

```
Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced
Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced
```

df_aaer3 is log transformed. df_aaer2 is the non log-transformed data.

Running both the df_aaer2 and df_aaer3 models in a seperate test script produced good results in terms of AUC. However, we could explore more…

We also saw that the raw accounting line items are inaccurate because of currency inconsistencies. Some records are in USD, while other records from other countries are in other currencies (eg. Argentina uses peso, Singapore uses SGD…).

This is captured in ITEM6099, "Currency Of Document".

```r
df_aaer %>%
  select(ITEM6099)|>
  unique()
```

```
# A tibble: 33 x 1
   ITEM6099
   <chr>
 1 USD
 2 EUR
 3 <NA>
 4 MXN
 5 ESP
 6 CLP
 7 ARS
 8 COP
 9 VES
10 HKD
# i 23 more rows
```

To mitigate this problem, we came up with 2 more models:

1. A percent change model, where each record is a percent change relative to the previous year.

- Potential flaw: Percent change model would not be able to differentiate between large companies and small companies, as the absolute values/magnitude of values are not captured. Instead, percent change values are captured.

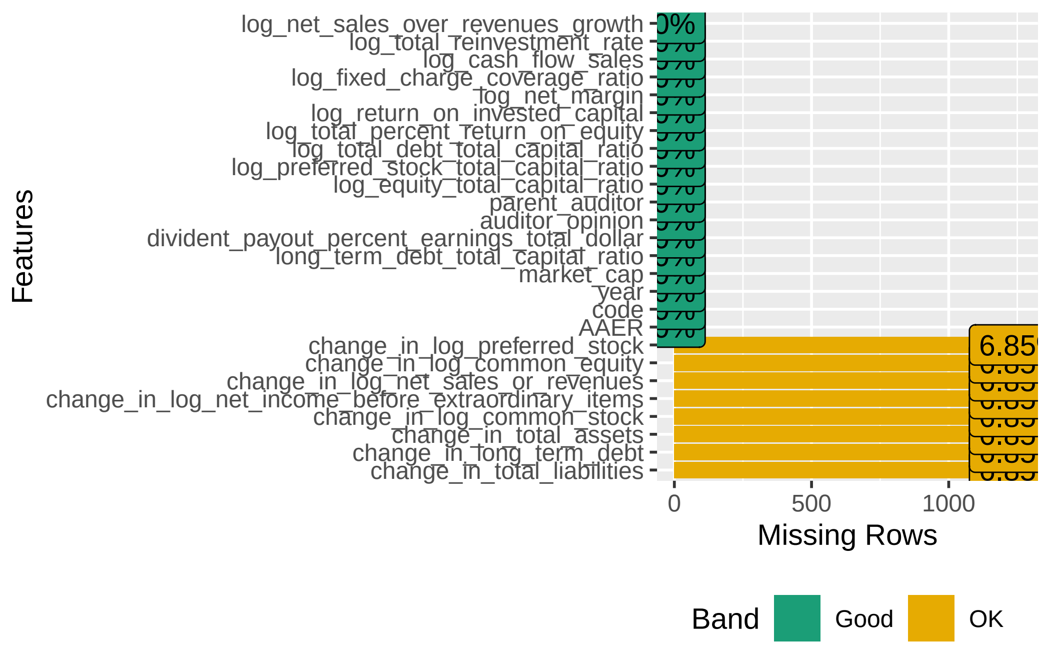
2. A currency adjusted model, where each record has been transformed from its respective foreign currency into the USD.

2.3 EDA on percent change model

One issue of taking percent change is that some variables (log preferred stock, long term debt) contain "0" in some of their records. In this case, the percent change will be infinity as even an increase of +1 to 0 is infinite. Hence, I will add an arbitrary small number to very small lagged values, to ensure there are no infinite percent change values.

```
df_aaer4 <- df_aaer3 %>%
  group_by(code) %>%
  arrange(code, year) %>%
  mutate(across(c(3:5, 11:15),    # Select the 8 raw accounting line items
              ~ {
                denominator <- lag(.)
                denominator[abs(denominator) < 0.01] <- 0.01  # Replace very small value
                ((. - lag(.)) / denominator) * 100
              },
              .names = 'change_in_{col}')) %>%
  # de-selecting the variables which we applied the percent change to
  select(-c(log_common_equity, log_net_income_before_extraordinary_items, log_net_sales_or
  ungroup()
```

```
plot_missing(df_aaer4)
```



NA values occur from records where there are no previous record from a year before. (eg. if 2000 is the first year of incidence, it is NA as there is no 1999 data to calculate the percent change).

For these NA values, I decided to just filter out the first record rows which contain these NA values. This is akin to simply dropping NA.

```
df_aaer4 <- df_aaer4|>
  na.omit()
```

2.4 EDA on currency adjusted model

```
df_aaer5 <- read_csv("C:/Users/rayne/Documents/y4s2/ACCT420_forensic_forecasting/currency_
  filter(ITEM6010 == 4) |>
  filter(year != 2023)
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
Rows: 37662 Columns: 547
-- Column specification --------------------------------------------------------
Delimiter: ","
chr   (40): freq, ITEM5601, ITEM6006, isin.x, ITEM6035, ITEM6038, ITEM6099, ...
dbl  (463): code, year, ITEM1001, ITEM1051, ITEM1084, ITEM1100, ITEM1101, IT...
lgl   (41): ITEM3257, ITEM6004, ITEM11557, ITEM11558, ITEM11559, ITEM18063, ...
date   (3): ITEM5350, ITEM7015, ITEM11516

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.
```

```
df_aaer5 <- df_aaer5 |>
  select(AAER, code, year, CurrAdj_ITEM3480, CurrAdj_ITEM1551, CurrAdj_ITEM3501, CurrAdj_I
  rename(adj_common_stock=CurrAdj_ITEM3480, adj_net_income_before_extraordinary_items=Curr
```

Now we simply repeat the data pre-processing steps (nth new, please skip this part and go to 2.5).

Backfill by company last value.

```
# backfill by company
df_aaer5 <- df_aaer5|>
  arrange(code,year)|>
  group_by(code)|>
  mutate(across(-c(year, auditor_opinion, parent_auditor), backfill_group))|>
  ungroup()
```

NA imputation based on mean market cap.

```
df_aaer5 = df_aaer5 %>%   mutate(market_cap = case_when(     market_cap > 200000000000 ~ "

mktcap_order = c("Unknown", "Micro-cap", "Small-cap",                   "Mid-cap","Large-c

df_aaer5$market_cap = na.fill(df_aaer5$market_cap, fill = "Unknown")

df_aaer5$market_cap <- factor(df_aaer5$market_cap,levels = mktcap_order,

df_aaer5 <- df_aaer5|>
  group_by(market_cap)|>
  # For all variables except AAER (y variable), if na, take mean, else take value in the r
  mutate(across(-c(AAER,auditor_opinion, parent_auditor), ~ifelse(is.na(.), mean(., na.rm=
  ungroup()
```

Converting to factor and filling auditor_opinion NAs to factor level of 1, “not audited”.

```
df_aaer5 <- df_aaer5|>
  mutate(parent_auditor = case_when(
    grepl("KPMG|Deloitte|PricewaterhouseCoopers|Ernst & Young",
          parent_auditor, ignore.case = TRUE) ~ "Big 4",
    TRUE ~ "Other"
  ))|>
  # convert Big 4 into factor, 1 if "Big 4", 0 if "Other"
  mutate(parent_auditor=factor(parent_auditor,levels = c("Other", "Big 4"), labels = c(0,
  mutate(auditor_opinion=as.factor(auditor_opinion))

df_aaer5 <- df_aaer5 |>
  mutate(auditor_opinion = if_else(is.na(auditor_opinion),as.factor(1),auditor_opinion))
```

Apply log to skewed variables.

```
df_aaer5 <- log_and_drop_skewed_numeric_columns(df_aaer5, skewness_threshold, small_number
```

```
Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced

Warning in log(df[[col]] + small_number): NaNs produced
```

2.5 Conclusion of feature engineering/EDA

In conclusion, we have build and tested 4 different model loadouts:

1. raw model (df_aaer2)
2. log-transformed model (df_aaer3)
3. log-transformed percent change model (df_aaer4)
4. log-transformed currency adjusted model (df_aaer5)

Upon running my model test script, I found that model 2 had the best AUC results (alogorithm: Xgboost), while model 4 was a very close second (also Xgboost). Therefore, for the Model building section, I will focus on model 2.

3. Model building

For model 2: log-transformed model

3.1 Splitting train test set

Creating a custom function that will perform stratified sampling in 80-20 train test ratio. We use stratified sampling as AAER occurrences are very small, and stratified sampling is able to deal with class imbalances.

```
split_df <- function(df){

set.seed(123)
split <- initial_split(df, prop = 0.8, strata = AAER)

# Extract training and testing data
train_data <- training(split)
test_data <- testing(split)

# Add indicator column for test data (1 for test, 0 for train)
test_data <- test_data %>%
  mutate(Test = 1)

# Add indicator column for train data (1 for test, 0 for train)
train_data <- train_data %>%
  mutate(Test = 0)

# Combine train and test data
```

```r
df <- bind_rows(train_data, test_data)
}
```

Create train test split and remove columns not in the model building process

```r
df_aaer3 <- split_df(df_aaer3)|>
  select(-market_cap)
```

3.2 Logistic regression

```r
logm1 <- glm(AAER ~. -code-year-Test, data=df_aaer3[df_aaer3$Test==0,], family=binomial)
```

```
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(logm1)
```

```

Call:
glm(formula = AAER ~ . - code - year - Test, family = binomial,
    data = df_aaer3[df_aaer3$Test == 0, ])

Coefficients:
                                              Estimate Std. Error z value
(Intercept)                                 -3.386e+00  5.038e-01  -6.720
total_liabilities                           -1.342e-11  3.041e-12  -4.412
long_term_debt                               3.419e-11  3.960e-12   8.634
total_assets                                 6.281e-12  2.625e-12   2.393
long_term_debt_total_capital_ratio           2.475e-02  5.420e-03   4.566
divident_payout_percent_earnings_total_dollar -9.678e-03  3.743e-03  -2.585
auditor_opinion5                            -1.253e+01  3.476e+02  -0.036
auditor_opinion6                             3.800e-02  2.259e-01   0.168
auditor_opinion7                            -1.876e+01  3.356e+04  -0.001
parent_auditor1                             -1.602e-01  2.283e-01  -0.702
log_common_stock                            -2.186e-02  2.164e-02  -1.010
log_net_income_before_extraordinary_items    3.870e-10  2.469e-10   1.568
log_net_sales_or_revenues                    8.599e-07  2.837e-05   0.030
log_common_equity                            1.498e-06  1.073e-06   1.395
log_preferred_stock                          4.899e-02  1.162e-02   4.216
```

```
log_equity_total_capital_ratio                -1.845e-02  6.900e-02  -0.267
log_preferred_stock_total_capital_ratio       -2.535e-01  6.976e-02  -3.633
log_total_debt_total_capital_ratio            -4.829e-01  8.378e-02  -5.765
log_total_percent_return_on_equity            -4.307e-03  4.933e-03  -0.873
log_return_on_invested_capital                -1.779e-02  1.364e-02  -1.305
log_net_margin                                 4.303e-03  5.707e-03   0.754
log_fixed_charge_coverage_ratio                5.410e-06  2.574e-05   0.210
log_cash_flow_sales                            4.841e-03  5.196e-03   0.932
log_total_reinvestment_rate                   -3.826e-03  4.458e-03  -0.858
log_net_sales_over_revenues_growth            -2.339e-02  7.578e-03  -3.086
                                              Pr(>|z|)
(Intercept)                                   1.81e-11 ***
total_liabilities                             1.03e-05 ***
long_term_debt                                 < 2e-16 ***
total_assets                                   0.01672 *
long_term_debt_total_capital_ratio            4.98e-06 ***
divident_payout_percent_earnings_total_dollar  0.00973 **
auditor_opinion5                               0.97125
auditor_opinion6                               0.86642
auditor_opinion7                               0.99955
parent_auditor1                                0.48283
log_common_stock                               0.31246
log_net_income_before_extraordinary_items      0.11699
log_net_sales_or_revenues                      0.97582
log_common_equity                              0.16289
log_preferred_stock                           2.49e-05 ***
log_equity_total_capital_ratio                 0.78919
log_preferred_stock_total_capital_ratio        0.00028 ***
log_total_debt_total_capital_ratio            8.19e-09 ***
log_total_percent_return_on_equity             0.38263
log_return_on_invested_capital                 0.19203
log_net_margin                                 0.45082
log_fixed_charge_coverage_ratio                0.83356
log_cash_flow_sales                            0.35144
log_total_reinvestment_rate                    0.39069
log_net_sales_over_revenues_growth             0.00203 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1855.1  on 14729  degrees of freedom
Residual deviance: 1570.9  on 14705  degrees of freedom
```

```
AIC: 1620.9

Number of Fisher Scoring iterations: 23
```

```
df_aaer3$pred_AAER <- predict(logm1, df_aaer3, type="response")
```

Analysis:

From the logistic regression, the significant variables in predicting AAER are total liabilities, long-term debt, total assets, long term debt to total capital ratio, total dollar percent earnings dividend payout, log preferred stock, log preferred stock to total capital ratio, log total debt to total capital ratio, and log total reinvestment rate.

The coefficient on these variables are how much these variables affect our prediction. For example, a 1 unit increase in total liabilities leads to a -1.438e-11 probability of AAER (total liabilities reduces the probability of AAER), while a 1 unit increase in long term debt leads to a 3.821e-11 probability of AAER (long-term debt increases the probability of AAER), and so on…

```
# in-sample confusion matrix
confmat.train <- confusionMatrix(as.factor(ifelse(logm1$fitted.values > 0.5, 1, 0)),
                                 as.factor(df_aaer3[df_aaer3$Test==0,]$AAER),
                                 positive = "1")

confmat.train
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 14552   157
         1     8    13

               Accuracy : 0.9888
                 95% CI : (0.987, 0.9904)
    No Information Rate : 0.9885
    P-Value [Acc > NIR] : 0.3684

                  Kappa : 0.1339

 Mcnemar's Test P-Value : <2e-16
```

```
            Sensitivity : 0.0764706
            Specificity : 0.9994505
         Pos Pred Value : 0.6190476
         Neg Pred Value : 0.9893263
             Prevalence : 0.0115411
         Detection Rate : 0.0008826
   Detection Prevalence : 0.0014257
      Balanced Accuracy : 0.5379606

       'Positive' Class : 1
```

```
# out-of-sample confusion matrix
confmat.test <- confusionMatrix(
  as.factor(ifelse(df_aaer3[df_aaer3$Test == 1,]$pred_AAER > 0.5, 1, 0)),
  as.factor(df_aaer3[df_aaer3$Test == 1,]$AAER),
  positive = "1"
)


confmat.test
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 3633   39
         1    5    6

               Accuracy : 0.9881
                 95% CI : (0.984, 0.9913)
    No Information Rate : 0.9878
    P-Value [Acc > NIR] : 0.4798

                  Kappa : 0.2105

 Mcnemar's Test P-Value : 6.527e-07

            Sensitivity : 0.133333
            Specificity : 0.998626
         Pos Pred Value : 0.545455
         Neg Pred Value : 0.989379
```

```
           Prevalence : 0.012218
       Detection Rate : 0.001629
 Detection Prevalence : 0.002987
    Balanced Accuracy : 0.565979

     'Positive' Class : 1
```

Analysis: Classification matrix with a 0.5 cut-off demonstrates…

- High Accuracy of 98.81%.
- Low Sensitivity of 13.33%, due to the class imbalances even despite stratified sampling.
- High Specitivity of 99.9%.

This could be due to the class imbalances where there are too many instances of the majority class in the data. For example,

No Information Rate (NIR) is the accuracy that could be achieved by always predicting the majority class. In this case, the NIR is 0.9878, which suggests that the majority class constitutes 98.78% of the dataset.

```r
auc_in <- df_aaer3 %>% filter(Test==0) %>% roc_auc(AAER, pred_AAER, event_level='second')
auc_out <- df_aaer3 %>% filter(Test==1) %>% roc_auc(AAER, pred_AAER, event_level='second')
curve_in <- df_aaer3 %>% filter(Test==0) %>% roc_curve(AAER, pred_AAER, event_level='secon
curve_out <- df_aaer3 %>% filter(Test==1) %>% roc_curve(AAER, pred_AAER, event_level='seco



#ggplot() +
  #geom_line(data=curve_in, aes(y=sensitivity, x=1-specificity, color="AAER, In Sample"))
  #geom_line(data=curve_out, aes(y=sensitivity, x=1-specificity, color="AAER, Out of Sampl
  #geom_abline(slope=1)

aucs <- c(auc_in$.estimate, auc_out$.estimate)
names(aucs) <- c("In sample AUC", "Out of sample AUC")
aucs
```

```
    In sample AUC Out of sample AUC
        0.7475618         0.7451041
```

Analysis: The logistic regression performs well with AUC of around 0.75 for both in-sample and out-sample.