Applying the CRISP-DM Data Science Methodology to Sales Volume Forecasting and Budgeting Problems

Project Documentation

Matthias Hofmaier (11944050)

August 2023

Contents

1. Introduction

Reliable forecasts of a company's sales volume can be of massive advantage in budgeting and strategic planning. Traditional methods to forecast sales often rely on univariate moving average models. In the last decade, machine learning methods, which can incorporate information over various dimensions gained a lot of popularity. Those models allow the inclusion of variables from annual financial statements, including balance sheets and profit and loss statements, that potentially increase the prediction performance. But dealing with this data can be challenging. Especially for people who are from different domains and are not used to data science workflows. The Cross Industry Standard Process for Data Mining (CRISP-DM) can help to perform data science projects of this kind in a well-defined way. Therefore the goal of this project is to create a guideline project for students in economics that showcases how the CRISP-DM methodology can be applied to sales volume forecasting and budgeting problems. The project will be carried out on the example of comparing a univariate model and a multivariate machine learning model within the context of U.S. stock corporations in the S&P 500 from 2002 to 2022.

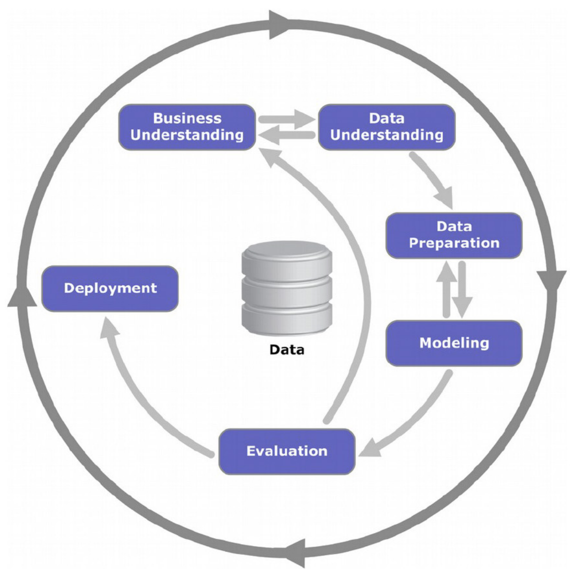


Figure 1: CRISP-DM Model

The CRISP-DM model consists of six stages that can be followed to successfully solve a data science problem. In the first stage, Business Understanding, the needs of a business are identified and project goals and KPIs are defined. As this project is done within a university course, the goals and KPIs were already defined within a proposal and therefore this stage will be skipped. The dataset that will be used within this project stems from the Thomson Reuters Datastream database and includes the quarterly sales and variables from annual financial statements for S&P 500 stock corporations from 2002 to 2022. The financial statements consist of balance sheets with attributes describing the assets, liabilities, and equity as well as profit and loss statements which contain attributes describing the sales and expenses of a particular company. Datastream comes with an Excel interface but is not made to retrieve data in a clean way for multiple companies at once. Thus, an initial data transformation has to be performed to convert the data spreading over several Excel sheets to a few tables before being able to execute the Data Understanding stage of this project. This will be done within this section.

Imports

```r
if(!require(readxl)) {
  install.packages("readxl")
}
```

```
## Loading required package: readxl
```

```r
library(readxl)

if (!require(tidyverse)) {
  install.packages("tidyverse")
}
```

```
## Loading required package: tidyverse

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- 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 become error
```

```r
library(tidyverse)

if (!require(stringr)) {
  install.packages("stringr")
}
library(stringr)
```

Constants

```r
RAW_DATA_PATH <- "../data/raw/datenabzug_sp500.xlsx"
OUTPUT_BASE_PATH <- "../data/processed"
SALES_OUTPUT_PATH <- paste(OUTPUT_BASE_PATH, "sales.csv", sep = "/")
BALANCE_SHEET_OUTPUT_PATH <- paste(OUTPUT_BASE_PATH, "balance_sheet.csv", sep = "/")
PROFIT_LOSS_OUTPUT_PATH <- paste(OUTPUT_BASE_PATH, "profit_loss.csv", sep = "/")
```

1.1 Sales

1.1.1 Load and transform quartely sales data

```r
load_transform_sales_quarter <- function(quarter) {
  # read and transpose
  df <-
    suppressMessages(read_excel(RAW_DATA_PATH, sheet = paste0("Sales Q", quarter)))
  df <- t(df) %>%
    as.data.frame()  %>%
    rownames_to_column("Name")

  colnames(df) <- df[1,]
```

```r
  df <- slice(df, -1)

  # extract company name and variable
  df <- df %>%
    mutate(
      company = str_trim(str_extract(Name, "^.+?(?=\\s-\\s)")),
      # use everything up until " - " as company name
      variable = str_trim(str_extract(Name, "(?<= - ).*(?= -)"))
      # use everything between "-" and "-" as variable name
    ) %>% filter(company != "NA")

  # pivot dataframe
  df <- df %>% select(-Name) %>%
    pivot_longer(
      cols = -c(company, variable),
      names_to = "Name",
      values_to = "interim_sales"
    )

  # extract year and quarter from Name column
  df <- df %>% mutate(year = as.integer(str_extract(Name, "\\s.*")),
                     quarter = as.integer(substr(Name, 2, 3))) %>% select(-Name)

  # remove rows with missing quarter values
  df <- df %>% filter(interim_sales != "NA")

  df$interim_sales <- as.integer(df$interim_sales)

  # remove variable column
  df <- df %>% select(-variable)

  return(df)
}
```

```r
# apply function to all 4 quarters
df_sales <- map_dfr(1:4, load_transform_sales_quarter)
cat(
  paste(
    "Transformed interim sales data frame contains ",
    nrow(df_sales),
    " records for ",
    n_distinct(df_sales$company),
    " companies\nfrom year ",
    min(df_sales$year),
    " to ",
    max(df_sales$year),
    ".",
    sep = ""
  )
)
```

```
## Transformed interim sales data frame contains 34841 records for 500 companies
## from year 2002 to 2022.
```

```r
head(df_sales)
```

```
## # A tibble: 6 x 4
##   company   interim_sales  year quarter
##   <chr>             <int> <int>   <int>
## 1 APPLE INC       1475000  2003       1
## 2 APPLE INC       1909000  2004       1
## 3 APPLE INC       3243000  2005       1
## 4 APPLE INC       4359000  2006       1
## 5 APPLE INC       5264000  2007       1
## 6 APPLE INC       7512000  2008       1
```

1.1.2 Save to CSV

```r
write.csv(df_sales, SALES_OUTPUT_PATH)
```

1.2 Balance sheet

1.2.1 Function to add spaces to some company names

```r
# add space to names of companies without space before the "-"
# without this, we will later have difficulties in correctly parsing the variable names
company_names_without_space <- c(
  "THERMO FISHER SCIENTIFIC",
  "ADOBE (NAS)",
  "CONSTELLATION BRANDS 'A'",
  "WALGREENS BOOTS ALLIANCE",
  "LYONDELLBASELL INDS.CL.A",
  "CITIZENS FINANCIAL GROUP",
  "MID-AMER.APT COMMUNITIES",
  "TERADYNE (XSC)",
  "UNITED AIRLINES HOLDINGS",
  "ALLIANT ENERGY (XSC)",
  "CBOE GLOBAL MARKETS(BTS)",
  "BIO-RAD LABORATORIES 'A'",
  "UNIVERSAL HEALTH SVS.'B'",
  "NEWELL BRANDS (XSC)"
)

company_names_without_space <-
  company_names_without_space[!duplicated(company_names_without_space)]

add_space_to_company_names <-
  function(name, c_names = company_names_without_space) {
    for (c_name in c_names) {
      if (str_detect(name, fixed(c_name))) {
        name = str_replace_all(name, fixed(c_name), paste0(c_name,  " "))
      }
    }
    return(name)
  }
```

1.2.2 Load and transform balance sheet data

```
load_transform_balance_sheet <- function() {
  # read and transpose
  df <- suppressMessages(read_excel(RAW_DATA_PATH, sheet = "Balance Sheet"))
  df <- t(df) %>%
    as.data.frame()  %>%
    rownames_to_column("Name")

  colnames(df) <- df[1, ] # set correct header
  df <- slice(df, -1) # and remove from rows

  # remove entries starting with #ERROR
  df <- df[!startsWith(df$Name, "#ERROR"), ]

  # add space to company names
  df <-
    df %>% rowwise() %>% mutate(Name = add_space_to_company_names(Name))

  # parse company and variable
  df <- df %>%
    mutate(company = str_trim(str_extract(Name, "^.+?(?=\\s-\\s)")),
           variable_name = str_trim(str_extract(Name, "\\s-\\s(?!.*\\s-\\s)(.*)")))
           #variable_name = str_trim(str_extract(Name, "([^-]+$)")))

  # pivot data frame to company, variable_name, year, value format
  # and cast to numeric values to integer
  df <-
    df %>% select(-1) %>% pivot_longer(
      cols = -c(company, variable_name),
      names_to = "year",
      values_to = "value"
    ) %>% mutate(year = as.integer(year), value = as.integer(value))

  # pivot data frame to wide format
  df <-
    df %>% pivot_wider(id_cols = c(company, year), names_from = variable_name)

  # Remove trailing '- ' from colnames
  colnames(df) <-
    sapply(
      colnames(df),
      FUN = function(colname)
        str_replace(colname, "- ", "")
    )

  return(df)
}
```

```
df_balance_sheet <- load_transform_balance_sheet()
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## i In argument: `value = as.integer(value)`.
```

```
## Caused by warning:
## ! NAs introduced by coercion
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
cat(
  paste(
    "Transformed balance sheet data frame contains",
    nrow(df_balance_sheet),
    " records",
    "with",
    ncol(df_balance_sheet),
    "variables\nfor",
    n_distinct(df_balance_sheet$company),
    " companies from year",
    min(df_balance_sheet$year),
    " to",
    max(df_balance_sheet$year),
    "."
  )
)
```

```
## Transformed balance sheet data frame contains 10563  records with 30 variables
## for 503  companies from year 2002  to 2022 .
```

```
head(df_balance_sheet)
```

```
## # A tibble: 6 x 30
##   company  year `BORROWINGS REPAYABLE < 1 YEAR` `EQUITY CAP. AND RESERVES`
##   <chr>   <int>                           <int>                      <int>
## 1 APPLE    2002                               0                    4095000
## 2 APPLE    2003                          304000                    4223000
## 3 APPLE    2004                               0                    5076000
## 4 APPLE    2005                               0                    7466000
## 5 APPLE    2006                               0                    9984000
## 6 APPLE    2007                               0                   14532000
## # i 26 more variables: `NET CURRENT ASSETS` <int>, `NET DEBT` <int>,
## #   `ORDINARY SHARE CAPITAL` <int>, `PREFERENCE CAPITAL` <int>,
## #   `TOTAL RESERVES` <int>, `ASSETS (TOTAL)` <int>,
## #   `TOTAL ASSETS EMPLOYED` <int>, `TOTAL CAPITAL EMPLOYED` <int>,
## #   `TOTAL CASH & EQUIVALENT` <int>, `TOTAL CURRENT ASSETS` <int>,
## #   `TOTAL CURRENT LIABLITIES` <int>, `TOTAL DEBT` <int>,
## #   `TOTAL DEBTORS & EQUIVALENT` <int>, ...
```

1.2.3 Save to CSV

```
write.csv(df_balance_sheet, BALANCE_SHEET_OUTPUT_PATH)
```

1.3. Profit and loss

1.3.1 Load and transform profit and loss data

```
load_transform_profit_loss <- function() {
  for (i in 1:2) {
    df_tmp <-
      suppressMessages(read_excel(RAW_DATA_PATH, sheet = paste("Profit & Loss", i)))
```

```r
    df_tmp <- t(df_tmp) %>%
      as.data.frame()  %>%
      rownames_to_column("Name")

    colnames(df_tmp) <- df_tmp[1, ] # set correct header
    df_tmp <- slice(df_tmp, -1) # and remove from rows

    if (i == 1) {
      df <- df_tmp
      first_iter <- FALSE
    } else {
      df <- rbind(df, df_tmp)
    }
  }

  # remove entries starting with #ERROR
  df <- df[!startsWith(df$Name, "#ERROR"),]

  # add space to company names
  df <-
    df %>% rowwise() %>% mutate(Name = add_space_to_company_names(Name))

  # parse company and variable
  df <- df %>%
    mutate(company = str_trim(str_extract(Name, "^.+?(?=\\s-\\s)")),
           variable_name = str_trim(str_extract(Name, "\\s-\\s(?!.*\\s-\\s)(.*)")))

  # pivot data frame to company, variable_name, year, value format
  # and cast to numeric values to integer
  df <-
    df %>% select(-1) %>% pivot_longer(
      cols = -c(company, variable_name),
      names_to = "year",
      values_to = "value"
    ) %>% mutate(year = as.integer(year), value = as.integer(value))

  # pivot data frame to wide format
  df <-
    df %>% pivot_wider(id_cols = c(company, year), names_from = variable_name)

  # Remove trailing '- ' from colnames
  colnames(df) <-
    sapply(
      colnames(df),
      FUN = function(colname)
        str_replace(colname, "- ", "")
    )

  return(df)
}
```

```r
df_profit_loss <- load_transform_profit_loss()
```

```
## Warning: There was 1 warning in `mutate()`.
```

```
## i In argument: `value = as.integer(value)`.
## Caused by warning:
## ! NAs introduced by coercion
```

```
cat(
  paste(
    "Transformed profit loss data frame contains",
    nrow(df_profit_loss),
    " records",
    "with",
    ncol(df_profit_loss),
    "variables\nfor",
    n_distinct(df_profit_loss$company),
    " companies from year",
    min(df_profit_loss$year),
    " to",
    max(df_profit_loss$year),
    "."
  )
)
```

```
## Transformed profit loss data frame contains 10563  records with 42 variables
## for 503  companies from year 2002  to 2022 .
```

```
head(df_profit_loss)
```

```
## # A tibble: 6 x 42
##   company  year `AFTER TAX PROFIT-ADJ` `A.W.O. INTANGIBLES`
##   <chr>   <int>                  <int>                <int>
## 1 APPLE    2002                  65000                   NA
## 2 APPLE    2003                  68000                   NA
## 3 APPLE    2004                 276000                   NA
## 4 APPLE    2005                1335000                38000
## 5 APPLE    2006                1989000                45000
## 6 APPLE    2007                3496000                68000
## # i 38 more variables: `CASH EARNINGS PER SHARE` <int>, `COST OF SALES` <int>,
## #   DEPRECIATION <int>, `DIVIDENDS PER SHARE` <int>, EBITDA <int>,
## #   `EARNED FOR ORDINARY` <int>, `EARNED FOR ORDINARY-ADJ` <int>, EBIT <int>,
## #   `EXCEPTIONAL ITEMS` <int>, `EXTRAORD. ITEMS AFTER TAX` <int>,
## #   `GROSS PROFIT ON SALES` <int>, `INTEREST CAPITALSED` <int>,
## #   `INTEREST INCOME` <int>, `INTEREST PAID` <int>, `MINORITY INTERESTS` <int>,
## #   `NET INTEREST CHARGES` <int>, `OPERATING PROFIT` <int>, ...
```

1.3.2 Save to CSV

```
write.csv(df_profit_loss, PROFIT_LOSS_OUTPUT_PATH)
```

2. Data Understanding

The Data Understanding stage is the first CRISP-DM stage we will perform within this project. The first goal of this stage is to examine the available data regarding properties like format, number of records or number of variables. The second goal is to gain a deeper understanding of the data by e.g., visualizing distributions and changes over time or performing correlation analysis. The final goal of this stage is to assess the data quality and to define tasks for the data preparation stage by using the insights that were gained by exploring the data. The data analysis will be performed separately for the quarterly sales, the balance sheet and the profit and loss data frames.

Imports

```
if(!require(tidyverse)) {
  install.packages("tidyverse")
}
library(tidyverse)

if (!require(modeest)) {
  install.packages("modeest")
}
```

```
## Loading required package: modeest
```

```
library(modeest)

if (!require(zoo)) {
  install.packages("zoo")
}
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(zoo)

if (!require(ggplot2)) {
  install.packages("ggplot2")
}
library(ggplot2)

if (!require(corrplot)) {
  install.packages("corrplot")
}
```

```
## Loading required package: corrplot

## corrplot 0.92 loaded
```

```
library(corrplot)

if(!require(devtools)) {
  install.packages("devtools")
}
```

```
## Loading required package: devtools

## Loading required package: usethis
```

```r
library(devtools)

if (!require(lares)) {
  # install lares correlation package from github
  devtools::install_github("laresbernardo/lares")
}
```

```
## Loading required package: lares

## Registered S3 method overwritten by 'httr':
##   method         from
##   print.response rmutil
```

```r
library(lares)
```

Constants

```r
Sys.unsetenv("LARES_FONT")
BASE_PATH <- "../data/processed"
SALES_PATH <- paste(BASE_PATH, "sales.csv", sep = "/")
BALANCE_SHEET_PATH <- paste(BASE_PATH, "balance_sheet.csv", sep = "/")
PROFIT_LOSS_PATH <- paste(BASE_PATH, "profit_loss.csv", sep = "/")
```

2.1. Sales

2.1.1 Load data

```r
df_sales <- read_csv(SALES_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```r
df_sales <- df_sales[, -1] # remove index column
head(df_sales)
```

```
## # A tibble: 6 x 4
##   company   interim_sales  year quarter
##   <chr>             <dbl> <dbl>   <dbl>
## 1 APPLE INC       1475000  2003       1
## 2 APPLE INC       1909000  2004       1
## 3 APPLE INC       3243000  2005       1
## 4 APPLE INC       4359000  2006       1
## 5 APPLE INC       5264000  2007       1
## 6 APPLE INC       7512000  2008       1
```

2.1.2 Show data description

First of all, we will show a general description of the data frame including the variable types, number of distinct values, minimums, maximums, number of records and variables.

```
## The data frame contains 34841 rows with 4 variables for 500  distinct companies from 2002 to 2022 .

##                     type n_distinct        min       max
## company        character        500 3M COMPANY    ZOETIS
```

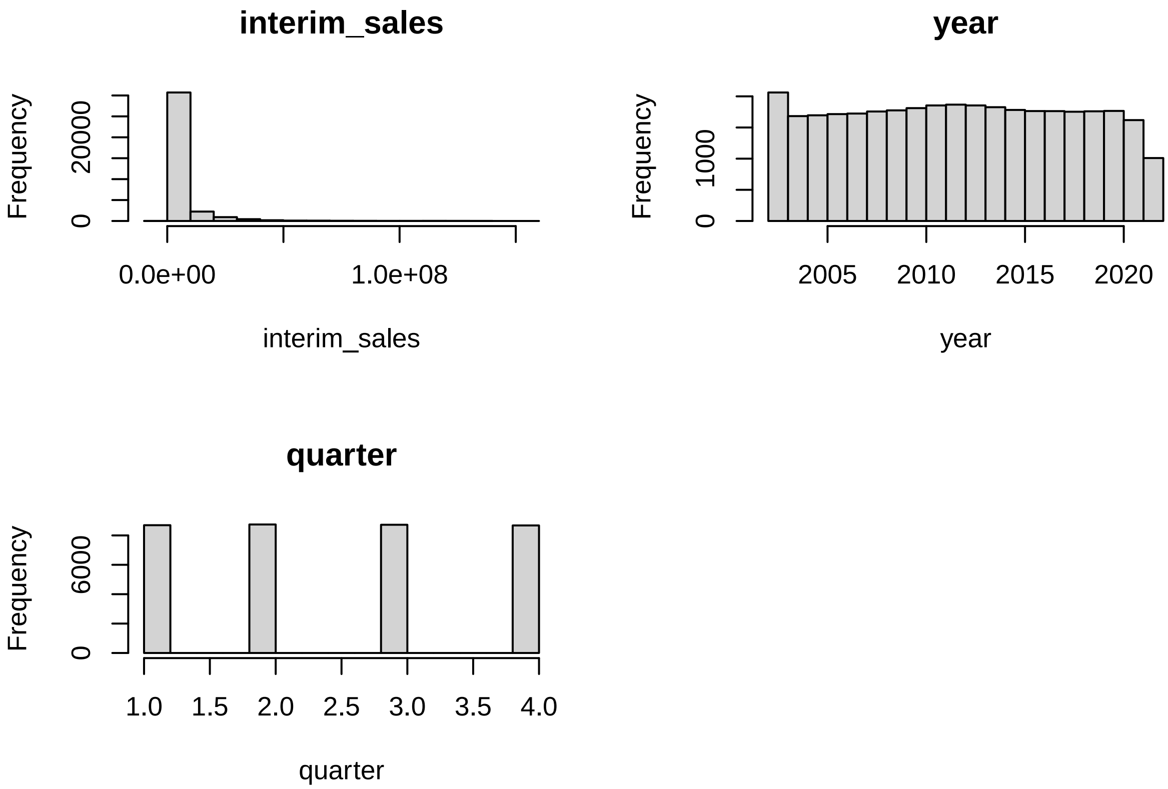
```
## interim_sales   numeric      29190    -393000 152859000
## year            numeric         21       2002      2022
## quarter         numeric          4          1         4
```

The output above shows that we have 34841 records with 4 variables for 500 distinct companies from 2002 to 2022. This equals a time range of 21 years. The company names are of type character. The interim_sales, year and quarter variables are of numeric type.

2.1.3 Distributions

The second step will be to visualize distributions of the numeric variables of the data frame. This can help to get a deeper understanding of the variables and lead to tasks that have to be done in the data preparation stage.

```
show_data_distribution <- function(df) {
  # get numeric columns of data frame
  num_cols <- sapply(df, is.numeric)

  # set 2x2 plot grid
  plot_grid <- c(2, 2)
  if (sum(num_cols) > 4) {
    # use 3x3 grid, if there are > 4 numeric variables
    plot_grid <- c(3, 3)
  }
  par(mfrow = plot_grid)

  # show histogram of numeric columns
  for (i in which(num_cols)) {
    tmp_variable <- as.numeric(unlist(df[, i]))
    hist(tmp_variable,
         main = names(df)[i],
         xlab = names(df)[i])
  }
}

show_data_distribution(df_sales)
```



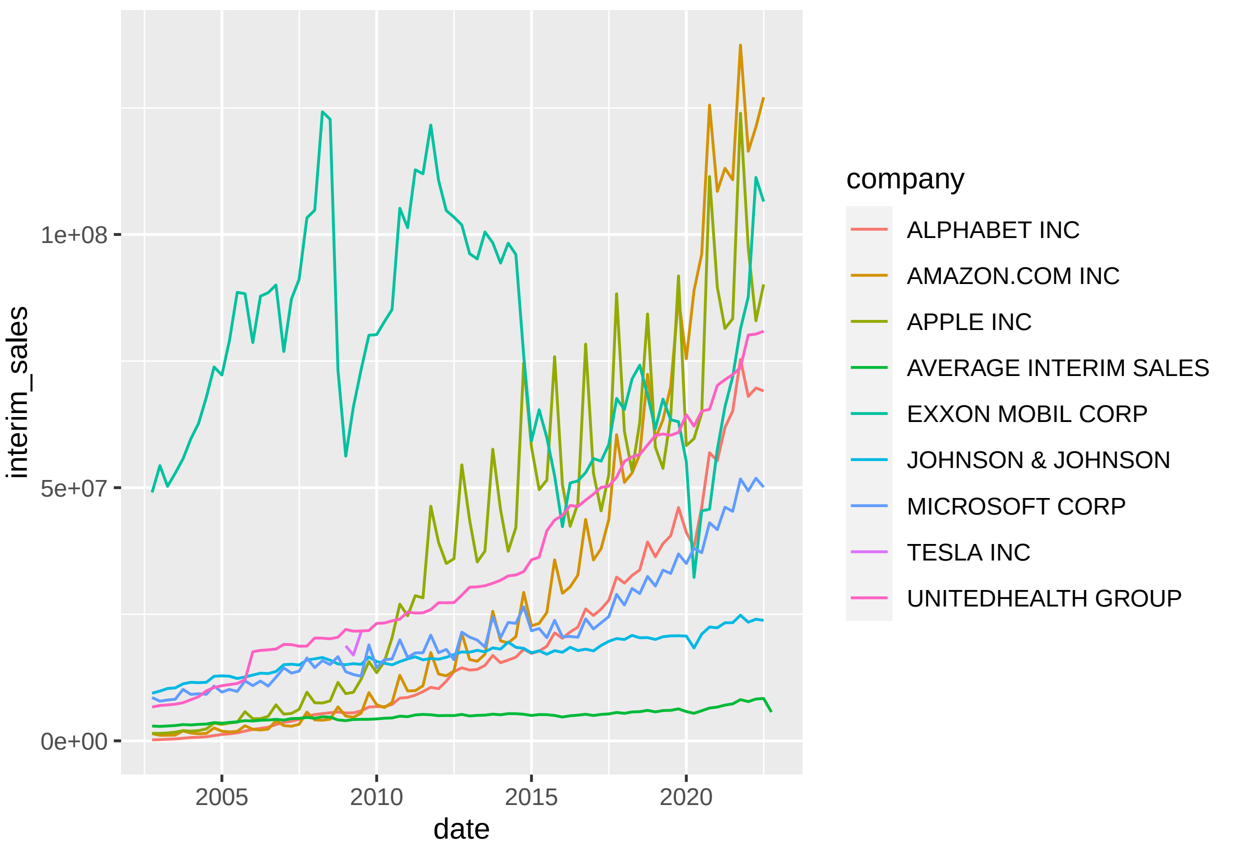
In the plots above, we can see a uniform distribution for the year and quarter variable. For the interim_sales variable, we can observe a right skewed distribution. This suggests, that we log transform this variable in the data preparation step, as this makes the distribution more symmetrical.

2.1.4 Change over time

Now we will visualize changes over time. As we have distinct sales time series' for each company, this visualization is only valuable if we draw a line for each distinct company separately. As 500 companies in one plot would only lead to visual clutter, we will only show a selection of 8 companies together with the average over all companies.

```r
# create date from year and quarter
df_sales$date <-
  as.Date(as.yearqtr(paste0(df_sales$year, "-", df_sales$quarter), format = "%Y-%q"))

# calculate average over all companies
average_interim_sales <-
  df_sales %>% group_by(date) %>% summarise(interim_sales = mean(interim_sales))
average_interim_sales$company <- "AVERAGE INTERIM SALES"

# define selection of companies as we cannot visualize all companies
selected_companies <-
  c(
    "APPLE INC",
    "MICROSOFT CORP",
    "AMAZON.COM INC",
    "TESLA INC",
    "ALPHABET INC",
    "UNITEDHEALTH GROUP",
    "EXXON MOBIL CORP",
```

```
    "JOHNSON & JOHNSON"
  )
df_sales_selected <-
  df_sales[df_sales$company %in% selected_companies,
           c("date", "interim_sales", "company")]

# bind selected companies with average over all companies
df_sales_selected <- rbind(df_sales_selected, average_interim_sales)


# show line plot
ggplot(df_sales_selected, aes(x = date, y = interim_sales)) +
  geom_line(aes(color = company))
```



The visualization of the selected companies looks reasonable. We can see that from 2010 on, all of the selected time series' have a higher interim sales value than the average. For TESLA INC we can observe a time series which is very short. As we cannot visualize all companies, we will try to find short companies in a programmatic (and not visual) way in the next analysis step.

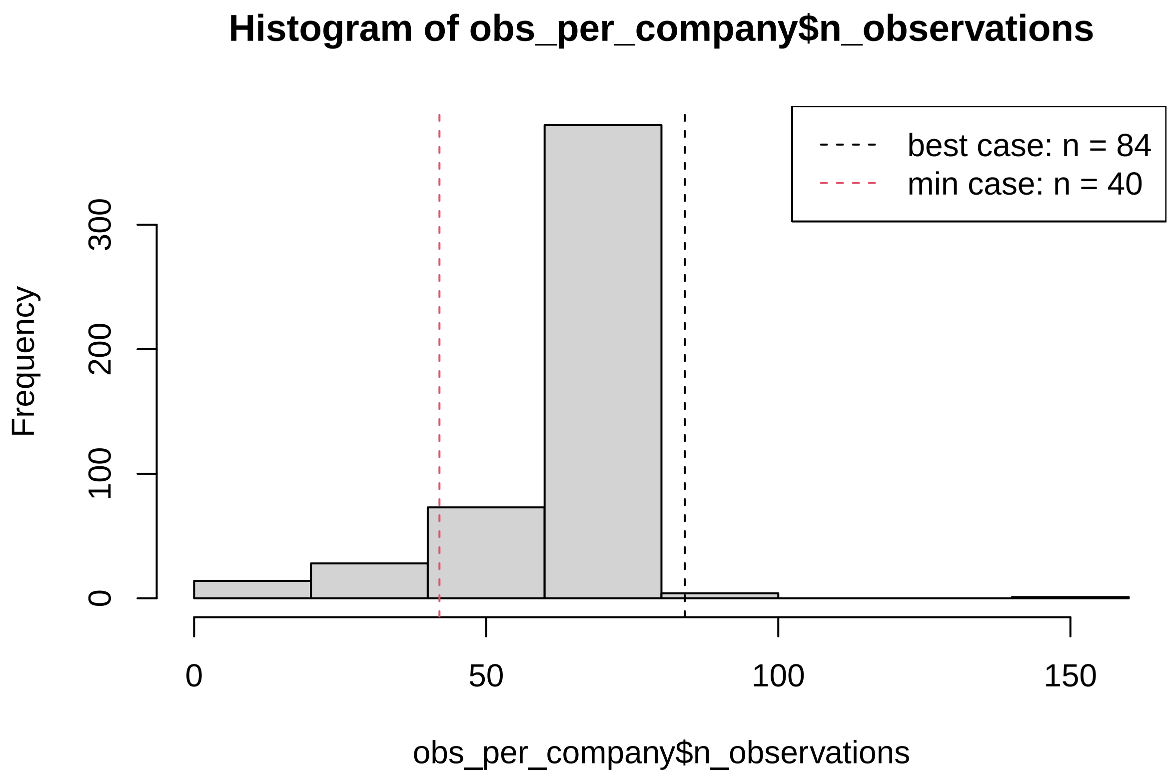
2.1.5 Investigate number of observations and continuity per company

The data set ranges from 2002 to 2022, which equals 21 distinct years. As we have 4 quarters in each year, each company should (in the best case) carry $21 * 4 = 84$ observations. Later on, we want to use 5 years for the evaluation of our prediction models. This equals $4 * 5 = 20$ observations. To train and calibrate the models, we need some training data. This data set should carry at least as many observations as the evaluation set, therefore the minimum number of observations for a company is 40.

```r
# calculate observations per company
obs_per_company <-
  df_sales %>% group_by(df_sales$company) %>%
  summarise(n_observations = n()) %>%
  arrange(n_observations)

# rename column
colnames(obs_per_company)[1] <- "company"
head(obs_per_company)
```

```
## # A tibble: 6 x 2
##   company              n_observations
##   <chr>                         <int>
## 1 ULTA BEAUTY INC                   2
## 2 PAYPAL HOLDINGS INC               3
## 3 SOLAREDGE TECH                    3
## 4 TARGA RESOURCES                   3
## 5 TESLA INC                         3
## 6 VICI PROPERTIES                   3
```

```r
n_best <- 84 # 21 years * 4 quarters = 84 observations per company
n_min <- 40 # at least 5 years for training and 5 years for evaluation * 4 = 40

# show histogram of number of observations
hist(obs_per_company$n_observations)
abline(v = 84, lty = 2, col = 1)
abline(v = 42, lty = 2, col = 2)
legend(
  "topright",
  legend = c(paste("best case: n =", n_best), paste("min case: n =", n_min)),
  lty = 2,
  col = c(1, 2)
)
```




```
cat("Found", paste(nrow(obs_per_company[obs_per_company$n_observations < n_min, ])),
    "companies with less than", paste(n_min), "observations.")
```

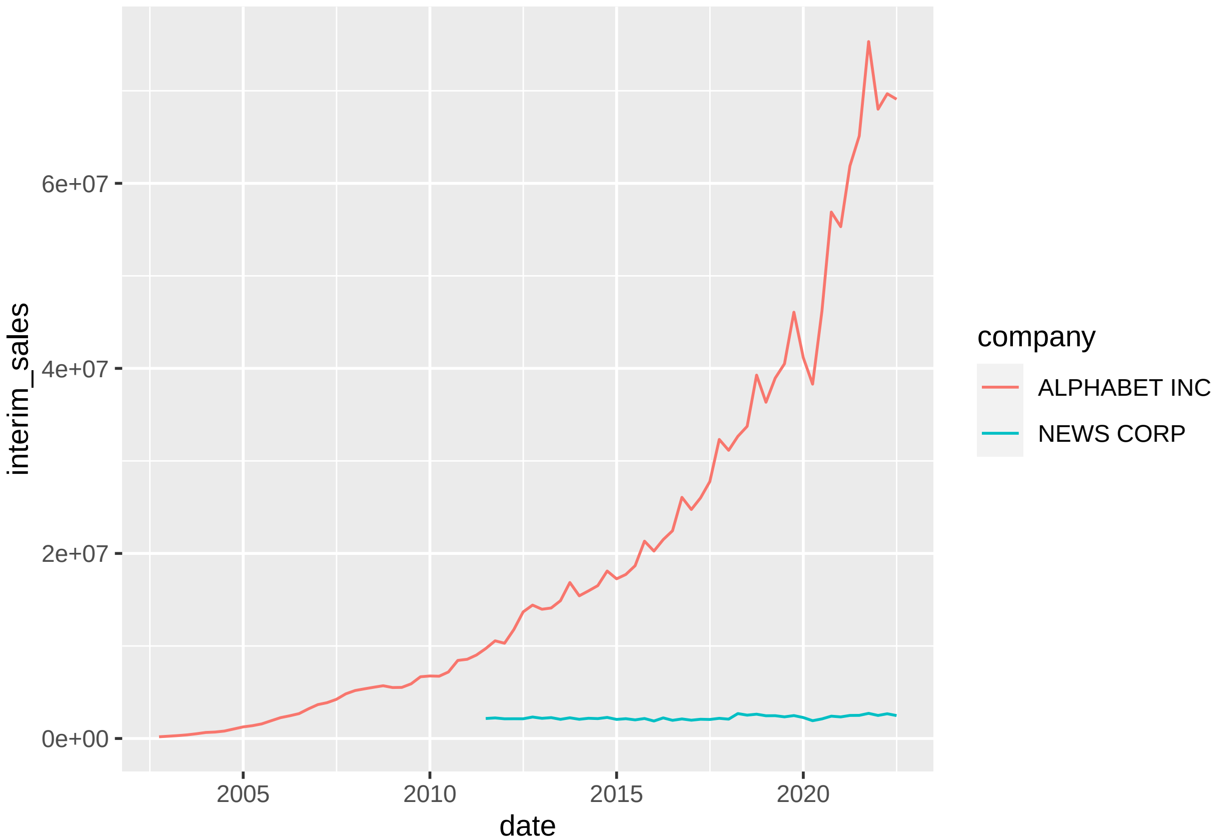
```
## Found 36 companies with less than 40 observations.
```

We can see, that there are 36 companies with less than 40 observations. We have to exclude them in the data preparation stage. We can also see, that there are companies with more than 84 observations. This is strange and has to be investigated in the next step by plotting those companies.

```
# show companies with n_observations > 84
obs_per_company[obs_per_company$n_observations > n_best,]
```

```
## # A tibble: 2 x 2
##   company       n_observations
##   <chr>                  <int>
## 1 NEWS CORP                 90
## 2 ALPHABET INC             160
```

```
# plot
companies_too_many_obs <- obs_per_company[obs_per_company$n_observations > n_best,]$company

# show line plot
ggplot(df_sales[df_sales$company %in% companies_too_many_obs, ], aes(x = date, y = interim_sales)) +
  geom_line(aes(color = company))
```



The displayed time series' look reasonable. A possible reason for the high number of observations for those two companies could be duplicates in the data. We will investigate this below.

```
for (company in companies_too_many_obs){
  cat("Found",
      paste(sum(duplicated(df_sales[df_sales$company == company, ]))),
      "duplicated records for company",
      paste(company),
      "\n"
      )
}
```

```
## Found 45 duplicated records for company NEWS CORP
## Found 80 duplicated records for company ALPHABET INC
```

We can indeed see that the number of observations is too high for those companies because of duplicated records. We will have to remove them in the data preparation step.

Let's now investigate if the dates are continuous, i.e., if there are missing values in between the time series of quarterly sales. For that, we order the time series by date for each company separately and calculate the time difference between each consecutive observation. If it is not equal to 3 months, the time series is not continuous and has to be interpolated later on.

```
# create new column that marks if ts is continuous,
# set to true for all companies in the beginning
obs_per_company$is_continous <- TRUE

for (company in unique(df_sales$company)) {
  # for each company
```

```r
  company_dates <- df_sales[df_sales$company == company,]$date
  company_dates <- sort(company_dates) # select sorted dates
  for (i in 1:(length(company_dates) - 1)) {
    # for each date index
    difference_in_days <-
      as.integer(company_dates[i + 1] - company_dates[i])
    # in case there are two months with 31 days in the quarter,
    # the maximum valid difference in days is 92
    if (difference_in_days > 92) {
      obs_per_company[obs_per_company$company == company, "is_continous"] <-
        FALSE
      break
    }
  }
}

cat("Found",
    paste(nrow(obs_per_company[obs_per_company$is_continous == FALSE, ])),
    "companies that do not have a continous sales time series.")
```

```
## Found 69 companies that do not have a continous sales time series.
```

Those 69 companies that do not have a continuous time series have to be interpolated in the data preparation stage.

2.1.6 Data quality assesment

Now we will assess the data quality by counting the number of missing values for each variable in the data frame.

```r
assess_data_quality <- function(df) {
  quality_df <- data.frame(
    absolute_missing_values = colSums(is.na.data.frame(df)),
    relative_missing_values = colSums(is.na.data.frame(df)) / nrow(df)
  )
  return(quality_df[order(quality_df$relative_missing_values,
                          decreasing = TRUE), ])
}

assess_data_quality(df_sales)
```

```
##               absolute_missing_values relative_missing_values
## company                             0                       0
## interim_sales                       0                       0
## year                                0                       0
## quarter                             0                       0
## date                                0                       0
```

We can not observe any variables with missing values for the quarterly sales data frame.

2.1.7 Tasks for data preparation

After the analysis of the sales data frame, we carry on the following tasks to the data preparation stage: 1. Log transform interim_sales variable 2. Exclude companies with less than 40 observations 3. Interpolate companies with non-continuous time series' 4. Remove duplicates for NEWS CORP and ALPHABET INC

2.2. Balance sheet

2.2.1 Load data

```
df_balance_sheet <- read_csv(BALANCE_SHEET_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```
df_balance_sheet <- df_balance_sheet[, -1]
head(df_balance_sheet)
```

```
## # A tibble: 6 x 30
##   company  year `BORROWINGS REPAYABLE < 1 YEAR` `EQUITY CAP. AND RESERVES`
##   <chr>   <dbl>                           <dbl>                      <dbl>
## 1 APPLE    2002                               0                    4095000
## 2 APPLE    2003                          304000                    4223000
## 3 APPLE    2004                               0                    5076000
## 4 APPLE    2005                               0                    7466000
## 5 APPLE    2006                               0                    9984000
## 6 APPLE    2007                               0                   14532000
## # i 26 more variables: `NET CURRENT ASSETS` <dbl>, `NET DEBT` <dbl>,
## #   `ORDINARY SHARE CAPITAL` <dbl>, `PREFERENCE CAPITAL` <dbl>,
## #   `TOTAL RESERVES` <dbl>, `ASSETS (TOTAL)` <dbl>,
## #   `TOTAL ASSETS EMPLOYED` <dbl>, `TOTAL CAPITAL EMPLOYED` <dbl>,
## #   `TOTAL CASH & EQUIVALENT` <dbl>, `TOTAL CURRENT ASSETS` <dbl>,
## #   `TOTAL CURRENT LIABLITIES` <dbl>, `TOTAL DEBT` <dbl>,
## #   `TOTAL DEBTORS & EQUIVALENT` <dbl>, ...
```

2.2.2 Show data description

```
show_data_description(df_balance_sheet)
```

```
## The data frame contains 10563 rows with 30 variables for 503  distinct companies from 2002 to 2022 .
```

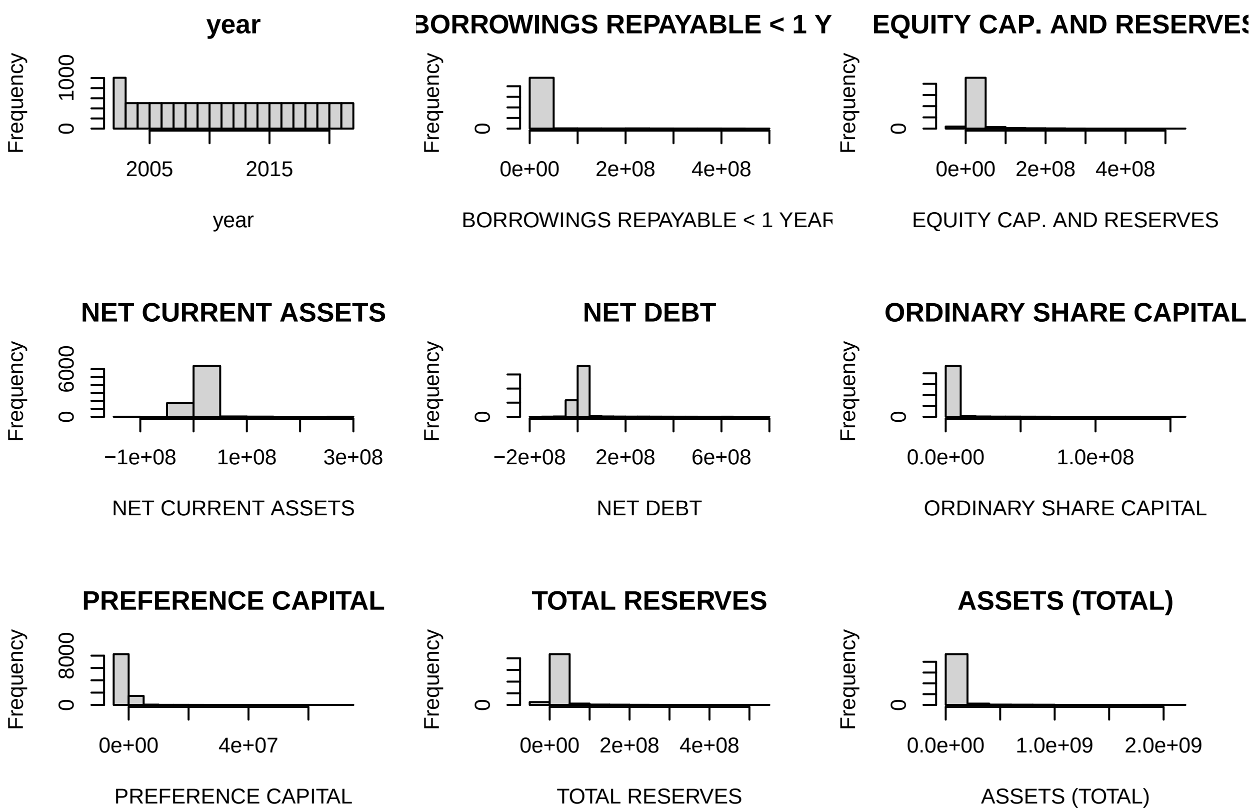
```
##                                    type n_distinct        min        max
## company                       character        503         3M   ZOETIS A
## year                            numeric         21       2002       2022
## BORROWINGS REPAYABLE < 1 YEAR   numeric       6323          0  484315900
## EQUITY CAP. AND RESERVES        numeric       9558  -25560000  506198800
## NET CURRENT ASSETS              numeric       7617 -149782000  290101800
## NET DEBT                        numeric       9485 -173495000  772553000
## ORDINARY SHARE CAPITAL          numeric       4345          0  158142000
## PREFERENCE CAPITAL              numeric        845     -49000   72148000
## TOTAL RESERVES                  numeric       9269  -40796990  506190800
## ASSETS (TOTAL)                  numeric       9685       1893 2119852000
## TOTAL ASSETS EMPLOYED           numeric       9617  -24160000  616640800
## TOTAL CAPITAL EMPLOYED          numeric       9617  -24160000  616640800
## TOTAL CASH & EQUIVALENT         numeric       8663          0  722433800
## TOTAL CURRENT ASSETS            numeric       7777       4693  413188900
## TOTAL CURRENT LIABLITIES        numeric       7714        984  248610000
## TOTAL DEBT                      numeric       8747          0  810758900
## TOTAL DEBTORS & EQUIVALENT      numeric       7369          0  263328000
## TOTAL DEFERRED & FUTURE TAX     numeric       7186  -55032000   89678990
## TOT FIXED ASSETS-NET            numeric       9049          0  259651000
## TOTAL INTANGIBLES               numeric       7963          0  310197000
```

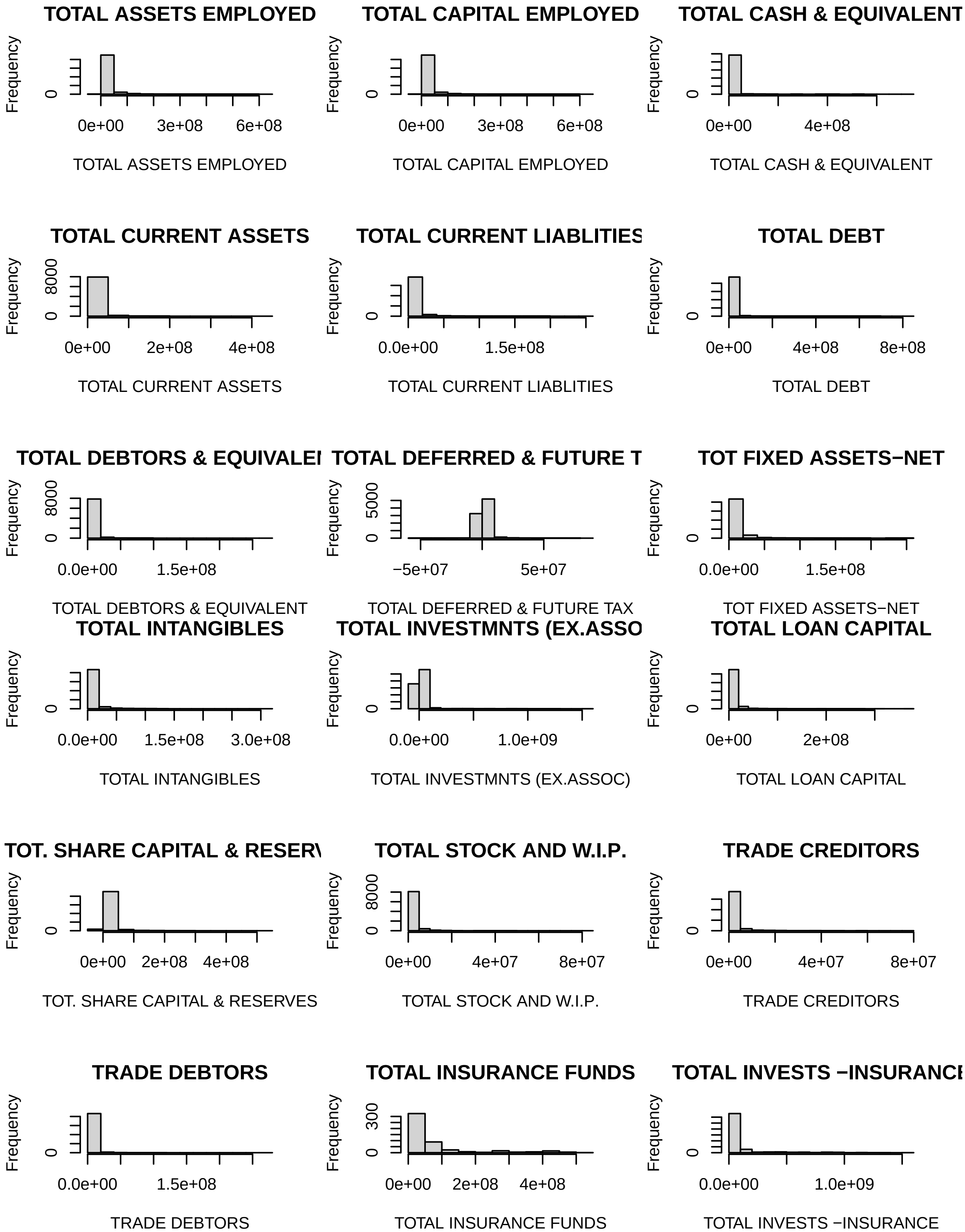
```
## TOTAL INVESTMNTS (EX.ASSOC)      numeric       4608  -23979870 1591976000
## TOTAL LOAN CAPITAL               numeric       8555          0  377137900
## TOT. SHARE CAPITAL & RESERVES    numeric       9550  -25560000  506198800
## TOTAL STOCK AND W.I.P.           numeric       5843          0   81714990
## TRADE CREDITORS                  numeric       6938          0   78664000
## TRADE DEBTORS                    numeric       7808          0  263328000
## TOTAL INSURANCE FUNDS            numeric        497      59565  523148800
## TOTAL INVESTS -INSURANCE         numeric       1471          0 1591976000
## CURRENT, DEPOSIT & OTHER A/CS    numeric        567          0 2144257000
## TOTAL ADVANCES                   numeric        631          0 1061328000
```

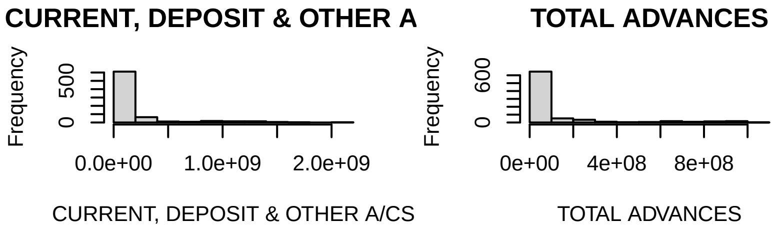
We can observe that the balance sheet data frame contains 10563 records with 30 variables. All of the variables except the company are of numeric type.

2.2.3 Distributions

```
show_data_distribution(df_balance_sheet)
```




TOTAL ASSETS EMPLOYED
Frequency
0
0e+00
3e+08
6e+08
TOTAL ASSETS EMPLOYED
TOTAL CAPITAL EMPLOYED
Frequency
0
0e+00
3e+08
6e+08
TOTAL CAPITAL EMPLOYED
TOTAL CASH & EQUIVALENT
Frequency
0
0e+00
4e+08
TOTAL CASH & EQUIVALENT
TOTAL CURRENT ASSETS
Frequency
0
8000
0e+00
2e+08
4e+08
TOTAL CURRENT ASSETS
TOTAL CURRENT LIABLITIES
Frequency
0
0.0e+00
1.5e+08
TOTAL CURRENT LIABLITIES
TOTAL DEBT
Frequency
0
0e+00
4e+08
8e+08
TOTAL DEBT
TOTAL DEBTORS & EQUIVALEI
Frequency
0
8000
0.0e+00
1.5e+08
TOTAL DEBTORS & EQUIVALENT
TOTAL DEFERRED & FUTURE T
Frequency
0
5000
−5e+07
5e+07
TOTAL DEFERRED & FUTURE TAX
TOT FIXED ASSETS−NET
Frequency
0
0.0e+00
1.5e+08
TOT FIXED ASSETS−NET
TOTAL INTANGIBLES
Frequency
0
0.0e+00
1.5e+08
3.0e+08
TOTAL INTANGIBLES
TOTAL INVESTMNTS (EX.ASSO
Frequency
0
0.0e+00
1.0e+09
TOTAL INVESTMNTS (EX.ASSOC)
TOTAL LOAN CAPITAL
Frequency
0
0e+00
2e+08
TOTAL LOAN CAPITAL
TOT. SHARE CAPITAL & RESER\
Frequency
0
0e+00
2e+08
4e+08
TOT. SHARE CAPITAL & RESERVES
TOTAL STOCK AND W.I.P.
Frequency
0
8000
0e+00
4e+07
8e+07
TOTAL STOCK AND W.I.P.
TRADE CREDITORS
Frequency
0
0e+00
4e+07
8e+07
TRADE CREDITORS
TRADE DEBTORS
Frequency
0
0.0e+00
1.5e+08
TRADE DEBTORS
TOTAL INSURANCE FUNDS
Frequency
0
300
0e+00
2e+08
4e+08
TOTAL INSURANCE FUNDS
TOTAL INVESTS −INSURANCI
Frequency
0
0.0e+00
1.0e+09
TOTAL INVESTS −INSURANCE



In the histograms above, we can see that many of the variables are highly right skewed. As for the sales data, we will use a log transformation for those variables in the data preparation step.

2.2.4 Investigate number of observations and continuity per company

Now we will investigate the number of observations and the continuity per company as we also did it for the sales data frame. In contrast to the sales data frame, where we got data in a quarterly frequency, the balance sheet data is on a yearly frequency. Therefore our best case number of observations is 21.

```
# calculate observations per company
obs_per_company <-
  df_balance_sheet %>%
  group_by(df_balance_sheet$company) %>%
  summarise(n_observations = n()) %>%
  arrange(n_observations)
head(obs_per_company)
```

```
## # A tibble: 6 x 2
##   `df_balance_sheet$company` n_observations
##   <chr>                               <int>
## 1 3M                                     21
## 2 ABBOTT LABORATORIES                    21
## 3 ABBVIE                                 21
## 4 ABIOMED                                21
## 5 ACCENTURE CLASS A                      21
## 6 ACTIVISION BLIZZARD                    21
```

```
cat("Minimum number of observations", paste(min(obs_per_company$n_observations)), "\n")
```

```
## Minimum number of observations 21
```

```
cat("Maximum number of observations", paste(max(obs_per_company$n_observations)), "\n")
```

```
## Maximum number of observations 21
```

The table above is sorted in ascending order, therefore we can already see, that there are no companies with not enough observations. The console output also confirms that there are no companies with too many observations. A quick check of if there are duplicates will show if our time series' is continuous and has no gaps in between.

```
cat("Found",
    paste(sum(duplicated(df_balance_sheet))),
    "duplicated records in the balance sheet data frame.")
```

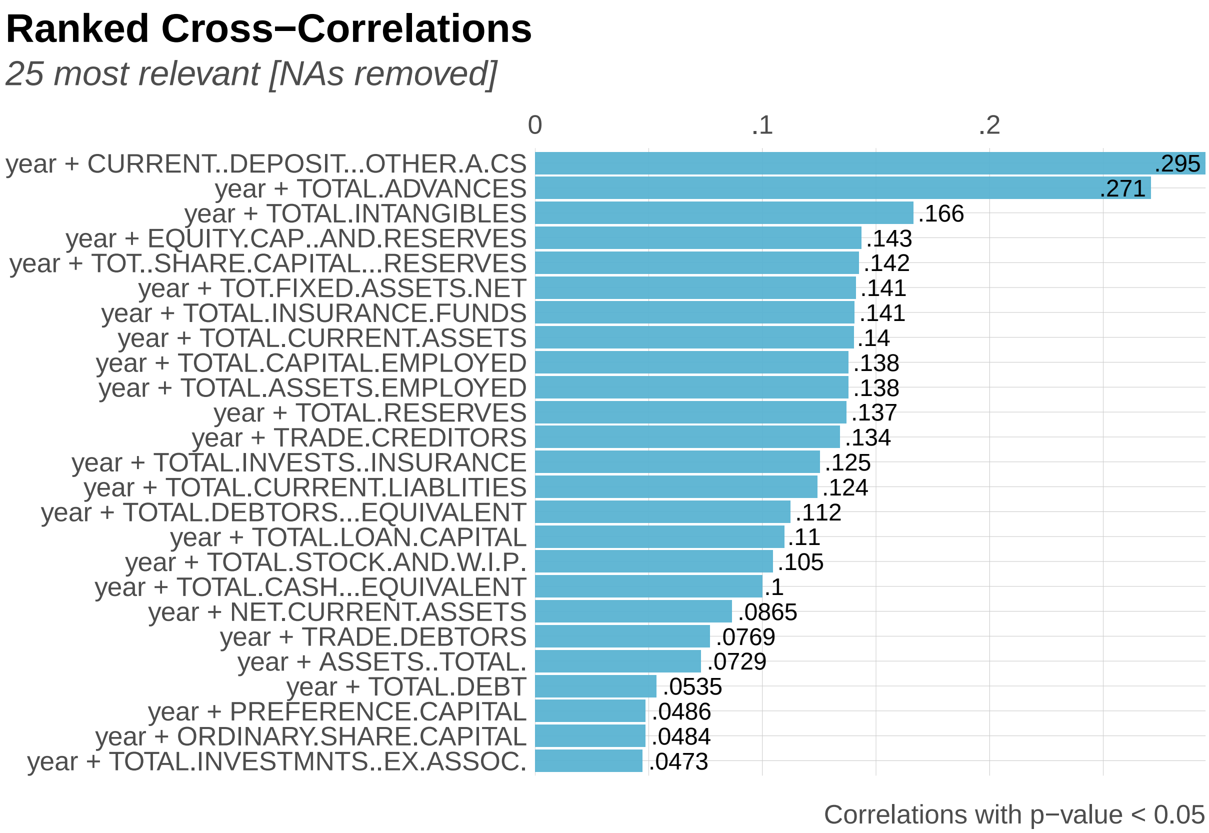
```
## Found 0 duplicated records in the balance sheet data frame.
```

2.2.5 Correlation analysis

Due to the high number of variables in the data frame, visualizing a correlation matrix leads to visual clutter and is therefore difficult. Thus we make use of the lares library which creates a table of the top 25 variable pairs ranked by their correlation value. Additionally, a significance test (at the 5% level) for the correlations is performed.

```
dfbs_num <- df_balance_sheet[, sapply(df_balance_sheet, is.numeric)] # obtain only numeric columns

corr_cross(dfbs_num, # name of dataset
  max_pvalue = 0.05, # display only significant correlations (at 5% level)
  rm.na = TRUE, # remove NAs
  top = 25 # show only top 25 variable pairs
)
```

```
## Returning only the top 25. You may override with the 'top' argument
```

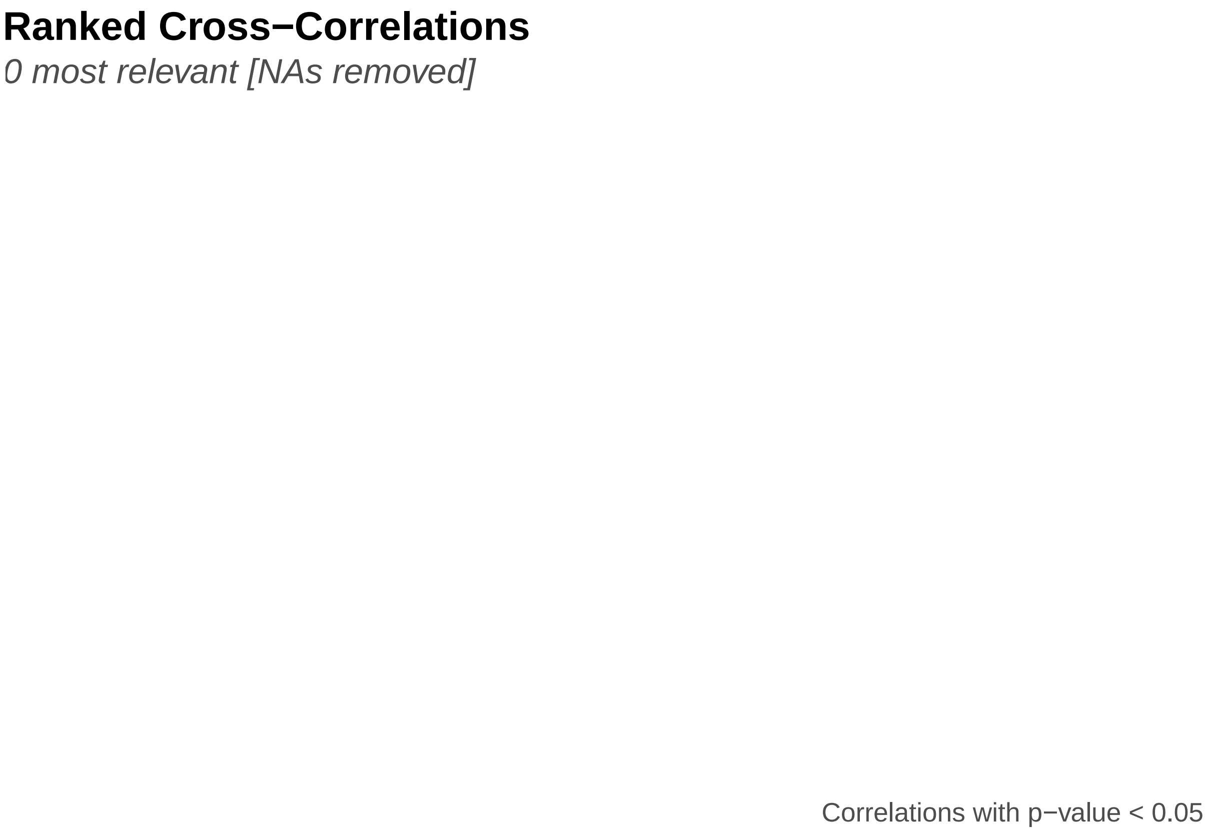


In the plot above, we can see that all of the top 25 correlations of the balance sheet variables include the year variable. The two pairs with the highest positive correlation value are year with CURRENT, DEPOSIT & OTHER A/CS and year with TOTAL ADVANCES. The remaining correlations can be neglected due to their low value. Let's now investigate if there are also correlations in between the variables by excluding the year variable from the analysis.

```
dfbs_num <- dfbs_num[, -1] # remove year variable

corr_cross(dfbs_num, # name of dataset
  max_pvalue = 0.05, # display only significant correlations (at 5% level)
  rm.na = TRUE, # remove NAs
  top = 25 # show only top 25 variable pairs
)
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `hjust = ifelse(.data$abs < max(.data$abs)/1.5, -0.1, 1.1)`.
## Caused by warning in `max()`:
```

```
## ! no non-missing arguments to max; returning -Inf
```



We can observe that there are no significant correlations between the variables if we exclude the year variable. This is good as we do not have to perform variable selection for this table, as we would have to do if there were high correlations due to possible instabilities when modeling the data.

2.2.6 Data quality assesment

```r
assess_data_quality(df_balance_sheet)
```

```
##                                absolute_missing_values relative_missing_values
## TOTAL INSURANCE FUNDS                            10067              0.95304364
## CURRENT, DEPOSIT & OTHER A/CS                     9813              0.92899744
## TOTAL ADVANCES                                    9751              0.92312790
## TOTAL INVESTS -INSURANCE                          8969              0.84909590
## TOTAL DEBTORS & EQUIVALENT                        2470              0.23383508
## TRADE CREDITORS                                   2423              0.22938559
## TOTAL CURRENT ASSETS                              2400              0.22720818
## TOTAL CURRENT LIABLITIES                          2400              0.22720818
## NET CURRENT ASSETS                                2399              0.22711351
## TOTAL DEFERRED & FUTURE TAX                       1833              0.17353025
## TOTAL STOCK AND W.I.P.                            1833              0.17353025
## TRADE DEBTORS                                     1701              0.16103380
## TOTAL INTANGIBLES                                 1102              0.10432642
## TOTAL INVESTMNTS (EX.ASSOC)                       1059              0.10025561
## TOTAL RESERVES                                     976              0.09239799
## ORDINARY SHARE CAPITAL                             975              0.09230332
## TOT FIXED ASSETS-NET                               868              0.08217362
```

```
## BORROWINGS REPAYABLE < 1 YEAR                     807           0.07639875
## PREFERENCE CAPITAL                                733           0.06939316
## NET DEBT                                          713           0.06749976
## TOTAL LOAN CAPITAL                                702           0.06645839
## TOTAL DEBT                                        700           0.06626905
## ASSETS (TOTAL)                                    696           0.06589037
## TOTAL CASH & EQUIVALENT                           695           0.06579570
## EQUITY CAP. AND RESERVES                          693           0.06560636
## TOTAL ASSETS EMPLOYED                             693           0.06560636
## TOTAL CAPITAL EMPLOYED                            693           0.06560636
## TOT. SHARE CAPITAL & RESERVES                     687           0.06503834
## company                                             0           0.00000000
## year                                                0           0.00000000
```

We can observe that all variables, except for company and year contain missing values. For some variables, we will perform data imputation in the data preparation step. To keep the imputation effort manageable and to not introduce artifacts by imputing variables, where too much data is missing. We will remove variables where more than 20% of the records have missing values.

2.2.7 Tasks for data preparation

After the analysis of the balance sheet data frame, we carry on the following tasks to the data preparation stage: 1. Remove variables where > 20% of values are missing 2. Log transform right-skewed variables 3. Impute missing values for other variables

2.3. Profit and loss

2.3.1 Load data

```r
df_profit_loss <- read_csv(PROFIT_LOSS_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```r
df_profit_loss <- df_profit_loss[, -1]
head(df_profit_loss)
```

```
## # A tibble: 6 x 42
##   company  year `AFTER TAX PROFIT-ADJ` `A.W.O. INTANGIBLES`
##   <chr>   <dbl>                  <dbl>                <dbl>
## 1 APPLE    2002                  65000                   NA
## 2 APPLE    2003                  68000                   NA
## 3 APPLE    2004                 276000                   NA
## 4 APPLE    2005                1335000                38000
## 5 APPLE    2006                1989000                45000
## 6 APPLE    2007                3496000                68000
## # i 38 more variables: `CASH EARNINGS PER SHARE` <dbl>, `COST OF SALES` <dbl>,
## #   DEPRECIATION <dbl>, `DIVIDENDS PER SHARE` <dbl>, EBITDA <dbl>,
## #   `EARNED FOR ORDINARY` <dbl>, `EARNED FOR ORDINARY-ADJ` <dbl>, EBIT <dbl>,
## #   `EXCEPTIONAL ITEMS` <dbl>, `EXTRAORD. ITEMS AFTER TAX` <dbl>,
## #   `GROSS PROFIT ON SALES` <dbl>, `INTEREST CAPITALSED` <dbl>,
## #   `INTEREST INCOME` <dbl>, `INTEREST PAID` <dbl>, `MINORITY INTERESTS` <dbl>,
## #   `NET INTEREST CHARGES` <dbl>, `OPERATING PROFIT` <dbl>, ...
```

2.3.2 Show data description

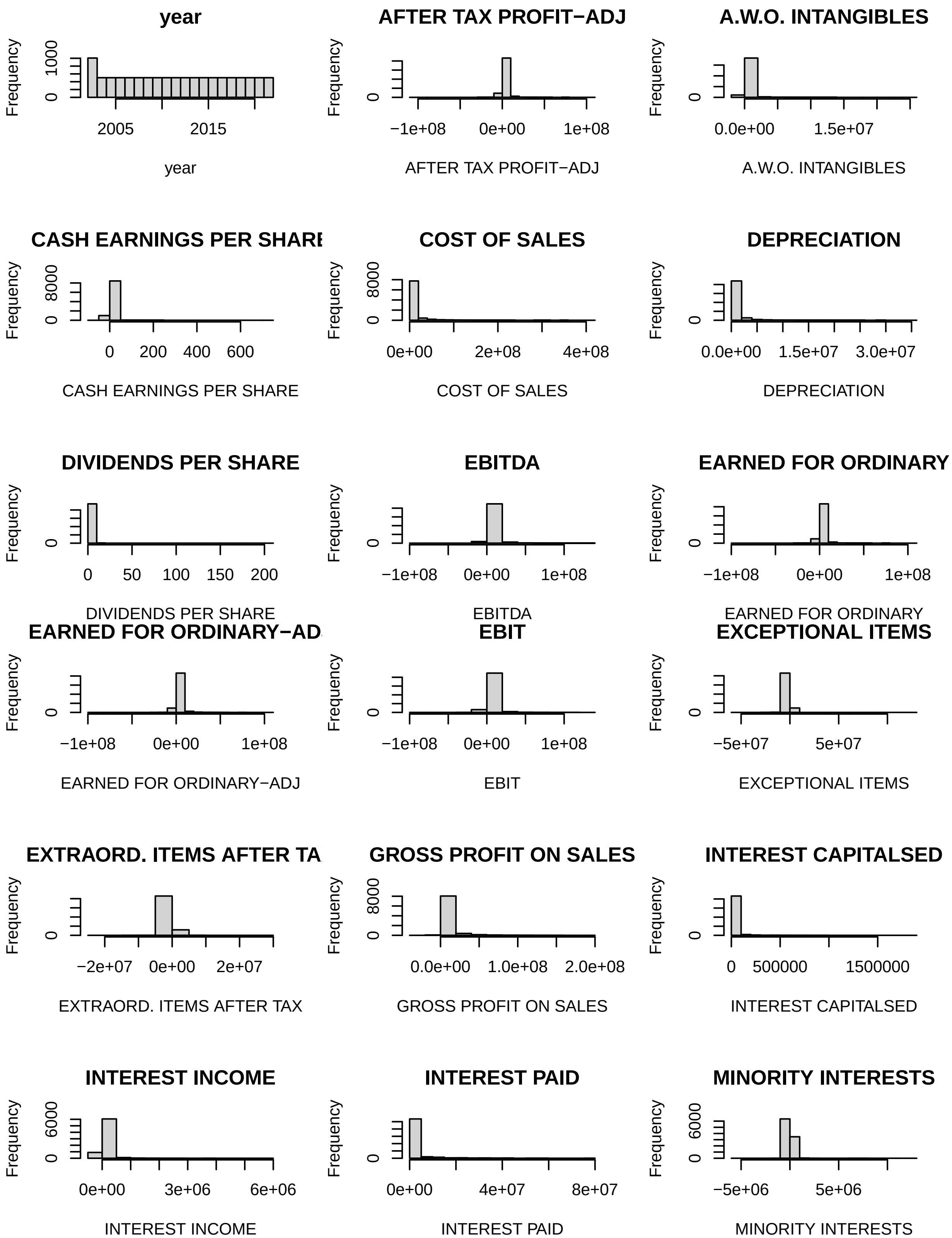
```
show_data_description(df_profit_loss)
```

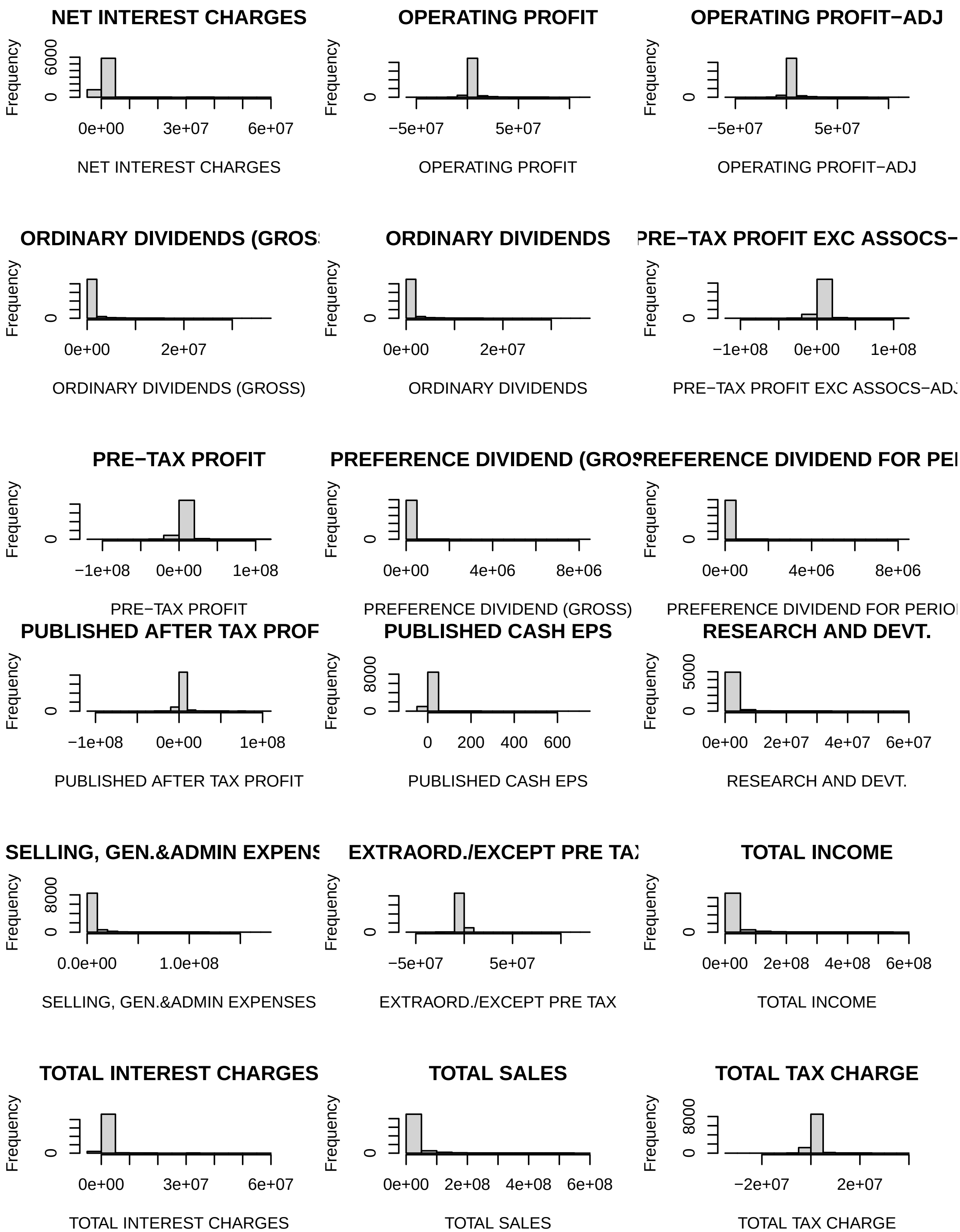
```
## The data frame contains 10563 rows with 42 variables for 503  distinct companies from 2002 to 2022 .
##                                    type n_distinct        min       max
## company                       character        503         3M  ZOETIS A
## year                            numeric         21       2002      2022
## AFTER TAX PROFIT-ADJ            numeric       8647 -100387000 103279000
## A.W.O. INTANGIBLES              numeric       4356     -13000  25516990
## CASH EARNINGS PER SHARE         numeric        115        -60       724
## COST OF SALES                   numeric       8253          0 418299900
## DEPRECIATION                    numeric       7598          0  34296000
## DIVIDENDS PER SHARE             numeric         26          0       210
## EBITDA                          numeric       8912  -88230990 133138000
## EARNED FOR ORDINARY             numeric       8695  -99688990 104690000
## EARNED FOR ORDINARY-ADJ         numeric       8695  -99688990 104690000
## EBIT                            numeric       8821  -91754000 122034000
## EXCEPTIONAL ITEMS               numeric       4963  -55648000 122860000
## EXTRAORD. ITEMS AFTER TAX       numeric       1654  -22600000  28200000
## GROSS PROFIT ON SALES           numeric       8304  -24842990 196966000
## INTEREST CAPITALSED             numeric       1339          0   1841000
## INTEREST INCOME                 numeric       3188     -34560   5686000
## INTEREST PAID                   numeric       1160         19  77530990
## MINORITY INTERESTS              numeric       2328   -5821000  12050000
## NET INTEREST CHARGES            numeric       5085   -2878000  57302000
## OPERATING PROFIT                numeric       8888  -51288990 119437000
## OPERATING PROFIT-ADJ            numeric       8888  -51288990 119437000
## ORDINARY DIVIDENDS (GROSS)      numeric       5130          0  36968000
## ORDINARY DIVIDENDS              numeric       5130          0  36968000
## PRE-TAX PROFIT EXC ASSOCS-ADJ   numeric       8862 -108761000 119103000
## PRE-TAX PROFIT                  numeric       8862 -108761000 119103000
## PREFERENCE DIVIDEND (GROSS)     numeric        877          0   8480000
## PREFERENCE DIVIDEND FOR PERIOD  numeric        877          0   8480000
## PUBLISHED AFTER TAX PROFIT      numeric       8647 -100387000 103279000
## PUBLISHED CASH EPS              numeric        115        -60       724
## RESEARCH AND DEVT.              numeric       3571          0  56052000
## SELLING, GEN.&ADMIN EXPENSES    numeric       8405          0 172537000
## EXTRAORD./EXCEPT PRE TAX        numeric       4963  -55648000 122860000
## TOTAL INCOME                    numeric       9670          0 572753900
## TOTAL INTEREST CHARGES          numeric       6230    -130595  57302000
## TOTAL SALES                     numeric       9670          0 572753900
## TOTAL TAX CHARGE                numeric       7213  -34830990  36530000
## NET PREMS EARNED                numeric        506      30918 226233000
## INTEREST RECEIVED               numeric        712          0 124467000
## NET INTEREST INCOME             numeric        451      37386  57244990
## PROV. FOR BAD DEBTS             numeric        475   -9256000  48570000
## TOTAL EMPLOYMENT COSTS          numeric       1565    -330000  46706990
```

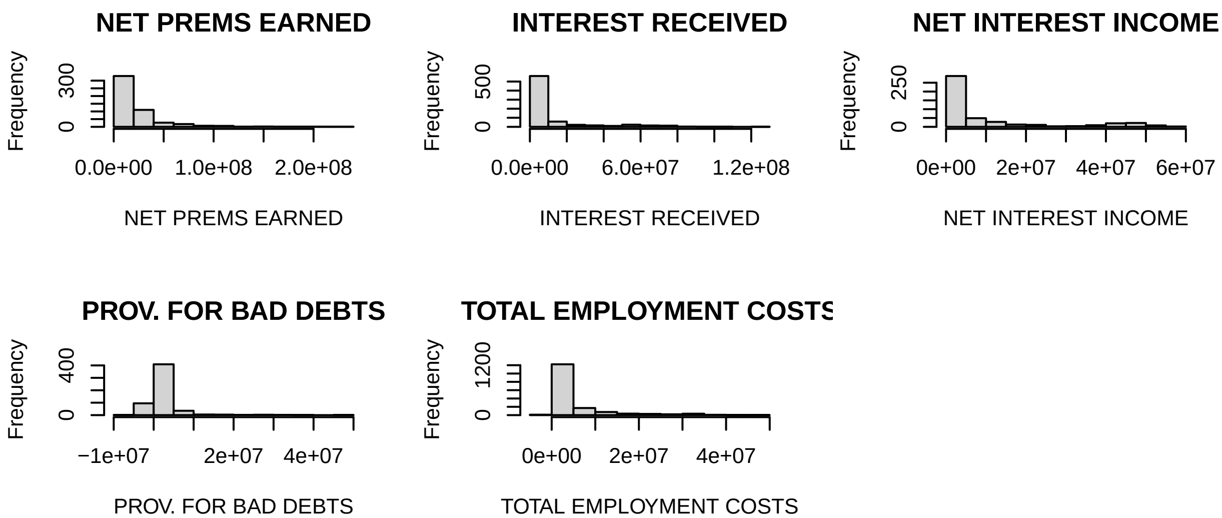
In the output above, we can see that the profit loss data frame contains 10563 records with 42 variables for 503 distinct companies. We can also see that similarly to the balance sheet data frame, all variables except for the company are numeric.

2.3.3 Distributions

```
show_data_distribution(df_profit_loss)
```




NET INTEREST CHARGES
Frequency
0
6000
0e+00
3e+07
6e+07
NET INTEREST CHARGES
OPERATING PROFIT
Frequency
0
−5e+07
5e+07
OPERATING PROFIT
OPERATING PROFIT−ADJ
Frequency
0
−5e+07
5e+07
OPERATING PROFIT−ADJ
ORDINARY DIVIDENDS (GROS
Frequency
0
0e+00
2e+07
ORDINARY DIVIDENDS (GROSS)
ORDINARY DIVIDENDS
Frequency
0
0e+00
2e+07
ORDINARY DIVIDENDS
PRE−TAX PROFIT EXC ASSOCS−
Frequency
0
−1e+08
0e+00
1e+08
PRE−TAX PROFIT EXC ASSOCS−AD
PRE−TAX PROFIT
Frequency
0
−1e+08
0e+00
1e+08
PRE−TAX PROFIT
PREFERENCE DIVIDEND (GRO
Frequency
0
0e+00
4e+06
8e+06
PREFERENCE DIVIDEND (GROSS)
REFERENCE DIVIDEND FOR PE
Frequency
0
0e+00
4e+06
8e+06
PREFERENCE DIVIDEND FOR PERIO
PUBLISHED AFTER TAX PROF
Frequency
0
−1e+08
0e+00
1e+08
PUBLISHED AFTER TAX PROFIT
PUBLISHED CASH EPS
Frequency
0
8000
0
200
400
600
PUBLISHED CASH EPS
RESEARCH AND DEVT.
Frequency
0
5000
0e+00
2e+07
4e+07
6e+07
RESEARCH AND DEVT.
SELLING, GEN.&ADMIN EXPENS
Frequency
0
8000
0.0e+00
1.0e+08
SELLING, GEN.&ADMIN EXPENSES
EXTRAORD./EXCEPT PRE TA
Frequency
0
−5e+07
5e+07
EXTRAORD./EXCEPT PRE TAX
TOTAL INCOME
Frequency
0
0e+00
2e+08
4e+08
6e+08
TOTAL INCOME
TOTAL INTEREST CHARGES
Frequency
0
0e+00
3e+07
6e+07
TOTAL INTEREST CHARGES
TOTAL SALES
Frequency
0
0e+00
2e+08
4e+08
6e+08
TOTAL SALES
TOTAL TAX CHARGE
Frequency
0
8000
−2e+07
2e+07
TOTAL TAX CHARGE



Also for the profit and loss data frame, we can see many right-skewed variables. To make those distributions more symmetrical, we have to perform a log transformation in the data preparation stage.

2.3.4 Investigate number of observations and continuity per company

As we previously did it for the other two tables, we will now investigate the number of observations for the profit and loss data frame. This data frame is also present in a yearly frequency, which leads to an optimal number of observations of 21 for each company.

```
# calculate observations per company
obs_per_company <-
  df_profit_loss %>%
  group_by(df_profit_loss$company) %>%
  summarise(n_observations = n()) %>%
  arrange(n_observations)
obs_per_company
```

```
## # A tibble: 503 x 2
##    `df_profit_loss$company` n_observations
##    <chr>                             <int>
##  1 3M                                   21
##  2 ABBOTT LABORATORIES                  21
##  3 ABBVIE                               21
##  4 ABIOMED                              21
##  5 ACCENTURE CLASS A                    21
##  6 ACTIVISION BLIZZARD                  21
##  7 ADOBE (NAS)                          21
##  8 ADV.AUTO PARTS                       21
##  9 ADVANCED MICRO DEVICES               21
## 10 AES                                  21
## # i 493 more rows
```

```
cat("Minimum number of observations", paste(min(obs_per_company$n_observations)), "\n")
```

```
## Minimum number of observations 21
```

```
cat("Maximum number of observations", paste(max(obs_per_company$n_observations)), "\n")
```

```
## Maximum number of observations 21
```

Similar to the balance sheet data frame, this data frame contains an optimal number of observations for each company. We will quickly perform a duplicate test to ensure that there are no gaps in the data.

```
cat("Found",
    paste(sum(duplicated(df_profit_loss))),
    "duplicated records in the profit loss data frame.")
```

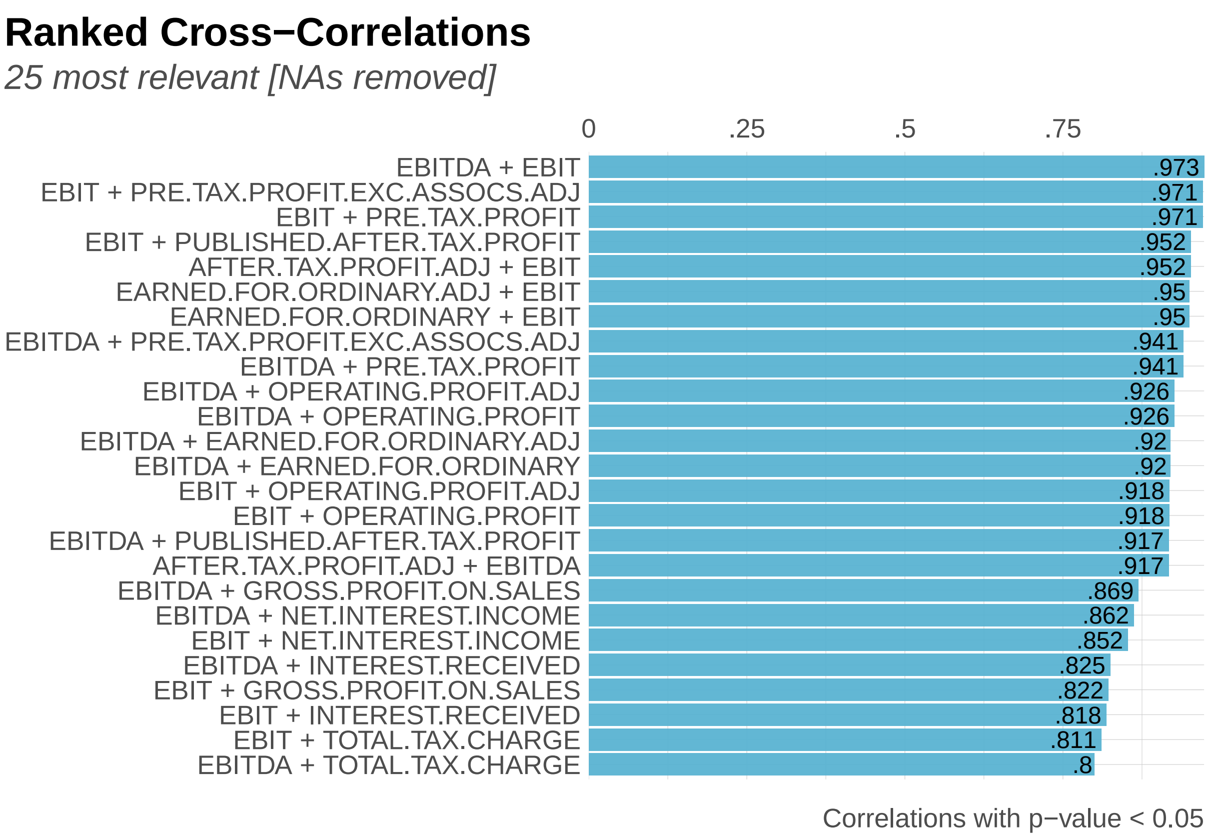
```
## Found 0 duplicated records in the profit loss data frame.
```

2.3.5 Correlation analysis

Similarly to the balance sheet table, we will now perform a pair-wise correlation analysis for the profit and loss data frame.

```
dfpl_num <- df_profit_loss[, sapply(df_profit_loss, is.numeric)] # obtain only numeric columns

corr_cross(dfpl_num, # name of dataset
  max_pvalue = 0.05, # display only significant correlations (at 5% level)
  rm.na = TRUE, # remove NAs
  top = 25 # show only top 25 variable pairs
)
```

```
## Returning only the top 25. You may override with the 'top' argument
```



In the plot above, we can now see a very different picture as before. All of the shown variable pairs are highly correlated for the profit and loss data frame. This makes sense, as the values of a profit and loss statement are all highly dependent on each other. EBIT and EBITDA for example only have the difference that the EBIT includes the earnings before interests and taxes, whereas the EBITDA includes the earnings before interests, taxes, depreciation and amortization. The EBIT therefore equals the EBITDA minus the depreciation and amortization value. As such highly correlated variables can lead to instability when modeling the data, we will perform two decorrelation strategies in the data preparation stage: (1) We will employ principal component analysis (PCA) to reduce the dimensionality of the profit and loss variables and thereby remove the correlation and (2) only keep the variables for which we have the most data and drop all other variables that are highly correlated to this variable.

2.3.6 Data quality assesment

```r
assess_data_quality(df_profit_loss)
```

```
##                                absolute_missing_values relative_missing_values
## NET INTEREST INCOME                              10106              0.95673578
## NET PREMS EARNED                                 10058              0.95219161
## PROV. FOR BAD DEBTS                              10005              0.94717410
## INTEREST RECEIVED                                 9838              0.93136420
## INTEREST PAID                                     9374              0.88743728
## TOTAL EMPLOYMENT COSTS                            8956              0.84786519
## RESEARCH AND DEVT.                                5373              0.50866231
## NET INTEREST CHARGES                              3545              0.33560542
## INTEREST INCOME                                   3463              0.32784247
## A.W.O. INTANGIBLES                                2667              0.25248509
## COST OF SALES                                     1835              0.17371959
## GROSS PROFIT ON SALES                             1825              0.17277289
## INTEREST CAPITALSED                               1594              0.15090410
## SELLING, GEN.&ADMIN EXPENSES                      1374              0.13007668
## DIVIDENDS PER SHARE                               1071              0.10139165
## CASH EARNINGS PER SHARE                           1070              0.10129698
## PUBLISHED CASH EPS                                1070              0.10129698
## EXCEPTIONAL ITEMS                                  967              0.09154596
## EXTRAORD./EXCEPT PRE TAX                           967              0.09154596
## EBITDA                                             868              0.08217362
## ORDINARY DIVIDENDS (GROSS)                         776              0.07346398
## ORDINARY DIVIDENDS                                 776              0.07346398
## EXTRAORD. ITEMS AFTER TAX                          762              0.07213860
## DEPRECIATION                                       761              0.07204393
## EBIT                                               749              0.07090789
## TOTAL INTEREST CHARGES                             749              0.07090789
## MINORITY INTERESTS                                 694              0.06570103
## AFTER TAX PROFIT-ADJ                               644              0.06096753
## PUBLISHED AFTER TAX PROFIT                         644              0.06096753
## TOTAL TAX CHARGE                                   644              0.06096753
## PREFERENCE DIVIDEND (GROSS)                        636              0.06021017
## PREFERENCE DIVIDEND FOR PERIOD                     636              0.06021017
## PRE-TAX PROFIT EXC ASSOCS-ADJ                      624              0.05907413
## PRE-TAX PROFIT                                     624              0.05907413
## OPERATING PROFIT                                   623              0.05897946
## OPERATING PROFIT-ADJ                               623              0.05897946
## TOTAL INCOME                                       614              0.05812743
## TOTAL SALES                                        614              0.05812743
## EARNED FOR ORDINARY                                612              0.05793809
## EARNED FOR ORDINARY-ADJ                            612              0.05793809
## company                                              0              0.00000000
## year                                                 0              0.00000000
```

Similar to the balance sheet data frame, the profit and loss data frame contains missing values for all variables except year and company. We will also exclude variables with more than 20% missing values for this data frame and impute others in the data preparation stage.

2.3.7 Tasks for data preparation

After the analysis of the profit and loss data frame, we carry on the following tasks to the data preparation stage: 1. Remove variables where > 20% of values are missing 2. Log transform right-skewed variables 3. Impute missing values for other variables 4. Perform decorrelation via PCA and variable selection

2.4. General remarks

1. By just comparing the outputs of the code chunks 1.1, 2.1 and 3.1, we can see that there are different company naming schemes (e.g., APPLE and APPLE INC). We will have to take care of that when joining all of the tables together.
2. The variable names for the balance sheet and profit loss data frame are capitalized and contain white spaces and special characters, which we should transform in the data preparation stage.

3. Data Preparation

The second stage of CRISP-DM within this project is data preparation. Within this section, we will transform the data according to the requirements we gained in the data understanding stage. For the sales data, the goals are to apply a log transformation to symmetrize the distribution of our target variable, to remove duplicated data points, to exclude companies with less than 40 observations, to interpolate the time series' of companies that have gaps or missing data and to finally split the target variable into one variable for each quarter, which will make following steps easier.
For the balance sheet and for the profit and loss data, the goals are to case fold and replace variable names to achieve a unified notation of the variables, to remove variables where more than 20% of the data is missing, to log transform right-skewed variables and to impute missing values of variables by applying spline interpolation. After that, we will join the three data sets into one combined table. As the naming schemes for the company names are different for the sales and other two tables, we will perform a fuzzy join method, that employs a string distance measure on the join attribute. After we joined the tables together, we will again perform a correlation analysis. Since strong correlations between variables can lead to instabilities when modeling the data, we will reduce the correlation with two approaches: (1) we will only select variables that are not highly correlated, and (2) we perform a principal component analysis to reduce the dimensionality of the data. The companies that are remaining after the aforementioned preparation steps will be enriched by their respective industries/business sectors from WikiData. This step will be performed in OpenRefine outside this notebook. Finally a train test split will be carried out. We will use the last five years of each company time series as test fold and the years before the testing period as the training fold.

Imports

```r
if(!require(corrplot)) {
  install.packages("corrplot")
}
library(corrplot)

if (!require(tidyverse)) {
  install.packages("tidyverse")
}
library(tidyverse)

if (!require(zoo)) {
  install.packages("zoo")
}
library(zoo)

if (!require(lares)) {
  # install lares correlation package from github
  devtools::install_github("laresbernardo/lares")
}
library(lares)

if (!require(imputeTS)) {
  install.packages("imputeTS")
}
```

```
## Loading required package: imputeTS
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## Registered S3 method overwritten by 'forecast':
##   method           from
##   predict.default statip
```

```
##
## Attaching package: 'imputeTS'
```

```
## The following object is masked from 'package:zoo':
##
##     na.locf
```

```
library(imputeTS)

if (!require(lubridate)) {
  install.packages("lubridate")
}
library(lubridate)

if (!require(caret)) {
  install.packages("caret")
}
```

```
## Loading required package: caret
```

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

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

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

```
library(caret)

if (!require(fuzzyjoin)) {
  install.packages("fuzzyjoin")
}
```

```
## Loading required package: fuzzyjoin
```

```
library(fuzzyjoin)

if (!require(dplyr)) {
  install.packages("dplyr")
}
library(dplyr)
```

Constants

```
Sys.unsetenv("LARES_FONT")
BASE_PATH <- "../data/processed"
SALES_PATH <- paste(BASE_PATH, "sales.csv", sep = "/")
BALANCE_SHEET_PATH <- paste(BASE_PATH, "balance_sheet.csv", sep = "/")
PROFIT_LOSS_PATH <- paste(BASE_PATH, "profit_loss.csv", sep = "/")
COMPANY_OUTPUT_PATH <- paste(BASE_PATH, "companies.csv", sep = "/")
DATA_JOINED_OUTPUT_PATH <- paste(BASE_PATH, "data_joined.csv", sep = "/")
TRAIN_VAR_SEL_OUTPUT_PATH <- paste(BASE_PATH, "train_var_sel.csv", sep = "/")
```

```
TEST_VAR_SEL_OUTPUT_PATH <- paste(BASE_PATH, "test_var_sel.csv", sep = "/")
TRAIN_PCA_OUTPUT_PATH <- paste(BASE_PATH, "train_pca.csv", sep = "/")
TEST_PCA_OUTPUT_PATH <- paste(BASE_PATH, "test_pca.csv", sep = "/")
```

3.1. Sales

3.1.1 Load data

```
df_sales <- read_csv(SALES_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

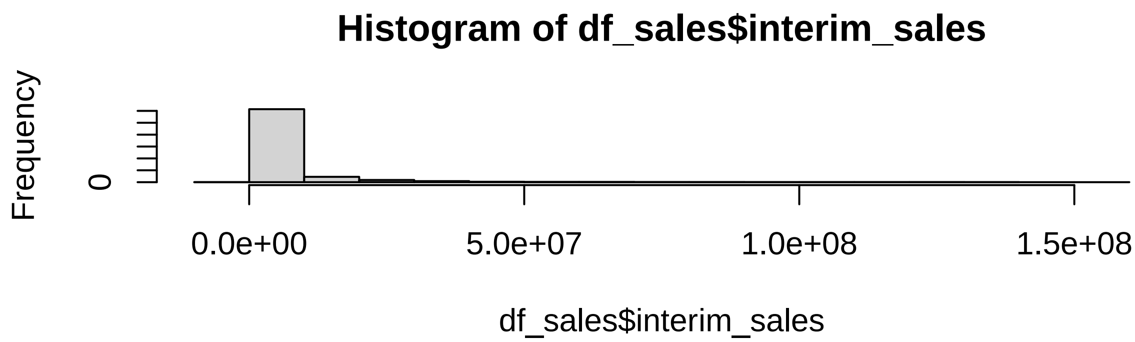
```
df_sales <- df_sales[, -1] # remove index column
head(df_sales)
```

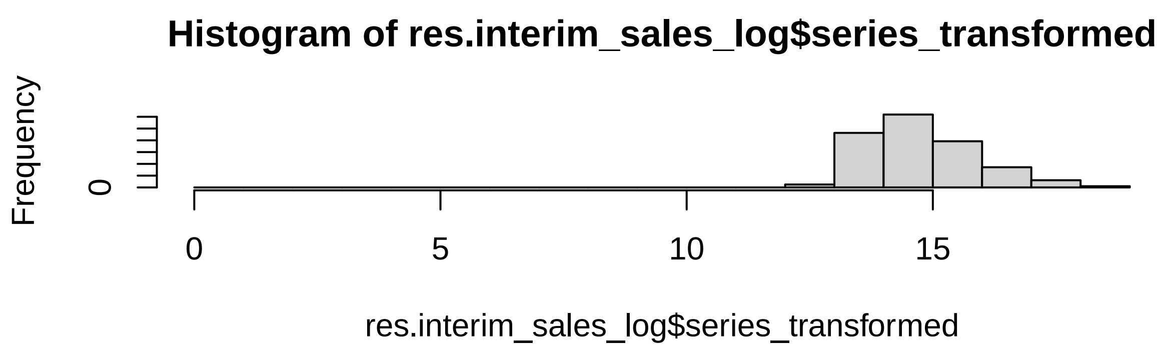
```
## # A tibble: 6 x 4
##   company   interim_sales  year quarter
##   <chr>             <dbl> <dbl>   <dbl>
## 1 APPLE INC       1475000  2003       1
## 2 APPLE INC       1909000  2004       1
## 3 APPLE INC       3243000  2005       1
## 4 APPLE INC       4359000  2006       1
## 5 APPLE INC       5264000  2007       1
## 6 APPLE INC       7512000  2008       1
```

3.1.2 Log transform interim_sales variable

To make the target variable interim_sales more symmetrical, we will use a log transformation. As the variable also contains negative values which cannot be log-transformed, we will add a constant to the variable to make all values positive. After modeling, we then just have to apply an exponential function and subtract the constant again to invert the transformation.

```
par(mfrow = c(2, 1))
hist(df_sales$interim_sales)

log_transform_with_constant <- function(series) {
  series_min <- min(na.omit(series)) # na.omit to exclude Nan from minimum

  if (series_min <= 0) {
    constant <- 1 + series_min * -1
  } else {
    constant <- 0
  }
  series_transformed <- log(series + constant)
  return(list("series_transformed" = series_transformed, "constant" = constant))
}

res.interim_sales_log <-
  log_transform_with_constant(df_sales$interim_sales)
INTERIM_SALES_LOG_CONSTANT <- res.interim_sales_log$constant
hist(res.interim_sales_log$series_transformed)
```






```
df_sales$log.interim_sales <- res.interim_sales_log$series_transformed
cat(paste("Used constant for log transformation:", res.interim_sales_log$constant))
```

```
## Used constant for log transformation: 393001
```

In the histogram of the log transformed interim sales variable, we can see that there are outlier points, i.e. points with a log transformed value of < 12. Such values will be challenging for our forecasting algorithms, let's now investigate for which data points this happens.

```
df_sales[df_sales$log.interim_sales < 12,]
```

```
## # A tibble: 1 x 5
##   company              interim_sales  year quarter log.interim_sales
##   <chr>                        <dbl> <dbl>   <dbl>             <dbl>
## 1 HARTFORD FINANCIAL         -393000  2008       3                 0
```

We can see, that there is only one data point, that is an outlier point. We can see that this observation occurred in the third quarter in 2008 and we can expect that this an outcome of the financial crisis. We will exclude this point and later on create a new value in the interpolation step.

```
df_sales <- df_sales[!df_sales$log.interim_sales < 12,]
hist(df_sales$log.interim_sales)
```

As we can see now, the distribution looks much cleaner with this point removed. In the next step, we will remove duplicated data points from the dataset.

3.1.3 Remove duplicated data points

```
# perform duplicate elimation
cat(paste("Number of data points before deduplication step:", nrow(df_sales)), "\n")
```

```
## Number of data points before deduplication step: 34840
```

```
df_sales_no_dup <- distinct(df_sales, .keep_all = TRUE)
cat(paste("Number of data points after deduplication step:", nrow(df_sales_no_dup)), "\n")
```

```
## Number of data points after deduplication step: 34694
```

```
cat(paste(nrow(df_sales) - nrow(df_sales_no_dup)), " data points were removed.", "\n")
```

```
## 146  data points were removed.
```

We can observe that 146 duplicated data points were removed from the data set. We will now exclude companies with less than 40 observations (5 years training, 5 years testing for 4 quarters per year).

3.1.4 Exclude companies with less than 40 observations

```
companies_before <- n_distinct(df_sales_no_dup$company)

# calculate observations per company
obs_per_company <-
  df_sales_no_dup %>%
  group_by(df_sales_no_dup$company) %>%
  summarise(n_observations = n()) %>%
```

```r
  arrange(n_observations)

# rename column
colnames(obs_per_company)[1] <- "company"

n_min <- 40 # at least 5 years for training and 5 years for evaluation * 4 = 40

companies_to_exclude <- obs_per_company[obs_per_company$n_observations < n_min, ]$company

df_sales_no_dup <- df_sales_no_dup[!(df_sales_no_dup$company %in% companies_to_exclude),]

companies_after <- n_distinct(df_sales_no_dup$company)
cat(paste(companies_before - companies_after,
          "companies were removed because of having less than", n_min, "observations\n"))
```

```
## 37 companies were removed because of having less than 40 observations
```

3.1.5 Interpolate companies with non-continous time series'

```r
# create date from year and quarter
df_sales_no_dup$date <-
  as.Date(as.yearqtr(paste0(df_sales_no_dup$year, "-",
                            df_sales_no_dup$quarter), format = "%Y-%q"))


check_continuity <- function(df, obs_per_company){
    # create new column that marks if ts is continuous,
    # set to true for all companies in the beginning
    obs_per_company$is_continous <- TRUE

    for (company in unique(df$company)) {
    # for each company
    company_dates <- df[df$company == company,]$date
    company_dates <- sort(company_dates) # select sorted dates
    for (i in 1:(length(company_dates) - 1)) {
      # for each date index
      difference_in_days <-
        as.integer(company_dates[i + 1] - company_dates[i])
      # in case there are two months with 31 days in the quarter,
      # the maximum valid difference in days is 92
      if (difference_in_days > 92) {
        obs_per_company[obs_per_company$company == company, "is_continous"] <-
          FALSE
        break
      }
    }
  }

  cat("Found",
      paste(nrow(obs_per_company[obs_per_company$is_continous == FALSE, ])),
      "companies that do not have a continous sales time series.")
  return(obs_per_company)
}
```

```r
obs_per_company <- check_continuity(df_sales_no_dup, obs_per_company)
```

```
## Found 64 companies that do not have a continous sales time series.
```

```r
# complete time series (i.e., create equidistant points) and interpolate missing values
df_sales_no_dup_intepol <-
  df_sales_no_dup %>%
  mutate(date =  as.Date(as.yearqtr(paste0(df_sales_no_dup$year, "-",
                                          df_sales_no_dup$quarter),
                                   format = "%Y-%q"))) %>%
  group_by(company) %>%
  complete(date = seq.Date(min(date), max(date), by="quarter")) %>%
  mutate(log.interim_sales = na_interpolation(log.interim_sales,
                                             option = "spline")) %>%
  ungroup()

# check continuity again
obs_per_company <- check_continuity(df_sales_no_dup_intepol, obs_per_company)
```

```
## Found 0 companies that do not have a continous sales time series.
```

```r
# the completion of the time series created missing values for some years and quarters,
# we will now employ the created date column to parse years and quarters.
df_sales_no_dup_intepol$quarter <- quarter(df_sales_no_dup_intepol$date)
df_sales_no_dup_intepol$year <- year(df_sales_no_dup_intepol$date)

# remove original interim sales
df_sales_no_dup_intepol <- subset(df_sales_no_dup_intepol,
                                  select=-c(interim_sales))

# check if previous steps were successful
colSums(is.na.data.frame(df_sales_no_dup_intepol))
```

```
##           company              date              year           quarter
##                 0                 0                 0                 0
## log.interim_sales
##                 0
```

```r
head(df_sales_no_dup_intepol)
```

```
## # A tibble: 6 x 5
##   company    date        year quarter log.interim_sales
##   <chr>      <date>     <dbl>   <int>             <dbl>
## 1 3M COMPANY 2002-10-01  2002       4              15.3
## 2 3M COMPANY 2003-01-01  2003       1              15.4
## 3 3M COMPANY 2003-04-01  2003       2              15.4
## 4 3M COMPANY 2003-07-01  2003       3              15.4
## 5 3M COMPANY 2003-10-01  2003       4              15.4
## 6 3M COMPANY 2004-01-01  2004       1              15.5
```

3.1.6 Create a seperate column for the sales variable in each quarter

```r
# we have to create a separate column for each quarter to later join on the year variable
df_sales_final <- df_sales_no_dup_intepol %>%
  select(-date) %>%
  pivot_wider(names_from = quarter,
```

```r
              names_prefix = "log.interim_sales_Q",
              values_from = log.interim_sales) %>%
  select( # reorder columns
    company,
    year,
    log.interim_sales_Q1,
    log.interim_sales_Q2,
    log.interim_sales_Q3,
    log.interim_sales_Q4
  )

df_sales_final
```

```
## # A tibble: 8,959 x 6
##    company   year log.interim_sales_Q1 log.interim_sales_Q2 log.interim_sales_Q3
##    <chr>    <dbl>                <dbl>                <dbl>                <dbl>
##  1 3M COMP~  2002                 NA                   NA                   NA
##  2 3M COMP~  2003                 15.4                 15.4                 15.4
##  3 3M COMP~  2004                 15.5                 15.5                 15.5
##  4 3M COMP~  2005                 15.5                 15.6                 15.6
##  5 3M COMP~  2006                 15.6                 15.6                 15.6
##  6 3M COMP~  2007                 15.7                 15.7                 15.7
##  7 3M COMP~  2008                 15.7                 15.8                 15.8
##  8 3M COMP~  2009                 15.5                 15.6                 15.7
##  9 3M COMP~  2010                 15.7                 15.8                 15.8
## 10 3M COMP~  2011                 15.9                 15.9                 15.9
## # i 8,949 more rows
## # i 1 more variable: log.interim_sales_Q4 <dbl>
```

3.2. Balance sheet

3.2.1 Load data

```r
df_balance_sheet <- read_csv(BALANCE_SHEET_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```r
df_balance_sheet <- df_balance_sheet[, -1]
head(df_balance_sheet)
```

```
## # A tibble: 6 x 30
##   company  year `BORROWINGS REPAYABLE < 1 YEAR` `EQUITY CAP. AND RESERVES`
##   <chr>   <dbl>                           <dbl>                      <dbl>
## 1 APPLE    2002                               0                    4095000
## 2 APPLE    2003                          304000                    4223000
## 3 APPLE    2004                               0                    5076000
## 4 APPLE    2005                               0                    7466000
## 5 APPLE    2006                               0                    9984000
## 6 APPLE    2007                               0                   14532000
## # i 26 more variables: `NET CURRENT ASSETS` <dbl>, `NET DEBT` <dbl>,
## #   `ORDINARY SHARE CAPITAL` <dbl>, `PREFERENCE CAPITAL` <dbl>,
## #   `TOTAL RESERVES` <dbl>, `ASSETS (TOTAL)` <dbl>,
## #   `TOTAL ASSETS EMPLOYED` <dbl>, `TOTAL CAPITAL EMPLOYED` <dbl>,
## #   `TOTAL CASH & EQUIVALENT` <dbl>, `TOTAL CURRENT ASSETS` <dbl>,
```

```
## #   `TOTAL CURRENT LIABLITIES` <dbl>, `TOTAL DEBT` <dbl>,
## #   `TOTAL DEBTORS & EQUIVALENT` <dbl>, ...
```

3.2.2 Casefold variable names and replace whitespaces

```r
transform_column_names <- function(colnames_df) {
  colnames_df_new <- c()
  for (i in 1:length(colnames_df)) {
    tmp <- casefold(colnames_df[i]) # lowercase
    tmp <- str_replace_all(tmp, " ", "_") # replace white spaces by underscore
    tmp <- str_replace_all(tmp, "\\.", "") # remove dot
    tmp <- str_replace_all(tmp, "\\,", "") # remove comma
    tmp <- str_replace_all(tmp, "\\(", "") # remove (
    tmp <- str_replace_all(tmp, "\\)", "") # remove )
    tmp <- str_replace_all(tmp, "\\/", "") # remove /
    tmp <- str_replace_all(tmp, "\\-", "") # remove -
    tmp <- str_replace_all(tmp, "\\&", "and") # replace & by and
    tmp <- str_replace_all(tmp, "\\<", "lt") # replace < by lt
    colnames_df_new[i] <- tmp
  }
  return(colnames_df_new)
}
colnames(df_balance_sheet) <- transform_column_names(colnames(df_balance_sheet))
colnames(df_balance_sheet)
```

```
##  [1] "company"                          "year"
##  [3] "borrowings_repayable_lt_1_year" "equity_cap_and_reserves"
##  [5] "net_current_assets"               "net_debt"
##  [7] "ordinary_share_capital"           "preference_capital"
##  [9] "total_reserves"                   "assets_total"
## [11] "total_assets_employed"            "total_capital_employed"
## [13] "total_cash_and_equivalent"        "total_current_assets"
## [15] "total_current_liablities"         "total_debt"
## [17] "total_debtors_and_equivalent"     "total_deferred_and_future_tax"
## [19] "tot_fixed_assetsnet"              "total_intangibles"
## [21] "total_investmnts_exassoc"         "total_loan_capital"
## [23] "tot_share_capital_and_reserves" "total_stock_and_wip"
## [25] "trade_creditors"                  "trade_debtors"
## [27] "total_insurance_funds"            "total_invests_insurance"
## [29] "current_deposit_and_other_acs"    "total_advances"
```

3.2.3 Remove variables where > 20% of values are missing

In the data understanding notebook, we saw that there are some variables where > 20% of the values are missing. We will remove those variables, as imputation will potentially introduce too much noise into the data.

```r
df_balance_sheet <- subset(df_balance_sheet, select=-c(
  total_insurance_funds,
  current_deposit_and_other_acs,
  total_advances,
  total_invests_insurance,
  total_debtors_and_equivalent,
  trade_creditors,
  total_current_assets,
```

```
  total_current_liablities,
  net_current_assets
  ))
```

3.2.4. Log transform right-skewed variables

For this table, we have 21 variables. Due to this fact, we wont be able to perform an in-depth investigation of each variable we want to log transform. Therefore, we will now only log transform highly right skewed variables that have positive values. We determine those variables by looking at the histograms generated in the data understanding notebook.

```
# define variables that are not negative, right-skewed and have not been removed yet
bs_variables_to_transform <- c(
  "borrowings_repayable_lt_1_year",
  "ordinary_share_capital",
  "preference_capital",
  "assets_total",
  "total_cash_and_equivalent",
  "total_debt",
  "tot_fixed_assetsnet",
  "total_intangibles",
  "total_loan_capital",
  "total_stock_and_wip",
  "trade_debtors"
)

# log transform with constant for each variable
for (variable in bs_variables_to_transform){
  df_balance_sheet[, variable] <- log_transform_with_constant(
    df_balance_sheet[, variable])$series_transformed
  names(df_balance_sheet)[names(df_balance_sheet) == variable] <-
    paste0("log.", variable)
}


head(df_balance_sheet)
```

```
## # A tibble: 6 x 21
##   company  year log.borrowings_repayable_lt_1_~1 equity_cap_and_reser~2 net_debt
##   <chr>   <dbl>                            <dbl>                  <dbl>    <dbl>
## 1 APPLE    2002                              0                  4095000  -4.02e6
## 2 APPLE    2003                             12.6                4223000  -4.26e6
## 3 APPLE    2004                              0                  5076000  -5.46e6
## 4 APPLE    2005                              0                  7466000  -8.26e6
## 5 APPLE    2006                              0                  9984000  -1.01e7
## 6 APPLE    2007                              0                 14532000  -1.54e7
## # i abbreviated names: 1: log.borrowings_repayable_lt_1_year,
## #   2: equity_cap_and_reserves
## # i 16 more variables: log.ordinary_share_capital <dbl>,
## #   log.preference_capital <dbl>, total_reserves <dbl>, log.assets_total <dbl>,
## #   total_assets_employed <dbl>, total_capital_employed <dbl>,
## #   log.total_cash_and_equivalent <dbl>, log.total_debt <dbl>,
## #   total_deferred_and_future_tax <dbl>, log.tot_fixed_assetsnet <dbl>, ...
```

3.2.5. Impute missing values for other variables with spline interpolation

While performing interpolation, we found out, that there are certain variables that are completely missing for certain companies. We have to options on how to deal with this issue: The first one is to remove those companies from the dataset, the second one is to remove those variables from the dataset. As we have many companies in the dataset, we will go with the first option.

```
# find companies where any variable has more than 50% missing values
bs_companies_remove <- df_balance_sheet %>%
  group_by(company) %>%
  summarise(across(log.borrowings_repayable_lt_1_year:log.trade_debtors,
                   ~ sum(is.na(.x)))) %>%
  group_by(company) %>%
  filter(if_any(-1, ~. > 10)) %>% # 10 is 50% of 20 years
  select(company)

# remove companies that have those variables
df_bs_count_before <- nrow(df_balance_sheet)
df_balance_sheet <-
  df_balance_sheet[!df_balance_sheet$company %in% bs_companies_remove$company,]
df_bs_count_after <- nrow(df_balance_sheet)

cat(paste("Removed", df_bs_count_before - df_bs_count_after,
          "data points from the balance sheet data frame."))
```

```
## Removed 2688 data points from the balance sheet data frame.
```

```
cat(paste("This equals", length(bs_companies_remove$company),
          "companies."))
```

```
## This equals 128 companies.
```

```
# perform spline interpolation for variables with missing values
df_balance_sheet_final <- df_balance_sheet %>%
  group_by(company) %>%
  mutate(across(
    where(is.numeric),
    ~ ifelse(is.na(.),
             na_interpolation(. , option = "spline"), # if true, interpolate
             .) # if false, do nothing
  )) %>%
  ungroup()

# check if previous steps were successful
colSums(is.na.data.frame(df_balance_sheet_final))
```

```
##                            company                             year
##                                  0                                0
## log.borrowings_repayable_lt_1_year          equity_cap_and_reserves
##                                  0                                0
##                           net_debt        log.ordinary_share_capital
##                                  0                                0
##             log.preference_capital                   total_reserves
##                                  0                                0
##                   log.assets_total             total_assets_employed
##                                  0                                0
##             total_capital_employed      log.total_cash_and_equivalent
```

```
##                                0                                 0
##                   log.total_debt      total_deferred_and_future_tax
##                                0                                 0
##            log.tot_fixed_assetsnet            log.total_intangibles
##                                0                                 0
##           total_investmnts_exassoc           log.total_loan_capital
##                                0                                 0
##     tot_share_capital_and_reserves            log.total_stock_and_wip
##                                0                                 0
##                 log.trade_debtors
##                                0
```

```r
head(df_balance_sheet_final)
```

```
## # A tibble: 6 x 21
##   company  year log.borrowings_repayable_lt_1_~1 equity_cap_and_reser~2 net_debt
##   <chr>   <dbl>                            <dbl>                  <dbl>    <dbl>
## 1 APPLE    2002                              0                  4095000  -4.02e6
## 2 APPLE    2003                             12.6                4223000  -4.26e6
## 3 APPLE    2004                              0                  5076000  -5.46e6
## 4 APPLE    2005                              0                  7466000  -8.26e6
## 5 APPLE    2006                              0                  9984000  -1.01e7
## 6 APPLE    2007                              0                 14532000  -1.54e7
## # i abbreviated names: 1: log.borrowings_repayable_lt_1_year,
## #   2: equity_cap_and_reserves
## # i 16 more variables: log.ordinary_share_capital <dbl>,
## #   log.preference_capital <dbl>, total_reserves <dbl>, log.assets_total <dbl>,
## #   total_assets_employed <dbl>, total_capital_employed <dbl>,
## #   log.total_cash_and_equivalent <dbl>, log.total_debt <dbl>,
## #   total_deferred_and_future_tax <dbl>, log.tot_fixed_assetsnet <dbl>, ...
```

3.3. Profit and loss

3.3.1 Load data

```r
df_profit_loss <- read_csv(PROFIT_LOSS_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```r
df_profit_loss <- df_profit_loss[, -1]
head(df_profit_loss)
```

```
## # A tibble: 6 x 42
##   company  year `AFTER TAX PROFIT-ADJ` `A.W.O. INTANGIBLES`
##   <chr>   <dbl>                  <dbl>                <dbl>
## 1 APPLE    2002                  65000                   NA
## 2 APPLE    2003                  68000                   NA
## 3 APPLE    2004                 276000                   NA
## 4 APPLE    2005                1335000                38000
## 5 APPLE    2006                1989000                45000
## 6 APPLE    2007                3496000                68000
## # i 38 more variables: `CASH EARNINGS PER SHARE` <dbl>, `COST OF SALES` <dbl>,
## #   DEPRECIATION <dbl>, `DIVIDENDS PER SHARE` <dbl>, EBITDA <dbl>,
## #   `EARNED FOR ORDINARY` <dbl>, `EARNED FOR ORDINARY-ADJ` <dbl>, EBIT <dbl>,
## #   `EXCEPTIONAL ITEMS` <dbl>, `EXTRAORD. ITEMS AFTER TAX` <dbl>,
```

```
## #   `GROSS PROFIT ON SALES` <dbl>, `INTEREST CAPITALSED` <dbl>,
## #   `INTEREST INCOME` <dbl>, `INTEREST PAID` <dbl>, `MINORITY INTERESTS` <dbl>,
## #   `NET INTEREST CHARGES` <dbl>, `OPERATING PROFIT` <dbl>, ...
```

3.3.2 Casefold variable names and replace whitespaces

```
colnames(df_profit_loss) <- transform_column_names(colnames(df_profit_loss))
colnames(df_profit_loss)
```

```
##  [1] "company"                          "year"
##  [3] "after_tax_profitadj"              "awo_intangibles"
##  [5] "cash_earnings_per_share"          "cost_of_sales"
##  [7] "depreciation"                     "dividends_per_share"
##  [9] "ebitda"                           "earned_for_ordinary"
## [11] "earned_for_ordinaryadj"           "ebit"
## [13] "exceptional_items"                "extraord_items_after_tax"
## [15] "gross_profit_on_sales"            "interest_capitalsed"
## [17] "interest_income"                  "interest_paid"
## [19] "minority_interests"               "net_interest_charges"
## [21] "operating_profit"                 "operating_profitadj"
## [23] "ordinary_dividends_gross"         "ordinary_dividends"
## [25] "pretax_profit_exc_assocsadj"      "pretax_profit"
## [27] "preference_dividend_gross"        "preference_dividend_for_period"
## [29] "published_after_tax_profit"       "published_cash_eps"
## [31] "research_and_devt"                "selling_genandadmin_expenses"
## [33] "extraordexcept_pre_tax"           "total_income"
## [35] "total_interest_charges"           "total_sales"
## [37] "total_tax_charge"                 "net_prems_earned"
## [39] "interest_received"                "net_interest_income"
## [41] "prov_for_bad_debts"               "total_employment_costs"
```

3.3.3 Remove variables where > 20% of values are missing

Also for this data frame, we saw that there are some variables where > 20% of the values are missing. We will remove those variables, as imputation will potentially introduce too much noise into the data.

```
df_profit_loss <- subset(df_profit_loss, select=-c(
  net_interest_income,
  net_prems_earned,
  prov_for_bad_debts,
  interest_received,
  interest_paid,
  total_employment_costs,
  research_and_devt,
  net_interest_charges,
  interest_income,
  awo_intangibles
  ))
```

3.3.4. Log transform right-skewed variables

For this table, we have 32 variables. Due to this fact, we wont be able to perform an in-depth investigation of each variable we want to log transform. Therefore, we will now only log transform highly right skewed variables that have positive values. We determine those variables by looking at the histograms generated in the data understanding notebook.

```r
# define variables that are not negative, right-skewed and have not been removed yet
pl_variables_to_transform <- c(
  "cost_of_sales",
  "depreciation",
  "dividends_per_share",
  "interest_capitalsed",
  "ordinary_dividends_gross",
  "ordinary_dividends",
  "preference_dividend_gross",
  "preference_dividend_for_period",
  "selling_genandadmin_expenses",
  "total_income",
  "total_sales"
)

# log transform with constant for each variable
for (variable in pl_variables_to_transform){
  df_profit_loss[, variable] <- log_transform_with_constant(
    df_profit_loss[, variable])$series_transformed
  names(df_profit_loss)[names(df_profit_loss) == variable] <-
    paste0("log.", variable)
}


head(df_profit_loss)
```

```
## # A tibble: 6 x 32
##   company  year after_tax_profitadj cash_earnings_per_share log.cost_of_sales
##   <chr>   <dbl>               <dbl>                   <dbl>             <dbl>
## 1 APPLE    2002               65000                       0              15.2
## 2 APPLE    2003               68000                       0              15.3
## 3 APPLE    2004              276000                       0              15.6
## 4 APPLE    2005             1335000                       0              16.1
## 5 APPLE    2006             1989000                       0              16.4
## 6 APPLE    2007             3496000                       0              16.6
## # i 27 more variables: log.depreciation <dbl>, log.dividends_per_share <dbl>,
## #   ebitda <dbl>, earned_for_ordinary <dbl>, earned_for_ordinaryadj <dbl>,
## #   ebit <dbl>, exceptional_items <dbl>, extraord_items_after_tax <dbl>,
## #   gross_profit_on_sales <dbl>, log.interest_capitalsed <dbl>,
## #   minority_interests <dbl>, operating_profit <dbl>,
## #   operating_profitadj <dbl>, log.ordinary_dividends_gross <dbl>,
## #   log.ordinary_dividends <dbl>, pretax_profit_exc_assocsadj <dbl>, ...
```

3.3.5. Impute missing values for other variables with spline interpolation

While performing interpolation, we found out, that there are certain variables that are completely missing for certain companies. As for the balance sheet data, we will remove those companies.

```r
# find companies where any variable has more than 50% missing values
pl_companies_remove <- df_profit_loss %>%
  group_by(company) %>%
  summarise(across(after_tax_profitadj:total_tax_charge,
                   ~ sum(is.na(.x)))) %>%
  group_by(company) %>%
```

```r
  filter(if_any(-1, ~. > 10)) %>% # 10 is 50% of 20 years
  select(company)

# remove companies that have those variables
df_pl_count_before <- nrow(df_profit_loss)
df_profit_loss <-
  df_profit_loss[!df_profit_loss$company %in% pl_companies_remove$company,]
df_pl_count_after <- nrow(df_profit_loss)

cat(paste("Removed", df_pl_count_before - df_pl_count_after,
          "data points from the profit and loss data frame."))
```

```
## Removed 3087 data points from the profit and loss data frame.
```

```r
cat(paste("This equals", length(pl_companies_remove$company),
          "companies."))
```

```
## This equals 147 companies.
```

```r
# perform spline interpolation for variables with missing values
df_profit_loss_final <- df_profit_loss %>%
  group_by(company) %>%
  mutate(across(
    where(is.numeric),
    ~ ifelse(is.na(.),
             na_interpolation(. , option = "spline"), # if true, interpolate
             .) # if false, do nothing
  )) %>%
  ungroup()

# check if previous steps were successful
colSums(is.na.data.frame(df_profit_loss_final))
```

```
##                         company                            year
##                               0                               0
##              after_tax_profitadj          cash_earnings_per_share
##                               0                               0
##                log.cost_of_sales               log.depreciation
##                               0                               0
##          log.dividends_per_share                          ebitda
##                               0                               0
##              earned_for_ordinary            earned_for_ordinaryadj
##                               0                               0
##                            ebit                exceptional_items
##                               0                               0
##         extraord_items_after_tax            gross_profit_on_sales
##                               0                               0
##          log.interest_capitalsed              minority_interests
##                               0                               0
##                operating_profit              operating_profitadj
##                               0                               0
##     log.ordinary_dividends_gross          log.ordinary_dividends
##                               0                               0
##       pretax_profit_exc_assocsadj                   pretax_profit
##                               0                               0
```

```
##      log.preference_dividend_gross log.preference_dividend_for_period
##                                   0                                  0
##         published_after_tax_profit                 published_cash_eps
##                                   0                                  0
##   log.selling_genandadmin_expenses               extraordexcept_pre_tax
##                                   0                                  0
##                   log.total_income             total_interest_charges
##                                   0                                  0
##                    log.total_sales                   total_tax_charge
##                                   0                                  0
```

```r
head(df_profit_loss_final)
```

```
## # A tibble: 6 x 32
##   company  year after_tax_profitadj cash_earnings_per_share log.cost_of_sales
##   <chr>   <dbl>               <dbl>                   <dbl>             <dbl>
## 1 APPLE    2002               65000                       0              15.2
## 2 APPLE    2003               68000                       0              15.3
## 3 APPLE    2004              276000                       0              15.6
## 4 APPLE    2005             1335000                       0              16.1
## 5 APPLE    2006             1989000                       0              16.4
## 6 APPLE    2007             3496000                       0              16.6
## # i 27 more variables: log.depreciation <dbl>, log.dividends_per_share <dbl>,
## #   ebitda <dbl>, earned_for_ordinary <dbl>, earned_for_ordinaryadj <dbl>,
## #   ebit <dbl>, exceptional_items <dbl>, extraord_items_after_tax <dbl>,
## #   gross_profit_on_sales <dbl>, log.interest_capitalsed <dbl>,
## #   minority_interests <dbl>, operating_profit <dbl>,
## #   operating_profitadj <dbl>, log.ordinary_dividends_gross <dbl>,
## #   log.ordinary_dividends <dbl>, pretax_profit_exc_assocsadj <dbl>, ...
```

3.4. Combine data frames

3.4.1 Fuzzy join company names of sales and balance sheet data frame

As mentioned in the data understanding notebook, the sales data frame and the other data frames have different naming schemes for the company names (e.g. 3M vs. 3M COMPANY). Therefore we will use a fuzzy join on the company names, that employs the Jaro-Winkler distance on the company names, such that they do not have to match completely.

```r
# fuzzy join company names of sales and balance sheet data frame
# needed because names are not similar for both tables
companies_joined <-
  stringdist_join(
    unique(df_sales_final[, "company"]),
    unique(df_balance_sheet_final[, "company"]),
    by = "company",
    mode = "inner",
    method = "jw",
    max_dist = 0.3, # maximum Jaro-Winkler distance
    ignore_case = TRUE,
    distance_col = "distance"
  )


# find companies that do have multiple matches
companies_multi_match <- companies_joined %>%
```

```r
  group_by(company.x) %>%
  summarise(n=n()) %>%
  filter(n > 1)

# with this fuzzy method, there can be multiple matches between companies
# we will only keep the matches with the smallest Jaro-Winkler distance
companies_joined_best <-
  companies_joined %>%
  group_by(company.x) %>%
  slice_min(distance)

head(companies_joined_best)
```

```
## # A tibble: 6 x 3
## # Groups:   company.x [6]
##   company.x            company.y            distance
##   <chr>                <chr>                   <dbl>
## 1 3M COMPANY           MERCK & COMPANY         0.204
## 2 A O SMITH CORP       MOSAIC                  0.246
## 3 ABBOTT LABORATORIES ABBOTT LABORATORIES    0
## 4 ABIOMED INC          ABIOMED                 0.121
## 5 ACCENTURE PLC        ACCENTURE CLASS A       0.151
## 6 ADOBE INC            ADOBE (NAS)             0.195
```

3.4.2 Manually resolve companies that could not be matched

By looking at the results above, we can see that the matching strategy worked for some cases, but that there are still some companies, where we have to perform the matching manually (e.g., 3M COMPANY). Furthermore, there are some companies, where we even can't perform the matching manually, because we removed them in previous steps. Thus, we have to remove those companies from the dataset.

```r
companies_joined_best[companies_joined_best$company.x == "3M COMPANY", "company.y"] <-
  "3M"
# we will use ALPHABET A and not 'C', as this is the regular stock with voting rights
companies_joined_best[companies_joined_best$company.x == "ALPHABET INC", "company.y"] <-
  "ALPHABET A"
companies_joined_best[companies_joined_best$company.x == "DEERE & COMPANY", "company.y"] <-
  "DEERE"
companies_joined_best[companies_joined_best$company.x == "NEWS CORP", "company.y"] <-
  "NEWS 'A'"
companies_joined_best[companies_joined_best$company.x == "UNITED AIR", "company.y"] <-
  "UNITED AIRLINES HOLDINGS"
companies_joined_best[companies_joined_best$company.x == "WEST PHARMACEUTICAL", "company.y"] <-
  "WEST PHARM.SVS."
companies_joined_best[companies_joined_best$company.x == "WESTINGHOUSE AIR", "company.y"] <-
  "WABTEC"
companies_joined_best[companies_joined_best$company.x == "NIKE INC.", "company.y"] <-
  "NIKE 'B'"

companies_not_resolvable <- c(
  "A O SMITH CORP",
  "AGILENT TECHNOLOGIES",
  "ALLEGION PLC",
  "ALLSTATE CORP",
```

```r
  "AMERICAN ELECTRIC",
  "AMERICAN INT'L GROUP",
  "AMERIPRISE FIN",
  "AMPHENOL CORP",
  "ARCH CAPITAL GROUP",
  "ASSURANT INC",
  "BANK OF NEW YORK",
  "CELANESE CORPORATION",
  "CENTENE CORP",
  "CHARLES RIVER LAB",
  "CHARLES SCHWAB CORP",
  "CHUBB",
  "CITIGROUP INC",
  "COMERICA INC.",
  "DISCOVER FINANCI",
  "ELEVANCE HEALTH",
  "EQUITY RESIDENTIAL",
  "EVEREST RE GROUP",
  "FRANKLIN RESOURCES",
  "GLOBAL PAYMENTS INC",
  "HOST HOTELS",
  "HUMANA INC.",
  "INTUIT INC",
  "IRON MOUNTAIN INC",
  "JOHNSON CONTROLS INT",
  "META PLATFORMS INC",
  "METLIFE INC",
  "MOLINA HEALTHCARE",
  "MORGAN STANLEY",
  "PRINCIPAL FINL GROUP",
  "PROGRESSIVE CORP",
  "PHILLIPS 66",
  "REGENCY CENTERS CORP",
  "REGIONS FINANCIAL",
  "SBA COMMUNICATIONS",
  "SEAGATE TECHNOLOGY",
  "SIGNATURE BANK",
  "SVB FINANCIAL GROUP",
  "TAKE",
  "TEXTRON INC",
  "TRAVELERS COS",
  "UNITEDHEALTH GROUP",
  "US BANCORP",
  "VERISK ANALYTICS",
  "VISA INC."
)

companies_joined_clean <-
  companies_joined_best[!companies_joined_best$company.x %in% companies_not_resolvable,] %>%
  ungroup() %>%
  rename(company.sales = company.x, company.rest = company.y) %>%
  select(-distance)
head(companies_joined_clean)
```

```
## # A tibble: 6 x 2
##   company.sales        company.rest
##   <chr>                <chr>
## 1 3M COMPANY           3M
## 2 ABBOTT LABORATORIES  ABBOTT LABORATORIES
## 3 ABIOMED INC          ABIOMED
## 4 ACCENTURE PLC        ACCENTURE CLASS A
## 5 ADOBE INC            ADOBE (NAS)
## 6 ADVANCE AUTO PARTS   ADV.AUTO PARTS
```

In the table above, we can now see that we were able to resolve 359 companies

3.4.3 Examine if balance sheet and profit and loss data frames share the same company naming scheme

```r
# check if balance sheet and profit and loss company names are similar
# for that we first of all load the profit and loss data again
# to ensure, that all company names are still there (and haven't been filtered yet)
df_pl_companies <-
  read_csv(PROFIT_LOSS_PATH, show_col_types = FALSE)
```

```
## New names:
## * `` -> `...1`
```

```r
df_pl_companies <-
  df_pl_companies %>% select(company) %>% distinct()

# if profit and loss data contains all company names in the joined table,
# the company names must be similar. This can be done by looking at the set difference
if (length(setdiff(
  unique(companies_joined_clean$company.rest),
  unique(df_pl_companies$company)
)) == 0) {
  # profit loss contains all company names in joined table
  cat(
    "Balance sheet data frame and profit and loss data frame share same company naming scheme."
  )
} else {
  cat("Balance sheet and profit and loss data frame have different company naming scheme.")
}
```

```
## Balance sheet data frame and profit and loss data frame share same company naming scheme.
```

3.4.5 Join sales, balance sheet and profit and loss to one data frame

```r
# join balance sheet and resolved company names
df_balance_sheet_with_2_company <-
  inner_join(companies_joined_clean,
             df_balance_sheet_final,
             by = c("company.rest" = "company"))

# join sales and balance sheet data frames together by company and year
df_sales_balance_sheet <-
  inner_join(
    df_balance_sheet_with_2_company,
```

```r
    df_sales_final,
    by = c("company.sales" = "company", "year")
  )

# join sales balance sheet data frame with profit and loss data frame
df_joined <- inner_join(
  df_sales_balance_sheet,
  df_profit_loss_final,
  by = c("company.rest" = "company", "year")
)

# add index to joined data frame
df_joined <- df_joined %>%
  mutate(index = row_number())

head(df_joined)
```

```
## # A tibble: 6 x 57
##   company.sales company.rest  year log.borrowings_repay~1 equity_cap_and_reser~2
##   <chr>         <chr>        <dbl>                  <dbl>                  <dbl>
## 1 3M COMPANY    3M            2002                   14.1                6086000
## 2 3M COMPANY    3M            2003                   14.0                5993000
## 3 3M COMPANY    3M            2004                   14.1                7885000
## 4 3M COMPANY    3M            2005                   14.6               10378000
## 5 3M COMPANY    3M            2006                   13.9               10100000
## 6 3M COMPANY    3M            2007                   14.7                9959000
## # i abbreviated names: 1: log.borrowings_repayable_lt_1_year,
## #   2: equity_cap_and_reserves
## # i 52 more variables: net_debt <dbl>, log.ordinary_share_capital <dbl>,
## #   log.preference_capital <dbl>, total_reserves <dbl>, log.assets_total <dbl>,
## #   total_assets_employed <dbl>, total_capital_employed <dbl>,
## #   log.total_cash_and_equivalent <dbl>, log.total_debt <dbl>,
## #   total_deferred_and_future_tax <dbl>, log.tot_fixed_assetsnet <dbl>, ...
```

3.4.6 Write joined data to file

```r
write.csv(df_joined, DATA_JOINED_OUTPUT_PATH)
```

3.5. Joint data analysis and preparation

3.5.1 Shift target (sales) variables backwards

As we want to predict the sales for the upcoming year by using features from the actual year (e.g., with features from 2002 we want to predict the sales for 2003), we have to shift our target variables one year backwards.

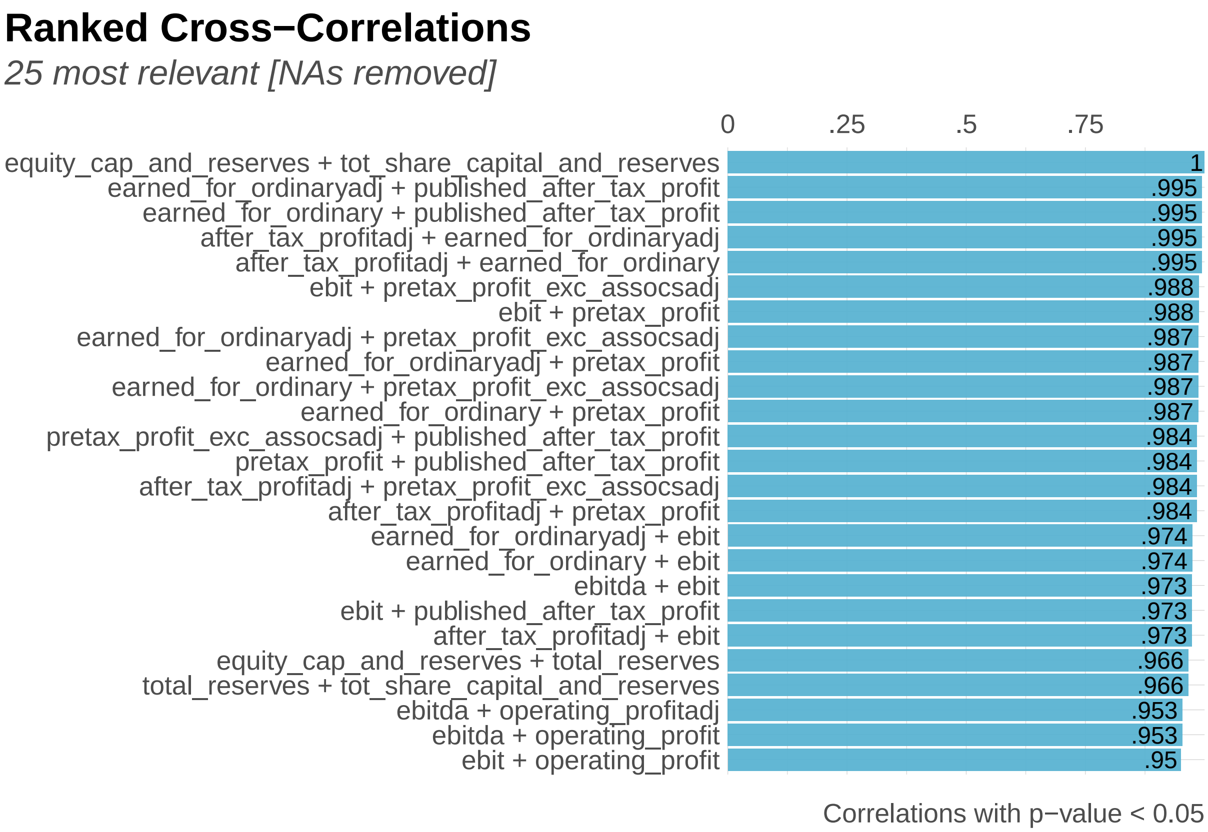
```r
df_joined_shift <- df_joined %>%
  group_by(company.sales) %>%
  arrange(year) %>%
  mutate(
    log.interim_sales_Q1 = lead(log.interim_sales_Q1),
    log.interim_sales_Q2 = lead(log.interim_sales_Q2),
    log.interim_sales_Q3 = lead(log.interim_sales_Q3),
    log.interim_sales_Q4 = lead(log.interim_sales_Q4),
```

```r
  ) %>%
  ungroup() %>%
  arrange(company.sales) %>%
  na.omit()
# the back shifting will create NA values for the last year in each series,
# we will omit those
```

3.5.2 Perform correlation analysis

```r
# Perform correlation analysis of joined data frame
corr_cross(
  df_joined_shift %>% select(
    # exclude target variables, year and company from analysis
    -c(
      index,
      company.sales,
      company.rest,
      year,
      log.interim_sales_Q1,
      log.interim_sales_Q2,
      log.interim_sales_Q3,
      log.interim_sales_Q4
    )
  ),
  # name of dataset
  max_pvalue = 0.05,
  # display only significant correlations (at 5% level)
  rm.na = TRUE,
  # remove NAs
  top = 25 # show only top 25 variable pairs
)
```

```
## Returning only the top 25. You may override with the 'top' argument
```



We can see that there are many highly correlated variable pairs. As this will introduce errors in our multivariate modeling approach, we have to remove those correlations. We will do this in 2 ways: 1. Performing variable selection by removing first variable in highly correlated pair. 2. Performing dimensionality reduction with principal component analysis (PCA).

3.5.3 Perform variable selection to remove highly correlated pairs

```
# select set of variables to perform variable selection
cor_vars_set <-
  df_joined_shift %>% select(
    # exclude target variables, year and company
    -c(
      index,
      company.sales,
      company.rest,
      year,
      log.interim_sales_Q1,
      log.interim_sales_Q2,
      log.interim_sales_Q3,
      log.interim_sales_Q4
  )
)

# calculate correlation matrix
cor_matrix <- cor(cor_vars_set)

# find indices of highly correlated variables (only 1 variable of highly correlated pair)
```

```r
# use an cutoff of 0.6 (remove variables with absolute correlation > 0.6)
vars_highly_correlated <- findCorrelation(cor_matrix, cutoff = 0.6, exact=TRUE)

# remove highly correlated variables and combine with company, year and target variables
df_joined_shift_var_sel <-
  bind_cols(
    df_joined_shift %>% select(c(
      index,
      company.sales,
      company.rest,
      year,
      log.interim_sales_Q1,
      log.interim_sales_Q2,
      log.interim_sales_Q3,
      log.interim_sales_Q4
  )),
    cor_vars_set %>%
      select(-all_of(vars_highly_correlated))
  )
cat(
  paste(
    "Removed",
    ncol(df_joined_shift) - ncol(df_joined_shift_var_sel),
    "variables from profit and loss data frame due to high correlation.\n",
    ncol(df_joined_shift_var_sel),
    "variables are remaining."
  )
)
```

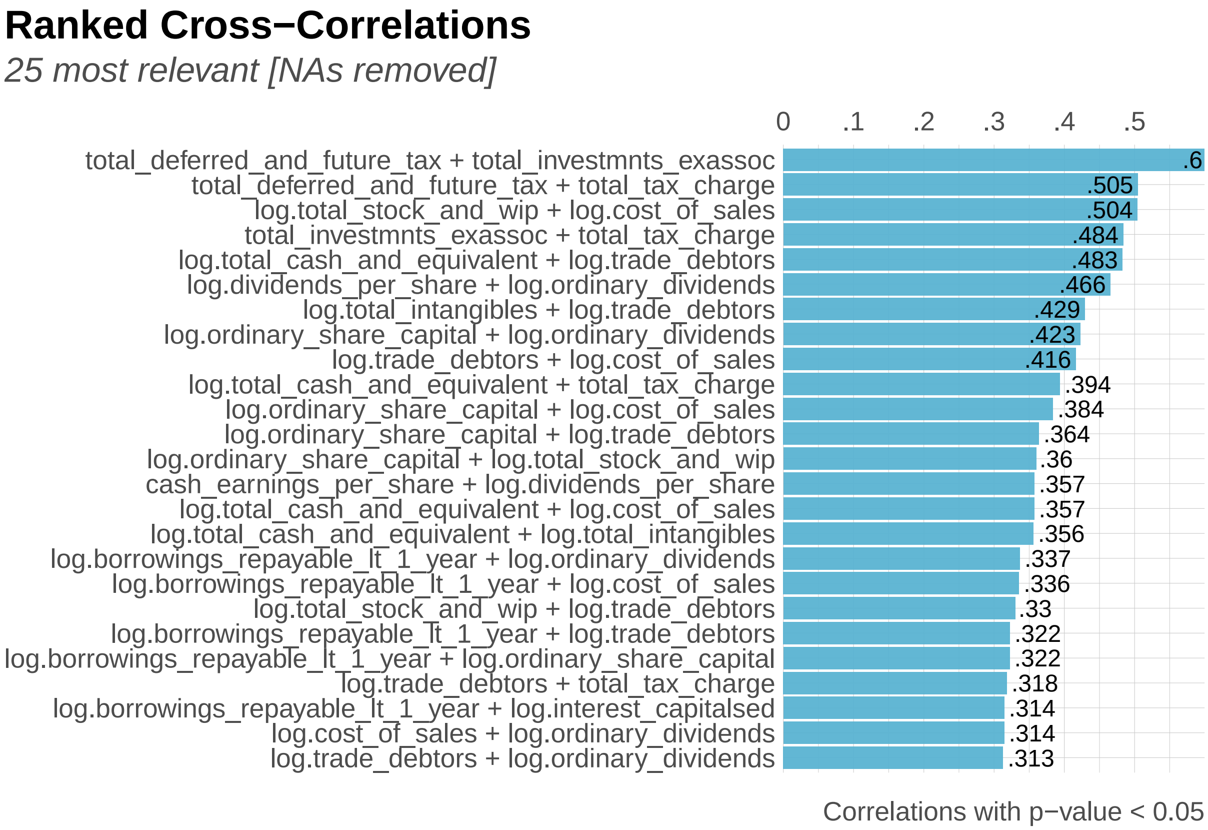
```
## Removed 30 variables from profit and loss data frame due to high correlation.
##  27 variables are remaining.
```

```r
df_joined_shift_var_sel
```

```
## # A tibble: 5,558 x 27
##    index company.sales company.rest  year log.interim_sales_Q1
##    <int> <chr>         <chr>        <dbl>                <dbl>
##  1     1 3M COMPANY    3M            2002                 15.4
##  2     2 3M COMPANY    3M            2003                 15.5
##  3     3 3M COMPANY    3M            2004                 15.5
##  4     4 3M COMPANY    3M            2005                 15.6
##  5     5 3M COMPANY    3M            2006                 15.7
##  6     6 3M COMPANY    3M            2007                 15.7
##  7     7 3M COMPANY    3M            2008                 15.5
##  8     8 3M COMPANY    3M            2009                 15.7
##  9     9 3M COMPANY    3M            2010                 15.9
## 10    10 3M COMPANY    3M            2011                 15.9
## # i 5,548 more rows
## # i 22 more variables: log.interim_sales_Q2 <dbl>, log.interim_sales_Q3 <dbl>,
## #   log.interim_sales_Q4 <dbl>, log.borrowings_repayable_lt_1_year <dbl>,
## #   net_debt <dbl>, log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>, ...
```

```
# check correlated pairs after variable removal
corr_cross(
  df_joined_shift_var_sel %>% select( # exclude target variables from analysis
    -c(
      index,
      company.sales,
      company.rest,
      year,
      log.interim_sales_Q1,
      log.interim_sales_Q2,
      log.interim_sales_Q3,
      log.interim_sales_Q4
    )
  ),
  # name of dataset
  max_pvalue = 0.05,
  # display only significant correlations (at 5% level)
  rm.na = TRUE,
  # remove NAs
  top = 25 # show only top 25 variable pairs
)
```

```
## Returning only the top 25. You may override with the 'top' argument
```



3.5.4 Perform dimensionality reduction with Principal Component Analysis (PCA)

Principal component analysis is a method of multivariate statistics that has the goal to down-project the data to less dimensions. This goal is achieved by a linear transformation of the data matrix X. The resulting principal components Z are obtained by multiplication with a so-called loadings matrix B. The coefficients of B are obtained by a eigen-decomposition of the covariance matrix of X and lead to maximum variance in each column vector (or principal component) in Z and orthogonality between each component. With this method, we can now only select the first k components of Z and therefore achieve dimensionality reduction. The resulting eigen-values of the decomposition contain the proportion of variance that is captured by each principal component.

```
# Perform PCA using prcomp()
pca_result <- prcomp(cor_vars_set, scale. = TRUE)

# Investigate captured variance per principal component
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4    PC5     PC6     PC7
## Standard deviation     4.3975 2.4046 1.83182 1.55556 1.4717 1.41095 1.25482
## Proportion of Variance 0.3947 0.1180 0.06848 0.04938 0.0442 0.04063 0.03213
## Cumulative Proportion  0.3947 0.5127 0.58114 0.63053 0.6747 0.71535 0.74749
##                            PC8     PC9   PC10    PC11    PC12    PC13    PC14
## Standard deviation     1.21918 1.12590 1.0193 0.97524 0.92516 0.89883 0.79947
## Proportion of Variance 0.03033 0.02587 0.0212 0.01941 0.01747 0.01649 0.01304
## Cumulative Proportion  0.77782 0.80369 0.8249 0.84431 0.86178 0.87826 0.89131
##                           PC15    PC16    PC17    PC18   PC19   PC20   PC21
## Standard deviation     0.78402 0.73469 0.70753 0.67899 0.6716 0.6529 0.6183
## Proportion of Variance 0.01254 0.01102 0.01022 0.00941 0.0092 0.0087 0.0078
## Cumulative Proportion  0.90385 0.91487 0.92508 0.93449 0.9437 0.9524 0.9602
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.59744 0.56051 0.52944 0.48061 0.45739 0.38749 0.34100
## Proportion of Variance 0.00728 0.00641 0.00572 0.00471 0.00427 0.00306 0.00237
## Cumulative Proportion  0.96748 0.97389 0.97961 0.98433 0.98860 0.99166 0.99403
##                          PC29    PC30    PC31    PC32   PC33    PC34    PC35
## Standard deviation     0.2521 0.24417 0.21778 0.21574 0.1985 0.12539 0.11166
## Proportion of Variance 0.0013 0.00122 0.00097 0.00095 0.0008 0.00032 0.00025
## Cumulative Proportion  0.9953 0.99655 0.99752 0.99847 0.9993 0.99959 0.99984
##                           PC36    PC37     PC38     PC39      PC40      PC41
## Standard deviation     0.08531 0.01784 0.004014 0.003583 1.953e-15 1.384e-15
## Proportion of Variance 0.00015 0.00001 0.000000 0.000000 0.000e+00 0.000e+00
## Cumulative Proportion  0.99999 1.00000 1.000000 1.000000 1.000e+00 1.000e+00
##                            PC42      PC43      PC44     PC45      PC46
## Standard deviation     1.19e-15 7.844e-16 6.284e-16 4.89e-16 3.962e-16
## Proportion of Variance 0.00e+00 0.000e+00 0.000e+00 0.00e+00 0.000e+00
## Cumulative Proportion  1.00e+00 1.000e+00 1.000e+00 1.00e+00 1.000e+00
##                             PC47      PC48      PC49
## Standard deviation     3.189e-16 3.138e-16 9.603e-17
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00
```

The first 20 principal component capture 95.2% of variance of the data, this should be sufficient for modelling and reduces the dimensionality from 56 to 27.

```
# retrieve first 20 components and combine with company and year
# as it was done for the variable selection approach
n_components <- 20
df_joined_shift_pca <-
  bind_cols(
    df_joined_shift %>% select(c(
```

```r
      index,
      company.sales,
      company.rest,
      year,
      log.interim_sales_Q1,
      log.interim_sales_Q2,
      log.interim_sales_Q3,
      log.interim_sales_Q4
  )),
    pca_result$x[, 1:n_components]
  )
df_joined_shift_pca
```

```
## # A tibble: 5,558 x 28
##    index company.sales company.rest  year log.interim_sales_Q1
##    <int> <chr>         <chr>        <dbl>                <dbl>
##  1     1 3M COMPANY    3M            2002                 15.4
##  2     2 3M COMPANY    3M            2003                 15.5
##  3     3 3M COMPANY    3M            2004                 15.5
##  4     4 3M COMPANY    3M            2005                 15.6
##  5     5 3M COMPANY    3M            2006                 15.7
##  6     6 3M COMPANY    3M            2007                 15.7
##  7     7 3M COMPANY    3M            2008                 15.5
##  8     8 3M COMPANY    3M            2009                 15.7
##  9     9 3M COMPANY    3M            2010                 15.9
## 10    10 3M COMPANY    3M            2011                 15.9
## # i 5,548 more rows
## # i 23 more variables: log.interim_sales_Q2 <dbl>, log.interim_sales_Q3 <dbl>,
## #   log.interim_sales_Q4 <dbl>, PC1 <dbl>, PC2 <dbl>, PC3 <dbl>, PC4 <dbl>,
## #   PC5 <dbl>, PC6 <dbl>, PC7 <dbl>, PC8 <dbl>, PC9 <dbl>, PC10 <dbl>,
## #   PC11 <dbl>, PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>,
## #   PC17 <dbl>, PC18 <dbl>, PC19 <dbl>, PC20 <dbl>
```

```r
# check correlated pairs after pca dimensionality reduction
corr_cross(
  df_joined_shift_pca %>% select( # exclude target variables from analysis
    -c(
      index,
      company.sales,
      company.rest,
      year,
      log.interim_sales_Q1,
      log.interim_sales_Q2,
      log.interim_sales_Q3,
      log.interim_sales_Q4
    )
  ),
  # name of dataset
  max_pvalue = 0.05,
  # display only significant correlations (at 5% level)
  rm.na = TRUE,
  # remove NAs
  top = 25 # show only top 25 variable pairs
)
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `hjust = ifelse(.data$abs < max(.data$abs)/1.5, -0.1, 1.1)`.
## Caused by warning in `max()`:
## ! no non-missing arguments to max; returning -Inf
```

Ranked Cross−Correlations

0 most relevant [NAs removed]

Correlations with p−value < 0.05

Above we can see that in contrast to the variable selection approach, where we used a absolute correlation cut off of 0.6, the data frame that was processed with PCA does not involve any correlations.

3.5.4 Show overview of processed data sets

```r
cat(paste(
  "Dataset with variable selection consists of",
  nrow(df_joined_shift_var_sel),
  "observations with",
  ncol(df_joined_shift_var_sel),
  "variables.\n"
))
```

```
## Dataset with variable selection consists of 5558 observations with 27 variables.
```

```r
cat(paste(
  "Dataset with PCA consists of",
  nrow(df_joined_shift_pca),
  "observations with",
  ncol(df_joined_shift_pca),
  "variables.\n"
))
```

```
## Dataset with PCA consists of 5558 observations with 28 variables.
```

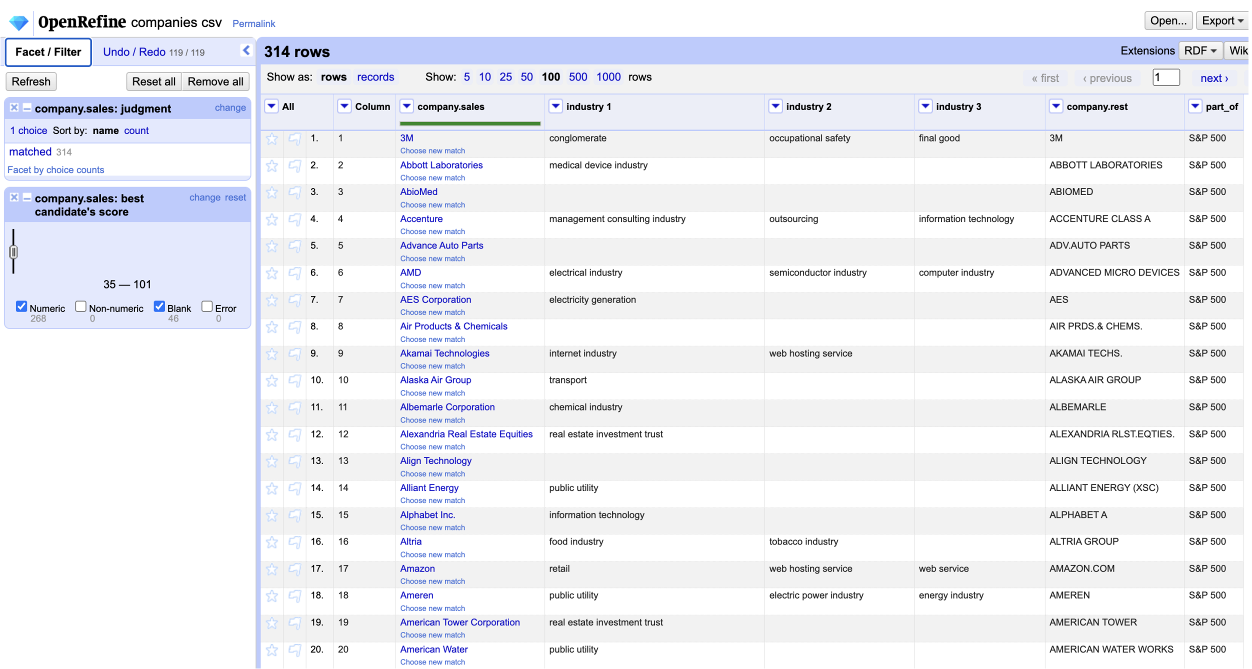
3.5.5 Save unique companies of joined data frame to enrich it with industry information with OpenRefine

To later on analyse if the performance of our models differs between industry sectors, we will enrich our data with this information by using OpenRefine.

```
write.csv(df_joined_shift_var_sel %>%
            select(c(company.sales, company.rest)) %>%
            distinct(),
          COMPANY_OUTPUT_PATH)
```

3.5.6 Add industry information from WikiData with Openrefine

This step was performed within Google OpenRefine. OpenRefine offers the possibility to import a data set and to link it with knowledge bases like WikiData. To achieve linkage (reconciliation) between our companies and the knowledge base, the company names are used. For some companies it was necessary to resolve them manually. After the reconciliation is done, we can add new variables from the data that is present on WikiData. In this case, we will use the top 3 industries that can be found. This information will later be used when evaluating the models. Here we will examine if the prediction quality differs for different industries/business sectors. Down below, you can see the the results we obtained from OpenRefine.



All	Column	company.sales	industry 1	industry 2	industry 3	company.rest	part_of
1.	1	3M Choose new match	conglomerate	occupational safety	final good	3M	S&P 500
2.	2	Abbott Laboratories Choose new match	medical device industry			ABBOTT LABORATORIES	S&P 500
3.	3	AbioMed Choose new match				ABIOMED	S&P 500
4.	4	Accenture Choose new match	management consulting industry	outsourcing	information technology	ACCENTURE CLASS A	S&P 500
5.	5	Advance Auto Parts Choose new match				ADV.AUTO PARTS	S&P 500
6.	6	AMD Choose new match	electrical industry	semiconductor industry	computer industry	ADVANCED MICRO DEVICES	S&P 500
7.	7	AES Corporation Choose new match	electricity generation			AES	S&P 500
8.	8	Air Products & Chemicals Choose new match				AIR PRDS.& CHEMS.	S&P 500
9.	9	Akamai Technologies Choose new match	internet industry	web hosting service		AKAMAI TECHS.	S&P 500
10.	10	Alaska Air Group Choose new match	transport			ALASKA AIR GROUP	S&P 500
11.	11	Albemarle Corporation Choose new match	chemical industry			ALBEMARLE	S&P 500
12.	12	Alexandria Real Estate Equities Choose new match	real estate investment trust			ALEXANDRIA RLST.EQTIES.	S&P 500
13.	13	Align Technology Choose new match				ALIGN TECHNOLOGY	S&P 500
14.	14	Alliant Energy Choose new match	public utility			ALLIANT ENERGY (XSC)	S&P 500
15.	15	Alphabet Inc. Choose new match	information technology			ALPHABET A	S&P 500
16.	16	Altria Choose new match	food industry	tobacco industry		ALTRIA GROUP	S&P 500
17.	17	Amazon Choose new match	retail	web hosting service	web service	AMAZON.COM	S&P 500
18.	18	Ameren Choose new match	public utility	electric power industry	energy industry	AMEREN	S&P 500
19.	19	American Tower Corporation Choose new match	real estate investment trust			AMERICAN TOWER	S&P 500
20.	20	American Water Choose new match	public utility			AMERICAN WATER WORKS	S&P 500

Figure 2: OpenRefine-Data-Enrichment

3.6. Train test split

The last step will now be to split our data into training and testing parts. For that we will select the last 5 years of each company as our testing part, and the rest of the data for the training of our models.

3.6.1 Retrieve train and test indices

```
n_year_test <- 5 # define test horizon of 5 years

indices_test <- df_joined_shift_var_sel %>%
```

```
  group_by(company.sales) %>% # for each company
  mutate(in_group_n = n()) %>% # count number of observations per company
  arrange(year) %>% # sort by year
  mutate(in_group_index = row_number()) %>% # add indices within each company group
  filter(in_group_index > in_group_n - n_year_test) %>% # filter for companies that are in testing
  ungroup() %>%
  arrange(index) %>%
  select(index)
indices_test <- indices_test$index

# use indices that are not in testing for training
indices_train <-
  setdiff(df_joined_shift_var_sel$index, indices_test)

cat(
  paste0(
    "Created ",
    length(indices_train),
    "/",
    100 * length(indices_train) / nrow(df_joined_shift_var_sel),
    "% indices for training\nand ",
    length(indices_test),
    "/",
    100 * length(indices_test) / nrow(df_joined_shift_var_sel),
    "% indices for testing."
  )
)
```

```
## Created 3993/71.8423893486866% indices for training
## and 1565/28.1576106513134% indices for testing.
```

3.6.2 Create train and test data sets and pivot to longer format

To make the data better usable for modeling, we will pivot it to a longer format, i.e., merge the interim sales variables for each quarter into one variable and add an additional variable that indicates the respective quarter.

```
# first variable selection data
# train
df_train_var_sel <- df_joined_shift_var_sel %>%
  filter(index %in% indices_train) %>%
  pivot_longer(
  cols = c(
    log.interim_sales_Q1,
    log.interim_sales_Q2,
    log.interim_sales_Q3,
    log.interim_sales_Q4
  ),
  names_to = "quarter",
  values_to = "log.interim_sales"
) %>%
  mutate(quarter = sub("log.interim_sales_Q", "", quarter))
head(df_train_var_sel)
```

```
## # A tibble: 6 x 25
```

```
##   index company.sales company.rest  year log.borrowings_repayable_lt_~1 net_debt
##   <int> <chr>         <chr>        <dbl>                          <dbl>    <dbl>
## 1     1 3M COMPANY    3M            2002                           14.1  2277000
## 2     1 3M COMPANY    3M            2002                           14.1  2277000
## 3     1 3M COMPANY    3M            2002                           14.1  2277000
## 4     1 3M COMPANY    3M            2002                           14.1  2277000
## 5     2 3M COMPANY    3M            2003                           14.0  2761000
## 6     2 3M COMPANY    3M            2003                           14.0  2761000
## # i abbreviated name: 1: log.borrowings_repayable_lt_1_year
## # i 19 more variables: log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>,
## #   cash_earnings_per_share <dbl>, log.cost_of_sales <dbl>,
## #   log.dividends_per_share <dbl>, exceptional_items <dbl>, ...
```

```
# test
df_test_var_sel <- df_joined_shift_var_sel %>%
  filter(index %in% indices_test) %>%
  pivot_longer(
  cols = c(
    log.interim_sales_Q1,
    log.interim_sales_Q2,
    log.interim_sales_Q3,
    log.interim_sales_Q4
  ),
  names_to = "quarter",
  values_to = "log.interim_sales"
) %>%
  mutate(quarter = sub("log.interim_sales_Q", "", quarter))
head(df_test_var_sel)
```

```
## # A tibble: 6 x 25
##   index company.sales company.rest  year log.borrowings_repayable_lt_~1 net_debt
##   <int> <chr>         <chr>        <dbl>                          <dbl>    <dbl>
## 1    15 3M COMPANY    3M            2016                           14.5  8716000
## 2    15 3M COMPANY    3M            2016                           14.5  8716000
## 3    15 3M COMPANY    3M            2016                           14.5  8716000
## 4    15 3M COMPANY    3M            2016                           14.5  8716000
## 5    16 3M COMPANY    3M            2017                           13.8  8869000
## 6    16 3M COMPANY    3M            2017                           13.8  8869000
## # i abbreviated name: 1: log.borrowings_repayable_lt_1_year
## # i 19 more variables: log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>,
## #   cash_earnings_per_share <dbl>, log.cost_of_sales <dbl>,
## #   log.dividends_per_share <dbl>, exceptional_items <dbl>, ...
```

```
# PCA
# train
df_train_pca <- df_joined_shift_pca %>%
  filter(index %in% indices_train) %>%
  pivot_longer(
  cols = c(
```

```r
    log.interim_sales_Q1,
    log.interim_sales_Q2,
    log.interim_sales_Q3,
    log.interim_sales_Q4
  ),
  names_to = "quarter",
  values_to = "log.interim_sales"
) %>%
  mutate(quarter = sub("log.interim_sales_Q", "", quarter))
head(df_train_pca)
```

```
## # A tibble: 6 x 26
##   index company.sales company.rest  year   PC1   PC2    PC3   PC4   PC5      PC6
##   <int> <chr>         <chr>        <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl>
## 1     1 3M COMPANY    3M            2002 -1.13  1.92 -0.868 0.655 0.180 -0.142
## 2     1 3M COMPANY    3M            2002 -1.13  1.92 -0.868 0.655 0.180 -0.142
## 3     1 3M COMPANY    3M            2002 -1.13  1.92 -0.868 0.655 0.180 -0.142
## 4     1 3M COMPANY    3M            2002 -1.13  1.92 -0.868 0.655 0.180 -0.142
## 5     2 3M COMPANY    3M            2003 -1.45  1.68 -0.889 0.965 0.149 -0.00661
## 6     2 3M COMPANY    3M            2003 -1.45  1.68 -0.889 0.965 0.149 -0.00661
## # i 16 more variables: PC7 <dbl>, PC8 <dbl>, PC9 <dbl>, PC10 <dbl>, PC11 <dbl>,
## #   PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>,
## #   PC18 <dbl>, PC19 <dbl>, PC20 <dbl>, quarter <chr>, log.interim_sales <dbl>
```

```r
# test
df_test_pca <- df_joined_shift_pca %>%
  filter(index %in% indices_test) %>%
  pivot_longer(
  cols = c(
    log.interim_sales_Q1,
    log.interim_sales_Q2,
    log.interim_sales_Q3,
    log.interim_sales_Q4
  ),
  names_to = "quarter",
  values_to = "log.interim_sales"
) %>%
  mutate(quarter = sub("log.interim_sales_Q", "", quarter))
head(df_test_pca)
```

```
## # A tibble: 6 x 26
##   index company.sales company.rest  year   PC1   PC2    PC3   PC4    PC5    PC6
##   <int> <chr>         <chr>        <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl>
## 1    15 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879
## 2    15 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879
## 3    15 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879
## 4    15 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879
## 5    16 3M COMPANY    3M            2017 -4.42  1.48 -0.866  1.98 -0.690 -0.782
## 6    16 3M COMPANY    3M            2017 -4.42  1.48 -0.866  1.98 -0.690 -0.782
## # i 16 more variables: PC7 <dbl>, PC8 <dbl>, PC9 <dbl>, PC10 <dbl>, PC11 <dbl>,
## #   PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>,
## #   PC18 <dbl>, PC19 <dbl>, PC20 <dbl>, quarter <chr>, log.interim_sales <dbl>
```

3.6.3 Write data to files

Finally, we write the data sets to csv files.

```
# variable selection data
# train
write.csv(
  df_train_var_sel,
  TRAIN_VAR_SEL_OUTPUT_PATH,
  row.names = FALSE
  )
# test
write.csv(
  df_test_var_sel,
  TEST_VAR_SEL_OUTPUT_PATH,
  row.names = FALSE
)

# PCA data
# train
write.csv(
  df_train_pca,
  TRAIN_PCA_OUTPUT_PATH,
  row.names = FALSE
)
# test
write.csv(
  df_test_pca,
  TEST_PCA_OUTPUT_PATH,
  row.names = FALSE
)
```

4. Modeling

Now we will enter the Modeling stage of CRISP-DM. In the following sections, we will first of all perform naive forecasts, that use the last observation of the training data set as the forecasting value. This simple method will act as a baseline to evaluate the more sophisticated approaches. Followed by that, we will fit ARIMA models for each particular company time series. Finally, we will train two XGBoost models, one using the data set with variable selection and one using the data set that we generated by using principal components, to forecast the interim sales variable.

4.1 Naive Forecasting

We will now perform a naive forecast that acts as a baseline for our subsequent models. For the naive forecast, we will simply use the latest value of the interim sales variable as forecast for all time points in the forecasting horizon (5 years) for a particular company.

Imports

```
if(!require(tidyverse)) {
  install.packages("tidyverse")
}
library(tidyverse)
```

Constants

```
BASE_PATH <- "../data/processed"
TRAIN_VAR_SEL_PATH <- paste(BASE_PATH, "train_var_sel.csv", sep = "/")
TEST_VAR_SEL_PATH <- paste(BASE_PATH, "test_var_sel.csv", sep = "/")
FORECAST_NAIVE_PATH <- paste(BASE_PATH, "forecast_naive.csv", sep= "/")
```

4.1.1 Load data

As we will only need the sales variable for our forecast, it does not matter if we load the data set with variable selection or with PCA.

```
df_train <- read_csv(TRAIN_VAR_SEL_PATH, show_col_types = FALSE)

# select relevant variables
df_train <-
  df_train %>% select(
    c(
      company.sales,
      company.rest,
      year,
      quarter,
      log.interim_sales
    )
  )
head(df_train)
```

```
## # A tibble: 6 x 5
##   company.sales company.rest  year quarter log.interim_sales
##   <chr>         <chr>        <dbl>   <dbl>             <dbl>
## 1 3M COMPANY    3M            2002       1              15.4
## 2 3M COMPANY    3M            2002       2              15.4
```

```
## 3 3M COMPANY    3M            2002       3              15.4
## 4 3M COMPANY    3M            2002       4              15.4
## 5 3M COMPANY    3M            2003       1              15.5
## 6 3M COMPANY    3M            2003       2              15.5
```

```r
df_test <- read_csv(TEST_VAR_SEL_PATH, show_col_types = FALSE)

# select relevant variables
df_test <-
  df_test %>% select(
    c(
      company.sales,
      company.rest,
      year,
      quarter,
      log.interim_sales
    )
  )
head(df_test)
```

```
## # A tibble: 6 x 5
##   company.sales company.rest  year quarter log.interim_sales
##   <chr>         <chr>        <dbl>   <dbl>             <dbl>
## 1 3M COMPANY    3M            2016       1              15.9
## 2 3M COMPANY    3M            2016       2              15.9
## 3 3M COMPANY    3M            2016       3              16.0
## 4 3M COMPANY    3M            2016       4              15.9
## 5 3M COMPANY    3M            2017       1              16.0
## 6 3M COMPANY    3M            2017       2              16.0
```

4.1.2 Perform Naive Forecast

We will now select the most recent observation for each company in the training data set and then assign this value as a prediction for the full prediction horizon of each company.

```r
# select most recent sales value per company
most_recent_observations <- df_train %>% group_by(company.sales) %>%
  filter(year == max(year) & quarter == max(quarter)) %>%
  select(company.sales, log.interim_sales) %>%
  ungroup()

# create copy of test data frame
df_forecast <- data.frame(df_test)

# assign most recent sales value to prediction data frame
for (company in unique(df_forecast$company.sales)) {
  forecast_value <-
    as.double(
      most_recent_observations[most_recent_observations$company.sales == company,
                               "log.interim_sales"])
  df_forecast[df_forecast$company.sales == company, "log.interim_sales"] <-
    forecast_value
}
head(df_forecast)
```

```
##   company.sales company.rest year quarter log.interim_sales
```

```
## 1    3M COMPANY           3M 2016       1          15.85958
## 2    3M COMPANY           3M 2016       2          15.85958
## 3    3M COMPANY           3M 2016       3          15.85958
## 4    3M COMPANY           3M 2016       4          15.85958
## 5    3M COMPANY           3M 2017       1          15.85958
## 6    3M COMPANY           3M 2017       2          15.85958
```

4.1.3 Write naive forecasts to file

Finally, we write the naive forecasts to a csv file.

```
write.csv(
  df_forecast,
  FORECAST_NAIVE_PATH,
  row.names = FALSE
)
```

4.2 ARIMA Forecasting

Within this notebook, we will first of all now fit will automatically fit an ARIMA model for three example companies and visualize the forecasts. After that, we will fit ARIMA models for each particular company and write the forecasting results to as csv file.

ARIMA (Autoregressive Integrated Moving Average) is a one-dimensional mathematical model for forecasting future values in time series data. It involves three key components: autoregression (AR), differencing (I), and moving average (MA). The autoregressive component examines the relationship between the current value of a variable and its previous values. It assumes that the current value can be explained by a linear combination of past observations. The order of autoregression (p) determines the number of previous observations that are considered in the model. The differencing component (I) is employed to transform the time series into a stationary series, i.e., a time series that does not have seasonality and trend. Seasonality and trend are eliminated by computing the differences between consecutive observations. The order of differencing (d), specifies the number of differencing operations needed to achieve stationarity. The moving average component (MA) captures the short-term fluctuations or noise in the data. It considers the relationship between the current value and past forecast errors. The order of the moving average (q) specifies the number of previous forecast errors to include in the model. To determine the appropriate values for the ARIMA parameters (p, d, q), the autocorrelation function (ACF) and partial autocorrelation function (PACF) are typically analyzed. As we deal with many time series' at once, we can not analyze the ACFs and PACFs by hand. Fortunately, there is a function in R (auto.arima) that automatically finds the ARIMA parameters for a particular time series.

Imports

```r
if(!require(tidyverse)) {
  install.packages("tidyverse")
}
library(tidyverse)

if (!require(forecast)) {
  install.packages("forecast")
}
```

```
## Loading required package: forecast

##
## Attaching package: 'forecast'

## The following object is masked from 'package:modeest':
##
##     naive
```

```r
library(forecast)
```

Constants

```r
BASE_PATH <- "../data/processed"
TRAIN_VAR_SEL_PATH <- paste(BASE_PATH, "train_var_sel.csv", sep = "/")
TEST_VAR_SEL_PATH <- paste(BASE_PATH, "test_var_sel.csv", sep = "/")
FORECAST_ARIMA_PATH <- paste(BASE_PATH, "forecast_ARIMA.csv", sep= "/")
```

4.2.1 Load data

As we will only need the sales variable for our forecast, it does not matter if we load the data set with variable selection or with PCA.

```r
df_train <- read_csv(TRAIN_VAR_SEL_PATH, show_col_types = FALSE)

# select relevant variables
df_train <-
  df_train %>% select(
    c(
      company.sales,
      company.rest,
      year,
      quarter,
      log.interim_sales
    )
  )
head(df_train)
```

```
## # A tibble: 6 x 5
##   company.sales company.rest  year quarter log.interim_sales
##   <chr>         <chr>        <dbl>   <dbl>             <dbl>
## 1 3M COMPANY    3M            2002       1              15.4
## 2 3M COMPANY    3M            2002       2              15.4
## 3 3M COMPANY    3M            2002       3              15.4
## 4 3M COMPANY    3M            2002       4              15.4
## 5 3M COMPANY    3M            2003       1              15.5
## 6 3M COMPANY    3M            2003       2              15.5
```

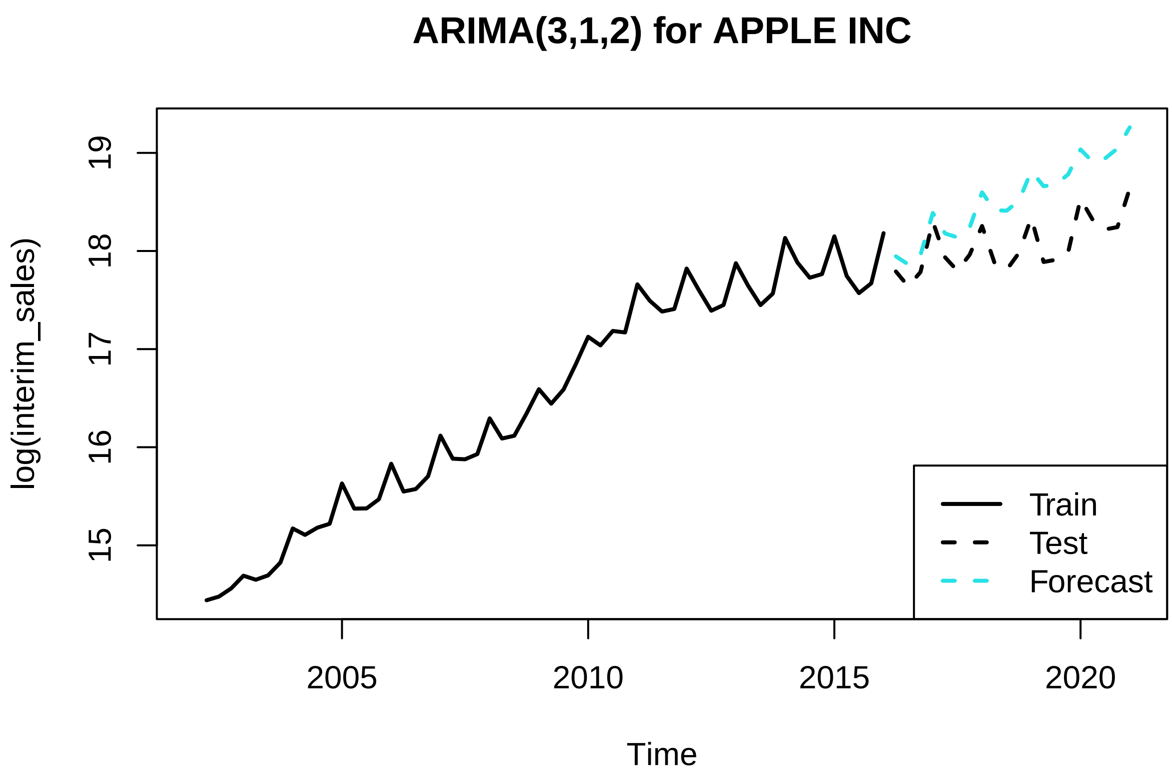
```r
df_test <- read_csv(TEST_VAR_SEL_PATH, show_col_types = FALSE)

# select relevant variables
df_test <-
  df_test %>% select(
    c(
      company.sales,
      company.rest,
      year,
      quarter,
      log.interim_sales
    )
  )
head(df_test)
```

```
## # A tibble: 6 x 5
##   company.sales company.rest  year quarter log.interim_sales
##   <chr>         <chr>        <dbl>   <dbl>             <dbl>
## 1 3M COMPANY    3M            2016       1              15.9
## 2 3M COMPANY    3M            2016       2              15.9
## 3 3M COMPANY    3M            2016       3              16.0
## 4 3M COMPANY    3M            2016       4              15.9
## 5 3M COMPANY    3M            2017       1              16.0
## 6 3M COMPANY    3M            2017       2              16.0
```

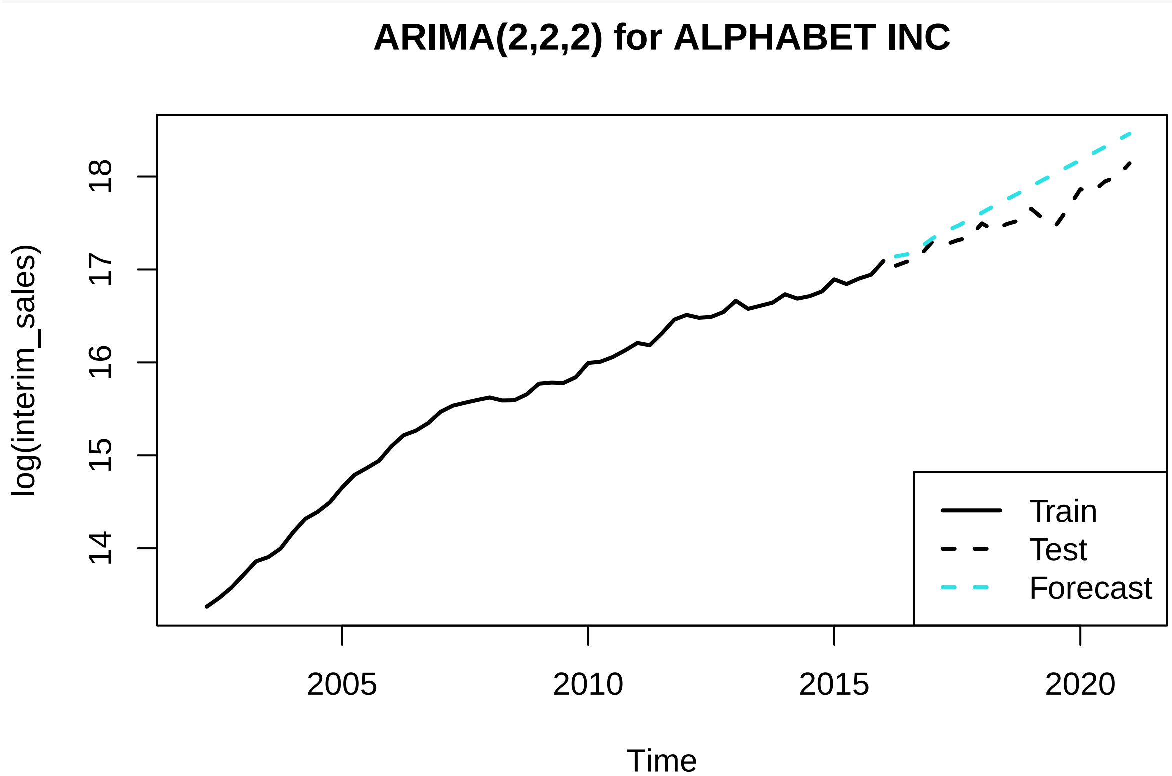
4.2.2 Perform ARIMA forecast on some example companies

As we can not inspect the results of auto.arima for all companies by hand, we will now fit ARIMA models on some example companies and inspect the results.

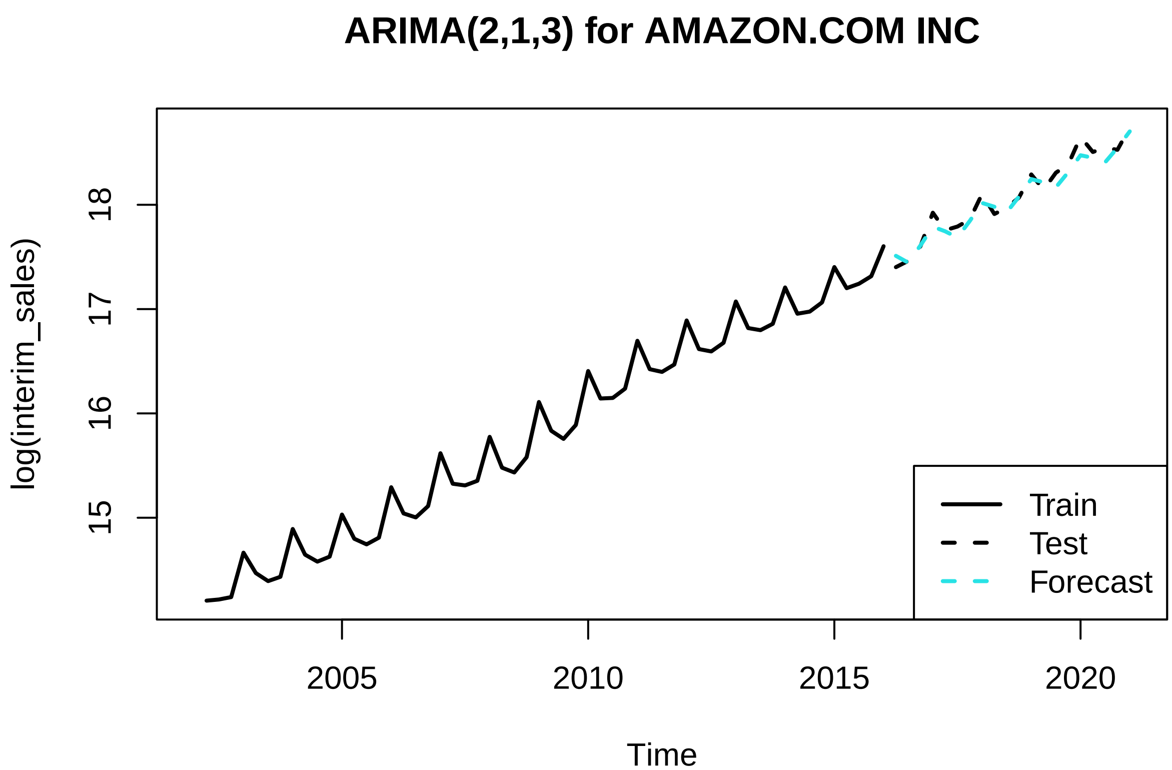
```r
plot_arima_forecast <-
  function(df_train, df_test, company) {
    df_train_sel <-
      df_train[df_train$company.sales == company,]
    df_test_sel <-
      df_test[df_test$company.sales == company,]

    # fit ARIMA model
    fit.arima <- auto.arima(df_train_sel$log.interim_sales)
    forecast.arima <- forecast(fit.arima, h = 20)
    orders = arimaorder(fit.arima)

    # plot train and test part of time series
    plot(
      df_train_sel$year + (as.integer(df_train_sel$quarter) / 4),
      df_train_sel$log.interim_sales,
      type = "l",
      lwd=2,
      xlim = c(min(df_train_sel$year), max(df_test_sel$year) + 1),
      ylim = c(
        min(
          df_train_sel$log.interim_sales,
          df_test_sel$log.interim_sales,
          forecast.arima$mean
          ),
        max(
          df_train_sel$log.interim_sales,
          df_test_sel$log.interim_sales,
          forecast.arima$mean
        )
      ),
      main = paste0("ARIMA(", orders[1], ",", orders[2], ",", orders[3],  ") for ", company),
      xlab = "Time",
      ylab = "log(interim_sales)"
    )
    lines(df_test_sel$year + (as.integer(df_test_sel$quarter) / 4),
          df_test_sel$log.interim_sales,
          lty = 2, lwd=2)

    lines(
      df_test_sel$year + (as.integer(df_test_sel$quarter) / 4),
      forecast.arima$mean,
      lty = 2,
      col = 5,
      lwd=2
    )

    legend("bottomright", legend = c("Train", "Test", "Forecast"), col = c(1, 1, 5), lty = c(1, 2, 2), 

  }

plot_arima_forecast(df_train, df_test, "APPLE INC")
```




```
plot_arima_forecast(df_train, df_test, "ALPHABET INC")
```




```
plot_arima_forecast(df_train, df_test, "AMAZON.COM INC")
```



In the plots above, we can see the ARIMA models that were automatically fitted for Apple, Alphabet and Amazon. We can see that auto.arima found different parameters for each of the companies. For Apple and Amazon, the automatically fitted ARIMA model uses first order differencing (d=1). For Alphabet, the algorithm chose second order differencing (d=2). A reason for this is the different structure of the time series', as it is also visible in the plots. If we look at the values forecasted by ARIMA in cyan, we can see that they look quite reasonable. Especially for Amazon where the forecasted values are almost identical to the values in the test data. For the Apple time series, ARIMA does a good job in capturing the seasonal pattern of the series, but overestimates the slope of the trend. For Alphabet, the model captures almost no seasonal structure and also slightly overestimates the trend of the time series.

4.2.3 Perform ARIMA forecast on all companies

After visualizing the ARIMA forecasts for three example companies, we will now perform automatic ARIMA forecasting for all companies.

```
# create copy of test data frame
df_forecast <- data.frame(df_test)
df_forecast[, "log.interim_sales"] <- -1

for (company in unique(df_forecast$company.sales)){
  # select train part of time series
  train_ts <- df_train[df_train$company.sales == company, "log.interim_sales"]

  # fit ARIMA model
  fit.arima <- auto.arima(train_ts)
  forecast.arima <- forecast(fit.arima, h = 20) # h = 5 years * 4 quarters

  # assign forecast values to data frames
  df_forecast[df_forecast$company.sales == company, "log.interim_sales"] <- forecast.arima$mean
}
```

```r
head(df_forecast)
```

```
##   company.sales company.rest year quarter log.interim_sales
## 1    3M COMPANY           3M 2016       1          15.86020
## 2    3M COMPANY           3M 2016       2          15.89551
## 3    3M COMPANY           3M 2016       3          15.90747
## 4    3M COMPANY           3M 2016       4          15.89619
## 5    3M COMPANY           3M 2017       1          15.90685
## 6    3M COMPANY           3M 2017       2          15.93205
```

4.2.4 Write ARIMA forecasts to file

Fianlly, we write the forecasted values to csv file.

```r
write.csv(
  df_forecast,
  FORECAST_ARIMA_PATH,
  row.names = FALSE
)
```

4.3 XGBoost Forecasting

Within this section, we will train a tree-based multivariate machine learning model called XGBoost (eXtreme Gradient Boosting). This model will use the exogenous variables from the balance sheet and profit and loss statements to forecast the quarterly sales of particular companies. In contrast to the ARIMA section, where we fitted a separate model for each time series, we will only train one XGBoost model that is capable of generating forecasts for all companies.

XGBoost is an ensemble model that combines multiple decision trees to create a forecasting model. It works by iteratively building a series of decision trees and then combining their predictions to obtain the final forecast. Each decision tree is trained to minimize the errors of the previous trees, resulting in a more accurate ensemble model. However, it's important to note that the prediction quality of a machine learning model like XGBoost normally increases with the number of data points it can use for training. Since our data set only consists of 15 training data points for each particular company, there is the possibility that XGBoost cannot show its full potential. Because we previously created two separate data sets, one using a variable selection approach and one using principal components as variables, we will train a separate model for each of those.

Imports

```r
if (!require(tidyverse)) {
  install.packages("tidyverse")
}
library(tidyverse)

if (!require(xgboost)) {
  install.packages("xgboost")
}
```

```
## Loading required package: xgboost

##
## Attaching package: 'xgboost'

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

```r
library(xgboost)

if (!require(caret)) {
  install.packages("caret")
}
library(caret
)

if (!require(doMC)) {
  install.packages("doMC")
}
```

```
## Loading required package: doMC

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: iterators

## Loading required package: parallel
```

```
library(doMC
)
```

Constants

```
BASE_PATH <- "../data/processed"
TRAIN_VAR_SEL_PATH <- paste(BASE_PATH, "train_var_sel.csv", sep = "/")
TEST_VAR_SEL_PATH <- paste(BASE_PATH, "test_var_sel.csv", sep = "/")
TRAIN_PCA_PATH <- paste(BASE_PATH, "train_pca.csv", sep = "/")
TEST_PCA_PATH <- paste(BASE_PATH, "test_pca.csv", sep = "/")
FORECAST_XGBOOST_VAR_SEL_PATH <- paste(BASE_PATH, "forecast_XGBoost_var_sel.csv", sep= "/")
FORECAST_XGBOOST_PCA_PATH <- paste(BASE_PATH, "forecast_XGBoost_pca.csv", sep= "/")
```

4.3.1 XGBoost Forecasting for data with variable selection

First of all, we will fit a model for the data set that was created with a variable selection approach.

```
# variable selection
df_train_var_sel <- read_csv(TRAIN_VAR_SEL_PATH, show_col_types = FALSE)
df_train_var_sel <- df_train_var_sel[, -1] # remove index col
head(df_train_var_sel)
```

4.3.1.1 Load data

```
## # A tibble: 6 x 24
##   company.sales company.rest  year log.borrowings_repayable_lt_1_year net_debt
##   <chr>         <chr>        <dbl>                              <dbl>    <dbl>
## 1 3M COMPANY    3M            2002                               14.1  2277000
## 2 3M COMPANY    3M            2002                               14.1  2277000
## 3 3M COMPANY    3M            2002                               14.1  2277000
## 4 3M COMPANY    3M            2002                               14.1  2277000
## 5 3M COMPANY    3M            2003                               14.0  2761000
## 6 3M COMPANY    3M            2003                               14.0  2761000
## # i 19 more variables: log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>,
## #   cash_earnings_per_share <dbl>, log.cost_of_sales <dbl>,
## #   log.dividends_per_share <dbl>, exceptional_items <dbl>,
## #   extraord_items_after_tax <dbl>, log.interest_capitalsed <dbl>, ...
```

```
df_test_var_sel <- read_csv(TEST_VAR_SEL_PATH, show_col_types = FALSE)
df_test_var_sel <- df_test_var_sel[, -1]
head(df_test_var_sel)
```

```
## # A tibble: 6 x 24
##   company.sales company.rest  year log.borrowings_repayable_lt_1_year net_debt
##   <chr>         <chr>        <dbl>                              <dbl>    <dbl>
## 1 3M COMPANY    3M            2016                               14.5  8716000
## 2 3M COMPANY    3M            2016                               14.5  8716000
## 3 3M COMPANY    3M            2016                               14.5  8716000
```

```
## 4 3M COMPANY    3M           2016                                 14.5  8716000
## 5 3M COMPANY    3M           2017                                 13.8  8869000
## 6 3M COMPANY    3M           2017                                 13.8  8869000
## # i 19 more variables: log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>,
## #   cash_earnings_per_share <dbl>, log.cost_of_sales <dbl>,
## #   log.dividends_per_share <dbl>, exceptional_items <dbl>,
## #   extraord_items_after_tax <dbl>, log.interest_capitalsed <dbl>, ...
```

4.3.1.2 Tune XGBoost model Now we will define a function that fits multiple XGBoost models with different parameters. The train function of the library caret allows to specify a parameter tuneLength which denotes the number of levels for each tuning parameter that is generated for parameter tuning. The set of candidate parameters will be obtained by a random search approach. To ensure that our model will not be overfitting to the training data (i.e., will not generalize to unseen data), we will use a group 5-fold cross-validation approach. Here, the training data set is divided into five equally sized folds based on the company names (such that a time series is not in multiple folds at the same time). The model is trained on four folds and validated on the remaining fold in each iteration, creating five separate evaluations. The final performance metric is the average of these five evaluations. Note that even though, we use multiple cores for fitting the models, this step will be computationally expensive.

```r
# register cores for multicore processing
registerDoMC(cores = 2)

tune_xgboost <- function(df_train, df_test) {

  # set fixed random seed
  set.seed(42)

  # Set the number of folds for cross-validation
  k_folds <- 5

  # Create cross-validation indices based on the company name
  cv_indices <- groupKFold(df_train$company.sales, k = k_folds)

  # Set up train control with k-fold cross-validation
  ctrl <- trainControl(method = "cv", index = cv_indices)

  # fit xgboost model
  model <- train(
    log.interim_sales ~ .,
    # Specify your target variable and predictors
    data = df_train %>% select(year:log.interim_sales),
    method = "xgbTree",
    # allows automatic tuning and specifies
    # number of different values to try for each parameter
    tuneLength = 5, # use 5 levels for each tuning parameter
    metric = "RMSE",
    # Root mean squared error as evaluation metric
    verbosity = 0, # suppress internal deprecation warning
    trControl = ctrl
  )
```

```r
  # filter for best parameter set that was found during optimization
  best_params <- model$results %>% filter(RMSE == min(RMSE))

  # forecast on test set
  df_forecast <- data.frame(df_test) # copy
  df_forecast$log.interim_sales <-
    predict(model$finalModel, newdata = as.matrix(df_test %>% select(year:quarter)))

  return(list(
    model = model,
    best_params = best_params,
    df_forecast = df_forecast
  ))
}

res.var_sel <- tune_xgboost(df_train = df_train_var_sel, df_test = df_test_var_sel)
res.var_sel$best_params
```

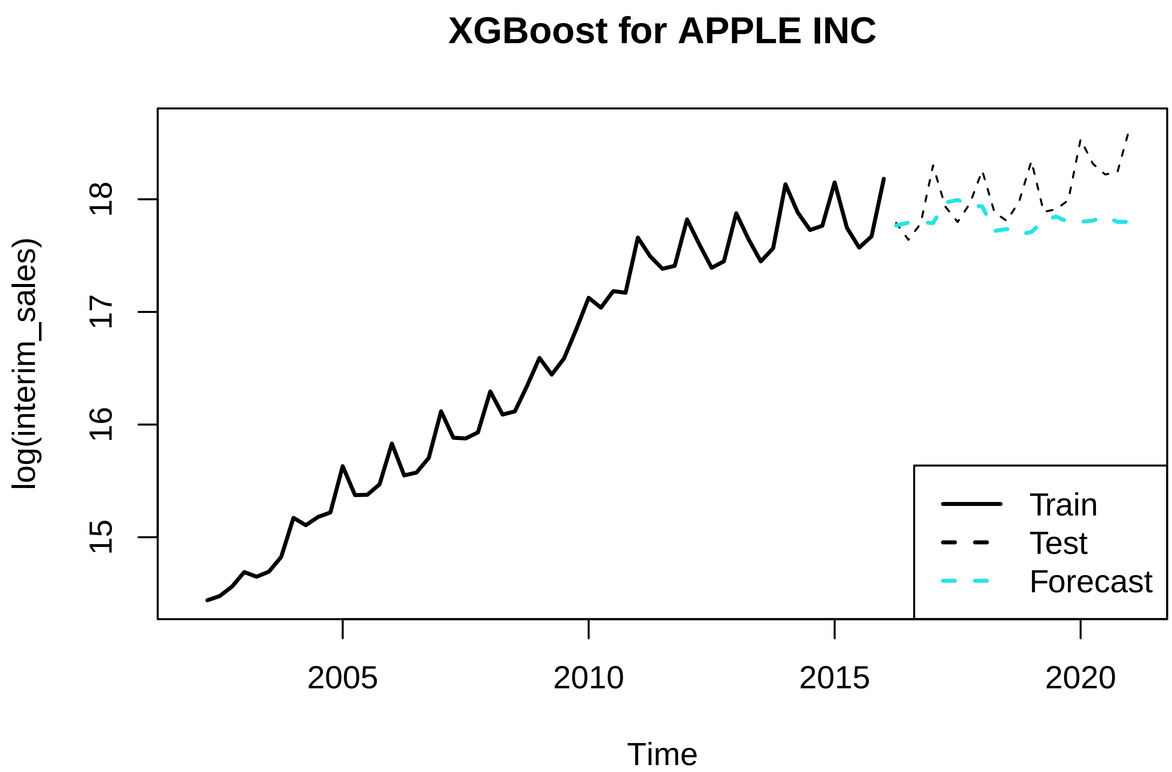
```
##   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.3         3     0              0.8                1     0.625      50
##        RMSE  Rsquared       MAE     RMSESD RsquaredSD     MAESD
## 1 0.2798877 0.9337713 0.1973493 0.02777704 0.01361717 0.0185569
```

In the table above, we can see the best parameters that were found during our XGBoost tuning. Eta controls how much information from a new tree will be used in the Boosting. If it is close to zero we will use only a small piece of information from each new tree. If we set eta to 1 we will use all information from the new tree. Max_depth controls the maximum depth of the trees. Deeper trees have more terminal nodes and fit more data, but are also more prone to overfitting. Gamma specifies the minimum loss reduction to make a split within a tree and is kept at zero. Colsample_bytree denotes the proportion of variables that will be used to fit a new tree. Min_child_weight defines the minimum sum of weights of all observations required in a child and is used to control overfitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Subsample denotes the proportion of observations that are selected to build a new tree and can also be used to control overfitting. Nrounds defines the number of trees that are included in the final model. RMSE, Rsquared and MAE are the evaluation metrics that are computed for this particular set of parameters. RMSESD, RsquaredSD, and RsquaredSD represent the standard deviations of those metrics. Note that we also tested a version that did not use a 5-fold cross-validation approach. This version did result in lower errors for the training data but higher ones for the test data, which is a clear sign of overfitting.

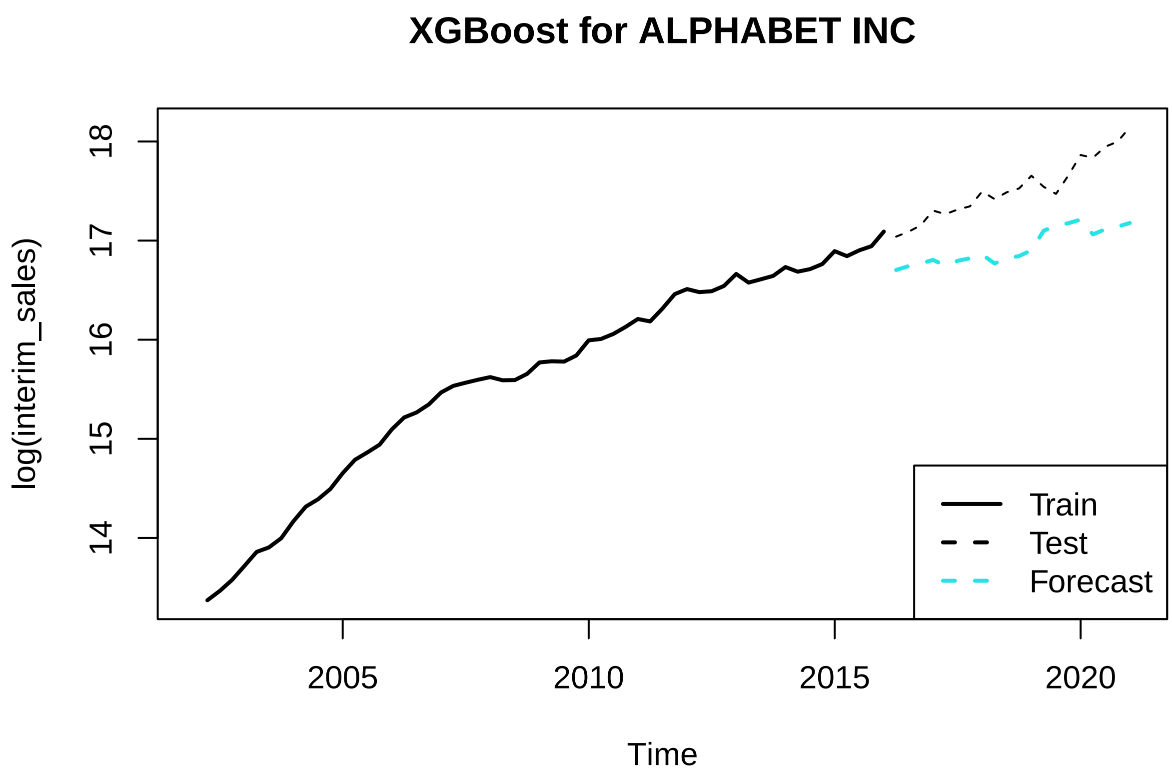
4.3.1.3 Visualize XGBoost forecasts for some example companies As for the ARIMA models, we will now visualize the forecasts by XGBoost on the variable selection data set for Apple, Alphabet and Amazon.

```r
plot_xgb_forecast <-
  function(df_train,
           df_test,
           df_forecast,
           company) {
    df_train_sel <-
      df_train[df_train$company.sales == company, ]
    df_test_sel <-
      df_test[df_test$company.sales == company, ]
    df_forecast_sel <-
      df_forecast[df_forecast$company.sales == company, ]
```

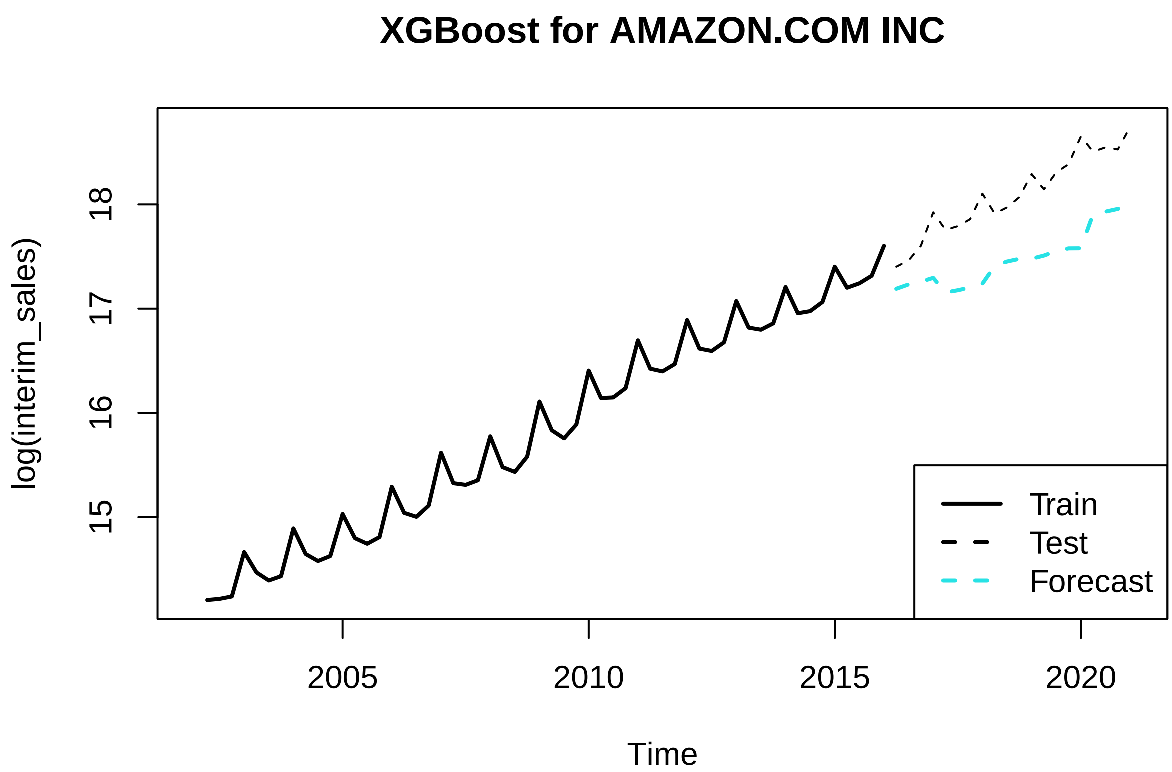
```
    # plot train and test part of time series
    plot(
      df_train_sel$year + (as.integer(df_train_sel$quarter) / 4),
      df_train_sel$log.interim_sales,
      type = "l",
      lwd=2,
      xlim = c(min(df_train_sel$year), max(df_test_sel$year) + 1),
      ylim = c(
        min(
          df_train_sel$log.interim_sales,
          df_test_sel$log.interim_sales,
          df_forecast_sel$log.interim_sales
        ),
        max(
          df_train_sel$log.interim_sales,
          df_test_sel$log.interim_sales,
          df_forecast_sel$log.interim_sales
        )
      ),
      main = paste0("XGBoost for ", company),
      xlab = "Time",
      ylab = "log(interim_sales)"
    )
    lines(df_test_sel$year + (as.integer(df_test_sel$quarter) / 4),
          df_test_sel$log.interim_sales,
          lty = 2)

    lines(
      df_forecast_sel$year + (as.integer(df_forecast_sel$quarter) / 4),
      df_forecast_sel$log.interim_sales,
      lty = 2,
      col = 5,
      lwd=2
    )

    legend(
      "bottomright",
      legend = c("Train", "Test", "Forecast"),
      col = c(1, 1, 5),
      lty = c(1, 2, 2),
      lwd=2
    )

  }

plot_xgb_forecast(
  df_train_var_sel,
  df_test_var_sel,
  res.var_sel$df_forecast,
  "APPLE INC"
)
```




```r
plot_xgb_forecast(
  df_train_var_sel,
  df_test_var_sel,
  res.var_sel$df_forecast,
  "ALPHABET INC"
)
```




```r
plot_xgb_forecast(
  df_train_var_sel,
  df_test_var_sel,
  res.var_sel$df_forecast,
  "AMAZON.COM INC"
)
```



In the plots above, we can see that the forecasts of the XGBoost model with the variable selection data set look less promising as the ARIMA forecasts. For Apple, XGBoost does a good job in forecasting the mean of the first few years. After that the forecast suddenly drops and stays constant. For Alphabet and Amazon, both the trend and the seasonality are not estimated correctly. A reason for that can be the few training observations for each particular company.

4.3.1.4 Write XGBoost forecasts with variable selection to file Finally, we will write the forecasts to a csv file.

```
write.csv(
  res.var_sel$df_forecast,
  FORECAST_XGBOOST_VAR_SEL_PATH,
  row.names = FALSE
)
```

4.3.2 XGBoost Forecasting for data with PCA

Now we will perform the similar steps for the data set with principal component analysis as we did it for the data set with variable selection.

```
# PCA
df_train_pca <- read_csv(TRAIN_PCA_PATH, show_col_types = FALSE)
df_train_pca <- df_train_pca[, -1]
head(df_train_pca)
```

4.3.2.1 Load data

```
## # A tibble: 6 x 25
##   company.sales company.rest  year   PC1   PC2    PC3   PC4   PC5      PC6   PC7
##   <chr>         <chr>        <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl> <dbl>
```

```
## 1 3M COMPANY    3M             2002 -1.13  1.92 -0.868 0.655 0.180 -0.142   0.389
## 2 3M COMPANY    3M             2002 -1.13  1.92 -0.868 0.655 0.180 -0.142   0.389
## 3 3M COMPANY    3M             2002 -1.13  1.92 -0.868 0.655 0.180 -0.142   0.389
## 4 3M COMPANY    3M             2002 -1.13  1.92 -0.868 0.655 0.180 -0.142   0.389
## 5 3M COMPANY    3M             2003 -1.45  1.68 -0.889 0.965 0.149 -0.00661 0.357
## 6 3M COMPANY    3M             2003 -1.45  1.68 -0.889 0.965 0.149 -0.00661 0.357
## # i 15 more variables: PC8 <dbl>, PC9 <dbl>, PC10 <dbl>, PC11 <dbl>,
## #   PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>,
## #   PC18 <dbl>, PC19 <dbl>, PC20 <dbl>, quarter <dbl>, log.interim_sales <dbl>
```

```
df_test_pca <- read_csv(TEST_PCA_PATH, show_col_types = FALSE)
df_test_pca <- df_test_pca[, -1]
head(df_test_pca)
```

```
## # A tibble: 6 x 25
##   company.sales company.rest  year   PC1   PC2    PC3   PC4    PC5    PC6   PC7
##   <chr>         <chr>        <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl> <dbl>
## 1 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 2 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 3 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 4 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 5 3M COMPANY    3M            2017 -4.42  1.48 -0.866  1.98 -0.690 -0.782 0.623
## 6 3M COMPANY    3M            2017 -4.42  1.48 -0.866  1.98 -0.690 -0.782 0.623
## # i 15 more variables: PC8 <dbl>, PC9 <dbl>, PC10 <dbl>, PC11 <dbl>,
## #   PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>,
## #   PC18 <dbl>, PC19 <dbl>, PC20 <dbl>, quarter <dbl>, log.interim_sales <dbl>
```

```r
res.pca <- tune_xgboost(df_train = df_train_pca, df_test = df_test_pca)
res.pca$best_params
```

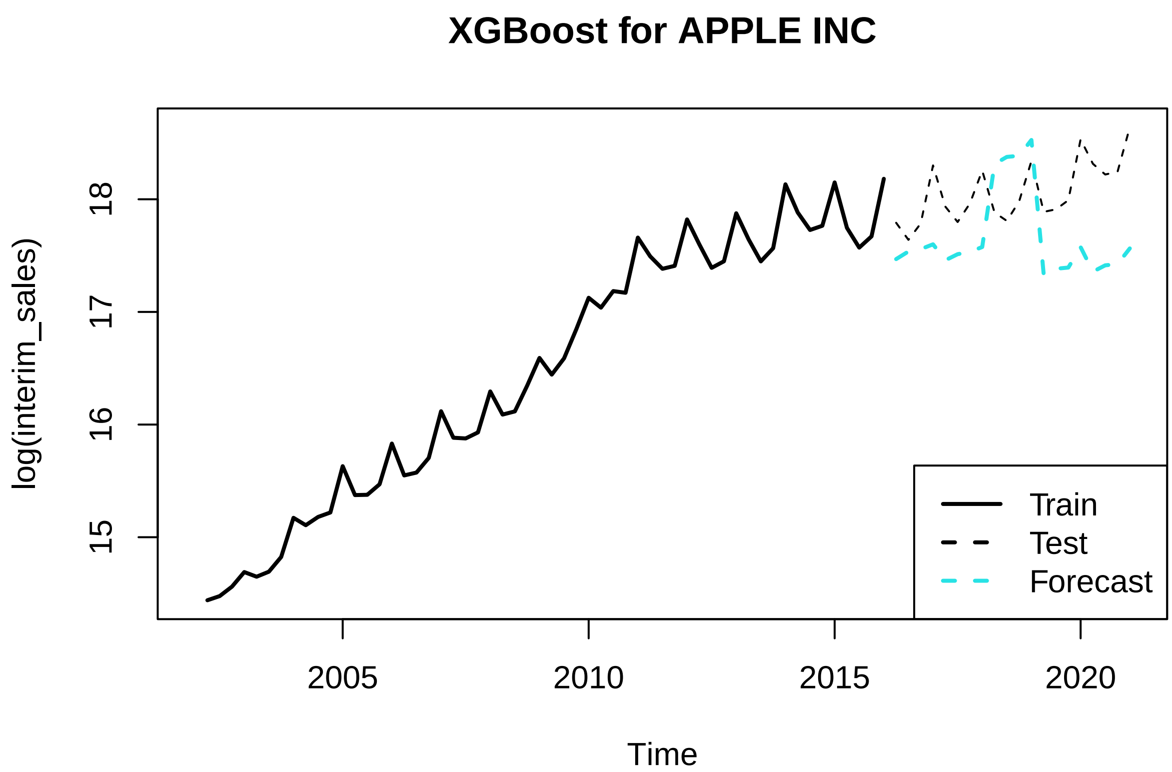
4.3.2.2 Tune XGBoost model

```
##   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.3         3     0              0.6                1     0.875     200
##        RMSE  Rsquared       MAE     RMSESD  RsquaredSD      MAESD
## 1 0.3117294 0.9174388 0.2223103 0.03535652 0.007328582 0.02388478
```

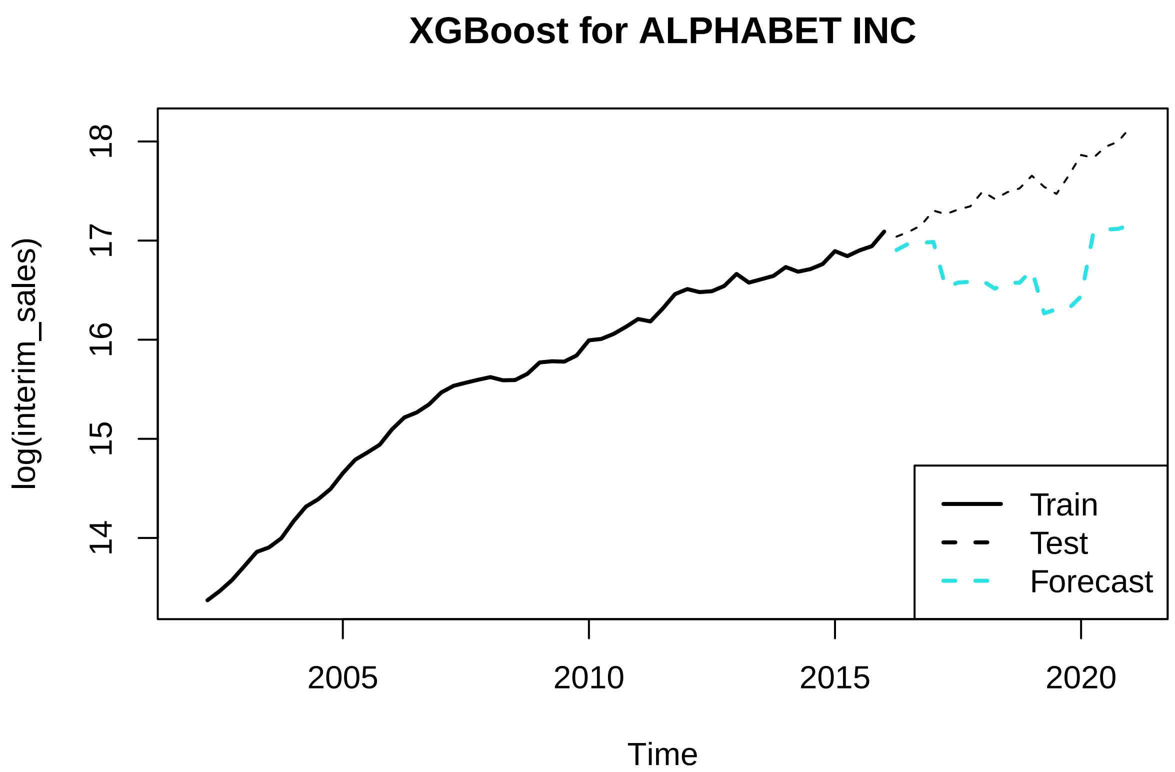
By looking at table above showing the found parameters of the tuning run with the principal component data, we can observe some different results. The parameter tuning found an eta value of 0.4 (vs. 0.3 for variable selection), a max_depth of 2 (vs. 4 for variable selection), a gamma value of 0 (similar to the variable selection data set), a colsample_bytree value of 0.8 and a min_child_weight value of 1, which are also similar to the variable selection data. The values for subsample (0.625 vs. 0.5) and nrounds (250 vs. 50) are again different. The found parameters represent an ensemble model that consists of more but less deep trees for the PCA data set as for the variable selection data set. By looking at the RMSE, we can see, that the error of the model that uses principal components is slightly higher as for the variable selection model. A more in-depth evaluation of both models will be performed in the evaluation notebook.

```r
plot_xgb_forecast(
  df_train_pca,
  df_test_pca,
  res.pca$df_forecast,
  "APPLE INC"
)
```

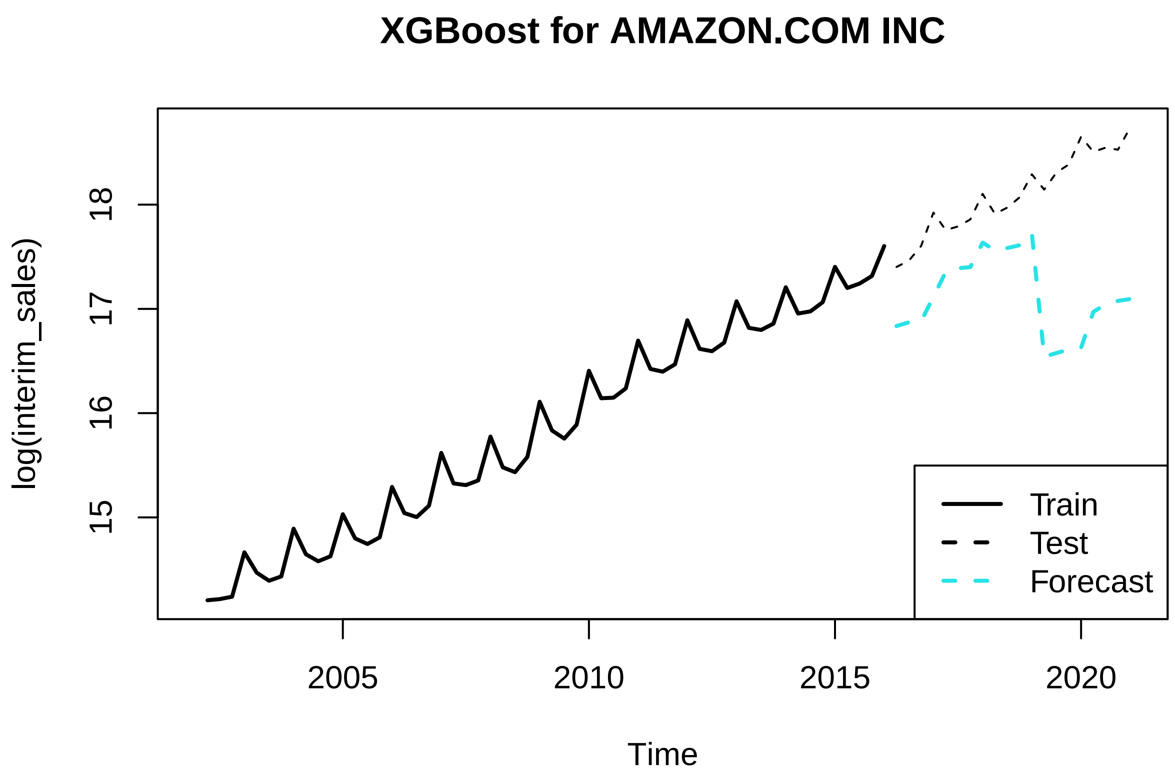
4.3.2.3 Visualize XGBoost forecasts for some example companies




```r
plot_xgb_forecast(
  df_train_pca,
  df_test_pca,
  res.pca$df_forecast,
  "ALPHABET INC"
)
```




```r
plot_xgb_forecast(
  df_train_pca,
  df_test_pca,
  res.pca$df_forecast,
  "AMAZON.COM INC"
)
```



In the visualized forecasts above, we can see that the PCA model shows a different behavior than the model with variable selection data. At least for Alphabet and Amazon, we can see that the forecasts are closer to the test data for the first years, but suddenly show a strange behavior afterwards. At least visually, the PCA model seems to make better forecasts for Apple and Alphabet than the model using variable selection data. But the forecasts of both models are not satisfactory and look unnatural compared to the ARIMA forecasts.

4.3.2.4 Write XGBoost forecasts with PCA to file Finally, we will write the forecasts to a csv file.

```
write.csv(
  res.pca$df_forecast,
  FORECAST_XGBOOST_PCA_PATH,
  row.names = FALSE
)
```

5. Evaluation

The Evaluation stage is the final CRISP-DM stage we will perform within this project. As the name suggests, the goal of this stage is to evaluate the models we produced in the previous stage. Within this section, we will first of all load the test data and the forecasts of all model variants to compute the evaluation metrics root mean squared error (RMSE), mean absolute error (MAE) and mean absolute scaled error (MASE). Followed by that we will use those metrics to perform significance testing with a Wilcoxon signed rank test. The final step will be to utilize the industry sector data obtained with OpenRefine in the Data Preparation stage and to analyze if the model performances vary between industry sectors.

Imports

```r
if (!require(Metrics)) {
  install.packages("Metrics")
}
```

```
## Loading required package: Metrics

##
## Attaching package: 'Metrics'

## The following object is masked from 'package:forecast':
##
##     accuracy

## The following objects are masked from 'package:caret':
##
##     precision, recall

## The following objects are masked from 'package:lares':
##
##     mae, mape, mse, rmse
```

```r
library(Metrics)

if (!require(MASS)) {
  install.packages("MASS")
}
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(MASS)

if (!require(ggplot2)) {
  install.packages("ggplot2")
}
library(ggplot2)

if (!require(tidyverse)) {
  install.packages("tidyverse")
```

```
}
library(tidyverse)
```

Constants

```
BASE_PATH <- "../data/processed"
TRAIN_VAR_SEL_PATH <- paste(BASE_PATH, "train_var_sel.csv", sep = "/")
TEST_VAR_SEL_PATH <- paste(BASE_PATH, "test_var_sel.csv", sep = "/")
FORECAST_NAIVE_PATH <- paste(BASE_PATH, "forecast_naive.csv", sep= "/")
FORECAST_ARIMA_PATH <- paste(BASE_PATH, "forecast_ARIMA.csv", sep= "/")
FORECAST_XGBOOST_VAR_SEL_PATH <- paste(BASE_PATH, "forecast_XGBoost_var_sel.csv", sep= "/")
FORECAST_XGBOOST_PCA_PATH <- paste(BASE_PATH, "forecast_XGBoost_pca.csv", sep= "/")
INDUSTRY_SECTOR_PATH <- paste(BASE_PATH, "industry_sector.csv", sep= "/")
```

5.1. Preparations

First of all, we will load the test data and define the functions to compute the evaluation metrics. ### 5.1.1 Load test data

```
df_test <- read_csv(TEST_VAR_SEL_PATH, show_col_types = FALSE)
head(df_test)
```

```
## # A tibble: 6 x 25
##   index company.sales company.rest  year log.borrowings_repayable_lt_~1 net_debt
##   <dbl> <chr>         <chr>        <dbl>                          <dbl>    <dbl>
## 1    15 3M COMPANY    3M            2016                           14.5  8716000
## 2    15 3M COMPANY    3M            2016                           14.5  8716000
## 3    15 3M COMPANY    3M            2016                           14.5  8716000
## 4    15 3M COMPANY    3M            2016                           14.5  8716000
## 5    16 3M COMPANY    3M            2017                           13.8  8869000
## 6    16 3M COMPANY    3M            2017                           13.8  8869000
## # i abbreviated name: 1: log.borrowings_repayable_lt_1_year
## # i 19 more variables: log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>,
## #   cash_earnings_per_share <dbl>, log.cost_of_sales <dbl>,
## #   log.dividends_per_share <dbl>, exceptional_items <dbl>, ...
```

5.1.2 Create evaluation data frame

```
df_eval <- data.frame()
```

5.1.3 Define function to calculate metrics and evaluate models

The functions below calculate the RMSE, the MAE and the MASE. All of those metrics will be calculated for the log transformed sales variable and the sales variable on the original scale. To back-transform the forecasts to the original scale, we only need an exponential function and the constant we added during the data preparation to make the sales variable non-negative. Although the RMSE and MAE are the more well-known evaluation metrics, we will use the MASE as our main measure. The MASE is a metric used to evaluate the accuracy of a forecast model by measuring the relative performance of a forecasting method by comparing the mean absolute forecast errors to the mean absolute errors of a naive forecast. The MASE is scale-independent, and therefore suitable to compare the forecast performance across our different company time series' with varying scales.

```r
calculate_metrics <- function(actual, forecast) {
  root_mean_squared_error <- rmse(actual, forecast)
  mean_absolute_error <- mae(actual, forecast)
  mean_absolute_scaled_error <-
    mase(actual, forecast, step_size = 4) # step_size = 4 due to quarterly data
  return(
    list(
      root_mean_squared_error = root_mean_squared_error,
      mean_absolute_error = mean_absolute_error,
      mean_absolute_scaled_error = mean_absolute_scaled_error
    )
  )
}

evaluate_model <-
  function(df_forecast, model, log_constant = 393001) {
    df_eval <- data.frame()
    # LOG SCALE
    # evaluate each company separately
    for (company in unique(df_forecast$company.sales)) {
      actual <-
        df_test[df_test$company.sales == company, ]$log.interim_sales
      forecast <-
        df_forecast[df_forecast$company.sales == company, ]$log.interim_sales
      metrics <- calculate_metrics(actual, forecast)
      df_eval <- rbind(
        df_eval,
        data.frame(
          company = company,
          model = model,
          scale = "log",
          rmse = metrics$root_mean_squared_error,
          mae = metrics$mean_absolute_error,
          mase = metrics$mean_absolute_scaled_error
        )
      )
    }

    # evaluate all companies
    actual <-
      df_test$log.interim_sales
    forecast <-
      df_forecast$log.interim_sales
    metrics <- calculate_metrics(actual, forecast)
    df_eval <- rbind(
      df_eval,
      data.frame(
        company = "ALL COMPANIES (AVERAGE)",
        model = model,
        scale = "log",
        rmse = metrics$root_mean_squared_error,
        mae = metrics$mean_absolute_error,
        mase = metrics$mean_absolute_scaled_error
```

```
      )
    )

    # ORIGINAL SCALE
    # evaluate each company separately
    for (company in unique(df_forecast$company.sales)) {
      actual <-
        exp(
          df_test[df_test$company.sales == company, ]$log.interim_sales) - log_constant
      forecast <-
        exp(
          df_forecast[df_forecast$company.sales == company, ]$log.interim_sales) - log_constant
      metrics <- calculate_metrics(actual, forecast)
      df_eval <- rbind(
        df_eval,
        data.frame(
          company = company,
          model = model,
          scale = "original",
          rmse = metrics$root_mean_squared_error,
          mae = metrics$mean_absolute_error,
          mase = metrics$mean_absolute_scaled_error
        )
      )
    }

    # evaluate all companies
    actual <-
      exp(df_test$log.interim_sales) - log_constant
    forecast <-
      exp(df_forecast$log.interim_sales) - log_constant
    metrics <- calculate_metrics(actual, forecast)
    df_eval <- rbind(
      df_eval,
      data.frame(
        company = "ALL COMPANIES (AVERAGE)",
        model = model,
        scale = "original",
        rmse = metrics$root_mean_squared_error,
        mae = metrics$mean_absolute_error,
        mase = metrics$mean_absolute_scaled_error
      )
    )
    return(df_eval)
  }
```

5.2. Naive Forecasting

5.2.1 Load data

```
df_forecast_naive <- read_csv(FORECAST_NAIVE_PATH, show_col_types = FALSE)
head(df_forecast_naive)
```

```
## # A tibble: 6 x 5
```

```
##   company.sales company.rest  year quarter log.interim_sales
##   <chr>         <chr>        <dbl>   <dbl>             <dbl>
## 1 3M COMPANY    3M            2016       1              15.9
## 2 3M COMPANY    3M            2016       2              15.9
## 3 3M COMPANY    3M            2016       3              15.9
## 4 3M COMPANY    3M            2016       4              15.9
## 5 3M COMPANY    3M            2017       1              15.9
## 6 3M COMPANY    3M            2017       2              15.9
```

5.2.2 Calculate metrics

```
df_eval <- rbind(df_eval, evaluate_model(df_forecast_naive, "naive forecast"))
head(df_eval[df_eval$model == "naive forecast",])
```

```
##               company          model scale      rmse       mae     mase
## 1          3M COMPANY naive forecast   log 0.1174372 0.1081874 1.985765
## 2 ABBOTT LABORATORIES naive forecast   log 0.3501985 0.3064275 2.406333
## 3         ABIOMED INC naive forecast   log 0.1627865 0.1478234 2.528438
## 4       ACCENTURE PLC naive forecast   log 0.2686279 0.2343274 2.324026
## 5  ADVANCE AUTO PARTS naive forecast   log 0.1746275 0.1412984 4.308032
## 6      ADVANCED MICRO naive forecast   log 0.6133469 0.4970228 1.748827
```

5.3. ARIMA Forecasting

5.3.1 Load data

```
df_forecast_arima <- read_csv(FORECAST_ARIMA_PATH, show_col_types = FALSE)
head(df_forecast_arima)
```

```
## # A tibble: 6 x 5
##   company.sales company.rest  year quarter log.interim_sales
##   <chr>         <chr>        <dbl>   <dbl>             <dbl>
## 1 3M COMPANY    3M            2016       1              15.9
## 2 3M COMPANY    3M            2016       2              15.9
## 3 3M COMPANY    3M            2016       3              15.9
## 4 3M COMPANY    3M            2016       4              15.9
## 5 3M COMPANY    3M            2017       1              15.9
## 6 3M COMPANY    3M            2017       2              15.9
```

5.3.2 Calculate metrics

```
df_eval <- rbind(df_eval, evaluate_model(df_forecast_arima, "ARIMA"))
head(df_eval[df_eval$model == "ARIMA",])
```

```
##                 company model scale       rmse        mae      mase
## 629          3M COMPANY ARIMA   log 0.04918051 0.03462164 0.6354755
## 630 ABBOTT LABORATORIES ARIMA   log 0.35157785 0.30784530 2.4174668
## 631         ABIOMED INC ARIMA   log 0.03791927 0.02937408 0.5024275
## 632       ACCENTURE PLC ARIMA   log 0.07360570 0.05873401 0.5825155
## 633  ADVANCE AUTO PARTS ARIMA   log 0.13167011 0.11483227 3.5011080
## 634      ADVANCED MICRO ARIMA   log 0.53897669 0.41418003 1.4573362
```

5.4. XGBoost Forecasting with variable selection

5.4.1 Load data

```r
df_forecast_xgb_var_sel <-
  read_csv(FORECAST_XGBOOST_VAR_SEL_PATH, show_col_types = FALSE)
head(df_forecast_xgb_var_sel)
```

```
## # A tibble: 6 x 24
##   company.sales company.rest  year log.borrowings_repayable_lt_1_year net_debt
##   <chr>         <chr>        <dbl>                              <dbl>    <dbl>
## 1 3M COMPANY    3M            2016                               14.5  8716000
## 2 3M COMPANY    3M            2016                               14.5  8716000
## 3 3M COMPANY    3M            2016                               14.5  8716000
## 4 3M COMPANY    3M            2016                               14.5  8716000
## 5 3M COMPANY    3M            2017                               13.8  8869000
## 6 3M COMPANY    3M            2017                               13.8  8869000
## # i 19 more variables: log.ordinary_share_capital <dbl>,
## #   log.total_cash_and_equivalent <dbl>, total_deferred_and_future_tax <dbl>,
## #   log.total_intangibles <dbl>, total_investmnts_exassoc <dbl>,
## #   log.total_stock_and_wip <dbl>, log.trade_debtors <dbl>,
## #   cash_earnings_per_share <dbl>, log.cost_of_sales <dbl>,
## #   log.dividends_per_share <dbl>, exceptional_items <dbl>,
## #   extraord_items_after_tax <dbl>, log.interest_capitalsed <dbl>, ...
```

5.4.2 Calculate metrics

```r
df_eval <-
  rbind(df_eval,
        evaluate_model(df_forecast_xgb_var_sel, "XGBoost variable selection"))
head(df_eval[df_eval$model == "XGBoost variable selection", ])
```

```
##                  company                      model scale       rmse        mae
## 1257          3M COMPANY XGBoost variable selection   log 0.07994920 0.07048407
## 1258 ABBOTT LABORATORIES XGBoost variable selection   log 0.28551339 0.23116164
## 1259         ABIOMED INC XGBoost variable selection   log 0.07097918 0.05795017
## 1260       ACCENTURE PLC XGBoost variable selection   log 0.18017486 0.16492088
## 1261  ADVANCE AUTO PARTS XGBoost variable selection   log 0.15496398 0.12244045
## 1262      ADVANCED MICRO XGBoost variable selection   log 0.36780244 0.27610457
##           mase
## 1257 1.2937257
## 1258 1.8152805
## 1259 0.9912057
## 1260 1.6356615
## 1261 3.7330728
## 1262 0.9715031
```

5.5. XGBoost Forecasting with PCA

5.5.1 Load data

```r
df_forecast_xgb_pca <-
  read_csv(FORECAST_XGBOOST_PCA_PATH, show_col_types = FALSE)
head(df_forecast_xgb_pca)
```

```
## # A tibble: 6 x 25
##   company.sales company.rest  year   PC1   PC2    PC3   PC4    PC5    PC6   PC7
##   <chr>         <chr>        <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl> <dbl>
## 1 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 2 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 3 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 4 3M COMPANY    3M            2016 -4.33  1.56 -0.830  1.88 -0.745 -0.879 0.554
## 5 3M COMPANY    3M            2017 -4.42  1.48 -0.866  1.98 -0.690 -0.782 0.623
## 6 3M COMPANY    3M            2017 -4.42  1.48 -0.866  1.98 -0.690 -0.782 0.623
## # i 15 more variables: PC8 <dbl>, PC9 <dbl>, PC10 <dbl>, PC11 <dbl>,
## #   PC12 <dbl>, PC13 <dbl>, PC14 <dbl>, PC15 <dbl>, PC16 <dbl>, PC17 <dbl>,
## #   PC18 <dbl>, PC19 <dbl>, PC20 <dbl>, quarter <dbl>, log.interim_sales <dbl>
```

5.5.2 Calculate metrics

```r
df_eval <-
  rbind(df_eval, evaluate_model(df_forecast_xgb_pca, "XGBoost PCA"))
head(df_eval[df_eval$model == "XGBoost PCA", ])
```

```
##                  company       model scale      rmse        mae     mase
## 1885          3M COMPANY XGBoost PCA   log 0.1517501 0.11078243 2.033397
## 1886 ABBOTT LABORATORIES XGBoost PCA   log 0.1797071 0.14456506 1.135249
## 1887         ABIOMED INC XGBoost PCA   log 0.1014499 0.07783931 1.331398
## 1888       ACCENTURE PLC XGBoost PCA   log 0.4377152 0.35232729 3.494331
## 1889   ADVANCE AUTO PARTS XGBoost PCA   log 0.2472541 0.21430685 6.533977
## 1890      ADVANCED MICRO XGBoost PCA   log 0.3972973 0.28839075 1.014733
```

```r
df_eval[df_eval$company == "ALL COMPANIES (AVERAGE)" & df_eval$scale == "log",]
```

```
##                      company                      model scale      rmse
## 314  ALL COMPANIES (AVERAGE)             naive forecast   log 0.3171083
## 942  ALL COMPANIES (AVERAGE)                      ARIMA   log 0.2915309
## 1570 ALL COMPANIES (AVERAGE) XGBoost variable selection   log 0.3136309
## 2198 ALL COMPANIES (AVERAGE)                XGBoost PCA   log 0.3517066
##            mae      mase
## 314  0.2030485 0.6077394
## 942  0.1787721 0.5350782
## 1570 0.2260055 0.6764512
## 2198 0.2386485 0.7142929
```

```r
df_eval[df_eval$company == "ALL COMPANIES (AVERAGE)" & df_eval$scale == "original",]
```

```
##                      company                      model    scale    rmse
## 628  ALL COMPANIES (AVERAGE)             naive forecast original 4773788
## 1256 ALL COMPANIES (AVERAGE)                      ARIMA original 4454155
## 1884 ALL COMPANIES (AVERAGE) XGBoost variable selection original 5338406
## 2512 ALL COMPANIES (AVERAGE)                XGBoost PCA original 8071377
##          mae      mase
## 628  1391934 0.5849556
## 1256 1297368 0.5452145
## 1884 1729721 0.7269094
## 2512 2271944 0.9547770
```

5.6. Significance tests

Now we will perform significance tests by using a Wilcoxon signed rank test with two samples. The Wilcoxon signed-rank test is especially useful when we want to compare two related groups, where the data does not meet the assumptions of parametric tests like the paired t-test (e.g., non-normal data or small sample sizes). We will use the MASE metric within our significance tests.

5.6.1 Naive forecast vs. ARIMA

The null hypothesis for this test is that the errors of the naive forecast and the ARIMA model are equally high or higher for the ARIMA model. The alternative hypothesis is that the errors of the naive forecast are higher than the errors of the ARIMA model.

```
naive_mase <- df_eval[df_eval$model == "naive forecast" &
          df_eval$scale == "log" &
          df_eval$company != "ALL COMPANIES (AVERAGE)", "mase"]
arima_mase <- df_eval[df_eval$model == "ARIMA" &
          df_eval$scale == "log" &
          df_eval$company != "ALL COMPANIES (AVERAGE)", "mase"]

wilcox.test(naive_mase,
          arima_mase,
          alternative = "greater"#,
          #paired = TRUE
          )
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  naive_mase and arima_mase
## W = 59012, p-value = 4.667e-06
## alternative hypothesis: true location shift is greater than 0
```

By looking at the p-value of 9.748e-06 displayed above, we can definitely reject the null hypothesis at the 5% significance level. This means that the errors of the naive forecast are significantly higher than the errors produced from ARIMA.

5.6.2 Naive Forecast vs. XGBoost with variable selection

The null hypothesis for this test is that the errors of the naive forecast and the XGBoost model with variable selection are equally high or higher for the XGBoost model with variable selection. The alternative hypothesis is that the errors of the naive forecast is higher than the errors of the XGBoost model with variable selection.

```
xgb_var_sel_mase <-
  df_eval[df_eval$model == "XGBoost variable selection" &
            df_eval$scale == "log" &
            df_eval$company != "ALL COMPANIES (AVERAGE)", "mase"]

wilcox.test(naive_mase,
            xgb_var_sel_mase,
            alternative = "greater",
            paired = TRUE)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  naive_mase and xgb_var_sel_mase
```

```
## V = 18142, p-value = 1
## alternative hypothesis: true location shift is greater than 0
```

The p-value of 0.9999 displayed above clearly shows that we cannot reject the null hypothesis. This means that the errors of the XGBoost model with variable selection data are not lower than the errors of the naive forecasts and that the forecasting of XGBoost with variable selection does not outperform the naive forecasting.

5.6.3 Naive Forecast vs. XGBoost with PCA

Similar as for the test above, the null hypothesis is that that the errors of the naive forecast and the XGBoost model with PCA data are equally high or higher for the XGBoost model with PCA. The alternative hypothesis is that the errors of the naive forecast is higher than the errors of the XGBoost model with PCA.

```
xgb_pca_mase <- df_eval[df_eval$model == "XGBoost PCA" &
          df_eval$scale == "log" &
          df_eval$company != "ALL COMPANIES (AVERAGE)", "mase"]

wilcox.test(naive_mase,
          xgb_pca_mase,
          alternative = "greater",
          paired = TRUE
          )
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  naive_mase and xgb_pca_mase
## V = 17820, p-value = 1
## alternative hypothesis: true location shift is greater than 0
```

Similarly to the significance test of the naive forecasts vs. the XGBoost forecasts with variable selection data, we observe a high p-value of 1, which means that we cannot reject the null hypothesis and the XGBoost model with PCA data does not outperform the naive forecasting.

5.6.4 ARIMA vs. XGBoost with variable selection

Now we will test if XGBoost with variable selection outperforms the ARIMA forecasts (even though the previous tests already showed that XGBoost is not able to outperform the naive forecasting). The null hypothesis is that the errors of ARIMA are equal or lower than the errors of XGBoost with variable selection. The alternative hypothesis is that the errors of ARIMA are greater than the errors of XGBoost with variable selection.

```
wilcox.test(arima_mase,
            xgb_var_sel_mase,
            alternative = "greater",
            paired = TRUE)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  arima_mase and xgb_var_sel_mase
## V = 13841, p-value = 1
## alternative hypothesis: true location shift is greater than 0
```

As expected, the p-value of 1 clearly shows that XGBoost with variable selection is not able to outperform ARIMA in terms of the MASEs.

5.6.5 ARIMA vs. XGBoost with PCA

With this test, we will examine if the XGBoost model using PCA data is able to outperform the ARIMA forecasts. The null hypothesis is that that the errors of ARIMA are equal or lower than the errors of XGBoost with PCA data. The alternative hypothesis is that the errors of ARIMA are greater than the errors of XGBoost with PCA.

```
wilcox.test(arima_mase,
            xgb_pca_mase,
            alternative = "greater",
            paired = TRUE)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  arima_mase and xgb_pca_mase
## V = 13370, p-value = 1
## alternative hypothesis: true location shift is greater than 0
```

As expected, we can also observe a p-value of 1 for this significance test. Which means that we can definitely not reject the null hypothesis and XGBoost with PCA data does not outperform the ARIMA forecasts.

5.7. Industry sector analysis

Finally, we will make use of the industry sector data we obtained from WikiData by using OpenRefine. In this section, we will join this data with the evaluation data and examine if the performance of models differs between industry sectors.

5.7.1 Load industry sector data

First of all, we will load the data.

```
df_industry_sector <- read_csv(INDUSTRY_SECTOR_PATH, show_col_types = FALSE)
head(df_industry_sector)
```

```
## # A tibble: 6 x 6
##   Column company.sales       industry1           industry2 industry3 company.rest
##    <dbl> <chr>               <chr>               <chr>     <chr>     <chr>
## 1      1 3M                  conglomerate        occupati~ final go~ 3M
## 2      2 Abbott Laboratories medical device in~ <NA>      <NA>      ABBOTT LABO~
## 3      3 AbioMed             <NA>                <NA>      <NA>      ABIOMED
## 4      4 Accenture           management consul~ outsourc~ informat~ ACCENTURE C~
## 5      5 Advance Auto Parts  <NA>                <NA>      <NA>      ADV.AUTO PA~
## 6      6 AMD                 electrical indust~ semicond~ computer~ ADVANCED MI~
```

```
colSums(is.na.data.frame(df_industry_sector)) # check missing values
```

```
##        Column company.sales     industry1     industry2     industry3
##             0             0            50           249           285
##  company.rest
##             0
```

In the output above, we can see that the column industry1 has the least missing values of the three collected columns. Therefore we will only use this column for our analysis.

5.7.2 Join industry data with evaluation data frame

Now we will join it with the test data first, because we need the column company.rest to join the evaluation data frame and the industry data frame, since the linkage in OpenRefine modified the company.sales column. Followed by that, we will join this intermediate data frame with the evaluation data frame.

```
df_eval_industry <-
  merge(
    merge(df_test, df_industry_sector, by = "company.rest") %>%  # join test and industry sector data
      mutate(company = company.sales.x) %>% # rename to company
      dplyr::select(company, industry1) %>% # select only relevant columns
      distinct(), # drop duplicates
    df_eval, # join with evaluation data frame
    by = "company"
  )

head(df_eval_industry)
```

```
##      company    industry1                        model    scale         rmse
## 1 3M COMPANY conglomerate                  XGBoost PCA original 1.540537e+06
## 2 3M COMPANY conglomerate XGBoost variable selection      log 7.994920e-02
## 3 3M COMPANY conglomerate                        ARIMA      log 4.918051e-02
## 4 3M COMPANY conglomerate                  XGBoost PCA      log 1.517501e-01
## 5 3M COMPANY conglomerate               naive forecast      log 1.174372e-01
## 6 3M COMPANY conglomerate               naive forecast original 9.751802e+05
##            mae      mase
## 1 1.051187e+06 2.2703825
## 2 7.048407e-02 1.2937257
## 3 3.462164e-02 0.6354755
## 4 1.107824e-01 2.0333966
## 5 1.081874e-01 1.9857653
## 6 8.911500e+05 1.9247300
```

5.7.3 Select industries with >= 5 companies

In this step, we will group the data by industry and model and mean aggregate the MASEs. Furthermore, we count the companies that fall in each particular industry sector and model and only keep industries where at least five companies are present. Additionally, we will create a new column that stores the industry sector together with the number of companies that are present for each sector which we will use for plotting.

```
df_eval_industry_viz <-
  df_eval_industry %>%
  filter(scale == "log") %>% # use log scale
  group_by(industry1, model) %>% # group by industry and model
  summarise(mean_mase = mean(mase), n = n()) %>% # aggregate
  drop_na() %>%  # drop NAs
  filter(n >= 5) %>% # filter companies that are present > 5 times
  ungroup() %>%
  arrange(industry1) %>% # sort by industry
  mutate(industry_n = paste0(industry1, " (", n, ")")) # construct column with occurrence
```

```
## `summarise()` has grouped output by 'industry1'. You can override using the
## `.groups` argument.
```

```
head(df_eval_industry_viz)
```

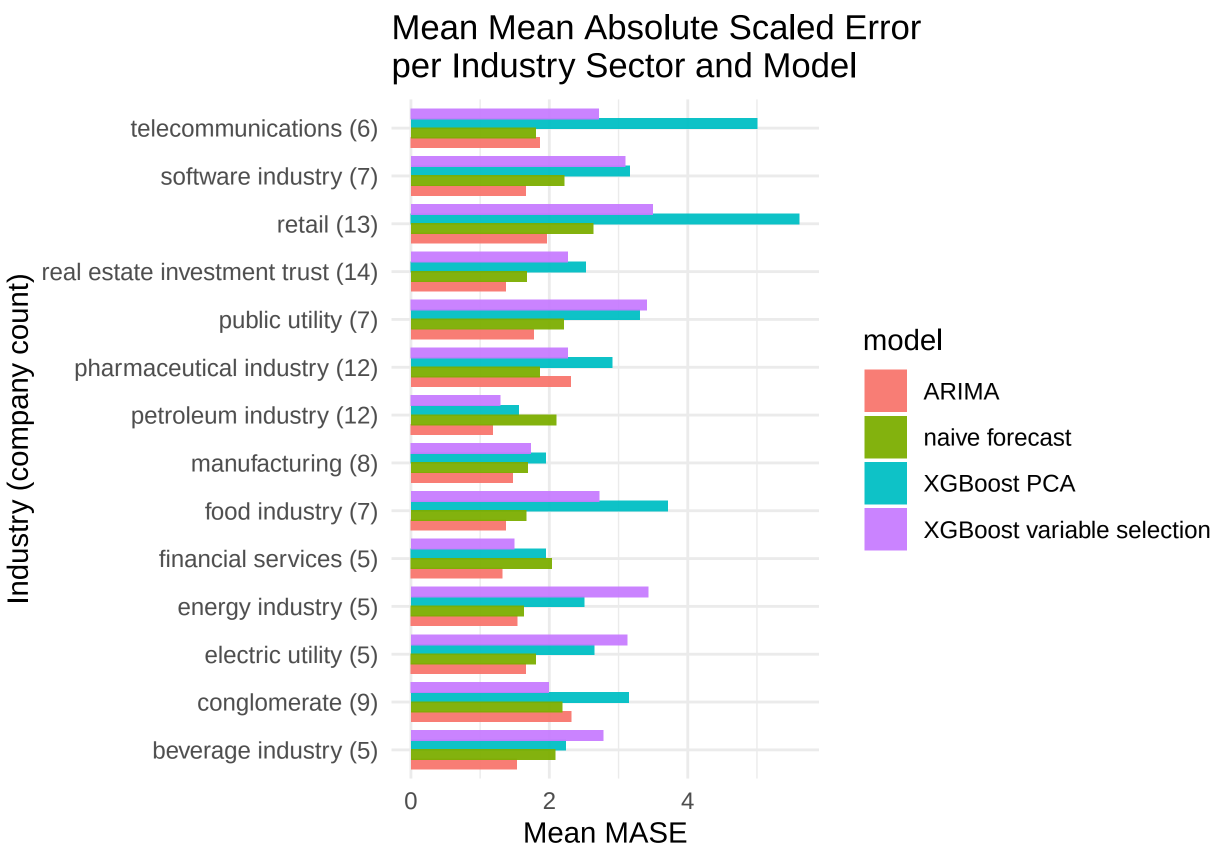
```
## # A tibble: 6 x 5
```

```
##   industry1         model                          mean_mase     n industry_n
##   <chr>             <chr>                              <dbl> <int> <chr>
## 1 beverage industry ARIMA                               1.53     5 beverage industr~
## 2 beverage industry XGBoost PCA                         2.24     5 beverage industr~
## 3 beverage industry XGBoost variable selection          2.78     5 beverage industr~
## 4 beverage industry naive forecast                      2.09     5 beverage industr~
## 5 conglomerate      ARIMA                               2.32     9 conglomerate (9)
## 6 conglomerate      XGBoost PCA                         3.15     9 conglomerate (9)
```

5.7.4 Plot mean MASE per industry sector and model

Finally, we plot the industry sectors vs. the mean aggregated MASE for each model variant.

```
ggplot(data=df_eval_industry_viz, aes(x=mean_mase, y=industry_n, fill=model)) +
  geom_col(position=position_dodge(width = 0.8)) +
labs(x = "Mean MASE", y = "Industry (company count)") +
  ggtitle("Mean Mean Absolute Scaled Error \nper Industry Sector and Model") +
  theme_minimal()
```



In the plot above, we can see that the performance of the models differs between the industry sectors. For the telecommunications industry and the pharmaceutical industry, for example, all models have higher errors than the naive forecast. For the petroleum industry and financial services, on the other hand, we can observe that all models - even the XGBoost variants - outperform the naive forecast. It is also interesting to see, that for some industry sectors, the XGBoost with PCA data produces lower errors than XGBoost using the variable selection data. All of those findings indicate, that it can be beneficial to add the industry sector information to the training data set or to train separate models for each industry sector.