

FGV MBA: Business Analytics & Big Data

- 1 Predictive analysis
 - 1.1 Professor
 - 1.2 Students / ID (matricula)
 - 1.3 Where to find the source code of this project?
- 2 Setting the scene
 - 2.1 Introduction
 - 2.2 Task description
 - 2.3 Data description
- 3 About the Czech bank database
- 4 Using a step by step approach
- 5 Create Functions (step 1)
 - 5.1 Specific data ingestion functions
 - 5.1.1 GetGenderFromBirthnumber
 - 5.1.2 GetBirthdateFromBirthnumber
 - 5.1.3 ConvertToDate
 - 5.1.4 GetAgeFromBirthnumber
 - 5.2 Metrics auxiliary functions
 - 5.2.1 calculateModelMetrics
 - 5.2.2 modelMetrics
 - 5.3 Data preparation functions
 - 5.4 Plot auxiliary functions
 - 5.4.1 Score_Histograms
 - 5.4.2 Score_Boxplot
 - 5.4.3 KS_Plot
 - 5.4.4 Plot_ROC
 - 5.4.5 accuracy
- 6 Data Ingestion (step 2)
- 7 Data Cleaning (step 3)
- 8 Label Translation (step 4)
- 9 Data Enhancement (step 5)
- 10 Data Preparation for Predictive Modeling (step 6)
 - 10.1 Selecting the target dataset
 - 10.2 Splitting dataset into Train and Test data
- 11 Gender Exploration
- 12 Loan Exploration
- 13 Account Balance Exploration
- 14 District exploration
- 15 Objective
- 16 Modeling
 - 16.1 Dataset preparation
 - 16.2 Variable selection
 - 16.2.1 Dummy variables
 - 16.2.2 Transaction type proportion variables
 - 16.2.3 Multicollinearity on feature variables
 - 16.3 Looking for outliers
 - 16.4 Sample split into Test and Training Data
 - 16.5 Fit the Logistic Regression model
- 17 Interpreting model output
- 18 Evaluating the model performance
 - 18.1 Getting Performance Measures
 - 18.2 Evaluating classification performance
- 19 Objective

- 20 Modeling
 - 20.1 Dataset preparation
 - 20.2 Variable selection
 - 20.3 Sample split into Test and Training Data
 - 20.4 Fiting the Decision Tree model
 - 20.5 Evaluating necessity of pruning
- 21 Interpreting model output
- 22 Evaluating the model performance
 - 22.1 Getting performance measures
 - 22.2 Evaluating classification performance
- 23 Objective
- 24 Modeling
 - 24.1 Dataset preparation
 - 24.2 Variable selection
 - 24.3 Sample split into Test and Training Data
 - 24.4 Fiting the Boosting model
- 25 Interpreting model output
- 26 Evaluating the model performance
 - 26.1 Getting Performance Measures
 - 26.2 Evaluating classification performance
- 27 Objective
- 28 Modeling
 - 28.1 Dataset preparation
 - 28.2 Variable selection
 - 28.3 Sample split into Test and Training Data
- 29 Selecting the best parameters values for the Random Forest
- 30 Interpreting model output
- 31 Evaluating the model performance
 - 31.1 Getting Performance Measures
 - 31.2 Evaluating classification performance
- 32 Objective
- 33 Model evaluation
 - 33.1 Getting the predicted score from each model
 - 33.2 Getting performance measures for each model.
 - 33.3 Evaluating performance of each model
 - 33.3.1 Density Plots
 - 33.3.2 Score Boxplots
 - 33.3.3 KS Plots
 - 33.3.4 ROC Curve
 - 33.3.5 Acurracy
- 34 Final considerations and project limitations
- 35 References used in this class assignment.
 - 35.1 Theory
 - 35.1.1 Books and articles
 - 35.1.2 Wikipedia
 - 35.2 R Programing
 - 35.3 Awesome functions from the community

1 Predictive analysis

In the predictive analysis discipline we aim to develop statistical models, based on data, for a particular outcome of interest. From this model, we make sure that behavioral learning and past experiences have a model with good generalization.

In this project, we developed a set of models using a real anonymized Czech bank transactions, account info, and loan records released for *PKDD'99 Discovery Challenge* (<https://raw.githubusercontent.com/ldaniel/Predictive-Analytics/master/enunciation/PKDD'99%C2%A0Discovery%C2%A0Challenge.pdf>)





This website intends to present the work analysis for the “*Análise Preditiva*” class assignment (https://raw.githubusercontent.com/ldaniel/Predictive-Analytics/master/enunciation/projeto_final.pdf).

Use the menu above to navigate and see the final report.

1.1 Professor

- João Rafael Dias (magister.analytica@gmail.com (<mailto:magister.analytica@gmail.com>))

1.2 Students / ID (matrícula)

-  (<https://github.com/DanielFCampos>) Daniel Campos (<mailto:daniel.ferraz.campos@gmail.com>) / A57635769
-  (<https://github.com/ldaniel>) Leandro Daniel (<mailto:contato@leandrodaniel.com>) / A57622988
-  (<https://github.com/RodriGonca>) Rodrigo Goncalves (<mailto:rodrigo.goncalves@me.com>) / A57566093
-  (<https://github.com/ygorlima1>) Ygor Lima (mailto:ygor_redesocial@hotmail.com) / A57549661

1.3 Where to find the source code of this project?

This project can be found and downloaded on GitHub: <https://github.com/ldaniel/Predictive-Analytics> (<https://github.com/ldaniel/Predictive-Analytics>)

Valar Morghulis! :)

2 Setting the scene

2.1 Introduction

Onde upon a time, there was a bank offering services to private persons. The services include managing of accounts, offerings loans, etc.

2.2 Task description

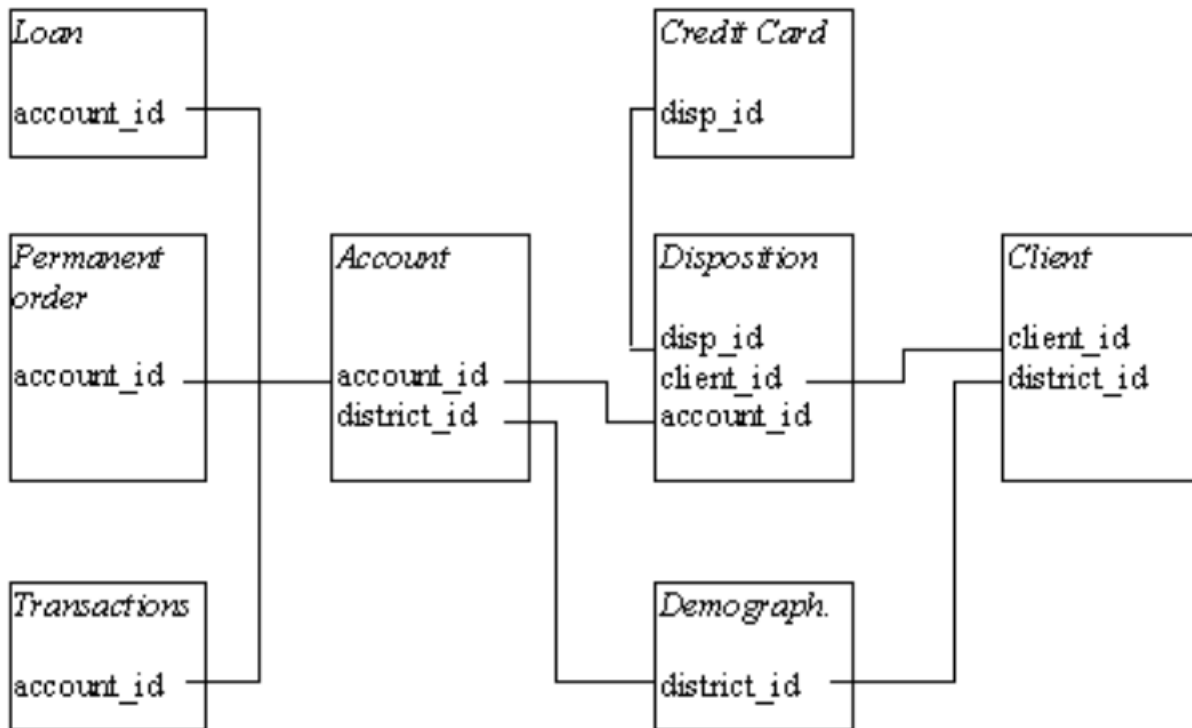
The bank wants to improve their services. For instance, the bank managers have only vague idea, who is a good client (whom to offer some additional services) and who is a bad client (whom to watch carefully to minimize the bank losses).

Fortunately, the bank stores data about their clients, the accounts (transactions within several months), the loans already granted, the credit cards issued.

The bank managers hope to improve their understanding of customers and seed specific actions to improve services. A mere application of discovery tool will not be convincing for them.

2.3 Data description

This database was prepared by Petr Berka and Marta Sochorova.



Simplified logical data model of Berka Bank.

3 About the Czech bank database

Data from a real Czech bank. From 1999.

The data about the clients and their accounts consist of following relations:

- relation **account** (4500 objects in the file ACCOUNT.ASC) - each record describes static characteristics of an account,
- relation **client** (5369 objects in the file CLIENT.ASC) - each record describes characteristics of a client,
- relation **disposition** (5369 objects in the file DISP.ASC) - each record relates together a client with an account i.e. this relation describes the rights of clients to operate accounts,
- relation **permanent order** (6471 objects in the file ORDER.ASC) - each record describes characteristics of a payment order,
- relation **transaction** (1056320 objects in the file TRANS.ASC) - each record describes one transaction on an account,
- relation **loan** (682 objects in the file LOAN.ASC) - each record describes a loan granted for a given account,
- relation **credit card** (892 objects in the file CARD.ASC) - each record describes a credit card issued to an account,
- relation **demographic data** (77 objects in the file DISTRICT.ASC) - each record describes demographic characteristics of a district.

Each account has both static characteristics (e.g. date of creation, address of the branch) given in relation “account” and dynamic characteristics (e.g. payments debited or credited, balances) given in relations “permanent order” and “transaction”. Relation “client” describes characteristics of persons who can manipulate with the accounts. One client can have more accounts, more clients can manipulate with single account; clients and accounts are related together in relation “disposition”. Relations “loan” and “credit card” describe some services which the bank offers to its clients; more credit cards can be issued to an account, at most one loan can be granted for an account. Relation “demographic data” gives some publicly available information about the districts (e.g. the unemployment rate); additional information about the clients can be deduced from this.

Source: *This database was prepared by Petr Berka and Marta Sochorova.* (<https://raw.githubusercontent.com/ldaniel/Predictive-Analytics/master/enunciation/PKDD'99%C2%A0Discovery%C2%A0Challenge.pdf>)

4 Using a step by step approach

Before starting the Berka Analysis, a few important steps were taken in order to prepare the source data files. These steps are listed below:

- **Step 01:** Create Functions;
- **Step 02:** Data Ingestion;
- **Step 03:** Data Cleaning;
- **Step 04:** Label Translation;
- **Step 05:** Data Enhancement;
- **Step 06:** Dataset Preparation.

5 Create Functions (step 1)

This step create functions to be used in the next steps. Following, all functions created are described.

5.1 Specific data ingestion functions

5.1.1 GetGenderFromBirthnumber

The birth_number column is given in the form of YYMMDD for men, and YYMM+50DD for women. The objective of this function is to return the gender of the client via the birth_number.

```
GetGenderFromBirthnumber <- function(var_birth_number) {
  month <- substr(var_birth_number, 3, 4)
  result <- ifelse(as.integer(month) > 50, "female", "male")

  return(as.factor(result))
}
```

5.1.2 GetBirthdateFromBirthnumber

The birth_number column is given in the form of YYMMDD for men, # and YYMM+50DD for women. The objective of this function is to return the final birthday as Date.

```
GetBirthdateFromBirthnumber <- function(var_birth_number, var_gender) {
  year <- paste("19", substr(var_birth_number, 1, 2), sep="")
  month <- ifelse(var_gender == "male", substr(var_birth_number, 3, 4),
    as.integer(substr(var_birth_number, 3, 4)) - 50)
  day <- substr(var_birth_number, 5, 6)
  result <- as.Date(paste(year, "-", month, "-", day, sep=""), format = "%Y-%m-%d")

  return(result)
}
```

5.1.3 ConvertToDate

The objective of this function is to convert the strange bank date style to the regular R Date datatype.

```
ConvertToDate <- function(var_date) {
  year <- paste("19", substr(var_date, 1, 2), sep="")
  month <- substr(var_date, 3, 4)
  day <- substr(var_date, 5, 6)
  result <- as.Date(paste(year, "-", month, "-", day, sep=""), format = "%Y-%m-%d")

  return(result)
}
```

5.1.4 GetAgeFromBirthnumber

The objective of this function is to get age given the birth_number.

```

GetAgeFromBirthnumber <- function(var_birth_number) {
  base_year <- 99 # considering 1999 as the base year for this exercise
  year <- substr(var_birth_number, 1, 2)
  result <- base_year - as.integer(year)

  return(result)
}

```

5.2 Metrics auxiliary functions

5.2.1 calculateModelMetrics

The objective of this function is to calculate main metrics of model performance according to a cutoff value.

```

calculateModelMetrics <- function(cutData, realData, predData){
  cuttedData <- as.factor(ifelse(predData>=cutData, 1, 0))

  invisible(capture.output(out <- CrossTable(realData, cuttedData,
                                              prop.c = F, prop.t = F, prop.r = T, prop.chisq = F)))

  out <- as.data.frame(out) %>%
    mutate(merged=paste0(t.x, t.y)) %>%
    dplyr::select(merged, val=t.Freq)

  TN <- filter(out, merged == "00")$val[1]
  FP <- filter(out, merged == "01")$val[1]
  FN <- filter(out, merged == "10")$val[1]
  TP <- filter(out, merged == "11")$val[1]

  return(data.frame(Cut = cutData,
                    TN = TN,
                    FP = FP,
                    FN = FN,
                    TP = TP,
                    TPR = TP/(TP+FN), TNR=TN/(TN+FP),
                    Error = (FP+FN)/(TP+TN+FP+FN),
                    Precision = TP/(TP+FP),
                    F1 = 2*(TP/(TP+FN))*(TP/(TP+FP))/((TP/(TP+FP)) + (TP/(TP+FN))))))
}

```

5.2.2 modelMetrics

The objective of this function is to calculate main metrics of model performance for cutoffs from 0-1 based on given step.

```

modelMetrics <- function(realData, predData, stepping = 0.01,
                          plot_title = "TPR/TNR by cutoff over full dataset"){
  probCuts <- seq(from = 0, to = 1, by = stepping)
  out <- bind_rows(lapply(probCuts, calculateModelMetrics, realData = realData, predData = predData))
  out <- out[complete.cases(out),] %>% mutate(Difference = abs(TPR-TNR))

  best <- out %>% arrange(Difference) %>% head(1) %>% dplyr::select(-Difference)

  p <- plot_ly(x = ~out$Cut, y = ~out$Difference, name = 'Abs. Diff.', type = 'bar', opacity = 0.3) %>%
    add_trace(x = ~out$Cut, y = ~out$TPR, name = 'TPR', type = 'scatter', mode = 'lines', opacity = 1) %>%
    add_trace(x = ~out$Cut, y = ~out$TNR, name = 'TNR', type = 'scatter', mode = 'lines', opacity = 1) %>%
    layout(xaxis = list(title = "Cutoff Value"),
           yaxis = list(title = "True Ratio (%)")) %>%
    add_annotations(
      text = sprintf("<b>%s</b>", plot_title),
      x = 0,
      y = 1.04,
      yref = "paper",
      xref = "paper",
      xanchor = "left",
      yanchor = "top",
      showarrow = FALSE,
      font = list(size = 15)
    ) %>%
    add_annotations(
      text = sprintf("<b>%s</b>", best$Cut),
      x = best$Cut,
      y = best$TPR,
      showarrow = FALSE,
      bgcolor = "white",
      opacity = 0.8
    )

  return(list(TableResults = out,
              BestCut = best,
              Plot = p))
}

```

5.3 Data preparation functions

5.3.0.1 SplitTestTrainDataset

See topic “Splitting dataset into Train and Test data” for further details.

5.4 Plot auxiliary functions

Functions used in the evaluation step to compare the models.

5.4.1 Score_Histograms

Function used to plot the score density plots of the model.

Needs to receive a dataset containing the predicted and actual values, the actual values vector the score (predicted) values value and a custom title.

```
Score_Histograms <- function(dataset, predicted, actual, title) {
  ggplot(data = dataset) +
    geom_density(aes(x = predicted, fill = as.factor(actual)),
      alpha = 0.5) +
    scale_fill_manual(values = c("0" = "#16a085", "1" = "#e74c3c")) +
    scale_x_continuous(limits = c(0, 1)) +
    theme_economist() +
    labs(title = title,
      y = 'Score',
      fill = 'Defaulter |1 = True|') +
    theme(panel.grid = element_blank(),
      axis.ticks.y = element_blank(),
      axis.text.y = element_blank(),
      legend.position = 0,
      plot.title = element_text(hjust = 0.5))
}
```

5.4.2 Score_Boxplot

Function used to plot the score box plot of the model.

Needs to receive a dataset containing the predicted and actual values, the actual values vector the score (predicted) values value and a custom title.

```
Score_Boxplot <- function(dataset, predicted, actual, title) {
  ggplot(data = dataset) +
    geom_boxplot(aes(y = predicted,
      fill = as.factor(actual))) +
    coord_flip() +
    scale_fill_manual(values = c("0" = "#16a085", "1" = "#e74c3c")) +
    scale_y_continuous(limits = c(0, 1)) +
    theme_economist() +
    labs(title = title,
      y = 'Score',
      fill = 'Defaulter |1 = True|') +
    theme(panel.grid = element_blank(),
      axis.ticks.y = element_blank(),
      axis.text.y = element_blank(),
      legend.position = 0,
      plot.title = element_text(hjust = 0.5))
}
```

5.4.3 KS_Plot

Function used to plot the cumulative probability distribution and KS metric of the model.

Needs to receive a vector with scores of Defaulters and a vector f scores of Non-Defaulters and a custom title.


```

KS_Plot <- function(zeros, ones, title) {
  group <- c(rep("Non Defaulters", length(zeros)), rep("Defauters", length(ones)))
  dat <- data.frame(KSD = c(zeros, ones), group = group)
  cdf1 <- ecdf(zeros)
  cdf2 <- ecdf(ones)
  minMax <- seq(min(zeros, ones), max(zeros, ones), length.out=length(zeros))
  x0 <- minMax[which( abs(cdf1(minMax) - cdf2(minMax)) ==
                     max(abs(cdf1(minMax) - cdf2(minMax))) )][1]
  y0 <- cdf1(x0)[1]
  y1 <- cdf2(x0)[1]
  ks <- round(y0 - y1, 2)

  ggplot(dat, aes(x = KSD, group = group, color = group))+
    stat_ecdf(size=1) +
    geom_segment(aes(x = x0[1], y = y0[1], xend = x0[1], yend = y1[1]),
                 linetype = "dashed", color = "blue") +
    geom_point(aes(x = x0[1], y = y0[1]), color="blue", size=4) +
    geom_point(aes(x = x0[1], y = y1[1]), color="blue", size=4) +
    geom_label(aes(x = x0[1], y = y1[1] + (y0[1] - y1[1]) / 2, label = ks),
               color = 'black') +
    scale_x_continuous(limits = c(0, 1)) +
    labs(title = title,
         y = 'Cumulative Probability Distribution',
         x = 'Score') +
    theme_economist() +
    theme(legend.title = element_blank(),
          panel.grid = element_blank(),
          legend.position = 0,
          plot.title = element_text(hjust = 0.5))
}

```

5.4.4 Plot_ROC

Function used to plot the combined ROC curves of each model.

Needs to receive a dataset with actual and predicted scores of each model.

```

Plot_ROC <- function(dataset, smooth_opt = FALSE) {
  roc_logistic      <- roc(logistic.actual ~ logistic.predicted,
                           dataset,
                           smooth = smooth_opt,
                           quiet = TRUE)

  roc_decision.tree <- roc(decision.tree.actual ~ decision.tree.predicted,
                           dataset,
                           smooth = smooth_opt,
                           quiet = TRUE)

  roc_boosting      <- roc(boosting.actual ~ boosting.predicted,
                           dataset,
                           smooth = smooth_opt,
                           quiet = TRUE)

  roc_random.forest <- roc(random.forest.actual ~ random.forest.predicted,
                           dataset,
                           smooth = smooth_opt,
                           quiet = TRUE)

  p <- ggplot() +
    geom_line(aes(x = 1 - roc_logistic$specificities,
                  y = roc_logistic$sensitivities,
                  colour = 'Logistic Regression'), # red
              size = 1,
              linetype = 1,
              alpha = 0.7) +
    geom_line(aes(x = 1 - roc_decision.tree$specificities,
                  y = roc_decision.tree$sensitivities,
                  colour = 'Decision Tree'), # blue
              size = 1,
              linetype = 1,
              alpha = 0.7) +
    geom_line(aes(x = 1 - roc_boosting$specificities,
                  y = roc_boosting$sensitivities,
                  colour = 'Boosting'), # green
              size = 1,
              linetype = 1,
              alpha = 0.7) +
    geom_line(aes(x = 1 - roc_random.forest$specificities,
                  y = roc_random.forest$sensitivities,
                  colour = 'Random Forest'), # purple
              size = 2,
              linetype = 1,
              alpha = 1) +
    geom_abline(aes(intercept = 0, slope = 1),
                linetype = 2,
                size = 1) +
    scale_colour_manual(name = NULL,
                        breaks = c('Logistic Regression',
                                   'Decision Tree',
                                   'Boosting',
                                   'Random Forest'),
                        labels = c('Logistic Regression',
                                   'Decision Tree',
                                   'Boosting',
                                   'Random Forest'),
                        values = c('#C0392B',
                                   '#3498DB',
                                   '#2ECC71',
                                   '#9B59B6'))

```

```

        '#28B463',
        '#9B59B6')) +
  labs(y = 'True Positive Rate',
       x = 'False Positive Rate',
       title = 'Receiver Operating Characteristic Curve - ROC',
       subtitle = 'Random Forest and Boosting are the models that best discriminate Defaulters and Non-Defaulters') +
  theme_economist() +
  theme(panel.grid = element_blank())

return (p)
}

```

5.4.5 accuracy

Function used to output confusion matrix and basic accuracy metrics of each model.

Needs to receive a vector of actual and predicted values.

```

accuracy <- function(score, actual, threshold = 0.5) {

  fitted.results <- ifelse(score > threshold ,1 ,0)

  misClassificError <- mean(fitted.results != actual)

  misClassCount <- misclassCounts(fitted.results, actual)

  print(kable(misClassCount$conf.matrix))

  print('-----')
  print(paste('Model General Accuracy of: ',
              round((1 - misClassCount$metrics['ER']) * 100, 2), '%',
              sep = ''))
  print(paste('True Positive Rate of      : ',
              round(misClassCount$metrics['TPR'] * 100, 2), '%',
              sep = ''))

}

```

6 Data Ingestion (step 2)

The process of data ingestion — preparing data for analysis — usually includes steps called extract (taking the data from its current location), transform (cleansing and normalizing the data), and load (placing the data in a database where it can be analyzed).

During this step, in addition to the loading data processes, it was performed data casting, column renaming and small touch-ups. The list below describe each table adjustment taken:

- **District:** renaming columns and casting columns with decimal or “?” values;
- **Credit Card:** casting column issued in creditcard table from string to datetime data type;
- **Account:** casting column date in account table from string to datetime data type;
- **Loan:** casting columns in table loan to the right data types;
- **Permanent Order:** casting columns with decimal values;
- **Transaction:** casting columns in table transaction to the right data types.

7 Data Cleaning (step 3)

The objective of this step is analysing missing values and other strange conditions. In order to accomplish this task, a few R functions were used to quickly discover missing values, like NA and empty fields.

First thing done, was fixing observations in k_symbol transaction table with ' ' (one space) to empty string (''), using the following command.

```
transaction$k_symbol = trimws(transaction$k_symbol)
```

Then, the command below was used to find out any NA values in each table.

```
sapply(TableName, function(x) sum(is.na(x)))
```

Solely the **transaction** table has NA values, in the following columns:

	x
trans_id	0
account_id	0
date	0
type	0
operation	0
amount	0
balance	0
k_symbol	0
bank	0
account	760931

Finally, the following command was used in each table to find out where empty values was hidden.

```
sapply(TableName, function(x) table(as.character(x) == "")["TRUE"])
```

Again, only the **transaction** table had empty values, according to the table below:

	x
trans_id.NA	NA
account_id.NA	NA
date.NA	NA
type.NA	NA
operation.TRUE	183114
amount.NA	NA
balance.NA	NA
k_symbol.TRUE	535314
bank.TRUE	782812
account.NA	NA

For the exploration analysis report, we did not take any additional action, since the missing values was not relevant.

8 Label Translation (step 4)

In order to make the data information more understandable, it was translated some relevant labels and domains from Czech to English.

```
# Translating relevant labels and domains to english -----

disposition$type <- plyr::mapvalues(disposition$type, c('OWNER', 'DISPONENT'),
                                   c('Owner', 'User'))

account$frequency <- plyr::mapvalues(account$frequency,
                                     c('POPLATEK MESICNE', 'POPLATEK TYDNE',
                                       'POPLATEK PO OBRATU'),
                                     c('Monthly', 'Weekly', 'On Transaction'))

permanent_order$k_symbol <- plyr::mapvalues(permanent_order$k_symbol,
                                             c('POJISTNE', 'SIPO', 'LEASING', 'UVER'),
                                             c('insurance payment', 'household',
                                               'leasing', 'loan payment'))

transaction$type <- plyr::mapvalues(transaction$type,
                                    c('PRIJEM', 'VYDAJ', 'VYBER'),
                                    c('credit', 'withdrawal', 'withdrawal in cash'))

transaction$operation <- plyr::mapvalues(transaction$operation,
                                          c('VYBER KARTOU', 'VKLAD', 'PREVOD Z UCTU',
                                            'VYBER', 'PREVOD NA UCET'),
                                          c('credit card withdrawal', 'credit in cash',
                                            'collection from another bank',
                                            'withdrawal in cash', 'remittance to nother bank'))

transaction$k_symbol <- plyr::mapvalues(transaction$k_symbol,
                                         c('POJISTNE', 'SLUZBY', 'UROK', 'SANKC. UROK',
                                           'SIPO', 'DUCHOD', 'UVER'),
                                         c('insurance payment', 'statement',
                                           'interest credited', 'sanction interest',
                                           'household', 'old age pension', 'loan payment'))
```

9 Data Enhancement (step 5)

This step aims to improve the analysis by adding auxiliary information. Data enhancement is all about making sure any data that is coming into the business is being looked at with a critical eye and is being filtered down to maximize its value.

The code below get gender, birthday and age from birth_number column using *GetGenderFromBirthnumber* and *GetBirthdateFromBirthnumber* functions.

```
client <- client %>%
  mutate(gender = GetGenderFromBirthnumber(birth_number)) %>%
  mutate(birth_date = GetBirthdateFromBirthnumber(birth_number, gender)) %>%
  mutate(age = GetAgeFromBirthnumber(birth_number))
```

The code below improved loan data by having a classification regarding its payment status.

```
loan <- mutate(loan, defaulter =
  as.logical(plyr::mapvalues(status, c('A','B','C','D'),
                              c(FALSE,TRUE,FALSE,TRUE))),
  contract_status = plyr::mapvalues(status, c('A','B','C','D'),
                                     c('finished','finished','running','running')),
  type = 'Owner')
```

The code below improved client data by having its age group.

```
client <- mutate(client, age_bin = paste(findInterval(age,
  c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)) * 10, '+'))
```

The code below calculate an additional table with current and average account balance for each account.

```
account_balance <- arrange(transaction, desc(date), account_id) %>%
  group_by(account_id) %>%
  mutate(avg_balance = mean(balance)) %>%
  filter(row_number() == 1) %>%
  dplyr::select(account_id, date, balance, avg_balance)

colnames(account_balance) <- c("account_id", "last_transaction_date", 'account_balance', 'avg_balance')
```

The code below calculate an additional table with the proportion of each transaction type (k_symbol) on total transaction amount of each account. That data will be used to fit various different predictive models.

```
account_transaction_pattern <- select(transaction, c(trans_id, account_id, date, amount, k_symbol)) %>%
  mutate(k_symbol = ifelse(k_symbol == '' | is.na(k_symbol), 'other', k_symbol)) %>%
  spread(key = k_symbol, value = amount) %>%
  replace(is.na(.), 0) %>%
  mutate(amount = rowSums(.[4:11])) %>%
  group_by(account_id) %>%
  summarise(transaction_count = n(),
    last_transaction_date = max(date),
    amount = sum(amount),
    prop_household = sum(household) / amount,
    prop_insurance_payment = sum(`insurance payment`) / amount,
    prop_interest_credited = sum(`interest credited`) / amount,
    prop_loan_payment = sum(`loan payment`) / amount,
    prop_old_age_pension = sum(`old age pension`) / amount,
    prop_other = sum(`other`) / amount,
    prop_sanction_interest = sum(`sanction interest`) / amount,
    prop_statement = sum(`statement`) / amount)
```

10 Data Preparation for Predictive Modeling (step 6)

10.1 Selecting the target dataset

The below function was created to be used in the modeling exercises to be performed, the idea is to have a standard way to get the prepared data set already prepared with correct data types and dummies.

```

# dataset preparation -----

# The objective of this step is to return a DataFrame to be used in predictive
# modeling. Therefore, it will join loan, client, district, creditcard,
# account_balance, account_balance_pattern. Finally, it will rename the variables
# and create the appropriate dummy variables to be used in the modeling process.

# joining datasets
source_dataset <- left_join(loan, disposition, by = c('account_id', 'type')) %>%
  left_join(client, by = 'client_id') %>%
  left_join(district, by = 'district_id') %>%
  left_join(creditcard, by = 'disp_id') %>%
  left_join(account_balance, by = 'account_id') %>%
  left_join(account_transaction_pattern, by = 'account_id') %>%
  mutate(card_age_month = (issued %--%
    make_date(1998, 12, 31)) / months(1),
    last_transaction_age_days = ((last_transaction_date.y %--%
    make_date(1998, 12, 31)) / days(1))) %>%
dplyr::select(c("amount.x", "duration", "payments", "status", "defaulter",
  "contract_status", "gender", "age", "district_name",
  "region", "no_of_inhabitants",
  "no_of_municip_inhabitants_less_499",
  "no_of_municip_500_to_1999", "no_of_municip_2000_to_9999",
  "no_of_municip_greater_10000", "no_of_cities",
  "ratio_of_urban_inhabitants",
  "average_salary", "unemployment_rate_1995",
  "unemployment_rate_1996",
  "no_of_entrepreneurs_per_1000_inhabitants",
  "no_of_committed_crimes_1995",
  "no_of_committed_crimes_1996", "type.y",
  "card_age_month", "account_balance",
  "avg_balance", "transaction_count", "amount.y",
  "last_transaction_age_days", "prop_old_age_pension",
  "prop_insurance_payment",
  "prop_sanction_interest", "prop_household",
  "prop_statement", "prop_interest_credited",
  "prop_loan_payment", "prop_other"))

# renaming variables
colnames(source_dataset) <- c("x_loan_amount", "x_loan_duration", "x_loan_payments",
  "x_loan_status", "y_loan_defaulter", "x_loan_contract_status",
  "x_client_gender", "x_client_age",
  "x_district_name", "x_region",
  "x_no_of_inhabitants", "x_no_of_municip_inhabitants_less_499",
  "x_no_of_municip_500_to_1999", "x_no_of_municip_2000_to_9999",
  "x_no_of_municip_greater_10000", "x_no_of_cities",
  "x_ratio_of_urban_inhabitants",
  "x_average_salary", "x_unemployment_rate_1995",
  "x_unemployment_rate_1996",
  "x_no_of_entrepreneurs_per_1000_inhabitants",
  "x_no_of_committed_crimes_1995",
  "x_no_of_committed_crimes_1996", "x_card_type",
  "x_card_age_month", "x_account_balance",
  "x_avg_account_balance", "x_transaction_count",
  "x_transaction_amount", "x_last_transaction_age_days",
  "x_prop_old_age_pension", "x_prop_insurance_payment",
  "x_prop_sanction_interest", "x_prop_household", "x_prop_statement",
  "x_prop_interest_credited", "x_prop_loan_payment", "x_prop_other")

# excluding redundant variables

```

```

source_dataset <- dplyr::select(source_dataset, -c("x_loan_status", "x_loan_contract_status",
                                                'x_prop_sanction_interest'))

# coercing variable domains and data types
source_dataset$x_card_type = ifelse(is.na(source_dataset$x_card_type), 'no card',
                                    as.character(source_dataset$x_card_type))

source_dataset$x_card_age_month = ifelse(is.na(source_dataset$x_card_age_month), 0,
                                         source_dataset$x_card_age_month)

source_dataset$y_loan_defaulter = as.integer(source_dataset$y_loan_defaulter)

# creating dummies
source_dataset <- fastDummies::dummy_cols(source_dataset,
                                           remove_first_dummy = TRUE,
                                           select_columns = c("x_client_gender", "x_district_name",
                                                             "x_region", "x_card_type"))

source_dataset <- dplyr::select(source_dataset, -c("x_client_gender", "x_district_name", "x_region",
                                                  "x_card_type"))

# reordering variables
source_dataset <- source_dataset[ , order(names(source_dataset))]

source_dataset <- dplyr::select(source_dataset, y_loan_defaulter, everything())

# excluding non desirable characters in variable names
colnames(source_dataset) <- stringr::str_replace_all(names(source_dataset), ' ', '_')
colnames(source_dataset) <- stringr::str_replace_all(names(source_dataset), '_-', '_')
colnames(source_dataset) <- trimws(names(source_dataset))

```

10.2 Splitting dataset into Train and Test data

The below function was created to be used in the modeling exercises to be split the source_dataset into train and test datasets.


```

# SplitTestTrainDataset -----
# The objective of this function is to split a given dataset
# in train and test datasets
SplitTestTrainDataset <- function(dataset) {
  set.seed(12345)

  dataset$y_loan_defaulter <- as.integer(dataset$y_loan_defaulter)

  index <- caret::createDataPartition(dataset$y_loan_defaulter,
                                       p= 0.7, list = FALSE)

  data.train <- dataset[index, ]
  data.test  <- dataset[-index,]

  # checking event proportion in sample and test datasets against full dataset.
  event_proportion <- bind_rows(prop.table(table(dataset$y_loan_defaulter)),
                                prop.table(table(data.train$y_loan_defaulter)),
                                prop.table(table(data.test$y_loan_defaulter)))

  event_proportion$scope = ''
  event_proportion$scope[1] = 'full dataset'
  event_proportion$scope[2] = 'train dataset'
  event_proportion$scope[3] = 'test dataset'

  event_proportion <- select(event_proportion, scope, everything())

  SplitDataset <- list()
  SplitDataset$data.train <- data.train
  SplitDataset$data.test  <- data.test
  SplitDataset$event.proportion <- event_proportion

  return(SplitDataset)
}

```

To make sure all the models uses the same datasets for Train and Testing we are saving the initial sampling to be reused across the models.

This will ensure consistency when comparing the models against each other.

```

# calling function to split and create train and test databases
# this function will split the dataset into train and test data and save the sampling in disk
# to resample just delete './models/source_train_test_dataset.rds' file and rerun this script
if (file.exists('./models/source_train_test_dataset.rds')) {
  source_train_test_dataset <- readRDS('./models/source_train_test_dataset.rds')
} else {
  source_train_test_dataset <- SplitTestTrainDataset(source_dataset)
  saveRDS(source_train_test_dataset, './models/source_train_test_dataset.rds')
}

```

11 Gender Exploration

At first glance, gender equality is well balanced in the bank, even when observed over the decades. Even more impressive, gender equality is everywhere in the country.

```
# gender distribution of clients in the bank
gender_plot <- ggplot(data = client) +
  aes(x = gender, fill = gender) +
  geom_bar() +
  labs(subtitle = "A well balanced bank",
       x = "Gender",
       y = "Total clients") +
  theme_economist()

clientGenderOverDecades <- client %>%
  group_by(decade = as.integer(substr(client$birth_number, 1,1)) * 10,
           gender = client$gender) %>%
  count()

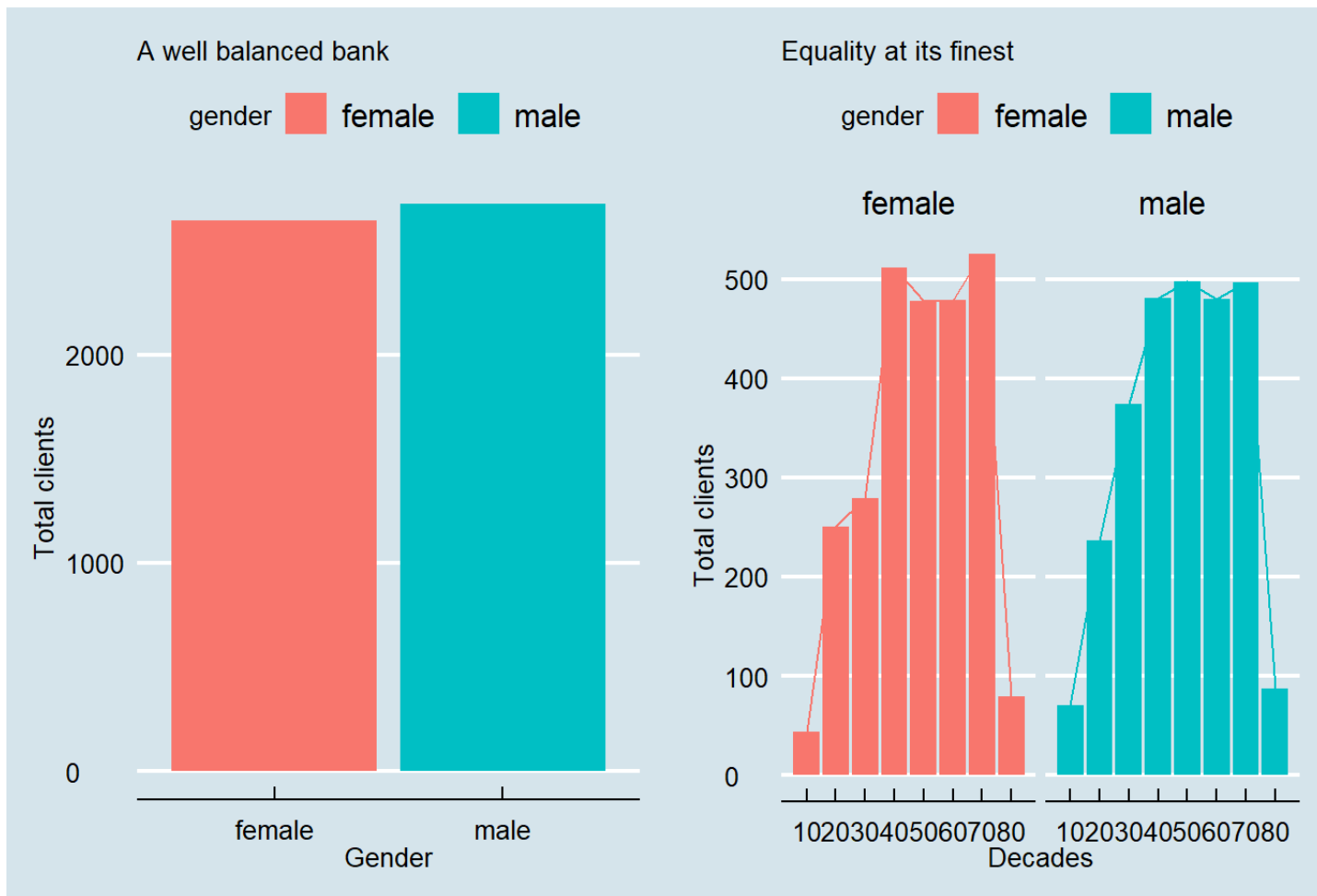
# gender distribution of clients in the bank over the decades
gender_dist_plot <- ggplot(data = clientGenderOverDecades) +
  aes(x = decade, fill = gender, weight = n) +
  scale_x_continuous(breaks = c(0, 10, 20, 30, 40, 50, 60, 70, 80)) +
  geom_bar() +
  geom_line(aes(y = n, color = gender)) +
  labs(subtitle = "Equality at its finest",
       x = "Decades",
       y = "Total clients") +
  theme_economist() +
  facet_wrap(vars(gender))

gender_plots <- ggarrange(gender_plot, gender_dist_plot)

gender_plots <- annotate_figure(gender_plots,
                              top = text_grob("Gender distribution of clients over the decades",
                                              color = "black", face = "bold",
                                              size = 14))

gender_plots
```

Gender distribution of clients over the decades

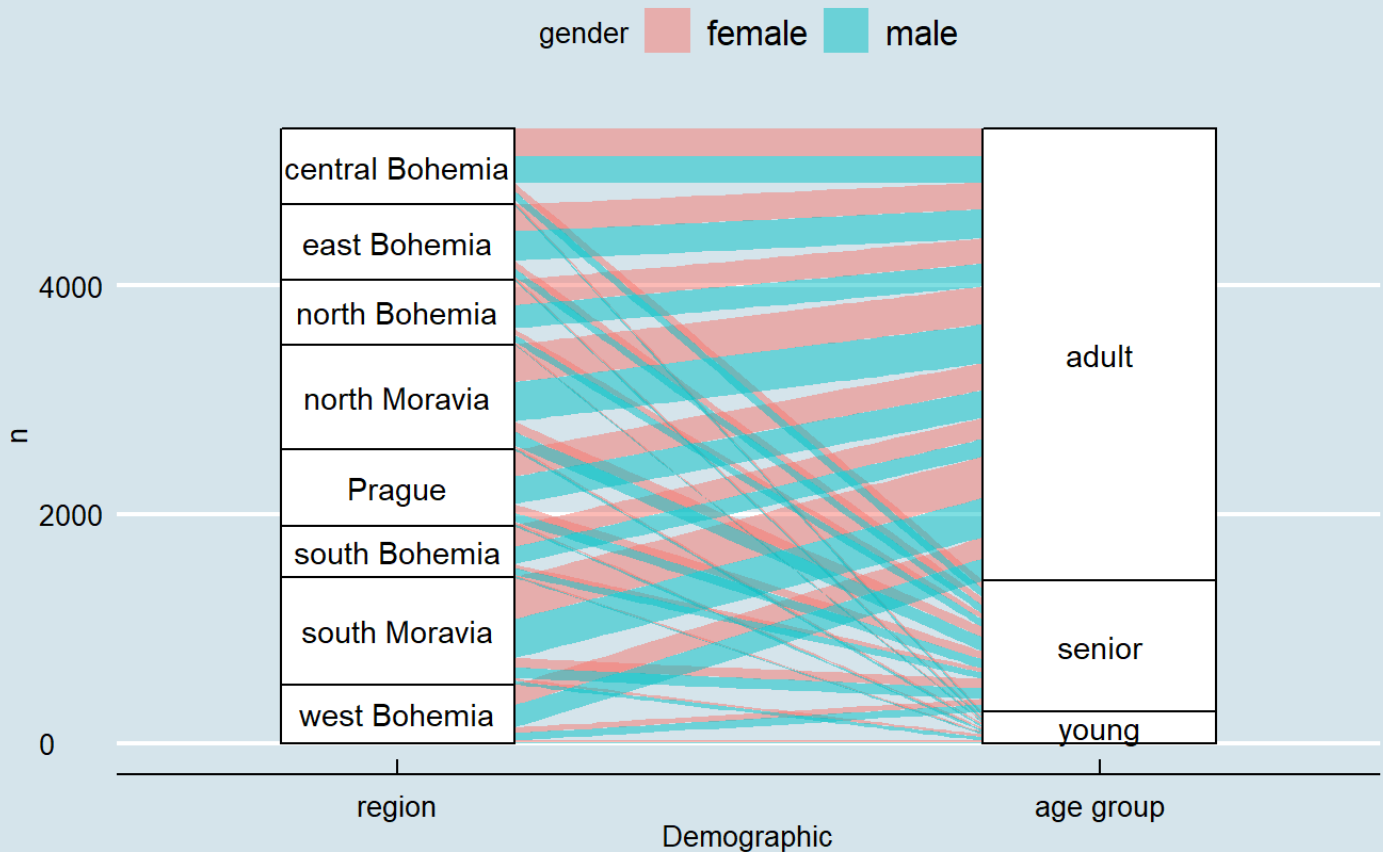


```
# alluvial diagram representation of gender, age group and region
clientGenderAgeGroupByRegion <- client %>%
  mutate(age_group = ifelse(age < 21, "young",
                             ifelse(age >= 21 & age <= 60, "adult", "senior"))) %>%
  inner_join(district, by = "district_id") %>%
  group_by(age_group, gender, region) %>%
  count()

ggplot(data = clientGenderAgeGroupByRegion,
       aes(axis1 = region, axis2 = age_group, y = n)) +
  scale_x_discrete(limits = c("region", "age group"), expand = c(.1, .1)) +
  xlab("Demographic") +
  geom_alluvium(aes(fill = gender), knot.pos = 0) +
  geom_stratum() +
  geom_text(stat = "stratum", label.strata = TRUE) +
  theme_economist() +
  ggtitle("Region and age group by gender", "Equality is everywhere")
```

Region and age group by gender

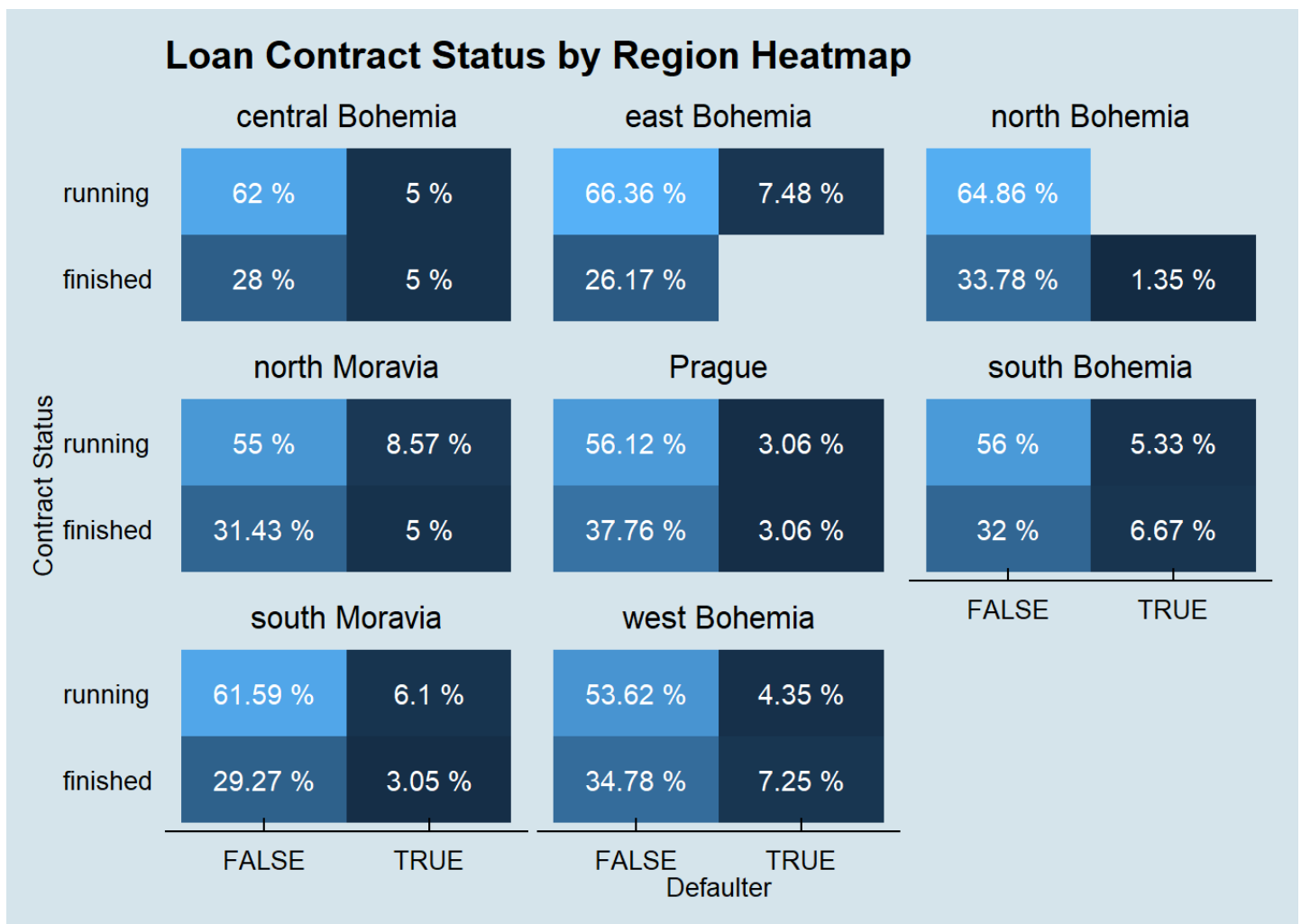
Equality is everywhere



12 Loan Exploration

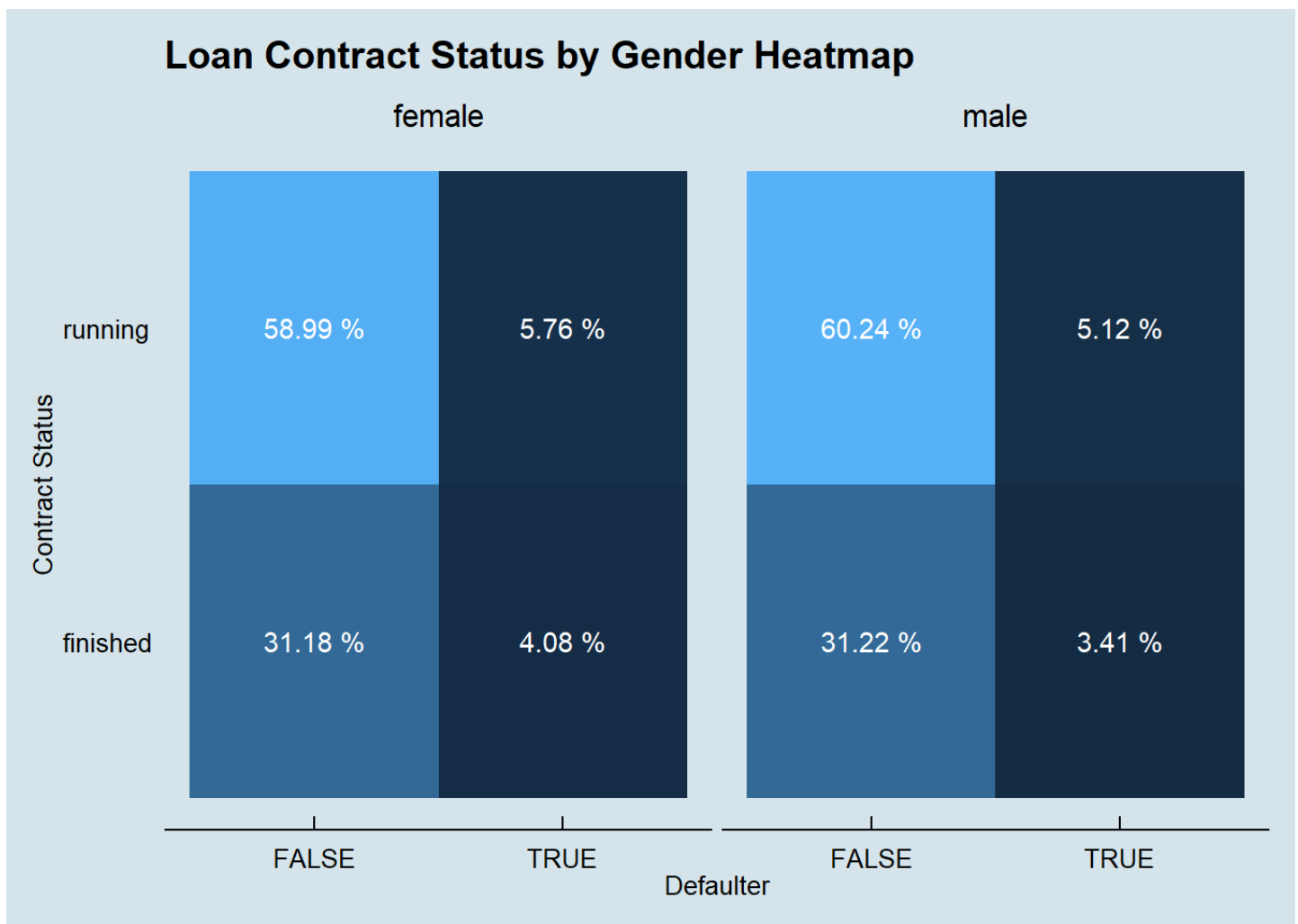
Here we investigate if there is any association between the region and the likelihood of default in the 682 loan observations in the dataset.

```
left_join(loan, disposition, by = 'account_id') %>%
  left_join(client, by = 'client_id') %>%
  left_join(district, by = 'district_id') %>%
  group_by(region, contract_status, defaulter) %>%
  summarise(count = n(),
            amount = sum(amount)) %>%
  group_by(region, contract_status) %>%
  mutate(count_contract_status = sum(count),
         amount_contract_status = sum(amount)) %>%
  group_by(region) %>%
  mutate(count_region = sum(count),
         amount_region = sum(amount)) %>%
  ggplot(aes(x = defaulter, y = contract_status, fill = count / count_region)) +
  geom_bin2d(stat = 'identity') +
  geom_text(aes(label = paste(round(count / count_region * 100, 2), '%')),
            color = 'white') +
  facet_wrap(~region) +
  theme_economist() +
  theme(legend.position = 'none', panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(x = 'Defaulter',
       y = 'Contract Status',
       title = 'Loan Contract Status by Region Heatmap')
```



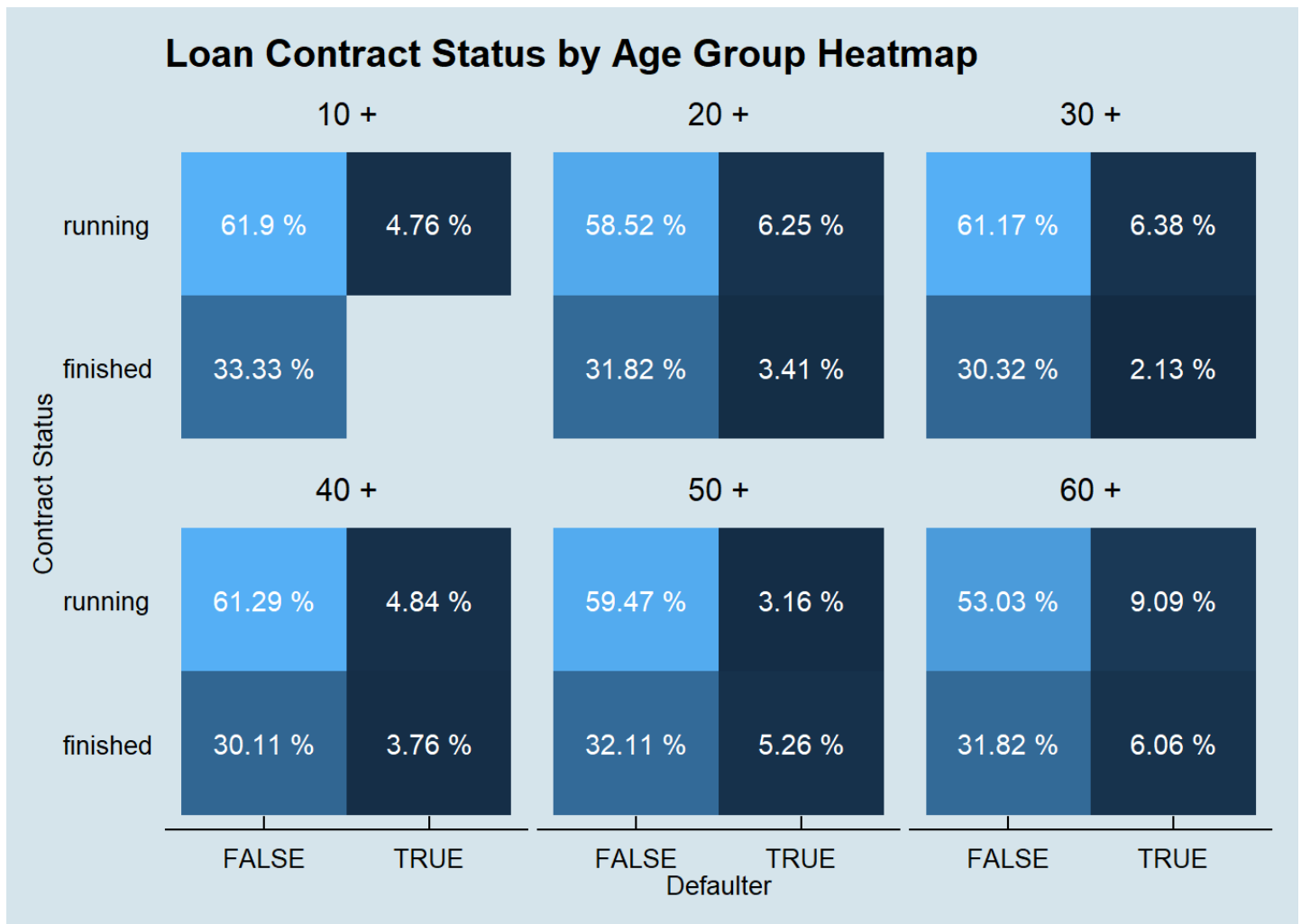
We perform the same investigation on the association between the client gender and the likelihood of default in the 682 loan observations in the dataset.

```
left_join(loan, disposition, by = 'account_id') %>%
  left_join(client, by = 'client_id') %>%
  left_join(district, by = 'district_id') %>%
  group_by(gender, contract_status, defaulter) %>%
  summarise(count = n(),
            amount = sum(amount)) %>%
  group_by(gender, contract_status) %>%
  mutate(count_contract_status = sum(count),
         amount_contract_status = sum(amount)) %>%
  group_by(gender) %>%
  mutate(count_gender = sum(count),
         amount_gender = sum(amount)) %>%
  ggplot(aes(x = defaulter, y = contract_status,
            fill = count / count_gender)) +
  geom_bin2d(stat = 'identity') +
  geom_text(aes(label = paste(round(count / count_gender * 100, 2), '%')),
            color = 'white') +
  facet_wrap(~gender) +
  theme_economist() +
  theme(legend.position = 'none', panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(x = 'Defaulter',
       y = 'Contract Status',
       title = 'Loan Contract Status by Gender Heatmap')
```



We finally do the same investigation on the association between the client age and the likelihood of default in the 682 loan observations in the dataset.

```
left_join(loan, disposition, by = 'account_id') %>%
  left_join(client, by = 'client_id') %>%
  left_join(district, by = 'district_id') %>%
  group_by(age_bin, contract_status, defaulter) %>%
  summarise(count = n(),
            amount = sum(amount)) %>%
  group_by(age_bin, contract_status) %>%
  mutate(count_contract_status = sum(count),
         amount_contract_status = sum(amount)) %>%
  group_by(age_bin) %>%
  mutate(count_age_bin = sum(count),
         amount_age_bin = sum(amount)) %>%
  ggplot(aes(x = defaulter,
            y = contract_status, fill = count / count_age_bin)) +
  geom_bin2d(stat = 'identity') +
  geom_text(aes(label = paste(round(count / count_age_bin * 100, 2), '%')),
            color = 'white') +
  facet_wrap(~age_bin) +
  theme_economist() +
  theme(legend.position = 'none', panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(x = 'Defaulter',
       y = 'Contract Status',
       title = 'Loan Contract Status by Age Group Heatmap')
```



It seems that none of these features, alone, are determinant on the odds of a client default.

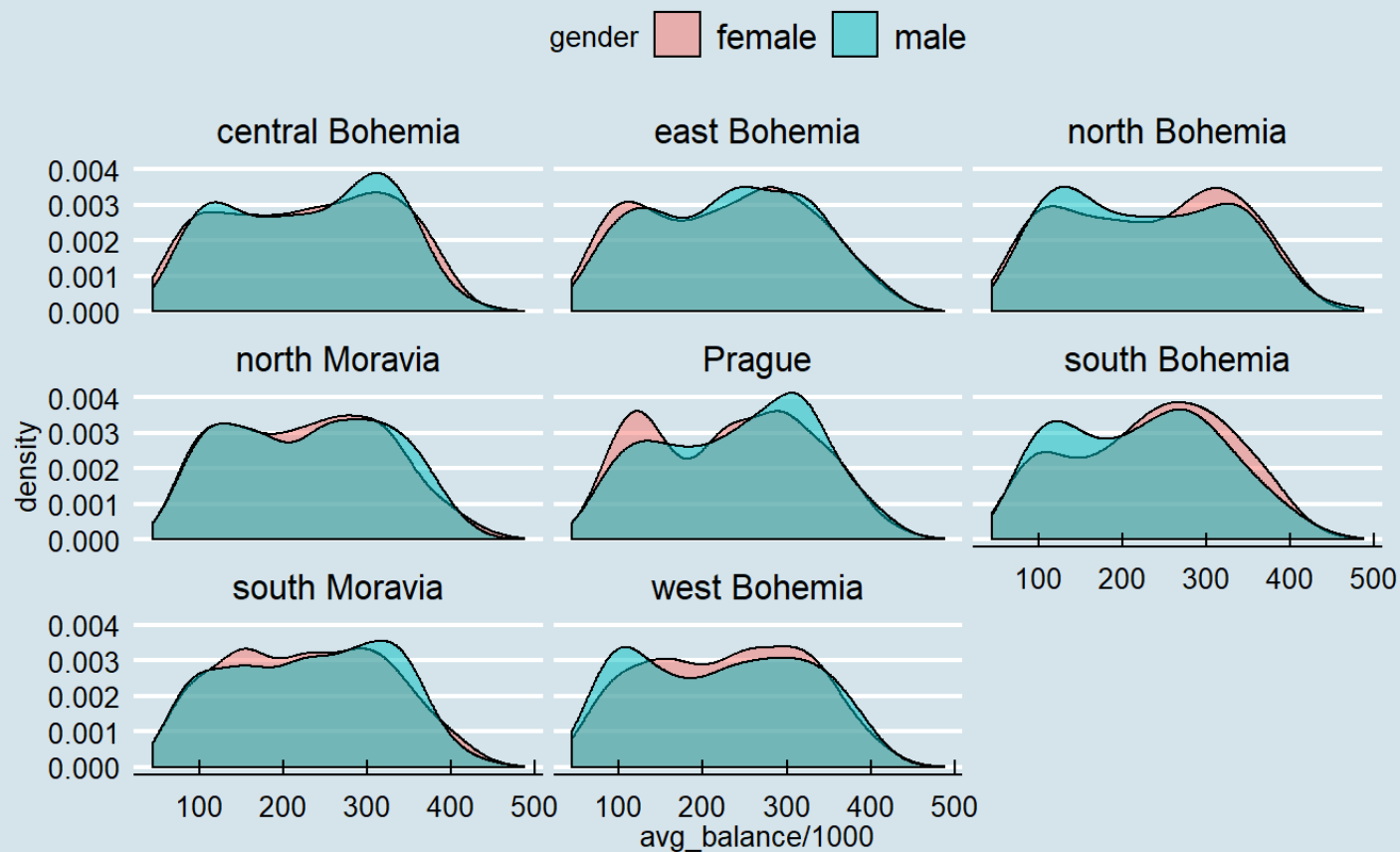
13 Account Balance Exploration

As we do not have the individual income of each client, we tried to perform a proxy of the client's wealth by calculating the average account balance of each account and investigating the distribution by region and age.

```
left_join(account_balance, disposition, by = 'account_id') %>%
  left_join(client, by = 'client_id') %>%
  left_join(district, by = 'district_id') %>%
  filter(type == 'Owner') %>%
  ggplot(aes(avg_balance / 1000)) +
    geom_density(alpha = 0.5, aes(fill = gender)) +
    scale_x_continuous(labels = scales::comma) +
    labs(title = 'Average Account Balance Distribution by Gender and Region',
         subtitle = 'k (Kč) - Koruna (CZK)') +
    theme_economist() +
    facet_wrap(~region)
```

Average Account Balance Distribution by Gender and Region

k (Kc) - Koruna (CZK)



One interest pattern shows up, males in prague region tend to have a bigger average account balance than females, and the opposite trend in South, North and West Bohemia.

14 District exploration

Regarding the Czech regions, we can notice in the exploration below which regions are more likely to have defaulters based on the finished loans operations so far.

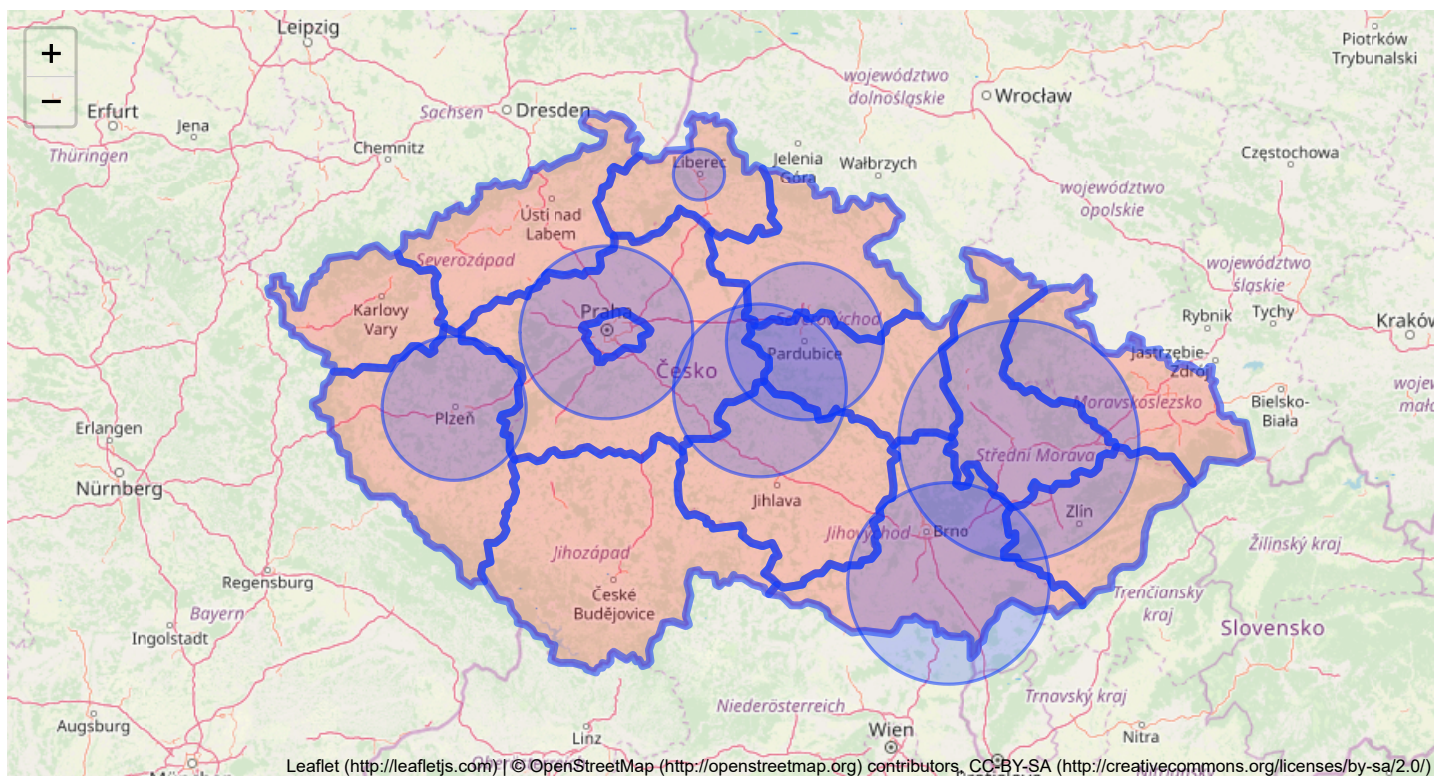

```

loan_amount_by_region <- dplyr::select(loan, account_id, amount, defaulter, contract_status) %>%
  filter(defaulter == TRUE) %>%
  inner_join(account) %>%
  inner_join(district) %>%
  group_by(region) %>%
  summarise(transaction_count = n(),
            amount = sum(amount)) %>%
  inner_join(czech_regions_coords)

jsonMapFile <- "../map/czech-republic-regions.json"
czech_regions <- as.json(geojson_read(jsonMapFile))

leaflet(loan_amount_by_region) %>%
  addTiles() %>%
  setView(lng = 15.3, lat = 49.8, zoom = 7) %>%
  addGeoJSON(czech_regions, fillColor = "red", stroke = "#555555") %>%
  addCircles(lng = ~long,
            lat = ~lat,
            weight = 2,
            radius = ~sqrt(amount) * 30,
            popup = ~region)

```



15 Objective

The goal of this session is trying to fit a Logistic Regression model on Loan data aiming to predict the probability of delinquency for each contract.

16 Modeling

16.1 Dataset preparation

Using the vanilla transaction dataset, we calculated several derived variables for each account as described in the Data Preparation session.

This dataset is joined with Loan, Client, Credit Card, District, Account and Account Balance tables.

We ended up having a data set with **118 variables**.

```
loan_dataset_logistic <- source_dataset

kable(tibble(variables = names(loan_dataset_logistic)))
```

variables

y_loan_defaulter

x_account_balance

x_average_salary

x_avg_account_balance

x_card_age_month

x_card_type_classic

x_card_type_gold

x_card_type_junior

x_client_age

x_client_gender_male

x_district_name_Benesov

x_district_name_Beroun

x_district_name_Blansko

x_district_name_Breclav

x_district_name_Brno_mesto

x_district_name_Brno_venkov

x_district_name_Bruntal

x_district_name_Ceska_Lipa

x_district_name_Ceske_Budejovice

x_district_name_Cesky_Krumlov

x_district_name_Cheb

x_district_name_Chomutov

x_district_name_Chrudim

x_district_name_Decin

x_district_name_Domazlice

x_district_name_Frydek_Mistek

x_district_name_Havlickuv_Brod

x_district_name_Hl.m._Praha

x_district_name_Hodonin

x_district_name_Hradec_Kralove

x_district_name_Jablonec_n._Nisou

x_district_name_Jesenik

variables

x_district_name_Jicin

x_district_name_Jihlava

x_district_name_Jindrichuv_Hradec

x_district_name_Karlovy_Vary

x_district_name_Karvina

x_district_name_Kladno

x_district_name_Klatovy

x_district_name_Kolin

x_district_name_Kromeriz

x_district_name_Kutna_Hora

x_district_name_Liberec

x_district_name_Litomerice

x_district_name_Louny

x_district_name_Melnik

x_district_name_Mlada_Boleslav

x_district_name_Most

x_district_name_Nachod

x_district_name_Novy_Jicin

x_district_name_Nymburk

x_district_name_Olomouc

x_district_name_Opava

x_district_name_Ostrava_mesto

x_district_name_Pardubice

x_district_name_Pelhrimov

x_district_name_Pisek

x_district_name_Plzen_jih

x_district_name_Plzen_mesto

x_district_name_Plzen_sever

x_district_name_Prachatice

x_district_name_Praha_vychod

x_district_name_Praha_zapad

x_district_name_Prerov

x_district_name_Pribram

x_district_name_Prostějov

x_district_name_Rakovnik

x_district_name_Rokycany

x_district_name_Rychnov_nad_Kneznou

variables

x_district_name_Semily

x_district_name_Strakonice

x_district_name_Sumperk

x_district_name_Svitavy

x_district_name_Tabor

x_district_name_Tachov

x_district_name_Teplice

x_district_name_Trebic

x_district_name_Trutnov

x_district_name_Uherske_Hradiste

x_district_name_Usti_nad_Labem

x_district_name_Usti_nad_Orlici

x_district_name_Vsetin

x_district_name_Vyskov

x_district_name_Zdar_nad_Sazavou

x_district_name_Zlin

x_district_name_Znojmo

x_last_transaction_age_days

x_loan_amount

x_loan_duration

x_loan_payments

x_no_of_cities

x_no_of_committed_crimes_1995

x_no_of_committed_crimes_1996

x_no_of_entrepreneurs_per_1000_inhabitants

x_no_of_inhabitants

x_no_of_municip_2000_to_9999

x_no_of_municip_500_to_1999

x_no_of_municip_greater_10000

x_no_of_municip_inhabitants_less_499

x_prop_household

x_prop_insurance_payment

x_prop_interest_credited

x_prop_loan_payment

x_prop_old_age_pension

x_prop_other

x_prop_statement

variables

x_ratio_of_urban_inhabitants

x_region_central_Bohemia

x_region_east_Bohemia

x_region_north_Bohemia

x_region_north_Moravia

x_region_Prague

x_region_south_Bohemia

x_region_south_Moravia

x_transaction_amount

x_transaction_count

x_unemployment_rate_1995

x_unemployment_rate_1996

16.2 Variable selection

Starting from this dataset we investigate the presence of redundant or variables not useful for the model such as the ones with not enough variability or with multicollinearity.

16.2.1 Dummy variables

Starting with the dummy variables. We will keep only dummies that has the event in at least 5% of the observation in the dataset.

The below table shows the dummy variables that will be kept in the model:

```
dummy_variables <- dplyr::select(loan_dataset_logistic,
                                starts_with('x_client_gender'),
                                starts_with('x_district_name'),
                                starts_with('x_region'),
                                starts_with('x_card_type'))

dummy_variables_high <- tibble(variables = names(dummy_variables),
                               zeros = sapply(dummy_variables,
                                                function(x) table(as.character(x) == 0)["TRUE"]),
                               ones = sapply(dummy_variables,
                                              function(x) table(as.character(x) == 1)["TRUE"]))) %>%
  mutate(prop_ones = round(ones / (zeros + ones) * 100, 2)) %>%
  arrange(prop_ones) %>%
  filter(prop_ones > 5)

kable(dummy_variables_high)
```

variables	zeros	ones	prop_ones
x_region_south_Bohemia	621	61	8.94
x_region_north_Bohemia	620	62	9.09
x_district_name_Hl.m._Praha	603	79	11.58
x_region_Prague	603	79	11.58
x_region_central_Bohemia	595	87	12.76
x_region_east_Bohemia	595	87	12.76

variables	zeros	ones	prop_ones
x_region_north_Moravia	565	117	17.16
x_region_south_Moravia	549	133	19.50
x_card_type_classic	549	133	19.50
x_client_gender_male	348	334	48.97

The remaining dummies will be excluded from the dataset as they do not have enough variability to fit a logistic model on them.

16.2.2 Transaction type proportion variables

After we investigated the low variability in the dummies, we take care of the calculated variables on the transaction type proportion we calculated during the data enhancement process.

```
prop_variables <- dplyr::select(loan_dataset_logistic,
                               starts_with('x_prop'))
```

```
prop_variables <- summary(prop_variables)
```

```
kable(t(prop_variables))
```

x_prop_household	Min.:0.00000	1st Qu.:0.00000	Median :0.08994	Mean :0.10975	3rd Qu.:0.20138	Max.:0.37573
x_prop_insurance_payment	Min.:0.00000	1st Qu.:0.00000	Median :0.00000	Mean :0.02088	3rd Qu.:0.00000	Max.:0.32182
x_prop_interest_credited	Min.:0.02873	1st Qu.:0.08951	Median :0.11634	Mean :0.13651	3rd Qu.:0.16048	Max.:0.63444
x_prop_loan_payment	Min.:0.00024081	1st Qu.:0.0475180	Median :0.0892142	Mean :0.0986434	3rd Qu.:0.1392529	Max.:0.3025147
x_prop_old_age_pension	Min.:0	1st Qu.:0	Median :0	Mean :0	3rd Qu.:0	Max.:0
x_prop_other	Min.:0.1549	1st Qu.:0.5096	Median :0.5962	Mean :0.5923	3rd Qu.:0.6889	Max.:0.8681
x_prop_statement	Min.:0.00012031	1st Qu.:0.0229538	Median :0.0283864	Mean :0.0391546	3rd Qu.:0.0361376	Max.:0.1883587

```
loan_dataset_logistic <- dplyr::select(loan_dataset_logistic, -x_prop_old_age_pension)
```

The variable **x_prop_old_age_pension** is also excluded from the dataset as it has not a single observation in this sample.

We ended up having a data set with **40 variables**.

variables

y_loan_defaulter

x_account_balance

x_average_salary

x_avg_account_balance

x_card_age_month

x_card_type_classic

x_client_age

x_client_gender_male

x_district_name_Hl.m._Praha

x_last_transaction_age_days

x_loan_amount

x_loan_duration

x_loan_payments

variables

x_no_of_cities
x_no_of_committed_crimes_1995
x_no_of_committed_crimes_1996
x_no_of_entrepreneurs_per_1000_inhabitants
x_no_of_inhabitants
x_no_of_municip_2000_to_9999
x_no_of_municip_500_to_1999
x_no_of_municip_greater_10000
x_no_of_municip_inhabitants_less_499
x_prop_household
x_prop_insurance_payment
x_prop_interest_credited
x_prop_loan_payment
x_prop_other
x_prop_statement
x_ratio_of_urban_inhabitants
x_region_central_Bohemia
x_region_east_Bohemia
x_region_north_Bohemia
x_region_north_Moravia
x_region_Prague
x_region_south_Bohemia
x_region_south_Moravia
x_transaction_amount
x_transaction_count
x_unemployment_rate_1995
x_unemployment_rate_1996

16.2.3 Multicollinearity on feature variables

Multicollinearity is the phenomenon in which a given predictor has a strong correlation with one or more predictors, in this scenario the multiple regression coefficient estimates may vary erratically depending on the data, or the model may not even be possible to be calculate in the case of a perfect correlation on the predictors.

So before we try to fit a model we ran a multicollinearity test to identify additional variables to drop from the model specification.

The idea here is to identify the variables with high correlation among them and keep just the variables that have no strong correlation to the other predictors. We will do so by preferring to keep the variables that have lowest correlation against the others.

Just looking at correlation indexes among pairs of predictors is limited as it is possible that the pairwise correlations are small, and yet a linear dependence exists among three or more variables.

That's why we will rely on a test called Variance Inflation Factors, VIF for short, to detect multicollinearity.

Multicollinearity is poison for Regression algorithms!!!

```
vars.quant <- select_if(loan_dataset_logistic, is.numeric)
VIF <- imcdiag(vars.quant, loan_dataset_logistic$y_loan_defaulter)

VIF_Table_Before <- tibble(variable = names(VIF$idiags[,1]),
                           VIF = VIF$idiags[,1]) %>%
  arrange(desc(VIF))

knitr::kable(VIF_Table_Before)
```

variable	VIF
x_district_name_Hl.m._Praha	Inf
x_region_Prague	Inf
x_no_of_committed_crimes_1996	1737.252782
x_no_of_inhabitants	675.671761
x_prop_other	158.883814
x_prop_household	121.322251
x_prop_interest_credited	63.016856
x_prop_loan_payment	42.844366
x_prop_insurance_payment	33.485319
x_unemployment_rate_1996	17.565631
x_average_salary	14.792865
x_unemployment_rate_1995	13.836322
x_transaction_amount	13.273428
x_prop_statement	12.772024
x_transaction_count	12.427778
x_region_north_Moravia	8.801723
x_no_of_municip_2000_to_9999	6.734151
x_region_south_Moravia	6.373857
x_no_of_cities	6.319146
x_no_of_entrepreneurs_per_1000_inhabitants	6.011420
x_ratio_of_urban_inhabitants	5.366908
x_no_of_municip_500_to_1999	4.801759
x_loan_amount	3.673608
x_region_central_Bohemia	3.666128
x_region_north_Bohemia	3.568482
x_no_of_municip_greater_10000	3.520489
x_no_of_committed_crimes_1995	3.515248
x_loan_payments	3.494173
x_no_of_municip_inhabitants_less_499	3.449762
x_region_east_Bohemia	3.426500

variable	VIF
x_loan_duration	2.654518
x_region_south_Bohemia	2.349037
x_avg_account_balance	1.826483
y_loan_defaulter	1.747653
x_card_type_classic	1.712001
x_card_age_month	1.692020
x_account_balance	1.444177
x_client_age	1.108422
x_last_transaction_age_days	1.086435
x_client_gender_male	1.079951

Having identified the predictors with high VIF we look at the correlogram on those variables.

Here we are hiding the variable names for readability.

```
low_VIF <- filter(VIF_Table_Before, VIF <= 5)$variable
high_VIF <- filter(VIF_Table_Before, VIF > 5)$variable

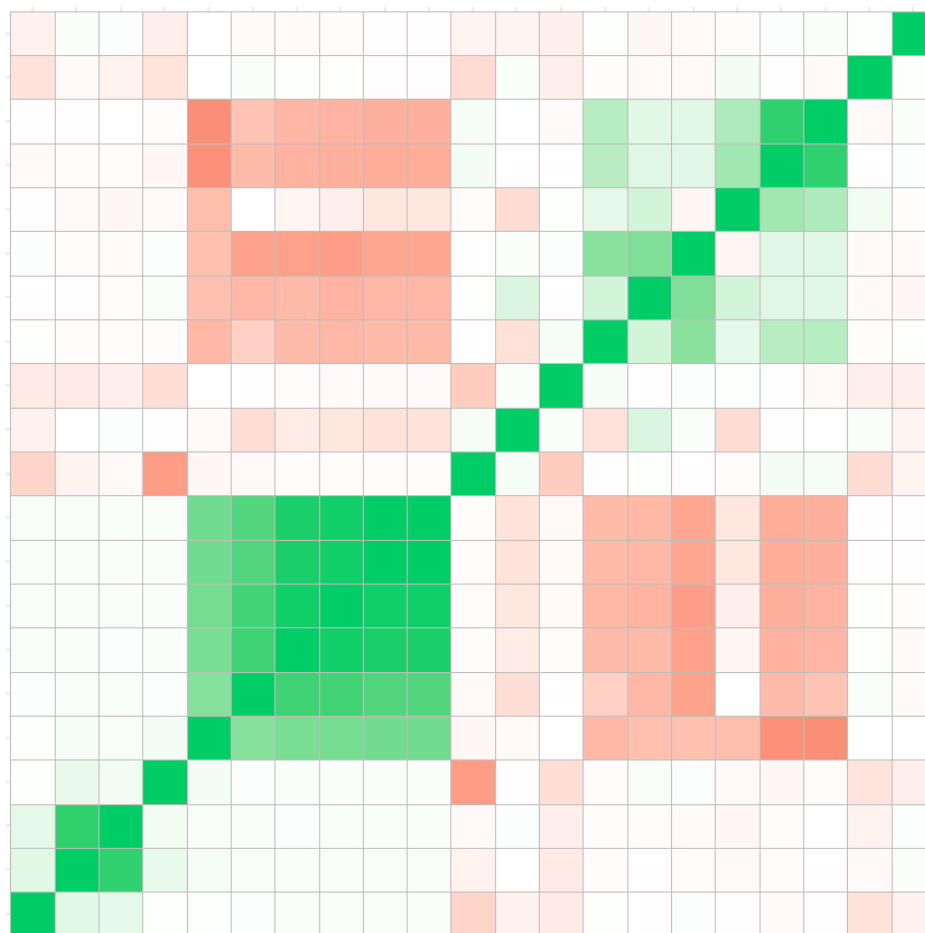
high_VIF_dataset <- dplyr::select(loan_dataset_logistic, high_VIF)

cor_mtx_high_VIF <- cor(high_VIF_dataset)

high_VIF_correlogram_before <- ggcorrplot(cor_mtx_high_VIF,
                                          hc.order = TRUE,
                                          lab = FALSE,
                                          lab_size = 3,
                                          method="square",
                                          colors = c("tomato2", "white", "springgreen3"),
                                          title="Correlation Matrix of Loan Dataset Variables with high VIF") +
  theme(axis.text = element_blank(),
        legend.position = 0)

print(high_VIF_correlogram_before)
```

Correlation Matrix of Loan Dataset Variables with high VIF



We will now exclude the variables with High VIF maintaining only the variables that are not correlated.

We will use the pairwise correlation indexes to choose the variable to reject, code below will exclude the variable in the pair that have the greater correlation index sum against the other predictors not in the pair being evaluated.

At the end we will have only variables that have a correlation index smaller than our selected **threshold (0.6)**.

```

correl_threshold <- 0.6

reject_variables_vector <- tibble(var_1 = row.names(cor_mtx_high_VIF)) %>%
  bind_cols(as_tibble(cor_mtx_high_VIF)) %>%
  melt(id = c("var_1")) %>%
  filter(var_1 != variable) %>%
  mutate(abs_value = abs(value)) %>%
  filter(abs_value > correl_threshold) %>%
  group_by(var_1) %>%
  mutate(sum_1 = sum(abs_value)) %>%
  ungroup() %>%
  group_by(variable) %>%
  mutate(sum_2 = sum(abs_value)) %>%
  ungroup() %>%
  mutate(reject = ifelse(sum_1 > sum_2, var_1, as.character(variable))) %>%
  distinct(reject)

reject_variables_vector <- reject_variables_vector$reject

clean_dataset <- dplyr::select(loan_dataset_logistic, -reject_variables_vector)

kable(reject_variables_vector)

```

x

x_district_name_Hl.m._Praha

x_no_of_committed_crimes_1996

x_no_of_inhabitants

x_no_of_entrepreneurs_per_1000_inhabitants

x_region_Prague

x_prop_other

x_prop_household

x_unemployment_rate_1995

x_transaction_amount

x_transaction_count

x_no_of_cities

After the high VIF predictors exclusion we reevaluate the correlation matrix of the original dataset and the cleaned dataset.

Variable names are hidden for readability. Trust us, there is no correlation greater than 0.6 here.

```

cor_mtx_full <- cor(loan_dataset_logistic)
cor_mtx_clean <- cor(clean_dataset,)

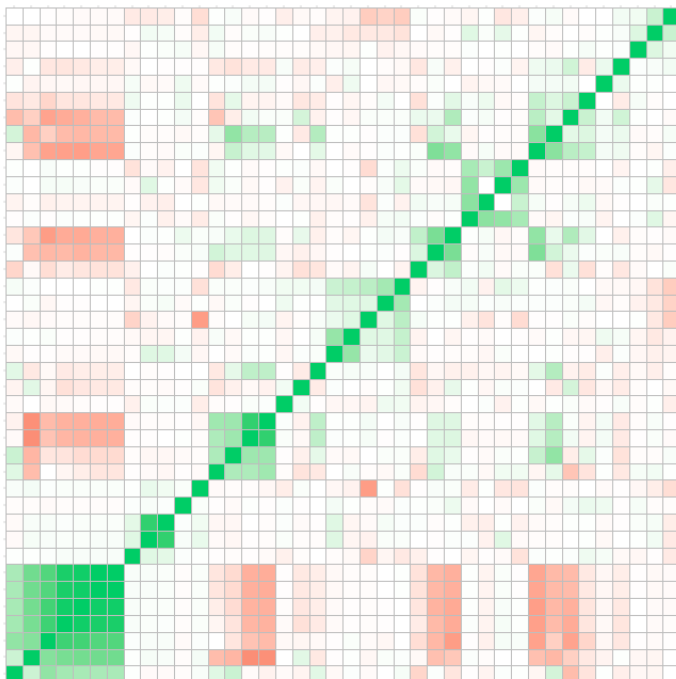
full = ggcorrplot(cor_mtx_full, hc.order = TRUE,
  lab = FALSE,
  lab_size = 3,
  method="square",
  colors = c("tomato2", "white", "springgreen3"),
  title="Correlation Matrix of Full Loan Dataset") +
  theme(axis.text = element_blank(),
    legend.position = 0)

clean = ggcorrplot(cor_mtx_clean, hc.order = TRUE,
  lab = FALSE,
  lab_size = 3,
  method="square",
  colors = c("tomato2", "white", "springgreen3"),
  title="Correlation Matrix of Clean Loan Dataset") +
  theme(axis.text = element_blank(),
    legend.position = 0)

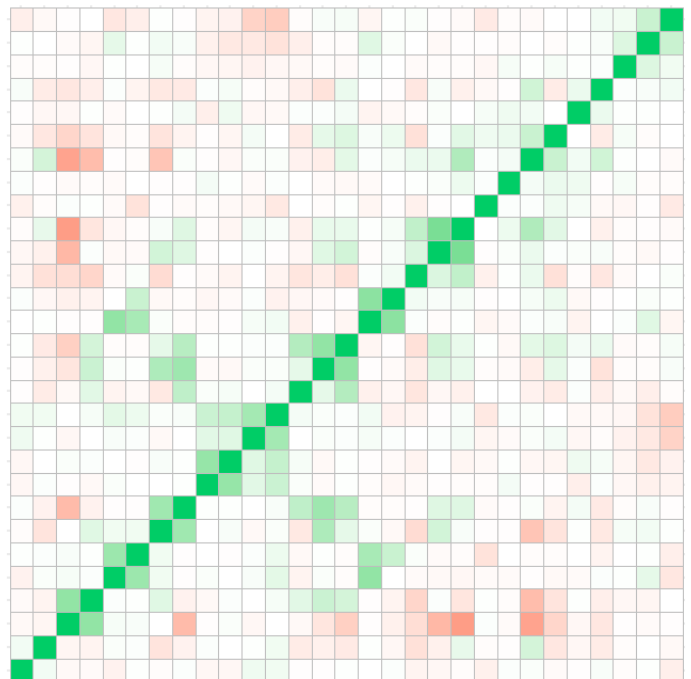
print(ggarrange(full, clean))

```

Correlation Matrix of Full Loan Dataset



Correlation Matrix of Clean Loan Dataset



```
loan_dataset_logistic <- clean_dataset
```

Once again let's look on the VIF estimates.

```
vars.quant <- select_if(loan_dataset_logistic, is.numeric)

VIF <- imcdiag(vars.quant, loan_dataset_logistic$y_loan_defaulter)

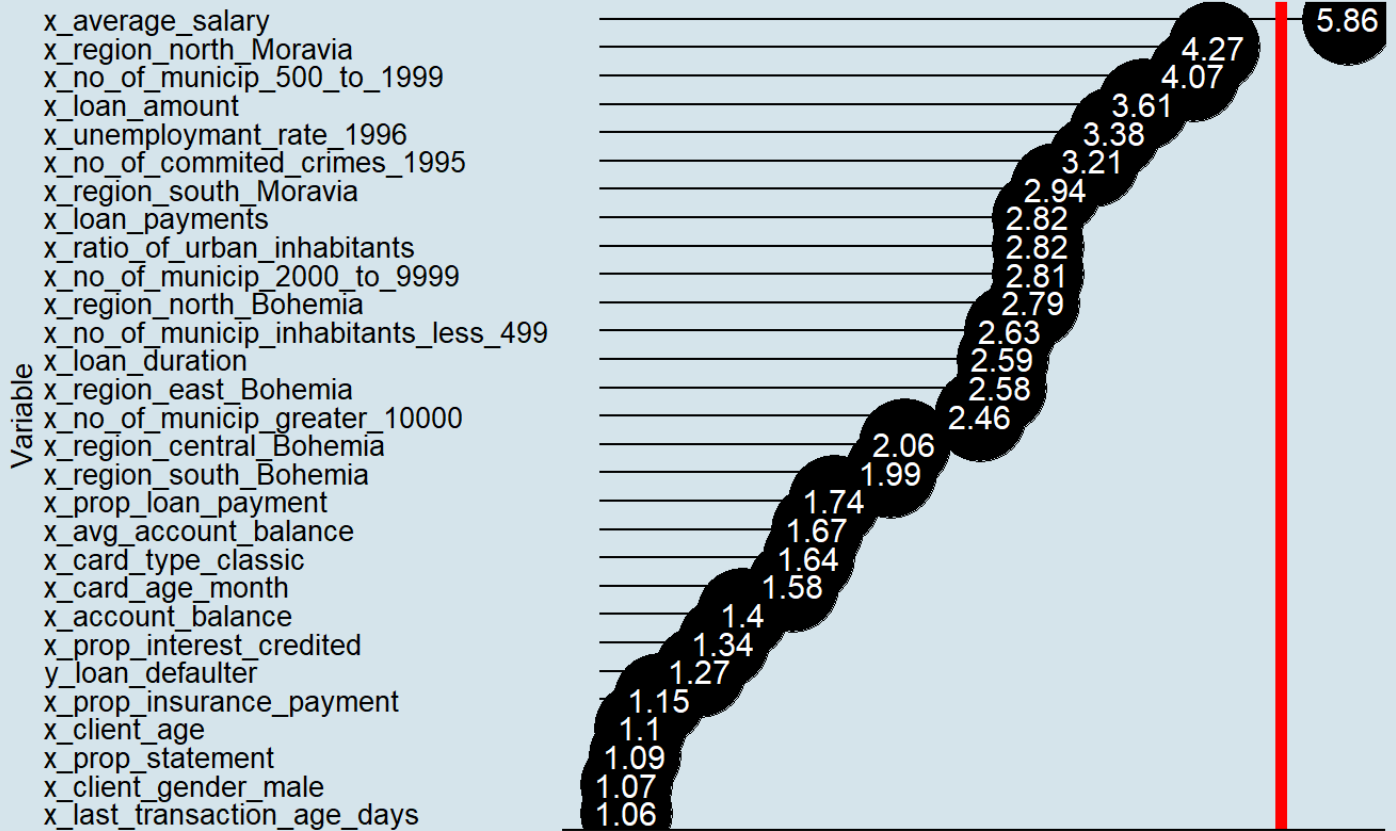
VIF_Table_After <- tibble(variable = names(VIF$idiags[,1]),
                          VIF = VIF$idiags[,1]) %>%
  arrange(desc(VIF))

ggplot(VIF_Table_After, aes(x = fct_reorder(variable, VIF),
                          y = log(VIF), label = round(VIF, 2))) +
  geom_point(stat='identity', fill="black", size=15) +
  geom_segment(aes(y = 0,
                  yend = log(VIF),
                  xend = variable),
              color = "black") +
  geom_text(color="white", size=4) +
  geom_hline(aes(yintercept = log(5)), color = 'red', size = 2) +
  scale_y_continuous(labels = NULL, breaks = NULL) +
  coord_flip() +
  theme_economist() +
  theme(legend.position = 'none',
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(x = 'Variable',
       y = NULL,
       title = 'Variance Inflation Factor',
       subtitle="Checking for multicollinearity in X's variables.
               Variables with VIF more than 5 will be dropped from the model")
```

Variance Inflation Factor

Checking for multicollinearity in X's variables.

Variables with VIF more than 5 will be dropped from the model



```
loan_dataset_logistic <- dplyr::select(loan_dataset_logistic, -x_average_salary)
```

Although **x_average_salary** have no correlation index greater than our defined **threshold (0.6)** it still have a VIF greater than 5, and for pure methodological puritanism we decided to exclude this variable as well.

We did try to keep it, but it had no real difference.

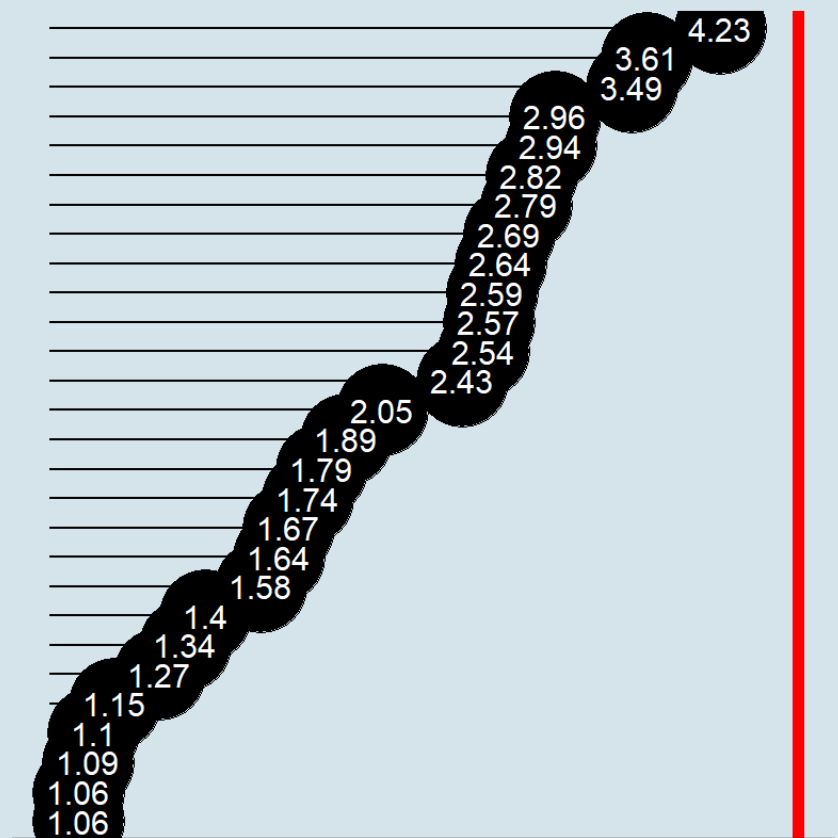
Variance Inflation Factor

Checking for multicollinearity in X's variables.

Variables with VIF more than 5 will be dropped from the model

Variable

x_region_north_Moravia
x_loan_amount
x_no_of_municip_500_to_1999
x_unemployment_rate_1996
x_region_south_Moravia
x_loan_payments
x_region_north_Bohemia
x_no_of_municip_2000_to_9999
x_ratio_of_urban_inhabitants
x_loan_duration
x_no_of_municip_inhabitants_less_499
x_region_east_Bohemia
x_no_of_municip_greater_10000
x_region_central_Bohemia
x_region_south_Bohemia
x_no_of_committed_crimes_1995
x_prop_loan_payment
x_avg_account_balance
x_card_type_classic
x_card_age_month
x_account_balance
x_prop_interest_credited
y_loan_defaulter
x_prop_insurance_payment
x_client_age
x_prop_statement
x_client_gender_male
x_last_transaction_age_days



After all this work we end up having **28 variables** in the dataset.

```
kable(tibble(variables = names(loan_dataset_logistic)))
```

variables

y_loan_defaulter
x_account_balance
x_avg_account_balance
x_card_age_month
x_card_type_classic
x_client_age
x_client_gender_male
x_last_transaction_age_days
x_loan_amount
x_loan_duration
x_loan_payments
x_no_of_committed_crimes_1995
x_no_of_municip_2000_to_9999
x_no_of_municip_500_to_1999

variables

x_no_of_municip_greater_10000

x_no_of_municip_inhabitants_less_499

x_prop_insurance_payment

x_prop_interest_credited

x_prop_loan_payment

x_prop_statement

x_ratio_of_urban_inhabitants

x_region_central_Bohemia

x_region_east_Bohemia

x_region_north_Bohemia

x_region_north_Moravia

x_region_south_Bohemia

x_region_south_Moravia

x_unemployment_rate_1996

16.3 Looking for outliers

And what about outliers?

Regression algorithms are known for its sensitivity for outliers. So, before we go for the modeling juice, we will look for those bastards!!!

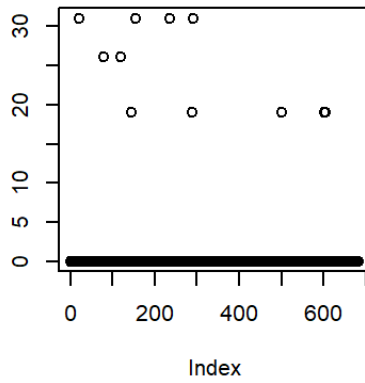
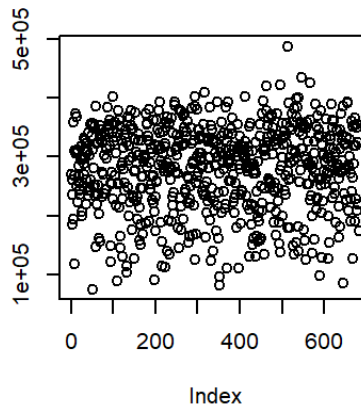
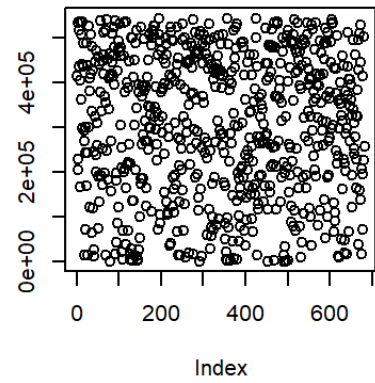
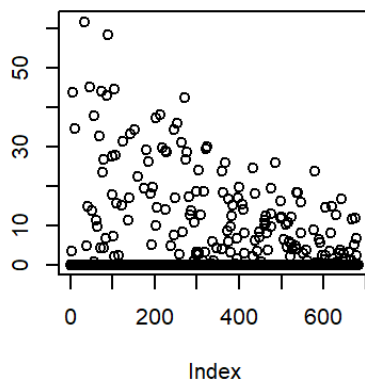
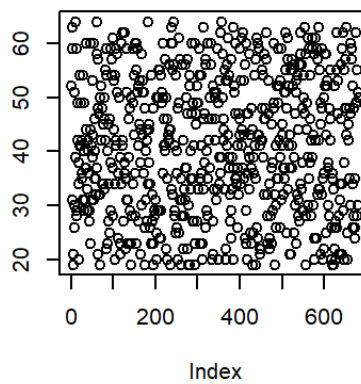
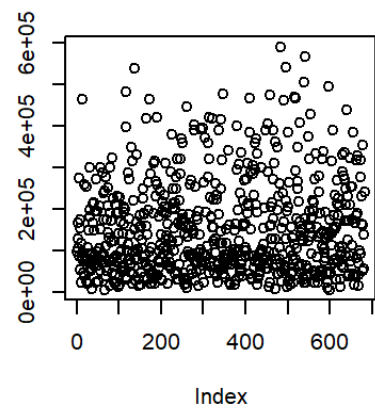
Looking at the plots we see no significant deviation in the quantitative variables.

x_last_transaction_age_days looks to have a discrepancy, but it is ranging from 0 to 30 days, so it is not real outliers.

```
# outliers -----
attach(loan_dataset_logistic)

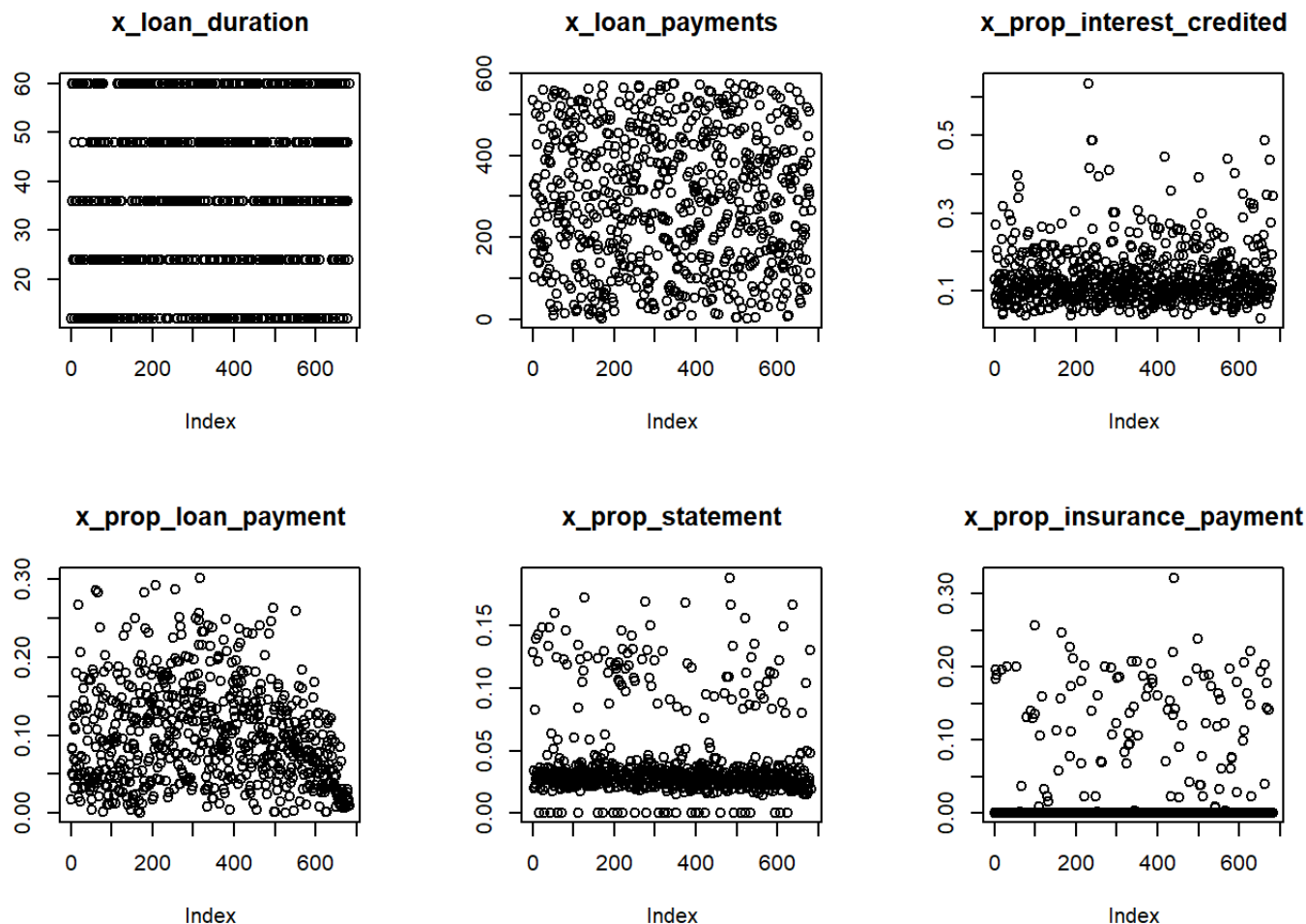
par(mfrow=c(2, 3))

plot(x_last_transaction_age_days, main = "x_last_transaction_age_days", ylab = '')
plot(x_avg_account_balance, main = "x_avg_account_balance", ylab = '')
plot(x_account_balance, main = "x_account_balance", ylab = '')
plot(x_card_age_month, main = "x_card_age_month", ylab = '')
plot(x_client_age, main = "x_client_age", ylab = '')
plot(x_loan_amount, main = "x_loan_amount", ylab = '')
```


x_last_transaction_age_days**x_avg_account_balance****x_account_balance****x_card_age_month****x_client_age****x_loan_amount**

```
par(mfrow=c(2, 3))

plot(x_loan_duration, main = "x_loan_duration", ylab = '')
plot(x_loan_payments, main = "x_loan_payments", ylab = '')
plot(x_prop_interest_credited, main = "x_prop_interest_credited", ylab = '')
plot(x_prop_loan_payment, main = "x_prop_loan_payment", ylab = '')
plot(x_prop_statement, main = "x_prop_statement", ylab = '')
plot(x_prop_insurance_payment, main = "x_prop_insurance_payment", ylab = '')
```



```
detach(loan_dataset_logistic)
```

16.4 Sample split into Test and Training Data

The available data in Loan Dataset is split into Train and Testing data on the following proportion:

- **Train Dataset** (70% 478 obs);
- **Test Dataset** (30% 204 obs).

```
SplitDataset <- source_train_test_dataset
data.train_logistic <- SplitDataset$data.train
data.test_logistic <- SplitDataset$data.test

data.train_logistic <- dplyr::select(data.train_logistic, names(loan_dataset_logistic))
data.test_logistic <- dplyr::select(data.test_logistic, names(loan_dataset_logistic))

kable(SplitDataset$event.proportion)
```

scope	0	1
full dataset	0.8885630	0.1114370
train dataset	0.8933054	0.1066946
test dataset	0.8774510	0.1225490

Both datasets keep the same proportion for the explained variable around 11%.

This split was saved in disk using the function described in the Data Prep session to be reused in the other models fitted in this exercise.

This will ensure consistency when comparing the models against each other.

16.5 Fit the Logistic Regression model

With the final cleaned dataset, we got from above steps we fit our Logistic Regression model for **y_loan_defaulter** on all **x_variables**.

```
# fit model
logistic.full <- glm(formula = y_loan_defaulter ~ .,
                     data= data.train_logistic,
                     family= binomial(link='logit'))

names(logistic.full$coefficients) <- stringr::str_sub(names(logistic.full$coefficients), 1, 25)
summary(logistic.full)

# save model
saveRDS(logistic.full, './models/logistic_full.rds')
```

With the full model fitted we will run the stepwise method to automate the selection of predictors using the **(Akaike Information Criterion) AIC** for short.

This is a broadly used estimator that evaluates the relative quality of statistical methods for a given set of data.

The lower the AIC the better.

```
logistic.step <- step(logistic.full, direction = "both", test = "F")

names(logistic.step$coefficients) <- stringr::str_sub(names(logistic.step$coefficients), 1, 25)
summary(logistic.step)

# save model
saveRDS(logistic.step, './models/logistic_step.rds')
```

```
##
## Call:
## glm(formula = y_loan_defaulter ~ x_card_type_classic + x_loan_amount +
##       x_loan_duration + x_no_of_committed_crimes_1995 + x_prop_interest_credited +
##       x_prop_statement + x_region_north_Bohemia, family = binomial(link = "logit"),
##       data = data.train_logistic)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.3847  -0.4555  -0.2810  -0.1517   2.6217
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -3.579e+00  6.289e-01  -5.691 1.26e-08 ***
## x_card_type_classic  -9.526e-01  6.307e-01  -1.510  0.13094
## x_loan_amount        9.176e-06  1.914e-06   4.794 1.64e-06 ***
## x_loan_duration     -3.878e-02  1.467e-02  -2.643  0.00821 **
## x_no_of_committed_crimes_1 -1.650e-02  7.139e-03  -2.311  0.02083 *
## x_prop_interest_credited  1.072e+01  1.895e+00   5.659 1.52e-08 ***
## x_prop_statement     9.416e+00  4.269e+00   2.206  0.02741 *
## x_region_north_Bohemia -1.547e+01  9.701e+02  -0.016  0.98728
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##    Null deviance: 324.61  on 477  degrees of freedom
## Residual deviance: 249.63  on 470  degrees of freedom
## AIC: 265.63
##
## Number of Fisher Scoring iterations: 17
```

17 Interpreting model output

Logistic Regression models are not only good for prediction but also for inference.

Meaning that it can be used not only in predicting the classification into Defaulters and Non-Defaulters but also to understand the relationship between the predictors and the likelihood of delinquency.

In the Logistic Regression the effect of an increase or decrease of a predictor is not linear, but we can see the predictors that have the greater influence and, by its coefficient signal, the association with an increase or decrease on the likelihood of default.

The probability of default is given by below equation:

$$P(Y = 1|X_n = x_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}$$

Where each β is the coefficient estimate for each variable in the final model.

In our model we see that below variables are associated to an increase in the likelihood of default:

- **x_loan_amount**
- **x_prop_interest_credited**
- **x_prop_statement**

On the other hand, below variables are associated to a decrease in the likelihood of default:

- **x_card_type_classic**
- **x_loan_duration**
- **x_no_of_committed_crimes_1995**
- **x_loan_amount**

- **x_region_north_Bohemia**

18 Evaluating the model performance

We started this step by making predictions using our model on the X's variables in our Train and Test datasets.

```
## making predictions for each model and consolidating in a single data frame
```

```
prob.full  = list()
prob.train = list()
prob.test  = list()

prob.full$logistic.actual      <- loan_dataset_logistic$y_loan_defaulter
prob.full$logistic.predicted  <- predict(logistic.step, type = "response",
                                          newdata = loan_dataset_logistic)

prob.train$logistic.actual     <- data.train_logistic$y_loan_defaulter
prob.train$logistic.predicted <- predict(logistic.step, type = "response",
                                          newdata = data.train_logistic)

prob.test$logistic.actual      <- data.test_logistic$y_loan_defaulter
prob.test$logistic.predicted   <- predict(logistic.step, type = "response",
                                          newdata = data.test_logistic)

prob.full  <- prob.full  %>% as_tibble()
prob.train <- prob.train %>% as_tibble()
prob.test  <- prob.test  %>% as_tibble()
```

18.1 Getting Performance Measures

To calculate the performance measures, derived from the confusion matrix, we need to find the score cut off that best split our test dataset into Defaulters and Non-Defaulters.

In this exercise we decide to not prioritize the accuracy on predicting Defaulters and Non-Defaulters, therefore we are looking for the score cut off that best predict each class equally.

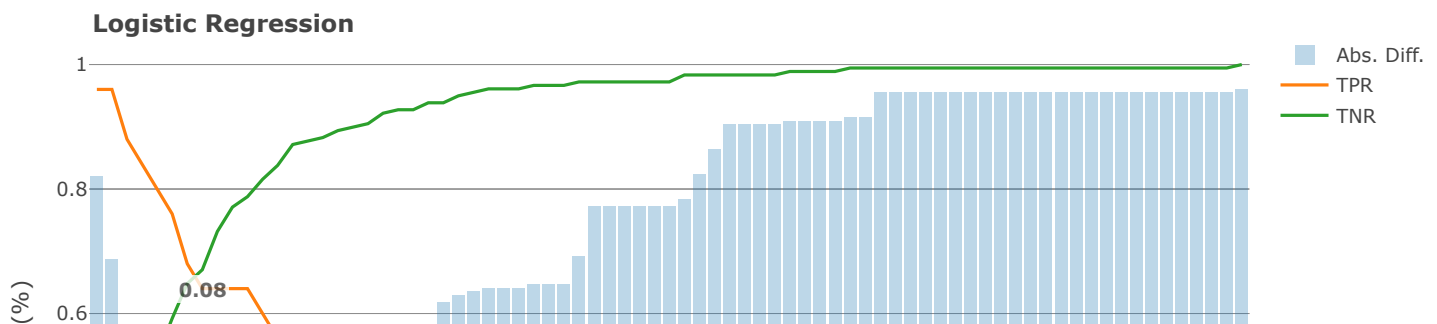
We will use the custom functions described in Auxiliary metrics functions topic in the Data Preparation session of this site.

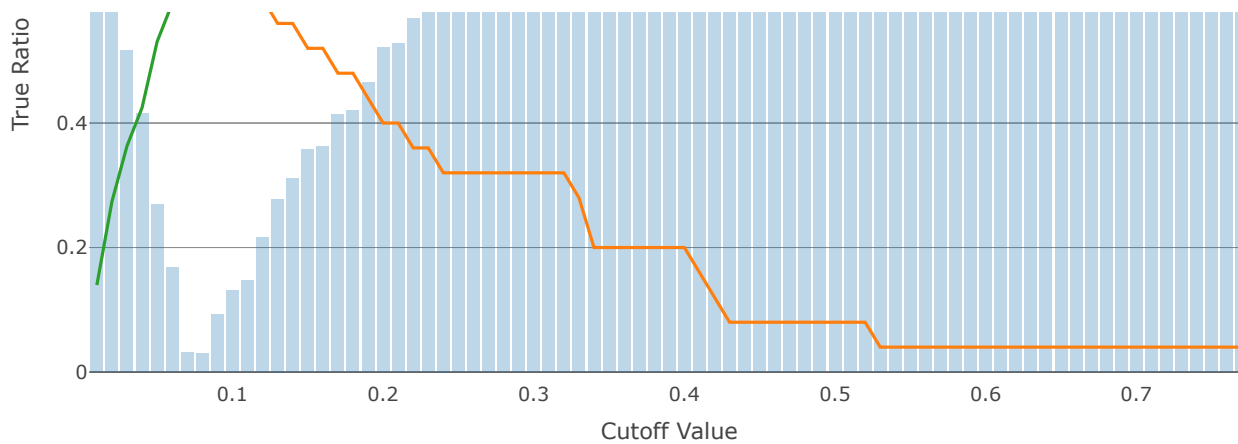
With the returned object from this function we can plot the comparison between TPR (True Positive Rate) and TNR (True Negative Rate) to find the best cut off.

```
## getting measures -----

metricsByCutoff.test_log  <- modelMetrics(prob.test$logistic.actual,
                                          prob.test$logistic.predicted,
                                          plot_title = 'Logistic Regression')

metricsByCutoff.test_log$Plot
```





With the optimized cut off we calculate the full set of model metrics using the function HMeasure from hmeasure library.

```
# logistic regression
measures.logistic.train <- HMeasure(prob.train$logistic.actual,
                                   prob.train$logistic.predicted,
                                   threshold = metricsByCutoff.test_log$BestCut['Cut'])
measures.logistic.test <- HMeasure(prob.test$logistic.actual,
                                   prob.test$logistic.predicted,
                                   threshold = metricsByCutoff.test_log$BestCut['Cut'])

# join measures in a single data frame
measures <- t(bind_rows(measures.logistic.train$metrics,
                       measures.logistic.test$metrics)
              ) %>% as_tibble(., rownames = NA)

colnames(measures) <- c('logistic - train', 'logistic - test')

measures$metric = rownames(measures)

measures <- dplyr::select(measures, metric, everything())
```

Below are the metrics on the train and test dataset:

```
kable(measures, row.names = FALSE)
```

metric	logistic - train	logistic - test
H	0.3612900	0.3070793
Gini	0.6641411	0.5195531
AUC	0.8320705	0.7597765
AUCH	0.8475456	0.7915084
KS	0.5492033	0.4370950
MER	0.0920502	0.1078431
MWL	0.0859316	0.1210592
Spec.Sens95	0.5339578	0.3072626
Sens.Spec95	0.3725490	0.3200000
ER	0.3347280	0.3333333
Sens	0.8431373	0.6400000

metric	logistic - train	logistic - test
Spec	0.6440281	0.6703911
Precision	0.2205128	0.2133333
Recall	0.8431373	0.6400000
TPR	0.8431373	0.6400000
FPR	0.3559719	0.3296089
F	0.3495935	0.3200000
Youden	0.4871654	0.3103911
TP	43.0000000	16.0000000
FP	152.0000000	59.0000000
TN	275.0000000	120.0000000
FN	8.0000000	9.0000000

18.2 Evaluating classification performance

In general the Logistic Regression model is not delivering good accuracy, we will compare how it performed against other classes of models in the Final Report session

Below the confusion matrix and general performance of the model using our custom function **accuracy** described in Auxiliary metrics functions topic in the Data Preparation session of this site.

```
# accuracy metrics -----
# Logistic regression
accuracy(score = prob.test$logistic.predicted,
         actual = prob.test$logistic.actual,
         threshold = metricsByCutoff.test_log[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      16      9
## actual.0      59     120
## [1] "-----"
## [1] "Model General Accuracy of: 66.67%"
## [1] "True Positive Rate of    : 64%"
```

We finally look at the score distribution charts to check how well the model is able to discriminate Defaulters and Non-Defaulters.

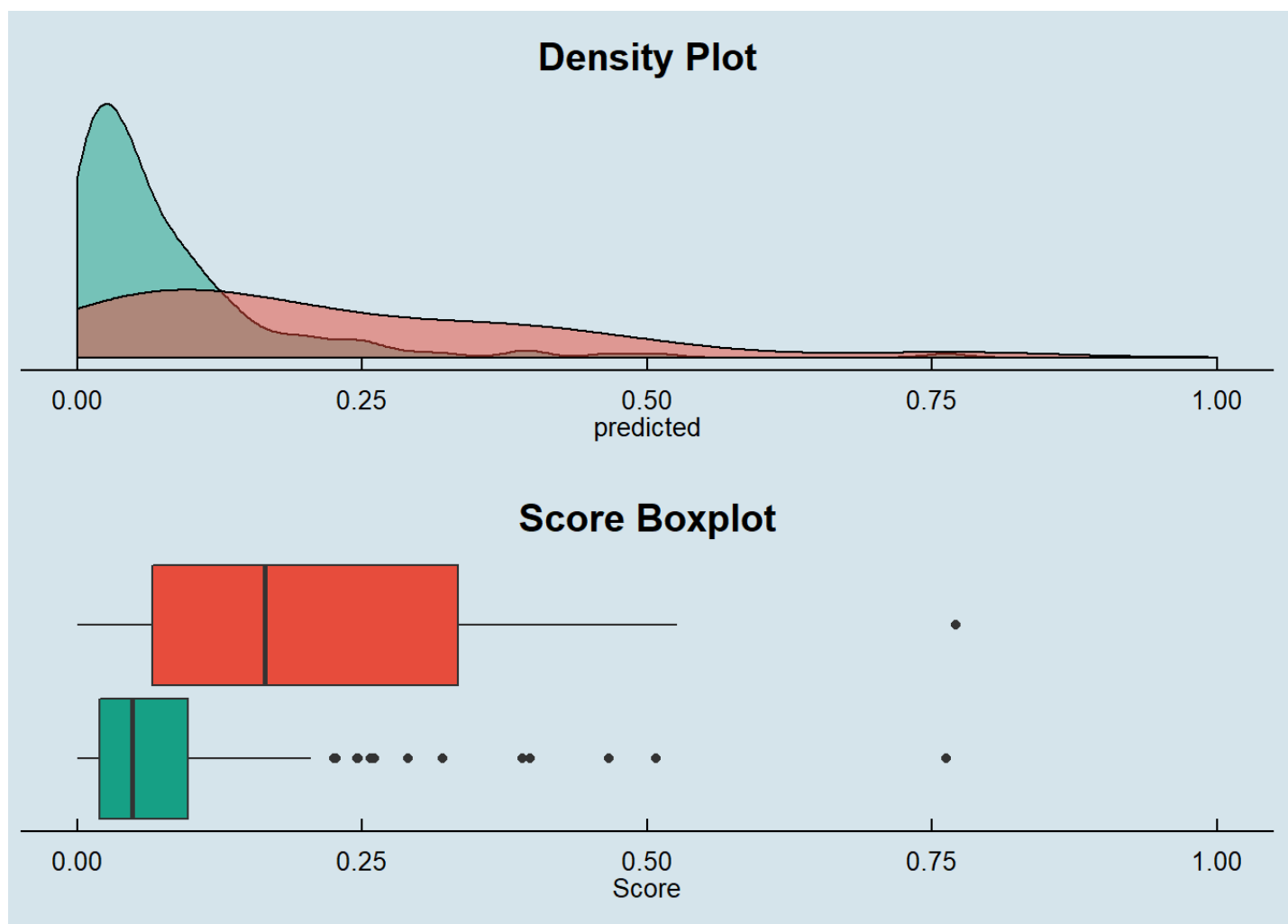
```

p1 <- Score_Histograms(prob.test,
  prob.test$logistic.predicted,
  prob.test$logistic.actual,
  'Density Plot') + theme(axis.title.y = element_blank())

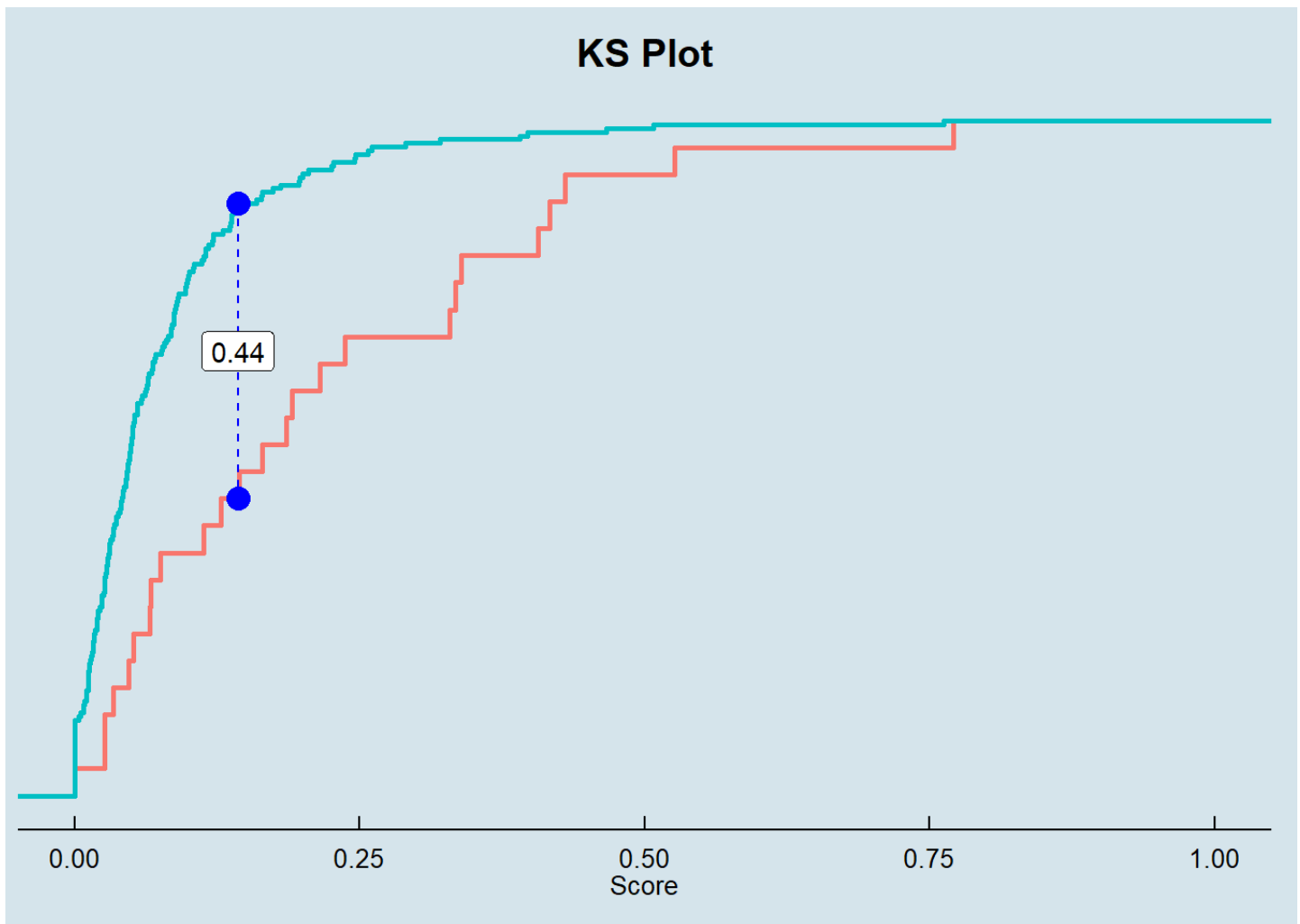
p2 <- Score_Boxplot(prob.test,
  prob.test$logistic.predicted,
  prob.test$logistic.actual,
  'Score Boxplot')

p3 <- KS_Plot(prob.test$logistic.predicted[prob.test$logistic.actual == 0],
  prob.test$logistic.predicted[prob.test$logistic.actual == 1],
  'KS Plot') + theme(axis.title.y = element_blank(),
  axis.text.y = element_blank())
ggarrange(p1, p2, nrow = 2)

```



p3



By the score density we see that our Logistic Regression model is very broad on the scores it assigns to the observations, especially for the real Defaulters not providing a good separation between each class.

The box plot also does not show a clear discrimination between Defaulters and Non-Defaulters.

Finally, the KS metric is far from what is expected for a reasonable classification model.

In the Final Report session, we will look more closely on the AUC and Gini metrics by plotting the ROC curve and comparing against other models.

19 Objective

The goal of this session is trying to fit a simple Decision Tree model on Loan data aiming to predict the probability of delinquency for each contract.

20 Modeling

20.1 Dataset preparation

Using the vanilla transaction dataset, we calculated several derived variables for each account as described in the Data Preparation session.

This dataset is joined with Loan, Client, Credit Card, District, Account and Account Balance tables.

We ended up having a data set with **118 variables**.

```
loan_dataset_DT <- source_dataset

kable(tibble(variables = names(loan_dataset_DT)))
```

variables

y_loan_defaulter

x_account_balance

x_average_salary

x_avg_account_balance

x_card_age_month

x_card_type_classic

x_card_type_gold

x_card_type_junior

x_client_age

x_client_gender_male

x_district_name_Benesov

x_district_name_Beroun

x_district_name_Blansko

x_district_name_Breclav

x_district_name_Brno_mesto

x_district_name_Brno_venkov

x_district_name_Bruntal

x_district_name_Ceska_Lipa

x_district_name_Ceske_Budejovice

x_district_name_Cesky_Krumlov

x_district_name_Cheb

x_district_name_Chomutov

x_district_name_Chrudim

x_district_name_Decin

x_district_name_Domazlice

x_district_name_Frydek_Mistek

x_district_name_Havlickuv_Brod

x_district_name_Hl.m._Praha

x_district_name_Hodonin

x_district_name_Hradec_Kralove

x_district_name_Jablonec_n._Nisou

x_district_name_Jesenik

x_district_name_Jicin

x_district_name_Jihlava

variables

x_district_name_Jindrichuv_Hradec

x_district_name_Karlovy_Vary

x_district_name_Karvina

x_district_name_Kladno

x_district_name_Klatovy

x_district_name_Kolin

x_district_name_Kromeriz

x_district_name_Kutna_Hora

x_district_name_Liberec

x_district_name_Litomerice

x_district_name_Louny

x_district_name_Melnik

x_district_name_Mlada_Boleslav

x_district_name_Most

x_district_name_Nachod

x_district_name_Novy_Jicin

x_district_name_Nymburk

x_district_name_Olomouc

x_district_name_Opava

x_district_name_Ostrava_mesto

x_district_name_Pardubice

x_district_name_Pelhrimov

x_district_name_Pisek

x_district_name_Plzen_jih

x_district_name_Plzen_mesto

x_district_name_Plzen_sever

x_district_name_Prachatice

x_district_name_Praha_vychod

x_district_name_Praha_zapad

x_district_name_Prerov

x_district_name_Pribram

x_district_name_Prostejov

x_district_name_Rakovnik

x_district_name_Rokycany

x_district_name_Rychnov_nad_Kneznou

x_district_name_Semily

x_district_name_Strakonice

variables

x_district_name_Sumperk
x_district_name_Svitavy
x_district_name_Tabor
x_district_name_Tachov
x_district_name_Teplice
x_district_name_Trebic
x_district_name_Trutnov
x_district_name_Uherske_Hradiste
x_district_name_Usti_nad_Labem
x_district_name_Usti_nad_Orlici
x_district_name_Vsetin
x_district_name_Vyskov
x_district_name_Zdar_nad_Sazavou
x_district_name_Zlin
x_district_name_Znojmo
x_last_transaction_age_days
x_loan_amount
x_loan_duration
x_loan_payments
x_no_of_cities
x_no_of_committed_crimes_1995
x_no_of_committed_crimes_1996
x_no_of_entrepreneurs_per_1000_inhabitants
x_no_of_inhabitants
x_no_of_municip_2000_to_9999
x_no_of_municip_500_to_1999
x_no_of_municip_greater_10000
x_no_of_municip_inhabitants_less_499
x_prop_household
x_prop_insurance_payment
x_prop_interest_credited
x_prop_loan_payment
x_prop_old_age_pension
x_prop_other
x_prop_statement
x_ratio_of_urban_inhabitants
x_region_central_Bohemia

variables

x_region_east_Bohemia
x_region_north_Bohemia
x_region_north_Moravia
x_region_Prague
x_region_south_Bohemia
x_region_south_Moravia
x_transaction_amount
x_transaction_count
x_unemployment_rate_1995
x_unemployment_rate_1996

20.2 Variable selection

One advantage of Decision Tree models is that it does not require heavy feature engineering.

We will only remove **x_prop_old_age_pension** that we know beforehand to have no variance in the dataset.

This model is also not sensible to outliers, missing values and multicollinearity.

```
loan_dataset_DT <- source_dataset
loan_dataset_DT <- dplyr::select(loan_dataset_DT, -x_prop_old_age_pension)
```

20.3 Sample split into Test and Training Data

The available data in Loan Dataset is split into Train and Testing data on the following proportion:

- **Train Dataset** (70% 478 obs);
- **Test Dataset** (30% 204 obs).

We are selecting exact the same samples we used for the Logistic Model to allow comparison across models.

```
SplitDataset <- source_train_test_dataset
data.train_DT <- SplitDataset$data.train
data.test_DT <- SplitDataset$data.test

data.train_DT <- dplyr::select(data.train_DT, names(loan_dataset_DT))
data.test_DT <- dplyr::select(data.test_DT, names(loan_dataset_DT))

kable(SplitDataset$event.proportion)
```

scope	0	1
full dataset	0.8885630	0.1114370
train dataset	0.8933054	0.1066946
test dataset	0.8774510	0.1225490

Both datasets keep the same proportion for the explained variable around 11%.

20.4 Fiting the Decision Tree model

With the final cleaned dataset, we got from above steps we fit our Decision Tree Model for **y_loan_defaulter** on all **x_variables**.

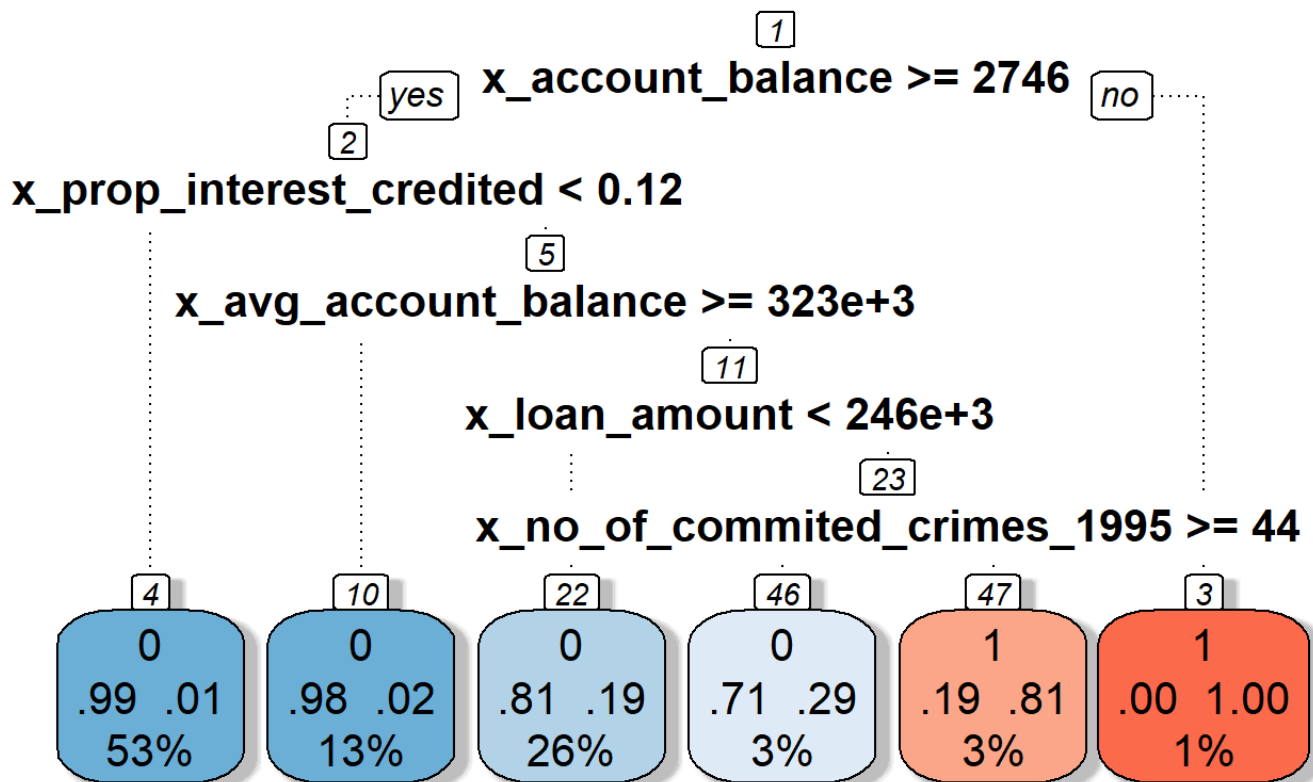
We choose to change the default parameters **minbucket** and **maxdepth** to **5** to deliberate create a fully-grown tree as the database is not big enough for this model.

```
tree.full <- rpart(data= data.train_DT, y_loan_defaulter ~ .,
  control = rpart.control(minbucket = 5,
    maxdepth = 5),
  method = "class")

rpart.plot(tree.full, cex = 1.3, type = 0,
  extra = 1, box.palette = 'BuRd',
  branch.lty = 3, shadow.col = 'gray',
  nn = TRUE, main = 'Decision Tree - Full')

# save model
saveRDS(tree.full, './models/decision_tree_full.rds')
```

Decision Tree - Full



20.5 Evaluating necessity of pruning

Pruning is a technique in machine learning used to reduce the size of decision trees by removing nodes that provide little power of classification.

The idea is reducing the complexity of the decision tree and thereof avoid overfit to the train dataset.

Pruning process is done by comparing different variations of the fully-grown tree and evaluating the relative error trend comparing to a zero node tree, each node that does not affect the classification power of the tree is replaced by a leaf node.

```

printcp(tree.full)
plotcp(tree.full)

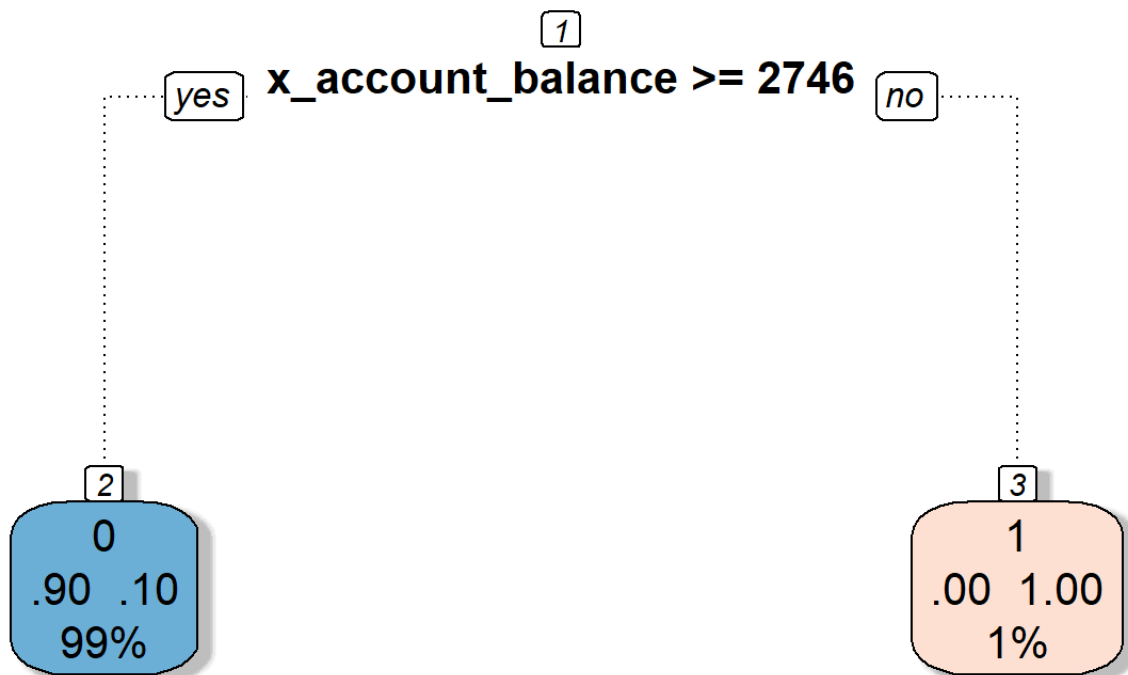
cp_prune = tree.full$cptable[which.min(tree.full$cptable[, "xerror"]), "CP"]
tree.prune <- prune(tree.full, cp = cp_prune)

rpart.plot(tree.prune, cex = 1.3, type = 0,
  extra = 104, box.palette = 'BuRd',
  branch.lty = 3, shadow.col = 'gray',
  nn = TRUE, main = 'Decision Tree - Prune')

# save model
saveRDS(tree.prune, './models/decision_tree_prune.rds')

```

Decision Tree - Prune



On performing the pruning process in our decision tree, we end up with a tree with only one node split. This is because splitting the train data set into just one node provide an overall accuracy greater than any other combination.

Here the technique falls short for this dataset, we would be interest in give a greater weight on correctly predict the real defaulters than the general accuracy of the model.

A Decision Tree is definitely not a good model for this dataset, but as we are just exercising the modeling technique (just having fun here !!!) we will use the full decision tree we got to compare its metrics and see how it perform against the other models we created in this class exercise.

We tried different parameters for **minbucket** and **maxdepth**, as our dataset is not big enough all of them presented huge differences in performance, the one we chose for this exercise seems to be a reasonable trade-off on overfitting to compare against the other models.

We can do this by simply feeding back the full tree to our prune tree object and move on to the performance metrics.

```
tree.prune <- tree.full
```

21 Interpreting model output

Decision Trees are known by its very descriptive rules on how it is classifying each observation. It provides a very clear human readable set of rules that can show the importance of each variable on the decision process.

Our full decision tree model clear shows that below variables are the key features that can be used to decide the likelihood of default in each contract:

- **x_account_balance**
- **x_prop_interest_credited**
- **x_avg_account_balance**
- **x_loan_amount**
- **x_no_of_committed_crimes_1995**

The result is similar to the Logistic Regression model we created in the session before.

The predictors selected are roughly the same.

22 Evaluating the model performance

Here we will perform basically the same steps we did in the Logistic Regression model.

A comparison against all the models will be provided in the Final Report session of this exercise.

We started this step by making predictions using our model on the X's variables in our Train and Test datasets.

```
## making predictions for each model and consolidating in a single data frame

prob.full = list()
prob.train = list()
prob.test = list()

prob.full$decision.tree.actual <- loan_dataset_DT$y_loan_defaulter
prob.full$decision.tree.predicted <- predict(tree.prune, type = "prob", newdata = loan_dataset_DT)[, 2]

prob.train$decision.tree.actual <- data.train_DT$y_loan_defaulter
prob.train$decision.tree.predicted <- predict(tree.prune, type = "prob", newdata = data.train_DT)[, 2]

prob.test$decision.tree.actual <- data.test_DT$y_loan_defaulter
prob.test$decision.tree.predicted <- predict(tree.prune, type = "prob", newdata = data.test_DT)[, 2]

prob.full <- prob.full %>% as_tibble()
prob.train <- prob.train %>% as_tibble()
prob.test <- prob.test %>% as_tibble()
```

22.1 Getting performance measures

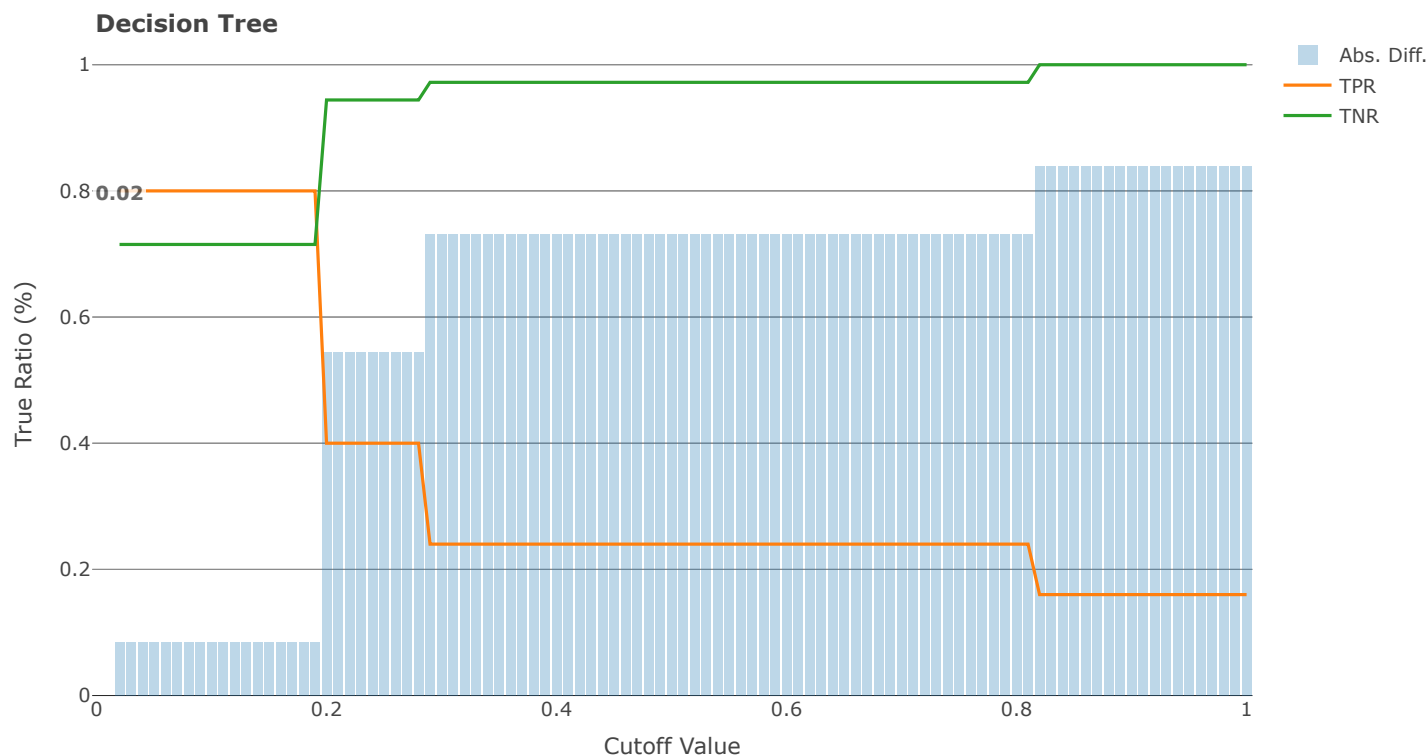
To calculate the performance measures, derived from the confusion matrix, we need to find the score cut off that best split our test dataset into Defaulters and Non-Defaulters.

In this exercise we decide to not prioritize the accuracy on predicting Defaulters and Non-Defaulters, therefore we are looking for the score cut off that best predict each class equally.

With the returned object from this function we can plot the comparison between TPR (True Positive Rate) and TNR (True Negative Rate) to find the best cut off.


```
## getting measures -----
metricsByCutoff.test_DT <- modelMetrics(prob.test$decision.tree.actual,
                                         prob.test$decision.tree.predicted,
                                         plot_title = 'Decision Tree')

metricsByCutoff.test_DT$Plot
```



With the optimized cut off we calculate the full set of model metrics using the function HMeasure from hmeasure library.

```
# decision tree
measures.decision.tree.train <- HMeasure(prob.train$decision.tree.actual,
                                         prob.train$decision.tree.predicted,
                                         threshold = metricsByCutoff.test_DT$BestCut['Cut'])
measures.decision.tree.test <- HMeasure(prob.test$decision.tree.actual,
                                         prob.test$decision.tree.predicted,
                                         threshold = metricsByCutoff.test_DT$BestCut['Cut'])

# join measures in a single data frame
measures <- t(bind_rows(measures.decision.tree.train$metrics,
                        measures.decision.tree.test$metrics
                        )) %>% as_tibble(., rownames = NA)

colnames(measures) <- c('decision.tree - train', 'decision.tree - test')

measures$metric = rownames(measures)

measures <- dplyr::select(measures, metric, everything())
```

Below are the metrics on the train and test dataset:

```
kable(measures, row.names = FALSE)
```

metric	decision.tree - train	decision.tree - test
--------	-----------------------	----------------------

metric	decision.tree - train	decision.tree - test
H	0.4815174	0.3299682
Gini	0.7575883	0.5883799
AUC	0.8787941	0.7941899
AUCH	0.8787941	0.7966480
KS	0.6522478	0.5150838
MER	0.0732218	0.1029412
MWL	0.0662891	0.1042868
Spec.Sens95	0.0878220	0.3840782
Sens.Spec95	0.8830450	0.2736000
ER	0.2489540	0.2745098
Sens	0.9215686	0.8000000
Spec	0.7306792	0.7150838
Precision	0.2901235	0.2816901
Recall	0.9215686	0.8000000
TPR	0.9215686	0.8000000
FPR	0.2693208	0.2849162
F	0.4413146	0.4166667
Youden	0.6522478	0.5150838
TP	47.0000000	20.0000000
FP	115.0000000	51.0000000
TN	312.0000000	128.0000000
FN	4.0000000	5.0000000

22.2 Evaluating classification performance

In general, such as the Logistic Regression model, this model is not delivering good accuracy, we will compare how it performed against other classes of models in the Final Report session

Below the confusion matrix and general performance of the model:

```
# accuracy metrics -----
# decision tree
accuracy(score = prob.test$decision.tree.predicted,
          actual = prob.test$decision.tree.actual,
          threshold = metricsByCutoff.test_DT[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      20      5
## actual.0      51     128
## [1] "-----"
## [1] "Model General Accuracy of: 72.55%"
## [1] "True Positive Rate of      : 80%"
```

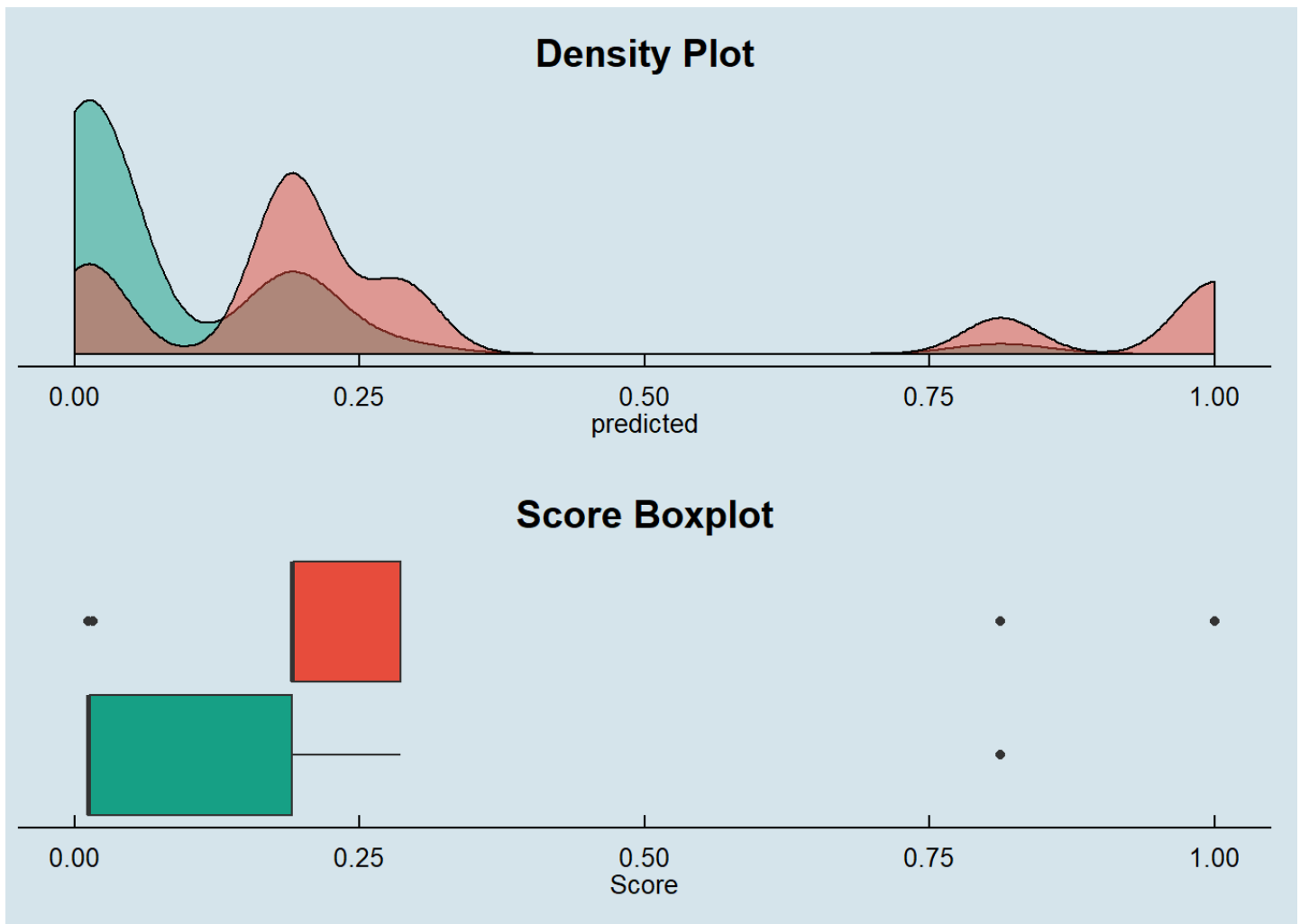
We finally look at the score distribution charts to check how well the model is able to discriminate Defaulters and Non-Defaulters.

```
p1 <- Score_Histograms(prob.test,
  prob.test$decision.tree.predicted,
  prob.test$decision.tree.actual,
  'Density Plot') + theme(axis.title.y = element_blank())

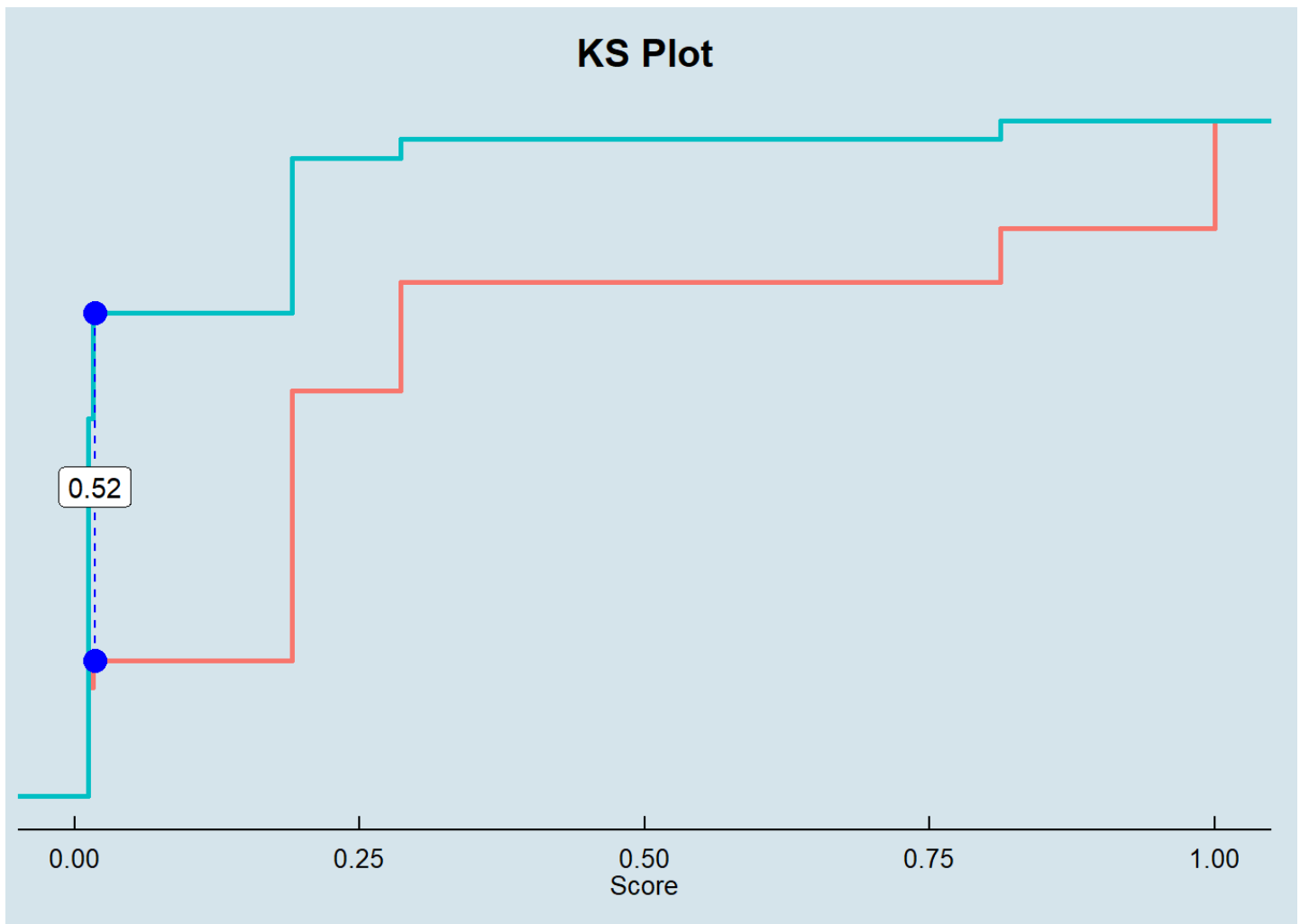
p2 <- Score_Boxplot(prob.test,
  prob.test$decision.tree.predicted,
  prob.test$decision.tree.actual,
  'Score Boxplot')

p3 <- KS_Plot(prob.test$decision.tree.predicted[prob.test$decision.tree.actual == 0],
  prob.test$decision.tree.predicted[prob.test$decision.tree.actual == 1],
  'KS Plot') + theme(axis.title.y = element_blank(),
  axis.text.y = element_blank())

ggarrange(p1, p2, nrow = 2)
```



p3



By the score density we see that our Decision Tree is very narrow on the scores it assigns to the observations.

This is expected from single Decision Trees.

The box plot also does not show a clear discrimination between Defaulters and Non-Defaulters.

Finally, the KS metric is far from what is expected for a reasonable classification model.

In the Final Report session, we will look more closely on the AUC and Gini metrics by plotting the ROC curve and comparing against other models.

Stay tuned!!!

23 Objective

The goal of this session is trying to fit a Boosting model on Loan data aiming to predict the probability of delinquency for each contract.

24 Modeling

24.1 Dataset preparation

Using the vanilla transaction dataset, we calculated several derived variables for each account as described in the Data Preparation session.

This dataset is joined with Loan, Client, Credit Card, District, Account and Account Balance tables.

We ended up having a data set with **118 variables**.

```
loan_dataset_boost <- source_dataset

kable(tibble(variables = names(loan_dataset_boost)))
```

variables

y_loan_defaulter

x_account_balance

x_average_salary

x_avg_account_balance

x_card_age_month

x_card_type_classic

x_card_type_gold

x_card_type_junior

x_client_age

x_client_gender_male

x_district_name_Benesov

x_district_name_Beroun

x_district_name_Blansko

x_district_name_Breclav

x_district_name_Brno_mesto

x_district_name_Brno_venkov

x_district_name_Bruntal

x_district_name_Ceska_Lipa

x_district_name_Ceske_Budejovice

x_district_name_Cesky_Krumlov

x_district_name_Cheb

x_district_name_Chomutov

x_district_name_Chrudim

x_district_name_Decin

x_district_name_Domazlice

x_district_name_Frydek_Mistek

x_district_name_Havlickuv_Brod

x_district_name_Hl.m._Praha

x_district_name_Hodonin

x_district_name_Hradec_Kralove

x_district_name_Jablonec_n._Nisou

x_district_name_Jesenik

x_district_name_Jicin

x_district_name_Jihlava

variables

x_district_name_Jindrichuv_Hradec

x_district_name_Karlovy_Vary

x_district_name_Karvina

x_district_name_Kladno

x_district_name_Klatovy

x_district_name_Kolin

x_district_name_Kromeriz

x_district_name_Kutna_Hora

x_district_name_Liberec

x_district_name_Litomerice

x_district_name_Louny

x_district_name_Melnik

x_district_name_Mlada_Boleslav

x_district_name_Most

x_district_name_Nachod

x_district_name_Novy_Jicin

x_district_name_Nymburk

x_district_name_Olomouc

x_district_name_Opava

x_district_name_Ostrava_mesto

x_district_name_Pardubice

x_district_name_Pelhrimov

x_district_name_Pisek

x_district_name_Plzen_jih

x_district_name_Plzen_mesto

x_district_name_Plzen_sever

x_district_name_Prachatice

x_district_name_Praha_vychod

x_district_name_Praha_zapad

x_district_name_Prerov

x_district_name_Pribram

x_district_name_Prostejov

x_district_name_Rakovnik

x_district_name_Rokycany

x_district_name_Rychnov_nad_Kneznou

x_district_name_Semily

x_district_name_Strakonice

variables

x_district_name_Sumperk
x_district_name_Svitavy
x_district_name_Tabor
x_district_name_Tachov
x_district_name_Teplice
x_district_name_Trebic
x_district_name_Trutnov
x_district_name_Uherske_Hradiste
x_district_name_Usti_nad_Labem
x_district_name_Usti_nad_Orlici
x_district_name_Vsetin
x_district_name_Vyskov
x_district_name_Zdar_nad_Sazavou
x_district_name_Zlin
x_district_name_Znojmo
x_last_transaction_age_days
x_loan_amount
x_loan_duration
x_loan_payments
x_no_of_cities
x_no_of_committed_crimes_1995
x_no_of_committed_crimes_1996
x_no_of_entrepreneurs_per_1000_inhabitants
x_no_of_inhabitants
x_no_of_municip_2000_to_9999
x_no_of_municip_500_to_1999
x_no_of_municip_greater_10000
x_no_of_municip_inhabitants_less_499
x_prop_household
x_prop_insurance_payment
x_prop_interest_credited
x_prop_loan_payment
x_prop_old_age_pension
x_prop_other
x_prop_statement
x_ratio_of_urban_inhabitants
x_region_central_Bohemia

variables

x_region_east_Bohemia
x_region_north_Bohemia
x_region_north_Moravia
x_region_Prague
x_region_south_Bohemia
x_region_south_Moravia
x_transaction_amount
x_transaction_count
x_unemployment_rate_1995
x_unemployment_rate_1996

24.2 Variable selection

One advantage of Boosting models is that it does not require heavy feature engineering.

We will only remove **x_prop_old_age_pension** that we know beforehand to have no variance in the dataset.

This model is also not sensible to outliers, missing values and multicollinearity.

```
loan_dataset_boost <- dplyr::select(loan_dataset_boost, -x_prop_old_age_pension)
```

24.3 Sample split into Test and Training Data

The available data in Loan Dataset is split into Train and Testing data on the following proportion:

- **Train Dataset** (70% 478 obs);
- **Test Dataset** (30% 204 obs).

We are selecting exact the same samples for all models to allow comparison between then.

```
SplitDataset <- source_train_test_dataset
data.train_boost <- SplitDataset$data.train
data.test_boost <- SplitDataset$data.test

kable(SplitDataset$event.proportion)
```

scope	0	1
full dataset	0.8885630	0.1114370
train dataset	0.8933054	0.1066946
test dataset	0.8774510	0.1225490

```
loan_dataset_boost$y_loan_defaulter <- as.factor(loan_dataset_boost$y_loan_defaulter)
data.train_boost$y_loan_defaulter <- as.factor(data.train_boost$y_loan_defaulter)
data.test_boost$y_loan_defaulter <- as.factor(data.test_boost$y_loan_defaulter)

data.train_boost <- dplyr::select(data.train_boost, names(loan_dataset_boost))
data.test_boost <- dplyr::select(data.test_boost, names(loan_dataset_boost))
```

Both datasets keep the same proportion for the explained variable around 11%.

24.4 Fiting the Boosting model

With the final cleaned dataset, we got from above steps fit our Boosting Model for **y_loan_defaulter** on all **x_variables**.

We made a lot of tests playing with control parameters trying to reduce the errors and we decided to keep **mfinal = 100**, **minbucket = 25** and **maxdepth = 1**.

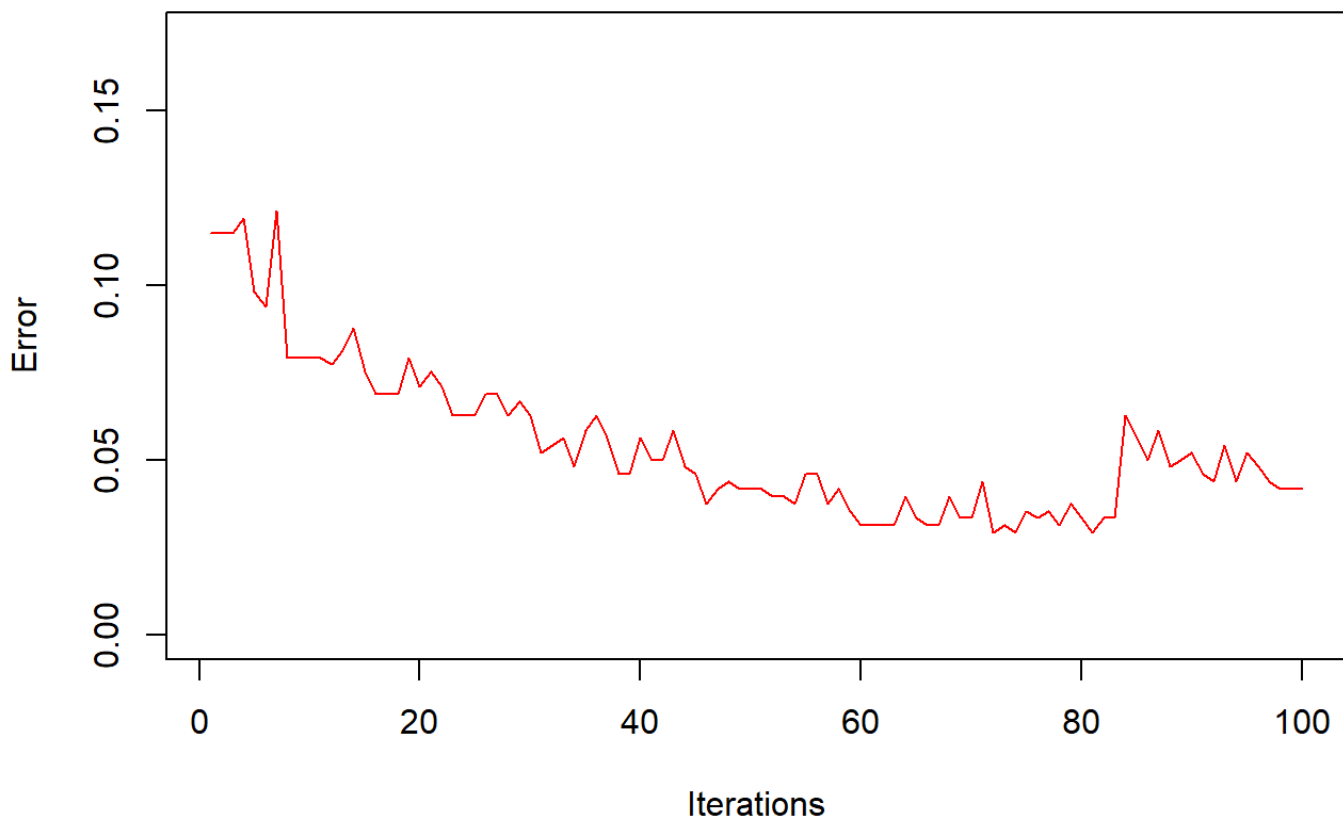
```
names <- names(data.train_boost) # saving the name of all vars to put on formula
f_full <- as.formula(paste("y_loan_defaulter ~",
                           paste(names[!names %in% "y_loan_defaulter"], collapse = " + ")))

boost <- boosting(f_full, data= data.train_boost, mfina1= 100,
                  coeflearn = "Freund",
                  control = rpart.control(minbucket= 25,maxdepth = 1))

saveRDS(boost, "./models/boosting.rds")
```

```
plot(erorevol(boost, data.train_boost))
```

Ensemble error vs number of trees



25 Interpreting model output

Boosting is a black box ensemble method! But the model can tell us the importance of each variable to predict the results. For this model the four principal vars are:

- **x_prop_interest_credited**
- **x_account_balance**
- **x_avg_account_balance**
- **x_loan_amount**

The result is similar to the Logistic Regression and Decision Tree models.

26 Evaluating the model performance

Here we will perform basically the same steps we did in the Logistic Regression and Decision Tree models.

A comparison against all the models will be provided in the Final Report session of this exercise.

We started this step by making predictions using our model on the X's variables in our Train and Test datasets.

```
## making predictions for each model and consolidating in a single data frame

prob.full = list()
prob.train = list()
prob.test = list()

prob.full$boosting.actual      <- loan_dataset_boost$y_loan_defaulter
prob.full$boosting.predicted   <- predict.boosting(boost,
                                                    newdata = loan_dataset_boost)$prob[, 2]

prob.train$boosting.actual     <- data.train_boost$y_loan_defaulter
prob.train$boosting.predicted  <- predict.boosting(boost, newdata =
                                                    data.train_boost)$prob[, 2]

prob.test$boosting.actual      <- data.test_boost$y_loan_defaulter
prob.test$boosting.predicted   <- predict.boosting(boost, newdata =
                                                    data.test_boost)$prob[, 2]

prob.full  <- prob.full %>% as_tibble()
prob.train <- prob.train %>% as_tibble()
prob.test  <- prob.test %>% as_tibble()
```

26.1 Getting Performance Measures

To calculate the performance measures, derived from the confusion matrix, we need to find the score cut off that best split our test dataset into Defaulters and Non-Defaulters.

In this exercise we decide to not prioritize the accuracy on predicting Defaulters and Non-Defaulters, therefore we are looking for the score cut off that best predict each class equally.

With the returned object from this function we can plot the comparison between TPR (True Positive Rate) and TNR (True Negative Rate) to find the best cut off.

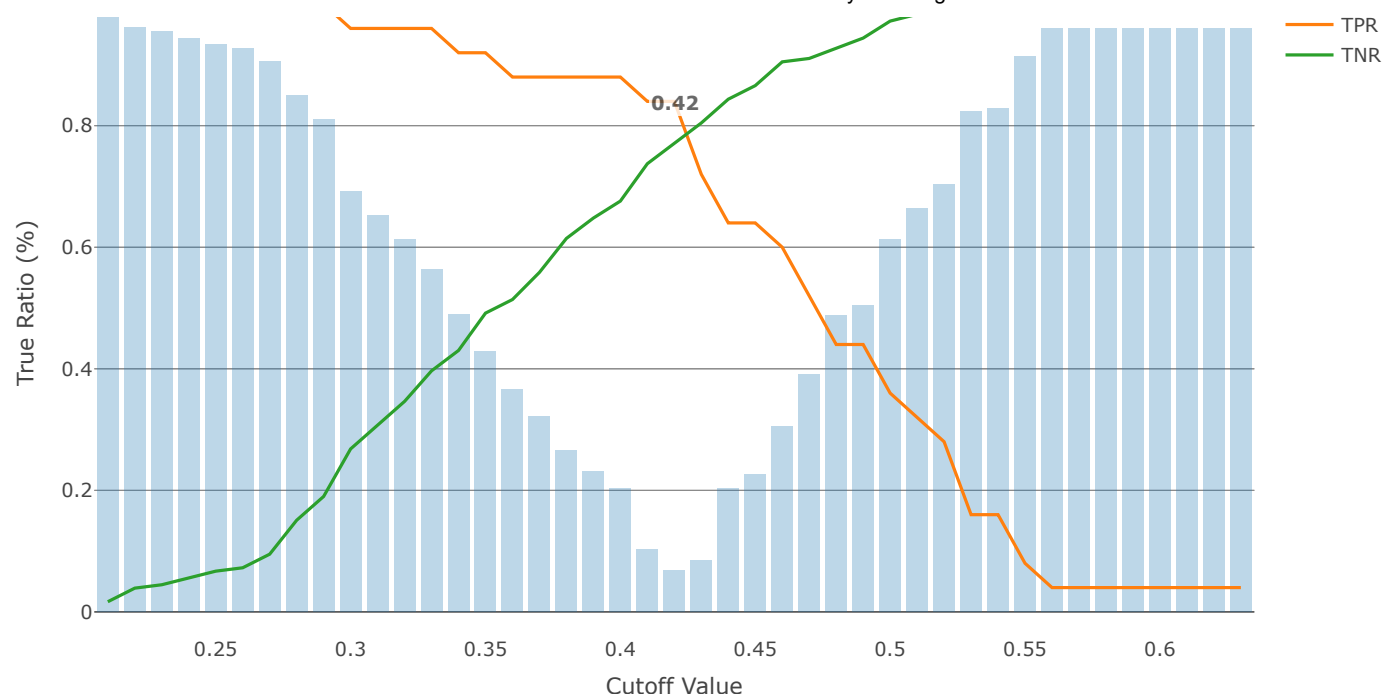
```
## getting measures -----
metricsByCutoff.test_boost <- modelMetrics(prob.test$boosting.actual,
                                           prob.test$boosting.predicted,
                                           plot_title = 'Boosting')

metricsByCutoff.test_boost$Plot
```

Boosting

1

 Abs. Diff.



With the optimized cut off we calculate the full set of model metrics using the function HMeasure from hmeasure library.

```
# Boosting
measures.boosting.train <- HMeasure(prob.train$boosting.actual,
                                   prob.train$boosting.predicted,
                                   threshold = metricsByCutoff.test_boost$BestCut['Cut'])
measures.boosting.test  <- HMeasure(prob.test$boosting.actual,
                                   prob.test$boosting.predicted,
                                   threshold = metricsByCutoff.test_boost$BestCut['Cut'])

# join measures in a single data frame
measures <- t(bind_rows(measures.boosting.train$metrics,
                        measures.boosting.test$metrics
                        )) %>% as_tibble(., rownames = NA)

colnames(measures) <- c('boosting - train', 'boosting - test')

measures$metric = rownames(measures)

measures <- dplyr::select(measures, metric, everything())
```

Below are the metrics on the train and test dataset:

```
kable(measures, row.names = FALSE)
```

metric	boosting - train	boosting - test
H	0.8386978	0.4670309
Gini	0.9729990	0.7059218
AUC	0.9864995	0.8529609
AUCH	0.9880378	0.8803352
KS	0.9077926	0.6332961

metric	boosting - train	boosting - test
MER	0.0397490	0.0980392
MWL	0.0175767	0.0788639
Spec.Sens95	0.9320843	0.4134078
Sens.Spec95	0.9215686	0.4400000
ER	0.1694561	0.2205882
Sens	1.0000000	0.8400000
Spec	0.8103044	0.7709497
Precision	0.3863636	0.3387097
Recall	1.0000000	0.8400000
TPR	1.0000000	0.8400000
FPR	0.1896956	0.2290503
F	0.5573770	0.4827586
Youden	0.8103044	0.6109497
TP	51.0000000	21.0000000
FP	81.0000000	41.0000000
TN	346.0000000	138.0000000
FN	0.0000000	4.0000000

26.2 Evaluating classification performance

This model delivered the best discrimination f all models done here (KS = 0.63)! We will compare how it performed against other classes of models in the Final Report session.

Below the confusion matrix and general performance of the model:

```
# accuracy metrics -----
# boosting
accuracy(score = prob.test$boosting.predicted,
          actual = prob.test$boosting.actual,
          threshold = metricsByCutoff.test_boost[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      21      4
## actual.0      41     138
## [1] "-----"
## [1] "Model General Accuracy of: 77.94%"
## [1] "True Positive Rate of      : 84%"
```

We finally look at the score distribution charts to check how well the model is able to discriminate Defaulters and Non-Defaulters.

```

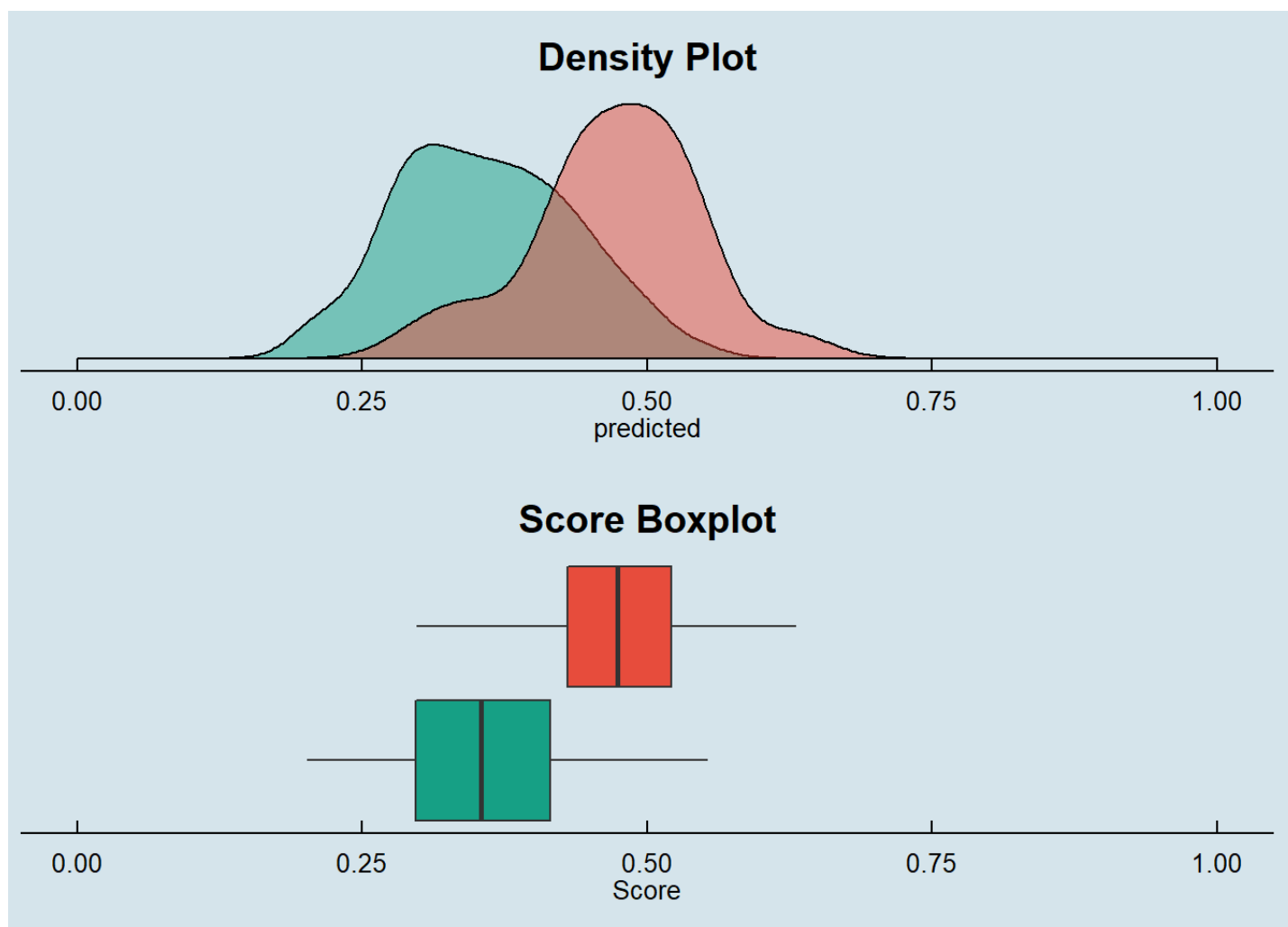
p1 <- Score_Histograms(prob.test,
  prob.test$boosting.predicted,
  prob.test$boosting.actual,
  'Density Plot') + theme(axis.title.y = element_blank())

p2 <- Score_Boxplot(prob.test,
  prob.test$boosting.predicted,
  prob.test$boosting.actual,
  'Score Boxplot')

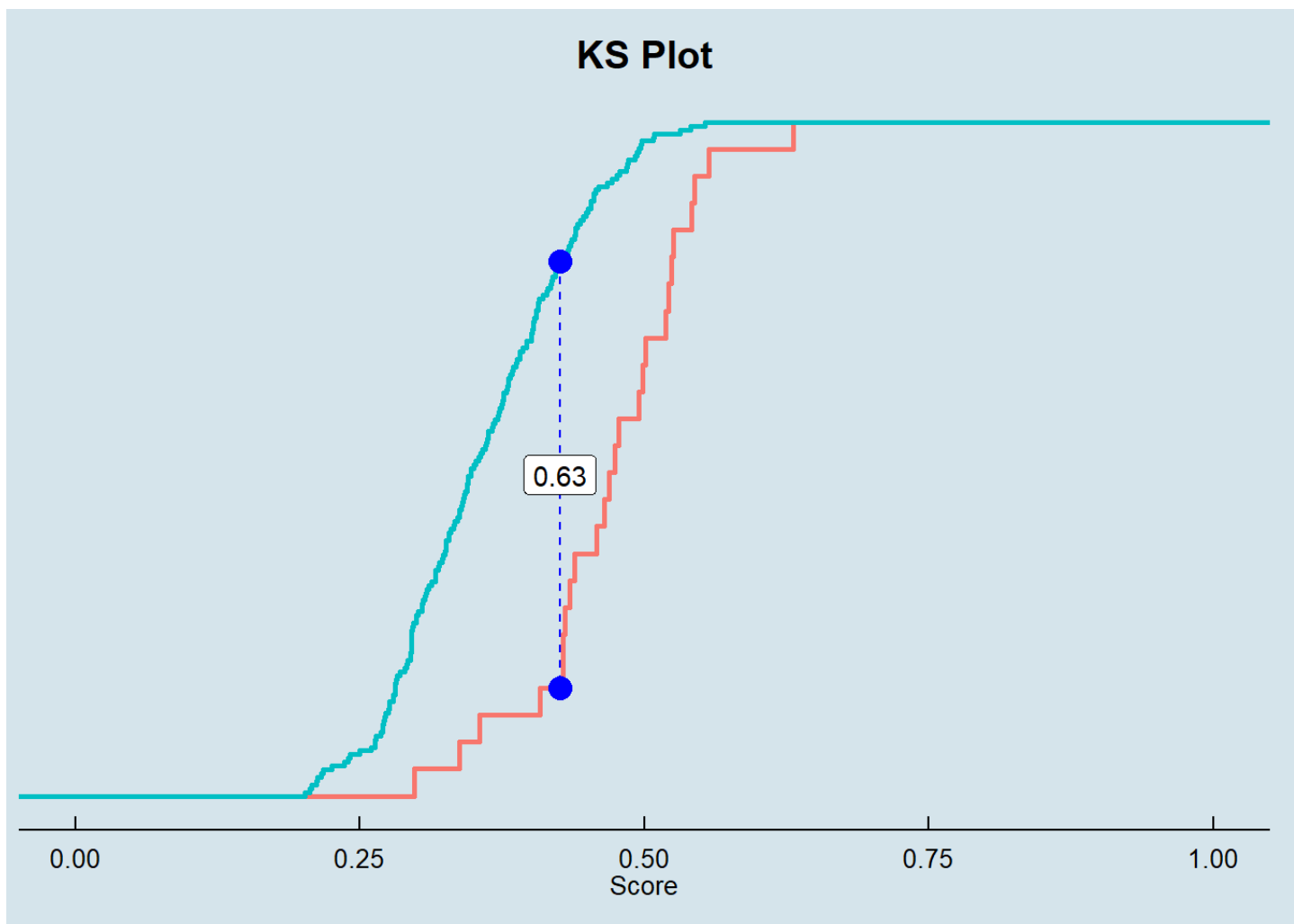
p3 <- KS_Plot(prob.test$boosting.predicted[prob.test$boosting.actual == 0],
  prob.test$boosting.predicted[prob.test$boosting.actual == 1],
  'KS Plot') + theme(axis.title.y = element_blank(),
  axis.text.y = element_blank())

ggarrange(p1, p2, nrow = 2)

```



p3



By the score density we can see that the boosting model is not as narrow as the decision tree on the scores it assigns to the observations.

The box plot can show us a clear discrimination between Defaulters and Non-Defaulters.

The KS metric .63 is considered good for this classification model.

In the Final Report session, we will look more closely on the AUC and Gini metrics by plotting the ROC curve and comparing against other models.

Stay with us!!!

27 Objective

The goal of this session is to fit a Random Forest model on Loan data aiming to predict the probability of delinquency for each contract.

Random forest, in essence, consists of a large set of individual decision trees operating as an ensemble. Therefore, each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is: A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

— — Tony Yiu

The Random Forest algorithm is a supervised algorithm that, even though it can be used for regression purposes, it was initially conceived as a classification tool.

The method follows the same concept as a decision tree but with the power of the crowd. So, basically, instead of using one big, deep and complex tree, the method relies on multiple randomly different (in multiple ways) trees voting for a class. This method usually perform better than one very well trained tree even if, individually, the trees are not as good classifiers, thus referring to the forementioned wisdom of the crowds.

Random forest thrives even in scenarios, when there is an abundance of chaos, i.e. many predictors. It's hard to know which predictor is important and which is not. All the other traditional statistical techniques might fail or struggle when we have an incredibly high number of independent variables.

— — Pranov Mishra

Because of the voting system inherent to the method it is often said to be democratic algorithm.

Now, if you think for a second, this is the way direct democracy works: each voter has access to a subset of the information and only sees that subset from a particular perspective (their own unique perspective). By using a majority vote, we are actually implementing a Random Forest.

— — Pablo Duboue

28 Modeling

28.1 Dataset preparation

Using the vanilla transaction dataset, we calculated several derived variables for each account as described in the Data Preparation session.

This dataset is joined with Loan, Client, Credit Card, District, Account and Account Balance tables.

We ended up having a dataset with **118 variables**.

```
loan_dataset_rf <- source_dataset

kable(tibble(variables = names(loan_dataset_rf)))
```

variables

y_loan_defaulter

x_account_balance

x_average_salary

x_avg_account_balance

x_card_age_month

x_card_type_classic

x_card_type_gold

x_card_type_junior

x_client_age

x_client_gender_male

x_district_name_Benesov

x_district_name_Beroun

x_district_name_Blansko

x_district_name_Breclav

variables

x_district_name_Brno_mesto

x_district_name_Brno_venkov

x_district_name_Bruntal

x_district_name_Ceska_Lipa

x_district_name_Ceske_Budejovice

x_district_name_Cesky_Krumlov

x_district_name_Cheb

x_district_name_Chomutov

x_district_name_Chrudim

x_district_name_Decin

x_district_name_Domazlice

x_district_name_Frydek_Mistek

x_district_name_Havlickuv_Brod

x_district_name_Hl.m._Praha

x_district_name_Hodonin

x_district_name_Hradec_Kralove

x_district_name_Jablonec_n._Nisou

x_district_name_Jesenik

x_district_name_Jicin

x_district_name_Jihlava

x_district_name_Jindrichuv_Hradec

x_district_name_Karlovy_Vary

x_district_name_Karvina

x_district_name_Kladno

x_district_name_Klatovy

x_district_name_Kolin

x_district_name_Kromeriz

x_district_name_Kutna_Hora

x_district_name_Liberec

x_district_name_Litomerice

x_district_name_Louny

x_district_name_Melnik

x_district_name_Mlada_Boleslav

x_district_name_Most

x_district_name_Nachod

x_district_name_Novy_Jicin

x_district_name_Nymburk

variables

x_district_name_Olomouc
x_district_name_Opava
x_district_name_Ostrava_mesto
x_district_name_Pardubice
x_district_name_Pelhrimov
x_district_name_Pisek
x_district_name_Plzen_jih
x_district_name_Plzen_mesto
x_district_name_Plzen_sever
x_district_name_Prachatice
x_district_name_Praha_vychod
x_district_name_Praha_zapad
x_district_name_Prerov
x_district_name_Pribram
x_district_name_Prostejov
x_district_name_Rakovnik
x_district_name_Rokycany
x_district_name_Rychnov_nad_Kneznou
x_district_name_Semily
x_district_name_Strakonice
x_district_name_Sumperk
x_district_name_Svitavy
x_district_name_Tabor
x_district_name_Tachov
x_district_name_Teplice
x_district_name_Trebic
x_district_name_Trutnov
x_district_name_Uherske_Hradiste
x_district_name_Usti_nad_Labem
x_district_name_Usti_nad_Orlici
x_district_name_Vsetin
x_district_name_Vyskov
x_district_name_Zdar_nad_Sazavou
x_district_name_Zlin
x_district_name_Znojmo
x_last_transaction_age_days
x_loan_amount

variables

x_loan_duration

x_loan_payments

x_no_of_cities

x_no_of_committed_crimes_1995

x_no_of_committed_crimes_1996

x_no_of_entrepreneurs_per_1000_inhabitants

x_no_of_inhabitants

x_no_of_municip_2000_to_9999

x_no_of_municip_500_to_1999

x_no_of_municip_greater_10000

x_no_of_municip_inhabitants_less_499

x_prop_household

x_prop_insurance_payment

x_prop_interest_credited

x_prop_loan_payment

x_prop_old_age_pension

x_prop_other

x_prop_statement

x_ratio_of_urban_inhabitants

x_region_central_Bohemia

x_region_east_Bohemia

x_region_north_Bohemia

x_region_north_Moravia

x_region_Prague

x_region_south_Bohemia

x_region_south_Moravia

x_transaction_amount

x_transaction_count

x_unemployment_rate_1995

x_unemployment_rate_1996

28.2 Variable selection

One advantage of Random Forest models is that it does not require heavy feature engineering.

We will only remove **x_prop_old_age_pension** that we know beforehand to have no variance in the dataset.

Mainly because of the randomness in variable selection of random forest algorithm, this model is not sensible to outliers, missing values and multicollinearity.

```
loan_dataset_rf <- dplyr::select(loan_dataset_rf, -x_prop_old_age_pension)
```

28.3 Sample split into Test and Training Data

The available data in Loan Dataset is split into Train and Testing data on the following proportion:

- **Train Dataset** (70% 478 obs);
- **Test Dataset** (30% 204 obs).

We are selecting exact the same samples for all models to allow comparison between then.

```
SplitDataset <- source_train_test_dataset
data.train_rf <- SplitDataset$data.train
data.test_rf <- SplitDataset$data.test

kable(SplitDataset$event.proportion)
```

scope	0	1
full dataset	0.8885630	0.1114370
train dataset	0.8933054	0.1066946
test dataset	0.8774510	0.1225490

```
loan_dataset_rf$y_loan_defaulter <- as.factor(loan_dataset_rf$y_loan_defaulter)
data.train_rf$y_loan_defaulter <- as.factor(data.train_rf$y_loan_defaulter)
data.test_rf$y_loan_defaulter <- as.factor(data.test_rf$y_loan_defaulter)

data.train_rf <- dplyr::select(data.train_rf, names(loan_dataset_rf))
data.test_rf <- dplyr::select(data.test_rf, names(loan_dataset_rf))
```

Both datasets kept the same proportion for the explained variable at about 11%.

29 Selecting the best parameters values for the Random Forest

The R community is a one of R's best features. There are many community members doing awesome improvements on existent libraries as well as sharing and spreading knowledge to the four corners of the Earth (if you still think the Earth is flat and square as I do).

Th algorithm selected from **randomForest package** (Please, see **References** (https://ldaniel.github.io/Predictive-Analytics/09_references.html) to reach out this amazing package) have two main parameters for random forest algorithm tuning: **mtry** representing the number of variables randomly sampled as candidates at each split (or the size of the trees) and **ntree** representing the number of trees to grow (or the size of the forest).

Another great package for modeling is caret (*seriously*, check this one out in **References** (https://ldaniel.github.io/Predictive-Analytics/09_references.html), it is fantastic in so many ways!) and, for the sake of our sanity, it has already implemented a good method for randomForest's package parameter tuning (cheers to them!). This, with the expanded grid search (also from caret package), provides a high performance, friendly and intelligent way of testing parameters. Unfortunately, it is not a perfect world and the implemented method only tune **mtry** parameter.

In order to have both parameters tuned using expanded grid search we need to extend caret's methodology, creating a custom method for parameter tuning and, for that, even under the risk of being repetitive, we must say: we can be saved by the grace of of R's community!

There are a lot of implementations accross the internet, however, a good example (and apparently an original one) is provided by Jason Brownlee, who created this customized function supports **mtry** AND **ntree** parameter tuning together. Please, see **References** (https://ldaniel.github.io/Predictive-Analytics/09_references.html) page for Jason's credits (cheers to him!).

Below is his implementation to be further used in our RF training.

```

customRF <- list(type = "Classification", library = "randomForest", loop = NULL)

customRF$parameters <- data.frame(parameter = c("mtry", "ntree"),
                                   class = rep("numeric", 2),
                                   label = c("mtry", "ntree"))

customRF$grid <- function(x, y, len = NULL, search = "grid") {}

customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
}

customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
  predict(modelFit, newdata)

customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
  predict(modelFit, newdata, type = "prob")

customRF$sort <- function(x) x[order(x[,1]),]

customRF$levels <- function(x) x$classes

```

Moving further, we can now set up our expanded grid search using our customized RF train method to test multiple parameters and its different combinations. After that, caret's method will automatically select best model according to the metric we defined (in our case, accuracy). Moreover, to provide a reliable method of training to avoid (as well as we can) overfitting, we used repeated k-fold cross validation as train control - also provided by caret's package (we warned you, this is, indeed, an amazing package!).

```

control <- trainControl(method="repeatedcv",
                        number=5,
                        repeats=3,
                        verboseIter = TRUE,
                        allowParallel = TRUE)

tuneparam <- expand.grid(.mtry=c(5, 25, 50, 75, 85, 100, 115, 125, 150, 175, 200),
                        .ntree=c(1000, 3000, 5000, 7000, 9000, 10000))

evalmetric <- "Accuracy"

set.seed(12345)

ini <- Sys.time()
cat(paste0("\nStarted RF training at: ", ini, " ...\n\n"))

rf.full <- train(y_loan_defaulter ~ .,
                data=data.train_rf,
                method=customRF,
                metric=evalmetric,
                tuneGrid=tuneparam,
                trControl=control,
                importance=TRUE)

elapsedTime <- difftime(Sys.time(), ini, units = "auto")
cat(paste0("\n\nFinished RF training. Total time taken: ", round(elapsedTime, 2), " ", units(elapsedTime)))

summary(rf.full)
plot(rf.full)

```

After some cups of coffee (and maybe some time spent on your preferred streaming provider), we have the training finished. And the winners are....:

- **mtry = 85**
- **ntree = 3000**

Last but not least, we saved the final model results on disk to be quickly consumed when necessary.

```
saveRDS(rf.full, "./models/random_forest.rds")
```

So, to save time, we only have to load the fitted model saved on disk.

```
rf.full <- readRDS("./models/random_forest.rds")
```

30 Interpreting model output

For this model the four principal vars are:

- **x_prop_interest_credited**
- **x_account_balance**
- **x_avg_account_balance**
- **x_loan_amount**

The result is similar to the Logistic Regression, Decision Tree and Boosting models.

31 Evaluating the model performance

Here we will perform basically the same steps we did in the Logistic Regression, Decision Tree and Boosting models.

A comparison against all the models will be provided in the Final Report session of this exercise.

We started this step by making predictions using our model on the X's variables in our Train and Test datasets.

```
## making predictions for each model and consolidating in a single data frame

prob.full = list()
prob.train = list()
prob.test = list()

prob.full$randomforest.actual <- loan_dataset_rf$y_loan_defaulter
prob.full$randomforest.predicted <- predict(rf.full, newdata = loan_dataset_rf,
                                             type = "prob")[,2]

prob.train$randomforest.actual <- data.train_rf$y_loan_defaulter
prob.train$randomforest.predicted <- predict(rf.full, newdata = data.train_rf,
                                             type = "prob")[,2]

prob.test$randomforest.actual <- data.test_rf$y_loan_defaulter
prob.test$randomforest.predicted <- predict(rf.full, newdata = data.test_rf,
                                             type = "prob")[,2]

prob.full <- prob.full %>% as_tibble()
prob.train <- prob.train %>% as_tibble()
prob.test <- prob.test %>% as_tibble()
```

31.1 Getting Performance Measures

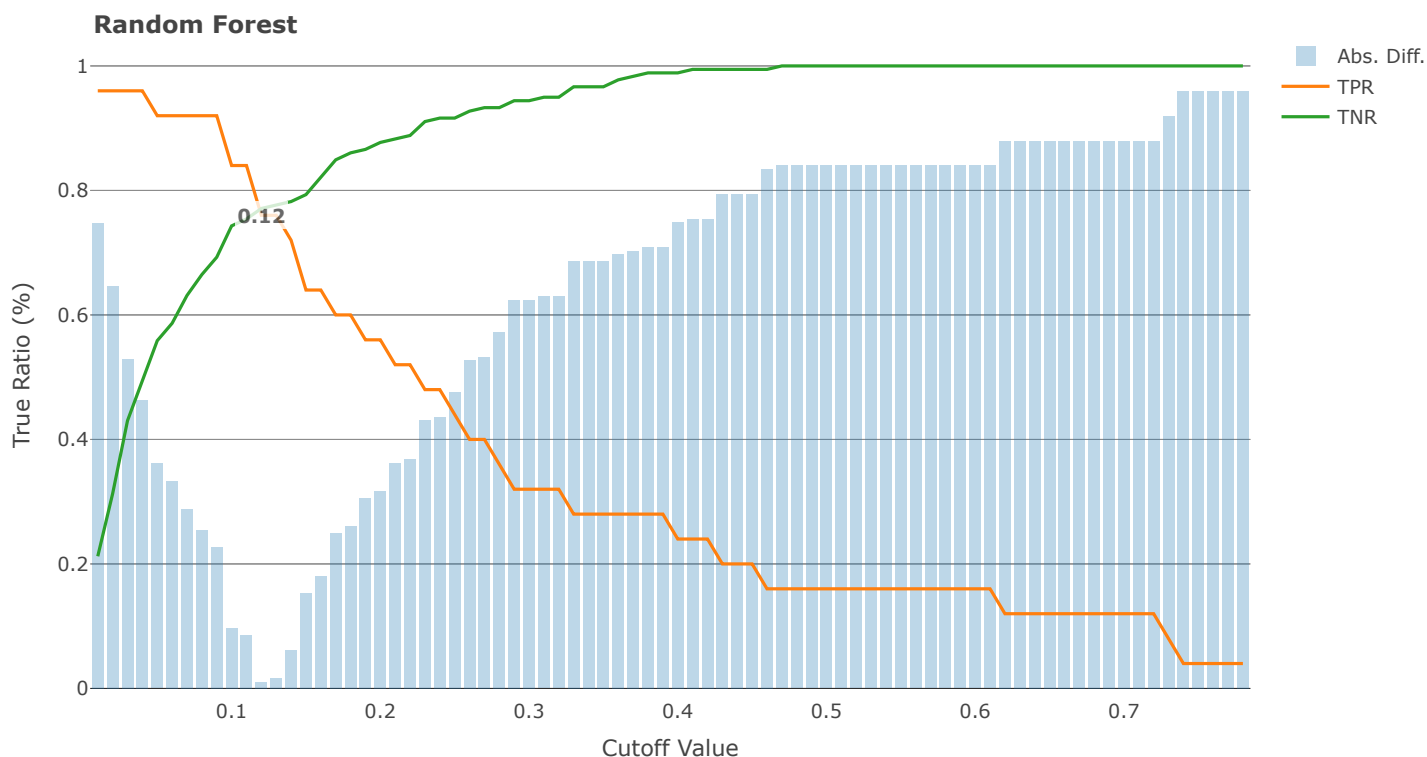
To calculate the performance measures, derived from the confusion matrix, we need to find the score cut off that best split our test dataset into Defaulters and Non-Defaulters.

In this exercise we decide to not prioritize the accuracy on predicting Defaulters and Non-Defaulters, therefore we are looking for the score cut off that best predict each class equally.

With the returned object from this function we can plot the comparison between TPR (True Positive Rate) and TNR (True Negative Rate) to find the best cut off.

```
## getting measures -----
metricsByCutoff.test_randomforest <- modelMetrics(prob.test$randomforest.actual,
                                                    prob.test$randomforest.predicted,
                                                    plot_title = 'Random Forest')

metricsByCutoff.test_randomforest$Plot
```



With the optimized cut off we calculate the full set of model metrics using the function HMeasure from hmeasure library (another very good package! Don't forget to check for our **references** (https://ldaniel.github.io/Predictive-Analytics/09_references.html)).

```
# Random Forest
measures.randomforest.train <- HMeasure(prob.train$randomforest.actual,
                                         prob.train$randomforest.predicted,
                                         threshold = metricsByCutoff.test_randomforest$BestCut[ 'Cut' ])
measures.randomforest.test  <- HMeasure(prob.test$randomforest.actual,
                                         prob.test$randomforest.predicted,
                                         threshold = metricsByCutoff.test_randomforest$BestCut[ 'Cut' ])

# join measures in a single data frame
measures <- t(bind_rows(measures.randomforest.train$metrics,
                        measures.randomforest.test$metrics
                        )) %>% as_tibble(., rownames = NA)

colnames(measures) <- c('random forest - train', 'random forest - test')

measures$metric = rownames(measures)

measures <- dplyr::select(measures, metric, everything())
```

Below are the metrics on the train and test dataset:

```
kable(measures, row.names = FALSE)
```

metric	random forest - train	random forest - test
H	1.0000000	0.4248541
Gini	1.0000000	0.6808939
AUC	1.0000000	0.8404469
AUCH	1.0000000	0.8654749
KS	1.0000000	0.6239106
MER	0.0000000	0.0980392
MWL	0.0000000	0.0808824
Spec.Sens95	1.0000000	0.5139665
Sens.Spec95	1.0000000	0.3200000
ER	0.0460251	0.2303922
Sens	1.0000000	0.7600000
Spec	0.9484778	0.7709497
Precision	0.6986301	0.3166667
Recall	1.0000000	0.7600000
TPR	1.0000000	0.7600000
FPR	0.0515222	0.2290503
F	0.8225806	0.4470588
Youden	0.9484778	0.5309497
TP	51.0000000	19.0000000
FP	22.0000000	41.0000000

metric	random forest - train	random forest - test
TN	405.0000000	138.0000000
FN	0.0000000	6.0000000

Our Random Forest model clearly overfitted (as we can see by most of metrics, AUC of 1.0 in train set and 0.84 in test set, for example). This happened mainly because of the size of our dataset.

31.2 Evaluating classification performance

This model delivered an amazing result but not the best one (beaten by the boosting model). We have a full session to compare how it performed against other models in the Final Report session. But wait, you should't hurry, there are still some steps to evaluate RF algorithm.

Below the confusion matrix and general performance of the model:

```
# accuracy metrics -----
# random forest
accuracy(score = prob.test$randomforest.predicted,
         actual = prob.test$randomforest.actual,
         threshold = metricsByCutoff.test_randomforest[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----
## actual.1      19      6
## actual.0      41     138
## [1] "-----"
## [1] "Model General Accuracy of: 76.96%"
## [1] "True Positive Rate of : 76%"
```

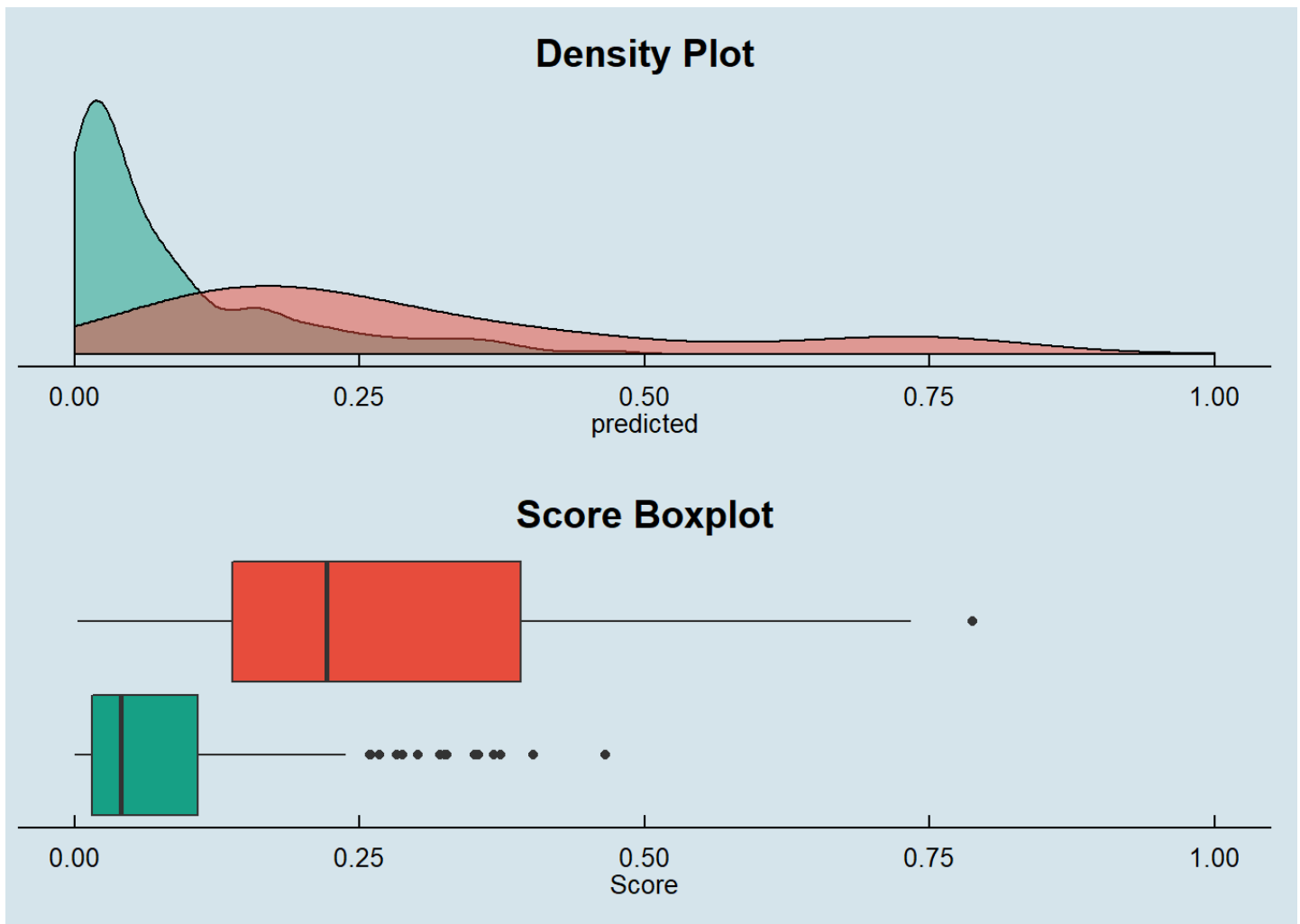
We finally look at the score distribution charts to check how well the model is able to discriminate Defaulters and Non-Defaulters.

```
p1 <- Score_Histograms(prob.test,
                      prob.test$randomforest.predicted,
                      prob.test$randomforest.actual,
                      'Density Plot') + theme(axis.title.y = element_blank())

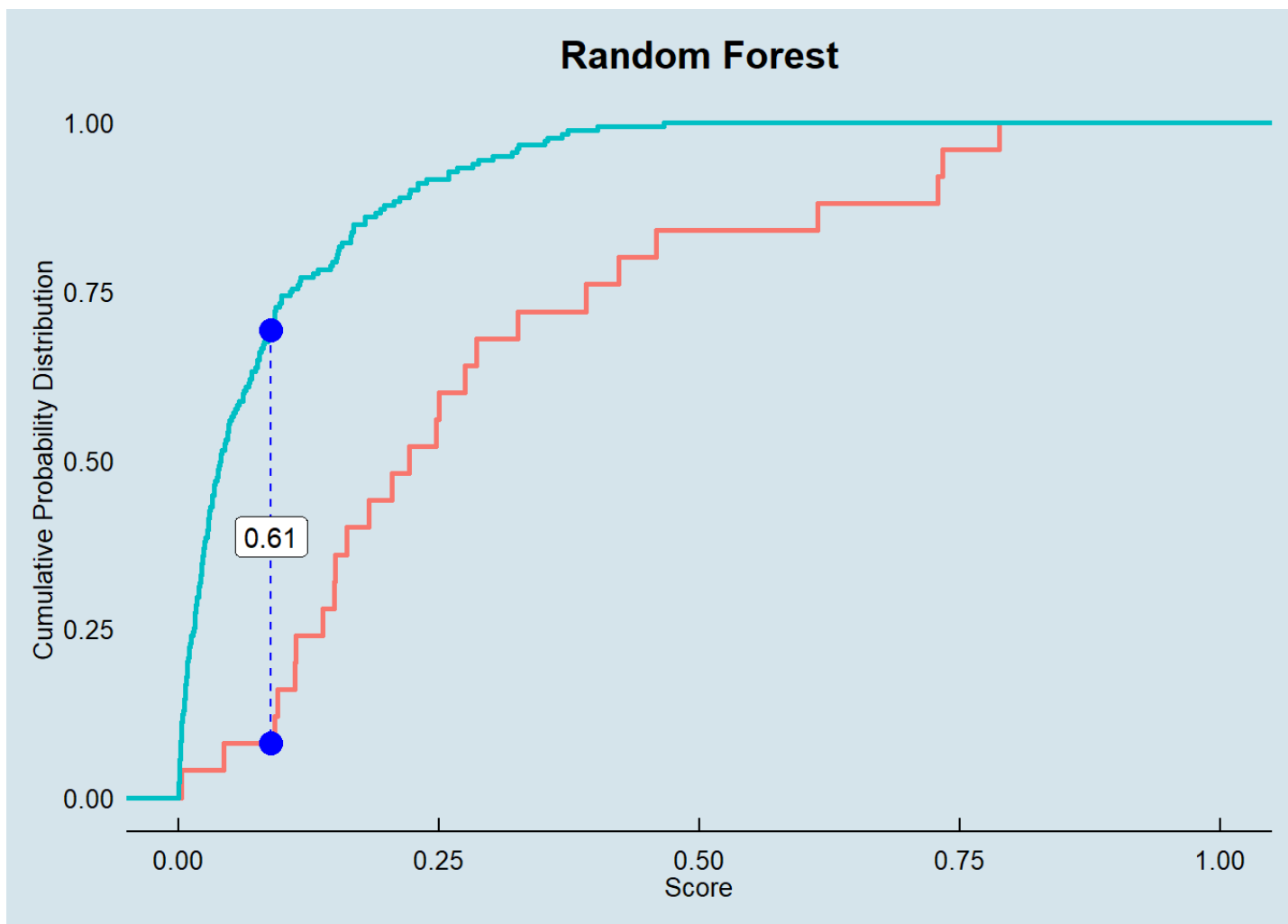
p2 <- Score_Boxplot(prob.test,
                   prob.test$randomforest.predicted,
                   prob.test$randomforest.actual,
                   'Score Boxplot')

p3 <- KS_Plot(prob.test$randomforest.predicted[prob.test$randomforest.actual == 0],
              prob.test$randomforest.predicted[prob.test$randomforest.actual == 1],
              'Random Forest')

ggarrange(p1, p2, nrow = 2)
```



p3



By the score density we see that our Random Forest model provides a narrow and precise discrimination around defaulters.

The box plot also shows a clear discrimination between Defaulters and Non-Defaulters.

Finally, the KS metric is also presented a good result for a reliable classification purpose.

In the Final Report session, we will look more closely on the AUC and Gini metrics by plotting the ROC curve and comparing against other models.

More to come in the final report!!!

32 Objective

The goal of this session is to compare the performance of all the models created during this exercise.

Here we will expand the analysis done individually in each model to compare how it is performing against each other.

33 Model evaluation

33.1 Getting the predicted score from each model

We will start this task by consolidating all the actual and predicted score for all models in a single data frame for the full, train and test datasets.

```

## making predictions for each model and consolidating in a single data frame
prob.full <- list()
prob.train <- list()
prob.test <- list()

# getting predicted and actual values in the full dataset
prob.full$logistic.actual <- loan_dataset_logistic$y_loan_defaulter
prob.full$logistic.predicted <- predict(logistic.step, type = "response",
                                       newdata = loan_dataset_logistic)
prob.full$decision.tree.actual <- loan_dataset_DT$y_loan_defaulter
prob.full$decision.tree.predicted <- predict(tree.prune, type = "prob",
                                             newdata = loan_dataset_DT)[, 2]
prob.full$boosting.actual <- loan_dataset_boost$y_loan_defaulter
prob.full$boosting.predicted <- predict.boosting(boost,
                                                  newdata = loan_dataset_boost)$prob[, 2]
prob.full$random.forest.actual <- loan_dataset_rf$y_loan_defaulter
prob.full$random.forest.predicted <- predict(rf.full, type = "prob",
                                             newdata = loan_dataset_rf)[, 2]

# getting predicted and actual values in the train dataset
prob.train$logistic.actual <- data.train_logistic$y_loan_defaulter
prob.train$logistic.predicted <- predict(logistic.step, type = "response",
                                       newdata = data.train_logistic)
prob.train$decision.tree.actual <- data.train_DT$y_loan_defaulter
prob.train$decision.tree.predicted <- predict(tree.prune, type = "prob",
                                             newdata = data.train_DT)[, 2]
prob.train$boosting.actual <- data.train_boost$y_loan_defaulter
prob.train$boosting.predicted <- predict.boosting(boost, newdata =
                                                  data.train_boost)$prob[, 2]
prob.train$random.forest.actual <- data.train_rf$y_loan_defaulter
prob.train$random.forest.predicted <- predict(rf.full, type = "prob",
                                             newdata = data.train_rf)[, 2]

# getting predicted and actual values in the test dataset
prob.test$logistic.actual <- data.test_logistic$y_loan_defaulter
prob.test$logistic.predicted <- predict(logistic.step, type = "response",
                                       newdata = data.test_logistic)
prob.test$decision.tree.actual <- data.test_DT$y_loan_defaulter
prob.test$decision.tree.predicted <- predict(tree.prune, type = "prob",
                                             newdata = data.test_DT)[, 2]
prob.test$boosting.actual <- data.test_boost$y_loan_defaulter
prob.test$boosting.predicted <- predict.boosting(boost, newdata =
                                                  data.test_boost)$prob[, 2]
prob.test$random.forest.actual <- data.test_rf$y_loan_defaulter
prob.test$random.forest.predicted <- predict(rf.full, type = "prob",
                                             newdata = data.test_rf)[, 2]

# converting lists into tibble
prob.full <- prob.full %>% as_tibble()
prob.train <- prob.train %>% as_tibble()
prob.test <- prob.test %>% as_tibble()

```

33.2 Getting performance measures for each model.

To calculate the performance measures, derived from the confusion matrix, of each model we need to find the score cut off that best split our test dataset into Defaulters and Non-Defaulters.

In this exercise we decide to not prioritize the accuracy on predicting Defaulters and Non-Defaulters, therefore we are looking for the score cut off that best predict each class equally.

We will use the custom functions described in Auxiliary metrics functions topic in the Data Preparation session of this site.

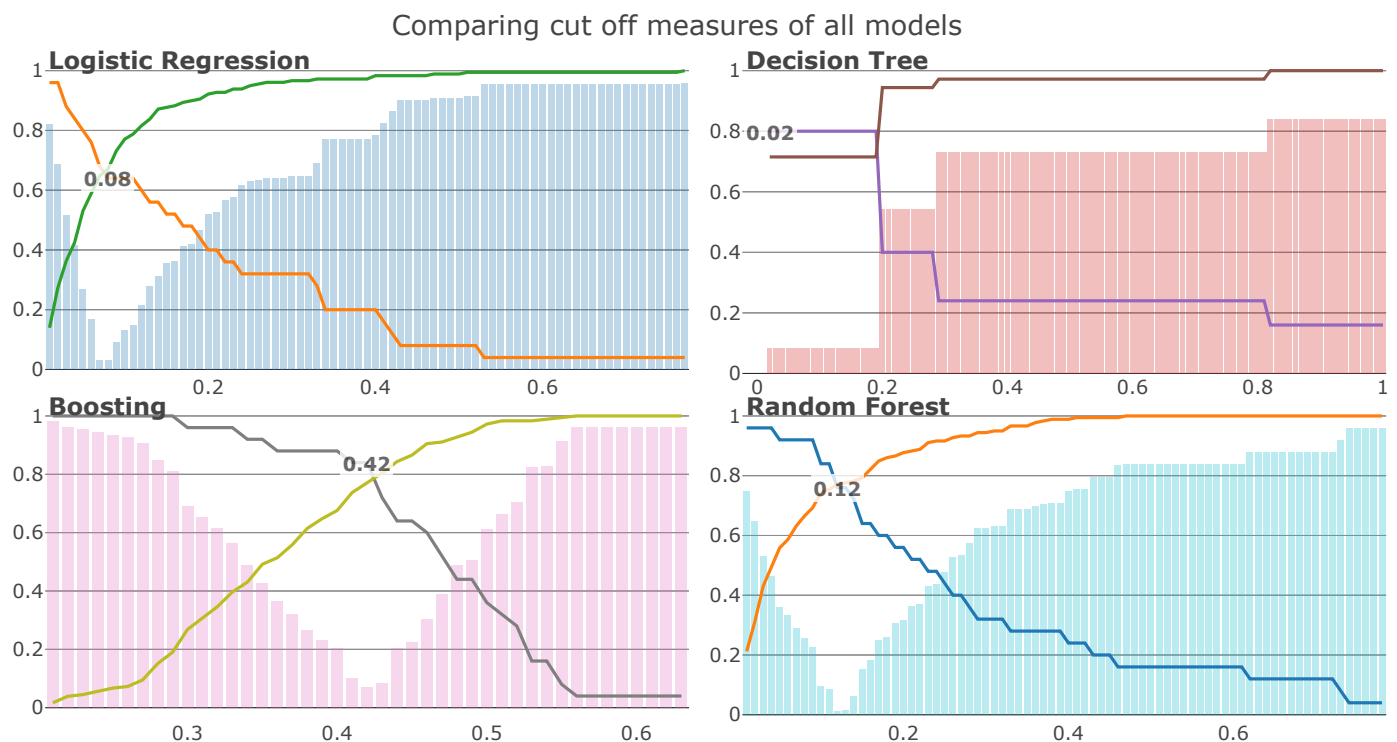
With the returned object from these functions we can plot the comparison between TPR (True Positive Rate) and TNR (True Negative Rate) to find the best cut off.

```
## getting cut off measures -----
```

```
metricsByCutoff.test_log <- modelMetrics(prob.test$logistic.actual,
                                          prob.test$logistic.predicted,
                                          plot_title = 'Logistic Regression')
metricsByCutoff.test_DT <- modelMetrics(prob.test$decision.tree.actual,
                                          prob.test$decision.tree.predicted,
                                          plot_title = 'Decision Tree')
metricsByCutoff.test_boost <- modelMetrics(prob.test$boosting.actual,
                                             prob.test$boosting.predicted,
                                             plot_title = 'Boosting')
metricsByCutoff.test_rf <- modelMetrics(prob.test$random.forest.actual,
                                         prob.test$random.forest.predicted,
                                         plot_title = 'Random Forest')
```

```
cutoffs <- plotly::subplot(metricsByCutoff.test_log$Plot,
                           metricsByCutoff.test_DT$Plot,
                           metricsByCutoff.test_boost$Plot,
                           metricsByCutoff.test_rf$Plot,
                           nrows = 2) %>% hide_legend() %>%
  layout(title="Comparing cut off measures of all models")
```

cutoffs



Having the best score cut off for each model we use **HMeasure()** function from **hmeasure** library to calculate the full set of metrics for classification methods.

```

# logistic regression
measures.logistic.train <- HMeasure(prob.train$logistic.actual,
                                   prob.train$logistic.predicted,
                                   threshold = metricsByCutoff.test_log$BestCut['Cut'])
measures.logistic.test <- HMeasure(prob.test$logistic.actual,
                                   prob.test$logistic.predicted,
                                   threshold = metricsByCutoff.test_log$BestCut['Cut'])

# decision tree
measures.decision.tree.train <- HMeasure(prob.train$decision.tree.actual,
                                          prob.train$decision.tree.predicted,
                                          threshold = metricsByCutoff.test_DT$BestCut['Cut'])
measures.decision.tree.test <- HMeasure(prob.test$decision.tree.actual,
                                          prob.test$decision.tree.predicted,
                                          threshold = metricsByCutoff.test_DT$BestCut['Cut'])

# boosting
measures.boosting.train <- HMeasure(prob.train$boosting.actual,
                                    prob.train$boosting.predicted,
                                    threshold = metricsByCutoff.test_boost$BestCut['Cut'])
measures.boosting.test <- HMeasure(prob.test$boosting.actual,
                                   prob.test$boosting.predicted,
                                   threshold = metricsByCutoff.test_boost$BestCut['Cut'])

# random forest
measures.random.forest.train <- HMeasure(prob.train$random.forest.actual,
                                          prob.train$random.forest.predicted,
                                          threshold = metricsByCutoff.test_rf$BestCut['Cut'])
measures.random.forest.test <- HMeasure(prob.test$random.forest.actual,
                                          prob.test$random.forest.predicted,
                                          threshold = metricsByCutoff.test_rf$BestCut['Cut'])

# join measures in a single data frame
measures <- t(bind_rows(measures.logistic.train$metrics,
                        measures.logistic.test$metrics,
                        measures.decision.tree.train$metrics,
                        measures.decision.tree.test$metrics,
                        measures.boosting.train$metrics,
                        measures.boosting.test$metrics,
                        measures.random.forest.train$metrics,
                        measures.random.forest.test$metrics
                      )) %>% as_tibble(., rownames = NA)

colnames(measures) <- c('logistic - train', 'logistic - test',
                        'decision.tree - train', 'decision.tree - test',
                        'boosting - train', 'boosting - test',
                        'random forest - train', 'random forest - test')

measures$metric = rownames(measures)

measures <- dplyr::select(measures, metric, everything())

```

Below are the metrics on the train dataset:

```
kable(dplyr::select(measures, contains('train')), row.names = TRUE)
```

logistic - train

decision.tree - train

boosting - train

random forest - train

	logistic - train	decision.tree - train	boosting - train	random forest - train
H	0.3612900	0.4815174	0.8386978	1.0000000
Gini	0.6641411	0.7575883	0.9729990	1.0000000
AUC	0.8320705	0.8787941	0.9864995	1.0000000
AUCH	0.8475456	0.8787941	0.9880378	1.0000000
KS	0.5492033	0.6522478	0.9077926	1.0000000
MER	0.0920502	0.0732218	0.0397490	0.0000000
MWL	0.0859316	0.0662891	0.0175767	0.0000000
Spec.Sens95	0.5339578	0.0878220	0.9320843	1.0000000
Sens.Spec95	0.3725490	0.8830450	0.9215686	1.0000000
ER	0.3347280	0.2489540	0.1694561	0.0460251
Sens	0.8431373	0.9215686	1.0000000	1.0000000
Spec	0.6440281	0.7306792	0.8103044	0.9484778
Precision	0.2205128	0.2901235	0.3863636	0.6986301
Recall	0.8431373	0.9215686	1.0000000	1.0000000
TPR	0.8431373	0.9215686	1.0000000	1.0000000
FPR	0.3559719	0.2693208	0.1896956	0.0515222
F	0.3495935	0.4413146	0.5573770	0.8225806
Youden	0.4871654	0.6522478	0.8103044	0.9484778
TP	43.0000000	47.0000000	51.0000000	51.0000000
FP	152.0000000	115.0000000	81.0000000	22.0000000
TN	275.0000000	312.0000000	346.0000000	405.0000000
FN	8.0000000	4.0000000	0.0000000	0.0000000

Below are the metrics on the test dataset:

```
kable(dplyr::select(measures, contains('test')), row.names = TRUE)
```

	logistic - test	decision.tree - test	boosting - test	random forest - test
H	0.3070793	0.3299682	0.4670309	0.4248541
Gini	0.5195531	0.5883799	0.7059218	0.6808939
AUC	0.7597765	0.7941899	0.8529609	0.8404469
AUCH	0.7915084	0.7966480	0.8803352	0.8654749
KS	0.4370950	0.5150838	0.6332961	0.6239106
MER	0.1078431	0.1029412	0.0980392	0.0980392
MWL	0.1210592	0.1042868	0.0788639	0.0808824
Spec.Sens95	0.3072626	0.3840782	0.4134078	0.5139665
Sens.Spec95	0.3200000	0.2736000	0.4400000	0.3200000
ER	0.3333333	0.2745098	0.2205882	0.2303922

	logistic - test	decision.tree - test	boosting - test	random forest - test
Sens	0.6400000	0.8000000	0.8400000	0.7600000
Spec	0.6703911	0.7150838	0.7709497	0.7709497
Precision	0.2133333	0.2816901	0.3387097	0.3166667
Recall	0.6400000	0.8000000	0.8400000	0.7600000
TPR	0.6400000	0.8000000	0.8400000	0.7600000
FPR	0.3296089	0.2849162	0.2290503	0.2290503
F	0.3200000	0.4166667	0.4827586	0.4470588
Youden	0.3103911	0.5150838	0.6109497	0.5309497
TP	16.0000000	20.0000000	21.0000000	19.0000000
FP	59.0000000	51.0000000	41.0000000	41.0000000
TN	120.0000000	128.0000000	138.0000000	138.0000000
FN	9.0000000	5.0000000	4.0000000	6.0000000

By looking at the metrics we can see that the models performed well in the train dataset (especially Boosting and Random Forest).

But on the test dataset, the models are performing the same, with a clear advantage for Boosting and Random forest.

This is going to be clearer when we look to the score distribution and the ROC curve of each model.

33.3 Evaluating performance of each model

In this session we interpret the set of metrics we got from above steps.

33.3.1 Density Plots

Here we look back to the score density plot produced by each model side by side.

Interesting enough to notice here how narrow the scores produced from Decision Tree model compared to the other models.

As discussed in the Decision Tree session this model is extremely limited for this dataset with a huge variance in the tree depending on the train data and hyper parameters tuning.

We are using our custom function **Score_Histograms** described in the Function topic of Data Preparation session of this site.


```

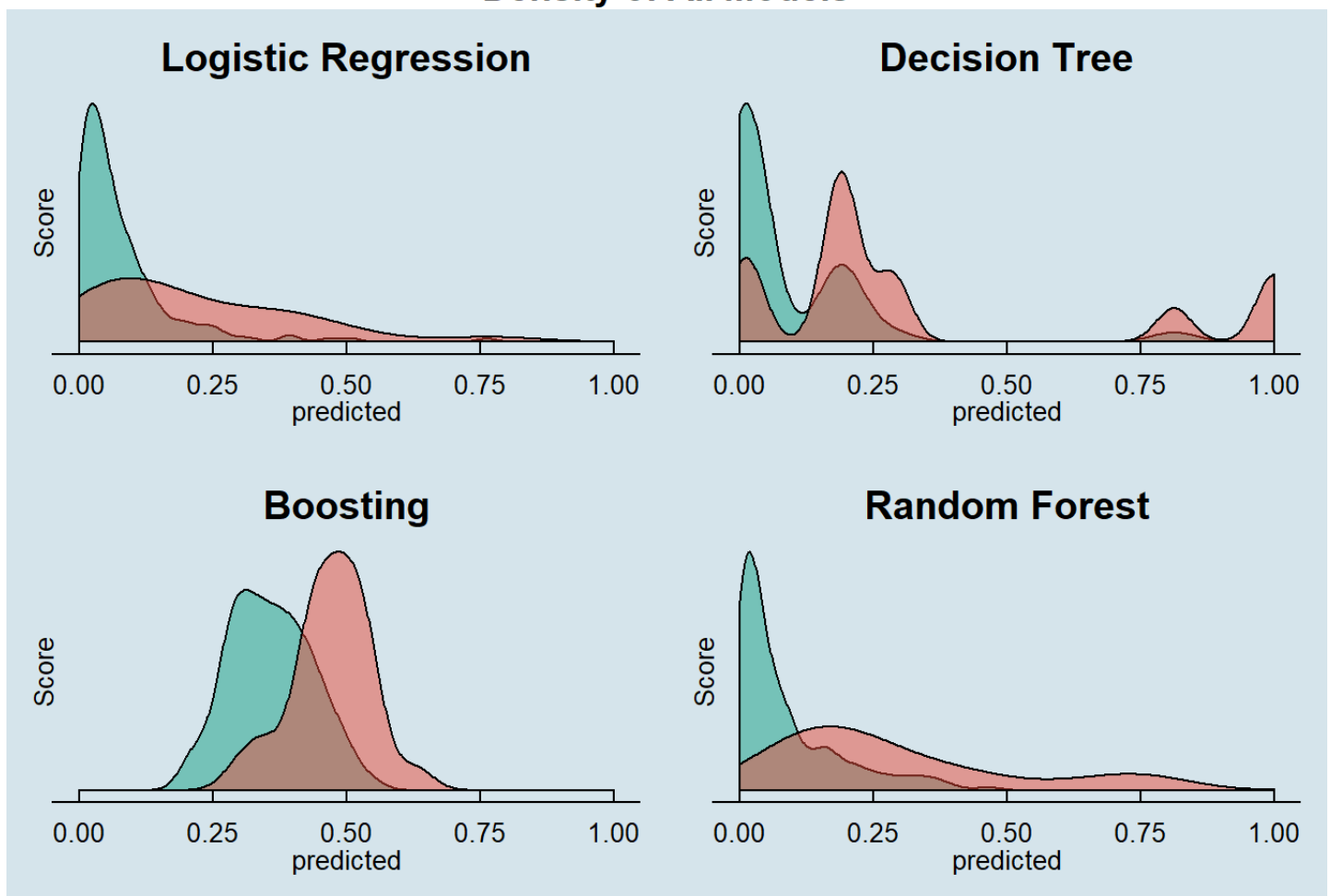
density_plots <- ggarrange(Score_Histograms(prob.test,
                                             prob.test$logistic.predicted,
                                             prob.test$logistic.actual,
                                             'Logistic Regression'),
                           Score_Histograms(prob.test,
                                             prob.test$decision.tree.predicted,
                                             prob.test$decision.tree.actual,
                                             'Decision Tree'),
                           Score_Histograms(prob.test,
                                             prob.test$boosting.predicted,
                                             prob.test$boosting.actual,
                                             'Boosting'),
                           Score_Histograms(prob.test,
                                             prob.test$random.forest.predicted,
                                             prob.test$random.forest.actual,
                                             'Random Forest'))

density_plots <- annotate_figure(density_plots,
                                top = text_grob("Density of All Models",
                                                color = "black", face = "bold",
                                                size = 14))

density_plots

```

Density of All Models



33.3.2 Score Boxplots

Let's now look at the score boxplots of each model side by side.

This plot is a great way to visualize how well each model discriminate Defaulters and Non-Defaulters.

Here we can see that Random Forest and Boosting have an edge on the classification power against the other models. Both of them have not presented interquartile overlap in their box plot, another important sign of discrimination power of the models.

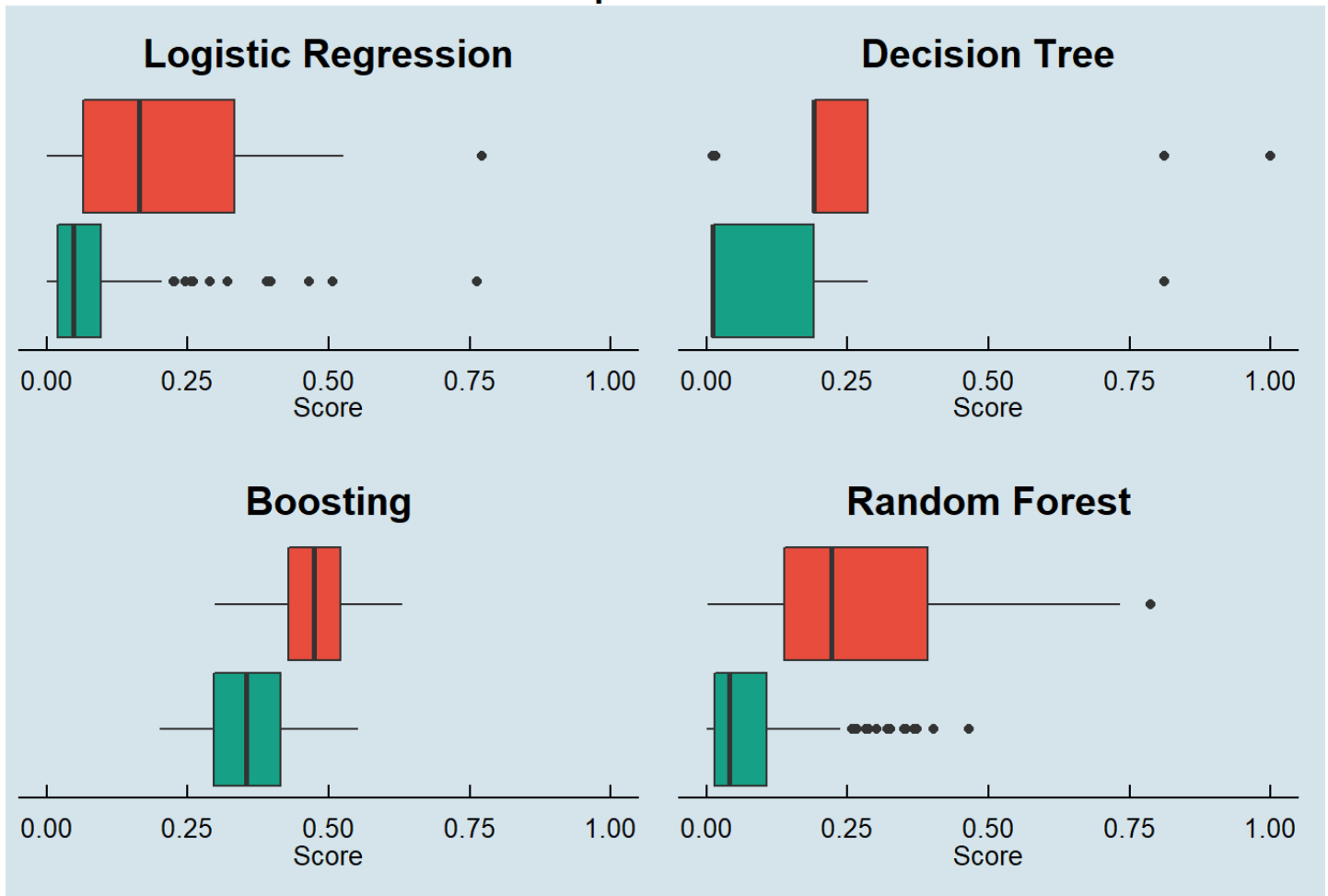
Our Decision Tree is right on the limit.

```
boxplots <- ggarrange(Score_Boxplot(prob.test,
                                   prob.test$logistic.predicted,
                                   prob.test$logistic.actual,
                                   'Logistic Regression'),
  Score_Boxplot(prob.test,
                 prob.test$decision.tree.predicted,
                 prob.test$decision.tree.actual,
                 'Decision Tree'),
  Score_Boxplot(prob.test,
                 prob.test$boosting.predicted,
                 prob.test$boosting.actual,
                 'Boosting'),
  Score_Boxplot(prob.test,
                 prob.test$random.forest.predicted,
                 prob.test$random.forest.actual,
                 'Random Forest'))

boxplots <- annotate_figure(boxplots,
  top = text_grob("Score Boxplots of All Models",
    color = "black", face = "bold",
    size = 14))

boxplots
```

Score Boxplots of All Models



33.3.3 KS Plots

Now we look to the custom KS plots of each model side by side.

The KS metric is the maximum distance between the cumulative distribution functions of two samples.

KS metric ranges from 0 to 1, 0 meaning that there is no discrimination at all and 1 meaning a full discrimination.

In this case we are comparing the cumulative distribution sample of Defaulters and Non-Defaulters.

The bigger the KS metric the better the model are to discriminate Defaulters to Non-Defaulters.

```

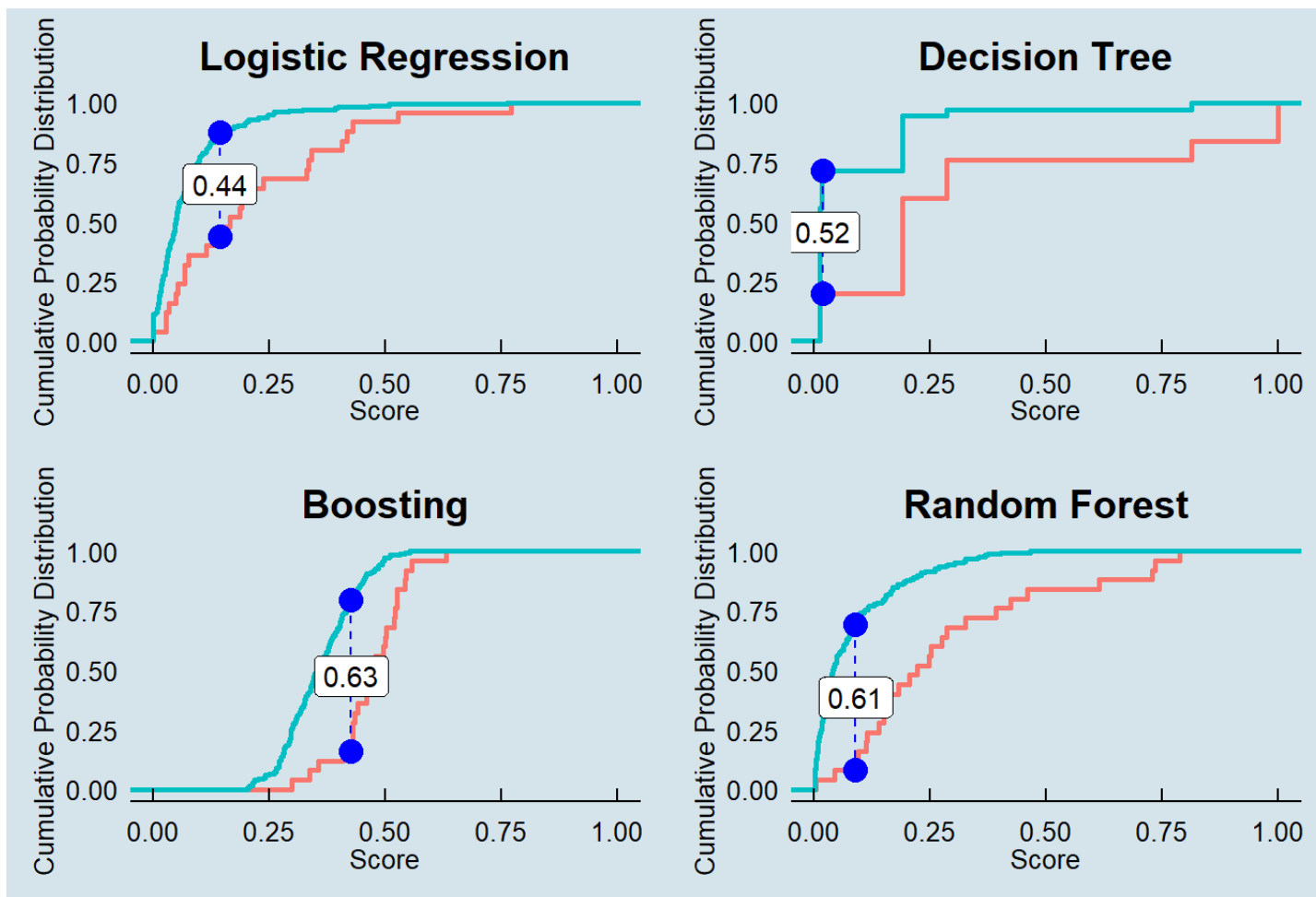
KS_plots <- ggarrange(
  KS_Plot(prob.test$logistic.predicted[prob.test$logistic.actual == 0],
    prob.test$logistic.predicted[prob.test$logistic.actual == 1],
    'Logistic Regression'),
  KS_Plot(prob.test$decision.tree.predicted[prob.test$decision.tree.actual == 0],
    prob.test$decision.tree.predicted[prob.test$decision.tree.actual == 1],
    'Decision Tree'),
  KS_Plot(prob.test$boosting.predicted[prob.test$boosting.actual == 0],
    prob.test$boosting.predicted[prob.test$boosting.actual == 1],
    'Boosting'),
  KS_Plot(prob.test$random.forest.predicted[prob.test$random.forest.actual == 0],
    prob.test$random.forest.predicted[prob.test$random.forest.actual == 1],
    'Random Forest'))

KS_plots <- annotate_figure(KS_plots,
  top = text_grob("KS Plots of All Models",
    color = "black", face = "bold",
    size = 14))

```

KS_plots

KS Plots of All Models



33.3.4 ROC Curve

Now we look to the ROC curve of our models.

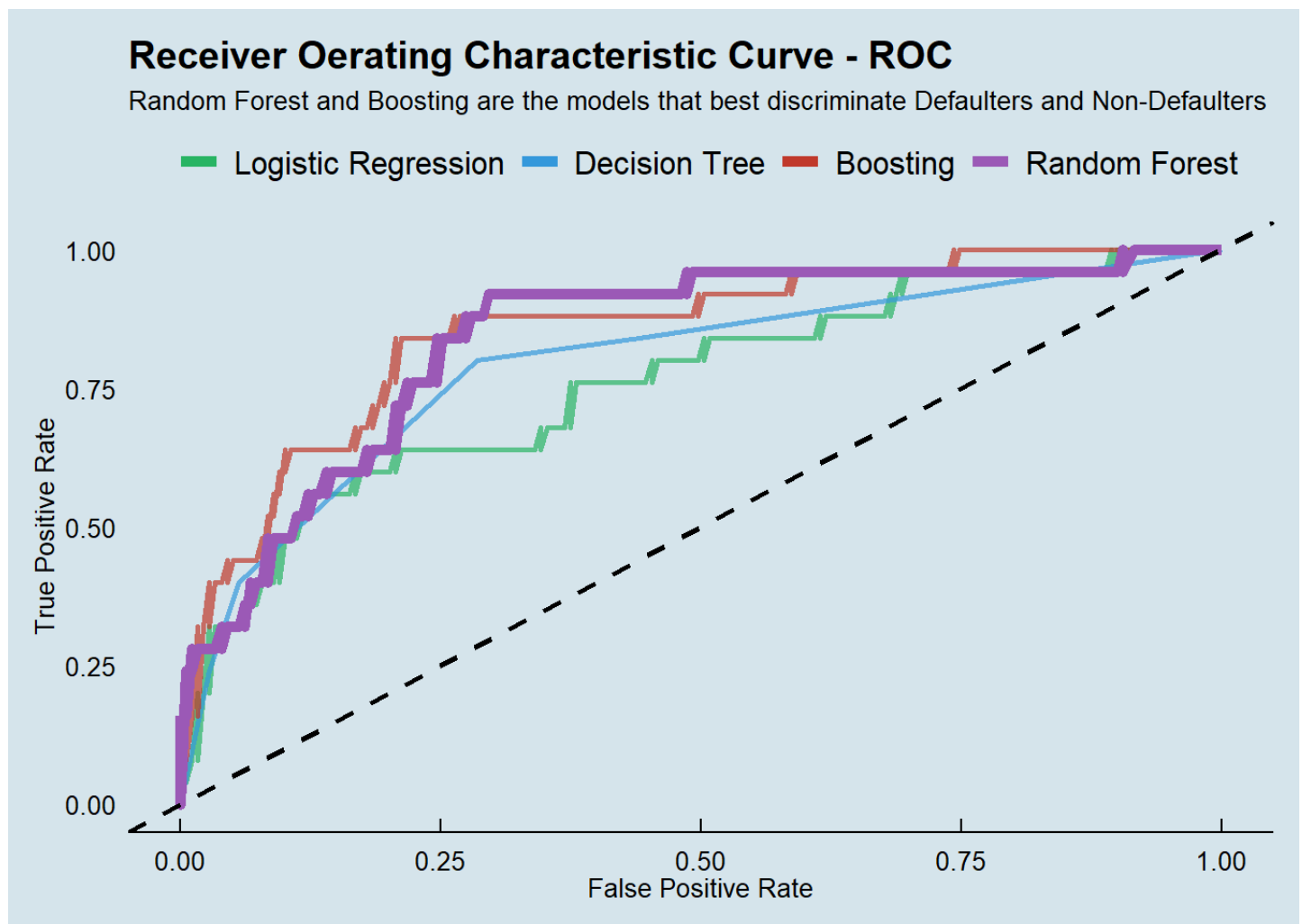
The Receiver Operating Characteristic Curve is a plot that shows the discrimination ability of a binary classifier model as its classification threshold changes.

We get this chart by plotting the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)** at different threshold settings.

The bigger the Area Under the Curve (AUC) the better the model is in classifying the observation.

```
ROC_test <- Plot_ROC(prob.test, smooth_opt = FALSE)

print(ROC_test)
```



33.3.5 Accuracy

We finally look at the Confusion Matrix at the best cut off of each model to get a sense of the accuracy we were able to get in this exercise.

We will use the custom functions described in Auxiliary metrics functions topic in the Data Preparation session of this site.

Boosting is the best model we got in this exercise following very closely by Random Forest.

```
# logistic regression
accuracy(score = prob.test$logistic.predicted,
         actual = prob.test$logistic.actual,
         threshold = metricsByCutoff.test_log[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      16      9
## actual.0      59     120
## [1] "-----"
## [1] "Model General Accuracy of: 66.67%"
## [1] "True Positive Rate of      : 64%"
```

```
# decision tree
accuracy(score = prob.test$decision.tree.predicted,
         actual = prob.test$decision.tree.actual,
         threshold = metricsByCutoff.test_DT[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      20      5
## actual.0      51     128
## [1] "-----"
## [1] "Model General Accuracy of: 72.55%"
## [1] "True Positive Rate of      : 80%"
```

```
# boosting
accuracy(score = prob.test$boosting.predicted,
         actual = prob.test$boosting.actual,
         threshold = metricsByCutoff.test_boost[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      21      4
## actual.0      41     138
## [1] "-----"
## [1] "Model General Accuracy of: 77.94%"
## [1] "True Positive Rate of      : 84%"
```

```
# random forest
accuracy(score = prob.test$random.forest.predicted,
         actual = prob.test$random.forest.actual,
         threshold = metricsByCutoff.test_rf[["BestCut"]][["Cut"]])
```

```
##
##
##          pred.1  pred.0
## -----  -----  -----
## actual.1      19      6
## actual.0      41     138
## [1] "-----"
## [1] "Model General Accuracy of: 76.96%"
## [1] "True Positive Rate of      : 76%"
```

34 Final considerations and project limitations

Unfortunately, this is not a real dataset with really interest variables such as client income, credit rate, and so on.

The dataset is also not big enough to deliver consistent results, we saw significant variation in the models depending on the split ratio, and the hyperparameters used in each model, but the dataset served very well for the intent of practicing the techniques and R programming skills learned in the class.

35 References used in this class assignment.

35.1 Theory

35.1.1 Books and articles

An Introduction to Statistical Learning (<http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf>)

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Understanding Random Forest: How the Algorithm Works and Why it is So Effective (<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>)

Tony Yiu

What Random Forests Tell Us About Democracy (<http://duboue.net/blog17.html>)

Pablo Duboue

Random Forest — A Model Designed to Provide Structure in Chaos (<https://medium.com/analytics-vidhya/random-forest-a-model-designed-to-provide-structure-in-chaos-e267d559ca04>)

Pranov Mishra

Tune Machine Learning Algorithms in R (random forest case study) (<https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/>)

Jason Brownlee

35.1.2 Wikipedia

Akaike Information Criterion (https://en.wikipedia.org/wiki/Akaike_information_criterion)

Kolmogorov–Smirnov Test (https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test)

Multicollinearity (<https://en.wikipedia.org/wiki/Multicollinearity>)

Receiver Operating Characteristic (https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

35.2 R Programming

R for Data Science (<https://r4ds.had.co.nz/>)

Garrett Golemund, Hadley Wickham

Top 50 ggplot2 Visualizations - The Master List (With Full R Code) (<http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>)

Selva Prabhakaran

caret Package (<http://topepo.github.io/caret/index.html>)

Max Kuhn

randomForest Package (<https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest>)

Andy Liaw

hmeasure Package (<https://www.hmeasure.net/>)

David J. Hand

35.3 Awesome functions from the community

Some awesome functions we found in ***the internet*** and adapted for this class assignment.

KS Plot (<https://stackoverflow.com/questions/39162178/kolmogorov-smirnov-plot-in-r-ggplot>)

Stack Overflow: **@Axeman** (<https://stackoverflow.com/users/4341440/axeman>)
