Final Project

Conor Ryan - 16340116

22/12/2021

```
knitr::opts_chunk$set(echo = TRUE)
```

```
# Ensuring environment is clear
rm(list = ls())
```

```
# Loading all required packages
pacman::p_load(tidyverse,scales,cowplot,DataExplorer,kableExtra,
               ROCR,rpart,yardstick)
```

```
# Setting the current working directory
setwd("C:\\Users\\conor\\Documents\\Data Analytics Masters\\Stage 1\\Semester 1\\Data Prog with R - STA
```

```
# Setting the random seed to ensure results are reproducible
set.seed(1759)
```

Part 1: Analysis

Overview of the data set

The data-set I have chosen to perform my analyses on is the German Credit Data data-set which is available from the UCI (University of California Irvine) machine learning data repository via the following link https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data.

The data-set was compiled by Professor Hans Hofmann from the institute of Statistics and Econometrics at the University of Hamburg and contains information on 1,000 German borrowers collected between 1973 and 1975. The data set classifies borrowers as having "good" or "bad" credit ratings and also provides 20 additional variables for each credit customer e.g. duration of loan, customer's credit history, purpose and amount of the loan etc. Further information regarding this data set can be found via this link https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29.

My analysis will consist of a portfolio overview which will attempt to gain a better understanding of the characteristics of the 1,000 customers in the data set. Firstly I will read in the German Credit data set. The data is saved as a ".data" file type which is simply a text file with the values separated by a certain character, a space in this case. I read the data directly from the UCI data repository website so that my analysis is more easily reproducible.

Please note during this project I have used several packages for various reasons, (formatting tables, axes on graphs, plotting graphs etc.) that were not used extensively throughout the module. I have included a bibliography in the appendices at the end of my report where I cite these packages and explain the rationale for using them

```
# Reading in the data from the UCI ML data repository
german_df <-
  read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data",
             sep = " ")

# Displaying the first 5 rows of the data set
# I use the kableExtra package to "scale-down" the table so it fits within the PDF
kbl(head(german_df,5),
    caption = "German Credit (Raw) Data",
    position = "h") %>%
  kable_styling(latex_options = c("striped","scale_down"))
```

Table 1: German Credit (Raw) Data

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
A11	6	A34	A43	1169	A65	A75	4	A93	A101	4	A121	67	A143	A152	2	A173	1	A192	A201	1
A12	48	A32	A43	5951	A61	A73	2	A92	A101	2	A121	22	A143	A152	1	A173	1	A191	A201	2
A14	12	A34	A46	2096	A61	A74	2	A93	A101	3	A121	49	A143	A152	1	A172	2	A191	A201	1
A11	42	A32	A42	7882	A61	A74	2	A93	A103	4	A122	45	A143	A153	1	A173	2	A191	A201	1
A11	24	A33	A40	4870	A61	A73	3	A93	A101	4	A124	53	A143	A153	2	A173	2	A191	A201	2

```
# Displaying size of the data set
print(c("Rows" = nrow(german_df),"Columns" = ncol(german_df)))
```

```
##    Rows Columns
##    1000      21
```

As we can see from the output above there are **1000** rows and **21** columns (20 customer characteristic columns and 1 binary credit rating) in the German Credit data set.

Data pre-processing

Adding column headers and coverting to factors

We notice that many of the categorical variables have been coded in a way which makes it impossible to determine their meaning without reading the data set description document I have mentioned previously. There are also no column headers present which makes it even more difficult to analyze the data. Therefore, before performing any analysis I will first rename the column headings and values to make the data set more usable.

```
# Creating a list of column names (obtained from description document mentioned in introduction)
german_colnames <- c("Checking_Acc_Bal","Loan_Duration","Credit_History",
                     "Loan_Purpose","Credit_Amount","Savings_Acc_Bal",
                     "Employment_Duration","Payments_Percent_Of_Income",
                     "Personal_Status","Other_Debtors","Resident_Since",
                     "Property","Age","Other_Install_Plans",
                     "Housing","Num_Existing_Credits","Employment_Status",
                     "Num_Dependents","Telephone","Foreign_Worker",
                     "Credit_Rating")

colnames(german_df) <- german_colnames
```

```
german_df <- german_df %>%
  # Renaming each of the coded categorical variables in the data set
  # To improve readability and user-friendliness of the data
  mutate(Checking_Acc_Bal = factor(Checking_Acc_Bal,
                                   levels = c("A14","A11","A12","A13"),
                                   labels = c("No Checking Account",
                                              "< 0 DM",
                                              "< 200 DM",
                                              ">= 200 DM"),
                                   ordered = TRUE),
         Credit_History = factor(Credit_History,
                                 levels = c("A30","A31","A32","A33","A34"),
                                 labels = c("All credits paid duly",
                                            "All credits at this bank paid duly",
                                            "Existing credits paid duly until now",
                                            "Delayed payments in the past",
                                            "Critical account"),
                                 ordered = FALSE),
          Loan_Purpose = factor(Loan_Purpose,
                                levels = c("A40","A41","A42","A43","A44",
                                           "A45","A46","A47","A48","A49","A410"),
                                labels = c("Car (new)","Car (used)","Furniture/Equipment",
                                           "Radio/TV","Domestic Appliance","Repairs",
                                           "Education","Holiday","Retraining",
                                           "Business","Other"),
                                ordered = FALSE),
         Savings_Acc_Bal = factor(Savings_Acc_Bal,
                                  levels = c("A65","A61","A62","A63","A64"),
                                  labels = c("No Savings Account",
                                             "< 100 DM",
                                             "100 - 500 DM",
                                             "500 - 1000 DM",
                                             "> 1000 DM"),
                                  ordered = TRUE),
         Employment_Duration = factor(Employment_Duration,
                                      levels = c("A71","A72","A73","A74","A75"),
                                      labels = c("Unemployed",
                                                 "< 1 year",
                                                 "< 4 years",
                                                 "< 7 years",
                                                 ">= 7 years"),
                                      ordered = TRUE),
         Personal_Status = factor(Personal_Status,
                                  levels = c("A91","A92","A93","A94","A95"),
                                  labels = c("Male - Divorced/Separated",
                                             "Female - Divorced/Separated/Married",
                                             "Male - Single",
                                             "Male - Married/Widowed",
                                             "Female - Single"),
                                  ordered = FALSE),
         Other_Debtors = factor(Other_Debtors,
                                levels = c("A101","A102","A103"),
                                labels = c("No Other Debtors",
```

```
                                        "Co-Applicant",
                                        "Guarantor"),
                              ordered = FALSE),
        Property = factor(Property,
                          levels = c("A121","A122","A123","A124"),
                          labels = c("Real Estate",
                                     "Savings Agreement/Life Insurance",
                                     "Car/Other",
                                     "Unknown/No Property"),
                          ordered = FALSE),
        Other_Install_Plans = factor(Other_Install_Plans,
                                     levels = c("A141","A142","A143"),
                                     labels = c("Bank",
                                                "Stores",
                                                "None"),
                                     ordered = FALSE),
        Housing = factor(Housing,
                         levels = c("A151","A152","A153"),
                         labels = c("Rent",
                                    "Owned",
                                    "Free"),
                         ordered = FALSE),
        Employment_Status = factor(Employment_Status,
                                   levels = c("A171","A172","A173","A174"),
                                   labels = c("Unemployed/Unskilled - Non-Resident",
                                              "Unskilled - Resident",
                                              "Skilled Employee/Official",
                                              "Management/Self-Employed/Highly-Skilled/Officer"),
                                   ordered = FALSE),
        Telephone = factor(Telephone,
                           levels = c("A191","A192"),
                           labels = c("N","Y"),
                           ordered = FALSE),
        Foreign_Worker = factor(Foreign_Worker,
                                levels = c("A201","A202"),
                                labels = c("Y","N"),
                                ordered = FALSE),
        Credit_Rating = factor(Credit_Rating,
                               levels = c(1,2),
                               labels = c("Good","Bad"),
                               ordered = FALSE))

# Displaying the first 5 rows and 5 columns of the dataset
# I use the kableExtra package to produce a nicely formatted output
kbl(head(german_df[,1:5],5),
    caption = "German Credit (Processed) Data",
    position = "h") %>%
  kable_styling(latex_options = c("striped","scale_down"))
```

Just to point out above that the "DM" characters within the checking account balance stands for Deutsche-Mark or D-Mark which was the currency used in Germany before the arrival of the euro in 2002.

Table 2: German Credit (Processed) Data

Checking_Acc_Bal	Loan_Duration	Credit_History	Loan_Purpose	Credit_Amount
< 0 DM	6	Critical account	Radio/TV	1169
< 200 DM	48	Existing credits paid duly until now	Radio/TV	5951
No Checking Account	12	Critical account	Education	2096
< 0 DM	42	Existing credits paid duly until now	Furniture/Equipment	7882
< 0 DM	24	Delayed payments in the past	Car (new)	4870

Removing unnecessary variables

Now that we have converted the previously coded categorical variables to user-friendly factors the next step is to remove the unnecessary and unusable variables for which we have no use for or where the descriptions of the variables in the data dictionary are unclear. The variables I have chosen to remove are the following;

1. **Property** - Description of variable unclear
2. **Other Installment Plans** - Description of variable unclear
3. **Number of Existing Credits** - Description of variable unclear
4. **Telephone** - Unnecessary
5. **Other Debtors** - Description of variable unclear

```
# Removing un-clear / unnecessary variables from the data
german_df <- german_df %>%
  select(-Property,-Other_Install_Plans,-Num_Existing_Credits,
         -Telephone,-Other_Debtors)

# Printing the updated dimensions of the data set
print(c("Rows" = nrow(german_df),"Columns" = ncol(german_df)))
```

```
##    Rows Columns
##    1000      16
```

Deriving additional variables

Next I would like to derive a few additional columns based on some of the variables within the data set. Firstly, I wish to split out the "Personal_Status" variable to produce a gender column. For the purposes of this analysis I assume there are no widowed females in the data set as this cohort is not included in any of the 5 values that the personal status variable can take.

I will also produce a banded age variable derived from the age column as I believe this will be useful in producing plots and graphical summaries. I produce a descriptive summary of the age variable to show the minimum and maximum values in order to inform my choice of age banding. As we can see below the minimum age of borrower in the data set is 19 and the maximum is 75.

```
# Firstly I produce a summary of the age variable to inform my age banding column
# (i.e. min and max ages within the data)

print(summary(german_df$Age))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   19.00   27.00   33.00   35.55   42.00   75.00
```

```r
# Producing gender and age banding variables based on the personal status and
# age variables respectively
german_df <- german_df %>%
  mutate(Gender = case_when(grepl(pattern = "Female",
                                  x = Personal_Status,ignore.case = TRUE) ~ "Female",
                            grepl(pattern = "Male",
                                  x = Personal_Status,ignore.case = TRUE) ~ "Male",
                            TRUE ~ "Unknown"),
         Gender = factor(Gender),
         Age_Banding = case_when(# In case of any missing values set the age band to "unknown age"
                                 is.na(Age) ~ "Unknown Age",
                                 Age <= 20 ~ "0 - 20 years",
                                 Age <= 30 ~ "21 - 30 years",
                                 Age <= 40 ~ "31 - 40 years",
                                 Age <= 50 ~ "41 - 50 years",
                                 Age <= 60 ~ "51 - 60 years",
                                 Age <= 70 ~ "61 - 70 years",
                                 Age <= 80 ~ "71 - 80 years",
                                 Age > 80 ~ "80+ years"),
         Age_Banding = factor(Age_Banding,
                              levels = c("Unknown Age",
                                         "0 - 20 years",
                                         "21 - 30 years",
                                         "31 - 40 years",
                                         "41 - 50 years",
                                         "51 - 60 years",
                                         "61 - 70 years",
                                         "71 - 80 years",
                                         "80+ years"),
                              ordered = TRUE))
```

Checking for missing data

The final task I will perform before going on and analyzing the data set is to check for any missing values. Missing values can cause certain functions (such as means, totals, averages etc.) to not work and obscure summaries and graphs. If we have missing values in the dataset we need to figure out a way of dealing with them such as removing any observation which contains missing data or imputing the missing data points using a variety of methods (e.g. mean / null imputation).
As we can see below we have no missing data points within the data set and therefore we are ready to move onto performing our analyses.

```r
# Calculating the total number of missing values in the data
no_missing_vals <- sum(is.na(german_df))

# Writing an if statement to tell us whether there is missing data or not
if (no_missing_vals == 0) {
  print("There are no missing values in the German Credit data set")
} else {
  print(paste0("There are ",no_missing_vals," missing values in the German Credit data set"))
}
```

```
## [1] "There are no missing values in the German Credit data set"
```

Analysis

Since we have completed the data pre-processing stage, we are ready to move onto the analysis of the data set. The variable of interest is whether or not the customer has a "good" or "bad" credit rating and if we were to perform a classification exercise using machine learning on this dataset then the "Credit Rating" variable would be our dependent or target variable. This means that we would be interested in predicting / explaining a customer's credit rating based on the remaining variables in the data (explanatory variables). Therefore, I will first calculate the proportion of our customers who have a "good" credit rating versus those who don't.

```
# Producing simple summary of the credit rating variable
print(summary(german_df$Credit_Rating) / length(german_df$Credit_Rating))
```

```
## Good  Bad
##  0.7  0.3
```

As we can see above 70% of customers have good credit ratings whereas 30% have bad credit ratings. This shows our data set is slightly unbalanced with respect to credit rating.

Now I will display the structure of the data set to determine what kind of variables we have e.g. numerical, nominal, ordinal etc. Then I will present a summary of each of these variables depending on their data type in order to explore how these variables are distributed.

```
# Displaying the structure of the data set
str(german_df)
```

```
## 'data.frame':    1000 obs. of  18 variables:
##  $ Checking_Acc_Bal          : Ord.factor w/ 4 levels "No Checking Account"<..: 2 3 1 2 2 1 1 3 1 3
##  $ Loan_Duration             : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Credit_History            : Factor w/ 5 levels "All credits paid duly",..: 5 3 5 3 4 3 3 3 3 5 ..
##  $ Loan_Purpose              : Factor w/ 11 levels "Car (new)","Car (used)",..: 4 4 7 3 1 7 3 2 4 1
##  $ Credit_Amount             : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ Savings_Acc_Bal           : Ord.factor w/ 5 levels "No Savings Account"<..: 1 2 2 2 2 1 4 2 5 2 .
##  $ Employment_Duration       : Ord.factor w/ 5 levels "Unemployed"<"< 1 year"<..: 5 3 4 4 3 3 5 3 4 2
##  $ Payments_Percent_Of_Income: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ Personal_Status           : Factor w/ 5 levels "Male - Divorced/Separated",..: 3 2 3 3 3 3 3 3 1 4
##  $ Resident_Since            : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                       : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ Housing                   : Factor w/ 3 levels "Rent","Owned",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ Employment_Status         : Factor w/ 4 levels "Unemployed/Unskilled - Non-Resident",..: 3 3 2 3 3
##  $ Num_Dependents            : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Foreign_Worker            : Factor w/ 2 levels "Y","N": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Credit_Rating             : Factor w/ 2 levels "Good","Bad": 1 2 1 1 2 1 1 1 1 2 ...
##  $ Gender                    : Factor w/ 2 levels "Female","Male": 2 1 2 2 2 2 2 2 2 2 ...
##  $ Age_Banding               : Ord.factor w/ 9 levels "Unknown Age"<..: 7 3 5 5 6 4 6 4 7 3 ...
```

```
# Producing summary proportions for nominal (non-ordered) variables

# Iterating over the columns in the data set (after dropping the credit rating)
for (col in colnames(select(german_df,-Credit_Rating))) {
```

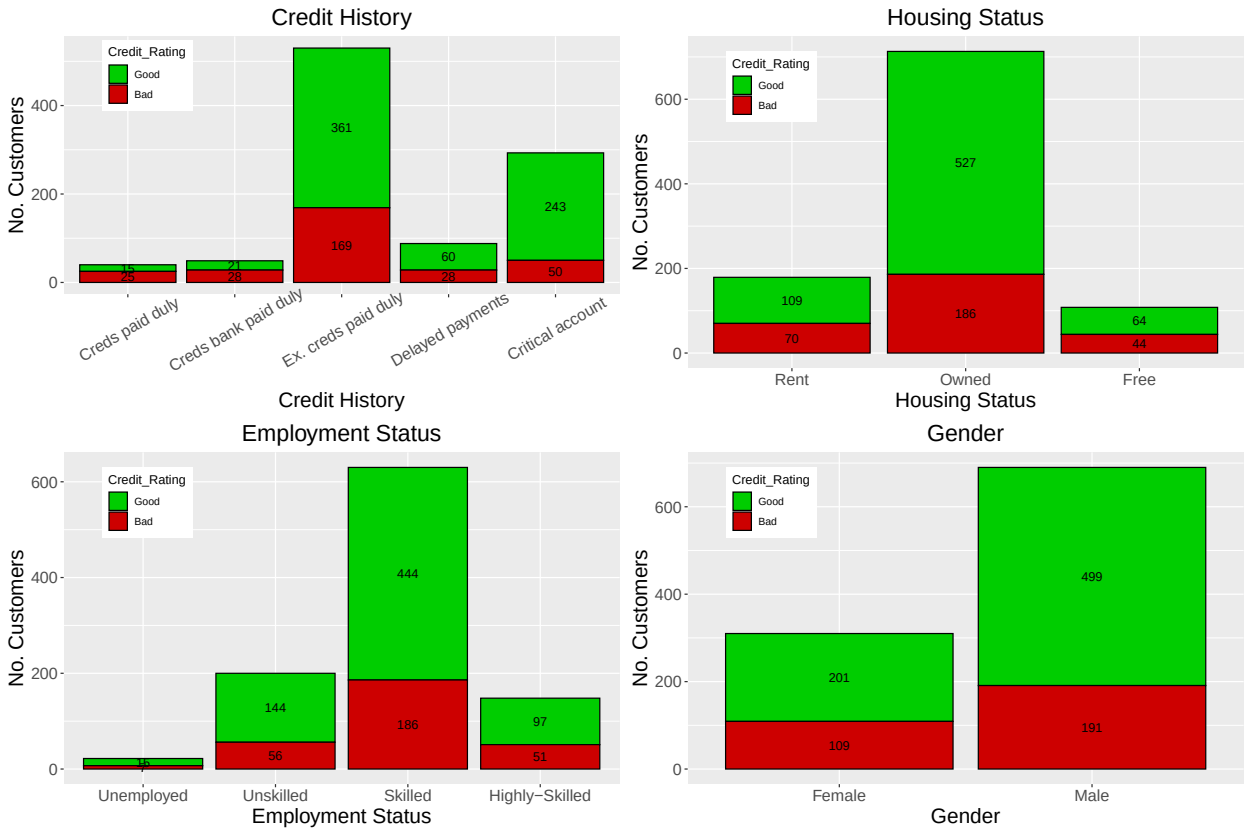
```
  # If the variable is a factor then we print the summary
  # (dividing by total rows to give proportions)
  if (class(german_df[,col])[1] == "factor") {
    cat(paste0("**",col,"**"),"\n")
    print(round(summary(german_df[,col]) / nrow(german_df),3))
    cat("\n")
  } else {
    # else do nothing
  }
}
```

```
## **Credit_History**
##                All credits paid duly   All credits at this bank paid duly
##                                0.040                                0.049
## Existing credits paid duly until now        Delayed payments in the past
##                                0.530                                0.088
##                     Critical account
##                                0.293
##
## **Loan_Purpose**
##           Car (new)          Car (used) Furniture/Equipment            Radio/TV
##               0.234               0.103               0.181               0.280
##  Domestic Appliance             Repairs           Education             Holiday
##               0.012               0.022               0.050               0.000
##          Retraining            Business               Other
##               0.009               0.097               0.012
##
## **Personal_Status**
##           Male - Divorced/Separated Female - Divorced/Separated/Married
##                                0.050                               0.310
##                        Male - Single             Male - Married/Widowed
##                                0.548                               0.092
##                     Female - Single
##                                0.000
##
## **Housing**
##  Rent Owned  Free
## 0.179 0.713 0.108
##
## **Employment_Status**
##             Unemployed/Unskilled - Non-Resident
##                                           0.022
##                             Unskilled - Resident
##                                           0.200
##                       Skilled Employee/Official
##                                           0.630
## Management/Self-Employed/Highly-Skilled/Officer
##                                           0.148
##
## **Foreign_Worker**
##     Y     N
## 0.963 0.037
##
```

```
## **Gender**
## Female   Male
##   0.31   0.69
```

```
# Producing bar charts for nominal variables

# Credit history
cred_hist_bar <- ggplot(data = german_df,aes(x = Credit_History,fill=Credit_Rating)) +
  geom_bar(position = "stack",aes(y = ..count..),col = "black") +
  # adding counts of "good" & "bad" credit ratings
  geom_text(aes(label = ..count..),stat = "count",position = position_stack(vjust = 0.5)) +
  # adding a title
  ggtitle("Credit History") +
  xlab("Credit History") +
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 14),
        axis.text.x = element_text(angle = 30,vjust = 0.7),
        legend.position = c(.15,.85)) +
  # manually adjusting x value labels for aesthetics
  scale_x_discrete(labels = c("Creds paid duly","Creds bank paid duly",
                              "Ex. creds paid duly","Delayed payments",
                              "Critical account")) +
  # adjusting colours of "good" and "bad" credit rating
  scale_fill_manual(values = c("green3","red3"))

# Housing
house_bar <- ggplot(data = german_df,aes(x = Housing,fill=Credit_Rating)) +
  geom_bar(position = "stack",aes(y = ..count..),col = "black") +
  # adding counts of "good" & "bad" credit ratings
  geom_text(aes(label = ..count..),stat = "count",position = position_stack(vjust = 0.5)) +
  # adding a title
  ggtitle("Housing Status") +
  xlab("Housing Status") +
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 14),
        legend.position = c(.15,.85)) +
  # adjusting colours of "good" and "bad" credit rating
  scale_fill_manual(values = c("green3","red3"))

# Employment status
employ_stat_bar <- ggplot(data = german_df,aes(x = Employment_Status,fill=Credit_Rating)) +
  geom_bar(position = "stack",aes(y = ..count..),col = "black") +
  # adding counts of "good" & "bad" credit ratings
  geom_text(aes(label = ..count..),stat = "count",position = position_stack(vjust = 0.5)) +
  # adding a title
  ggtitle("Employment Status") +
  xlab("Employment Status") +
  ylab("No. Customers") +
```

```
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 14),
        legend.position = c(.15,.85)) +
  # manually adjusting x value labels for aesthetics
  scale_x_discrete(labels = c("Unemployed","Unskilled",
                              "Skilled","Highly-Skilled")) +
  # adjusting colours of "good" and "bad" credit rating
  scale_fill_manual(values = c("green3","red3"))

# Gender
gender_bar <- ggplot(data = german_df,aes(x = Gender,fill=Credit_Rating)) +
  geom_bar(position = "stack",col = "black") +
  # adding counts of "good" & "bad" credit ratings
  geom_text(aes(label = ..count..),stat = "count",position = position_stack(vjust = 0.5)) +
  # adding a title
  ggtitle("Gender") +
  xlab("Gender") +
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 14),
        legend.position = c(.15,.85)) +
  # adjusting colours of "good" and "bad" credit rating
  scale_fill_manual(values = c("green3","red3"))

# plotting all charts in the one figure using plot grid
plot_grid(cred_hist_bar,house_bar,employ_stat_bar,gender_bar,
          nrow = 2,ncol = 2)
```



Above I have produced the proportion summaries for each of the nominal variables in the data set. I have also created stacked bar charts for the credit history, housing, employment status and gender variables which I have split by credit rating. It is important to analyse the breakdown of each of these variables in terms of the customer's credit rating as we are interested in identifying patterns in the data which may influence a given customer's credit score.

Observing the numerical summary output we can see that over half of the customers have either all their credits paid on time (at the current or different bank) or their existing credits paid on time up to the point of data collection (62%). There are however, 29% of customers whose account histories are critical. There is not much information given on the critical account cohort in the data dictionary and therefore it is difficult to understand what is meant by "critical". Observing the bar chart for the credit history we can see that the majority of the "critical account" cohort have "good" credit ratings (243 vs 50). We note that there are a large number of "good" credit ratings in the "Existing credits paid duly til now" cohort and surprisingly there is a large proportion of "bad" credit ratings in the two "paid duly" cohorts both at this bank and at another bank.

We can see that the majority of loans are being used either to purchase a radio or TV (28%) or to buy a new car (23%). On the other hand there are not many loans being used to go on holidays, purchase domestic appliances or for retraining purposes (all <= 1%). I will dive a bit deeper into the purposes of the loans further on in my analysis.

In terms of gender demographics, 69% of the portfolio are males and of those males 79% are single. We have no single females in the data set whatsoever. Potentially this could be due to a lack of women's civil rights in Germany during the period 1973-1975 as it was not until 1977 that women were allowed to work without first getting permission from their husbands (Wikipedia - [https://en.wikipedia.org/wiki/Timeline_of_women%27s_legal_rights_(other_than_voting)_in_the_20th_century]). Again, there is no obvious pattern with regards to credit ratings and gender as you would expect intuitively.

We see that most of the customer's in this portfolio are owners of their place of residence and we also see

that home ownership has a far greater proportion of "good" credit ratings compared with renting or living for free (assuming in their parent's house). If you own your own house it is likely you have either successfully paid off a mortgage or are currently paying one off and therefore you would have a better credit rating than people who do not own their own house.

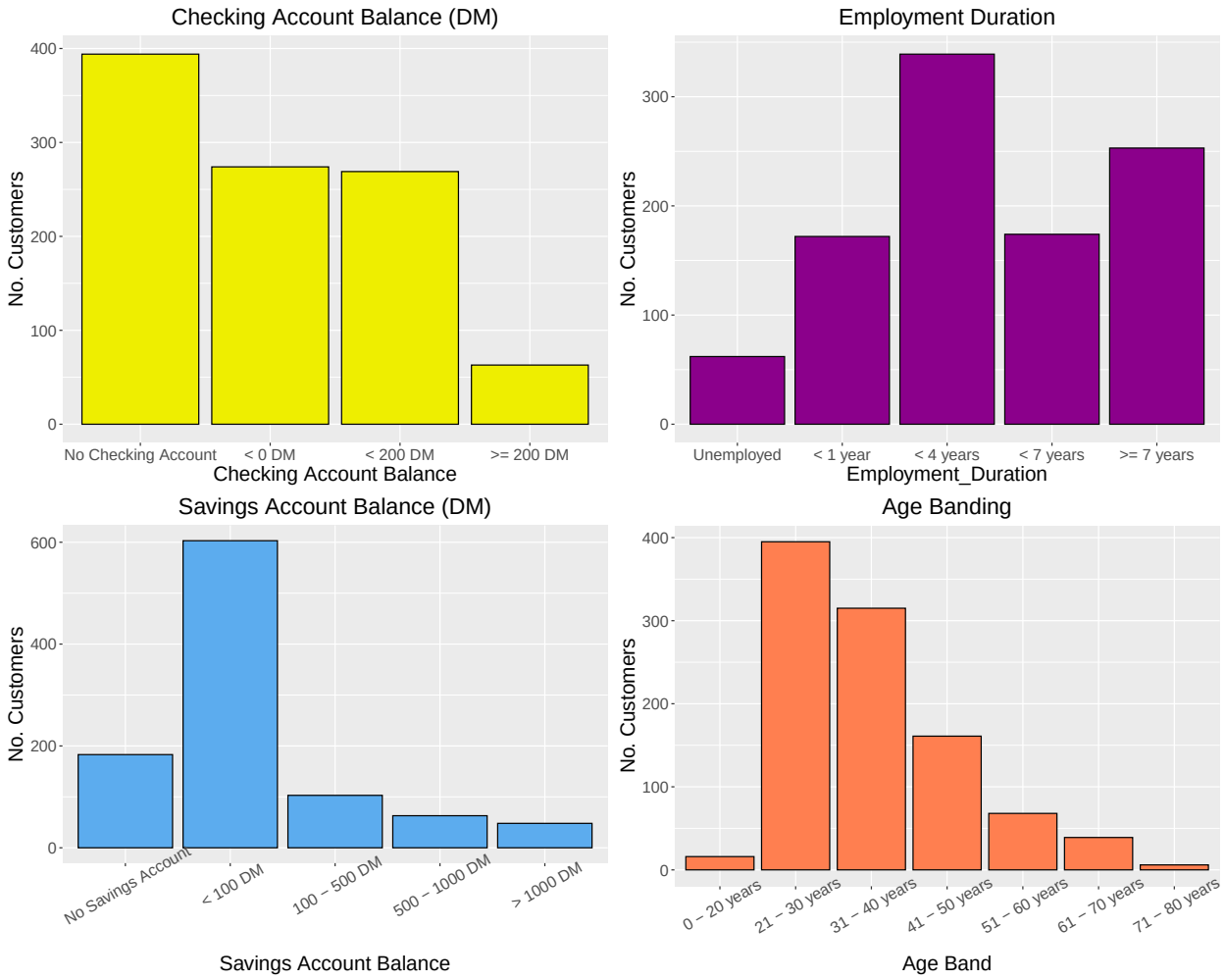
One of the proportions that really stands out in the data set is the proportion of customers who are foreign workers. Foreign workers make up a massive 96% of the portfolio and this is something we must factor in when making any inferences about the German credit landscape as a whole. This is because our data is heavily biased towards foreign workers and may not represent a typical German credit portfolio.

```
# Producing summary proportions for ordinal variables

# Iterating over the columns in the data set (after dropping the credit rating)
for (col in colnames(select(german_df,-Credit_Rating))) {
  # If the variable is an ordered factor then we print the summary
  # (dividing by total rows to give proportions)
  if (class(german_df[,col])[1] == "ordered") {
    cat(paste0("**",col,"**"),"\n")
    print(round(summary(german_df[,col]) / nrow(german_df),3))
    cat("\n")
  } else {
    # else do nothing
  }
}
```

```
## **Checking_Acc_Bal**
## No Checking Account              < 0 DM            < 200 DM           >= 200 DM
##               0.394               0.274               0.269               0.063
##
## **Savings_Acc_Bal**
## No Savings Account           < 100 DM       100 - 500 DM      500 - 1000 DM
##              0.183              0.603              0.103              0.063
##          > 1000 DM
##              0.048
##
## **Employment_Duration**
## Unemployed   < 1 year  < 4 years  < 7 years >= 7 years
##      0.062      0.172      0.339      0.174      0.253
##
## **Age_Banding**
##   Unknown Age  0 - 20 years 21 - 30 years 31 - 40 years 41 - 50 years
##         0.000         0.016         0.395         0.315         0.161
## 51 - 60 years 61 - 70 years 71 - 80 years     80+ years
##         0.068         0.039         0.006         0.000
```

```
# Plotting bar charts & densities of Checking account balance, Savings account balance,
# Duration of employment and Age banding

# Checking account balance
check_acc_bar <- ggplot(data = german_df,aes(x = Checking_Acc_Bal)) +
  geom_bar(aes(y = ..count..),fill = "yellow2",col = "black") +
  # adding a title
  ggtitle("Checking Account Balance (DM)") +
  xlab("Checking Account Balance") +
```

```
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 14))

# Savings account balance
Savings_acc_bar <- ggplot(data = german_df,aes(x = Savings_Acc_Bal)) +
  geom_bar(aes(y = ..count..),fill = "steelblue2",col = "black") +
  # adding a title
  ggtitle("Savings Account Balance (DM)") +
  xlab("Savings Account Balance") +
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text.x = element_text(angle = 30,vjust = 0.85),
        axis.text = element_text(size = 14))

# Employment duration
employ_dur_bar <- ggplot(data = german_df,aes(x = Employment_Duration)) +
  geom_bar(aes(y = ..count..),fill = "darkmagenta",col = "black") +
  # adding a title
  ggtitle("Employment Duration") +
  xlab("Employment_Duration") +
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 14))

# Age banding
age_band_bar <- ggplot(data = german_df,aes(x = Age_Banding)) +
  geom_bar(aes(y = ..count..),fill = "coral",col = "black") +
  # adding a title
  ggtitle("Age Banding") +
  xlab("Age Band") +
  ylab("No. Customers") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text.x = element_text(angle = 30,vjust = 0.85),
        axis.text = element_text(size = 14))


# plotting all of the bar charts within the same plot using plot_grid function
plot_grid(check_acc_bar,employ_dur_bar,Savings_acc_bar,age_band_bar,
          nrow = 2,
          ncol = 2)
```



I have produced similar proportion summaries for the ordinal variables along with graphical summaries in the form of bar charts as shown above. We can see that c.39% of customers do not have a checking account and a further c.27% are overdrawn on their checking account (balance < 0 dms). Furthermore, c.79% of customers either do not have a savings account or have less than 100 D-Marks saved. This could cause alarm to a portfolio manager especially if the loan happens to be unsecured as if the loan were to default the bank would have nothing to repossess or sell in order to make back some of its loss.

A more positive statistic in terms of credit risk is that c.77% of customers have been in their current employment for over 4 years which bodes well in terms of job security and for the bank manager's point of view, this corresponds to a steadier stream of income which can be used to pay back the debts owed to the bank. We can see in terms of age demographics that the vast majority of customer's are between the ages of 21 and 50 (c.87%).

```
# Producing summary proportions for numerical variables

# Iterating over the columns in the data set (after dropping the credit rating)
for (col in colnames(select(german_df,-Credit_Rating))) {
  # If the variable is numerical then we print the summary
  if (!class(german_df[,col])[1] %in% c("factor","ordered")) {
    cat(paste0("**",col,"**"),"\n")
    print(c(summary(german_df[,col]),
            "Variance" = var(german_df[,col]),
```

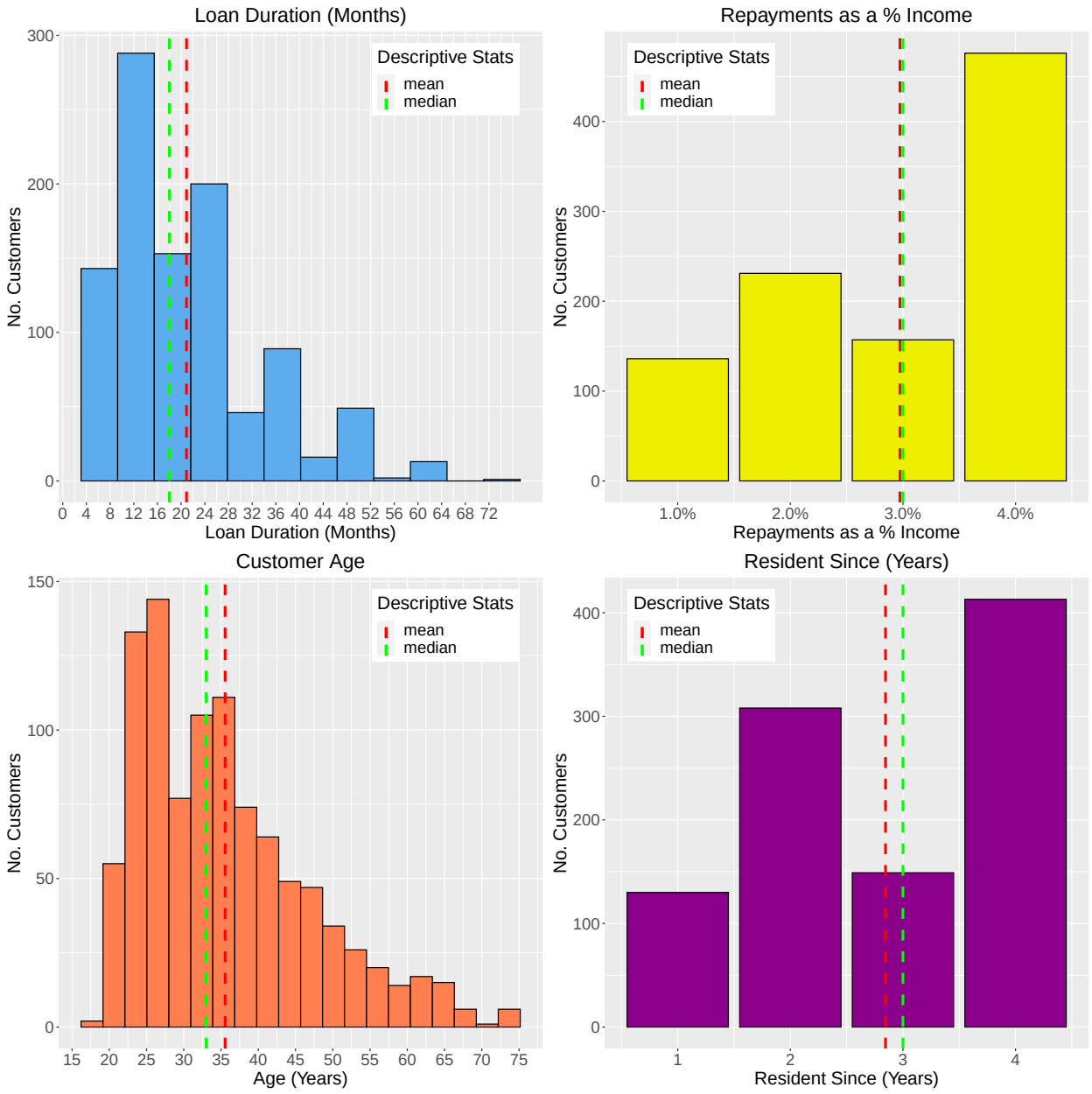
```
            "St.Dev" = sd(german_df[,col]),
            "IQR" = IQR(german_df[,col])))
    cat("\n")
  } else {
    # else do nothing
  }
}
```

```
## **Loan_Duration**
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.  Variance    St.Dev
##   4.00000  12.00000  18.00000  20.90300  24.00000  72.00000 145.41501  12.05881
##       IQR
##  12.00000
##
## **Credit_Amount**
##        Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
##     250.000    1365.500    2319.500    3271.258    3972.250   18424.000
##    Variance      St.Dev         IQR
## 7967843.471    2822.737    2606.750
##
## **Payments_Percent_Of_Income**
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. Variance   St.Dev
## 1.000000 2.000000 3.000000 2.973000 4.000000 4.000000 1.251523 1.118715
##      IQR
## 2.000000
##
## **Resident_Since**
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. Variance   St.Dev
## 1.000000 2.000000 3.000000 2.845000 4.000000 4.000000 1.218193 1.103718
##      IQR
## 2.000000
##
## **Age**
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.  Variance    St.Dev
##  19.00000  27.00000  33.00000  35.54600  42.00000  75.00000 129.40129  11.37547
##       IQR
##  15.00000
##
## **Num_Dependents**
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.  Variance    St.Dev
## 1.0000000 1.0000000 1.0000000 1.1550000 1.0000000 2.0000000 0.1311061 0.3620858
##       IQR
## 0.0000000
```

```
# Plotting histograms / bar charts of Loan Duration,Payments as a % of income,
# Age and Resident since variables

# First plot
# Simple histogram of loan duration

loan_duration_hist <- ggplot(data = german_df,aes(x = Loan_Duration)) +
  geom_histogram(aes(y = ..count..),bins = 12,fill = "steelblue2",col = "black") +
  # adding mean line
```

```
  geom_vline(aes(xintercept = mean(Loan_Duration),col="mean"),size=1.25,linetype="dashed") +
  # adding median line
  geom_vline(aes(xintercept = median(Loan_Duration),col="median"),size=1.25,linetype="dashed") +
  # setting the colours for the mean and median lines
  scale_color_manual(name="Descriptive Stats",values = c(mean = "red",median = "green")) +
  # adding a title
  ggtitle("Loan Duration (Months)") +
  ylab("No. Customers") +
  # formatting the x axis with "scale" package
  scale_x_continuous(name = "Loan Duration (Months)",
                     breaks = seq(0,72,4)) +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 16),
        legend.position = c(.8,.9),
        legend.title = element_text(size = 18),
        legend.text = element_text(size = 16))

# Second plot
# Simple bar chart of Payments as a % of income

# Dividing payments percent of income by 100 to convert to  a %age value
pay_percent_income_bar <- ggplot(data = german_df,aes(x = Payments_Percent_Of_Income/100)) +
  geom_bar(aes(y = ..count..),fill = "yellow2",col = "black") +
   # adding mean line
  geom_vline(aes(xintercept = mean(Payments_Percent_Of_Income/100),col="mean"),size=1.25,linetype="dash
  # adding median line
  geom_vline(aes(xintercept = median(Payments_Percent_Of_Income/100),col="median"),size=1.25,linetype="
   # setting the colours for the mean and median lines
  scale_color_manual(name="Descriptive Stats",values = c(mean = "red",median = "green")) +
  # adding a title
  ggtitle("Repayments as a % Income") +
  ylab("No. Customers") +
  # formatting the x axis with "scale" package
  scale_x_continuous(labels =  percent,
                     name = "Repayments as a % Income") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 16),
        legend.position = c(.2,.9),
        legend.title = element_text(size = 18),
        legend.text = element_text(size = 16))

# Third plot
# Simple histogram of Customer age

age_hist <- ggplot(data = german_df,aes(x = Age)) +
  geom_histogram(aes(y = ..count..),bins = 20,fill = "coral",col = "black") +
  # adding mean line
  geom_vline(aes(xintercept = mean(Age),col="mean"),size=1.25,linetype="dashed") +
  # adding median line
```

```
  geom_vline(aes(xintercept = median(Age),col="median"),size=1.25,linetype="dashed") +
  # setting the colours for the mean and median lines
  scale_color_manual(name="Descriptive Stats",values = c(mean = "red",median = "green")) +
  # adding a title
  ggtitle("Customer Age") +
  ylab("No. Customers") +
  # formatting the x axis with "scale" package
  scale_x_continuous(name = "Age (Years)",
                     breaks = seq(10,80,5)) +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 16),
        legend.position = c(.8,.9),
        legend.title = element_text(size = 18),
        legend.text = element_text(size = 16))

# Fourth plot
# Simple histogram of Resident since
# (i.e. in current residence for x years)

resident_bar <- ggplot(data = german_df,aes(x = Resident_Since)) +
  geom_bar(aes(y = ..count..),fill = "darkmagenta",col = "black") +
  # adding mean line
  geom_vline(aes(xintercept = mean(Resident_Since),col="mean"),size=1.25,linetype="dashed") +
  # adding median line
  geom_vline(aes(xintercept = median(Resident_Since),col="median"),size=1.25,linetype="dashed") +
  # setting the colours for the mean and median lines
  scale_color_manual(name="Descriptive Stats",values = c(mean = "red",median = "green")) +
  # adding a title
  ggtitle("Resident Since (Years)") +
  ylab("No. Customers") +
  xlab("Resident Since (Years)") +
  # adjusting titles and text size
  theme(plot.title = element_text(hjust = 0.5,size = 20),
        axis.title = element_text(size = 18),
        axis.text = element_text(size = 16),
        legend.position = c(.2,.9),
        legend.title = element_text(size = 18),
        legend.text = element_text(size = 16))


plot_grid(loan_duration_hist,pay_percent_income_bar,age_hist,resident_bar,
          nrow = 2, ncol = 2)
```



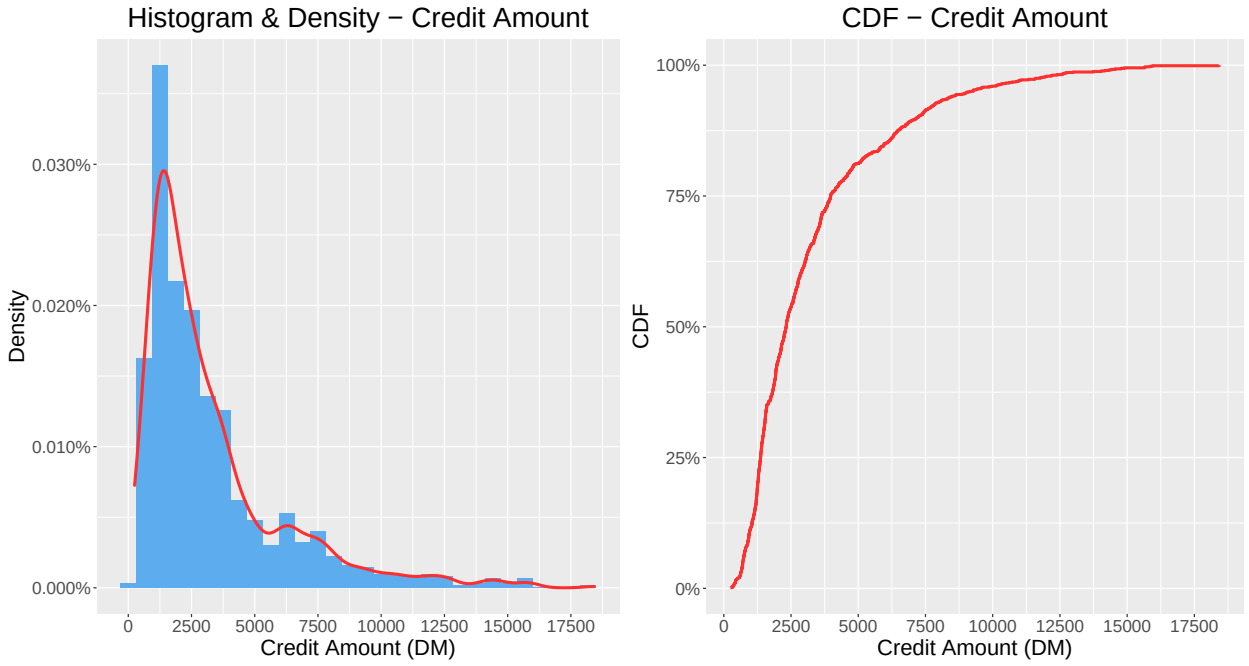
Finally, I have produced the summary statistics for the numerical variables in the German Credit data set. Starting with the duration of the loan we can see that the average term of a loan is 20.9 months. This is quite a short period of time particularly for loans, however if we observe the loan purpose variable we can see that none of these loans are mortgages which would generally have a term of at least 10 years. We also see that the max term is 72 months and therefore, we can conclude that we are looking at a short to medium term loan portfolio.

I have also created graphical summaries (histograms and bar charts) of four of the numerical variables mentioned above, namely; Loan duration, payment as a % income, resident since (years) and customer age. While the numerical summaries provide a great amount of information, the histogram and bar charts shown above give a much clearer and illustrative view of the distribution of each of these variables.

We can see from the plots above that both Age and Loan Duration are right-skewed. We observe nearly 300 loans in the 8-12 month range which is 30% of the entire portfolio. Interestingly in terms of age we can see a lot of the customers >250 are between the ages of 22 and 28. We can see the majority of customers spend

4% of their income on loan repayments, while this is the highest value within this data set, I believe 4% is still relatively low. Similarly over 400 customers have been in their current residence for 4 years.

```
# First plot
# Simple histogram and density plot of the distribution of the credit amount
cred_amt_density <- ggplot(data = german_df,aes(x = Credit_Amount)) +
  geom_histogram(aes(y = ..density..),bins = 30,fill = "steelblue2") +
  geom_density(col = "firebrick1",size = 1.25) +
  ggtitle("Histogram & Density - Credit Amount") +
# Changing the y-axis to be in percentages (using scales package)
  scale_y_continuous(labels = percent,name = "Density") +
  scale_x_continuous(name = "Credit Amount (DM)",
                     breaks = seq(0,20000,2500)) +
  theme(plot.title = element_text(hjust = 0.5,size = 24),
        axis.title = element_text(size = 19),
        axis.text = element_text(size = 15))

# Second plot
# Cumulative distribution function of the credit amount variable
cred_amt_cdf <- ggplot(data = german_df,aes(x = Credit_Amount)) +
  stat_ecdf(geom = "step",pad = FALSE,col = "firebrick1",size = 1.25) +
  ggtitle("CDF - Credit Amount") +
# Changing the y-axis to be in percentage (using scales package)
  scale_y_continuous(labels = percent,name = "CDF") +
  scale_x_continuous(name = "Credit Amount (DM)",
                     breaks = seq(0,20000,2500)) +
  theme(plot.title = element_text(hjust = 0.5,size=24),
        axis.title = element_text(size = 19),
        axis.text = element_text(size = 15))

# using plot grid from the cowplot package to arrange plots side by side
plot_grid(cred_amt_density,cred_amt_cdf)
```

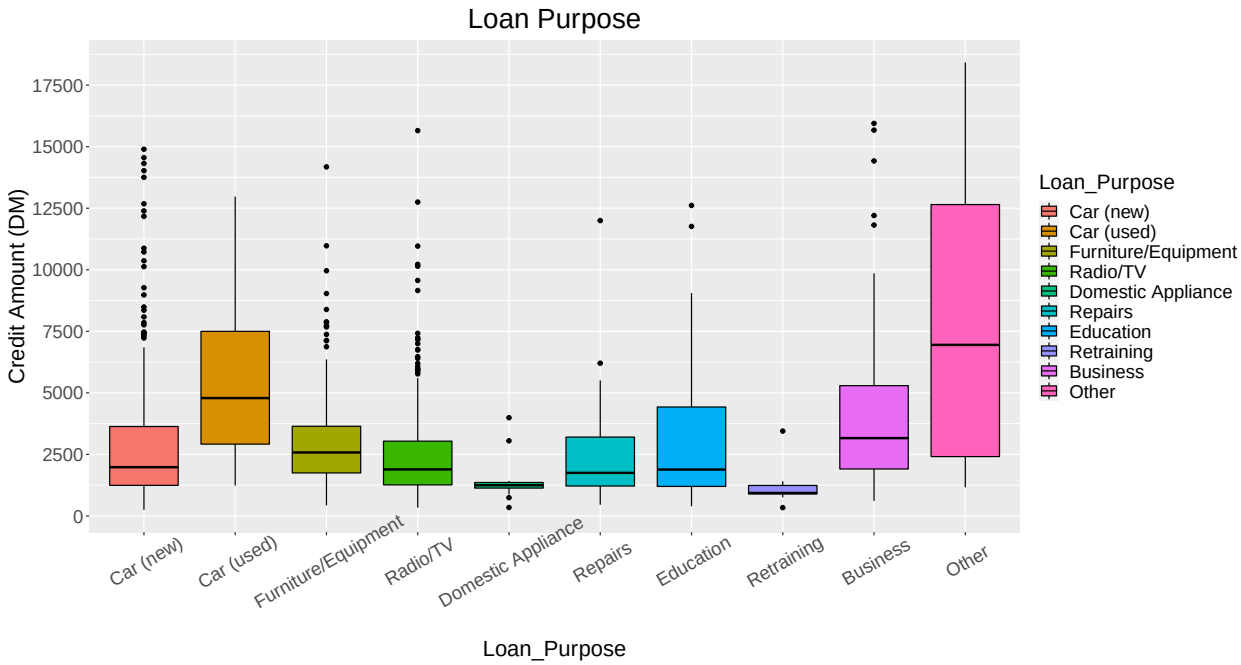


Observing the statistics for the credit or loan amount we see the average loan amount is 3,271 dm whereas the min and max are 250 dm and 18,424 dm respectively. This indicates that the loans amounts are quite varied as we have a large range (c.18,150 dm) and standard deviation (2,822.74 dms). I have produced a density plot in order to visualise the distribution of the credit amount within the portfolio (shown above).

Looking at the histogram and density plot on the left we can see that the credit amount is heavily skewed to the right. This means that the majority of customer loans within the portfolio are for smaller amounts (<5000 dm). Viewing the cumulative distribution function (CDF) on the right we can see that just over half of all customer loans are for 2,500 dm or less, 53.7% to be precise.

The distribution of loan amounts in the portfolio is consistent with the purposes for which the loans were taken out. As mentioned previously the majority of these loans were taken out to purchase either a radio or TV or to buy a new car. These are small - mid range purchases and is reflected in the skewed distribution of loan amounts. Had this portfolio be comprised of mortgages we would expect to observe a left-skewed distribution of loan amount. I will now further explore the make-up of this loan portfolio by looking more closely at the loan purpose variable.

```
barplot_loan_purpose <- ggplot(data = german_df,aes(y = Credit_Amount,x = Loan_Purpose,fill=Loan_Purpos
  geom_boxplot(col = "black") +
  ggtitle("Loan Purpose") +
# Formatting the y-axis using scales package
  scale_y_continuous(name = "Credit Amount (DM)",
                     breaks = seq(0,20000,2500)) +
  theme(plot.title = element_text(hjust = 0.5,size = 24),
        axis.title = element_text(size = 19),
        axis.text.x = element_text(angle = 30,vjust = 0.8),
        axis.text = element_text(size = 16),
        legend.title = element_text(size = 18),
        legend.text = element_text(size = 16))

barplot_loan_purpose
```



I have created a box-plot of the loan purpose variable against the credit amount of the loan, shown above.

We can see that most of the cohorts are centred around the 2,500 dm level which is consistent with what we have seen in the previous density and CDF plots. Interestingly, the "other" cohort is by far the most varied with a very large inter-quartile range ranging from c.2,500 to 12,500 dm. Intuitively this makes sense as it's likely the "other" group is made up of many different loan purposes which would lead to increase variance in the loan amounts. We see that the "other" group also has the largest median value of c.7,000 dm.

Another interesting insight gleaned from the boxplot above is the number of outliers we have within the "Radio/TV" group, our most popular group in the data set. We can see one of the outliers has a credit amount of over 15,000 dm! Immediately this would cause concern with a portfolio manager as it is unlikely even back in 1970s Germany that a radio or TV would cost more than a new car. This could be an error in the inputting of the data or the purpose of the loan being incorrectly classified as "Radio/TV".

Comparing the "Car (new)" and "Car (used)" groups we can see another interesting anomaly. The median value for the used car group is much larger than that of a new car. This is not what we would expect to see in the data as cars generally depreciate in value after their original purchase and thus used cars tend to be much cheaper than new cars. Granted we do observe values in the new car group that are higher than the maximum used car amounts but in terms of the distributions we would expect to see the opposite of what we observe in this data set.

Summary of Results

I have analyzed the German Credit data set using a variety of numerical and graphical summaries in order to get a better idea the characteristics within the portfolio of 1,000 customer loans and how these characteristics may affect a customer's credit rating. The 1,000 customer's are predominantly male (690) but the proportion of males vs females with "bad" credit ratings is roughly equal and therefore we can't say that gender is a contributing factor to credit score. The portfolio is a small-medium term loan portfolio and is made up predominantly of smaller loan amounts (skewed to the right). This is because the purpose of many of these loans are for items such as cars, radios & TVs and domestic appliances and not for homes or private jets. The average age of a customer is 35 and over 85% of customers are between the ages of 21 and 50. This means the majority of customers are in prime working age groups and should correspond to higher levels of repayment should they remain in employment. As at time of data collection 22 customers are unemployed and of these 7 have "bad" credit ratings.

We have also discovered that the duration of these loans is short with an average of 20.9 months and a maximum of 6 years. From a repayment ability perspective, most customers have been in their current employment for the last 4 years or more (c.77%) and on average have been resident in their current residence for 2.85 years. Both of these statistics demonstrate an ability of the majority of customer's to hold down a job and to pay their rent / mortgage repayments which lessens the risk of customers within the portfolio defaulting on their debt.

In the next part of the project I will demonstrate some of the functions within the "DataExplorer" package to show how much of the exploratory data analysis I have conducted here can be automated.

Part 2: DataExplorer Package

Introduction and Overview

The package I have chosen to base my report on is the DataExplorer package. According to the Cran repository, The DataExplorer package provides an "automated data exploration process for analytic tasks and predictive modeling, so that users can focus on understanding data and extracting insights". The DataExplorer package has three primary use cases;

1. Exploratory Data Analysis (EDA)

2. Feature Engineering
3. Data Reporting

Within the EDA topic, the DataExplorer package has several functions for visaulising the structure of the data set you are working with (plot_str) and also the composition of the data set in terms of variable types. The package also has very useful functions when dealing with missing data which allow you to quickly identify columns which have lots of missing data and actually outputs the percentage of missing values within each of the variables in your data set.

The DataExplorer package has several functions for plotting data visaualisations such as bar charts, histograms, qq-plots and many more. Each of these functions can be run on the entire data set and the DataExplorer function will automatically graph the correct variables based on their data type which can save you lots of time.

In terms of feature engineering, the DataExplorer package has a function that can be used to replace missing values in the data set. The function can replace both values from numeric or categorical variables and given an input of a number or string from the user, replace these automatically by matching the argument for either numeric or categorical variables. There is also a function to perform one hot encoding on all categorical variables in the data set which can be very useful when working with machine learning algorithms.

Finally, the DataExplorer package has a single function that will generate a report and run all of the previously mentioned EDA functions for you which can save a tonne of time on data analysis. To demonstrate some of the main functions within this package I will again use the German Credit data set I analysed in question 1 as I think the analysis in part 1 can provide a nice comparison in terms of workload between manually conducting your EDA and using a package like DataExplorer.

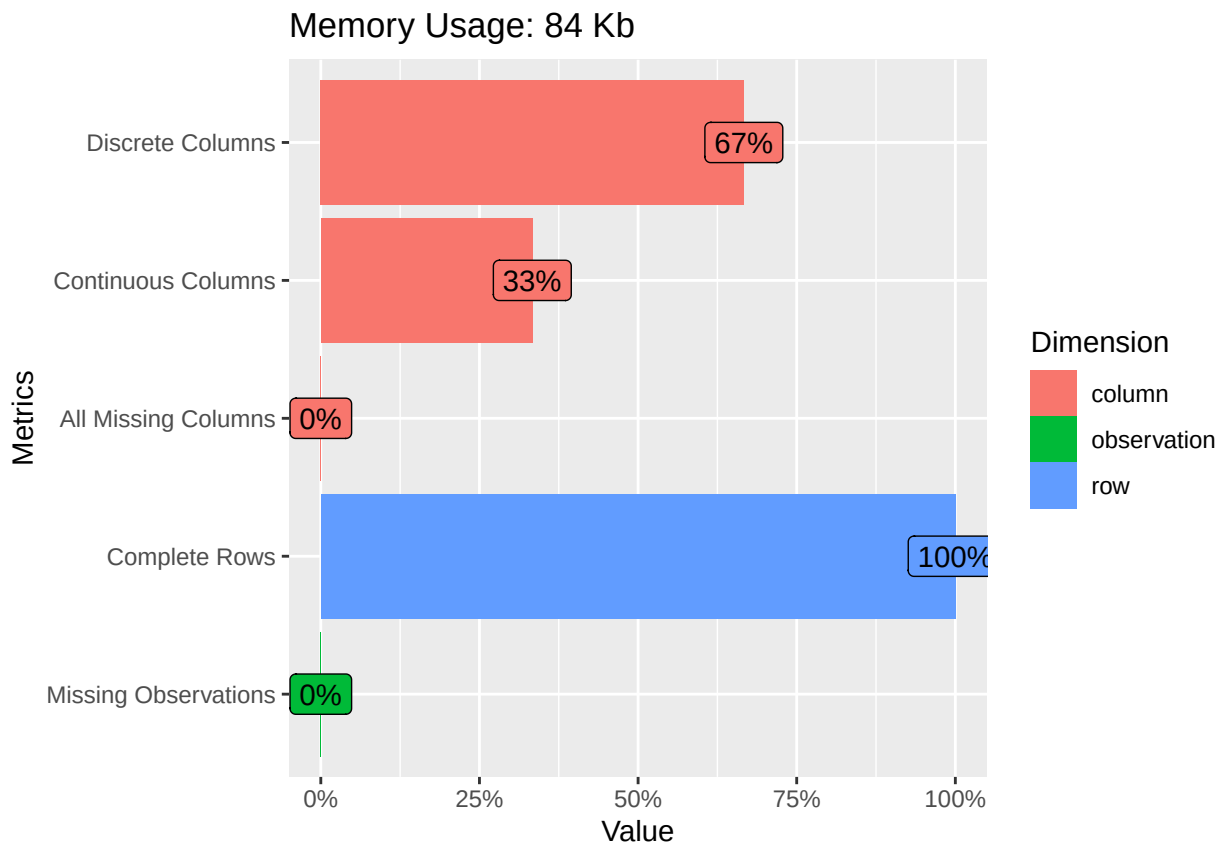
Exploratory Data Analysis

plot_intro

The first function I will demonstrate is the plot_intro function. This function can be used to visually display the structure of a data set in terms of it's variable types i.e. do we have mainly categorical (discrete) variables or mainly numerical (continuous) variables. It also gives you a summary of the missing data within your data set, the % of columns completely missing, the % of fully complete rows and the % of missing observations.

```
# Installing and reading in the Data Explorer package using pacman
pacman::p_load(DataExplorer)
```

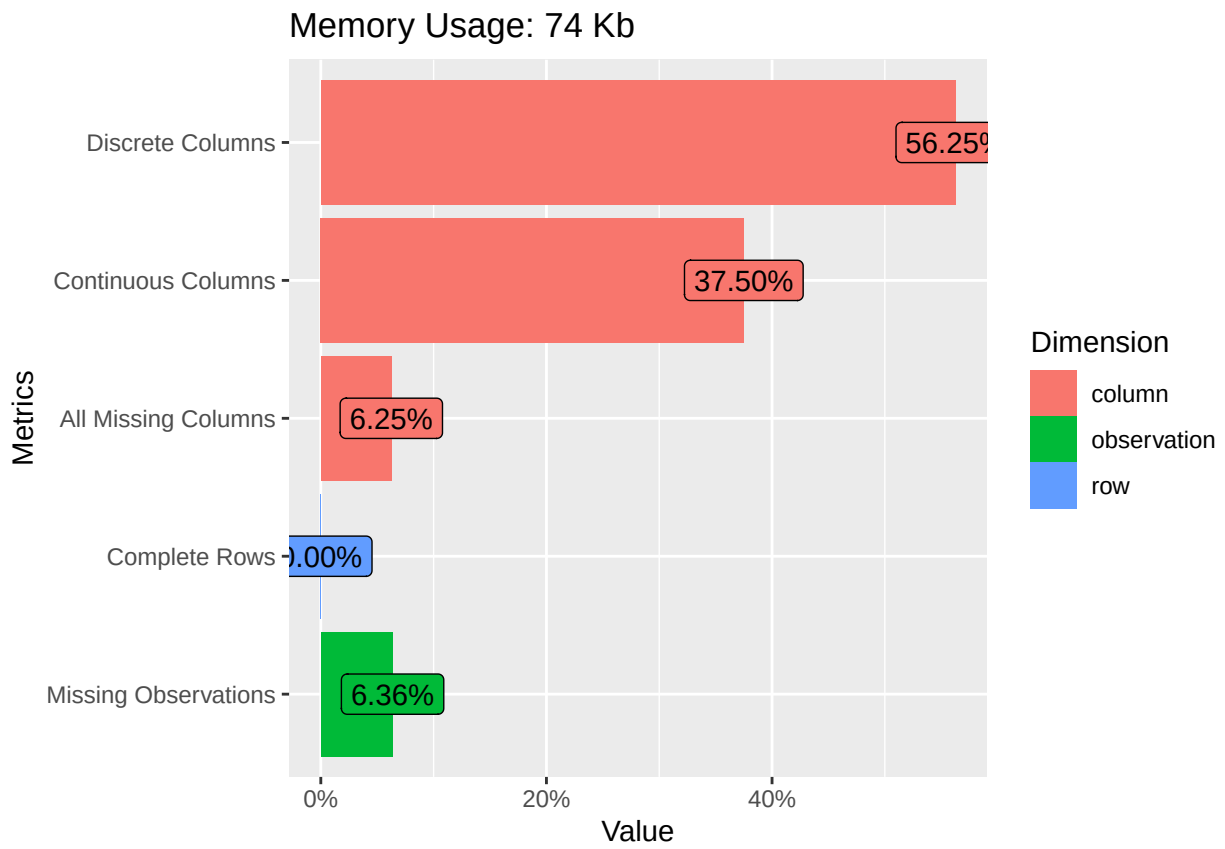
```
# Using the plot_intro function on the German Credit data set
plot_intro(german_df)
```



As we already knew from my analysis in part 1, there are no missing observations (0%) within the German Credit data set and all rows are fully complete (100%). We also see that 33% of the columns are continuous (numerical) and 67% are discrete (categorical - nominal or ordered). In part 1 I wrote a piece of code to determining the amount of missing observations within the data set whereas with the plot_info function from the DataExplorer package you get a graphical display with much more information and in a much quicker time.

In order to check the function performs when we alter the data set I am going to remove some columns and row values and re-run the function.

```
# dropping the housing and gender columns
test_df <- german_df %>%
  select(-Housing,-Gender)

# setting the entire checking account balance column to NA
test_df$Checking_Acc_Bal <- NA_integer_

# setting any age greater than 65 to NA
test_df <- test_df %>%
  mutate(Age = case_when(Age > 65 ~ NA_integer_,
                         TRUE ~ Age))

# re-running the plot intro function on the test data frame
plot_intro(test_df)
```

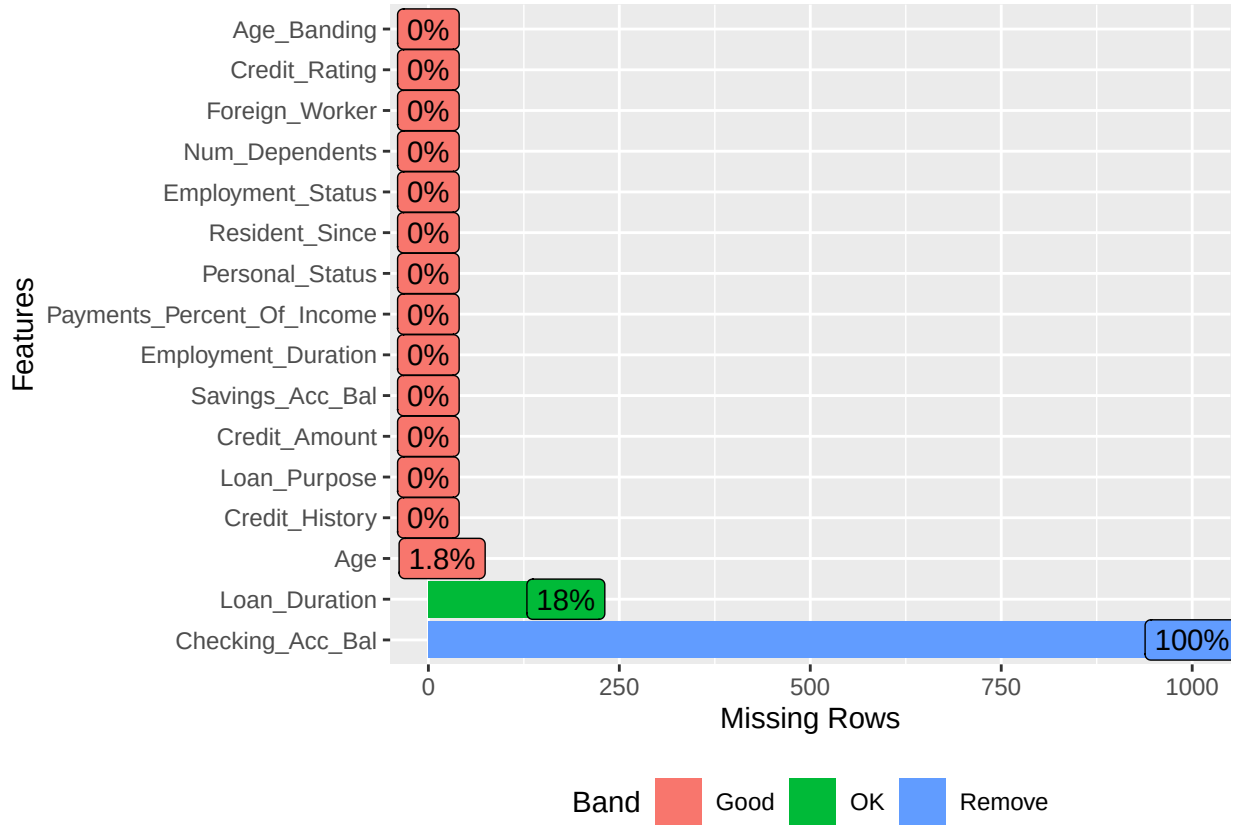


After the changes made to the test data set we can see that we now have no complete rows whatsoever (due to the fact the entire checking account balance column is NA) and we now have 6.36% of observations are missing and 6.25% of columns are fully missing. We can see from the output above that this is a really handy function to quickly summarise both the variable types and status of missing data within a data set.

plot missing

There's no doubt that plot_intro is a useful function for summarising the level of missing data within a data set but what if you need to find out which variables are to blame for the missing data, well plot missing does exactly that. Plot missing displays the level of missing data in each variable in the data frame so you can quickly identify ones you need to drop or to source more data for. I will use the test data set I created while demonstrating the plot_intro function to demonstrate the use of plot_missing.

```
# replacing all the loan durations of < 12 months with NAs
test_df <- test_df %>%
  mutate(Loan_Duration = case_when(Loan_Duration < 12 ~ NA_integer_,
                                   TRUE ~ Loan_Duration))

# demonstrating the plot missing function on the test data frame
plot_missing(test_df)
```

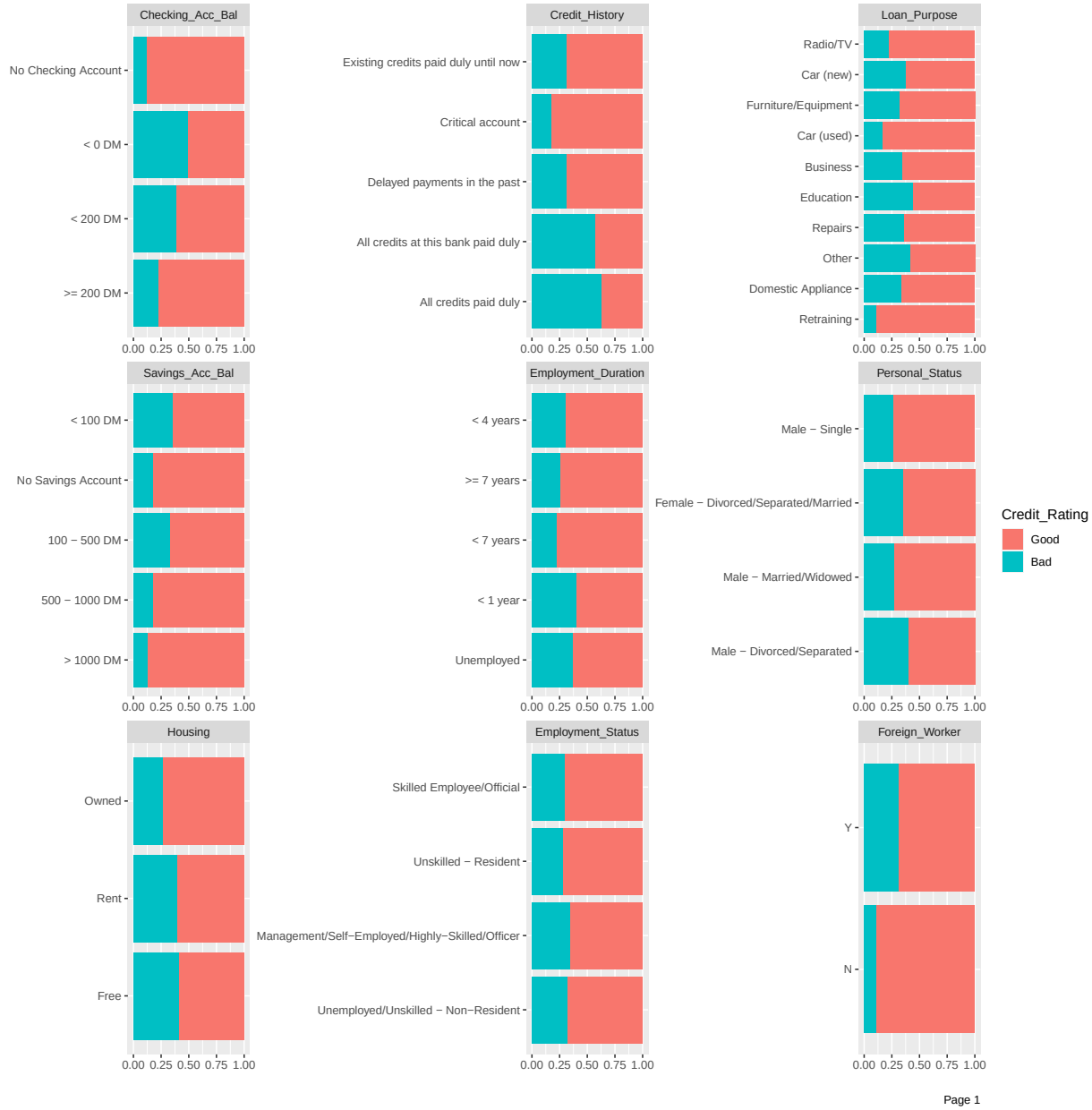


As we can see from the output of the plot_missing function above, the checking account balance column is fully missing and the plot_missing function recommends removing this column. We also see that loan duration is "OK" in terms of missing data with the remaining variables being described as "good". Again this is a really useful function to pinpoint where the problems with missing data is within a data set.

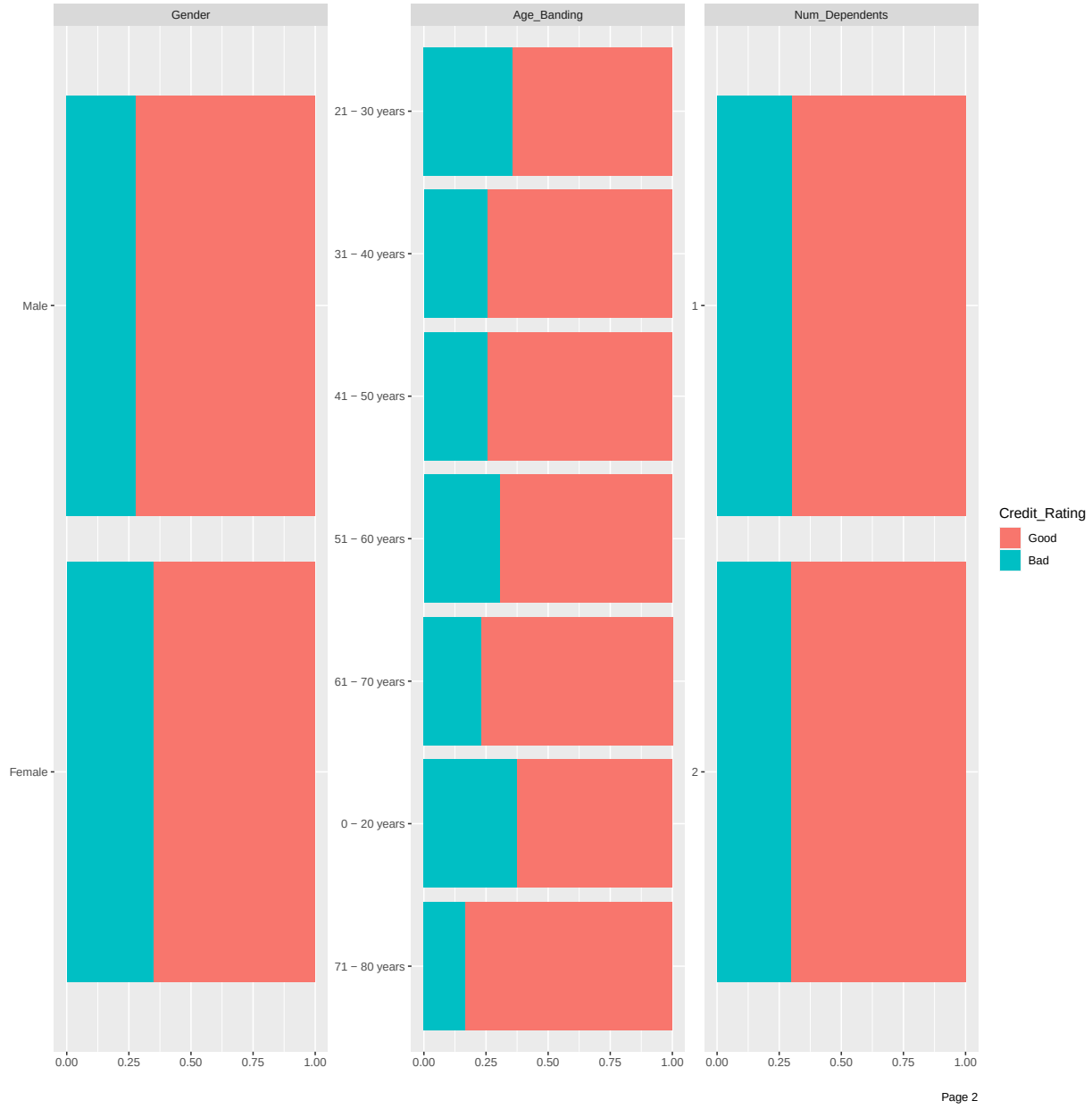
plot_bar

Next up I will demonstrate the abilities of the plot bar function. This function plots a bar chart for all discrete (categorical) variables within a data set and outputs the result all together in one clean plot. You also have the ability to pass another variable to the plot_bar function in order to observe the bi-variate distributions of each of our discrete variables against this other variable. In the case of our German Credit data, the plot bar function can be used to plot the distribution of all our discrete variables against the customer's credit rating which I will demonstrate both below. It is also possible to pass a continuous variable to the plot_bar function and it works in a similar fashion.

```
# running the plot_bar function on the German Credit data set
plot_bar(german_df,by = "Credit_Rating")
```


Checking_Acc_Bal
No Checking Account
< 0 DM
< 200 DM
>= 200 DM
Credit_History
Existing credits paid duly until now
Critical account
Delayed payments in the past
All credits at this bank paid duly
All credits paid duly
Loan_Purpose
Radio/TV
Car (new)
Furniture/Equipment
Car (used)
Business
Education
Repairs
Other
Domestic Appliance
Retraining
Savings_Acc_Bal
< 100 DM
No Savings Account
100 − 500 DM
500 − 1000 DM
> 1000 DM
Employment_Duration
< 4 years
>= 7 years
< 7 years
< 1 year
Unemployed
Personal_Status
Male − Single
Female − Divorced/Separated/Married
Male − Married/Widowed
Male − Divorced/Separated
Credit_Rating
Good
Bad
Housing
Owned
Rent
Free
Employment_Status
Skilled Employee/Official
Unskilled − Resident
Management/Self−Employed/Highly−Skilled/Officer
Unemployed/Unskilled − Non−Resident
Foreign_Worker
Y
N
0.00 0.25 0.50 0.75 1.00

Page 1



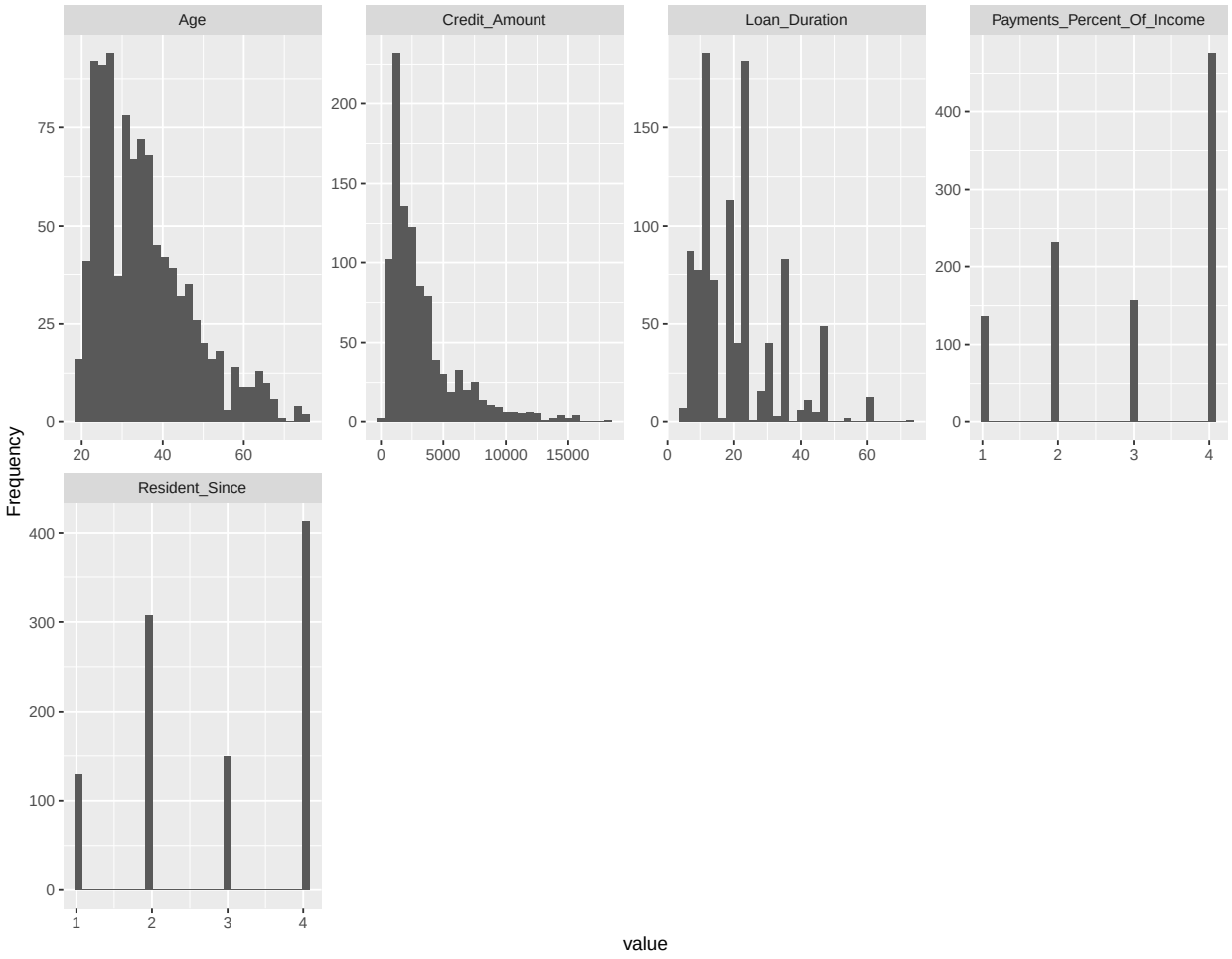
As we can see above the plot_bar function produces the frequency distributions for each of the discrete variables in our data set and also shows how the customer's credit rating is distributed within each of the factors of each of the different variables. This is an extremely powerful function for quick data analysis as it is very easy to pick out patterns within the data. Immediately we can see that having a checking account balance of < 0 dm has a large proportion of "bad" rated customers. Similarly having more than 1,000 dm in your savings account makes you far less likely to have a "bad" credit rating.

plot_histogram

Previously I demonstrated the use of plot_bar to plot the frequency distributions of all of the discrete variables within a data set. The plot_histogram function does exactly the same thing only for the continuous variables within a data set. Between the two functions plot_bar and plot_histogram I could have greatly reduced the amount of coding needed to produce my analysis in part 1 and this is the goal of DataExplorer,

to automate as much of the EDA process as possible so that you can focus on extracting insights from the data. I will now demonstrate the plot histogram function on the German Credit data set.

```
# running the plot histogram function on the German Credit data
plot_histogram(german_df)
```



This is another useful function which can be used to quickly and efficiently plot the histogram of every continuous variable within a data set.

correlation_plot

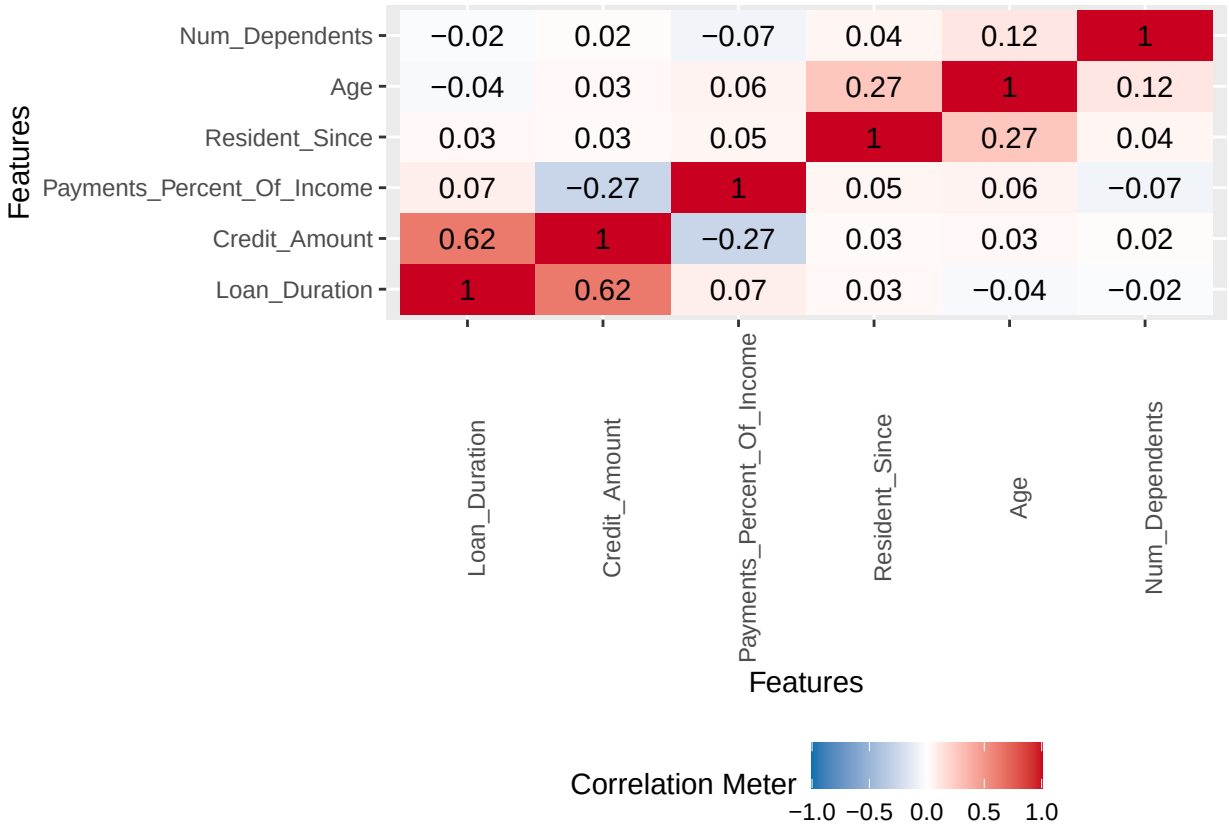
The final function within the EDA suite of functions that I will demonstrate within the DataExplorer package is the correlation_plot function. correlation_plot takes a data frame as its only required argument and produces a pair-wise correlation heat-map of all the variables within the data. This is a really useful function to quickly visualise which variables are most correlated with one another. Highly positively correlated variables appear as a dark red whereas the highly negatively correlated variables appear blue.

There is also options to ignore variables which have too many sub-categories using the "maxcat" argument (the number of categories can be specified by the user). This is a useful argument to keep the correlation plot

from becoming too large (each category within a categorical variable is plotted as its own sub-variable (one-hot encoding)). You also have the ability of plotting just categorical, just numerical or both on the correlation heat-map by passing the "type" argument to function. Note that DataExplorer defines categorical variables as "Discrete" and numerical variables as "Continuous", therefore by passing type = "c" to the correlation plot function we are telling R to only show the continuous i.e. numerical variables correlation plot.

I will now demonstrate the use of this function on the numerical variables in the German Credit data set to identify any highly correlated variables.

```
# Plotting the correlation plot for the numerical variables in the German Credit data
plot_correlation(german_df, type = "c")
```



As we can see from the correlation plot above many of the numerical variables in the German credit data are uncorrelated. We do see a fairly strong correlation between credit amount and loan duration however, with a correlation value of .62. If we were conducting any statistical regressions or performing machine learning algorithms we would have to consider removing one of these variables or conducting a PCA (principle component analysis) to separate the correlated part of the variables from the uncorrelated part and then remove the correlated part.

There are several other functions within DataExplorer for performing EDA such as conducting a Principle Component Analysis as I mention above along with creating box plots and scatter plots of the variables within the data set. There is also a qq-plot function within DataExplorer. The qq-plot function enables you to visualize the deviation of variables in your data set from a given probability distribution (e.g. Normal,Student-t,Poisson,Beta etc.). This can be useful to test whether a certain assumption about a variable i.e. that it is normally distributed is a fair assumption to make. However, I will not be demonstrating the use of these functions on the German Credit data set but I mention them for completeness sake.

Feature Engineering

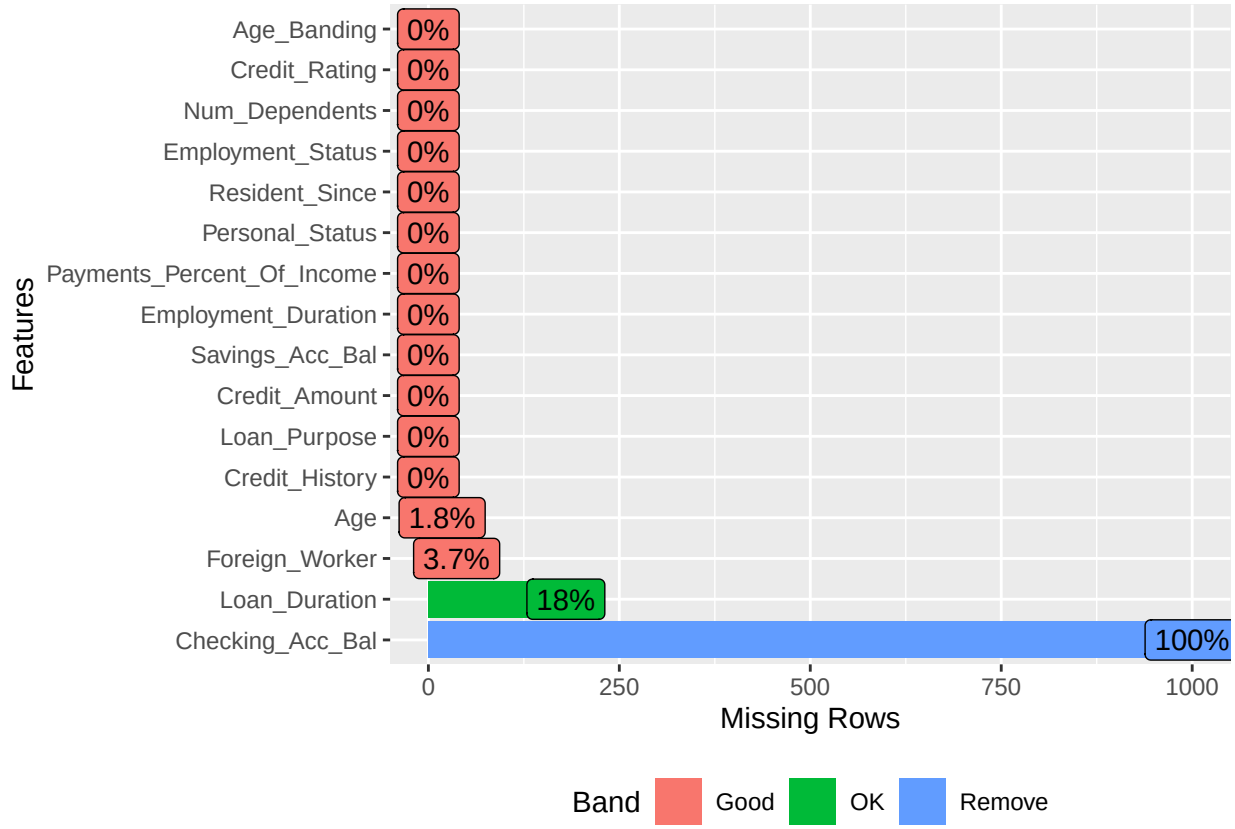
There are two functions for feature engineering that I will demonstrate that are available in the DataExplorer function, namely; set_missing and dummify. Feature engineering involves the creation of new features (variables) from the ones that exist already in the data. This can be done by creating stand-alone new variables based on existing features in the data or by updating existing variables in some way. The set_missing function alters original variables whereas the dummify creates new variables based on ones already present in the data.

set_missing

The set_missing function is used to impute missing data in our data set. This is where we want to replace NAs in our data set with other values such as 0, the column mean or by a more complicated process like using a K-nearest-neighbours algorithm to make the best guess of what the value should be based on observations that our similar to our current observation. The handy thing about the set_missing function is that it can take both numerical and string values to impute the missing data with and then automatically replaces the NAs in our variables based on whether they are categorical or numerical columns.

As I have mentioned above there are many complex methods that can be used for data imputation, however in order to demonstrate the use of the set_missing function on the test data frame that I created from the German Credit data, I will keep it simple. I will replace all the missing numerical values with 0 and the missing categorical values with "Unknown".

```
# firstly I will use the plot missing function to remind
# ourselves of the status of missing data in the test data-frame

# setting all the "N" foreign worker values to NA in order to have missing
# values in a categorical variable
test_df <- test_df %>%
  mutate(Foreign_Worker = case_when(as.character(Foreign_Worker) == "N" ~ NA_character_,
                                    TRUE ~ as.character(Foreign_Worker)))
plot_missing(test_df)
```



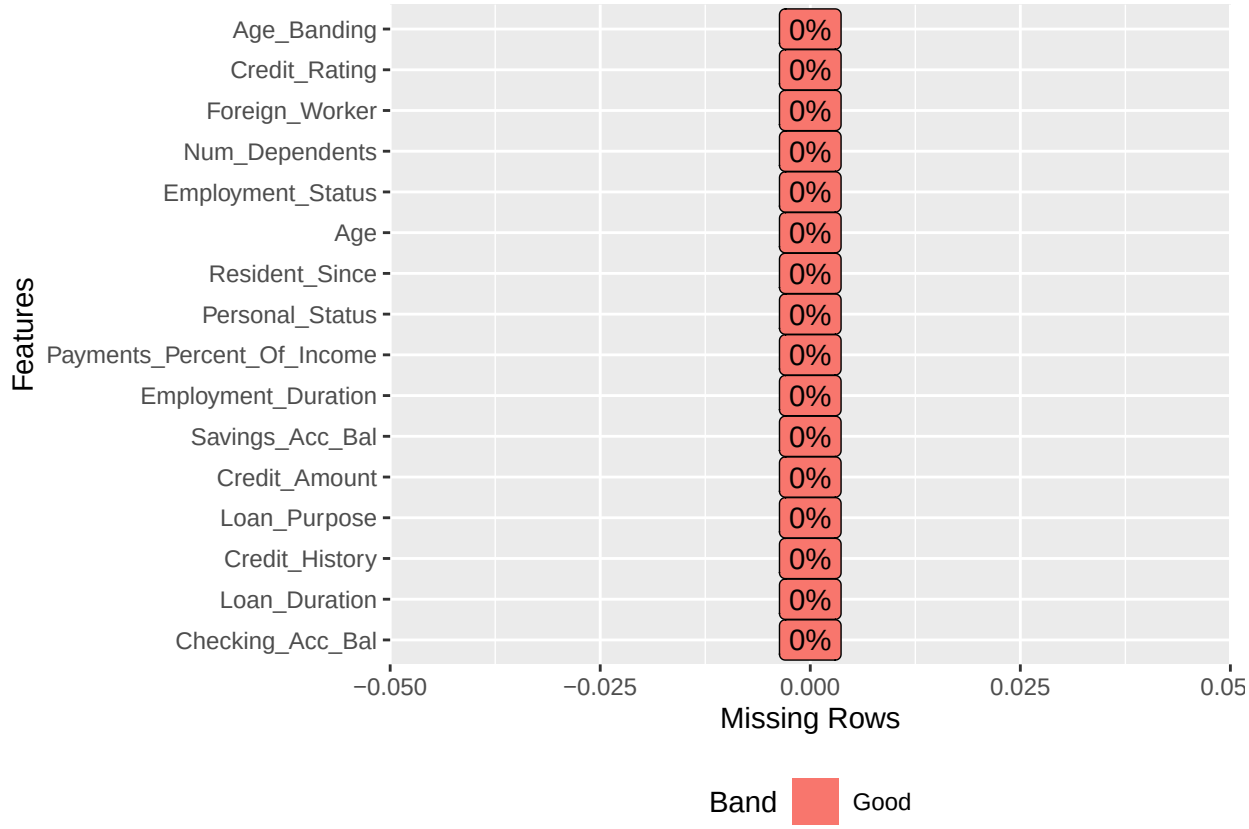
Observing the plot above, the entire Checking account balance is missing, along with 18% of loan duration, 3.7% of foreign worker and 1.8% of the age variable. Now I will impute these variables with 0 if they are numerical and "Unknown" if they are categorical.

```
# Demonstrating the set missing function on our test data set
test_df <- set_missing(test_df,list(0,"Unknown"))
```

```
## Column [Checking_Acc_Bal]: Set 1000 missing values to 0

## Column [Loan_Duration]: Set 180 missing values to 0

## Column [Age]: Set 18 missing values to 0

## Column [Foreign_Worker]: Set 37 missing values to Unknown
```

We can see that the set_missing function has worked and has made the adjustments outlined above. I will now re-plot the missing values using the plot_missing function to see what has changed. We expect that none of the variables in the data set will have any missing values now and if we observe the plot below we can see that this is the case.

```
# Using the plot missing function to compare results before and after
# applying the set missing function to impute missing values
plot_missing(test_df)
```



dummify

The last function within the DataExplorer package that I will be demonstrating is the dummify function. The dummify function allows us to perform "one-hot encoding" or more simply create binary variables (0,1) from all of the categorical variables in a data set. This can be especially useful when performing statistical analyses that involves machine learning. The reason we would want to convert categorical variables to a series of dummy variables is because many machine learning algorithms cannot operate on categorical data and require numerical variables only.

The dummify function makes the process of one-hot encoding very simple by creating binary variables from each categorical variable within the data set while preserving the remaining structure of the data. You can also pass a "maxcat" argument which will tell the dummify function to leave categorical variables with x or more sub-categories unchanged (x to be specified by the user). I will now create a "dummified" version of the German credit data set using DataExplorer.

```
# transforming all categorical variables in the German credit data set
# to binary variables using dummify
dummy_german_df <- dummify(german_df)

# using the str function to display the structure of the data set
str(dummy_german_df)
```

```
## 'data.frame':    1000 obs. of  59 variables:
##  $ Loan_Duration                                                  : int  6 48 12 42 24 36 24 36 12
##  $ Credit_Amount                                                  : int  1169 5951 2096 7882 4870 9
```

```
##  $ Payments_Percent_Of_Income                                      : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ Resident_Since                                                  : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                                                             : int  67 22 49 45 53 35 53 35 6
##  $ Num_Dependents                                                  : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Checking_Acc_Bal_...200.DM                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Checking_Acc_Bal_..0.DM                                         : int  1 0 0 1 1 0 0 0 0 0 ...
##  $ Checking_Acc_Bal_..200.DM                                       : int  0 1 0 0 0 0 0 1 0 1 ...
##  $ Checking_Acc_Bal_No.Checking.Account                            : int  0 0 1 0 0 1 1 0 1 0 ...
##  $ Credit_History_All.credits.at.this.bank.paid.duly               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Credit_History_All.credits.paid.duly                            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Credit_History_Critical.account                                 : int  1 0 1 0 0 0 0 0 0 1 ...
##  $ Credit_History_Delayed.payments.in.the.past                     : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ Credit_History_Existing.credits.paid.duly.until.now             : int  0 1 0 1 0 1 1 1 1 0 ...
##  $ Loan_Purpose_Business                                           : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Loan_Purpose_Car..new.                                          : int  0 0 0 0 1 0 0 0 0 1 ...
##  $ Loan_Purpose_Car..used.                                         : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ Loan_Purpose_Domestic.Appliance                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Loan_Purpose_Education                                          : int  0 0 1 0 0 1 0 0 0 0 ...
##  $ Loan_Purpose_Furniture.Equipment                                : int  0 0 0 1 0 0 1 0 0 0 ...
##  $ Loan_Purpose_Other                                              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Loan_Purpose_Radio.TV                                           : int  1 1 0 0 0 0 0 0 1 0 ...
##  $ Loan_Purpose_Repairs                                            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Loan_Purpose_Retraining                                         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Savings_Acc_Bal_..100.DM                                        : int  0 1 1 1 1 0 0 1 0 1 ...
##  $ Savings_Acc_Bal_..1000.DM                                       : int  0 0 0 0 0 0 0 0 0 1 0 ...
##  $ Savings_Acc_Bal_100...500.DM                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Savings_Acc_Bal_500...1000.DM                                   : int  0 0 0 0 0 0 0 1 0 0 0 ...
##  $ Savings_Acc_Bal_No.Savings.Account                              : int  1 0 0 0 0 0 1 0 0 0 0 ...
##  $ Employment_Duration_...7.years                                  : int  1 0 0 0 0 0 0 1 0 0 0 ...
##  $ Employment_Duration_..1.year                                    : int  0 0 0 0 0 0 0 0 0 0 0 ...
##  $ Employment_Duration_..4.years                                   : int  0 1 0 0 1 1 0 1 0 0 ...
##  $ Employment_Duration_..7.years                                   : int  0 0 1 1 0 0 0 0 1 0 ...
##  $ Employment_Duration_Unemployed                                  : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal_Status_Female...Divorced.Separated.Married             : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ Personal_Status_Male...Divorced.Separated                       : int  0 0 0 0 0 0 0 0 1 0 ...
##  $ Personal_Status_Male...Married.Widowed                          : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal_Status_Male...Single                                   : int  1 0 1 1 1 1 1 1 0 0 ...
##  $ Housing_Free                                                    : int  0 0 0 1 1 1 0 0 0 0 ...
##  $ Housing_Owned                                                   : int  1 1 1 0 0 0 1 0 1 1 ...
##  $ Housing_Rent                                                    : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ Employment_Status_Management.Self.Employed.Highly.Skilled.Officer: int  0 0 0 0 0 0 0 1 0 1 ...
##  $ Employment_Status_Skilled.Employee.Official                     : int  1 1 0 1 1 0 1 0 0 0 ...
##  $ Employment_Status_Unemployed.Unskilled...Non.Resident           : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Employment_Status_Unskilled...Resident                          : int  0 0 1 0 0 1 0 0 1 0 ...
##  $ Foreign_Worker_N                                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Foreign_Worker_Y                                                : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Credit_Rating_Bad                                               : int  0 1 0 0 1 0 0 0 0 1 ...
##  $ Credit_Rating_Good                                              : int  1 0 1 1 0 1 1 1 1 0 ...
##  $ Gender_Female                                                   : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ Gender_Male                                                     : int  1 0 1 1 1 1 1 1 1 1 ...
##  $ Age_Banding_0...20.years                                        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Age_Banding_21...30.years                                       : int  0 1 0 0 0 0 0 0 0 1 ...
##  $ Age_Banding_31...40.years                                       : int  0 0 0 0 0 1 0 1 0 0 ...
##  $ Age_Banding_41...50.years                                       : int  0 0 1 1 0 0 0 0 0 0 ...
```

```
##  $ Age_Banding_51...60.years                                    : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ Age_Banding_61...70.years                                    : int  1 0 0 0 0 0 0 0 1 0 ...
##  $ Age_Banding_71...80.years                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

As we can see from the output above the dummify function has transformed our categorical variables into a series of binary variables. For example, if we observe the housing variables, previously we had one variable which could take the values "Free","Owned" and "Rent". Following the transformation by dummify, we now have 3 variables namely; "Housing_Free", "Housing_Owned" and "Housing_Rent" which can take the values 0 and 1. This is an extremely useful function when pre-processing your data before building a machine learning model.

This is the final function from the Featuring Engineering suite of functions within DataExplorer however there are others such as drop_columns and update_columns which I mention for completeness. As I mentioned during the introduction to the DataExplorer package there are functions that can be used to create reports which bring together most if not all of the functions I have previously demonstrated. The function create_report along with configure_report can be used to customise and produce exploratory data analysis reports. I will not be demonstrating their use but it is important to know that all of my above demonstrations can be streamlined into one single function.

Summary

In summary, I believe DataExplorer is an extremely useful package for data pre-processing and data exploration and could save an analyst lots of time which can instead be used to derive insights from the data. This is the intention of DataExplorer and I believe it performs really well in that regard. There is no doubt I will be using functions from the DataExplorer package in the future.

Part 3: Functions/Programming

Introduction

For the final part of my project I will write a function to perform three different classification algorithms and implement them on the German Credit data set. The main purpose of the data set and the reason it is available from UCI's machine learning repository, is to train and test various machine learning algorithms. The "Credit_Rating" variable is our target variable (this is the one we wish to explain / predict) and the remaining variables will be our explanatory variables (these are used to explain / predict the target variable).

As the credit rating variable is binary and can take values of either "good" or "bad", we have a "Classification" problem. These are problems in which we wish to classify the target variable into certain cohorts based on the values of the explanatory variables. There are a plethora of classification algorithms available in various packages in R, however to keep it simple I have chosen to use just two and compare their results. The two algorithms I have chosen to implement are **logistic regression** and **decision tree**.

I will create a function to perform each of the algorithms and record their respective results. Then I will create another function which runs each of the individual classification functions and collates the results of each one and producing an object of S3 class. Following this I will produce three distinct methods for this S3 class; a print method, a summary method and finally a plot method. Firstly, I must convert the credit rating to a binary variable in order to perform the two classification methods. We wish to predict whether or not a customer has a "bad" credit rating and therefore I map the value 1 to the "bad" cohort and 0 to the "good" cohort.

```r
# converting the credit rating variable to binary
# We want to predict whether the customer has a bad credit rating
# therefore we set that to be 1 and good becomes 0
german_df <- german_df %>%
  mutate(Credit_Rating = case_when(Credit_Rating == "Bad" ~ 1,
                                   Credit_Rating == "Good" ~ 0))
```

```r
# I drop the age banding column as it is bound to be highly correlated with
# the age column (as its derived from this column)
german_df <- german_df %>%
  select(-Age_Banding)
```

```r
# Using the DataExplorer package I will transform the categorical variables
# into dummy variables before fitting any models
german_df <- dummify(german_df)
```

I also wish to create a function which will split up my data into training and testing data randomly into proportions of my choosing. I have created a function below which splits the data into training and test sets with a proportion "x" to be specified by the user. I also set a seed in order to ensure my results are reproducible.

```r
# creating a function to split a data set into training and test sets

train_test <- function(data,x){
  # setting the seed so results can be reproduced
  set.seed(1759)

  # selecting x% of rows of n rows randomly
  rand_rows <- sample.int(n = nrow(data),size = round(x*nrow(data),0),replace = FALSE)

  # selecting training set
  train <- data[rand_rows,]

  # selecting test set
  test <- data[-rand_rows,]

  return(list(train,test))
}

# running function and outputting result
result_train_test <- train_test(german_df,0.8)

# creating a training set with 80% of the data
train_german_df <- result_train_test[[1]]

# creating a test set with the remaining 20%
test_german_df <- result_train_test[[2]]
```

Classification Functions

Logistic Regression

Below I create the logistic regression function which builds a logistic model using the training data set and then predicts if a customer is "bad" or "good" based on the test set. The function then computes various performance metrics such as accuracy which is the number of true positives (TP) plus true negatives (TN) over the total number of positive and negative samples (i.e. the total number of rows in the test data). It also computes the mis-classification rate (1 - accuracy), the true and false positive rates and the AUC, (area under the curve) which is one of the most important performance metrics for binary classifiers. The function also computes the TPR and FPR rates for a wide variety of TPR values which we can plot on a ROC curve later on.

```
# Defining a function to perform a logistic regression

# Function will take 2 arguments, train and test
# train is the training data set used to train the model
# test is the testing data set used to evaluate model performance
# For the purposes of this project I assume there are no missing values
# in the data set and also assume all categorical variables have been
# converted to binary dummy variables

log_reg_fun <- function(train,test){
  # We can use the glm function along with family = binomial to train a logistic model
  mod <- glm(formula = Credit_Rating ~ .,
             family = binomial(link = "logit"),
             data = train)

  # predicting the credit rating on the test data
  res <- predict(mod,newdata = test,type = "response")

  # We want to predict whether a customer has a "bad" or "good" credit rating
  res <- ifelse(res > 0.5,1,0)

  # computing model accuracy
  mod_acc <- round(mean(res == test$Credit_Rating),3)

  # computing misc-classification rates
  mod_mis_class <- round(mean(res != test$Credit_Rating),3)

  # creating a confusion matrix
  cm_table <- table("target" = test$Credit_Rating,
                    "prediction" = res)
  row.names(cm_table) <- c("Good","Bad")
  colnames(cm_table) <- c("Good","Bad")

  # calculating tp,fn,fp,tn counts (true positive,false negative,false positive,true negative)
  # tp = target = 1, pred = 1
  tp <- cm_table[2,2]
  # fn = target = 1, pred = 0
  fn <- cm_table[2,1]
  # fp = target = 0, pred = 1
  fp <- cm_table[1,2]
  # tn = target = 0, pred = 0
```

```
  tn <- cm_table[1,1]

  # calculating TPR and FPR rates (true and false positive rates)
  # also known as recall and fallout
  TPR <- round(tp / (tp+fn),3)
  FPR <- round(fp / (fp+tn),3)

  # calculating auc (area under the curve) using the ROCR package

  # predicting the model on the test data again
  pred <- predict(mod,newdata = test,type = "response")
  # creating a prediction object using the ROCR package
  preds <- prediction(pred,test$Credit_Rating)
  # calculating auc
  auc <- performance(preds,measure = "auc")
  auc <- round(auc@y.values[[1]],3)

  # calculating x and y values for the ROC plot (plot fo tpr vs fpr for various values of tpr)
  roc <- performance(preds,measure = "tpr",x.measure = "fpr")

  # collating and returning the results
  return(list(mod_acc,mod_mis_class,TPR,FPR,auc,roc,cm_table))
}
```

Decision Tree

Similar to the logistic regression function above, I have produced a function to perform a decision tree classification algorithm and compile the results. I return the same metrics in order to be able to conduct a comparison between the two different methods.

```
# Defining a function to perform a decision tree classification
# Uses the rpart package to do this

# Function will take 2 arguments, train and test
# train is the training data set used to train the model
# test is the testing data set used to evaluate model performance
# For the purposes of this project I assume there are no missing values
# in the data set and also assume all categorical variables have been
# converted to binary dummy variables

tree_fun <- function(train,test){
  # We can use the rpart function along with method = class to train a decision tree model
  mod <- rpart(formula = Credit_Rating ~ .,
               data = train,
               method = "class") # classification problem

  # predicting the credit rating on the test data
  res <- predict(mod,newdata = test,type = "class")

  # computing model accuracy
  mod_acc <- round(mean(res == test$Credit_Rating),3)

  # computing misc-classification rates
```

```
  mod_mis_class <- round(mean(res != test$Credit_Rating),3)

  # creating a confusion matrix
  cm_table <- table("target" = test$Credit_Rating,
                    "prediction" = res)
  row.names(cm_table) <- c("Good","Bad")
  colnames(cm_table) <- c("Good","Bad")

  # calculating tp,fn,fp,tn counts (true positive,false negative,false positive,true negative)
  # tp = target = 1, pred = 1
  tp <- cm_table[2,2]
  # fn = target = 1, pred = 0
  fn <- cm_table[2,1]
  # fp = target = 0, pred = 1
  fp <- cm_table[1,2]
  # tn = target = 0, pred = 0
  tn <- cm_table[1,1]

  # calculating TPR and FPR rates (true and false positive rates)
  # also known as recall and fallout
  TPR <- round(tp / (tp+fn),3)
  FPR <- round(fp / (fp+tn),3)

  # calculating auc (area under the curve) using the ROCR package

  # predicting the model on the test data again this time using
  # type = prob in order to get the probabilities of each observation
  # being from the "bad" class (this is in the second column)
  pred <- predict(mod,newdata = test,type = "prob")[,2]
  # creating a prediction object using the ROCR package
  preds <- prediction(pred,test$Credit_Rating)
  # calculating auc
  auc <- performance(preds,measure = "auc")
  auc <- round(auc@y.values[[1]],3)

  # calculating x and y values for the ROC plot (plot fo tpr vs fpr for various values of tpr)
  roc <- performance(preds,measure = "tpr",x.measure = "fpr")

  # collating and returning the results
  return(list(mod_acc,mod_mis_class,TPR,FPR,auc,roc,cm_table))
}
```

Next I will create the function which will perform both the logistic and decision tree algorithms, collate their results together and finally create an S3 class object which I will write summary methods for.

```
# Producing a function to collate the results of the regressions

classification_summary <- function(train,test){

  # Performing Logistic classification
  log_res <- log_reg_fun(train,test)

  # Performing Decision Tree classification
```

```
  tree_res <- tree_fun(train,test)

  # Model Accuracy
  Accuracy <- c("Logistic" = log_res[[1]],
                "Decision Tree" = tree_res[[1]])

  # Model Error (mis-classification rates)
  Error <- c("Logistic" = log_res[[2]],
             "Decision Tree" = tree_res[[2]])

  # Model TPR
  TPR <- c("Logistic" = log_res[[3]],
             "Decision Tree" = tree_res[[3]])

  # Model FPR
  FPR <- c("Logistic" = log_res[[4]],
             "Decision Tree" = tree_res[[4]])

  # Model AUC
  AUC <- c("Logistic" = log_res[[5]],
             "Decision Tree" = tree_res[[5]])

  # Model ROC Curves
  ROC_Curve <- c("Logistic" = log_res[[6]],
             "Decision Tree" = tree_res[[6]])

  # Model Confusion Matrices
  Confusion_Matrix <- list(log_res[[7]],tree_res[[7]])

  # Summary table of results
  Summary_Table <- rbind(Accuracy,Error,TPR,FPR,AUC)

  # generating results
  results <- list("Accuracy" = Accuracy,
                  "Mis-classification Error" = Error,
                  "TPR" = TPR,
                  "FPR" = FPR,
                  "AUC" = AUC,
                  "ROC Curve points" = ROC_Curve,
                  "Confusion Matrices" = Confusion_Matrix,
                  "Summary Table" = Summary_Table)

  # setting the class of results to be class = "classification"
  class(results) <- "classification"

  invisible(results)
}

# setting a variable equal to the output of the function to demonstrate
# the summary methods later on
reg_results <- classification_summary(train_german_df,test_german_df)
x <- classification_summary(train_german_df,test_german_df)
```

```r
# printing the class of the reg_results variable - "classification"
print(class(reg_results))
```

```
## [1] "classification"
```

Summary Methods for S3 Class

Print Method

I will now write three functions to produce print, summary and plot methods for my newly developed "classification" class. The first of these functions, print, will be fairly basic and simply output the accuracy of the logistic regression model and decision tree model. I then demonstrate the use of this print function on the "reg_results" variable which was set equal to the output of my main summary function and has a class of "classification".

```r
# Creating a print method for the "classification" class
print.classification <- function(x){
  cat("The accuracy of the classification models are shown below.\n\n")
  print(x$Accuracy)
  cat("\n")
  if (x$Accuracy["Logistic"] > x$Accuracy["Decision Tree"]) {
    cat("The Logistic Model is more accurate")
  } else {
    cat("The Decision Tree Model is more accurate")
  }
}

# Demonstrating the print function on the summary results variable
print(reg_results)
```

```
## The accuracy of the classification models are shown below.
##
##      Logistic Decision Tree
##         0.750         0.735
##
## The Logistic Model is more accurate
```

As we can see from the output of the print method above, the logistic model has an accuracy of 75% whereas the decision tree has an accuracy of 73.5%. In terms of the logistic model an accuracy of 75% means that model predicted 3 out of every 4 of the customer's credit ratings correctly and mis-classified 1 out of every 4. The decision tree model is slightly worse but still relatively good. An accuracy of 75% is not bad however I believe model accuracy could be improved if K-fold cross validation was performed during the training phase in order to fine-tune the parameters of the model.

Summary Method

I will now create the summary method for my the "classification" S3 class. I want the summary method to display various performance metrics such as accuracy, mis-classification rate, the true and false positive rates which are the number of correctly identified positives divided by the total number of positive samples. In the case of the German Credit data, this would be the total number of correctly identified "bad" credit ratings divided by the total number of "bad" credit ratings in the test data set. The false positive rate follows the

same logic, the number of incorrectly identified "good" credit ratings divided by the total number of "good" credit ratings in the test data. I also wish to output the area under the curve or (AUC).

The AUC is a very powerful metric in terms of binary classifier performance and represents the area underneath the ROC (receiver operating characteristic curve). The AUC can take values on [0,1] and can be interpreted as the expected proportion of positives i.e. "bad" credit customers ranked before a uniformly drawn random negative i.e. "good" credit customer. A model which randomly assigns customers as either "bad" or "good" would have an AUC of 0.5 and a model which perfectly predicts customer's credit ratings would have an AUC of 1. The larger the AUC the better as this means are model can better predict whether or not a customer has a "bad" credit rating.

```
# Creating a summary method for the "classification" class
summary.classification <- function(x){
  cat("The summary performance metrics of the models are shown below.\n\n")
  print(x$`Summary Table`)
  cat("\n")
}

# Demonstrating the summary function on the reg_results variable
summary(reg_results)
```

```
## The summary performance metrics of the models are shown below.
##
##          Logistic Decision Tree
## Accuracy    0.750         0.735
## Error       0.250         0.265
## TPR         0.446         0.536
## FPR         0.132         0.188
## AUC         0.749         0.727
```

As we can see above the logistic regression outperforms the decision tree in every metric aside from the true positive rate (TPR). This means the decision tree model is actually better at predicting "bad" credit rating customer's than the logistic regression model and this is the more important classification due to its commercial nature. What I mean by this is that it is less of a problem to classify a "good" customer as being "bad" than classifying a "bad" customer as being "good". The reason for this is that we will likely lend money out to the "good" customers if we are a bank and if we have incorrectly identified a "bad" customer as being "good" then we will lend out money to a customer with a bad credit rating and will be far more likely to lose money on that loan. Conversely, if we identified a "good" customer as being "bad" we will just not lend any money to this customer. While there is an opportunity cost in terms of interest we could earn by loaning money to a good customer, there is no actual credit risk to the bank as we haven't lent any money.

We also see that the AUC's of both the logistic and decision tree is above 0.7 and suggests that both models perform relatively well at classifying customer's credit ratings. As the logistic model has a higher AUC and accuracy we can say that this model overall, is a better classifier than the decision tree model. It must be noted that both of these models have been fitted without any model tuning and as such we could likely achieve better results if we first tune the models (using tools such as cross-validation) in order to optimise model parameters.

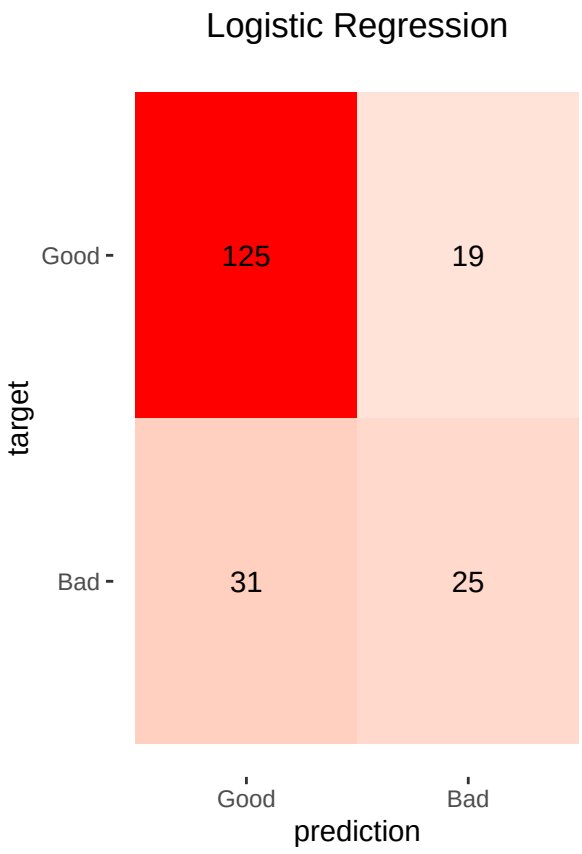
Plot Method

Finally, I will produce a plot method for the "classification" class. I want the plot function to plot the confusion matrices for each of the models and also the ROC curves. The confusion matrix is a table of

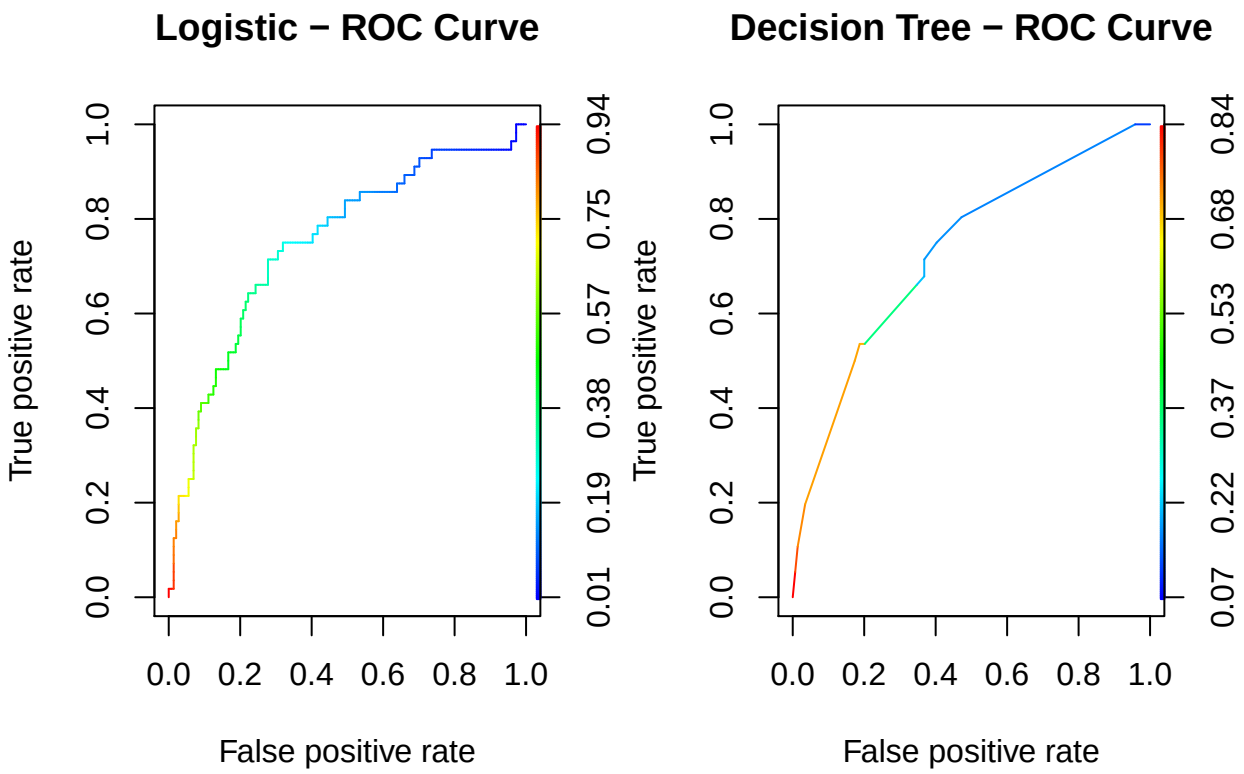
predicted vs actual counts for each of the credit ratings “bad” and “good”. The ROC curve is a plot of the TPR (true positive rate) against the FPR (false positive rate) for various values of the TPR.

```
plot.classification <- function(x){
  # firstly plot the confusion matrices
  cm_logistic <- conf_mat(data = x$`Confusion Matrices`[[1]])
  # specifying "heatmap" type and also adding titles and colour scheme
  cm_logistic_plot <- autoplot(cm_logistic,type = "heatmap") +
    ggtitle("Logistic Regression") +
    theme(plot.title = element_text(hjust = 0.5)) +
    scale_fill_gradient2(low = "blue",high = "red")

  cm_tree <- conf_mat(data = x$`Confusion Matrices`[[2]])
  # specifying "heatmap" type and also adding titles and colour scheme
  cm_tree_plot <- autoplot(cm_tree,type = "heatmap") +
    ggtitle("Decision Tree") +
    theme(plot.title = element_text(hjust = 0.5)) +
    scale_fill_gradient2(low = "blue",high = "red")



  # Next plot the ROC curves already computed by the ROCR package
  par(mfrow = c(1,2))
  plot(plot_grid(cm_logistic_plot,cm_tree_plot))

  par(mfrow = c(1,2))
  plot(x$`ROC Curve points`[[1]],main = "Logistic - ROC Curve",colorize = TRUE)
  plot(x$`ROC Curve points`[[2]],main = "Decision Tree - ROC Curve",colorize = TRUE)

}

# Demonstrating the use of the plot function for the classification class
plot(reg_results)
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```


Logistic Regression
Good
Bad
target
125
19
31
25
Good
Bad
prediction


Decision Tree
Good
Bad
target
117
27
26
30
Good
Bad
prediction



Finally, we can see the output of the plot method above. The plot method outputs each of the model's confusion matrices which is a table of the predicted customer ratings vs the actual (target) customer ratings. We can see that the logistic model correctedly predicted 125 "goos" customers compared to the 117 for the decision tree model. This is related to the true positive rates (TPRs) we observed using the summary method. As we noted previously, the decision tree is better at predicting "bad" customers, and managed to correctly predict 30 bad customer's credit ratings compared to only 25 for the logistic model.

The plot method also produces the ROC (receiver operating characteristic) curves for the two regression models. The ROC curve is a plot of the FPR vs the TPR for various threshold values (i.e. various probabilities at which we classify a "bad" customer). The ROC is the curve for which the area under the curve (AUC) is calculated. The higher the AUC value i.e. the more to the top left of the graph the ROC curve goes, the better the model is at distinguishing between classes. It is difficult to see which model is actually performing better based on the plots above, and that's why it is a good idea to export the AUC as well as the plot as we did in the summary method.

Appendices

Bibliography

In this section I cite and provide a reason for using each of the packages that were not dealt with extensively during the course.

Scales

I use the scales package throughout the project (particularly in part 1) to adjust the x and y axis values and in particular to convert them to percentages for some of the graphs I produced in part 1.

```
citation("scales")
```

```
##
## To cite package 'scales' in publications use:
##
##   Hadley Wickham and Dana Seidel (2020). scales: Scale Functions for
##   Visualization. R package version 1.1.1.
##   https://CRAN.R-project.org/package=scales
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {scales: Scale Functions for Visualization},
##     author = {Hadley Wickham and Dana Seidel},
##     year = {2020},
##     note = {R package version 1.1.1},
##     url = {https://CRAN.R-project.org/package=scales},
##   }
```

Cowplot

I use the cowplot package to arrange several ggplot2 plots in the same figure. The function in particular I used throughout the project was the plot_grid function.

```
citation("cowplot")
```

```
##
## To cite package 'cowplot' in publications use:
##
##   Claus O. Wilke (2020). cowplot: Streamlined Plot Theme and Plot
##   Annotations for 'ggplot2'. R package version 1.1.1.
##   https://CRAN.R-project.org/package=cowplot
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'},
##     author = {Claus O. Wilke},
##     year = {2020},
##     note = {R package version 1.1.1},
##     url = {https://CRAN.R-project.org/package=cowplot},
##   }
```

DataExlporer

I chose the DataExplorer package to demonstrate in my answer to part 2.

```
citation("DataExplorer")
```

```
##
## To cite package 'DataExplorer' in publications use:
##
##   Boxuan Cui (2020). DataExplorer: Automate Data Exploration and
##   Treatment. R package version 0.8.2.
##   https://CRAN.R-project.org/package=DataExplorer
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {DataExplorer: Automate Data Exploration and Treatment},
##     author = {Boxuan Cui},
##     year = {2020},
##     note = {R package version 0.8.2},
##     url = {https://CRAN.R-project.org/package=DataExplorer},
##   }
```

KableExtra

I use the KableExtra in some of the early tables in part 1 in order to fit them nicely on the PDF with pleasant formatting.

```
citation("kableExtra")
```

```
##
## To cite package 'kableExtra' in publications use:
##
##   Hao Zhu (2021). kableExtra: Construct Complex Table with 'kable' and
##   Pipe Syntax. R package version 1.3.4.
##   https://CRAN.R-project.org/package=kableExtra
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {kableExtra: Construct Complex Table with 'kable' and Pipe Syntax},
##     author = {Hao Zhu},
##     year = {2021},
##     note = {R package version 1.3.4},
##     url = {https://CRAN.R-project.org/package=kableExtra},
##   }
```

ROCR

I use the ROCR package in part 3 in order to calculate the AUC (area under the curve statistic) and also to plot the ROC (Receiver Operating Characteristic) curves.

```
citation("ROCR")
```

```
##
## To cite ROCR in publications use:
##
## Sing T, Sander O, Beerenwinkel N, Lengauer T (2005). "ROCR: visualizing
## classifier performance in R." _Bioinformatics_, *21*(20), 7881. <URL:
## http://rocr.bioinf.mpi-sb.mpg.de>.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     entry = {article},
##     title = {ROCR: visualizing classifier performance in R},
##     author = {T. Sing and O. Sander and N. Beerenwinkel and T. Lengauer},
##     year = {2005},
##     journal = {Bioinformatics},
##     volume = {21},
##     number = {20},
##     pages = {7881},
##     url = {http://rocr.bioinf.mpi-sb.mpg.de},
##   }
##
## We have invested a lot of time and effort in creating ROCR, please cite
## it when using it for data analysis.
```

rpart

I use the rpart package in order to fit a decision tree model in part 3.

```
citation("rpart")
```

```
##
## To cite package 'rpart' in publications use:
##
##   Terry Therneau and Beth Atkinson (2019). rpart: Recursive
##   Partitioning and Regression Trees. R package version 4.1-15.
##   https://CRAN.R-project.org/package=rpart
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {rpart: Recursive Partitioning and Regression Trees},
##     author = {Terry Therneau and Beth Atkinson},
##     year = {2019},
##     note = {R package version 4.1-15},
##     url = {https://CRAN.R-project.org/package=rpart},
##   }
```

yardstick

I use the yardstick package in order to graph my basic homemade confusion matrices and make them look more professional.

```
citation("yardstick")
```

```
##
## To cite package 'yardstick' in publications use:
##
##   Max Kuhn and Davis Vaughan (2021). yardstick: Tidy Characterizations
##   of Model Performance. R package version 0.0.9.
##   https://CRAN.R-project.org/package=yardstick
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {yardstick: Tidy Characterizations of Model Performance},
##     author = {Max Kuhn and Davis Vaughan},
##     year = {2021},
##     note = {R package version 0.0.9},
##     url = {https://CRAN.R-project.org/package=yardstick},
##   }
```