# Data Science Assessment Exercise

*Cennen Del Rosario*

## 0. Initial setup

The following libraries, setup and constants are used althrough out the code.

```
library(dplyr)
library(tidyverse)
library(kableExtra)
library(ggplot2)
library(fastDummies)
library(MASS)
library(e1071)
library(randomForest)

DIRECTORY_FOLDER <- "/Users/ken/Documents/Applications/Teradata"
setwd(DIRECTORY_FOLDER)

SEED_NUMBER <- 12345
```

## 1. Data Loading

```
# data loading
raw.data <- read.csv("credit_data_track2_part_2_1.csv", stringsAsFactors = FALSE)
all.variables <- setdiff(names(raw.data), c("X","gender"))
all.variables
```

```
##  [1] "checking_status"        "duration"
##  [3] "credit_history"         "purpose"
##  [5] "credit_amount"          "savings_status"
##  [7] "employment"             "installment_commitment"
##  [9] "personal_status"        "other_parties"
## [11] "residence_since"        "property_magnitude"
## [13] "age"                    "asnm"
## [15] "other_payment_plans"    "housing"
## [17] "existing_credits"       "job"
## [19] "num_dependents"         "own_telephone"
## [21] "foreign_worker"         "accepted"
```

The data that was given appears to have 21 independent variables (instead of 20) and 1 target variable called “accepted”. At first glance, columns “X” and “gender” seem redundant. For the column gender, we can quickly check that it is by applying the following code. The matching yield no rows, indicating that the gender part of the “personal_status” is the same as in the “gender” field.

```r
# verify dependency of gender from personal_status variables
separate(raw.data, "personal_status", into = c("status_gender","others"), sep = "_") %>%
  dplyr::select("status_gender","gender") %>%
    filter(status_gender != gender)
```

```
## [1] status_gender gender
## <0 rows> (or 0-length row.names)
```

## 2. Data Exploration

Next we go to data exploration. The following functions are useful when getting some information about each input variable.

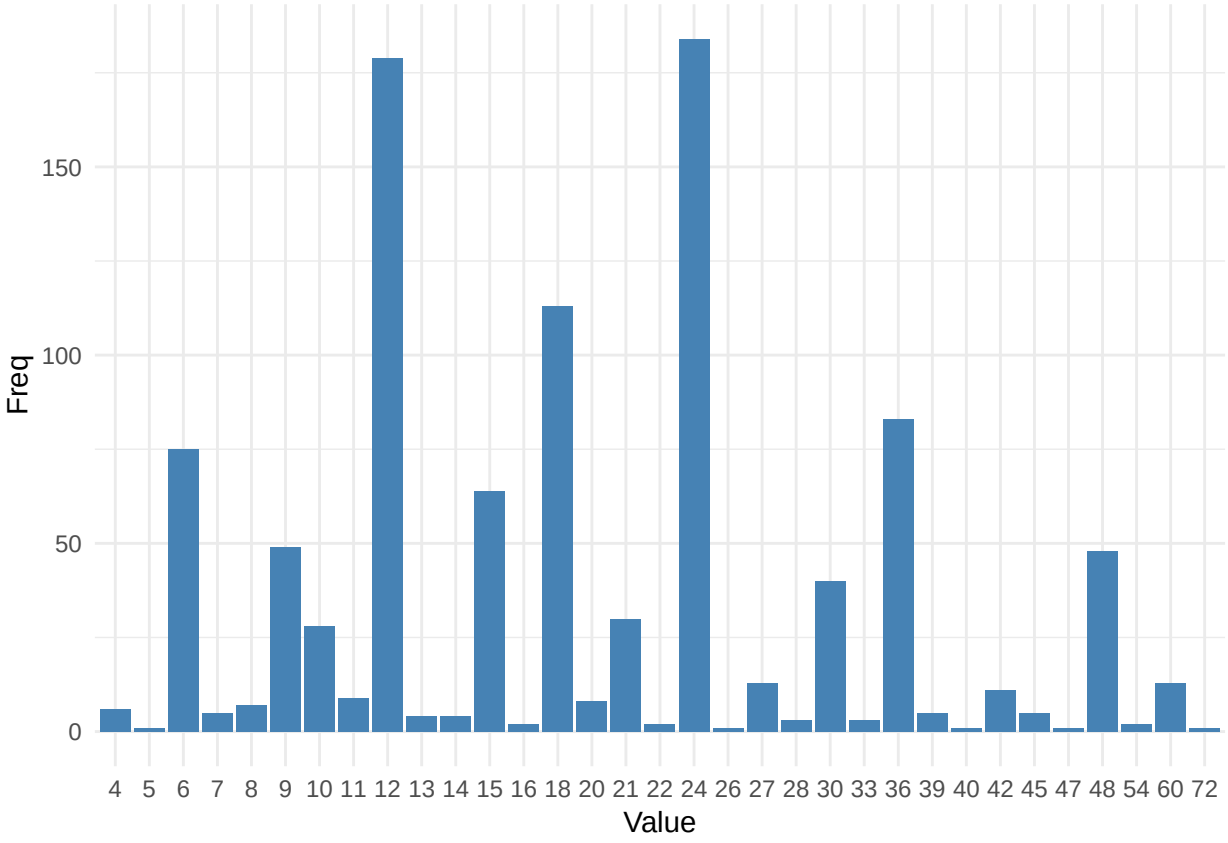
```r
# utility functions for data exploration
plot.table <- function(column){
  count.values <- table(column)
  count.values <- as.matrix(count.values)
  count.values <- round(100*count.values/sum(count.values), digits = 2)
  count.values[,1] <- paste(count.values, "%", sep = "")
  knitr::kable(count.values) %>%
    kable_styling(latex_options = "striped", position = "left")
}

plot.summary <- function(summary){
  values <- as.data.frame(summary) %>%
              dplyr::select(Freq) %>%
                separate(col = "Freq", sep = ":", into = c("Stat", "Value"))
  knitr::kable(values) %>%
    kable_styling(latex_options = "striped", position = "left") %>%
      row_spec(0, bold = TRUE)
}

plot.histo <- function(column, name){
  hist(column, col = c("gold", "gold1", "gold2", "gold3"), main = "", xlab = name)
}

plot.bar <- function(column){
  count.values <- as.data.frame(table(column))
  names(count.values)[1] <- "Value"
  par(mfrow=c(1,2))
  p <- ggplot(data = count.values, aes(x = Value, y = Freq)) +
          geom_bar(stat="identity", fill="steelblue") +
          theme_minimal()

  p
}
```

**Variable 1: checking_status**

| | |
|---|---|
| 0_to_200DM | 26.9% |
| <0DM | 27.4% |
| >200DM | 6.3% |
| None | 39.4% |

At first sight, the categorical values above can be decomposed into dummy variables and assigned with 0 or 1.

**Variable 2: duration**




**Variable 3: credit_history**

| | |
|---|---|
| All_credits_paid_duly | 4.9% |
| Critical_acct_other_credits_existing | 29.3% |
| Delay_in_past | 8.8% |
| Existing_credits_paid_till_now | 53% |
| No_credits_taken_or_all_paid | 4% |

Similarly, values above can also be translated into 0 or 1. Also, it can be observed that most of the applicants have existing credits.

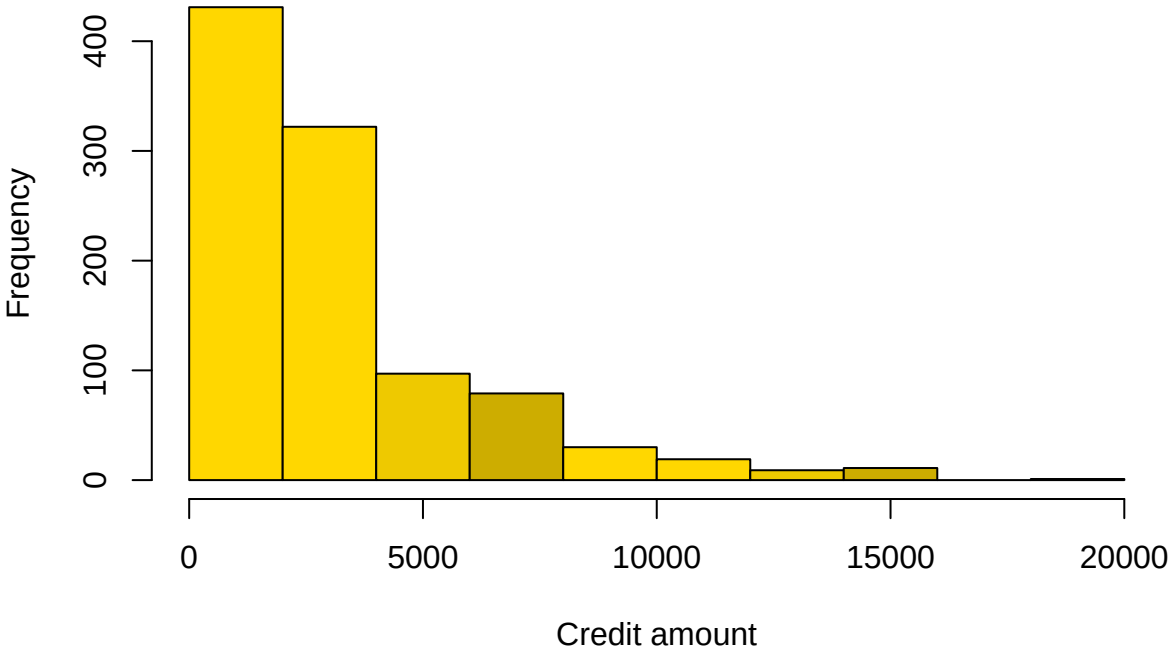
**Variable 4: purpose**

| | |
|---|---|
| business | 9.7% |
| domestic_appliances | 1.2% |
| education | 5% |
| furniture/equipment | 18.1% |
| new_car | 23.4% |
| other | 1.2% |
| radio/television | 28% |
| repairs | 2.2% |
| retraining | 0.9% |
| used_car | 10.3% |

Here, it can be observed that most of the loaners are spending (as they state) to material things (gadgets, car, furniture, etc.) when applying for loan. For the analysis, these values can be decomposed into dummy variables and assigned with 0 or 1 later on.

**Variable 5: credit_amount**

| **Stat** | **Value** |
|---|---|
| Min. | 250 |
| 1st Qu. | 1368 |
| Median | 2320 |
| Mean | 3273 |
| 3rd Qu. | 3972 |
| Max. | 18424 |
| NA's | 1 |




Missing values are handled appropriately.

**Variable 6: savings_status**

| | |
|---|---|
| 100_to_500DM | 10.3% |
| 500_to_1000DM | 6.3% |
| <100DM | 60.3% |
| >1000DM | 4.8% |
| Unknown_or_no_savings_acct | 18.3% |

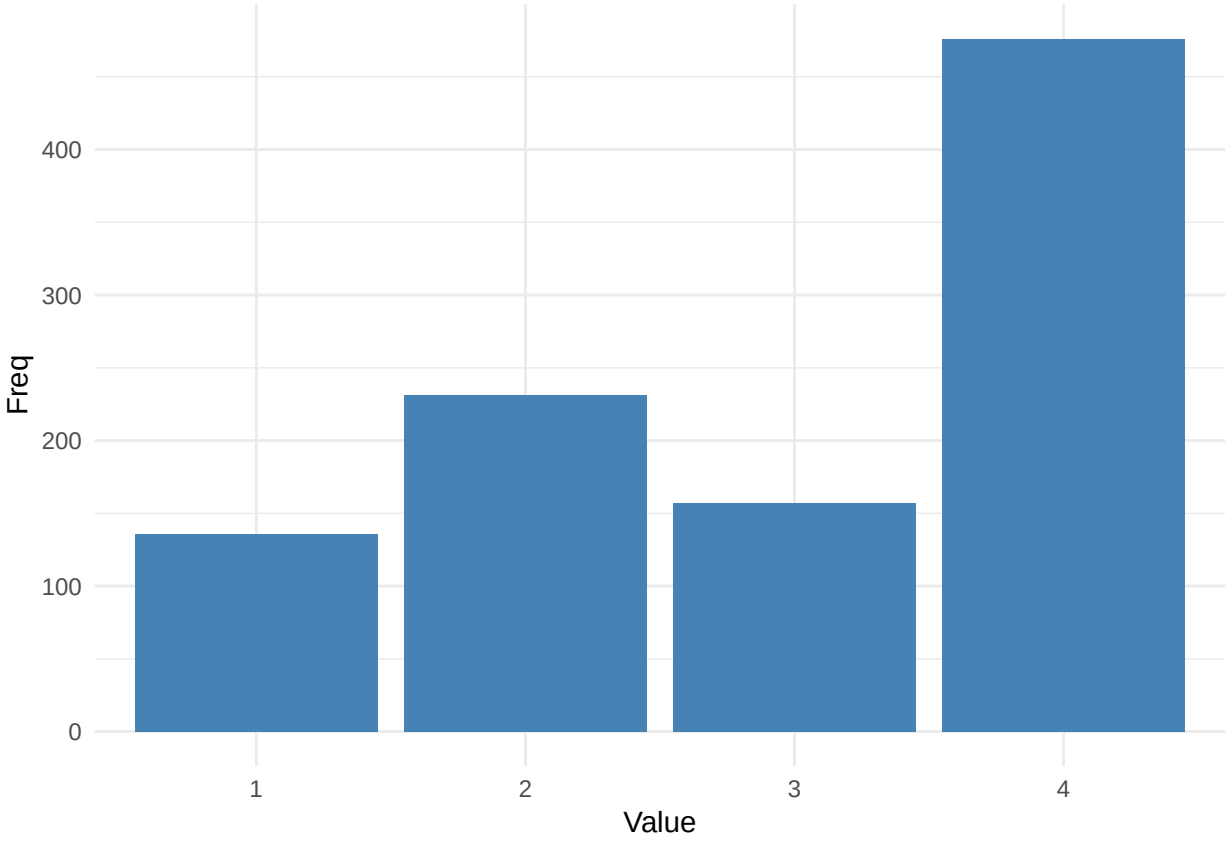
Here, categorical values can be with assigned with 0 or 1. Alternatively, each category can be represented by a numeric value (e.g. mean of the group).

**Variable 7: employment**

| | |
|---|---|
| 1_to_4yrs | 33.9% |
| 4_to_7yrs | 17.4% |
| <1yr | 17.2% |
| >7yrs | 25.3% |
| unemployed | 6.2% |

Similarly, each category here can be represented by a numeric value (e.g. group mean, 0, or 1).

**Variable 8: installment_commitment**




It can be observed that applicants tend to stretch out their loan terms to maximum.

**Variable 9: personal_status**

| | |
|---|---|
| female_divorced/separated/married | 31% |
| male_divorced/separated | 5% |
| male_married/widowed | 9.2% |
| male_single | 54.8% |

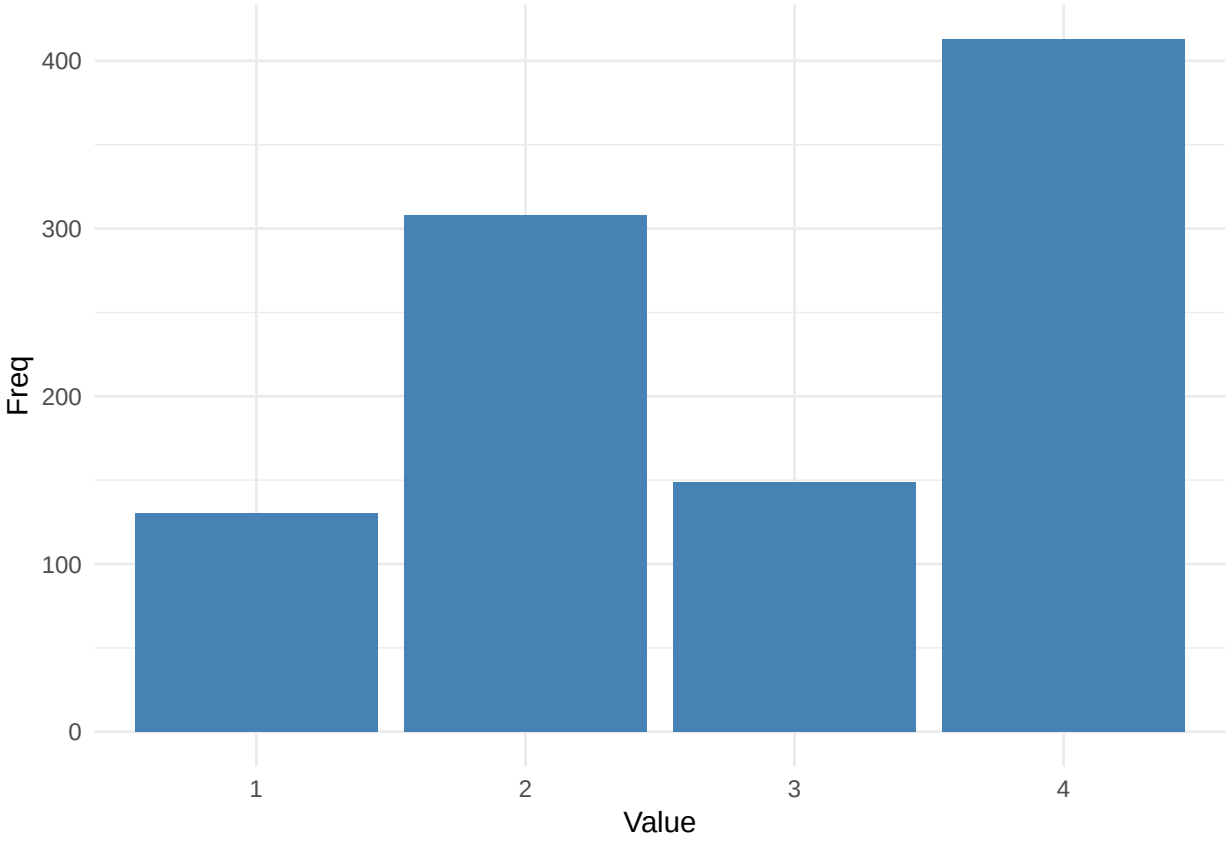
Here, gender is better to be extracted into a separate column. On the other hand, a separate "civil status" variable should be created. This will take the values: single, married, separated, divorced, and widowed, which in turn can be translated into dummy variables. It good to note that seldomly, maried, divorced and widowed men apply for loans.

**Variable 10: other_parties**

| | |
|---|---|
| None | 90.7% |
| co-applicant | 4.1% |
| guarantor | 5.2% |

Categorical values here can also assigned with 0 or 1 for analysis later. Or can be considered for ignoring or regrouping because more than 90% of the data has no other parties involved.

**Variable 11: residence_since**

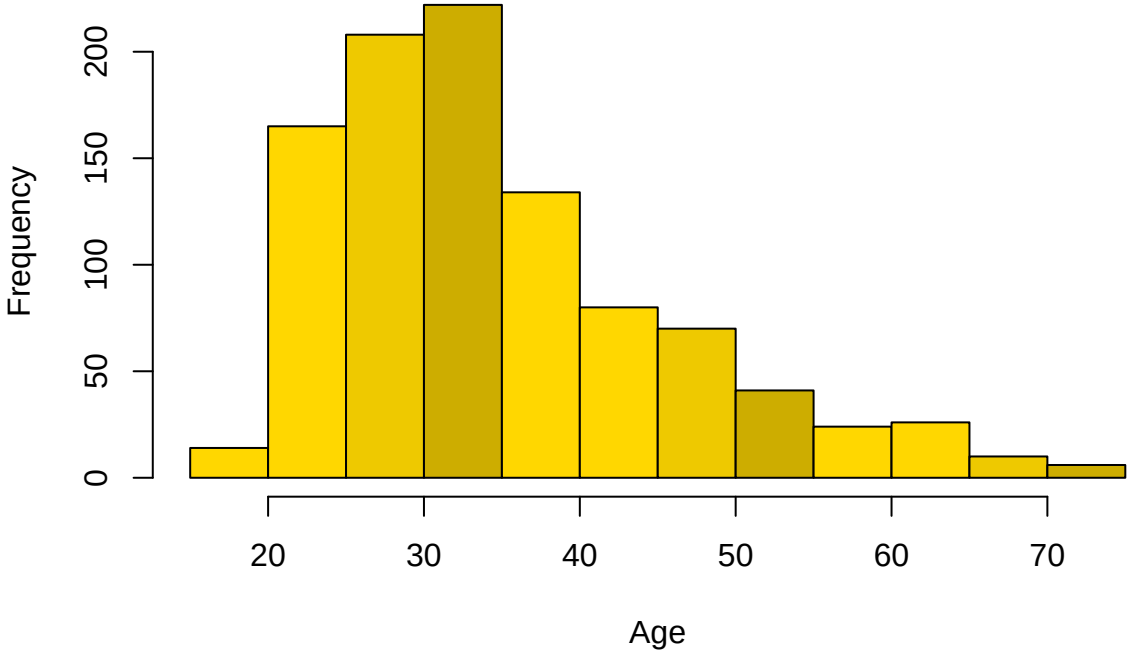


**Variable 12: property_magnitude**

| | |
|---|---|
| building_society_savings_agreement/life_insurance | 23.2% |
| car_or_other_nonsavings | 33.2% |
| real_estate | 28.2% |
| unknown/no_property | 15.4% |

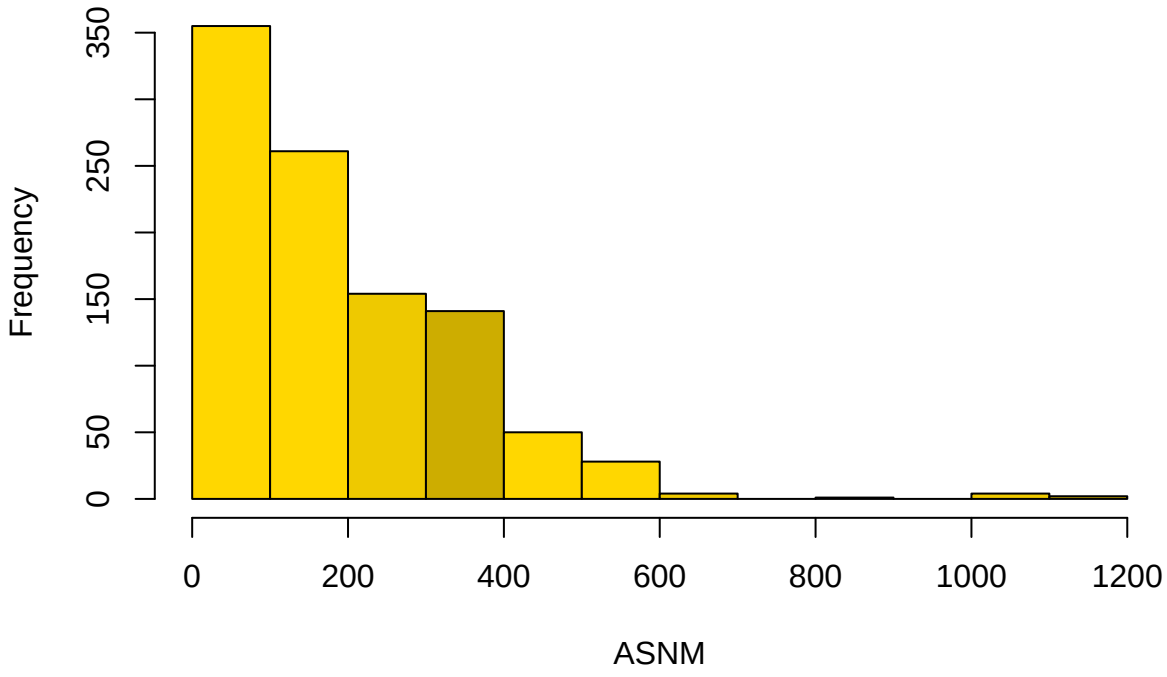
**Variable 13: age**

| Stat | Value |
|---|---|
| Min. | 19.00 |
| 1st Qu. | 27.00 |
| Median | 34.00 |
| Mean | 35.52 |
| 3rd Qu. | 41.00 |
| Max. | 75.00 |




**Variable 14: asnm**

| Stat | Value |
|---|---|
| Min. | 24.0 |
| 1st Qu. | 60.0 |
| Median | 168.0 |
| Mean | 193.9 |
| 3rd Qu. | 280.0 |
| Max. | 1113.0 |




**Variable 15: other_payment_plans**

| | |
|---|---|
| bank | 13.9% |
| none | 81.4% |
| stores | 4.7% |

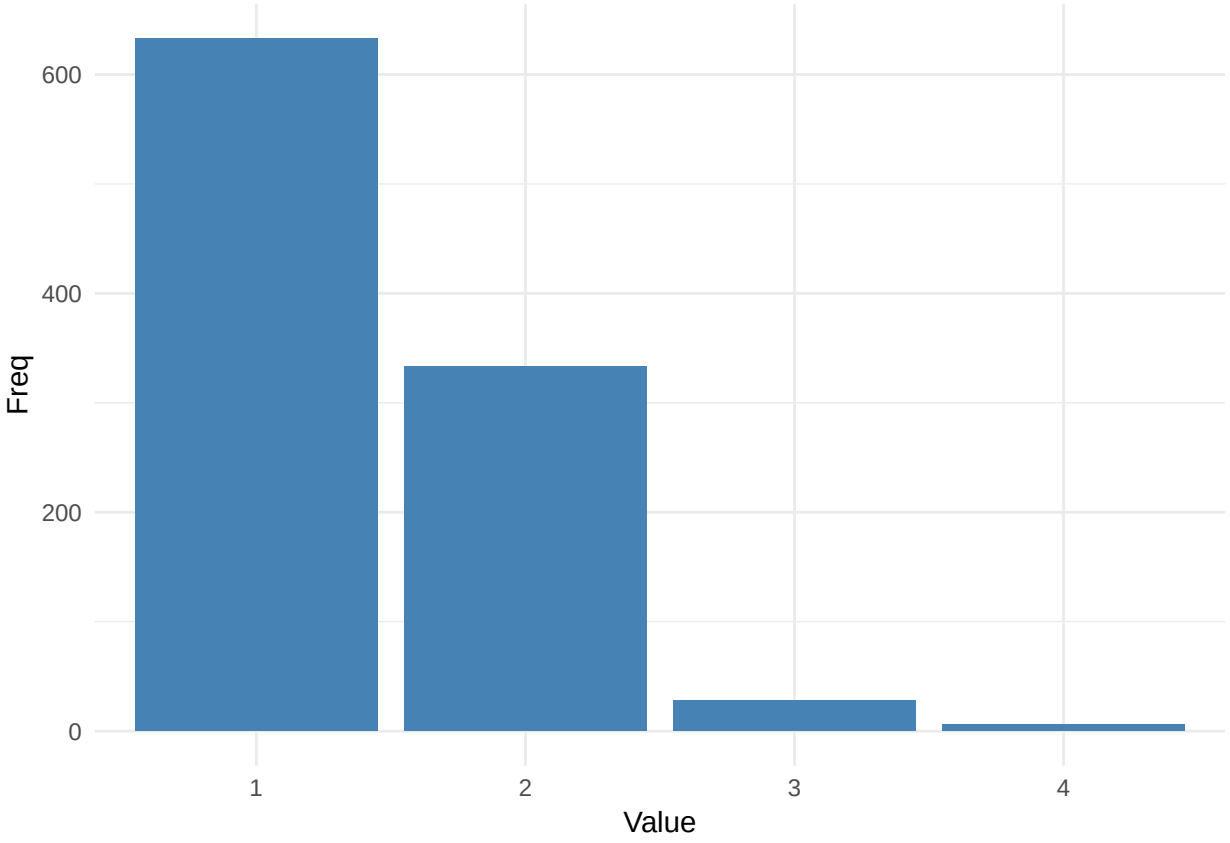
Categorical values can be also be assigned with 0 or 1.

**Variable 16: housing**

| | |
|---|---|
| for_free | 10.8% |
| own | 71.3% |
| rent | 17.9% |

Categorical values here can be also decomposed into dummy variables.

**Variable 17: existing_credits**

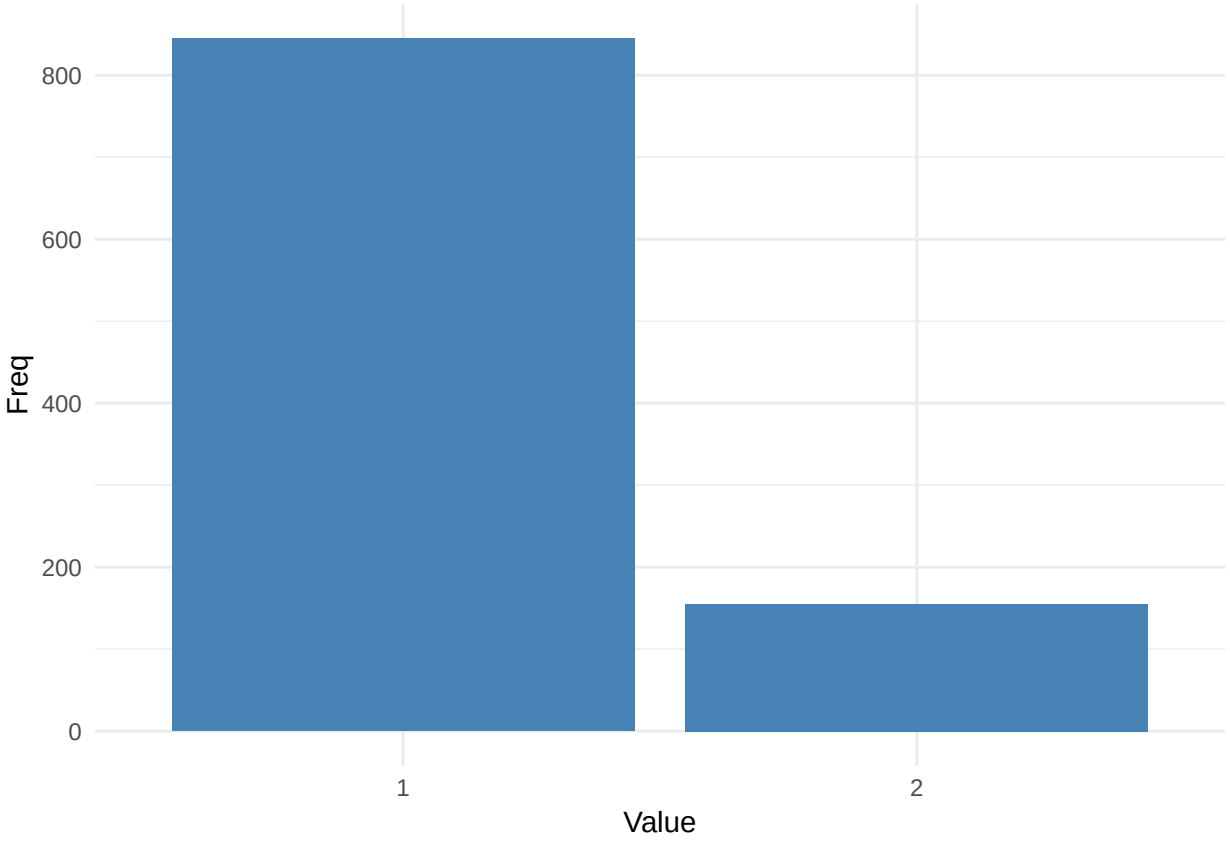



**Variable 18: job**

| | |
|---|---|
| management_self-employed_highly_qualified/officer | 14.8% |
| skilled_employee/official | 63% |
| unemployed/unskilled_nonresident | 2.2% |
| unskilled_resident | 20% |

Categorical values can be assigned with 0 or 1. There is no need to wrangle it because no clear and repeated pattern can be observed in the values.

**Variable 19: num_dependents**




**Variable 20: own_telephone**

| | |
|---|---|
| none | 59.6% |
| yes | 40.4% |

Values can be re-assigned with 0 or 1.

**Variable 21: foreign_worker**

| | |
|---|---|
| no | 3.7% |
| yes | 96.3% |

Values here can be translated into 0 or 1.

# 3. Data Wrangling

Basically, categorical variables will be separated into dummy variable (with one left unassigned to avoid multicollinearity). The target is to obtain two sets of data sets: one with pure dummy encoded variables and another one where bins are replaced by representive values. It has been decided this way to see if there is any effect when representative values are used in the modeling.

```r
# impute NAs
row.with.na <- which(is.na(raw.data["credit_amount"]))
if(length(row.with.na) != 0){
  raw.data$credit_amount[row.with.na] <- mean(raw.data$credit_amount[-row.with.na])
}

# wrangle multiple values
civil.status <- raw.data["personal_status"] %>%
                  separate(col = "personal_status", sep = "_",
                           into = c("personal.gender","civil.status")) %>%
                    dplyr::select(civil.status)

unique(civil.status)
```

```
##                   civil.status
## 1                       single
## 2  divorced/separated/married
## 9           divorced/separated
## 10             married/widowed
```

```r
raw.data <- cbind(raw.data, civil.status) %>%
              mutate("cs.single" = ifelse(civil.status == 'single', 1, 0),
                     "cs.divorced.separated" = ifelse(grepl('(.)*divorced/separated(.)*',
                                                            civil.status), 1, 0),
                    "cs.married" = ifelse(grepl('(.)*married(.)*', civil.status), 1, 0),
                    "cs.widowed" = ifelse(grepl('(.)*widowed', civil.status), 1, 0))
```

```r
# dummy coding
sparse.data <- raw.data
sparse.data <- dummy_cols(sparse.data, remove_first_dummy = TRUE,
              select_columns = c("checking_status", "credit_history", "purpose",
                                 "savings_status", "employment",
                                 "other_parties", "property_magnitude",
                                 "other_payment_plans", "housing", "job")) %>%
                mutate("own_telephone" = ifelse(own_telephone == 'none', 0, 1),
                       "foreign_worker" = ifelse(foreign_worker == 'no', 0, 1),
                       "gender" = ifelse(gender == 'male', 1, 0))
```

```r
sparse.cols <- setdiff(names(sparse.data), c("X", "checking_status", "credit_history",
                                             "purpose", "savings_status", "employment",
                                             "other_parties",
                                             "property_magnitude", "personal_status",
                                             "other_payment_plans", "housing", "job",
                                             "civil.status", "cs.widowed"))
```

```r
# convert numeric bins to representative numbers
converted.sparse.data <- raw.data
converted.sparse.data <- dummy_cols(raw.data, remove_first_dummy = TRUE,
                    select_columns = c("checking_status", "credit_history",
                                       "purpose", "other_parties",
                                       "property_magnitude", "employment",
                                       "savings_status", "other_payment_plans",
                                       "housing", "job")) %>%
                  mutate("own_telephone" = ifelse(own_telephone == 'none', 0, 1),
                         "foreign_worker" = ifelse(foreign_worker == 'no', 0, 1),
```

```
                            "gender" = ifelse(gender == 'male', 1, 0))
converted.sparse.data <- converted.sparse.data %>%
      mutate("savings_status" = ifelse(savings_status == '1000DM', 1200,
                                  ifelse(savings_status == '500_to_1000DM', 750,
                                    ifelse(savings_status == '100_to_500DM', 300,
                                      ifelse(savings_status == "100DM", 50, 0)))))

converted.sparse.data <- converted.sparse.data %>%
      mutate("employment" = ifelse(employment == ">7yrs", 9,
                                ifelse(employment == "4_to_7yrs", 6.5,
                                  ifelse(employment == "1_to_4yrs", 2.5,
                                    ifelse(employment == "<1yr", 0.5, 0)))))
```

```
# convert numeric bins to a number
converted.cols <- setdiff(names(converted.sparse.data),
                          c("X", "checking_status", "credit_history", "purpose",
                            "personal_status", "other_parties", "property_magnitude",
                            "personal_status", "other_payment_plans", "housing",
                            "job", "civil.status", "cs.widowed", "employment_1_to_4yrs",
                            "employment_4_to_7yrs","employment_unemployed",
                            "employment_<1yr", "savings_status_<100DM",
                            "savings_status_500_to_1000DM",
                            "savings_status_>1000DM", "savings_status_100_to_500DM"))
```

Below is the code that will generate the two data sets mentioned aboved.

```
# Set A: training and test data
n.observations <- nrow(raw.data)
set.seed(SEED_NUMBER)
rows <- sample(1:n.observations, 0.8*n.observations)
training.a <- sparse.data[rows, sparse.cols]
test.a <- sparse.data[-rows, sparse.cols]

# Set B: training and test data
training.b <- converted.sparse.data[rows, converted.cols]
test.b <- converted.sparse.data[-rows, converted.cols]
```

# 4. Predictive modeling

## 4.a Logistic Regression

The glm package of R and stepAIC are used to speed up the feature selection in logistic regression. The stepwise selection: forward, backward and both directions, are all considered. Furthermore, variables resulting from these algorithms are checked for significant. The level of signifance used in here is $\alpha = 0.05$.

```
# utility functions
compute.error.rate <- function(model, test.data){
  probabilities <- predict(model, newdata = test.data, type = "response")
```

```
  predicted <- ifelse(probabilities > 0.5, 1, 0)
  1 - mean(predicted == test.data$accepted)
}
```

**Set A: Data pure of dummy variables**

```
# fitting via bidirectional
logistic.both.model <- glm(accepted ~., data = training.a, family = binomial) %>%
                        stepAIC(direction = "both", trace = FALSE)
summary(logistic.both.model)

logistic.both.reduced.model <- glm(accepted ~ duration + installment_commitment +
    asnm + foreign_worker + cs.divorced.separated +
    checking_status_0_to_200DM + checking_status_None + `checking_status_>200DM` +
    credit_history_Existing_credits_paid_till_now +
    credit_history_No_credits_taken_or_all_paid + credit_history_All_credits_paid_duly +
    purpose_education + purpose_new_car +
    `savings_status_<100DM` + savings_status_100_to_500DM + employment_4_to_7yrs +
    other_parties_guarantor + other_payment_plans_bank + other_payment_plans_stores +
    housing_rent, family = binomial, data = training.a)
summary(logistic.both.reduced.model)


# fitting via forward direction
logistic.forward.model <- glm(accepted ~ ., data = training.a, family = binomial)  %>%
                        stepAIC(direction = "forward", trace = FALSE)
summary(logistic.forward.model)

logistic.forward.reduced.model <- glm(accepted ~ duration + installment_commitment +
    foreign_worker +  checking_status_0_to_200DM + checking_status_None +
    `checking_status_>200DM` + credit_history_Existing_credits_paid_till_now +
    credit_history_No_credits_taken_or_all_paid + credit_history_All_credits_paid_duly +
    purpose_education + purpose_new_car + `savings_status_<100DM` +
    savings_status_100_to_500DM + employment_4_to_7yrs +
    other_parties_guarantor + other_payment_plans_bank,
    data = training.a, family = binomial)
summary(logistic.forward.reduced.model)


# fitting via backward direction
logistic.backward.model <- glm(accepted ~ ., data = training.a, family = binomial) %>%
                        stepAIC(direction = "backward", trace = FALSE)
summary(logistic.backward.model)

logistic.backward.reduced.model <- glm(accepted ~ duration + installment_commitment +
    asnm + foreign_worker + cs.divorced.separated +
    checking_status_0_to_200DM + checking_status_None + `checking_status_>200DM` +
    credit_history_Existing_credits_paid_till_now +
    credit_history_No_credits_taken_or_all_paid + credit_history_All_credits_paid_duly +
    purpose_education + purpose_new_car +
    `savings_status_<100DM` + savings_status_100_to_500DM + employment_4_to_7yrs +
    other_parties_guarantor + other_payment_plans_bank + other_payment_plans_stores +
    housing_rent,
```

```
    data = training.a, family = binomial)
summary(logistic.backward.reduced.model)


# collecting results
log.models.a <- list(logistic.both.model, logistic.forward.model,
                     logistic.backward.model, logistic.both.reduced.model,
                     logistic.forward.reduced.model,
                    logistic.backward.reduced.model)
error.rates.a <- matrix(sapply(log.models.a, function(m){ compute.error.rate(m, test.a) }),
                    nrow = 3, ncol = 2)
```

**Set B: Mixture of dummy variables and representative values**

```
# fitting via bidirectional
logistic.both.model <- glm(accepted ~., data = training.b, family = binomial) %>%
                        stepAIC(direction = "both", trace = FALSE)
summary(logistic.both.model)

logistic.both.reduced.model <- glm(accepted ~ duration +
     installment_commitment + asnm + foreign_worker + cs.divorced.separated +
     checking_status_0_to_200DM +
     checking_status_None + `checking_status_>200DM` +
     credit_history_Existing_credits_paid_till_now +
     credit_history_No_credits_taken_or_all_paid +
     credit_history_All_credits_paid_duly + purpose_education +
     purpose_new_car + other_parties_guarantor +
     other_payment_plans_bank + other_payment_plans_stores + housing_rent,
    family = binomial, data = training.b)
summary(logistic.both.reduced.model)


# fitting via forward direction
logistic.forward.model <- glm(accepted ~ ., data = training.b, family = binomial)  %>%
                        stepAIC(direction = "forward", trace = FALSE)
summary(logistic.forward.model)
#compute.error.rate(logistic.forward.model, test.b)

logistic.forward.reduced.model <- glm(accepted ~ duration + installment_commitment +
    asnm +foreign_worker + checking_status_0_to_200DM + checking_status_None +
    `checking_status_>200DM` + credit_history_Existing_credits_paid_till_now +
    credit_history_No_credits_taken_or_all_paid +
    credit_history_All_credits_paid_duly + purpose_new_car +
    other_parties_guarantor +
    other_payment_plans_bank + other_payment_plans_stores,
    data = training.b,
    family = binomial)
summary(logistic.forward.reduced.model)


# fitting via backward direction
logistic.backward.model <- glm(accepted ~ ., data = training.b, family = binomial) %>%
                        stepAIC(direction = "backward", trace = FALSE)
```

Table 1: Error rates obtained through logistic regression

| | Set A | | Set B | |
|---|---|---|---|---|
| | Algorithm | Reduced | Algorithm | Reduced |
| **Bi-directional** | 0.270 | 0.280 | 0.265 | 0.285 |
| **Forward** | 0.285 | 0.255 | 0.260 | 0.275 |
| **Backward** | 0.270 | 0.280 | 0.265 | 0.290 |

```
summary(logistic.backward.model)

logistic.backward.reduced.model <- glm(accepted ~ duration + installment_commitment +
    asnm + foreign_worker + cs.divorced.separated + checking_status_0_to_200DM +
    checking_status_None + `checking_status_>200DM` +
    credit_history_Existing_credits_paid_till_now +
    credit_history_No_credits_taken_or_all_paid +
    credit_history_All_credits_paid_duly + purpose_education +
    purpose_new_car + other_parties_guarantor +
    other_payment_plans_bank + other_payment_plans_stores,
    data = training.b, family = binomial)
summary(logistic.backward.reduced.model)

# collecting results
log.models.b <- list(logistic.both.model, logistic.forward.model,
                     logistic.backward.model, logistic.both.reduced.model,
                     logistic.forward.reduced.model,
                    logistic.backward.reduced.model)
error.rates.b <- matrix(sapply(log.models.b, function(m){ compute.error.rate(m, test.b) }),
                    nrow = 3, ncol = 2)
```

**Best logistic regression model:** Using the Set A data, the forward selection model has been analyzed further to get rid of variables that have no sufficient proof of being significant. The test error rate in this case is 25.5%.

## 4.b Support Vector Machines

First, the feature variables need to be scaled in order to improve the performance of the SVM optimizer.

```
scale.features <- function(test.data){
  matrix.data <- as.matrix(test.data)
  target.variable <- matrix.data[,"accepted"]
  scaled.data <- scale(matrix.data, center = TRUE, scale = TRUE)
  scaled.data <- data.frame(scaled.data)
  scaled.data$accepted <- target.variable
  scaled.data$accepted <- as.factor(scaled.data$accepted)
  scaled.data
}

scaled.train.a <- scale.features(training.a)
scaled.train.b <- scale.features(training.b)
scaled.test.a <- scale.features(test.a)
scaled.test.b <- scale.features(test.b)
```

Then, two variants of the model will be considered: one using the linear kernel, and the other one with radial kernel. Both model types are trained under the two sets of data.

```
# fitting with linear kernel
set.seed(SEED_NUMBER)
svm.tune.linear.a <- tune(svm, accepted ~ ., data = scaled.train.a, kernel = "linear",
                          ranges = list(cost = c(0.01, 0.1, 1, 5, 8, 10, 12, 20, 30, 50)))
summary(svm.tune.linear.a)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     8
##
## - best performance: 0.2325
##
## - Detailed performance results:
##     cost   error dispersion
## 1   0.01 0.26000 0.05394184
## 2   0.10 0.24000 0.04322101
## 3   1.00 0.23375 0.04251225
## 4   5.00 0.23375 0.04168749
## 5   8.00 0.23250 0.04216370
## 6  10.00 0.23250 0.04216370
## 7  12.00 0.23250 0.04216370
## 8  20.00 0.23250 0.04216370
## 9  30.00 0.23375 0.04168749
## 10 50.00 0.23375 0.04168749
```

```
# fitting with linear kernel
set.seed(SEED_NUMBER)
svm.tune.linear.b <- tune(svm, accepted ~ ., data = scaled.train.b, kernel = "linear",
                          ranges = list(cost = c(0.01, 0.1, 1, 5, 8, 10, 50, 100)))
summary(svm.tune.linear.b)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.245
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.26375 0.05665747
```

```
## 2 1e-01 0.24625 0.05466120
## 3 1e+00 0.24500 0.05596378
## 4 5e+00 0.24625 0.05001736
## 5 8e+00 0.24500 0.04830459
## 6 1e+01 0.24500 0.04830459
## 7 5e+01 0.24625 0.04966904
## 8 1e+02 0.24750 0.04779877
```

```r
# fitting with radial kernel
set.seed(SEED_NUMBER)
svm.tune.radial.a <- tune(svm, accepted ~ ., data = scaled.train.a, kernel = "radial",
                          ranges = list(cost = c(100, 150, 180, 200, 220, 250, 300),
                                        gamma = c(01e-05, 0.0001, 0.001, 0.01, 0.1, 1)))
summary(svm.tune.radial.a)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   300 1e-04
##
## - best performance: 0.23375
##
## - Detailed performance results:
##    cost gamma   error dispersion
## 1   100 1e-05 0.30375 0.04450733
## 2   150 1e-05 0.30250 0.04851976
## 3   180 1e-05 0.29875 0.04226652
## 4   200 1e-05 0.29000 0.03322900
## 5   220 1e-05 0.28750 0.04487637
## 6   250 1e-05 0.28500 0.04199868
## 7   300 1e-05 0.27500 0.05103104
## 8   100 1e-04 0.24375 0.04218428
## 9   150 1e-04 0.23875 0.04387878
## 10  180 1e-04 0.24125 0.04291869
## 11  200 1e-04 0.24000 0.04281744
## 12  220 1e-04 0.23750 0.03773077
## 13  250 1e-04 0.23625 0.03701070
## 14  300 1e-04 0.23375 0.03537988
## 15  100 1e-03 0.23875 0.05846711
## 16  150 1e-03 0.24625 0.05304937
## 17  180 1e-03 0.24500 0.05502525
## 18  200 1e-03 0.24375 0.05870418
## 19  220 1e-03 0.24375 0.05597929
## 20  250 1e-03 0.24500 0.05143766
## 21  300 1e-03 0.24875 0.05935124
## 22  100 1e-02 0.28500 0.06003471
## 23  150 1e-02 0.28500 0.06089609
## 24  180 1e-02 0.28375 0.06209592
## 25  200 1e-02 0.28375 0.06209592
```

```
## 26  220 1e-02 0.28375 0.06209592
## 27  250 1e-02 0.28375 0.06209592
## 28  300 1e-02 0.28375 0.06209592
## 29  100 1e-01 0.28750 0.04208127
## 30  150 1e-01 0.28750 0.04208127
## 31  180 1e-01 0.28750 0.04208127
## 32  200 1e-01 0.28750 0.04208127
## 33  220 1e-01 0.28750 0.04208127
## 34  250 1e-01 0.28750 0.04208127
## 35  300 1e-01 0.28750 0.04208127
## 36  100 1e+00 0.30375 0.04450733
## 37  150 1e+00 0.30375 0.04450733
## 38  180 1e+00 0.30375 0.04450733
## 39  200 1e+00 0.30375 0.04450733
## 40  220 1e+00 0.30375 0.04450733
## 41  250 1e+00 0.30375 0.04450733
## 42  300 1e+00 0.30375 0.04450733
```

```r
# fitting with radial kernel
set.seed(SEED_NUMBER)
svm.tune.radial.b <- tune(svm, accepted ~ ., data = scaled.train.b, kernel = "radial",
                      ranges = list(cost = c(100, 150, 180, 200, 220, 250, 300),
                                    gamma = c(01e-05, 0.0001, 0.001, 0.01, 0.1, 1)))
summary(svm.tune.radial.b)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   200 1e-04
##
## - best performance: 0.2475
##
## - Detailed performance results:
##    cost gamma   error dispersion
## 1   100 1e-05 0.30375 0.04450733
## 2   150 1e-05 0.30500 0.04721405
## 3   180 1e-05 0.30500 0.04866267
## 4   200 1e-05 0.29625 0.04450733
## 5   220 1e-05 0.29375 0.04796194
## 6   250 1e-05 0.29250 0.04133199
## 7   300 1e-05 0.29000 0.04887626
## 8   100 1e-04 0.25250 0.05974483
## 9   150 1e-04 0.24875 0.05285265
## 10  180 1e-04 0.25000 0.05527708
## 11  200 1e-04 0.24750 0.05521423
## 12  220 1e-04 0.25000 0.05034602
## 13  250 1e-04 0.25125 0.05084358
## 14  300 1e-04 0.24875 0.05219155
## 15  100 1e-03 0.25625 0.05659616
```

```
## 16  150 1e-03 0.26000 0.06061032
## 17  180 1e-03 0.26125 0.05905800
## 18  200 1e-03 0.26375 0.05604128
## 19  220 1e-03 0.26250 0.05335937
## 20  250 1e-03 0.26375 0.05318012
## 21  300 1e-03 0.26375 0.04839436
## 22  100 1e-02 0.29500 0.04174992
## 23  150 1e-02 0.29750 0.03425801
## 24  180 1e-02 0.29750 0.03106892
## 25  200 1e-02 0.29750 0.03106892
## 26  220 1e-02 0.29625 0.03175973
## 27  250 1e-02 0.29375 0.03738408
## 28  300 1e-02 0.29500 0.04257347
## 29  100 1e-01 0.28000 0.04048319
## 30  150 1e-01 0.28000 0.04048319
## 31  180 1e-01 0.28000 0.04048319
## 32  200 1e-01 0.28000 0.04048319
## 33  220 1e-01 0.28000 0.04048319
## 34  250 1e-01 0.28000 0.04048319
## 35  300 1e-01 0.28000 0.04048319
## 36  100 1e+00 0.30375 0.04450733
## 37  150 1e+00 0.30375 0.04450733
## 38  180 1e+00 0.30375 0.04450733
## 39  200 1e+00 0.30375 0.04450733
## 40  220 1e+00 0.30375 0.04450733
## 41  250 1e+00 0.30375 0.04450733
## 42  300 1e+00 0.30375 0.04450733
```

The results are summarized in the table below.

```
# utility function
error.rate.svm <- function(svm.model, test.data){
  predicted <- predict(svm.model, newdata = test.data)
  1 - mean(predicted == test.data$accepted)
}

# get the best fit models
result <- matrix(c(svm.tune.linear.a$best.performance,
                   svm.tune.radial.a$best.performance,
                   error.rate.svm(svm.tune.linear.a$best.model, scaled.test.a),
                   error.rate.svm(svm.tune.radial.a$best.model, scaled.test.a),
                   svm.tune.linear.b$best.performance,
                   svm.tune.radial.b$best.performance,
                   error.rate.svm(svm.tune.linear.b$best.model, scaled.test.b),
                   error.rate.svm(svm.tune.radial.b$best.model, scaled.test.b)),
                   nrow = 2, ncol = 4)
rownames(result) <- c("Linear Kernel", "Radial Kernel")
colnames(result) <- c("Training", "Test", "Training", "Test")

kable(result, caption = "Error rates obtained through SVM") %>%
  kable_styling("striped") %>%
   column_spec(1, bold = TRUE) %>%
    add_header_above(c(" ", "Set A" = 2, "Set B" = 2))
```

Table 2: Error rates obtained through SVM

| | Set A | | Set B | |
|---|---|---|---|---|
| | Training | Test | Training | Test |
| **Linear Kernel** | 0.23250 | 0.27 | 0.2450 | 0.28 |
| **Radial Kernel** | 0.23375 | 0.26 | 0.2475 | 0.26 |

**Best SVM model:** The model that is based on radial kernel performed very well. It has parameters: cost = 200 or 300, and gamma = 0.0001, regardless of which data set it is trained to. It yielded a test error rate of 26%.

## 4.c Random Forest

For the final machine learning technique, unwrangled variables are preferred. Random Forest works by establishing a solid tree structure, so categorical values are more favorable this time.

```
# wrangled data set
var.names <- setdiff(names(raw.data), c("X", "personal_status",
                            "cs.single", "cs.divorced.separated",
                            "cs.married", "cs.widowed"))
rf.data <- as.data.frame(unclass(raw.data[var.names]))
training.rf <- rf.data[rows,]
test.rf <- rf.data[-rows,]

feature.size <- length(var.names) - 1
p.range <- floor(sqrt(feature.size)/2):feature.size
```

```
# wrangled data
error.rates <- sapply(p.range, function(p) {
  set.seed(SEED_NUMBER)
  rand.model <- randomForest(accepted ~ ., data = training.rf,
                             mtry = p, ntree = 1000,importance = TRUE)

  probabilities <- predict(rand.model, test.rf, type = "response")
  predicted <- ifelse(probabilities > 0.5, 1, 0)
  1 - mean(predicted == test.rf$accepted)
})
```

```
result <- data.frame("Features" = p.range, "Error rate" = error.rates)
best.rf <- randomForest(accepted ~ ., data = training.rf,
                            mtry = 12, ntree = 1000, importance = TRUE)

kable(result, caption = "Error rates obtained through Random Forest") %>%
  kable_styling("striped") %>%
   column_spec(1, bold = TRUE)
```

**Best Random Forest model:** The one that has p = 12 (i.e. 12 feature variables are considered at random in each split/branch made by the random forest) and 1000 ensemble trees (repetitions).

Table 3: Error rates obtained through Random Forest

| **Features** | Error.rate |
|---|---|
| **2** | 0.245 |
| **3** | 0.235 |
| **4** | 0.220 |
| **5** | 0.220 |
| **6** | 0.225 |
| **7** | 0.210 |
| **8** | 0.220 |
| **9** | 0.215 |
| **10** | 0.215 |
| **11** | 0.210 |
| **12** | 0.200 |
| **13** | 0.220 |
| **14** | 0.220 |
| **15** | 0.220 |
| **16** | 0.230 |
| **17** | 0.235 |
| **18** | 0.215 |
| **19** | 0.225 |
| **20** | 0.215 |
| **21** | 0.215 |
| **22** | 0.210 |

Table 4: Summary of best performing models

| **Technique** | **Set-up** | **Error rate** | **Data set used** |
|---|---|---|---|
| Logistic Regression | Reduced forward search result | 25.5% | full dummy encoded variables |
| Support Vector Machine | Radial kernel | 26% | either set but Set A is better |
| **Random Forest** | **12 variables per split** | **20%** | **no variable encoding required** |

# 5. Model Evaluation

We summarize the best performing models below.

```
summary.result <- matrix(c("Logistic Regression", "Reduced forward search result", "25.5%",
         "full dummy encoded variables",
         "Support Vector Machine",  "Radial kernel", "26%",
         "either set but Set A is better",
         "Random Forest", "12 variables per split", "20%",
         "no variable encoding required"),
          nrow = 3, ncol = 4, byrow = TRUE)
colnames(summary.result) <- c("Technique", "Set-up", "Error rate", "Data set used")

kable(summary.result, caption = "Summary of best performing models") %>%
  kable_styling("striped") %>%
   row_spec(c(0,3), bold = TRUE)
```

It can be seen that the random forest model, which was obtained by limiting to 12 the number of random

predictors per decision split, has the most superior performance. It is 80% accurate in predicting whether the applicant will be accepted or not in the loan. Hence, it will be my preferred model.

Unfortunately, it is hard to identify how the feature variables affect the prediction because Random Forest is a type of black-box modeling. We cannot interpret the effect of each variable. This came as a good trade off for having the highest chance of correctly identifying whether an applicant will be rejected or not in his loan application in future.