

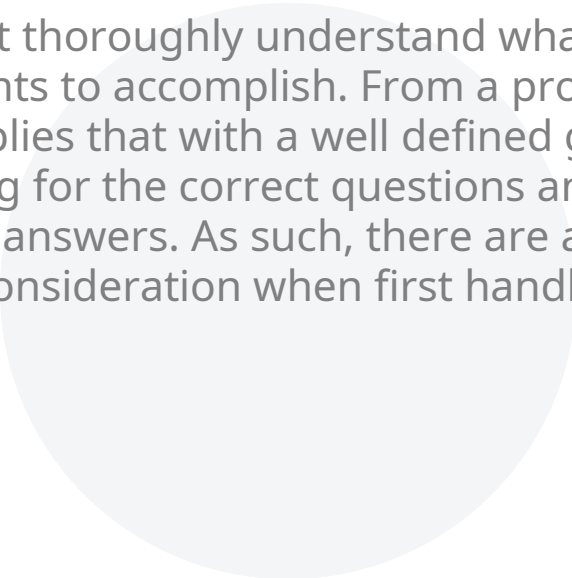


Data Mining Project

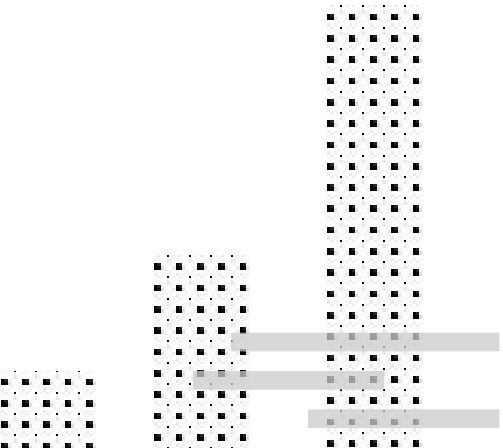
Miguel Amorim, up201907756
Rita Mendes, up201907877
Tiago Rodrigues, up201907021



Domain



It is essential to first thoroughly understand what the customer really wants to accomplish. From a productivity standpoint, this implies that with a well defined goal, it is easy to go searching for the correct questions and, as follows, the correct answers. As such, there are a few key aspects to take in consideration when first handling this project.

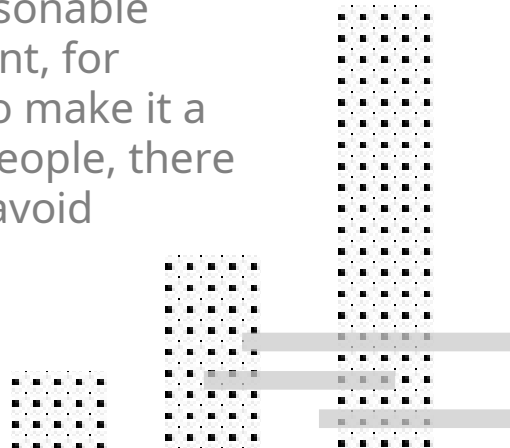




Customer Background

In this project, we are dealing with a banking company. The bank managers wish to improve the services of their bank, and make better decisions alongside it, improving customer satisfaction. This not only drives profits for the bank, but good clients can be rewarded more often, leading them to visit the bank even more.

The most pressing issue is that the bank cannot tell with reasonable certainty who is a good client and who is not. This is important, for example, when issuing loans, as they need to be paid back to make it a sound business decision. When issuing loans to the wrong people, there are a load of problems associated that a bank would rather avoid altogether in the first place.



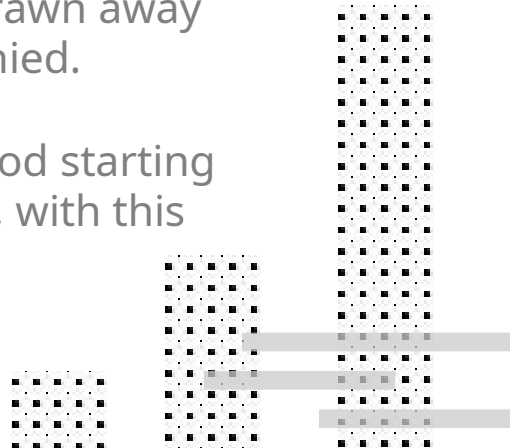


Business objectives

The first and most important goal is to reliably distinguish between a good and a bad client. With this in mind, a more concrete statement that the bank wishes to be fulfilled is the prediction of whether some loans will end up successfully or not.

Ideally, the bank would not lose a single customer with this new system, but it is very likely that customers identified as bad can be drawn away from visiting the service, as their loan requests would be denied.

While the bank did not present any success criteria, it is a good starting point to try and get to a 85% success rate on loan prediction, with this number increasing as the system is refined.



Initial situation

Requirements

The main requirement provided was the completion date, which was set to be on the 28th of November, 2022.

Assumptions

- All clients of the bank are present in the dataset
- All transactions of said clients are also present
- There are no impossible balances in the dataset (all are positive)
- Each client can have more than one account, and each account can have several people managing it
- No more than one loan can be granted per account
- Several credit cards can be issued to an account

The first 3 assumptions are made without looking at the data, and they exist to ensure data validity when exploring the set. All others are true after looking at the set, and they allow us to draw a better picture of the model behind the data.

Constraints

There are no foreseeable constraints in the development of this project.



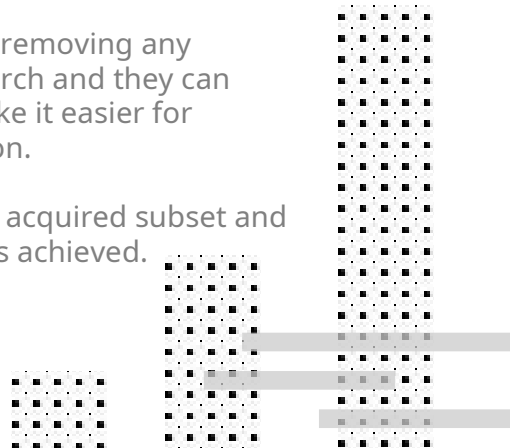
Project Plan

This project will mostly follow a CRISP-DM approach, and as such our plan revolves a lot around the same stages as the ones detailed in the guides. It is to note that, in the beginning of this assignment, the stages are crude and only overall descriptions. More concrete steps can only be obtained as the stages advance and it is clear what to do next.

The first and most important step is to understand the data at hand. This not only allows us to get a better understanding of the issues faced, but it can give rise to new insights and alter the project's course entirely. This stage will be tightly integrated with every stage, as finding new things about the data can change the entire perspective of the project.

With the data thoroughly analyzed, we can proceed to select a good subset and clean it, removing any attributes deemed as unnecessary in the process. These will contribute little to the research and they can skew the results in the wrong direction. New attributes are also built in this stage, to make it easier for further iterations to operate on the data. With the data prepared, it is possible to move on.

The final stage is a simpler one, since most of the work is automated. It entails using the acquired subset and running it through a series of models, continuously improving it until a satisfying value is achieved.





Success criteria

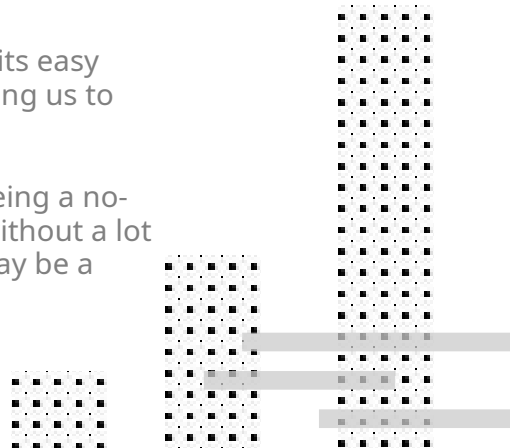
The success criteria are, essentially, equivalent to those of the business at hand, which are successfully determining if a given loan will end up successful with a higher than 85% accuracy. This system should try to create a profile from the available data for what makes a good client and determine if the bank should issue a loan or not.

Tools and techniques

As our tool of choice for Data Understanding and Preparation, the programming language R was chosen, as it provides an incredible out of the box API for data understanding, with its built-in charting features and Data Frame structures. When trying to figure out how the data is structured and seeing some patterns emerge, having a programming language like R, which allows us to do this seamlessly, is a great advantage.

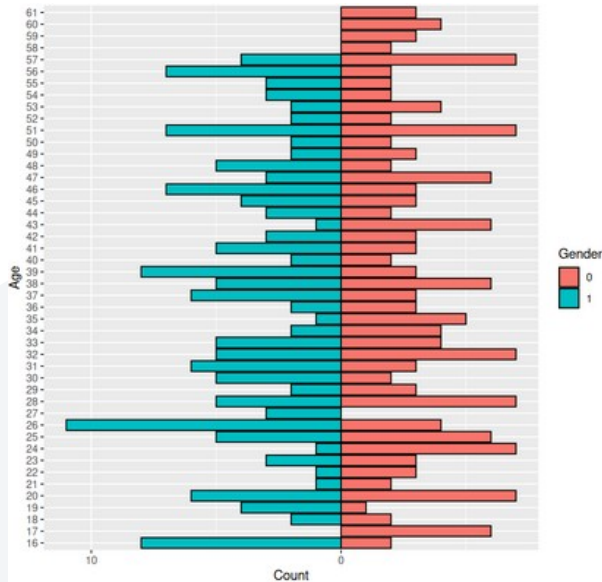
When it comes to Data Modeling, the most suited tool available was Python, primarily for its easy access to excellent libraries implementing the algorithms in an easy to use manner, allowing us to test a lot of different models quickly and efficiently.

As a backup tool, we may end up using RapidMiner, for its ease of use and simplicity by being a no-code tool, and for its ability to quickly generate some predictive analysis on the dataset, without a lot of work from its user. While we will try to stick to R only during this project, RapidMiner may be a valuable asset in the future.

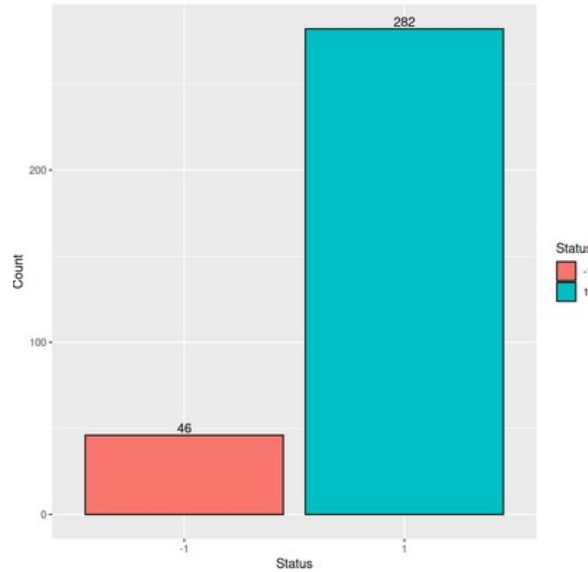


Data Understanding

To start, we plotted some graphs that allowed us to better know the data we will be working with.



Dataset Population Characterization

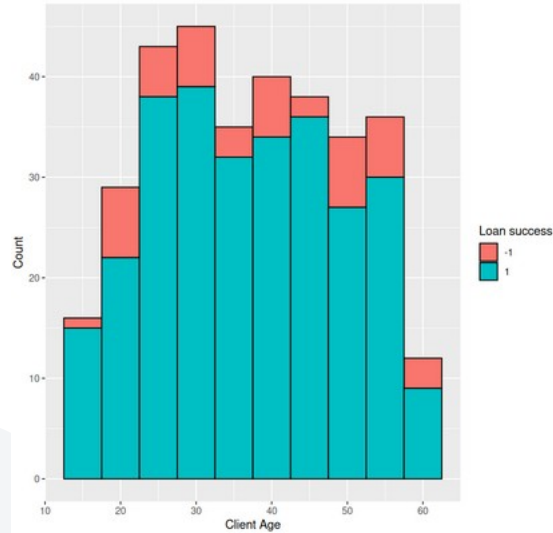


Successful and unsuccessful loans

Looking at the client distribution by age and gender, we find that there are about the same number of males as women, aged between 16 and 61 years old.

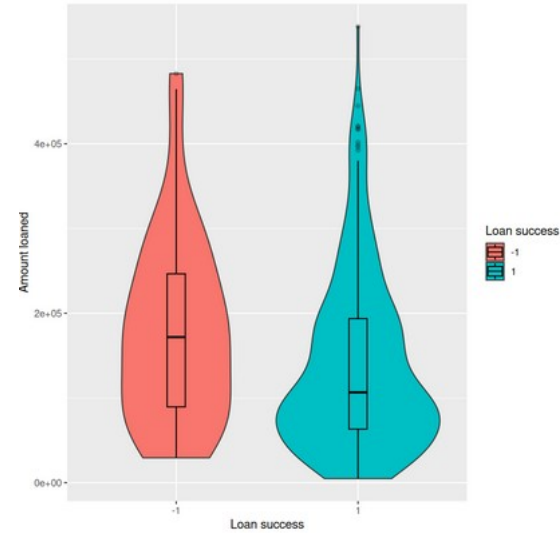
We then looked at the distribution of successful and failed loans, where we could see that the dataset is unbalanced, having way more successful than failed loans.

Data Understanding



Loan success and client age

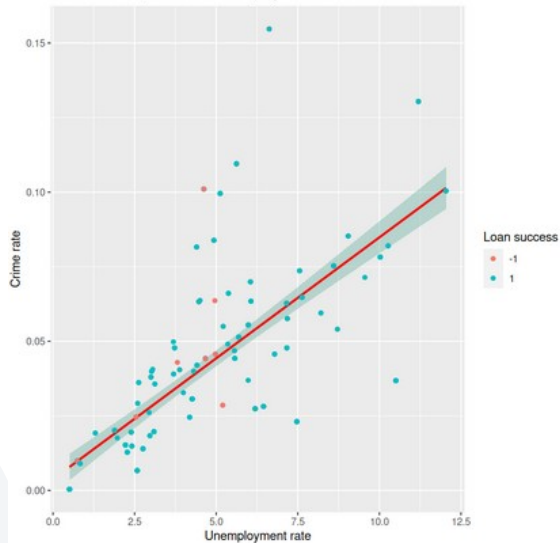
Plotting the client age distribution against the loan success, though, we could not find a meaningful correlation.



Loan success and distribution of amount loaned

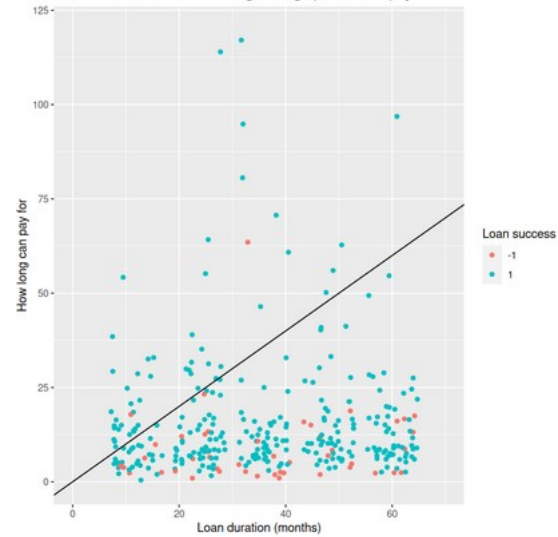
Unsuccessful loans seem to have a lower average amount of money loaned, but have more outliers way above the mean.

Data Understanding



Loan success, crime and unemployment rates

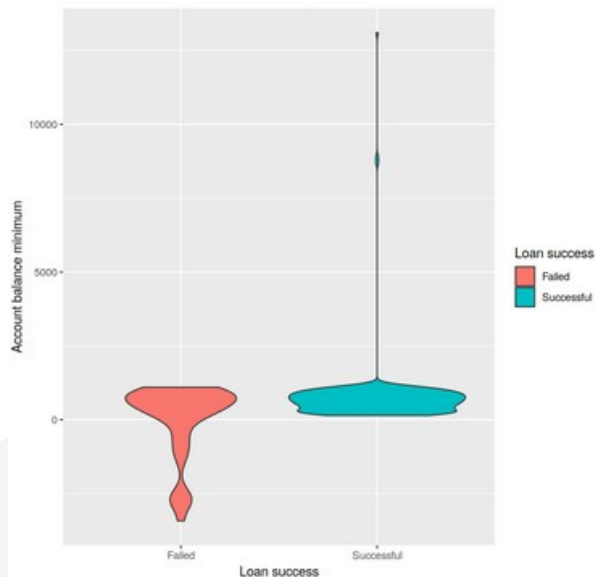
Trying to find any kind of relation between crime and unemployment rate and loan success, we could not take any conclusion, other than unemployment and crime rates are correlated.



Loan success and how long average person can pay it for

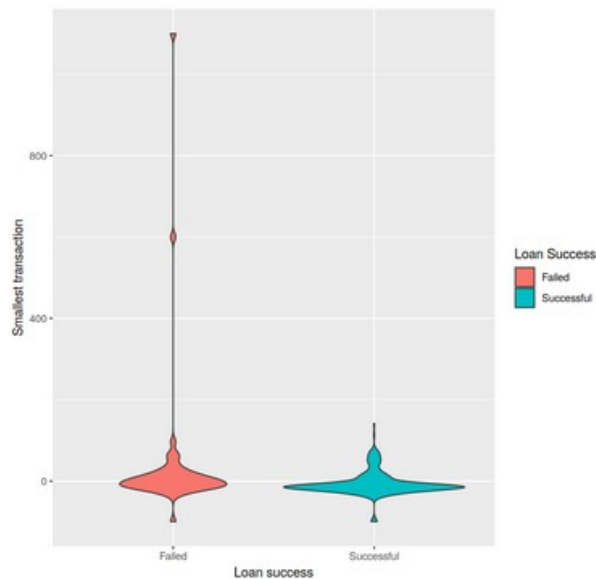
In this plot, the dots under the line represent the loans in which the average person from the client's district could not afford to pay.

Data Understanding



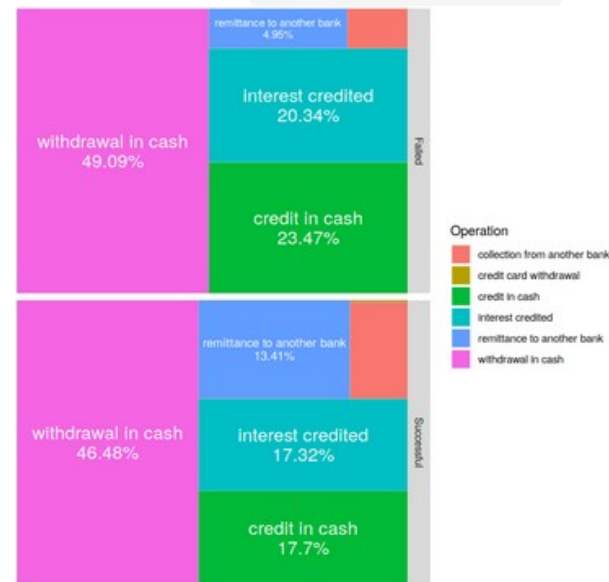
Loan status and the lowest balance the account has had

Looking at both of the distributions, we can see that accounts with failed loans seem to have more negative values.



Loan status and smallest transaction made

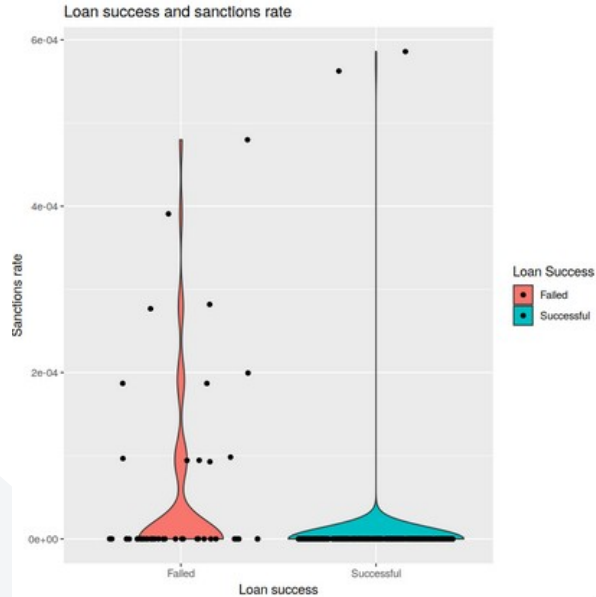
Here, we can conclude that some of the failed loans belong to accounts that have large values of smallest transaction done.



Loan status and operations the account has done

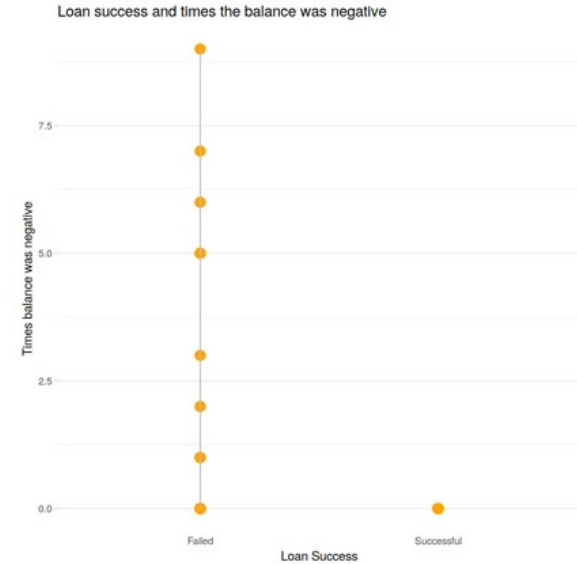
Here, the main difference we see is that accounts with unsuccessful loans tend to come from accounts with less "remittance to another bank" operation.

Data Understanding



Loan status and number of sanctions per day

Looking at the plot, we find that unsuccessful loans tend to come from accounts that have been sanctioned multiple times.



Loan status and times the balance was negative

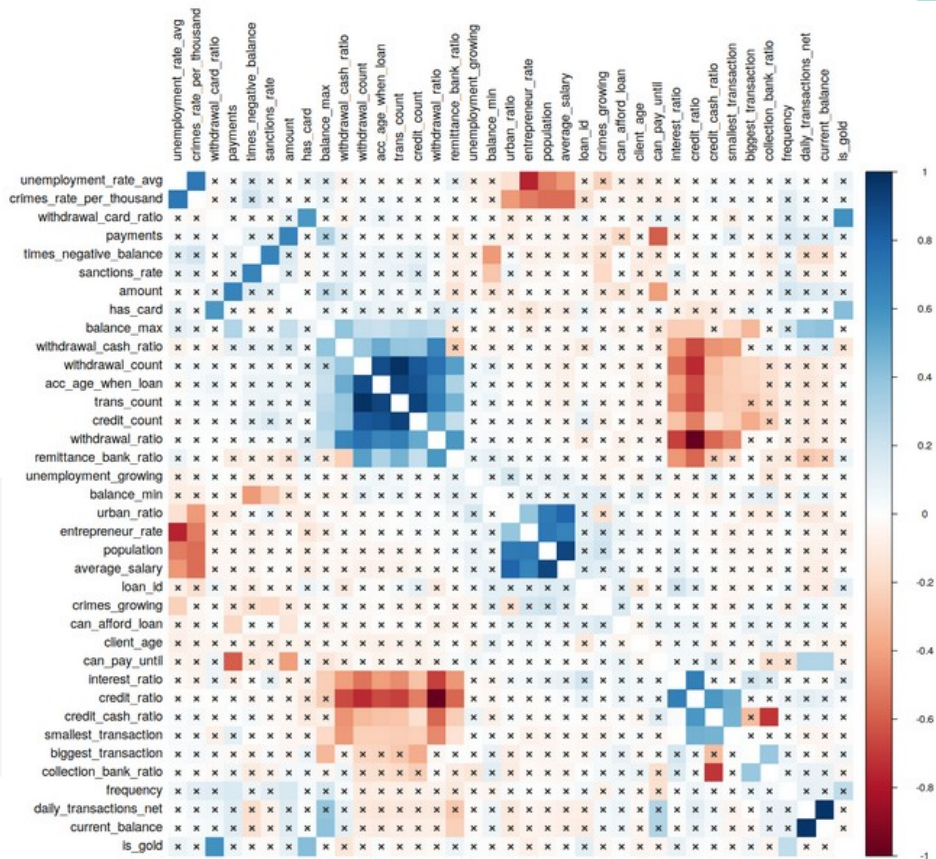
From this plot, we can clearly see that only accounts that have had unsuccessful loans seem to have had a negative balance, sometimes even multiple times.

Data Understanding

Correlation matrix

In this matrix, the insignificant correlations are marked with an "x", so we won't consider those for analysis. Let's look at what attributes seem to be more directly correlated with our target variable, the status, and how:

- **sanctions_rate**: High sanctions rate seem to be correlated with unsuccessful loans;
- **times_negative_balance**: Having an account that has had negative balance more times seems to be correlated with unsuccessful loans;
- **remittance_bank_ratio**: Having an account that has had more "remittance to another bank" operations seems to be correlated with successful loans;
- **balance_min**: Higher lowest balance on the account seems to be correlated with successful loans;
- **can_pay_until**: Having money to cover the payments for a longer duration seems to be associated with successful loans;
- **interest_ratio**: Having an account that has had more "interest credited" operations seems to be correlated with unsuccessful loans;
- **credit_ratio**: Having an account that has had more "credit card withdrawal" operations seems to be correlated with unsuccessful loans;
- **credit_cash_ratio**: Having an account that has had more "credit in cash" operations seems to be correlated with unsuccessful loans;
- **smallest_transaction**: Having had a higher amount for the smallest_transactions (no small transactions made) seems to be correlated to unsuccessful loans.




Data Understanding



Conclusions

From the analysis performed, we could start to create profiles with unusual behaviors found in accounts that had unsuccessful loans, such as having had negative balances, having been sanctioned before or having made only big transactions.

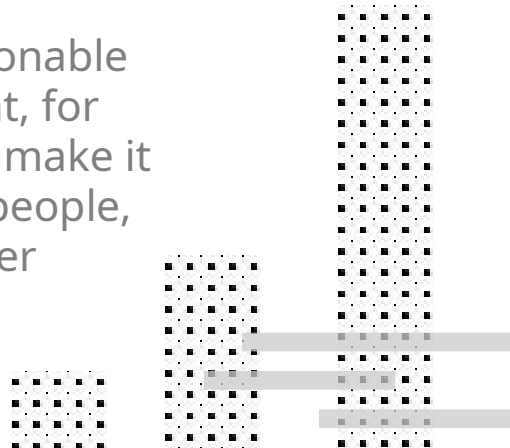




Problem definition

We are dealing with a banking company. The bank managers wish to improve the services of their bank and make better decisions alongside it, improving customer satisfaction. This not only drives profits for the bank but good clients can be rewarded more often, leading them to visit the bank even more.

The most pressing issue is that the bank cannot tell with reasonable certainty who is a good client and who is not. This is important, for example, when issuing loans, as they need to be paid back to make it a sound business decision. When issuing loans to the wrong people, there are a lot of problems associated that a bank would rather avoid altogether in the first place.



Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Account data

For the account dataset, we had to transform the frequency values into integer values that could be used in later steps, and used the date attribute to derive the account's age in days, as that could be useful to calculate other attributes.

```
prepare_account <- function() {  
  # Load dataset  
  account_data <- read.csv("../data/account.csv", sep = ";")  
  account_data <-  
    replace(account_data, (account_data == "" | account_data == " "), NA)  
  
  # Make date more readable  
  account_data <- transform(account_data, acc_creation_date = as.Date(  
    paste(  
      paste("19", date %/% 10000, sep = ""),  
      (date %/% 100) %% 100,  
      date %% 100,  
      sep = "-"  
    ),  
    format = "%Y-%m-%d"  
  ))  
  
  # Change string variables to integers, works better for later steps  
  account_data[account_data$frequency == "monthly issuance", ]$frequency <- 1  
  account_data[account_data$frequency == "weekly issuance", ]$frequency <- 2  
  account_data[  
    account_data$frequency == "issuance after transaction",  
  ]$frequency <- 3  
  
  # Calculate age of account in days  
  account_data$age_days <- trunc(as.numeric(  
    difftime(Sys.Date(), account_data$acc_creation_date, units = "days")  
  ))  
  
  return(account_data)  
}
```


Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Card data

For the card dataset, we kept everything as it was, but reformatted the issued date.

```
prepare_card <- function(train = TRUE) {  
  card_data <- read.csv(if_else(train,  
    "../data/card_dev.csv", "../data/card_comp.csv"),  
    sep = ";")  
  card_data <- replace(card_data, (card_data == "" | card_data == " "), NA)  
  # Make date more readable  
  card_data <- transform(card_data, issued = format(as.Date(  
    paste(  
      paste("19", issued %/% 10000, sep = ""),  
      (issued %/% 100) %% 100,  
      issued %% 100,  
      sep = "-"  
    ),  
    format = "%Y-%m-%d"  
  ), "%Y"))  
  return(card_data)  
}
```

Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Client data

For the client dataset, we extracted the gender and birthday from the birth number. We will be using the birthday later in order to calculate the client's age as of the last transaction in the dataset.

```
prepare_client <- function() {  
  # Load dataset  
  client_data <- read.csv("../data/client.csv", sep = ";")  
  client_data <-  
    replace(client_data, (client_data == "" | client_data == " "), NA)  
  
  client_data <- transform(client_data,  
    gender = ifelse(((birth_number %/% 100) %% 100) <= 12, 0, 1)  
  )  
  
  client_data <- transform(client_data, birthday = as.Date(  
    paste(  
      paste("19", birth_number %/% 10000, sep = ""),  
      ifelse(((birth_number %/% 100) %% 100) <= 12,  
        (birth_number %/% 100) %% 100,  
        ((birth_number %/% 100) %% 100) - 50  
      ),  
      birth_number %/% 100,  
      sep = "-"  
    ),  
    format = "%Y-%m-%d"  
  ))  
  
  client_data <- subset(client_data, select = -c(birth_number, district_id))  
  
  return(client_data)  
}
```

Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Disposition data

The disposition data was already clean, so we kept it as it was.

```
prepare_disp <- function() {  
  # Load dataset  
  disp_data <- read.csv("../data/disp.csv", sep = ";")  
  disp_data <-  
    replace(disp_data, (disp_data == "" | disp_data == " "), NA)  
}
```

Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

District data

For the district data, we started by renaming some columns for consistency. Then, we found that the columns related to unemployment and crime had a few null values, which we filled with the year's average. Then, we merged the data from 1995 and 1996, calculating the average between them, which we stored in a new attribute. We also created an attribute that is true if the values grew from 1995 to 1996. Finally, we removed the columns that we did not find relevant.

```
prepare_district <- function() {  
  # Load dataset  
  district_data <- read_csv("../data/district.csv",  
    sep = ";", na_strings = c("NaN", "?"))  
  district_data <-  
    replace(district_data, (district_data == "" | district_data == " "), NA)  
  
  # Rename code to district id to ease joins  
  # Rename columns for consistency  
  colnames(district_data)[colnames(district_data) == "code"] <- "district_id"  
  colnames(district_data)[  
    colnames(district_data) == "average_salary"  
  ] <- "average_salary"  
  colnames(district_data)[  
    colnames(district_data) == "no..of.entrepreneurs.per.1000.inhabitants"  
  ] <- "entrepreneur_rate"  
  colnames(district_data)[  
    colnames(district_data) == "no..of.inhabitants"  
  ] <- "population"  
  colnames(district_data)[  
    colnames(district_data) == "ratio.of.urban.inhabitants"  
  ] <- "urban_ratio"  
  
  # Fix values that were "?" to be the column average  
  district_data$unemployment.rate..95[  
    is.na(district_data$unemployment.rate..95)  
  ] <- mean(as.numeric(district_data$unemployment.rate..95), na.rm = TRUE)  
  district_data$unemployment.rate..96[  
    is.na(district_data$unemployment.rate..96)  
  ] <- mean(as.numeric(district_data$unemployment.rate..96), na.rm = TRUE)  
  
  district_data$no..of.committed.crimes..95[  
    is.na(district_data$no..of.committed.crimes..95)  
  ] <- mean(as.numeric(district_data$no..of.committed.crimes..95), na.rm = TRUE)  
  district_data$no..of.committed.crimes..96[  
    is.na(district_data$no..of.committed.crimes..96)  
  ] <- mean(as.numeric(district_data$no..of.committed.crimes..96), na.rm = TRUE),  
}
```

```
# Calculate average between 95 and 96  
district_data <- transform(district_data, unemployment_rate_avg =  
  as.numeric(district_data$unemployment.rate..95)  
  + as.numeric(district_data$unemployment.rate..96) / 2  
)  
district_data <- transform(district_data, crimes_rate_per_thousand =  
  (as.numeric(district_data$unemployment.rate..95)  
  + as.numeric(district_data$unemployment.rate..96) / 2)  
  / as.numeric(district_data$population)  
  * 1000  
)  
  
# Calculate whether or not the unemployment/crimes has been growing  
district_data <- transform(district_data, unemployment_growing =  
  ifelse(  
    as.numeric(district_data$unemployment.rate..96) >  
    as.numeric(district_data$unemployment.rate..95),  
    1,  
    0  
  )  
)  
district_data <- transform(district_data, crimes_growing =  
  ifelse(  
    as.numeric(district_data$no..of.committed.crimes..96) >  
    as.numeric(district_data$no..of.committed.crimes..95),  
    1,  
    0  
  )  
)  
  
district_data <- subset(district_data, select = -c(unemployment.rate..95,  
  unemployment.rate..96, no..of.committed.crimes..95, no..of.committed.crimes..96,  
  no..of.municipalities.with.inhabitants..499,  
  no..of.municipalities.with.inhabitants.500.1999,  
  no..of.municipalities.with.inhabitants.2000.9999,  
  no..of.municipalities.with.inhabitants..10000,  
  region,  
  no..of.cities  
)  
)  
return(district_data)
```

Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Loan data

For the loan data, we reformatted the date and dropped the duration column, since it was given by payments divided by amount, and was therefore redundant.

To prepare for later steps, we also had to change the status values to 0 if the loan was successful or 1 if it failed.

```
prepare_loan <- function(train = TRUE) {  
  loan_data <- read.csv(ifelse(train,  
    "../data/loan_dev.csv", "../data/loan_comp.csv"),  
    sep = ";")  
  )  
  loan_data <- replace(loan_data, (loan_data == "" | loan_data == " "), NA)  
  loan_data <- transform(loan_data, date = as.Date(  
    paste(  
      paste("19", date %/% 10000, sep = ""),  
      (date %/% 100) %% 100,  
      date %% 100,  
      sep = "-"  
    ),  
    format = "%Y-%m-%d"  
  ))  
  if (train) {  
    loan_data[loan_data$status == 1,]$status <- 0  
    loan_data[loan_data$status == -1,]$status <- 1  
  }  
  # payments * duration = amount, so don't need to keep them all  
  loan_data <- subset(loan_data, select = -c(duration))  
  return(loan_data)  
}
```

Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Transaction data

In the transaction data, we formatted the date and changed the amount column to reflect if it was referring to money entering or leaving the account, which allowed us to drop the type attribute. Then we figured that whenever the operation column was null, the k_symbol had the same value of interest credited, so we used that to fill the missing values. We renamed the k_symbol column to category, and filled the null values with the default value of “other”.

```
prepare_trans <- function(train = TRUE) {  
  trans_data <- read.csv(ifelse(train,  
    "../data/trans_dev.csv", "../data/trans_comp.csv"),  
    sep = ";")  
  trans_data <- replace(trans_data, (trans_data == "" | trans_data == " "), NA)  
  # Rename k_symbol column  
  colnames(trans_data)[colnames(trans_data) == "k_symbol"] <- "category"  
  if (!train) {  
    trans_data$account <- replace(  
      trans_data$account,  
      (trans_data$account == 0), NA  
    )  
  }  
  # Make date more readable  
  trans_data <- transform(trans_data, date = as.Date(  
    paste(  
      paste("19", date %/% 10000, sep = ""),  
      (date %/% 100) %%, 100,  
      date %%, 100,  
      sep = "-"  
    ),  
    format = "%Y-%m-%d"  
  ))  
  # Make amount reflect if money entered or left the account  
  trans_data <- transform(trans_data,  
    amount = ifelse(type == "credit", amount, -1 * amount)  
  )  
}
```

```
# Every entry where the operation is null, the k symbol is defined as  
# "interest credited", so we can fill operation column with new type  
# - interest credited  
trans_data <- transform(trans_data,  
  operation = ifelse(  
    is.na(trans_data$operation) | trans_data$operation == "" |  
    trans_data$operation == " ",  
    "interest credited",  
    operation  
  )  
)  
trans_data$category[is.na(trans_data$category)] <- "other"  
# Drop type column: withdrawal in cash already in operation, withdrawal  
# vs credit already in amount sign  
trans_data <- subset(trans_data, select = -c(type))  
return(trans_data)  
}
```

Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Transactions aggregation

In order to use transactions data to preview loan success, we need to aggregate it by the account.

With that, we were able to derive new attributes, such as number of transactions of the account and statistics related to the operations made, balance history and amount transacted. We then dropped some attributes that were now redundant.

```
aggregate_trans_data <- function(trans_data) {  
  # Aggregate transactions data  
  aggregated_trans <- trans_data %>%  
    # Group by account  
    group_by(account_id) %>%  
    arrange(date, .by_group = TRUE) %>%  
    # Add number of transactions per account  
    mutate(trans_count = n()) %>%  
    # Count credits/withdrawals  
    mutate(credit_count = sum(amount >= 0)) %>%  
    mutate(credit_ratio = mean(amount >= 0)) %>%  
    mutate(withdrawal_count = sum(amount < 0)) %>%  
    mutate(withdrawal_ratio = mean(amount < 0)) %>%  
    # Amount stats  
    mutate(smallest_transaction = amount[which.min(abs(amount))][1]) %>%  
    mutate(biggest_transaction = amount[which.max(abs(amount))][1]) %>%  
    mutate(transactions_net = sum(amount)) %>%  
    # Balance stats  
    mutate(balance_min = min(balance)) %>%  
    mutate(balance_max = max(balance)) %>%  
    mutate(current_balance = last(balance)) %>%  
    mutate(times_negative_balance = sum(balance < 0)) %>%  
    # Operation ratios  
    mutate(credit_cash_ratio =  
      mean(as.character(operation) == "credit in cash")) %>%  
    mutate(collection_bank_ratio =  
      mean(as.character(operation) == "collection from another bank")) %>%  
    mutate(interest_ratio =  
      mean(as.character(operation) == "interest credited")) %>%  
    mutate(withdrawal_cash_ratio =  
      mean(as.character(operation) == "withdrawal in cash")) %>%  
    mutate(remittance_bank_ratio =  
      mean(as.character(operation) == "remittance to another bank")) %>%  
    mutate(withdrawal_card_ratio =  
      mean(as.character(operation) == "credit card withdrawal")) %>%  
    mutate(sanctions =  
      sum(as.character(category) == "sanction interest if negative balance")) %>%  
    rename(trans_date = date) %>%  
  
    distinct()  
  
  trans_agg <- subset(aggregated_trans, select =  
    ~c(trans_id, operation, amount, balance, category)  
  )  
  
  return(trans_agg)  
}
```


Data preparation

To prepare the data for the next steps, we started by applying some simple transformations to each dataset and then we merged them, deriving new attributes and deleting others.

Merging the datasets

Finally, we joined every table into a single one, so that we could use it for the later steps.

We derived more attributes that related data with the account's age, which avoided bias when comparing recent and old accounts. We also created attributes relating the average district salary to the loan information, such as `can_afford_loan` and `can_pay_until`, which reflect how well the loan would go for the average person in the same district of the client.

```
join_tables <- function(account_data, card_data, client_data,
                        disp_data, district_data, loan_data,
                        trans_data) {

  last_transaction <- max(trans_data$trans_date)
  trans_data <- trans_data %>%
    select(-trans_date)

  # Join tables and create more derived attributes
  data <- loan_data %>%
    rename(loan_date = date) %>%
    left_join(account_data, by = "account_id") %>%
    left_join(trans_data, by = "account_id") %>%
    mutate(transactions_net = transactions_net / age_days) %>%
    mutate(sanctions_rate = sanctions / age_days) %>%
    rename(daily_transactions_net = transactions_net) %>%
    left_join(disp_data, by = "account_id") %>%
    filter(type == "OWNER") %>%
    select(-age_days, -type, -sanctions) %>%
    left_join(card_data, "disp_id") %>%
    mutate(has_card = ifelse(!is.na(card_id), 1, 0)) %>%
    mutate(is_gold = ifelse(!is.na(type) & type == "gold", 1, 0)) %>%
    select(-card_id, -type, -issued) %>%
    left_join(client_data, by = "client_id") %>%
    mutate(client_age = trunc(as.numeric(
      difftime(as.Date(last_transaction), birthday, units = "weeks")
    ) / 52.25)) %>%
    left_join(district_data, by = "district_id") %>%
    select(-c(name)) %>%
    mutate(can_afford_loan = ifelse(average_salary > payments, 1, 0)) %>%
    mutate(can_pay_until = current_balance / payments) %>%
    mutate(acc_age_when_loan = trunc(as.numeric(
      difftime(loan_date, acc_creation_date, units = "days")
    ))) %>%
    select(-c(acc_creation_date, account_id, district_id, date,
      disp_id, client_id, birthday))

  distinct()

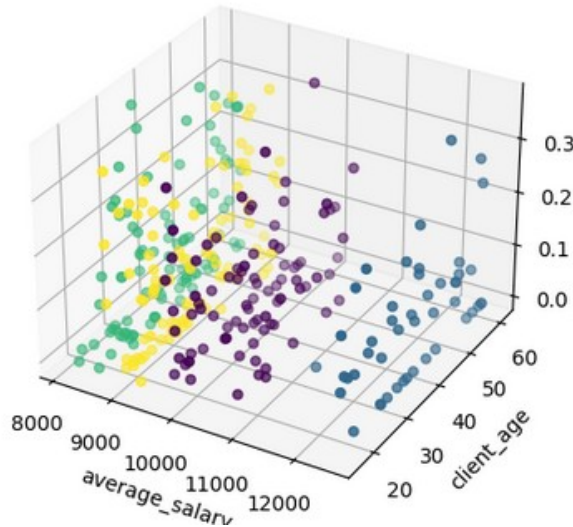
  return(data)
}
```


Data definition

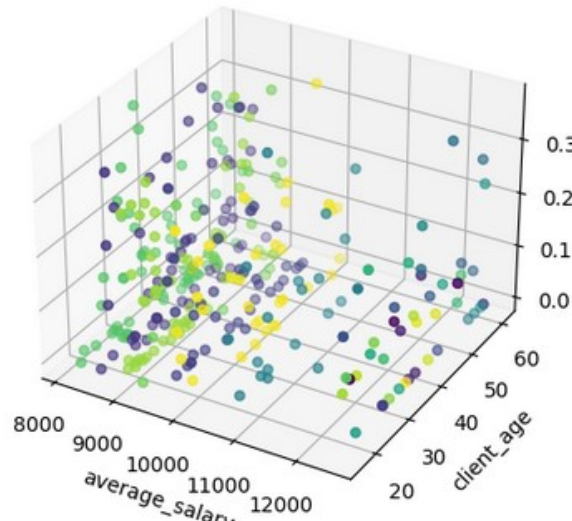
In order to better understand the nature of the data, we can try to cluster it into different groups, and see if any pattern emerges. To do this, we can apply several different algorithms, that may give us valuable insights over the data and the relations between the different entries.

We decided to try and use 3D clusters in order to try and relate more meaningful variables and, in turn, generate some more insightful plots.

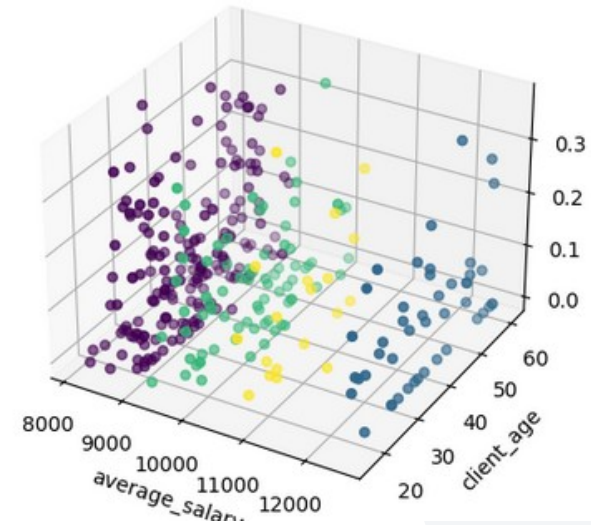
KMeans - Results



AffinityPropagation - Results



GaussianMixture - Results



Data definition

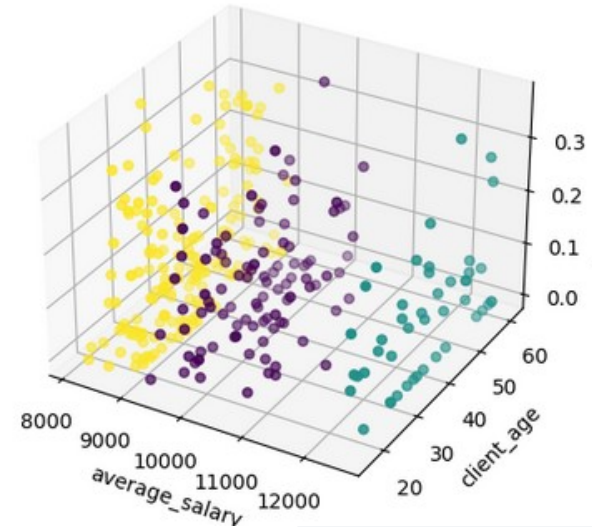
With these results we can see some patterns start emerging. From the **Affinity Propagation Clustering**, we cannot conclude much, and it is evident that it was not a good fit for this data distribution or problem, as it had its clusters scattered across the place.

Meanwhile, the results from **K-Means** and **Gaussian Mixture** seem to be nearly identical, and show some stratification regarding the age and salary, into 4 main categories (being especially salary biased), but make no real meaning of the remittance ratio, although they do indicate that some of the more wealthy individuals tend to have a lower remittance rate.

Finally, the **Bayesian Gaussian Mixture**, even though it was set to divide into 4 clusters, decided (apparently, correctly) that 3 components were necessary to divide the data truthfully. And, of all the distributions, it appears to be the one that gives the best insight, as it seems to show that the cluster with the lower salary has more entries with high remittance ratio, the middle one has a bit less, and the higher salary entries have the lowest remittance ratio of all.

Overall, almost all of the clusters can give some insight onto the data, but taking advantage of the power of the **Bayesian Gaussian Mixture** proved to lead to more insight than the other methods, making it the most suitable method for this particular problem instance.

BayesianGaussianMixture - Results



Experimental Setup

We start by loading the training data and fit several models to it. The criteria for evaluation was the Area Under the Curve (AUC)

- Load the actual data that will be used in the models
- Start trying to find the best model to predict the results. To find the best hyperparameters for each model, we will use **Grid Searching**, as randomized searches would not yield any better results, and evaluate the results using the **AUC metric**. As for splitting, we use a time-aware split, sorting the loans by date beforehand.
- We can check the results on the testing data and try to find out which one performs the best out of all of them. We will once again use the **Area Under the Curve (AUC)** as our metric, and we will show a Confusion Matrix to detail exactly what happened with the data, and decide if it is a good fit or not.
- Look to apply it to the competition data as well, and (ideally) obtain great results with it as well

```
SPLITTER = TimeSeriesSplit()

def train_model_grid(model, params, cv=SPLITTER):
    grid_search = GridSearchCV(estimator=model,
                               param_grid=params,
                               n_jobs=-1,
                               cv=cv,
                               scoring='roc_auc',
                               verbose=1,
                               refit=True)

    grid_search.fit(X, y)
    print(f"Best params for {model.__class__.__name__}: {grid_search.best_params_}")
    print(f"Best Score: {grid_search.best_score_}")

    return grid_search.best_estimator_
```

```
def evaluate(model):
    y_pred = model.predict_proba(X_test)

    # Area Under the Curve, the higher the better
    auc = metrics.roc_auc_score(y_test, y_pred[:,1])
    print(f"AUC Score on Testing Data: {auc}")

    y_pred_normalized = np.argmax(model.predict_proba(X_test), axis=1)
    cm = metrics.confusion_matrix(y_test, y_pred_normalized)
    ax = plt.subplot(1)
    sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap="Blues")

    # labels, title and ticks
    ax.set_xlabel("Predicted labels")
    ax.set_ylabel("True labels")
    ax.set_title("Confusion Matrix")
    ax.xaxis.set_ticklabels(["Accepted", "Not Accepted"])
    ax.yaxis.set_ticklabels(["Accepted", "Not Accepted"])

    # We want the results for the other class, so we use that one in the end
    return y_pred[:,1]
```

```
def export_results(model, file):
    test = test_data[features]

    confidences = model.predict_proba(test)
    confidences = confidences[:,1]

    confidences = [0 if x < 0.000001 else x for x in confidences]
    confidences = ["{:f}".format(x) for x in confidences]

    submission_data = pd.DataFrame()

    submission_data["Id"] = test_data["loan_id"]
    submission_data["Predicted"] = confidences

    submission_data.to_csv(f"../results/{file}.csv", sep=",", index=False)

    print(f"Done exporting to: {file}.csv")
```

```
def apply(model, params):
    best_model = train_model_grid(model, params)
    predicted_results = evaluate(best_model)
    export_results(best_model, best_model.__class__.__name__)

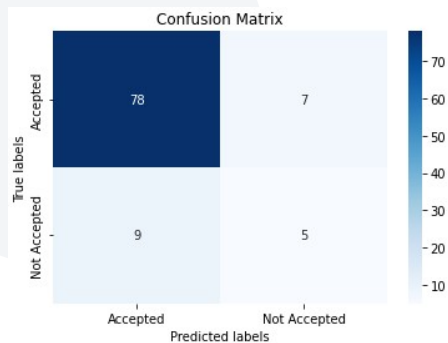
    return predicted_results
```

Results

Decision Tree

```
dt_results = apply(DecisionTreeClassifier(), {  
    'criterion': ['gini', 'entropy'],  
    'max_depth': range(1, 20),  
    'min_samples_split': range(2,10),  
    'min_samples_leaf': range(1,6)  
})
```

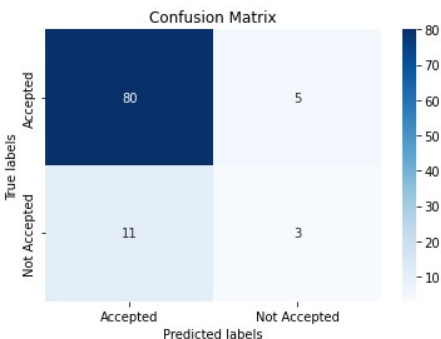
- Best params for DecisionTreeClassifier: {'criterion': 'gini', 'max_depth': 19, 'min_samples_leaf': 5, 'min_samples_split': 7}
- Best Score: 0.8251
- AUC Score on Testing Data: 0.6517



Random Forest

```
rf_results = apply(RandomForestClassifier(random_state=0), {  
    'n_estimators': [100, 150],  
    'max_features': ['sqrt', 'log2'],  
    'max_depth': [8,9,10,11,12],  
    'criterion': ['gini', 'entropy'],  
    'bootstrap': [True],  
    'min_samples_leaf': [1,2,3],  
    'min_samples_split': [2,3],  
})
```

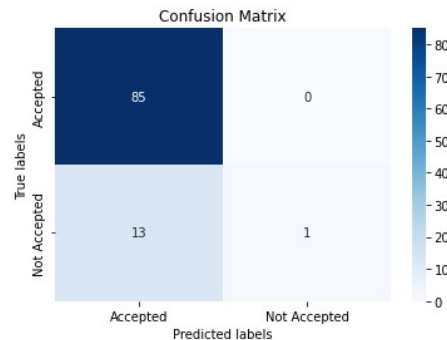
- Best params for RandomForestClassifier: {'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 150}
- Best Score: 0.8726
- AUC Score on Testing Data: 0.8529



Support Vector Machine

```
svm_results = apply(SVC(probability=True),  
    {'C': [1, 10, 50, 100],  
    'kernel': ['poly', 'rbf'],  
    'degree': [1, 2]  
})
```

- Best params for SVC: {'C': 10, 'degree': 1, 'kernel': 'poly'}
- Best Score: 0.7356
- AUC Score on Testing Data: 0.7681

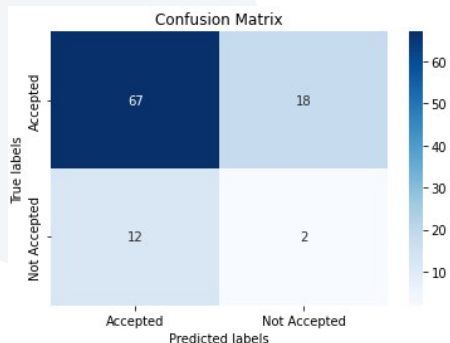


Results

MLP

```
mlb_results = apply(MLPClassifier(),{  
    'alpha': [0.1, 0.25, 0.45, 0.5, 0.55, 0.75, 0.85],  
    'hidden_layer_sizes': [10, 20, 30, 60, 100]  
})
```

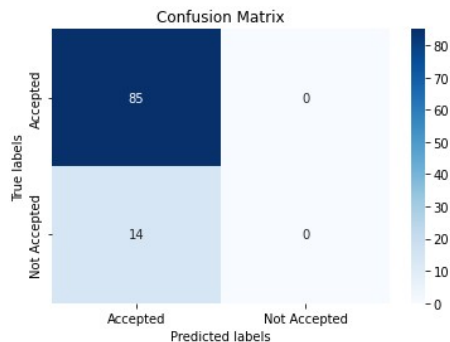
- Best params for MLPClassifier: {'alpha': 0.5, 'hidden_layer_sizes': 30}
- Best Score: 0.6600
- AUC Score on Testing Data: 0.4303



K-Nearest Neighbors

```
knn_results = apply(KNeighborsClassifier(),{  
    'n_neighbors': list(range(1,31)),  
    'weights': ['uniform', 'distance'],  
})
```

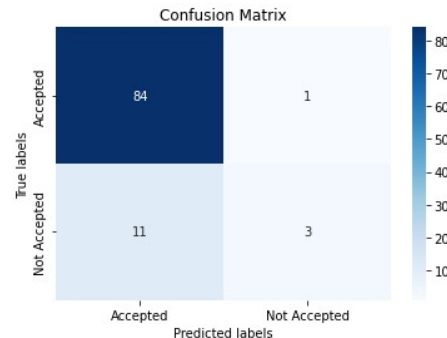
- Best params for Random ForestClassifier: KNeighborsClassifier: {'n_neighbors': 28, 'weights': 'distance'}
- Best Score: 0.6756
- AUC Score on Testing Data: 0.5786



Gradient Boosting

```
logistic_results = apply(GradientBoostingClassifier(), {  
    'max_features': range(7,20,2),  
    'min_samples_split':range(1000,2100,200),  
    'min_samples_leaf':range(30,71,10),  
    'max_depth':range(5,16,2),  
    'min_samples_split':range(200,1001,200)  
})
```

- Best params for GradientBoostingClassifier: {'max_depth': 5, 'max_features': 7, 'min_samples_leaf': 30, 'min_samples_split': 200}
- Best Score: 0.5
- AUC Score on Testing Data: 0.805






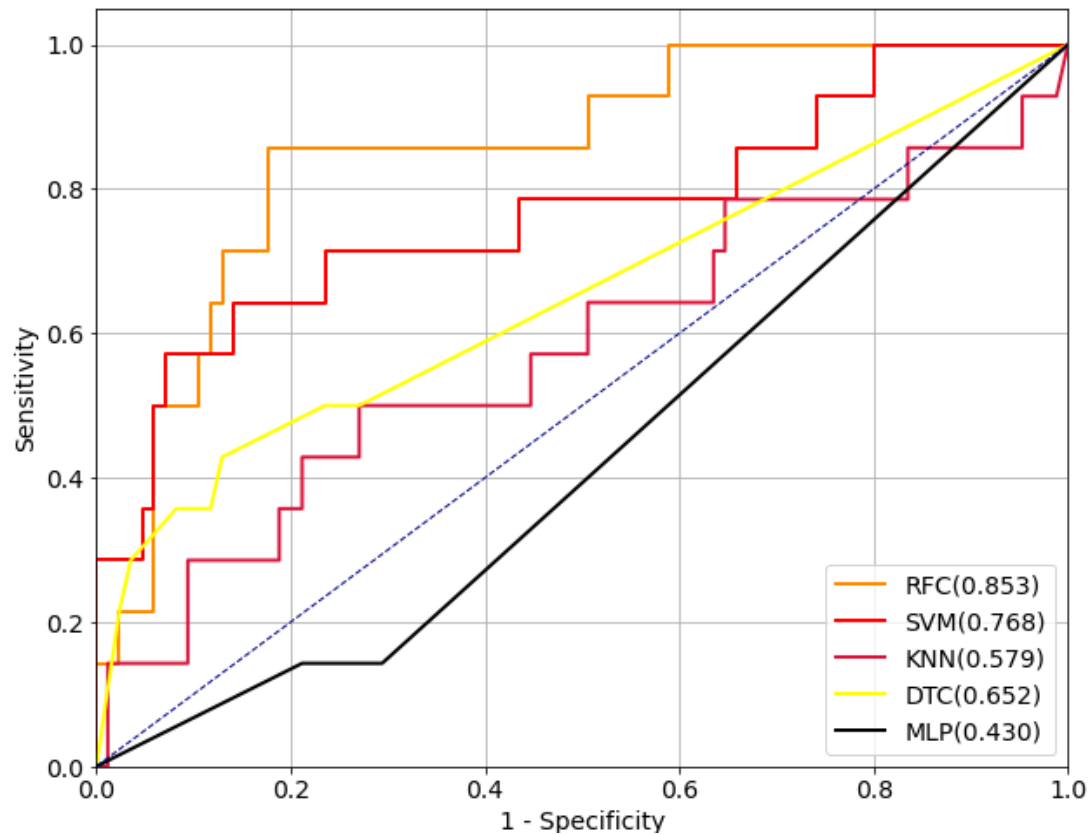
Results Analysis

Due to its robustness, **Random Forest** proved to be the most efficient algorithm overall, and it was the one with the best scores on all runs. A close second place was **Gradient Boosting**, which closely matched the Random Forest but it slightly underperformed. **Support Vector Machines** show signs to be applicable, but the amount of dimensions in the data make it hard to be good in their results consistently.

Algorithms like **KNN** and **MLP** are not a good match, mostly for their inability to find relevant relations like the ensemble methods described above



Compare the different curves





Conclusions

Throughout the development of this project, although feeling limited about the lack of data on the loans, we managed to achieve good results and score on kaggle competition, as we got the opportunity to explore different models.

On the future, we would like to search and try more models as well as get deep into the models we already used.

