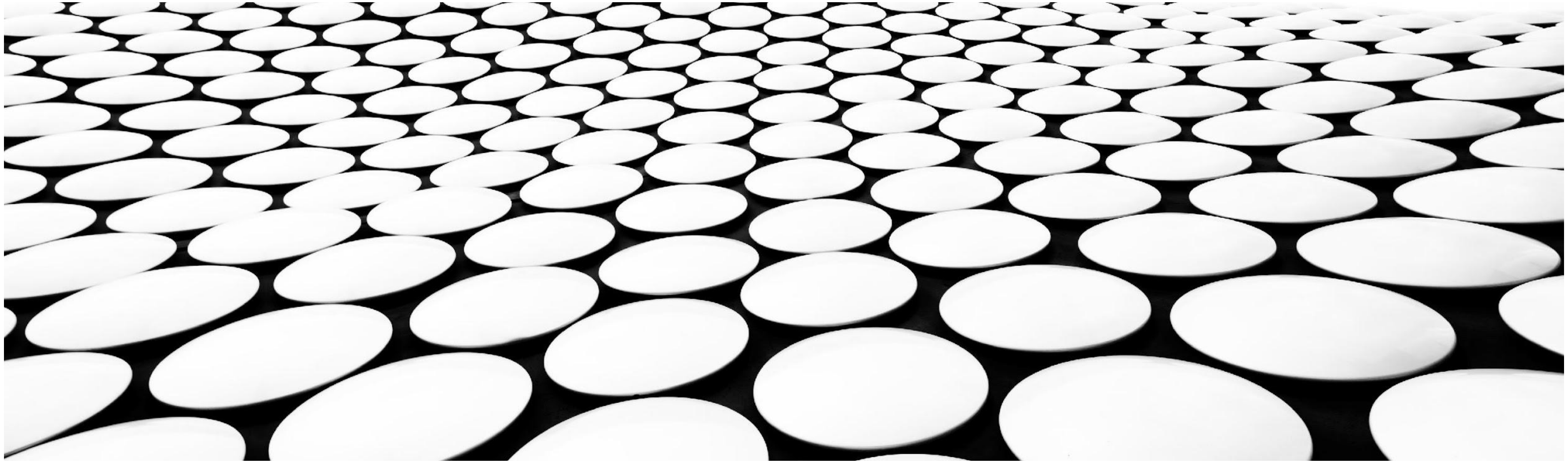
MACHINE LEARNING

PREDICTIVE DATA MINING EXERCISE

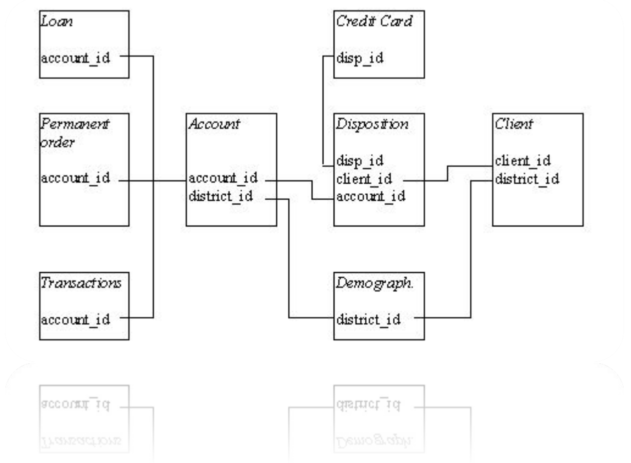


BUSINESS UNDERSTANDING
Case Description

We are given a large Dataset carrying the records of a **Czech bank**. The oldest record dates back to 1993 with the newest records being set in 1998. These records are stored in various comma-separated-values (CSV) files, each one related to a specific domain (Accounts, Transactions, Accounts, etc…).

Here are some quick conclusions about the overall structure:

- There are **77 distinct Czech districts** in these records**.**
- There is a total of **426 888** transactions, regardless of the type.
- We have **682 loans in total**. **328** have a known status. **354** do not.
- There are **4500 accounts** that have more than one client.



BUSINESS UNDERSTANDING
General Goals

Goal: Attempt to predict how probable it will be for a client to pay their loan based on numerous amounts of data and previous decisions taken.

Structure: The positive value will be -1, that being the ***unsuccessful*** class.

Data analysis goals: Pick the dataset and apply any necessary modifications that can help build a reliable and predictive method, capable of ***reducing the chances of the bank losing money*** from clients who would not pay their loan.

The results must be probabilistic, from a 0 to 1 range as a result, according to the positive class (*unsuccessful*).

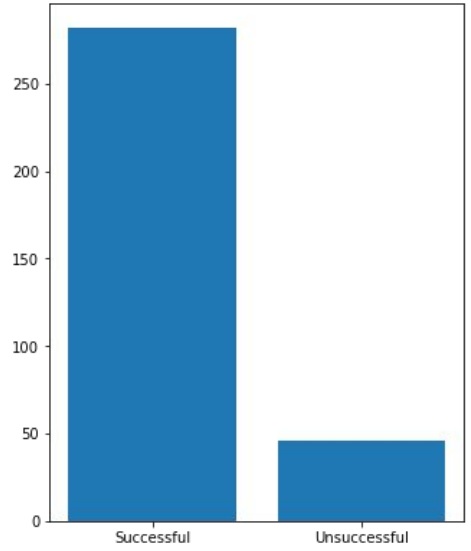
DATA UNDERSTANDING

Exploratory Data Analysis

After careful analysis of the data, we got the following results:

(1) - 86% (282) of the known loans have a "1" (successful) status, while 14% (46) have a "-1" (unsuccessful) status.

Therefore, the data is very **unbalanced.**



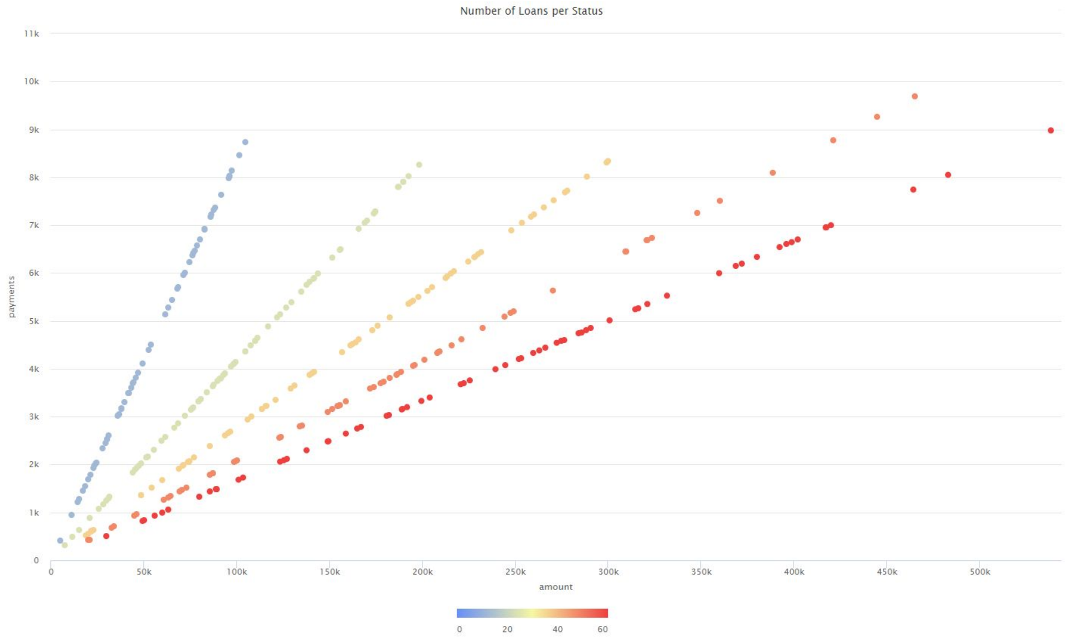
DATA UNDERSTANDING

Exploratory Data Analysis

After careful analysis of the data, we got the following results:

(2) – There is a correlation between the loan amount, the payments and its duration.

The colors define the duration.



DATA UNDERSTANDING

Exploratory Data Analysis

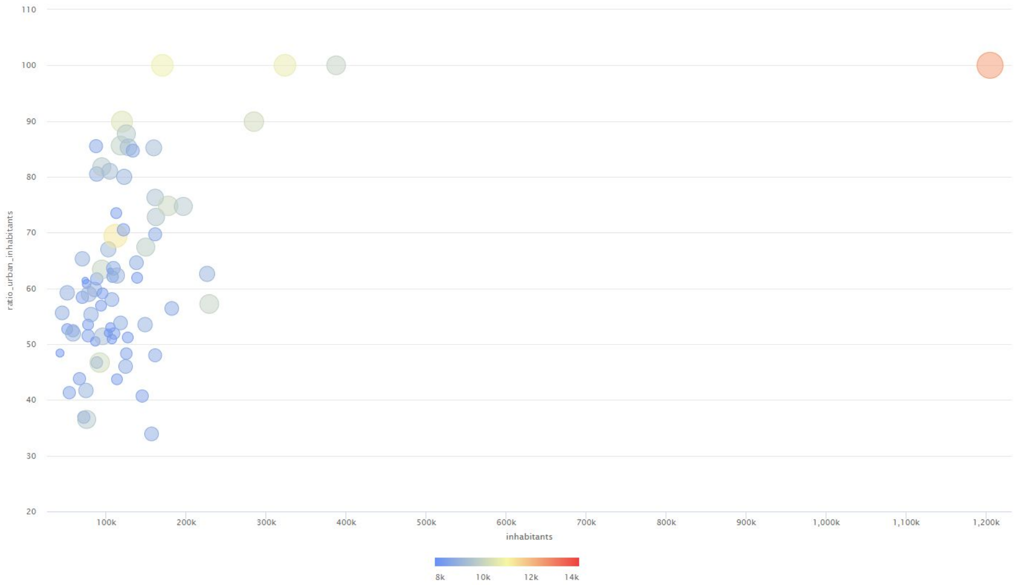
After careful analysis of the data, we got the following results:

(3) – All transactions occur before the loan is made or requested.

(4) – Every single user has, at most, one loan.

(5) – There is a tendency for higher salaries the higher the number of inhabitants and ration of urban inhabitants.

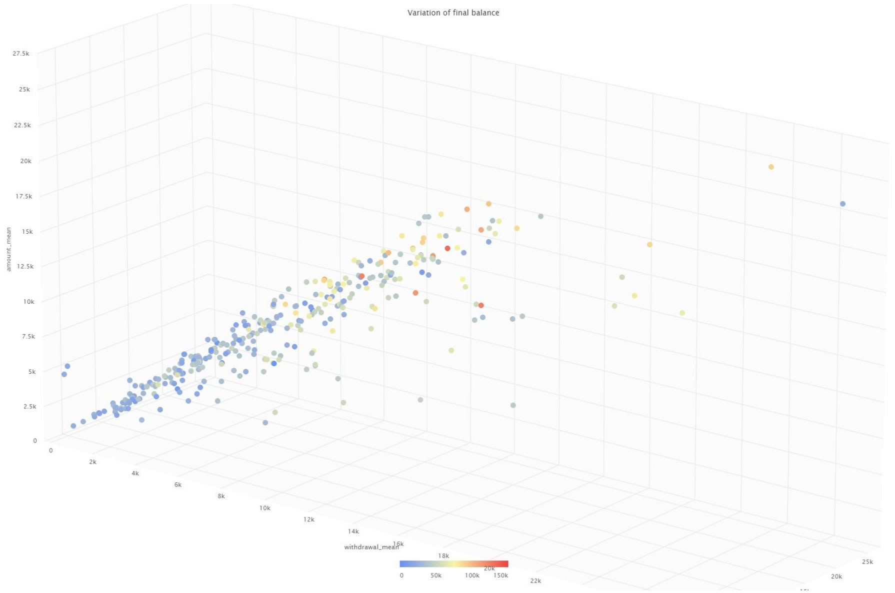
The color and circle size defines the avg_salary value.

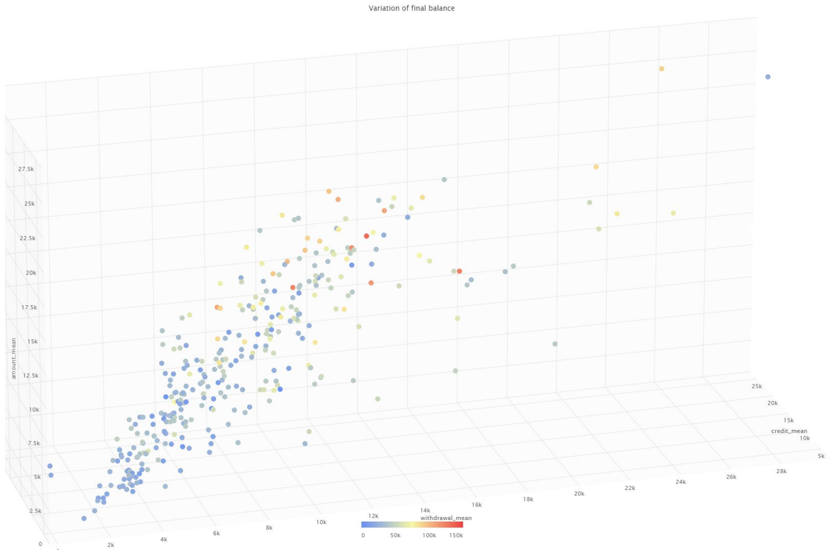


DATA UNDERSTANDING

Exploratory Data Analysis

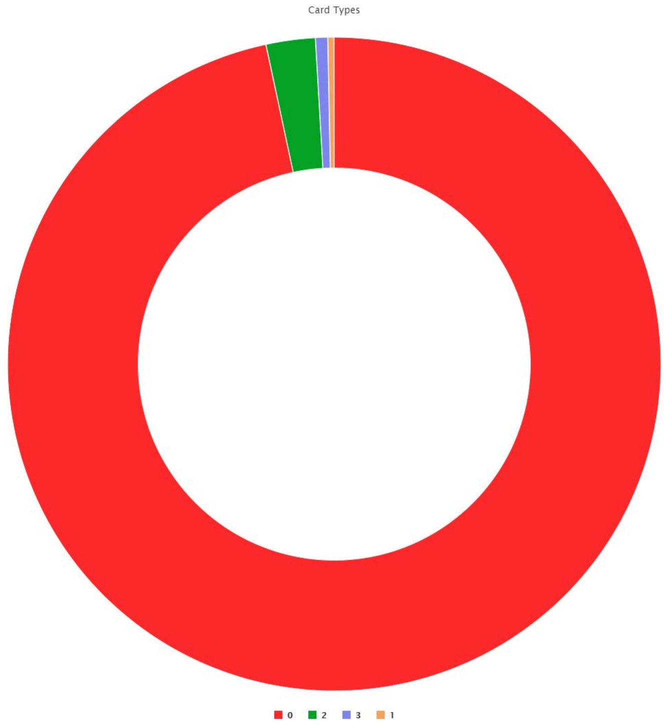
(6) – The loan amount has a tendency to rise based on the mean values of the credits and withdrawals.

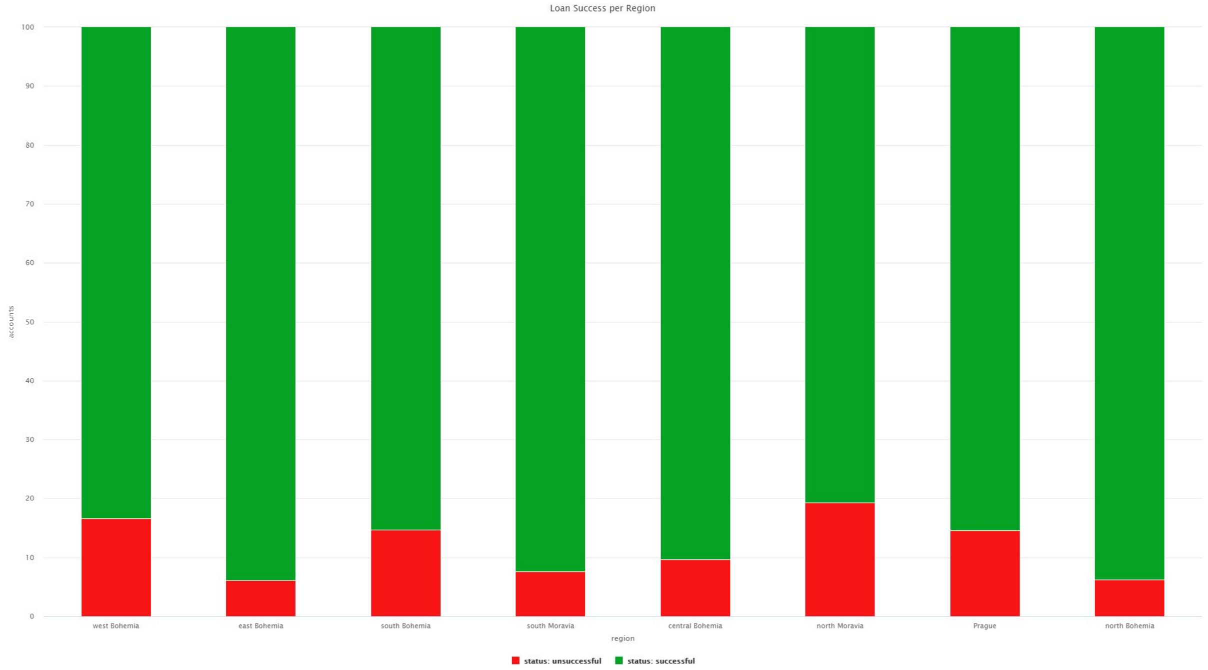




DATA UNDERSTANDING

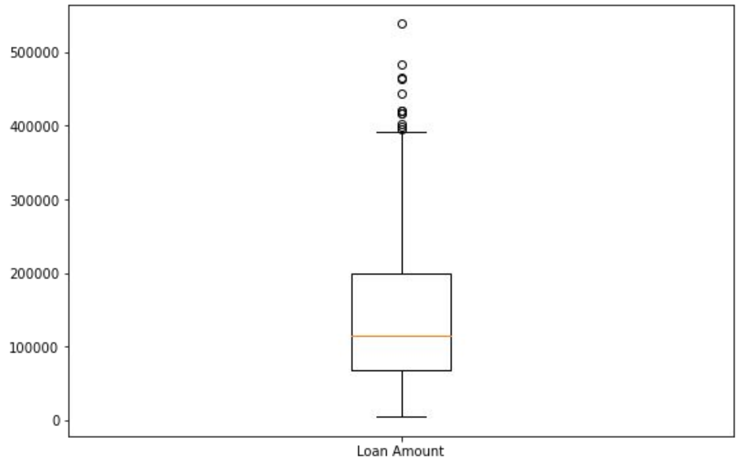
Exploratory Data Analysis

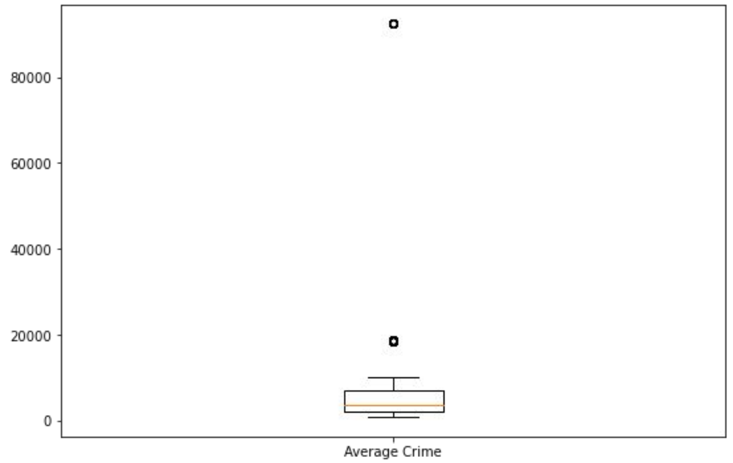


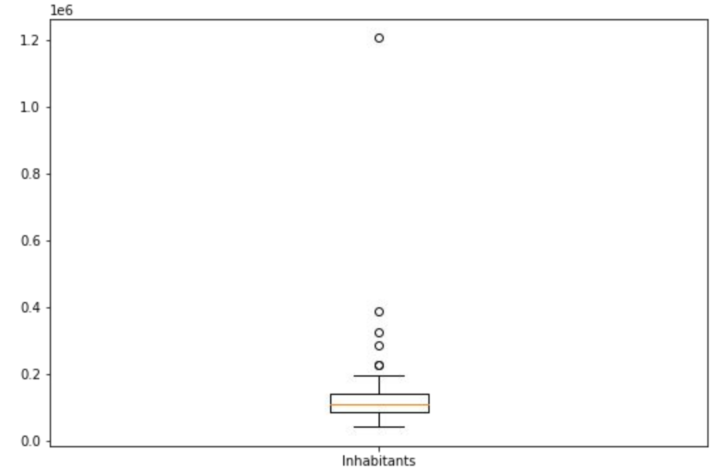


DATA UNDERSTANDING

Outliers







DATA PREPARATION
Adaptation

In order to use the data for our modeling process, we changed some of its contents to better fit our modeling software:

- We extracted and broke down the birthdate and the gender out of their *birth_number*
- Dates were transformed into the *yyyy-mm-dd* format
- We turned all categorical classes into numbers such as the type of credit and gender (e.g: M □ 0, F □ 1)
- Replacing our “?” values in the District dataset with the interpolation of the two closest cities.
- Detection of outliers
- Removal of unused accounts (automatically removed in the JOIN processes between tables)
- …

DATA PREPARATION

Feature Engineering

Domain knowledge was used to select and transform certain parts of our dataset into a more workable and understandable format that could help us apply it on a learning model. Most engineered features came from the ***Transactions*** class.

Here are some examples of generated features:

- **Statistical metrics** – max, min, mean – for the amount and balance of each client that did transactions.
- **Age** of the Account holder
- Tracking of **how many credits and withdrawals** were made.
- Standard Deviation of the withdrawals amount and credits amount for each client involved
- The Final Balance of each client (checking the latest transaction)
- The “delta” of the Balance for each client (used to see if they spent more than they earned)

EXPERIMENTAL APPLICATION

Application Pipeline

We have set-up a static pipeline to be used throughout each step of our process:

- Preprocessing according to previous slides
- Prediction Process:
    - **Feature Selection** (next slide)
    - Application of **Downsampling**, **Upsampling** or **both**.
    - Application of **grid search** (with respect to the AUC statistic) for hyper-parameter optimization. Combinations also include the best SMOTE, Undersampling (or both) values for each model used.
    - Usage of Cross Validation of the Training set with respect to the **AUC** statistic.
    - Visualize, using RapidMiner’s **ROC** comparison, which models have the highest **AUC** value.
    - Apply the manually selected models, acquire the prediction results and make an average of them.

FEATURE SELECTION

Multiple Alternatives

In order to select which features were most relevant, the group chose a unique approach, but didn’t render out other possible methods. Therefore, the methods used were:

- No feature selection whatsoever
- Manual removal of attributes based on weighted results (***chosen for submission***)
- Backwards elimination
- Both of the last two.

Note: Regardless of the method used, any ‘id’-discriminated attribute was manually removed before the alternative’s inception.

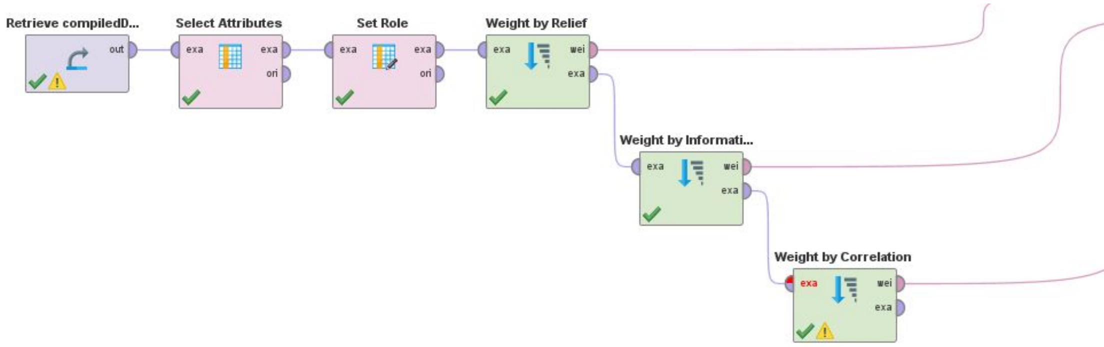
FEATURE SELECTION

Manual Removal based on Weighted results

The manual removal of features was done based on three metrics:

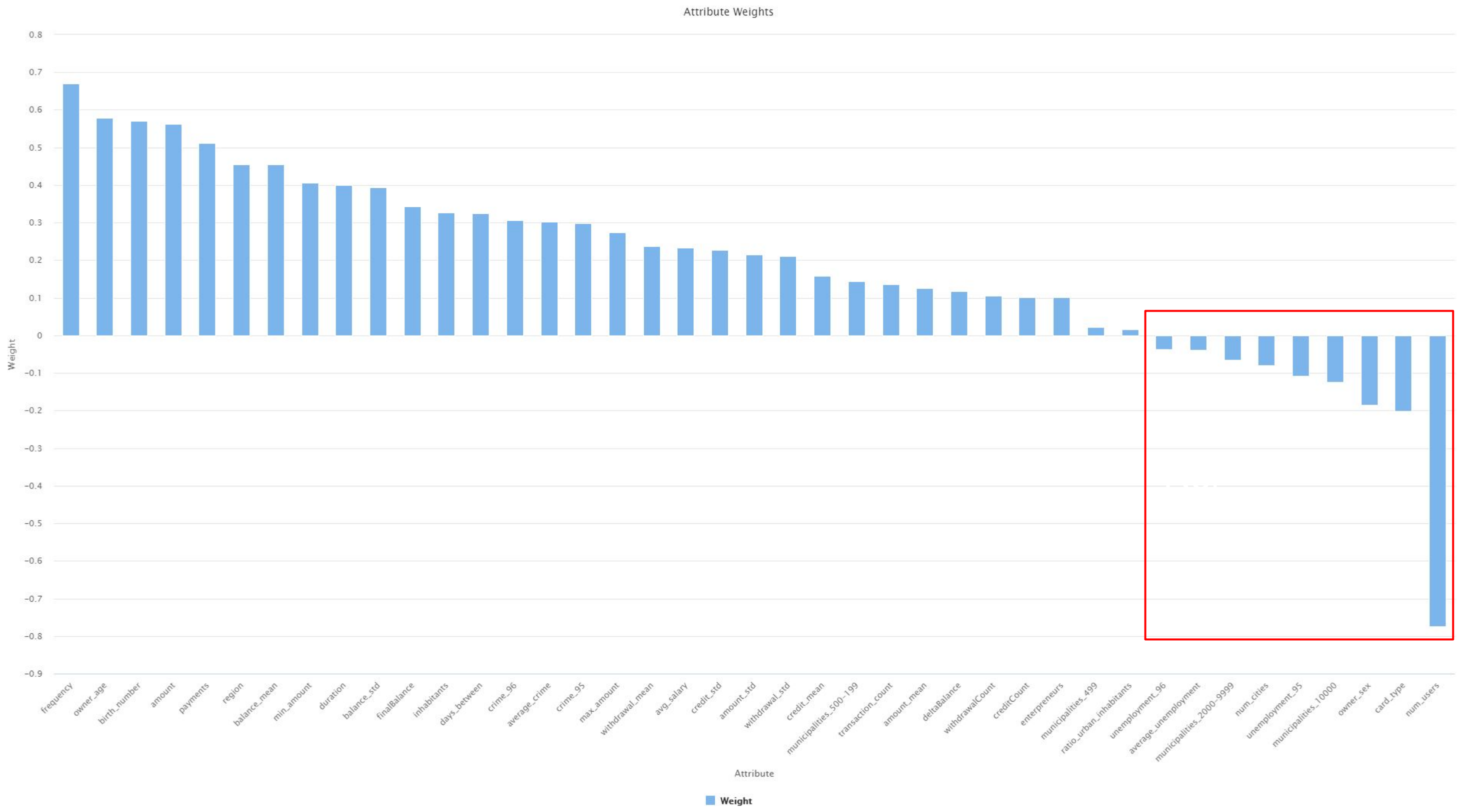
- Weight by Relief
- Weight by Information Gain Ratio
- Weight by Correlation

Based on the results we removed all negative-relief attributes, removed low-information gain ratio'ed values, and attributes whose correlation wasn't satisfactory.



FEATURE SELECTION

Manual Removal based on Weighted results

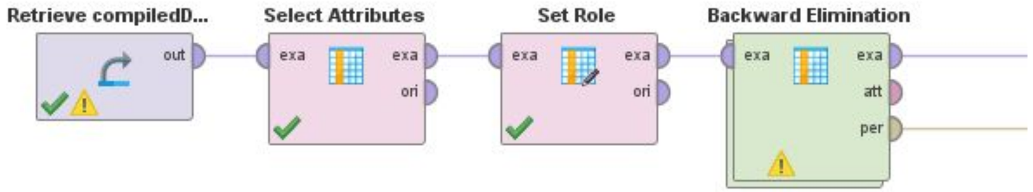


Relief Example

FEATURE SELECTION

Backwards Elimination

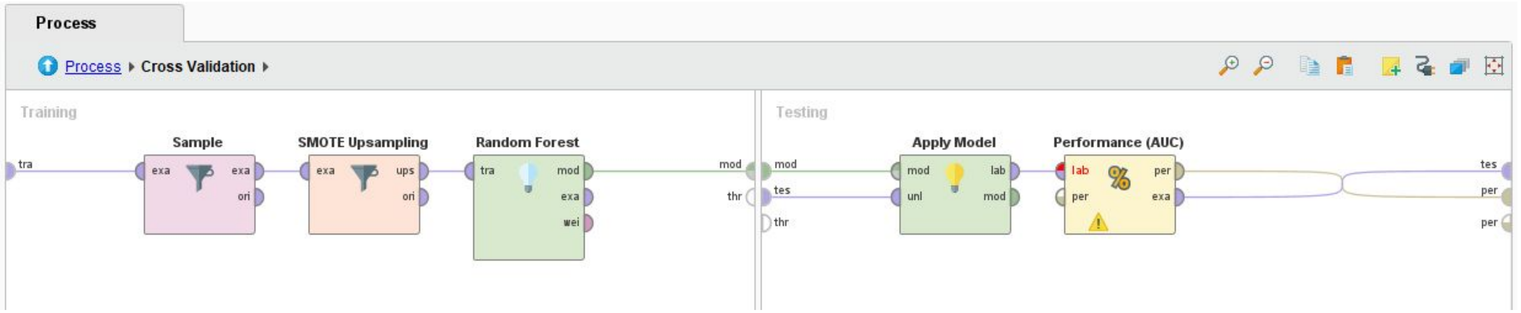
The Backwards Elimination method consists on using a Cross Validation technique for each step of the Backwards Elimination process. This is done automatically by RapidMiner, producing a unique result based on each Model we intended to use.



AUC Metric

Performance Evaluation

The AUC result comes as a result of the Cross-Validation method's performance evaluation. In order to produce an accurate result without poisoning the test dataset, the SMOTE processing and Undersampling application is done within the **training** set, as listed in the process bellow (Random Forest):



Note: The ***Grid Search*** mentioned in previous slides also follows this disposition.

AUC Results

Tabular Output (I)

	Feature Selection	None				Manual Removal - Weighted			
	Sampling	None	Down	Up	Both	None	Down	Up	Both
Algorithms	Random Forest	0.831	0.826	0.801	0.811	0.824	0.827	0.823	**0.847**
	Decision Tree	0.614	0.615	0.588	0.519	0.587	0.577	0.519	0.436
	Logistic Regression	0.784	0.783	0.791	0.788	0.795	0.795	0.784	0.778
	Gradient Boosted Trees	0.803	0.795	0.802	0.792	0.798	0.798	0.772	0.778

AUC Results

Tabular Output (II)

	Feature Selection	Backwards Elimination				Both			
	Sampling	**None**	**Down**	**Up**	**Both**	**None**	**Down**	**Up**	**Both**
Algorithms	**Random Forest**	0.817	0.820	0.809	0.825	0.823	0.827	0.779	0.765
	Decision Tree	0.576	0.600	0.542	0.536	0.507	0.553	0.613	0.510
	Logistic Regression	0.794	0.796	0.784	0.790	0.809	0.811	0.804	0.815
	Gradient Boosted Trees	0.798	0.796	0.796	0.727	0.802	0.808	0.756	0.785

AUC Results - Conclusion

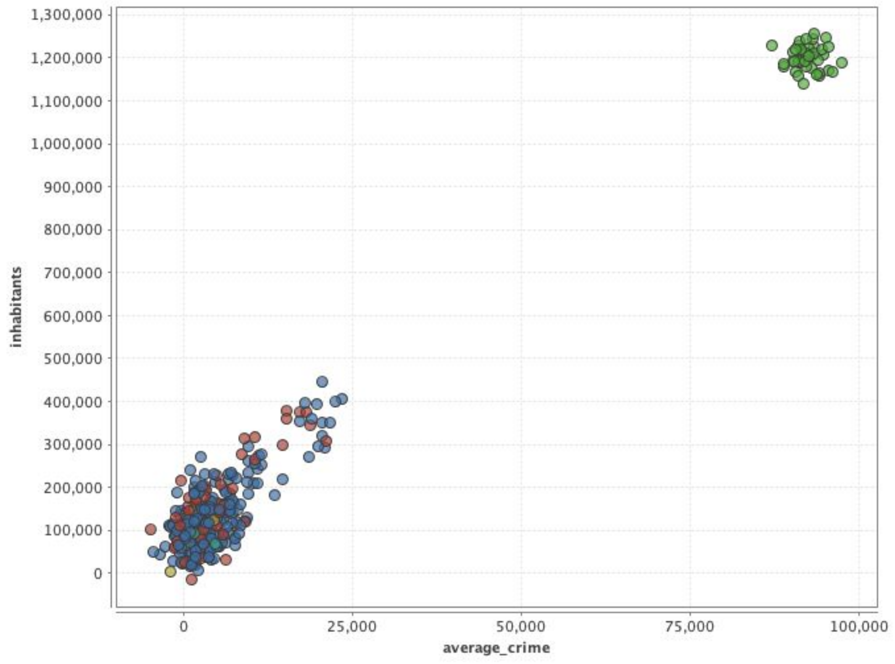
In other words …

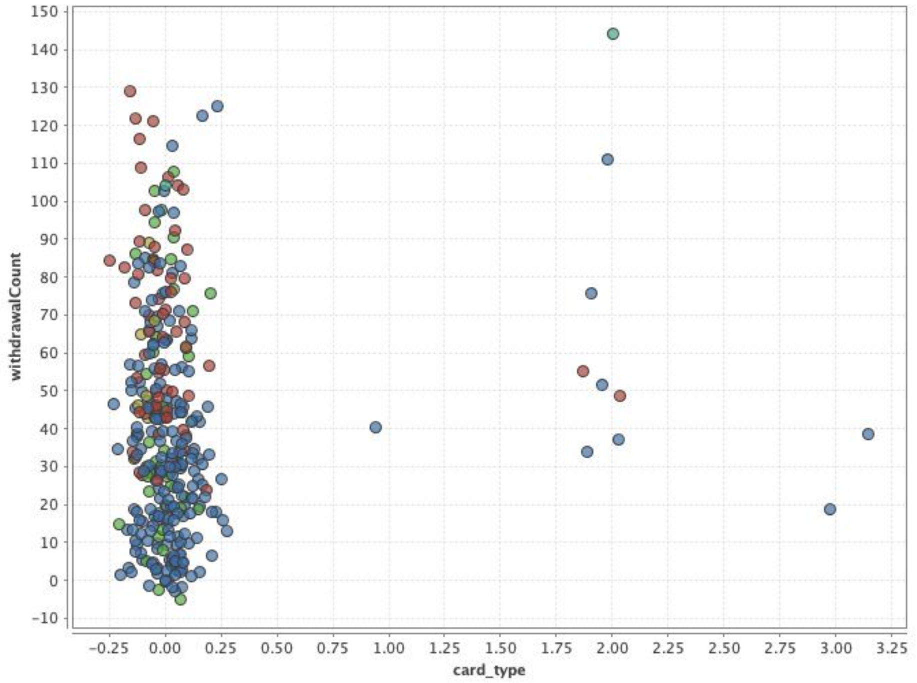
- Average score of 0.816 for ***Random Forest*** across all possible combinations.
- Random Forest scores the best next to the other models used, with the highest average score.
- Oversampling and Undersampling ***seem to increase*** performance ***in some cases***.
- Using ***both methods*** of Feature Selection did not work. After the oversampling-undersampling process, the Random Forest algorithm, supposedly the best, ranked the worst compared to others, second to the Decision Tree.
- The Decision Tree is not a good algorithm for this scenario. Despite the optimized hyper-parameters, its performance is fairly low and a terrible choice. *Thus, it was not used in the average-step for submission*.
- The highest possible score was **0.847**, using Random Forest with Manual Feature Selection and both sampling techniques.

Data Description

Clusters - K-Means

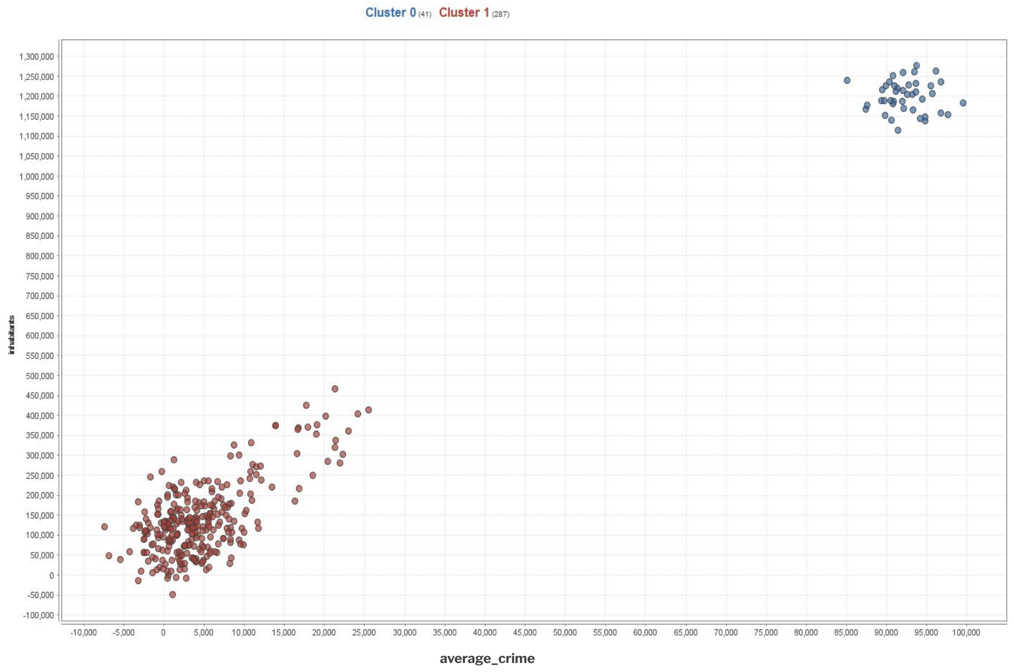
Cluster 0 (210) Cluster 1 (3) Cluster 2 (41) Cluster 3 (6) Cluster 4 (68)

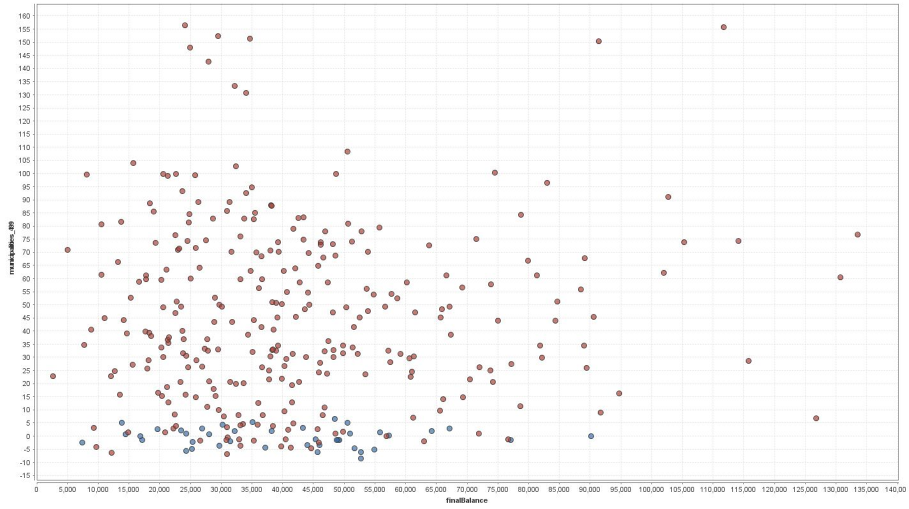




Data Description

Clusters - K-Medoids





Conclusion

Kaggle Competition score:

- Public: 0.95679
- Private: 0.92448

Things we could have done to improve:

- Adding covariance between attributes throughout the feature engineering process on the Transactions Table.
- Testing more learning models

Contributions

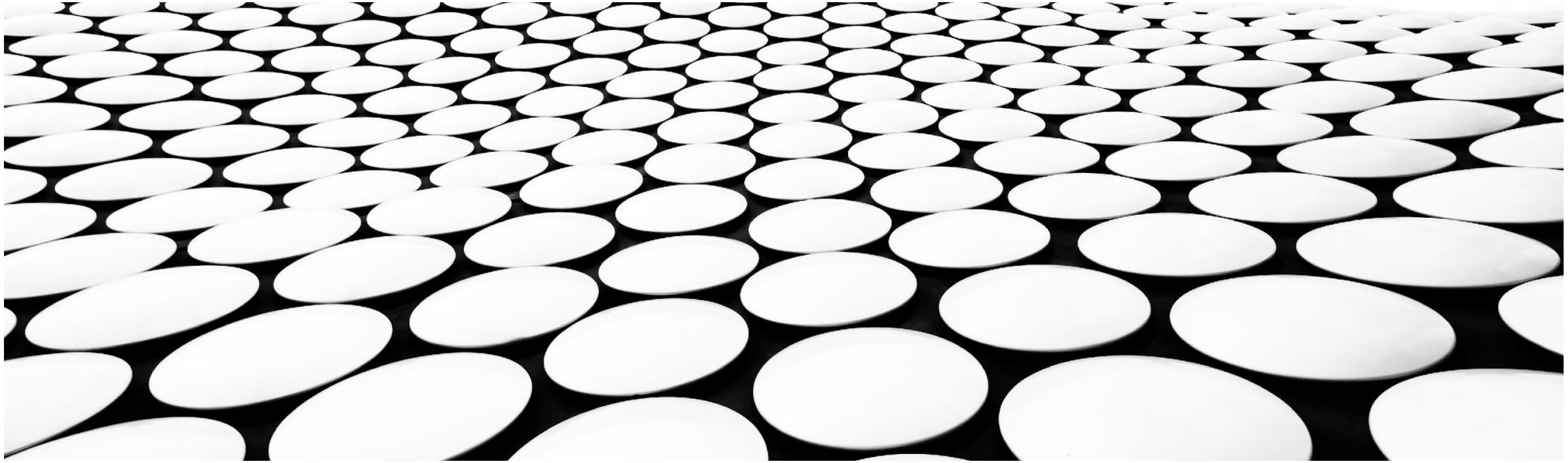
Daniel da Silva Gonçalves - 47.5%

André Pedro de Melo Malheiro - 10%

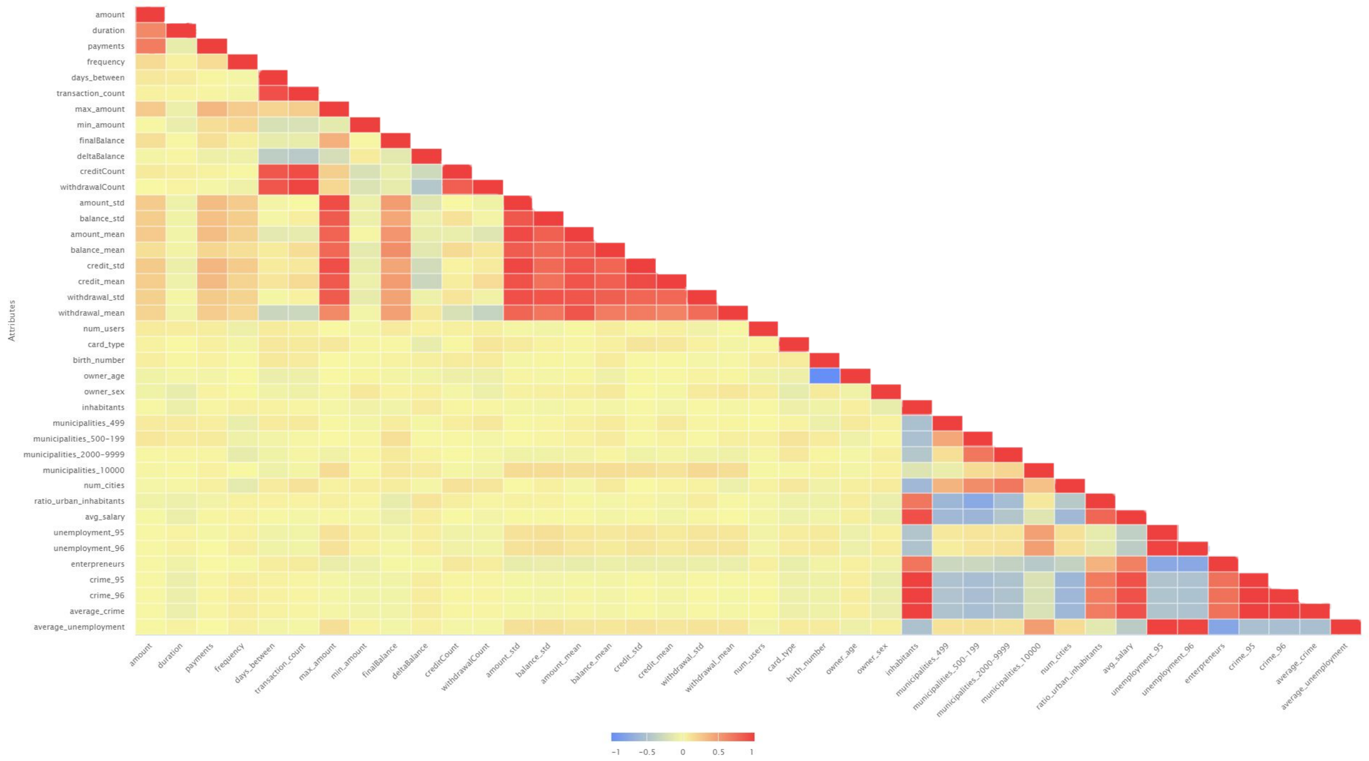
Pedro Miguel Novais do Vale - 42.5%

ANNEXES

Partial Information



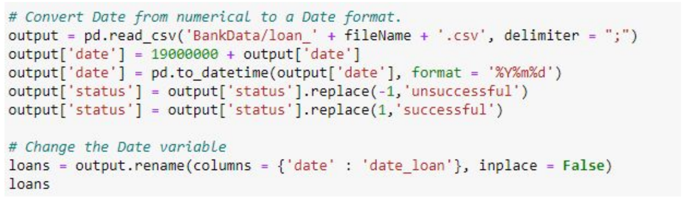
Correlation Matrix



Data Processing

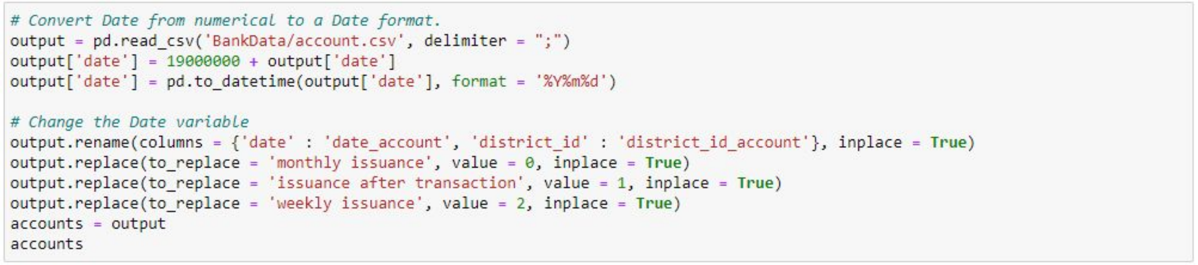
Python Scripts

Loan Table - Handler

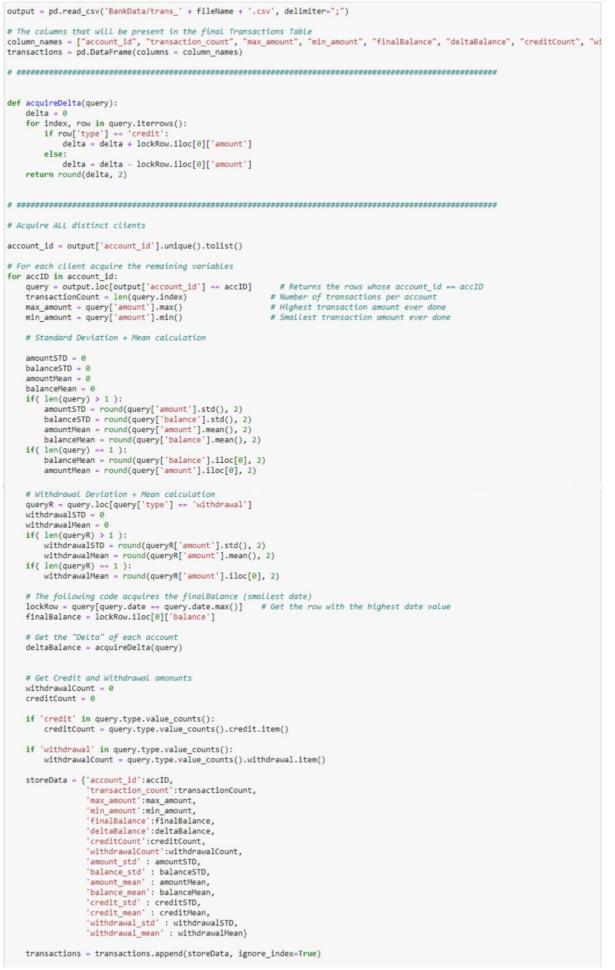


```
# Convert Date from numerical to a Date format.
output = pd.read_csv('BankData/loan_' + fileName + '.csv', delimiter = ";")
output['date'] = 19000000 + output['date']
output['date'] = pd.to_datetime(output['date'], format = '%Y%m%d')
output['status'] = output['status'].replace(-1,'unsuccessful')
output['status'] = output['status'].replace(1,'successful')

# Change the Date variable
loans = output.rename(columns = {'date' : 'date_loan'}, inplace = False)
loans
```

Account Table - Handler



```
# Convert Date from numerical to a Date format.
output = pd.read_csv('BankData/account.csv', delimiter = ";")
output['date'] = 19000000 + output['date']
output['date'] = pd.to_datetime(output['date'], format = '%Y%m%d')

# Change the Date variable
output.rename(columns = {'date' : 'date_account', 'district_id' : 'district_id_account'}, inplace = True)
output.replace(to_replace = 'monthly issuance', value = 0, inplace = True)
output.replace(to_replace = 'issuance after transaction', value = 1, inplace = True)
output.replace(to_replace = 'weekly issuance', value = 2, inplace = True)
accounts = output
accounts
```



```
output = pd.read_csv('BankData/trans_' + fileName + '.csv', delimiter=";")

# The columns that will be present in the final Transactions Table
column_names = ["account_id", "transaction_count", "max_amount", "min_amount", "finalBalance", "deltaBalance", "creditCount", "wi
transactions = pd.DataFrame(columns = column_names)

# ###############################################################################################


def acquireDelta(query):
    delta = 0
    for index, row in query.iterrows():
        if row['type'] == 'credit':
            delta = delta + lockRow.iloc[0]['amount']
        else:
            delta = delta - lockRow.iloc[0]['amount']
    return round(delta, 2)


# ###############################################################################################

# Acquire ALL distinct clients

account_id = output['account_id'].unique().tolist()

# For each client acquire the remaining variables
for accID in account_id:
    query = output.loc[output['account_id'] == accID]      # Returns the rows whose account_id == accID
    transactionCount = len(query.index)                   # Number of transactions per account
    max_amount = query['amount'].max()                    # Highest transaction amount ever done
    min_amount = query['amount'].min()                    # Smallest transaction amount ever done

    # Standard Deviation + Mean calculation

    amountSTD = 0
    balanceSTD = 0
    amountMean = 0
    balanceMean = 0
    if( len(query) > 1 ):
        amountSTD = round(query['amount'].std(), 2)
        balanceSTD = round(query['balance'].std(), 2)
        amountMean = round(query['amount'].mean(), 2)
        balanceMean = round(query['balance'].mean(), 2)
    if( len(query) == 1 ):
        balanceMean = round(query['balance'].iloc[0], 2)
        amountMean = round(query['amount'].iloc[0], 2)


    # Withdrawal Deviation + Mean calculation
    queryR = query.loc[query['type'] == 'withdrawal']
    withdrawalSTD = 0
    withdrawalMean = 0
    if( len(queryR) > 1 ):
        withdrawalSTD = round(queryR['amount'].std(), 2)
        withdrawalMean = round(queryR['amount'].mean(), 2)
    if( len(queryR) == 1 ):
        withdrawalMean = round(queryR['amount'].iloc[0], 2)

    # The following code acquires the finalBalance (smallest date)
    lockRow = query[query.date == query.date.max()]    # Get the row with the highest date value
    finalBalance = lockRow.iloc[0]['balance']

    # Get the "Delta" of each account
    deltaBalance = acquireDelta(query)


    # Get Credit and Withdrawal amonunts
    withdrawalCount = 0
    creditCount = 0

    if 'credit' in query.type.value_counts():
        creditCount = query.type.value_counts().credit.item()

    if 'withdrawal' in query.type.value_counts():
        withdrawalCount = query.type.value_counts().withdrawal.item()

    storeData = {'account_id':accID,
                 'transaction_count':transactionCount,
                 'max_amount':max_amount,
                 'min_amount':min_amount,
                 'finalBalance':finalBalance,
                 'deltaBalance':deltaBalance,
                 'creditCount':creditCount,
                 'withdrawalCount':withdrawalCount,
                 'amount_std' : amountSTD,
                 'balance_std' : balanceSTD,
                 'amount_mean' : amountMean,
                 'balance_mean': balanceMean,
                 'credit_std' : creditSTD,
                 'credit_mean' : creditMean,
                 'withdrawal_std' : withdrawalSTD,
                 'withdrawal_mean' : withdrawalMean}

    transactions = transactions.append(storeData, ignore_index=True)
```

Transactions Table - Handler

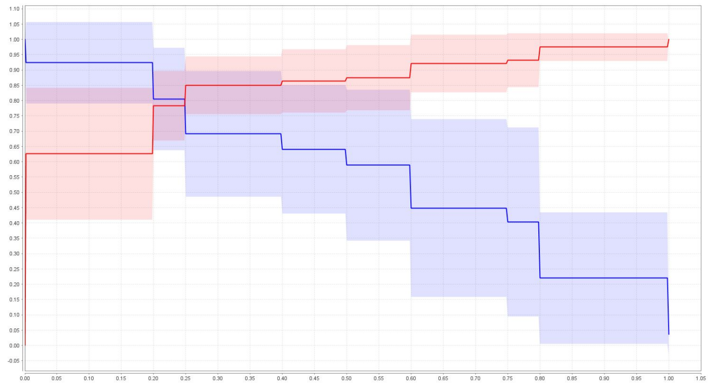
Algorithm Analysis

Logistic Regression

Functioning: Tries to create a prediction model based on a logit function through mathematical means.

Characterization / Keypoints:

- Used when the dependent variable is categorical (***this is the case!***)
- Fast and simpler.
- There should be no non-meaningful features as they may spoil the results.
- The model does not handle outliers very well.



AUC - 0.815

Algorithm Analysis

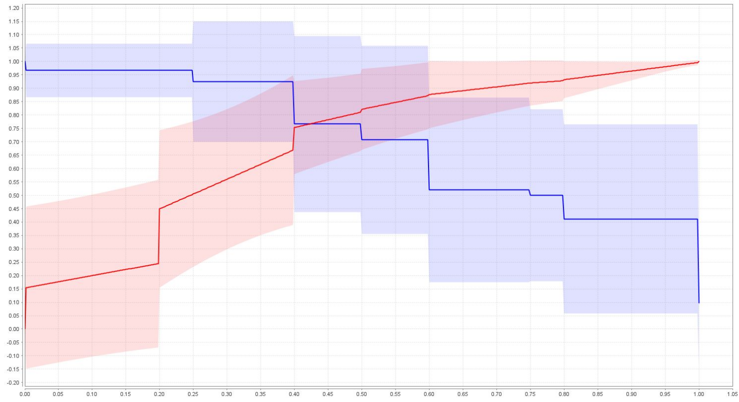
Decision Tree

Functioning: It uses multiple algorithms to decide when to branch out a given node, creating sub-nodes and leafs, until it reaches a maximum depth. The final result is a tree-like structure.

Prediction Election: Starting at the root, we travel down the nodes, choosing the branches based on the conditions that meet the given data. When we reach a leaf, we obtain its prediction.

Characterization / Keypoints:

- Prone to overfitting.
- ... therefore likely to create biased trees if some class dominates the other (***this is the case!***)
- Small variations may lead to a completely different decision tree.
- Easier to understand and visualize.



AUC - 0.615

Algorithm Analysis

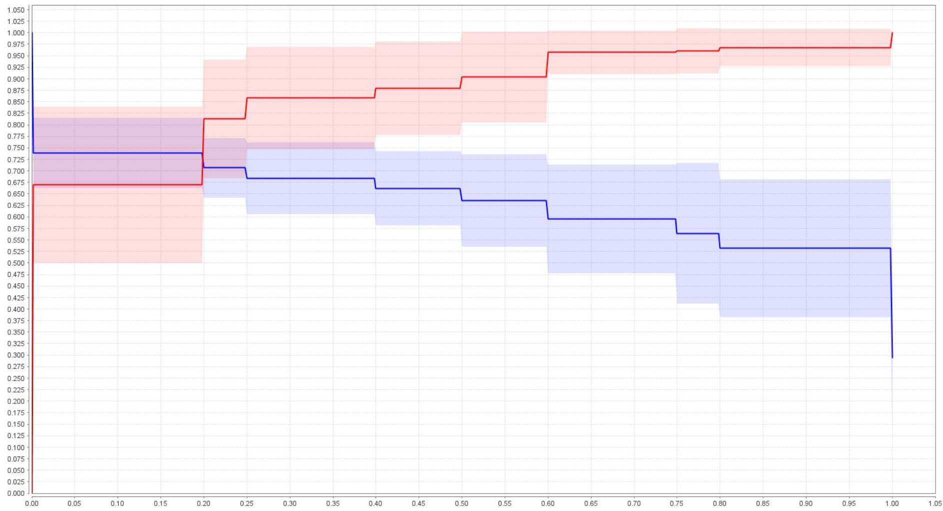
Random Forest

Functioning: The algorithm generates a given number of uncorrelated models (decision trees) based on input data.

Prediction Election: All decision trees will vote out their own prediction, and the most voted one will be chosen as our model's prediction.

Characterization / Keypoints:

- Less likely to overfit due to its bag of trees**.**
- Much harder to visualize than a Decision Tree.
- Has its own method of feature selection.
- Based on the "*Wisdom of the Crowds*"



AUC - 0.847

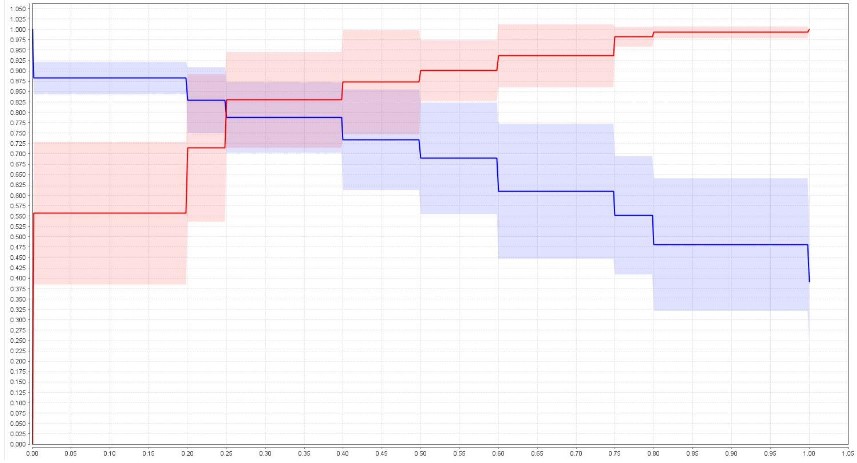
Algorithm Analysis
Gradient Boosted Trees

Functioning: Uses boosting to create a stronger learner. Relies on the intuition that the error is reduced when combining previous models (weak learners) together. In this case, the weak learner is a decision tree.

Prediction Election: Election, same as random forest, after the learning model is set-up.

Characterization / Keypoints:

- Too many trees, unlike random forest, leads to overfitting.
- Highly efficient.
- Requires careful tuning of its hyper-parameters ...
- Sensitive to outliers (like logistic regression)


AUC - 0.808

Tools used

Python: Preprocessing of the data, feature engineering, row-distance measuring.

RapidMiner: Prediction components, Grid Search, Model training, Feature Selection, Cross Validation.

Excel / Visual Studio Code: Table visualization (*w/ extensions*)