U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

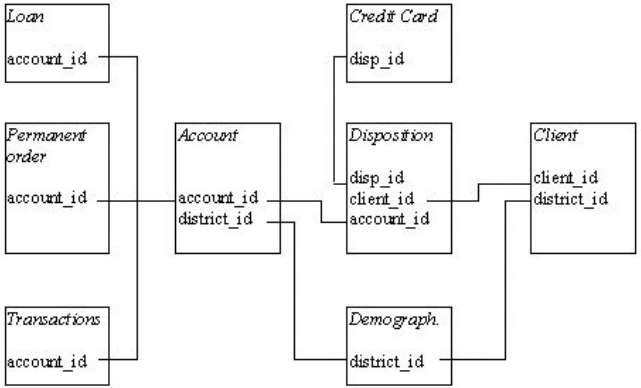
To loan or not to loan?

Machine Learning
2021/22

Group 28

David R Ferreira, 202102686
Rui Pinto, 201806441
Tiago Gomes, 201806658

Domain and Business Understanding



Records from a Czech bank from 1993 to 1998.

- 5369 clients.
- 4500 accounts.
- 329 loans with status.
- 354 loans without status.
- 77 geographical districts.
- 426 885 transactions.

Goal: predict whether a client will pay the loan successfully or not.

Business Goals

- A bank doesn't want to lose money with the concession of loans to customers who are not going to be able to pay them.
- A bank wants to improve its understanding of its customers.

Data Mining Goals

- A model that will predict the probability of whether a customer will be able to pay a loan or not, given his personal information and the information of past customers and their success status.
- Determine patterns, relations, and behaviors of attributes in a given dataset.

Tools


matplotlib


NumPy


pandas


plotly


python™


scikit
learn


seaborn


SciPy


GitHub


git

Assessment of Data Quality

Completeness	• Critical and required data values are generally present. • But there are some records with missing attributes (card details dataset, 60% of missing values for *bank* and *account* properties on the transactional relation, …).
Conformity	• In most cases, provided data matches rules specified and expected for it. • But, for example, birth_number field in the Client dataset mixes birth date with gender.
Timeliness	• All loans' data are from between 1993 and 1998.
Uniqueness	• We can assume that the records are unique only by their IDs.

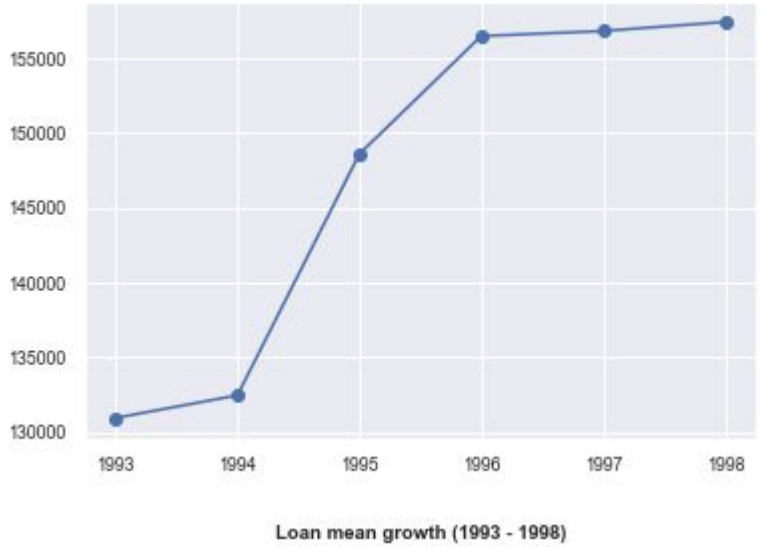
Data preparation

- Extract age and gender from ‘birth_number’ on the Client dataset.
- Convert some dates to “years since...” (for example, on Account dataset date was converted to years since account was created).
- Replace ‘unemployment rate 95’ and ‘no. of commited crimes 95’ missing values on the District dataset by computing the mean variation between 95’ and 96’ for known values and applying it to the known respective 96’ value.
- Convert values of withdrawals to negative.
- Relation permanent order is missing so it was ignored.
- Credit Card relation had entries for a very limited subset of the total clients. After exploring the possible importance of those attributes with the help of plots, it was decided not to use this table to train our model.
- Transactions’ last date is previous to the loan date.
- Renamed some columns for easier use.

Feature Engineering

- Discretization of equal frequency age groups.
- Created new features from the “operation” column in the transactions relation:
  - IC (Interest Credited).
  - CC (Credit in cash).
  - CAB (Collection from another bank).
  - WC (Withdrawal in cash).
  - RAB (Remittance to another bank).
  - CCW (Credit card withdrawal).
  - total_ops (Total number of operations per client).
  - credit_ratio and withdrawal_ratio (Ratio of credits and withdrawals).
  - reached_negative_balance (Boolean for when an account balance reaches negative values).
  - etc.
- Created new features in the district relation:
  - ratio_entrepeneurs, criminality_growth, unemploymant_growth.
- Statistic methods for many existing features such as balances, credits, withdrawals (min, max, mean, count, standard deviation).
- Complete list in the **Jupyter Notebook.**

Loan Statistics

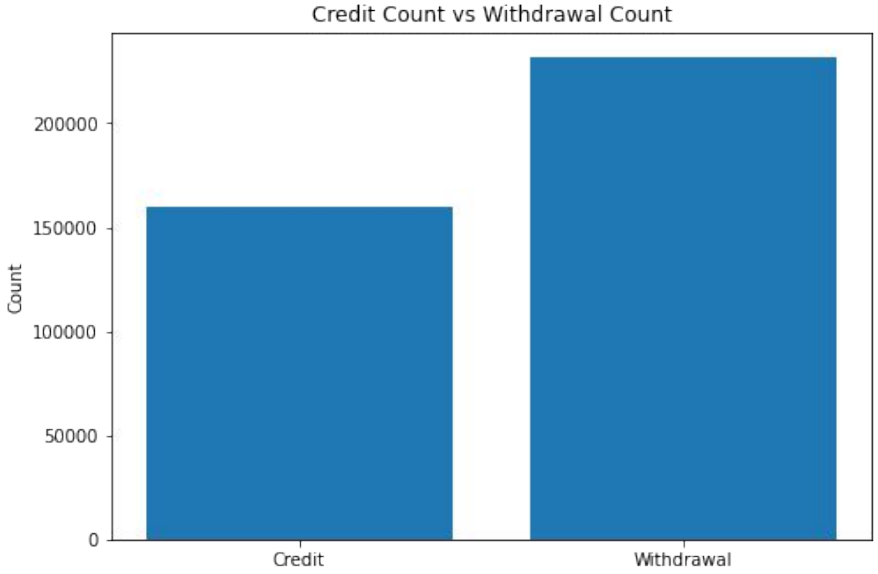


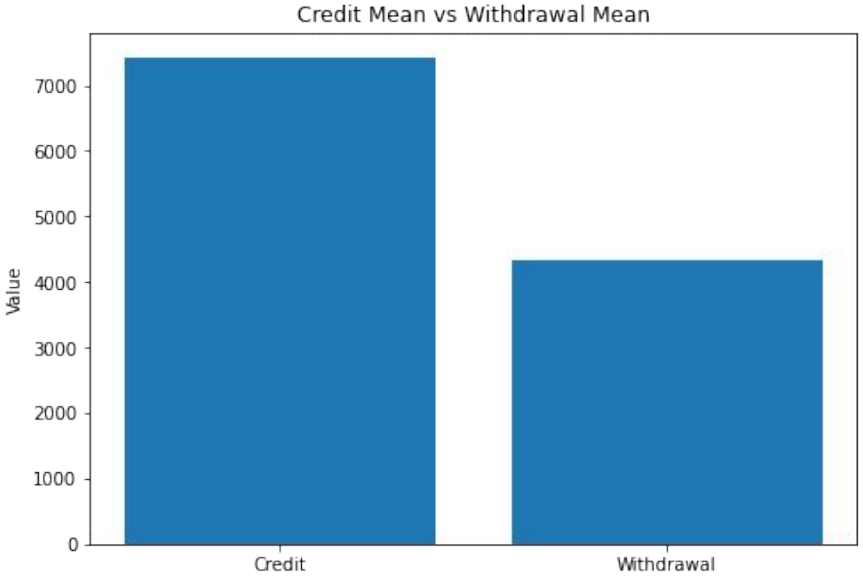
Loan mean growth (1993 - 1998)

Loan amount Standard Deviation (1993 - 1998)

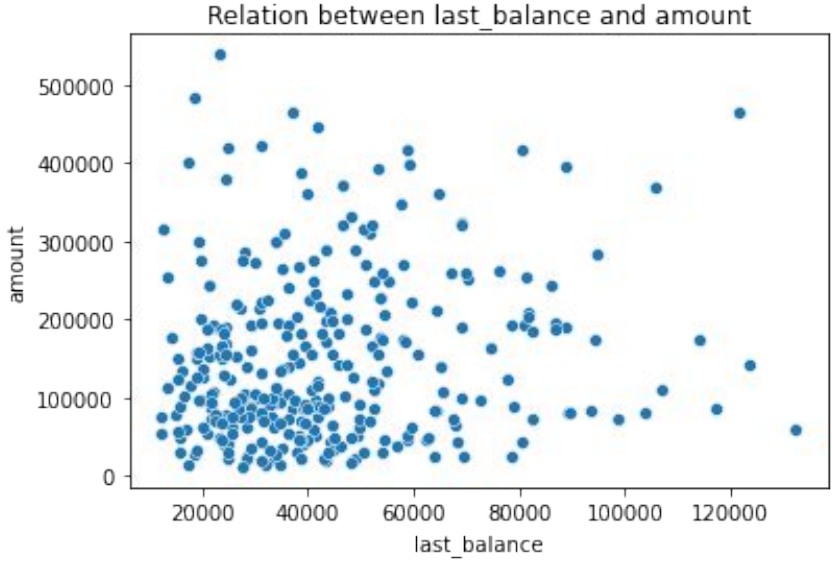
Through the years there was an increase in the loan amounts and those amounts were more spread relative to the mean loan amount asked.

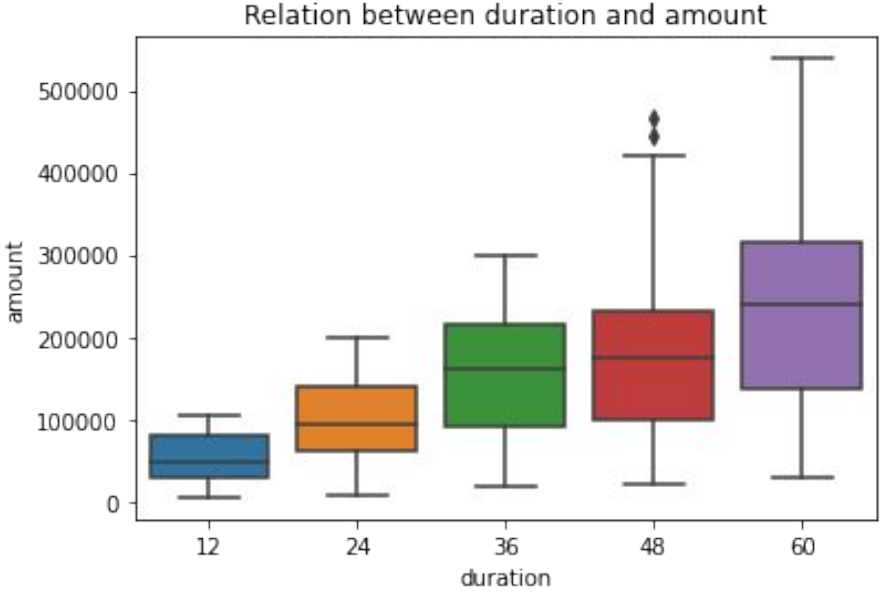
Credit and Withdrawal data





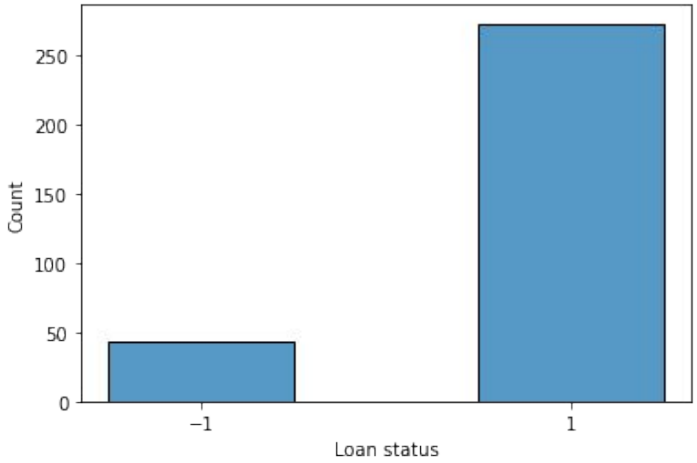
Although there were more withdrawal transactions than the Credit ones, the Credit's amounts were higher.



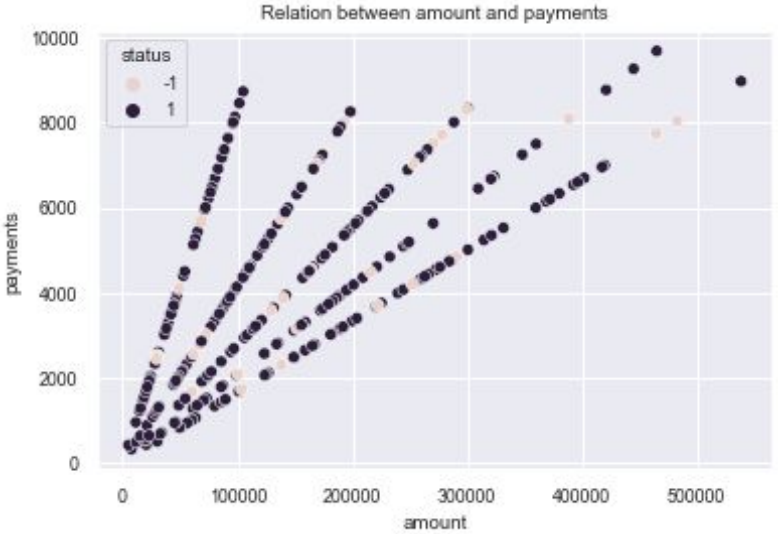


Big concentration of people with low balances and asking for low amount loans.

Typically, higher duration loans mean higher amounts.

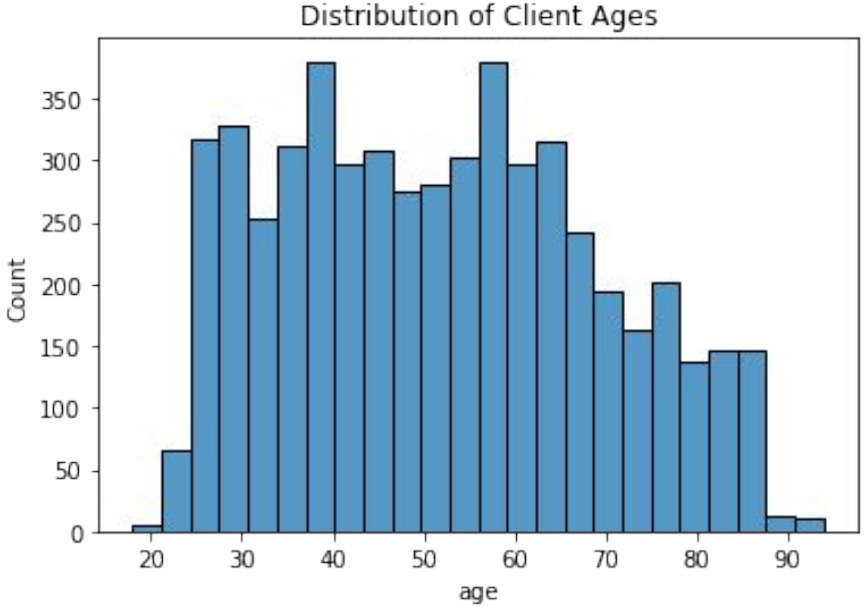


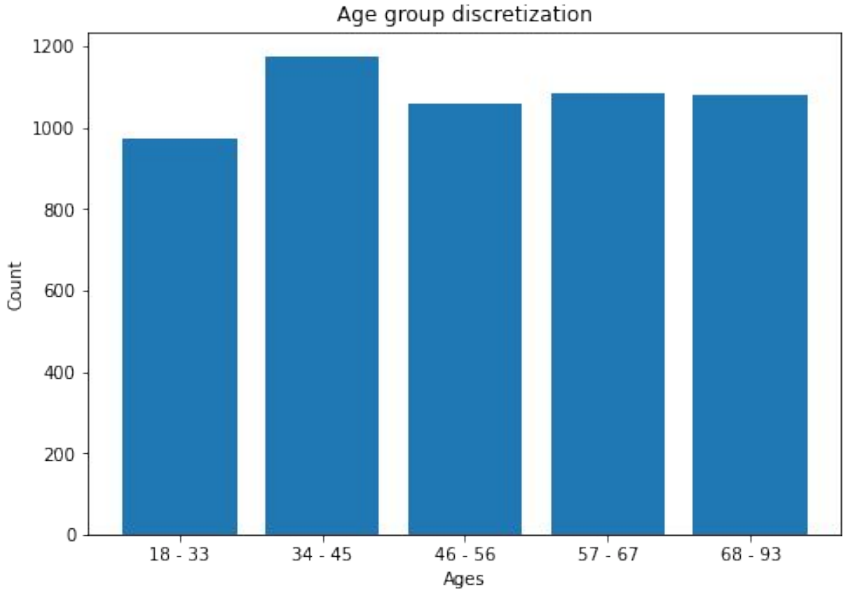
Imbalanced loan status class.



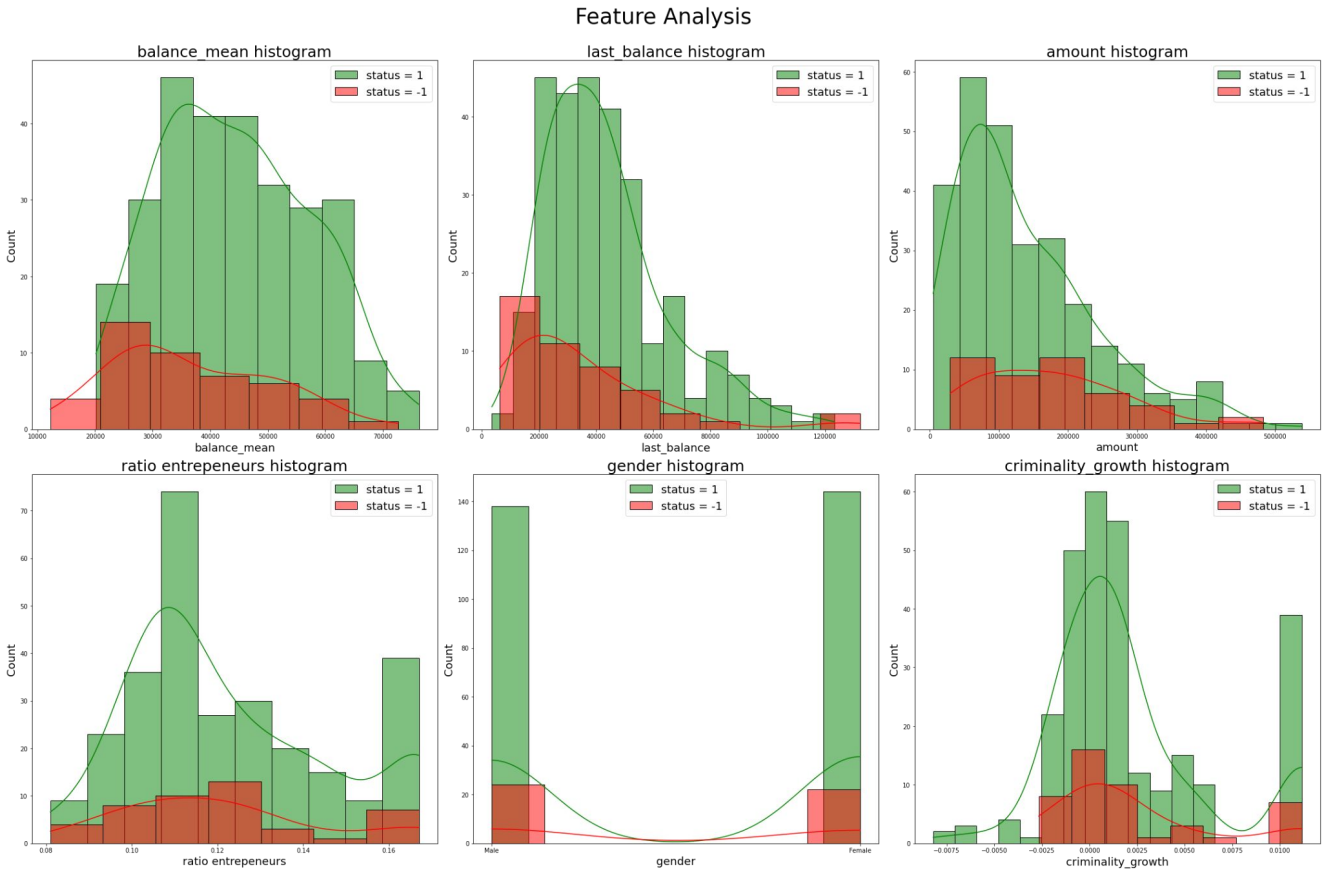
There is a correlation between payments value and the loan amount.

*loan amount = payments value * duration*





The majority of people asking for a loan were between 25 and 60 years old, as of 2003. On the right side, we can see the effect of the equal frequency age discretization.

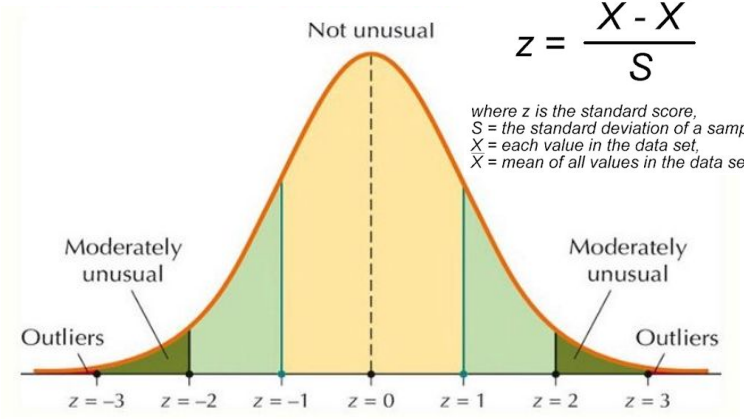


Clients with lower account balances are more prone to ask for loans.

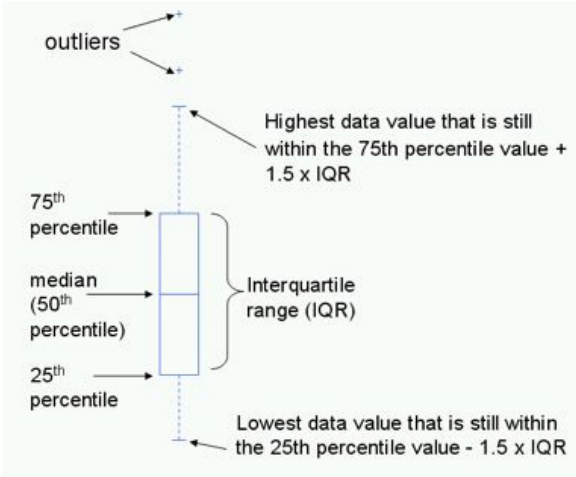
There are more loans with lower asked amounts.

The criminality growth in a district and the gender have no influence on the success of a loan.

Outliers



Z-Score



Box Plots

- Implemented Z-Score and Box Plots methods
- After the data understanding phase, we discovered that there are much fewer loans that failed. For that reason, the values of the columns of the failed loans can be considered outliers. Therefore, we didn't remove the outliers of the features that will be used in the machine learning models.

Feature Selection

SelectKBest (Filter method)

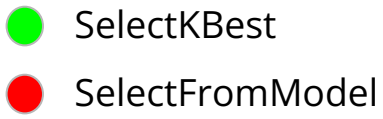
- Selects the k features with the highest score
- Uses the f_classif as the score function
    - Computes the ANOVA F-value.

SelectFromModel with RandomForestClassifier as estimator (Embbeded method)

- Random Forest is a bag of Decision Trees.
- Each node of the Decision Tree divides the dataset into 2 buckets.
    - Importance of each feature is derived from how “pure” each of the buckets is.

The selected features are almost identical, as we can see in the next slide.

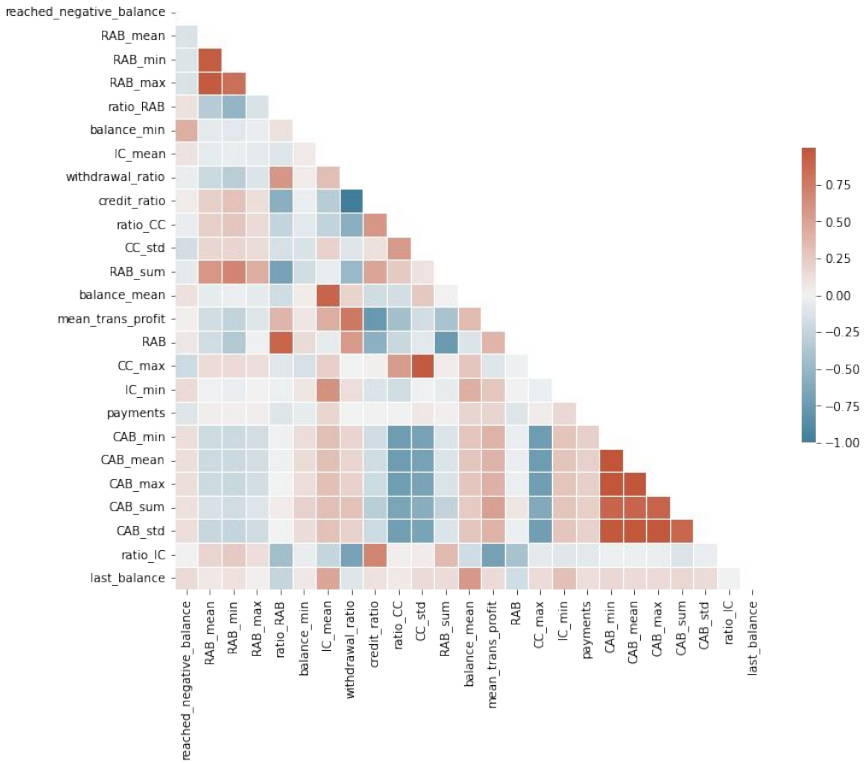
Selected Features



The selected features were the ones provided by SelectKBest algorithm. We tried using the SelectFromModel features with Random Forest but we got worse results and that was the reason for us to stick only with the SelectKBest features.

- reached_negative_balance
- RAB_mean
- ratio_RAB
- credit_ratio
- balance_min
- RAB_sum
- mean_trans_profit
- balance_mean
- ratio_CC
- RAB_min
- RAB_max
- ratio_IC
- CC_std
- IC_min
- CC_max
- ratio_IC
- withdrawal_max
- WC_max
- last_balance
- CAB_mean
- IC_mean
- withdrawal_ratio
- RAB
- IC_max
- payments
- CAB_min
- amount
- last_first_balance_ratio
- last_max_balance_ratio
- last_trans_amount
- balance_max
- CAB_max
- CAB_sum
- CAB_std

Redundancy

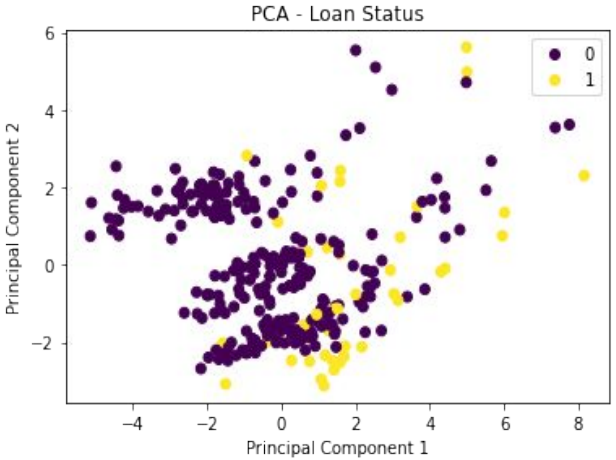


In this slide we check the correlation between the previously selected features from SelectKBest and the following attributes were dropped due to high correlation:

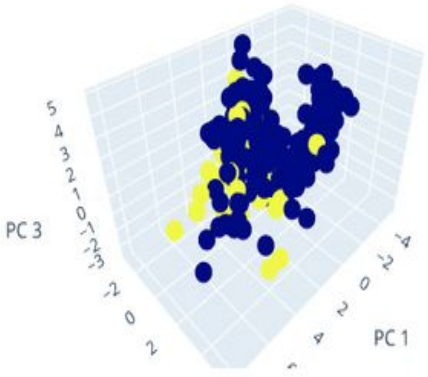
- RAB_min
- RAB_max
- RAB
- withdrawal_ratio
- IC_mean
- CAB_min
- CAB_max
- CAB_sum
- CAB_std

Descriptive Task - PCA

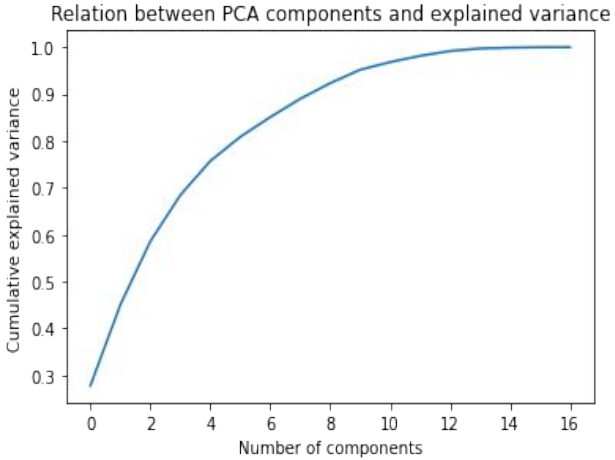
In a first approach, the main goal of the Descriptive Task was to visualize how data was distributed across different types of loans. PCA was the first technique used. All the methods explained serve as a timeline that shows the development of the Descriptive task. Note that loans that previously had a status of -1 now have a status of 1 and the ones that had a status of 1 now have a status of 0.



In this plot, 2 principal components were used and we can see there isn't a clear distinction between each type of loan.



Here, 3 principal components were used and we can start to see a border appear between each type of loan.



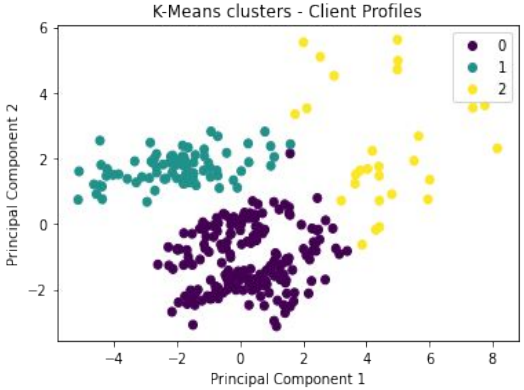
This plot shows the increase of the explained variance according to the number of components used. The explained variance evaluates the usefulness of principal components.

Descriptive Task - Clustering

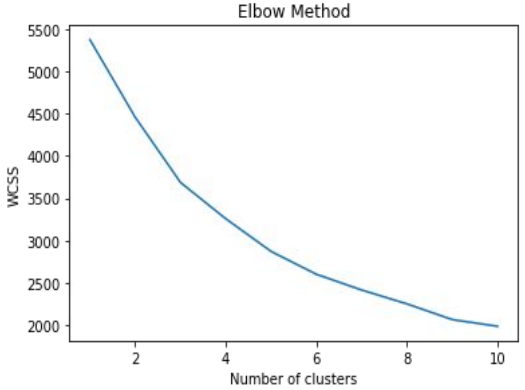
In the Descriptive task, we had to take into consideration the meaning of each cluster in the business context. In our case, the goal was to create clusters that aimed to discover the different types of clients. This was an iterative process and was composed of varying the number of clusters, adding and removing attributes that were taken into consideration in the process of clustering, and creating new attributes.

Algorithms used:

- K-Means
- Spectral Clustering
- DBScan



This plot shows the use of K-Means with 3 clusters, the clustering algorithm that gave us the best results. As the features passed to the K-Means are mostly related to the transactions data, we can fully differentiate each type of client. In this plot, we first use K-Means to obtain the clusters associated with each client and then we use PCA to reduce the dimension of our dataset into 2 dimensions and assign each data point to the respective cluster.



The Elbow method is a heuristic used to determine the number of clusters in a dataset. We relate the WCSS, the sum of the squared error (SSE), with the number of clusters needed. More specifically, the SSE is defined as the sum of the squared Euclidean distances of each point to its closest centroid. Since this is a measure of error, the objective of a clustering algorithm is to try to minimize this value. Therefore, we pick the right number of clusters based on the point where the next number of clusters has a small slope when diminishing the WCSS according to the number of clusters. In this case, although it's a bit subjective, we chose **3 clusters**.

Descriptive Task - Clustering

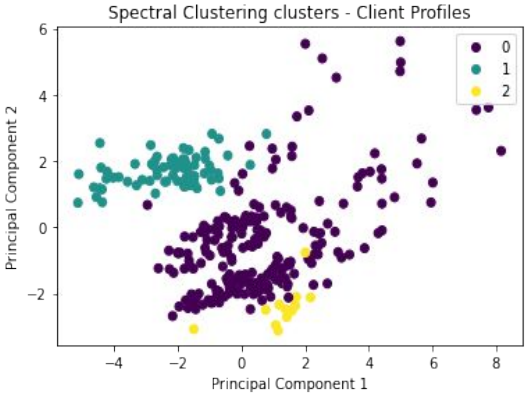
client profile	balance_min	balance_max	balance_mean	balance_std	ratio_IC	ratio_CC	ratio_CAB	ratio_WC	ratio_RAB	ratio_CCW
0	-3424.6	164648.6	42987.38	18403.16	0.184	0.262	0.001	0.449	0.103	0.001
1	200.0	139880.9	46453.87	17916.18	0.187	0.037	0.183	0.493	0.101	0.001
2	200.0	123310.1	37281.21	17166.44	0.337	0.327	0.084	0.248	0.004	0

With regards to the K-Means usage, in the above table, we present some statistics regarding each client profile namely the average minimum balance, maximum balance, mean balance, and respective standard deviation. Furthermore, there are also some statistics regarding the type of operations each client performs: Interest Credited (IC), Credit in Cash (CC), Collection from another Bank (CAB), Withdrawal in Cash (WC), Remittance to another Bank (RAB), and Credit Card Withdrawal (CCW). Clusters 1 and 2 are the biggest ones, as we can see from the above plot. They are all very different in terms of the distribution of transactions between each client. Cluster 2 seems to have a low ratio of Withdrawal in Cash when compared to the other clusters, but they have more frequent operations regarding Credit in Cash and Interests Credited, which means they probably handle their savings more efficiently. Regarding the balances, cluster number 1 works with higher balances according to the mean balance, and cluster 0 has clients that reached negative balances which can be a strong indicator of a division in different client profiles.

Descriptive Task - Clustering

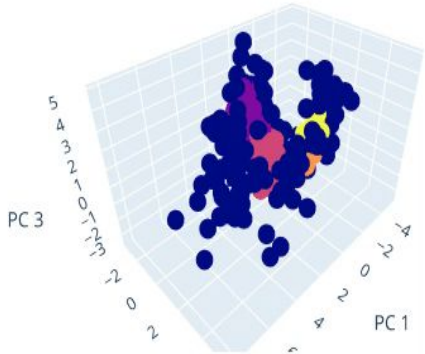
As a conclusion to the clustering phase, as mentioned, the K-Means algorithm gave us the best results. So we opted to use the generated clusters for each type of client in the predictive models. We also tried the creating clusters with the following algorithms:

Spectral Clustering



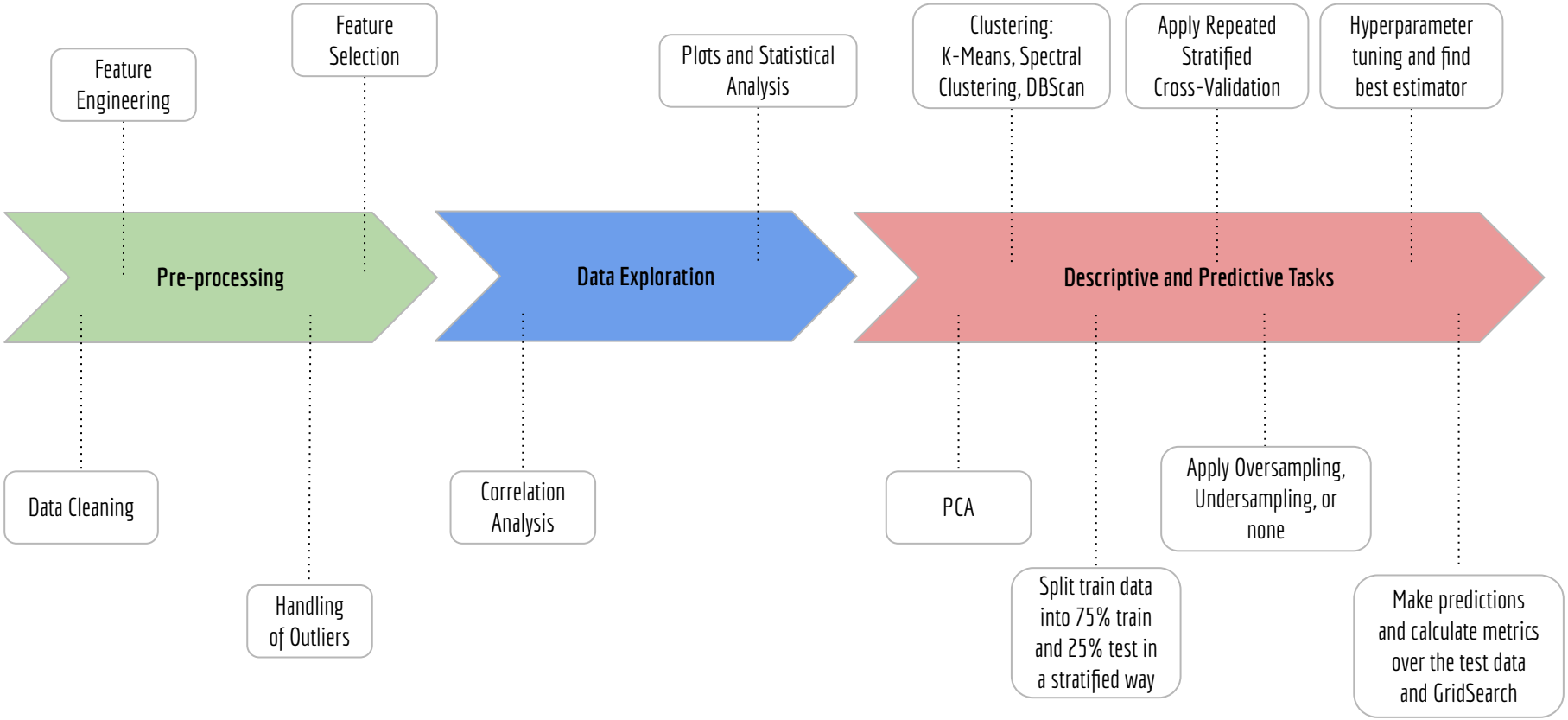
This plot shows the use of the Spectral Clustering algorithm with 3 clusters. The disposition of points is exactly the same as for the K-Means plot, but the division in cluster is not suitable for our problem, and for that reason, we can conclude that the Spectral Clustering algorithm is no the right choice here.

DBScan



By using the DBScan, as we can't specify the number of clusters we want, we used an epsilon of 0.65 and a minimum number of samples in a neighborhood for a point to be considered as a core point of 8. We obtain 4 different clusters and all the blue points are noise points, which means they are neither border nor core points.

Experimental Setup - Pipeline Visualization



Experimental Setup

The project as a pipeline has the following steps:

- Preprocessing (as seen in the previous slides)
  - **Feature Engineering, Feature Selection, handling outliers, etc.**
- Data Exploration
- Descriptive Task
  - **PCA, K-Means, Spectral Clustering, and DBScan**
- Predictive Task:
  - Creation of a pipeline with the steps needed for the algorithm execution:
    - **Scaling** with MinMaxScaling or Standard Scaler whenever needed.
    - Apply **undersampling** with RandomUnderSampler, SMOTE **oversampling** or **none.** Using imblearn's Pipelines, we guarantee that sampling is only applied to train data.
    - Apply the selected model.
  - Use **RepeatedStratifiedKFold** as a **Cross-Validation** technique.
  - **GridSearch** on the chosen algorithms for **hyperparameter tuning** - **Logistic Regression**, **Random Forest**, **SVM**, and **Light GBM -** find the best estimator based on the best AUC score.
  - Perform a **train_test_split** (75/25) in a **stratified** way to maintain the balance between train and test data.
  - Apply the **best estimator** with the **GridSearch's** best configurations and get predictions.
  - Calculate **metrics for the predictions** obtained from the **test data** with ROC Curves and Precision-Recall plots. We also calculate the accuracy, precision, recall, and F1 Measure associated.
  - Calculate **metrics for each GridSearch iteration** namely: **mean** fit time, accuracy, precision, recall, AUC, and F1 Measure.

Example

```
pipeline = Pipeline([('under',
RandomUnderSampler(random_state=0)), ('classification',
RandomForestClassifier(random_state=0))])

param_grid = {
'under__sampling_strategy': [0.2, 0.5, 0.75, 0.9],
'classification__n_estimators': [50, 150, 250],
'classification__max_features': ['sqrt', 'auto'],
'classification__criterion': ['gini', 'entropy'],
'classification__class_weight': ["balanced",
"balanced_subsample", None]
}


# Cross Validation, GridSearch => Best estimator =>
Predictions => Metrics
run_model()
```

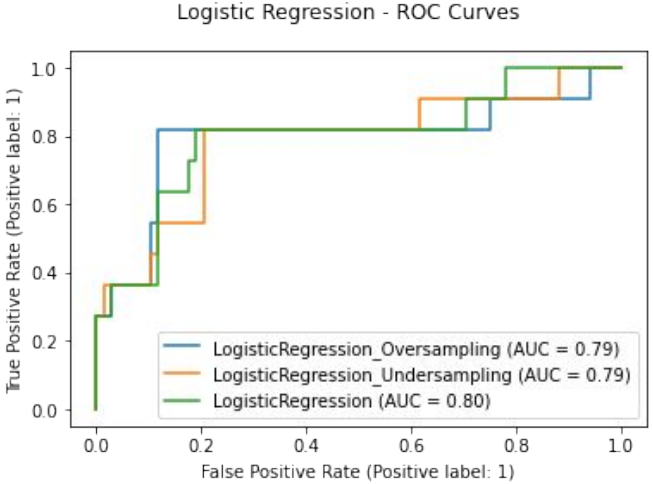
Experimental Setup - Running Models

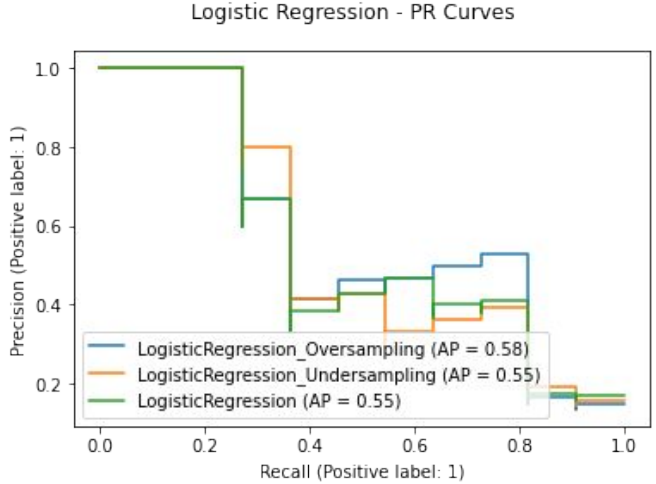
This function is responsible for receiving each pipeline and doing all the steps mentioned in the Experimental Setup section.

```
scoring = {
 "precision_score": make_scorer(precision_score, zero_division=0),
 "recall_score": "recall",
 "roc_auc_score": "roc_auc",
 "f1_score": "f1",
 "accuracy_score": "accuracy",
}

def run_model(pipeline, param_grid, X_train, y_train, X_test_pred, clfName, X_test=[], y_test=[]):
 global algorithms, metrics, plots_data
  # Define evaluation procedure
 cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

 grid = GridSearchCV(estimator=pipeline, param_grid=param_grid,
   scoring=scoring, n_jobs=-1, cv=cv, refit="roc_auc_score")

 grid_result = grid.fit(X_train, y_train)
 print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_))

 # Evaluate the model
 p_pred = grid_result.predict_proba(X_test_pred)

 # ROC Curve & PR
 estimator = grid_result.best_estimator_

 if SAMPLING:
   plots_data[clfName] = [estimator, X_test, y_test]
   RocCurveDisplay.from_estimator(estimator, X_test, y_test)
   PrecisionRecallDisplay.from_estimator(estimator, X_test, y_test)
 else:
   plots_data[clfName] = [estimator, X_train, y_train]
   RocCurveDisplay.from_estimator(estimator, X_train, y_train)
   PrecisionRecallDisplay.from_estimator(estimator, X_train, y_train)

# GridSearch Metrics
 mean_fit_time = mean(grid_result.cv_results_.get('mean_fit_time'))
 mean_accuracy_score= mean(grid_result.cv_results_.get('mean_test_accuracy_score'))
 mean_precision_score= mean(grid_result.cv_results_.get('mean_test_precision_score'))
 mean_recall_score= mean(grid_result.cv_results_.get('mean_test_recall_score'))
 mean_roc_auc_score= mean(grid_result.cv_results_.get('mean_test_roc_auc_score'))
 mean_f1_score = mean(grid_result.cv_results_.get('mean_test_f1_score'))

 algorithms.append(clfName)
 metrics[clfName] = {}
 metrics[clfName]["fit_time"] = mean_fit_time
 metrics[clfName]["accuracy"] = mean_accuracy_score
 metrics[clfName]["precision"] = mean_precision_score
 metrics[clfName]["recall"] = mean_recall_score
 metrics[clfName]["roc_auc"] = mean_roc_auc_score
 metrics[clfName]["f1"] = mean_f1_score
 # ...
 if SAMPLING: # Test Data Metrics
   predictions_for_metrics= grid_result.predict(X_test)
   test_accuracy = accuracy_score(y_test, predictions_for_metrics)
   test_precision = precision_score(y_test, predictions_for_metrics)
   test_recall = recall_score(y_test, predictions_for_metrics)
   test_f1 = f1_score(y_test, predictions_for_metrics)
   metrics[clfName]["test_accuracy"] = test_accuracy
   metrics[clfName]["test_precision"] = test_precision
   metrics[clfName]["test_recall"] = test_recall
   metrics[clfName]["test_f1"] = test_f1
```

Predictive Task - Logistic Regression

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In Logistic Regression, the dependent variable is binary. In other words, the Logistic Regression model predicts P(Y=1) as a function of X. It's a **fast algorithm** but **gave us bad results**, comparing to other algorithms.

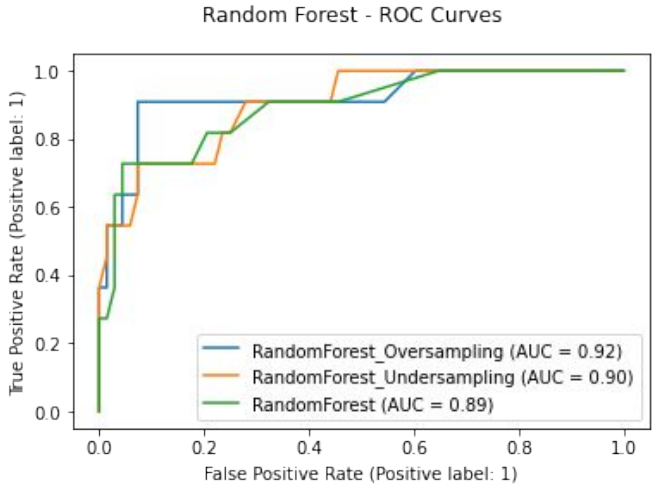


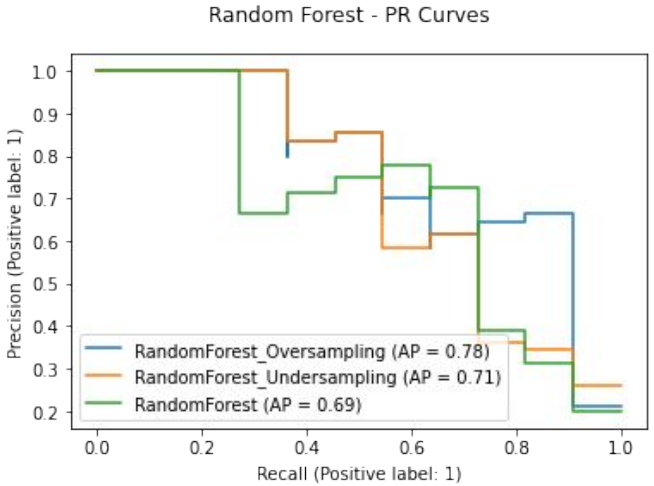


Above we can see the ROC Curves and Precision-Recall plots regarding the predictions against the test data and combined with the different variations of the algorithm. In this case, they are all very similar.

Predictive Task - Random Forest

Random Forest is a bag of Decision Trees meaning it's very good at preventing overfitting. It's a **very slow algorithm** as we will later see but it was the one who **gave us the best results**!

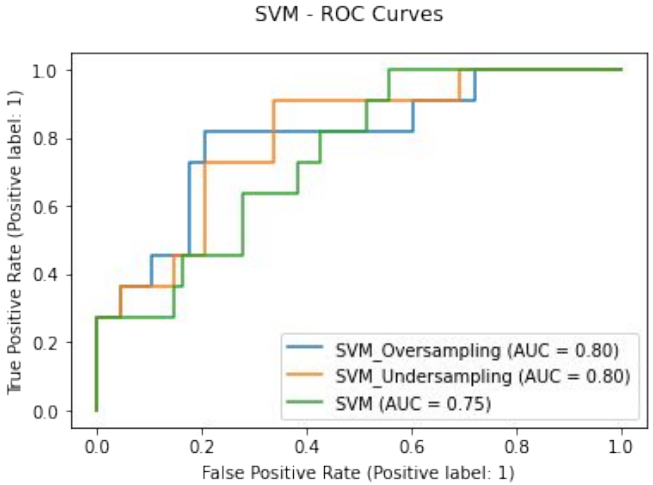


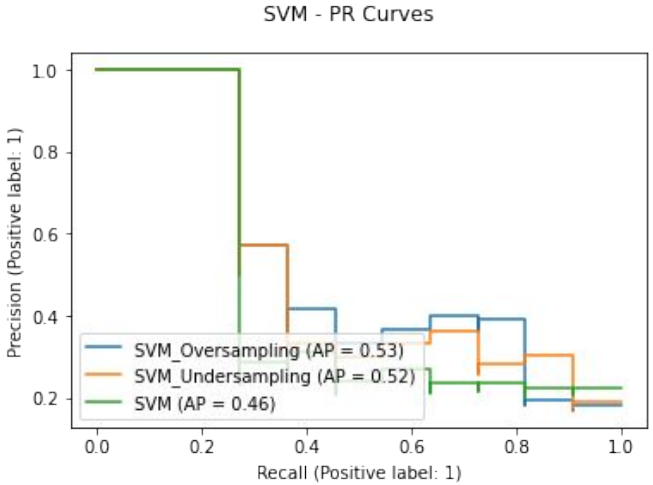


Above we can see the ROC Curves and Precision-Recall plots regarding the predictions against the test data and combined with the different variations of the algorithm. In this case, we select Random Forest with Oversampling as our best variation because it's the one with higher AUC and Average Precision (AP).

Predictive Task - SVM

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite-dimensional space. Along with Logistic Regression, it was our **worst algorithm**.

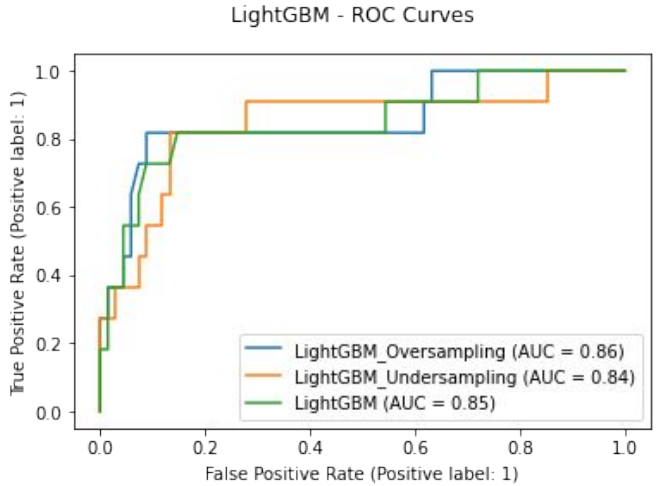


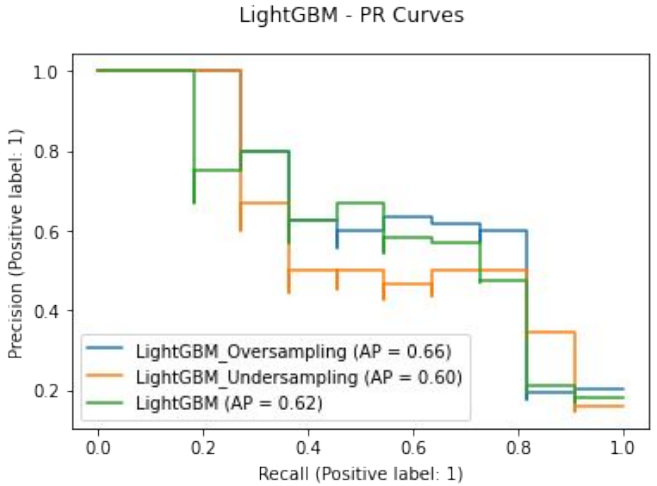


Above we can see the ROC Curves and Precision-Recall plots regarding the predictions against the test data and combined with the different variations of the algorithm. In this case, we select SVM with Oversampling as our best variation simply because it's slightly better than the one with Undersampling in the Precision-Recall plot.

Predictive Task - Light GBM

Light GBM is a gradient boosting framework that uses a tree-based learning algorithm. It's a light version of Gradient Boosting and was developed by Microsoft.

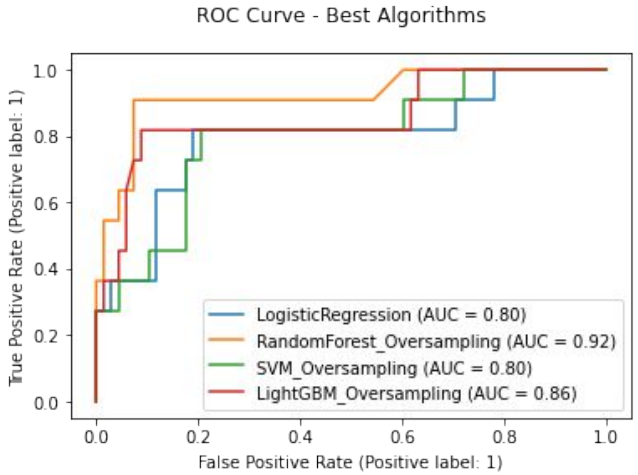


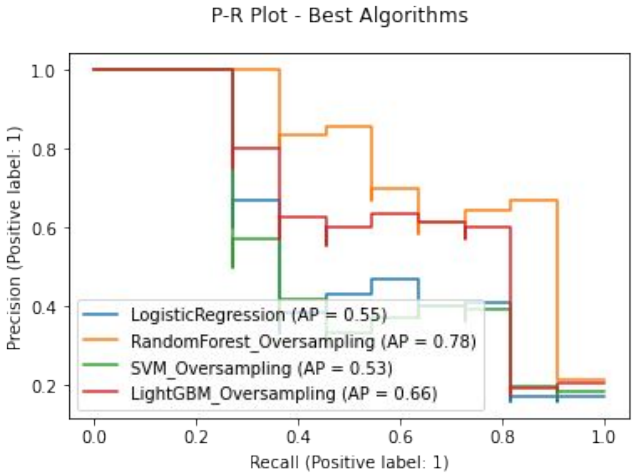


Above we can see the ROC Curves and Precision-Recall plots regarding the predictions against the test data and combined with the different variations of the algorithm. In this case, Light GBM with oversampling was our second best algorithm.

Predictive Task - Best Algorithms Comparison

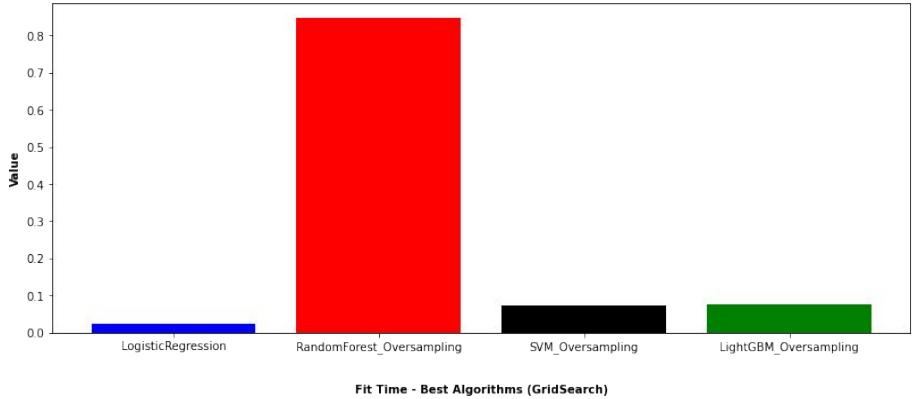
The selected best algorithms were Logistic Regression with neither Oversampling nor Undersampling, Random Forest with Oversampling, SVM with Oversampling, and Light GBM also with Oversampling. These results are based on the results obtained when testing the models against the test data. From the two plots below that compare the different ROC Curves and Precision-Recall relations between the aforementioned algorithms, we can firmly assert that our **best model** was **Random Forest with Oversampling** (AUC = 0.92, AP = 0.78) and the two models that gave us **poorer results** were **Logistic Regression** (AUC = 0.80, AP = 0.55) and **SVM with Oversampling** (AUC = 0.80, AP = 0.53).

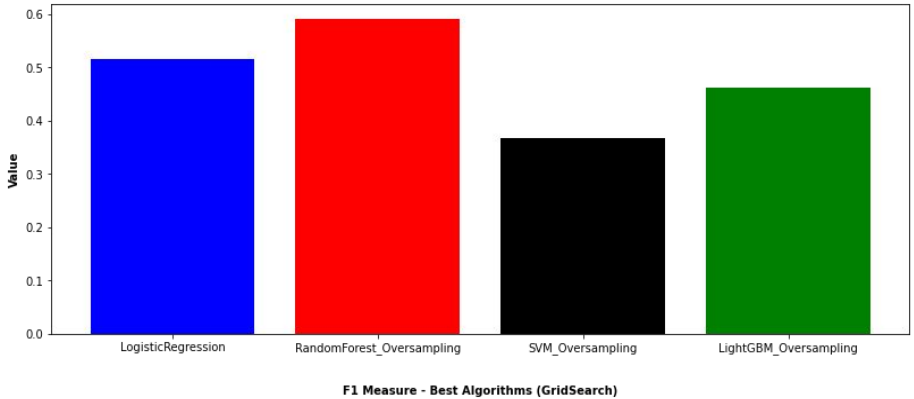




Predictive Task - Best Algorithms Comparison (GridSearch Metrics)

Regarding the metrics already mentioned over **GridSearch**, on the right, we can see a plot that compares the average time each GridSearch iteration takes (**Fit Time**) for all the selected best algorithms. Random Forest, as stated, takes approximately 0.8 seconds per iteration being by far the most performance costly algorithm. Logistic regression tends to be the fastest algorithm and the other two take approximately 0.1 seconds per iteration.

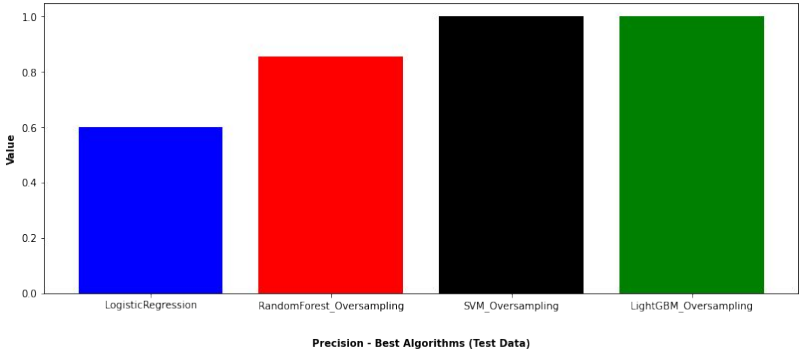




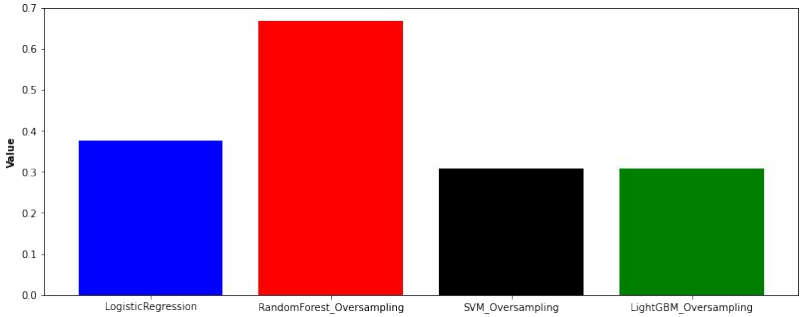
In the plot on the left we can see a comparison of the F1 measure, an important metric in our problem, also regarding the **GridSearch** metrics. As expected, Random Forest was the one who gave us the best metric while SVM had the lower value.

Predictive Task - Best Algorithms Comparison (Test Data Metrics)

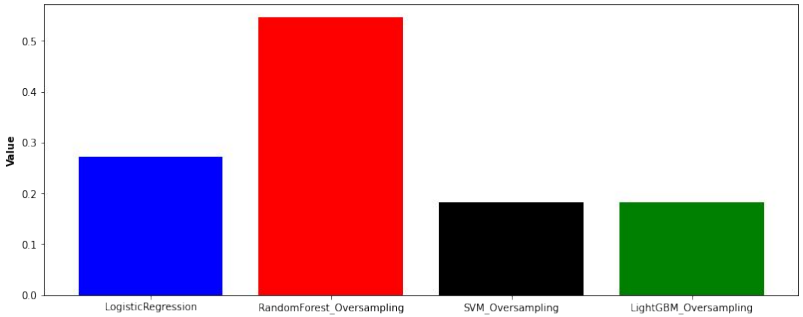
In the previous slide we discussed the metrics for the **GridSearch** iterations. They all were based in mean values. But it's also important to see what were the **metrics for our test data** and not taking into consideration only mean values! As we can see from the test data metrics in the F1 Measure plot the Random Forest clearly got better results of approximately 0.7. In the Recall plot we also see Random Forest with Oversampling completely destroying the competition. The same doesn't happen in the Precision plot where SVM and Light GBM, both with Oversampling had a very high score. Also, we see SVM and Light GBM having very close results in these plots which is something interesting.



Precision - Best Algorithms (Test Data)



F1 Measure - Best Algorithms (Test Data)



Recall - Best Algorithms (Test Data)

Conclusions, Limitations and Future Work

Conclusions:

- Feature Engineering and Feature Selection are crucial steps in the entire process.
- *Light GBM* had a bad performance, probably because of low amounts of data passed to it.
- Random Forest was the best algorithm!

Limitations:

- Small dataset.

Future Work:

- Improve features being used, by Feature Engineering and Feature Selection.

Individual Factor

Being 1 the total divided by the 3 elements of the group.

David R Ferreira - 0.27

Rui Pinto - 0.39

Tiago Gomes - 0.34

1 - Max Participation; 0 - No Participation.