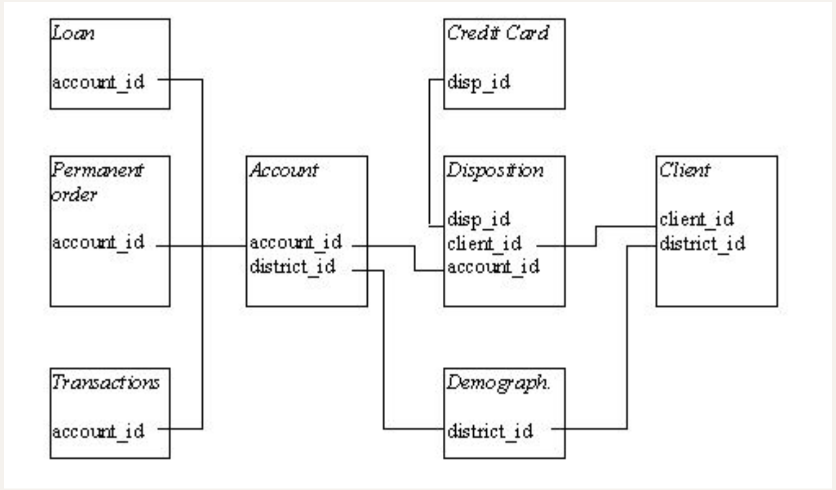
Data Mining Project

To loan or not to loan - that is the question

Group 21
Gustavo Sena - up201806078
Pedro Ferreira - up201806506
Pedro Ponte - up201809694

Domain Description

Our project focuses on an analysis of records from a bank in the Czech Republic. The dataset contains information about the bank clients, the accounts, transactions within several months, the loans already granted, the credit cards issued and geographical information.



- 4500 accounts
- 5369 clients
- 5369 dispositions
- 6471 permanent orders - missing table
- 1056320 transactions
- 682 loans (328 for train, 354 for test)
- 892 credit cards
- 77 demographic data

Some Considerations About Domain

- The objective of a bank when making loans is to make money through interest rates;
- Therefore, a good analysis of each client's data (economic situation, social environment, ...) is necessary to decide whether a loan should be granted or not;
- For a bank, in case of doubt whether a customer will pay or not, it is preferable not to grant the loan, so that the bank does not run the risk of losing money;
- In terms of our analysis of the results of the classification models, the number of false positives should be minimized, that is, the number of loans in which the customer is considered to pay, but ultimately does not pay, as these are the cases that make the bank lose money.

Exploratory Data Analysis

- The dataset is imbalanced: there are too many examples of successful loans (282) and few examples of unsuccessful loans (46) (*Appendix, Figure 1*);
- The relation between loan *amoun*t and *duration* is not very strong (*Appendix, Figure 2*);
- There is no relation between loan *duration* and *status* (*Appendix, Figure 3*);
- There is no relation between *age*, *genre* and loan *amount* (*Appendix, Figure 4*);
- The loan *amount* is equal to the product of loan *duration* by loan *payment*;
- There is no relation between *genre* and loan *status* (*Appendix, Figure 5*);
- There is a strong relation between unemployment rate in 95 and 96 is very strong (0.9585 of correlation coefficient)(*Appendix, Figure 6*);
- Clients *genre* distribution is balanced (2724 male vs 2645 female)(*Appendix, Figure 7*);
- All null values in the *operation* column in transactions have ‘interested credited’ value in *k_symbol* column (*Appendix, Figure 8*);

Exploratory Data Analysis (continue)

- Looking at transfers per month, we verify that December and January are the months with more transfers in each year (*Appendix, Figure 9*);
- There is no relation between loan *status* and *balance* (*Appendix, Figure 10*);
- The card *type* with more use is the Classic one (*Appendix, Figure 11*);
- All clients that have a card have a successful loan (*Appendix, Figure 12*);
- Only 11 clients that have asked for a loan have a card (*Appendix, Figure 12*);
- The age distribution when the loan is asked demonstrates an expected pattern: there are more loan requests when people are younger (*Appendix, Figure 13*);
- Each client only has a loan;
- Each client only has an account;
- We have identified some outliers in our data. However, when we try to remove them, the results when we apply the classification models were worse, so we opt to not remove them (*Appendix, Figure 14 and 15*).

Problem Definition

The objective of our problem is to train and test a Machine Learning predictive model that can predict if a new loan will be paid by the client or not. To train the model, the bank gives us a set of past loans and clients information (accounts, transactions, credit cards issued and geographical information).
So, the main question that our model tries to answer is:

To loan or not to loan?

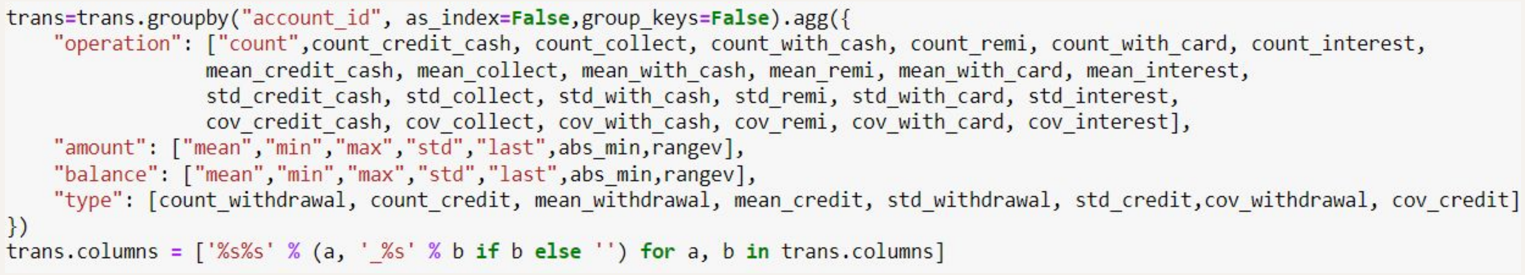
Data Preparation

- We want to predict the probability of the *status* of a loan be equal to -1, but in initial dataset -1 represent an unsuccessful loan, so we have to replace the -1 by 1 and 1 by -1 in this column;
- District with *id* 69 has 2 null values: *unemployment rate '95* filled with median value (*Appendix, Figure 16*) and *no. of committed crimes \'95* filled with mean (*Appendix, Figure 17*);
- Since only clients with *type* OWNER can issue permanent orders and ask for a loan, we opt to create a new column with the number of disponents for each account (*type_count_disponents*) and then delete all clients with *type* DISPONENT (*Appendix, Figure 18*);
- Conversion of dates to “YY-MM-DD” format;
- Create a column to distinguish clients by genre, based on fact that women *birth_number* is initially in format YYMM+50DD (*Appendix, Figure 19*);
- Based on dates, calculate the age of each client when he creates the account and the age of the client when he asks for a loan. Then, create age ranges to insert these values (10-20, 20-30, 30-40, ...) (*Appendix, Figure 20*);

Data Preparation (continue)

- Convert all transactions with *type* ‘withdrawal in cash’ in ‘withdrawal’ *type* (*Appendix, Figure 21*);
- As all null values in the *operation* column have ‘interested credited’ value in *k_symbol* column, we replace the null value in *operation* by the respective value in the *k_symbol* and remove the *k_symbol* column;
- Remove all transactions that occur after the client asked for a loan;
- Remove *bank*, *account* and *date* attributes from transactions dataset;
- Create column *hasCard* that identifies if a client a card or not (*Appendix, Figure 22*);
- Drop *date_account*, *birth_number*, *age_loan*, *age_account*, *issued*, *type* and *amount* attributes;
- Drop all id attributes;

Data Preparation (continue)



```
trans=trans.groupby("account_id", as_index=False,group_keys=False).agg({
    "operation": ["count",count_credit_cash, count_collect, count_with_cash, count_remi, count_with_card, count_interest,
                  mean_credit_cash, mean_collect, mean_with_cash, mean_remi, mean_with_card, mean_interest,
                  std_credit_cash, std_collect, std_with_cash, std_remi, std_with_card, std_interest,
                  cov_credit_cash, cov_collect, cov_with_cash, cov_remi, cov_with_card, cov_interest],
    "amount": ["mean","min","max","std","last",abs_min,rangev],
    "balance": ["mean","min","max","std","last",abs_min,rangev],
    "type": [count_withdrawal, count_credit, mean_withdrawal, mean_credit, std_withdrawal, std_credit,cov_withdrawal, cov_credit]
})
trans.columns = ['%s%s' % (a, '_%s' % b if b else '') for a, b in trans.columns]
```

- From transactions data, generate some new relevant features that proved to be important;
- Generate dummies for categorical features, like *frequency*, *age_loan_range*, *age_account_range)* (*Appendix, Figure 23*);
- Tools used:
    - Excel;
    - Jupyter Notebooks (Python w/ libs).
- In total, we have 81 features.

Clusters

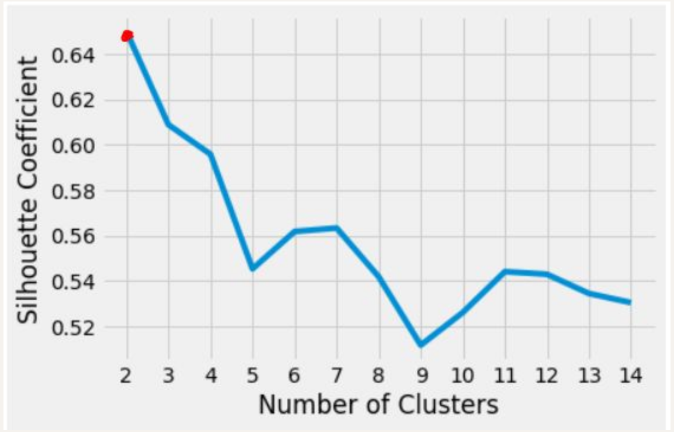
So we could aggregate clients based on certain characteristics they had in common, we resorted to the use of clusters. To achieve this we used 3 different algorithms:

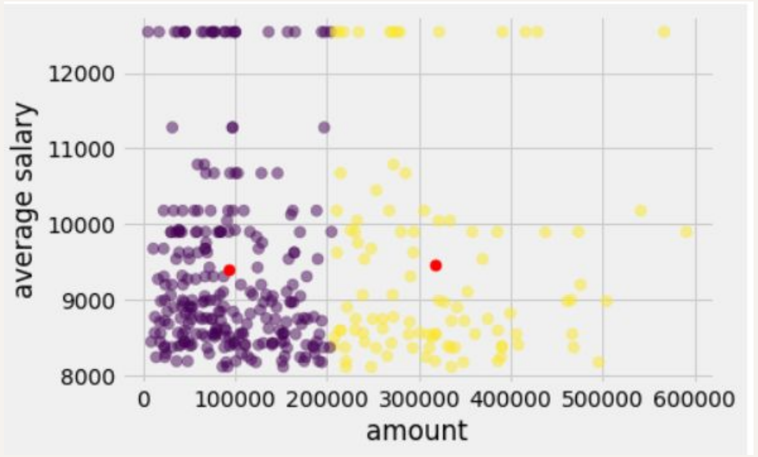
- **KMEANS** - It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand;
- **DBSCAN** (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) - It's a non-parametric density-based clustering technique that takes a set of points in space and clusters the closely packed together ones, labelling points that are alone in low-density regions as outliers;
- **Agglomerative** - This algorithm is one of the two types of hierarchical clustering, being the other the Divisive algorithm. Each observation begins in its own cluster, and as one progresses up the hierarchy, pairs of clusters are merged. The difference for the divisive is the last splits are performed recursively as one moves down the hierarchy.

In the appendix, other examples of columns chosen to examine, other than the ones used in the following slides, are shown.

KMEANS Clustering - Silhouette Evaluation

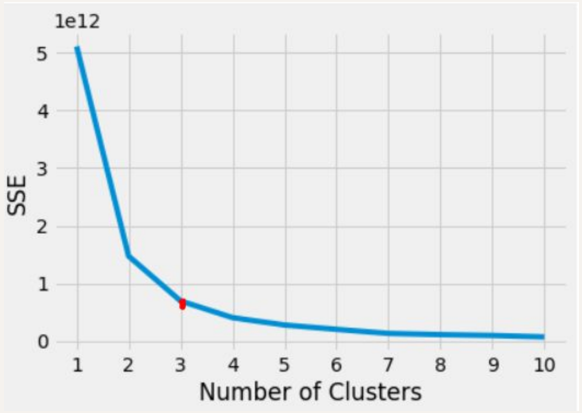
- We chose various columns to try and make clusters
- To get the optimal number we used the silhouette score, where it corresponds to where the the lowest SSE (sum of the squared error) value and highest silhouette coefficient is. In this case 2.
- This values correspond to the combination of the Amount and Average Salary
- Some other examples can be seen in *Appendix, Figures 24 and 30*.

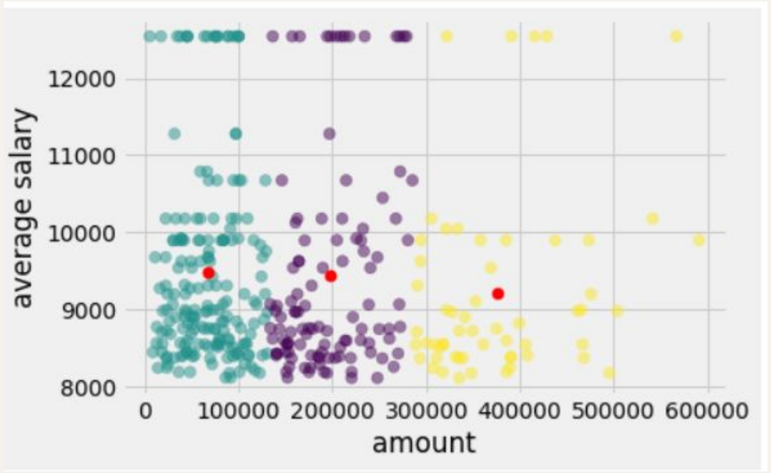




KMEANS Clustering - Elbow Evaluation

- In the elbow evaluation, the point of inflexion on the curve is the value of the optimal number of clusters. To know this value we used KneeLocator function, resulting in the number 3.
- This values also correspond to the combination of the Amount and Average Salary.
- Some other examples can be seen in *Appendix, Figures 25 and 30 (in the last, same value as in the silhouette evaluation).*

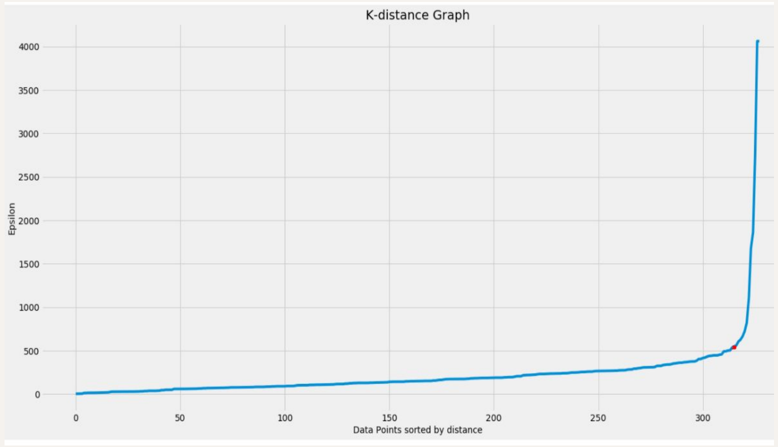


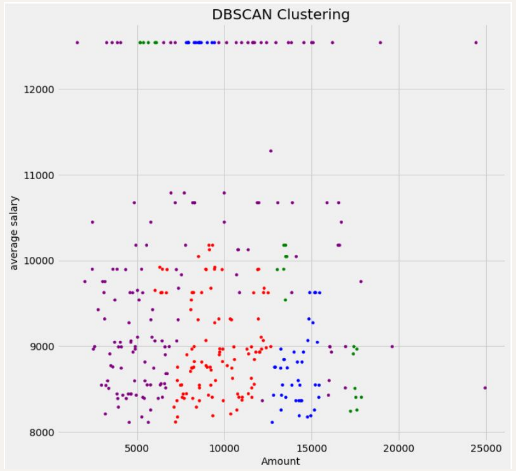


DBSCAN Clustering - K-distance

For the DBSCAN algorithm and to know the optimal number of epsilon we used the K-distance technique.

- The maximum curvature point (elbow) in this graph indicates the value of epsilon. If the epsilon value used is too little, more clusters will be formed, and more data points will be considered noise. If the size is too large, the little clusters will merge into a large cluster, and the details will be lost.
- Some other examples can be seen in *Appendix, Figure 26*.

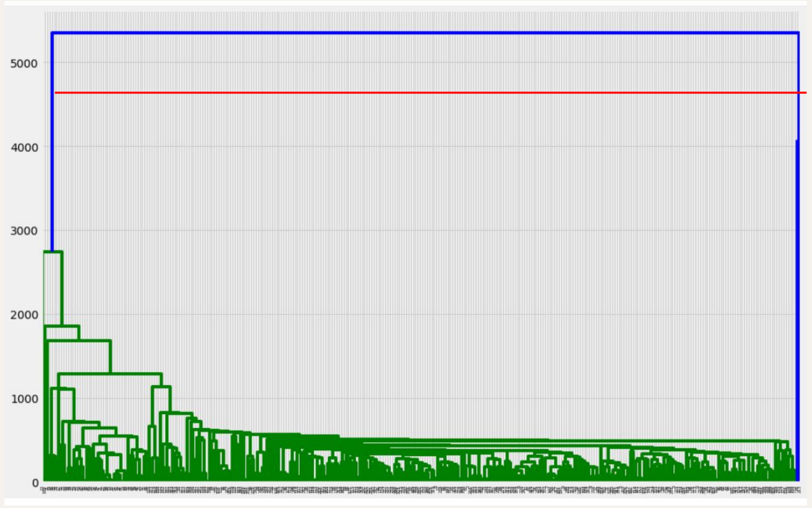


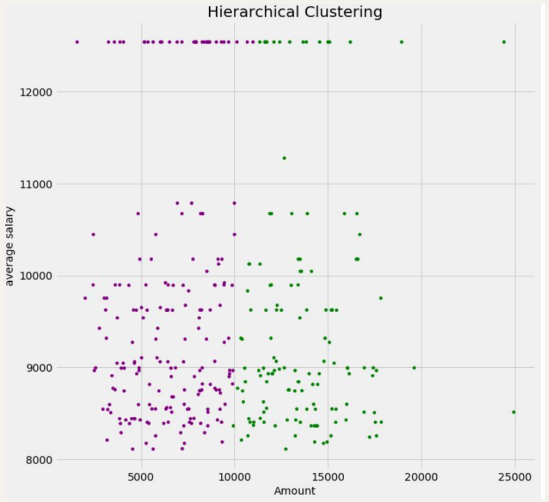


Agglomerative Clustering - Dendogram Eval.

For the DBSCAN algorithm and to know the optimal number of epsilon we used the dendogram technique (bottom left image).

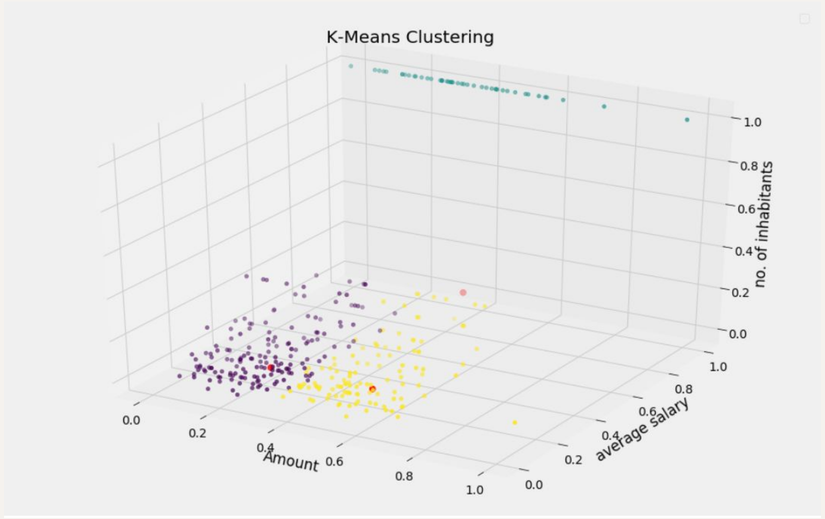
- Once one big cluster is formed, the longest vertical distance without any horizontal line passing through it is selected and the number of clusters is equal to the number of vertical lines that a horizontal line goes through. In this case 2.
- Some other examples can be seen in *Appendix, Figures 27, 28 and 29*.





KMEANS 3D Clustering

So we could also compare three different columns to better organize the dataset in agglomerations, we used Kmeans again, but in a 3 dimensional graph. In this case we added the number of inhabitants to the cluster:



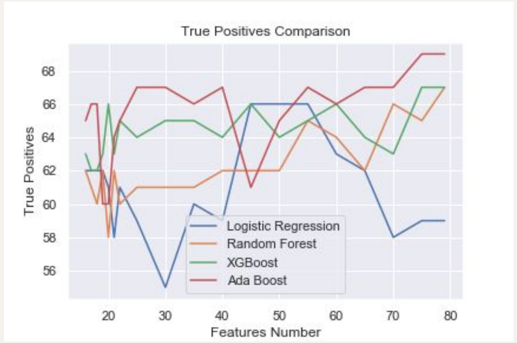
Experimental Setup

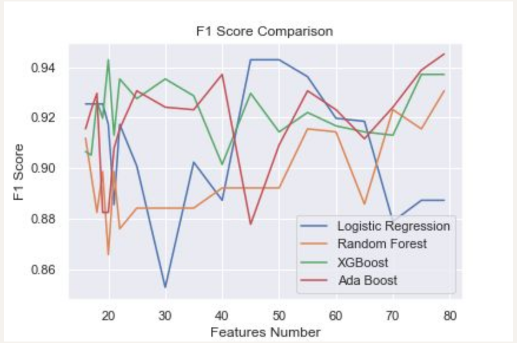
- **Feature Selection:**
  - using SelectKBest function of sklearn.feature_selection library;
  - score_func = f_classif;
  - variable number of selected features for experiments.
- **Split the prepared dataset into train and test subsets:**
  - using train_test_split function of sklearn.model_selection library;
  - test_size = 0.25.
- **Apply SMOTE to achieve class balance:**
  - there is a disparity between the number of successful and unsuccessful loans in train data;
  - Random_state = 42.

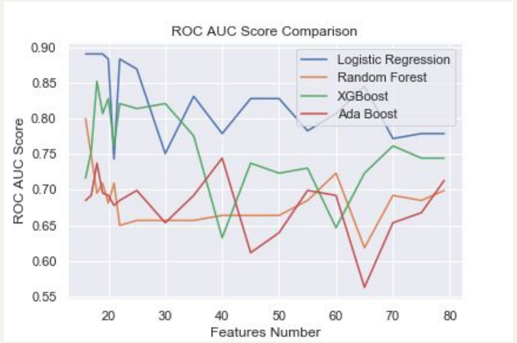
Results

- Implemented algorithms:
    - Decision Tree;
    - K-Nearest Neighbors;
    - Support Vector Machine;
    - Neural Networks;
    - Logistic Regression;
    - Gaussian Naive Bayes;
    - Random Forest;
    - XGBoost;
    - AdaBoost.
- In all algorithms, there is a significant improvement when we tuned some parameters;
- The best results were achieved using Logistic Regression, XGBoost and AdaBoost;
- To compare the algorithms, we essentially use 2 parameters:
    - ROC-AUC, which is the fraction of the negatively labeled sample that is correctly labeled - not looking at the fraction of your positively labeled samples that is correctly labeled;
    - F1-Score, which is the fraction of the positively labeled sample that is correctly labeled - don't consider the purity of the sample labeled as negative.
- In our case, we take more attention to F1-Score, because the lower this score value is, the more false positives there will be, that is, the worse it will be for the bank.

Best Results







- Looking at F1-Score, we see that XGBoost and AdaBoost tend to be the more consistent algorithms taking into account the number of features;
- In its turn, if we look for ROC-AUC score, the best algorithms tend to be Logistic Regression and XGBoost;

- Comparing F1-Score and ROC-AUC score evolution, we conclude that as the number of features increase, the positives correctly labelled tend to increase (except for Logistic Regression), but the correctly classified negatives tend to decrease;
- As we are more interested in knowing how many positives are correctly classified, we consider that XGBoost and AdaBoost are the best algorithms.

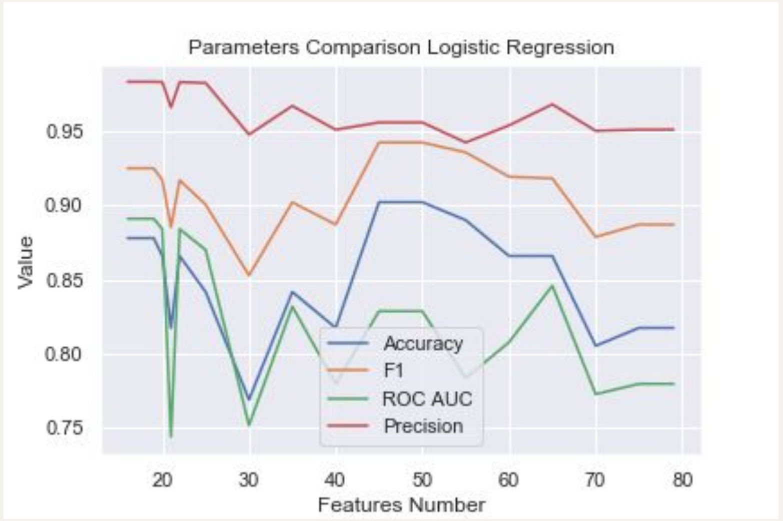
Results - Logistic Regression

Tuning:

```
lr_classifier = LogisticRegression()

    param_grid = {'penalty': ['l1', 'l2', 'elasticnet', 'none'],
                  'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
                  'max_iter': [20, 50, 100, 200, 500, 1000],
                  'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
                 }

    lr_grid_search = GridSearchCV(lr_classifier,
scoring="roc_auc", cv=5, param_grid={}, refit = True)
```



Some confusion matrices can be seen in *Appendix, Figures 31, 32, 33 and 34*.

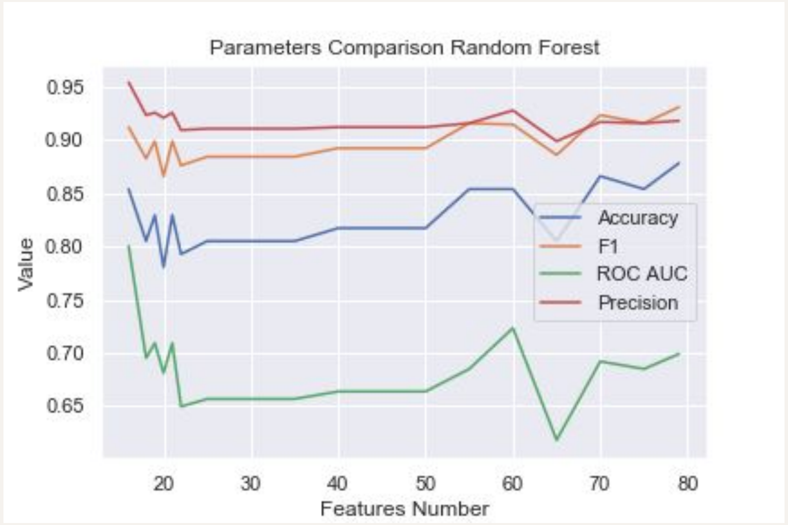
Results - Random Forest

Tuning:

```
rf_classifier = RandomForestClassifier()

    param_grid = {'bootstrap': [True],
                'max_depth': [80, 90, 100, 110],
                'max_features': [2, 3],
                'min_samples_leaf': [3, 4, 5],
                'min_samples_split': [8, 10, 12]
                }

    rf_grid_search = GridSearchCV(rf_classifier,
scoring="roc_auc", cv=5, param_grid=param_grid)
```



Some confusion matrices can be seen in *Appendix, Figures 35, 36, 37 and 38.*

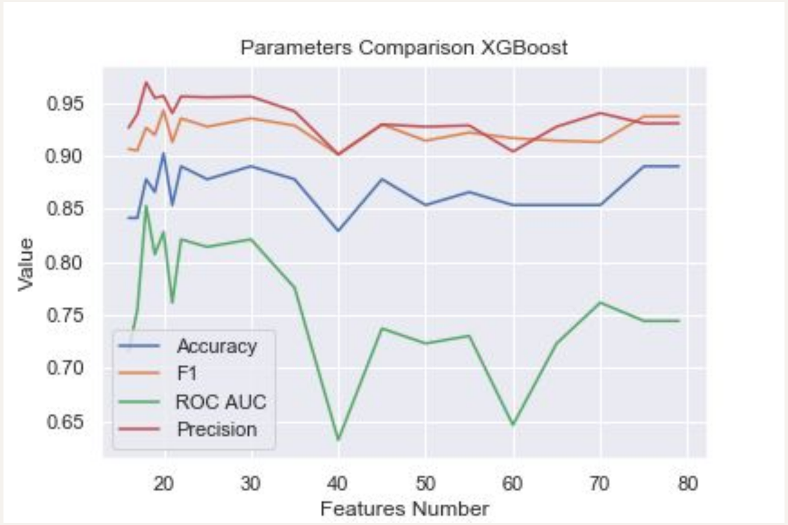
Results - XGBoost

Tuning:

```
xgb_classifier = XGBClassifier(metrics='rmse')

   param_grid = {'max-depth': [3, 6, 10],
              'n_estimators': [100, 300, 500, 1000],
              'learning_rate': [0.01, 0.05, 0.1],
              'colsample_bytree': [0.3, 0.7]
             }

   xgb_grid_search = GridSearchCV(xgb_classifier,
scoring='roc_auc', cv=5, param_grid=param_grid)
```



Some confusion matrices can be seen in *Appendix, Figures 39, 40, 41 and 42*.

Results - AdaBoost

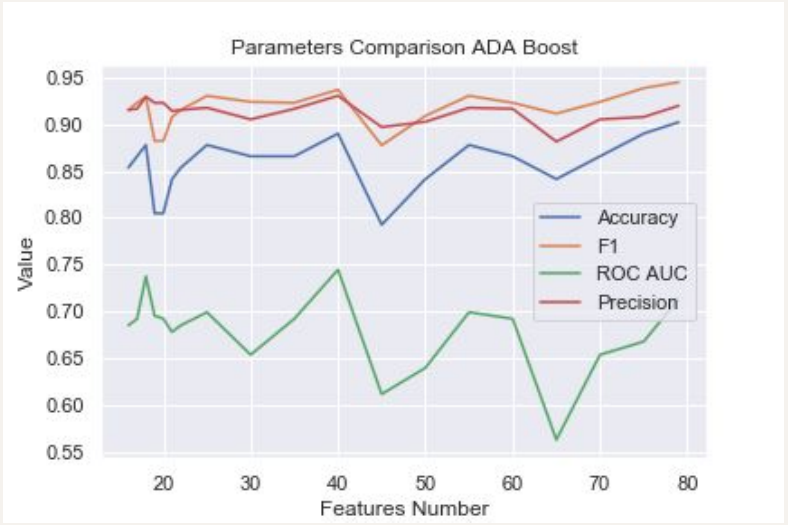
Tuning:

```
param_grid = {'base_estimator__max_depth':[i for i in range(2,11,2)],
          'base_estimator__min_samples_leaf':[5,10],
          'n_estimators':[10,50,250,1000],
          'learning_rate':[0.01,0.1]}


   DTC = DecisionTreeClassifier(random_state = 11, max_features = "auto", class_weight = "balanced",max_depth = None)

   ada_classifier = AdaBoostClassifier(base_estimator = DTC)

   # run grid search
   ada_grid_search = GridSearchCV(ada_classifier, param_grid=param_grid, scoring = 'roc_auc')
```



Some confusion matrices can be seen in *Appendix, Figures 43, 44, 45 and 46.*

Conclusions and Future Work

- Train data has a very small size and is imbalanced, which is bad for training classification models;
- Feature Selection is a crucial step, as it helps us to have a big improvement in results;
- Oversampling with SMOTE also helps to improve the score;
- In the future, further study the SVM, KNN and NB algorithms, to see how we can tune them better, so the results can be improved;
- Train subsets were transformed to only have numerical values. In the future, we could try to use numerical and categorical values in algorithms that allow it.

Appendix

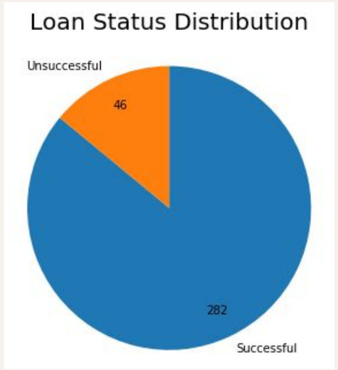
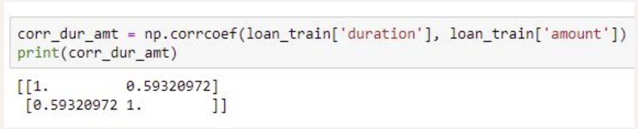


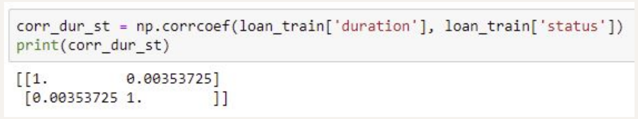
Figure 1 - Loan Status Distribution



```
corr_dur_amt = np.corrcoef(loan_train['duration'], loan_train['amount'])
print(corr_dur_amt)
```

```
[[1.         0.59320972]
 [0.59320972 1.        ]]
```

Figure 2 - Correlation between loan duration and amount



```
corr_dur_st = np.corrcoef(loan_train['duration'], loan_train['status'])
print(corr_dur_st)
```

```
[[1.         0.00353725]
 [0.00353725 1.        ]]
```

Figure 3 - Correlation between loan duration and status

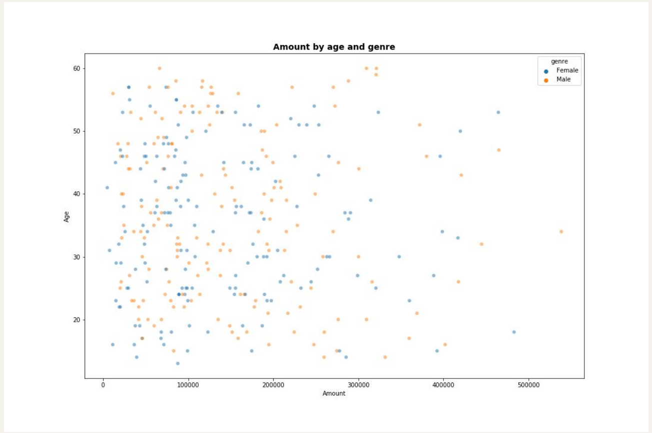


Figure 4 - Loan amount distribution by age and genre

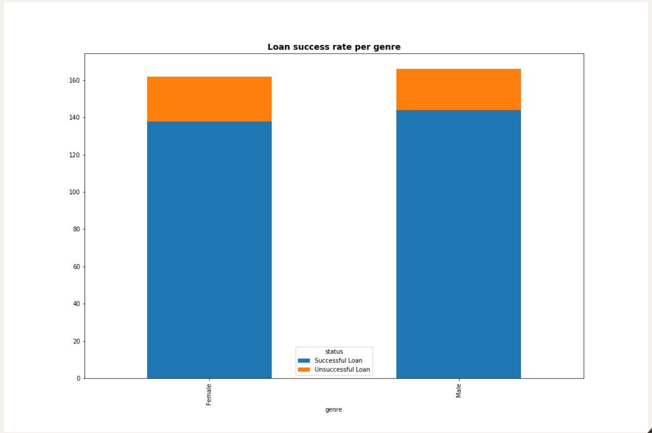


Figure 5 - Loan status distribution by genre

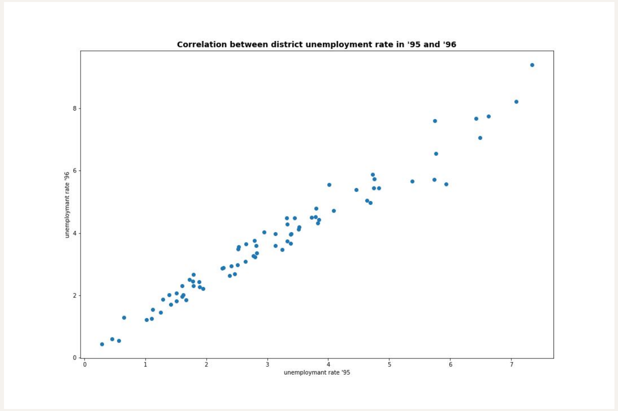


Figure 6 - Correlation between unemployment '95 and '96

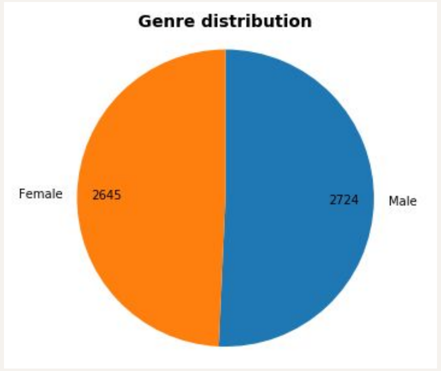


Figure 7 - Genre distribution

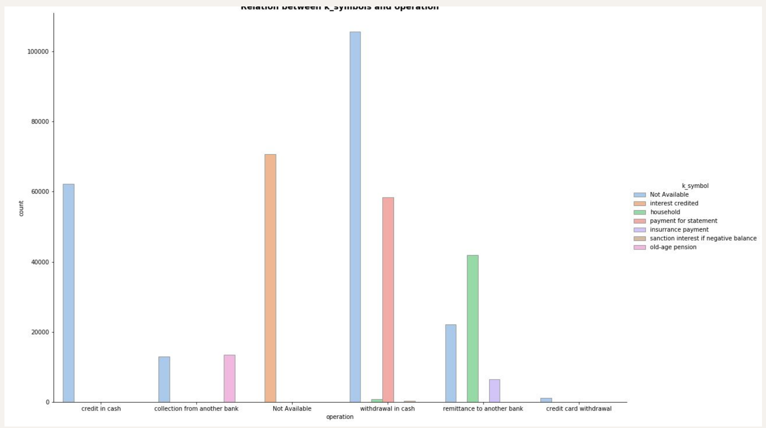


Figure 8 - Relation between operation and k_symbol

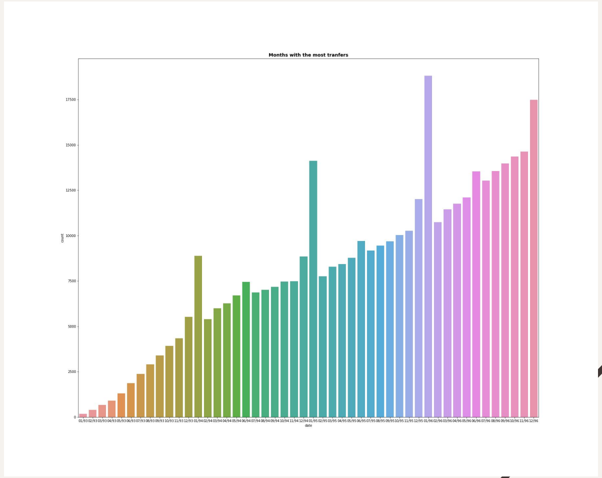


Figure 9 - Transactions distribution by month

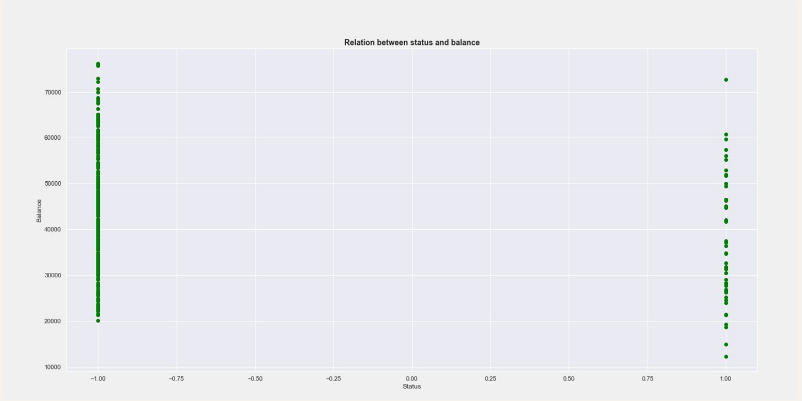


Figure 10 - Relation between status and balance

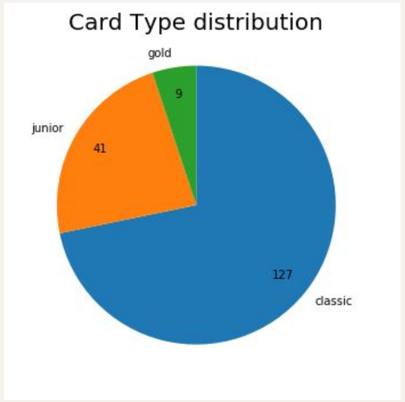


Figure 11 - Card type distribution

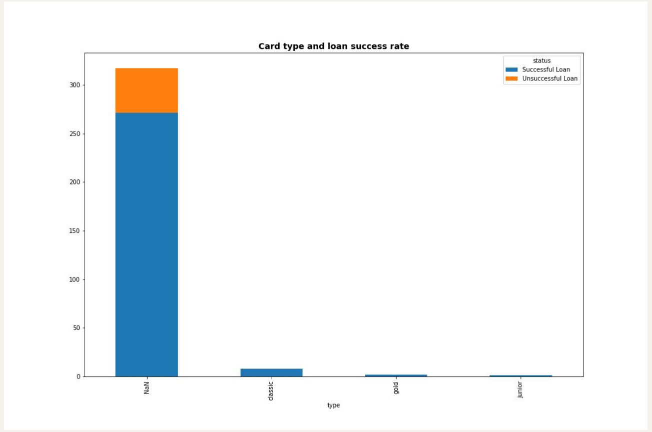


Figure 12 - Relation between card type and loan success

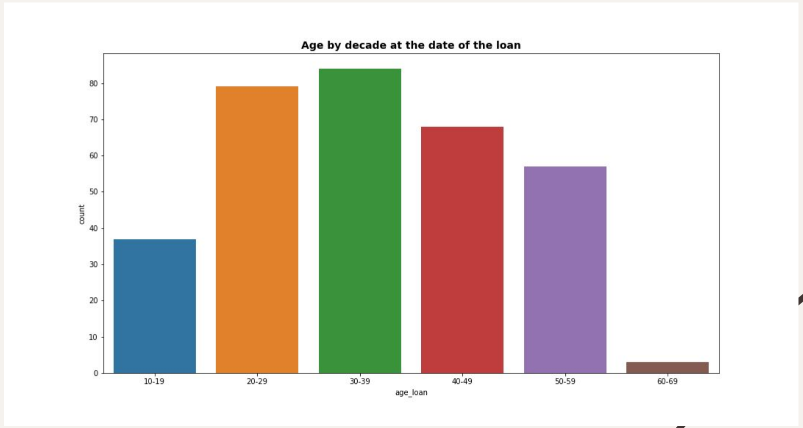


Figure 13 - Age at loan distribution

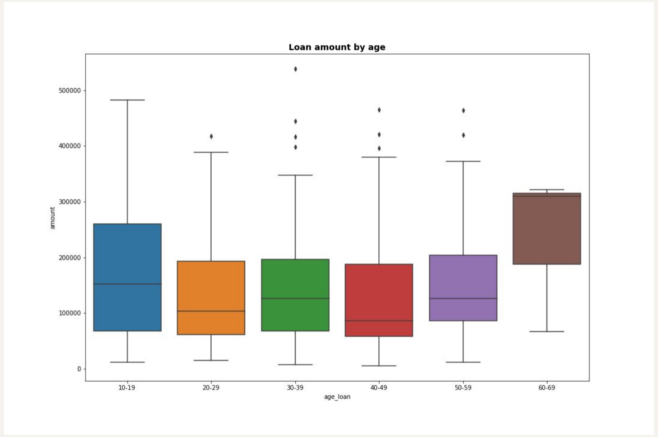


Figure 14 - Loan amount distribution by age at loan

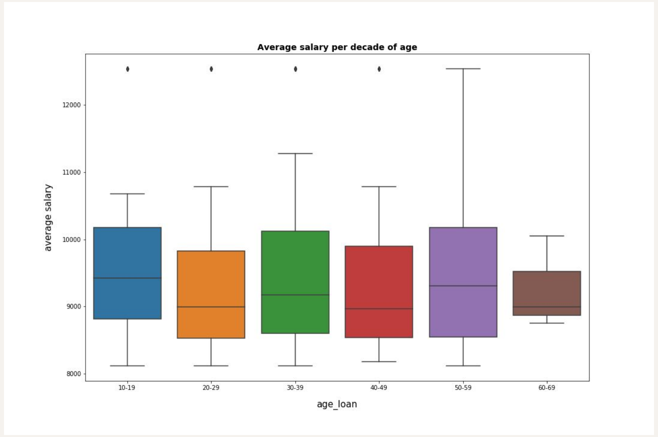


Figure 15 - Average salary distribution by age at loan

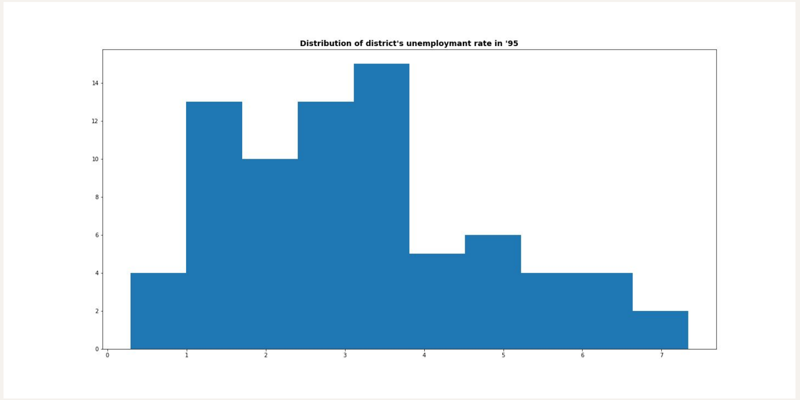


Figure 16 - District unemployment rate in ‘95 distribution

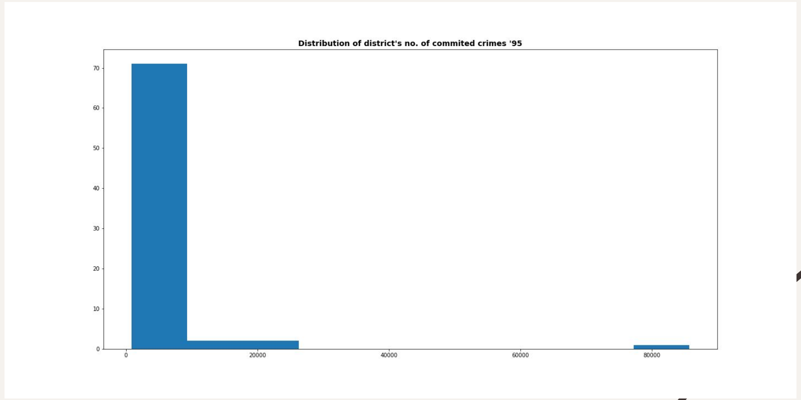
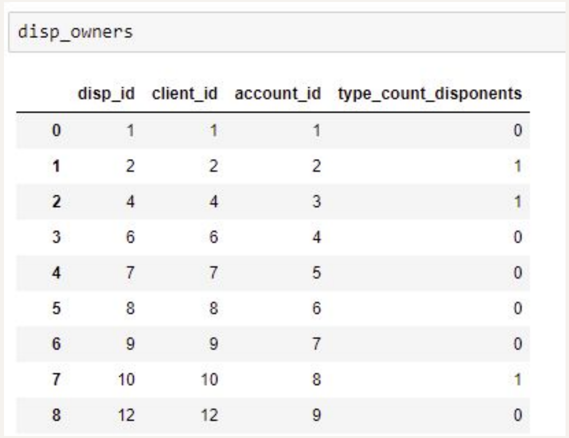


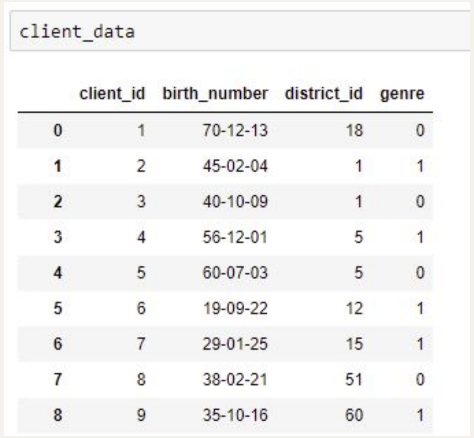
Figure 17 - District unemployment rate in ‘96 distribution



```
disp_owners
```

	disp_id	client_id	account_id	type_count_disponents
0	1	1	1	0
1	2	2	2	1
2	4	4	3	1
3	6	6	4	0
4	7	7	5	0
5	8	8	6	0
6	9	9	7	0
7	10	10	8	1
8	12	12	9	0

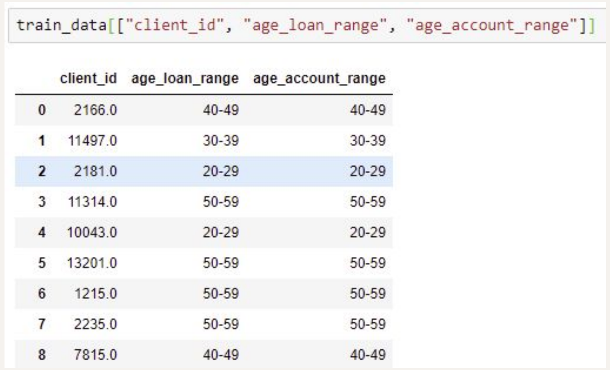
Figure 18 - Disp table



```
client_data
```

	client_id	birth_number	district_id	genre
0	1	70-12-13	18	0
1	2	45-02-04	1	1
2	3	40-10-09	1	0
3	4	56-12-01	5	1
4	5	60-07-03	5	0
5	6	19-09-22	12	1
6	7	29-01-25	15	1
7	8	38-02-21	51	0
8	9	35-10-16	60	1

Figure 19 - Client table



```
train_data[["client_id", "age_loan_range", "age_account_range"]]
```

	client_id	age_loan_range	age_account_range
0	2166.0	40-49	40-49
1	11497.0	30-39	30-39
2	2181.0	20-29	20-29
3	11314.0	50-59	50-59
4	10043.0	20-29	20-29
5	13201.0	50-59	50-59
6	1215.0	50-59	50-59
7	2235.0	50-59	50-59
8	7815.0	40-49	40-49

Figure 20 - Age loan range and age account range

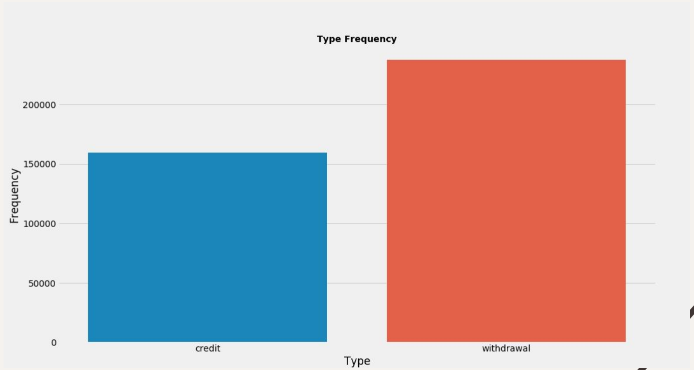
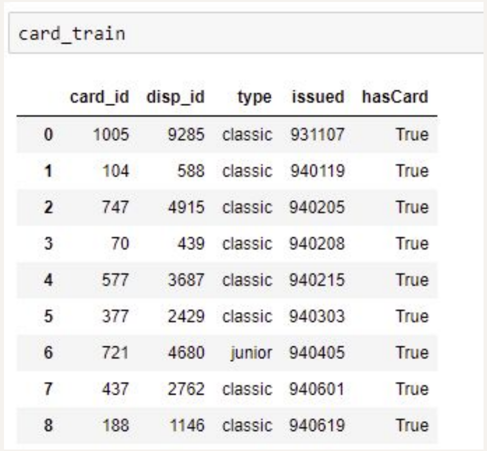


Figure 21 - Transactions type distribution


```
card_train
```

	card_id	disp_id	type	issued	hasCard
0	1005	9285	classic	931107	True
1	104	588	classic	940119	True
2	747	4915	classic	940205	True
3	70	439	classic	940208	True
4	577	3687	classic	940215	True
5	377	2429	classic	940303	True
6	721	4680	junior	940405	True
7	437	2762	classic	940601	True
8	188	1146	classic	940619	True

Figure 22 - Card table

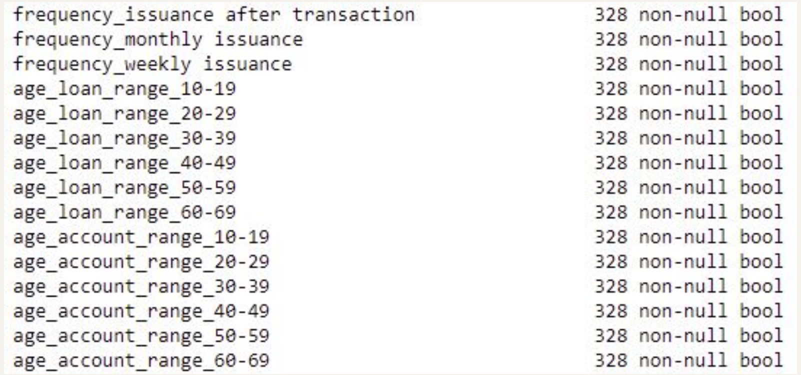

```
frequency_issuance after transaction          328 non-null bool
frequency_monthly issuance                    328 non-null bool
frequency_weekly issuance                     328 non-null bool
age_loan_range_10-19                          328 non-null bool
age_loan_range_20-29                          328 non-null bool
age_loan_range_30-39                          328 non-null bool
age_loan_range_40-49                          328 non-null bool
age_loan_range_50-59                          328 non-null bool
age_loan_range_60-69                          328 non-null bool
age_account_range_10-19                       328 non-null bool
age_account_range_20-29                       328 non-null bool
age_account_range_30-39                       328 non-null bool
age_account_range_40-49                       328 non-null bool
age_account_range_50-59                       328 non-null bool
age_account_range_60-69                       328 non-null bool
```

Figure 23 - Dummies attributes

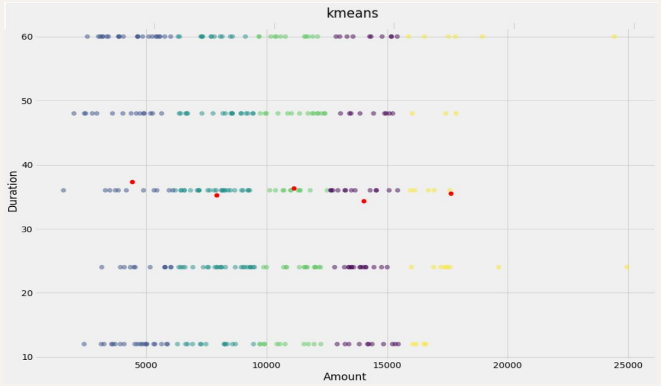


Figure 24 - Cluster Duration and Amount

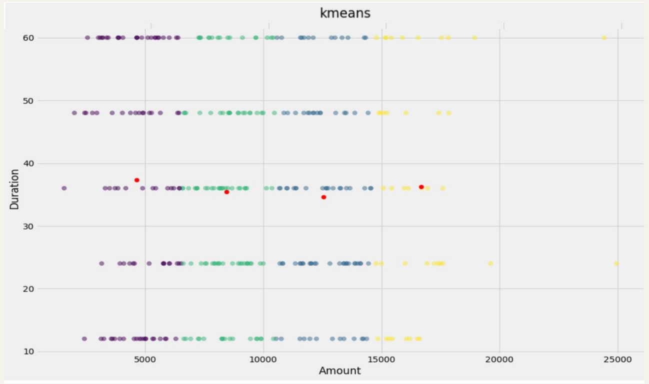


Figure 25 - Cluster Duration and Amount

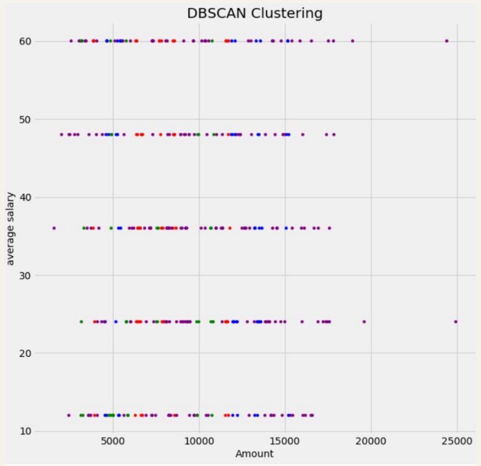


Figure 26 - Cluster Duration and Amount

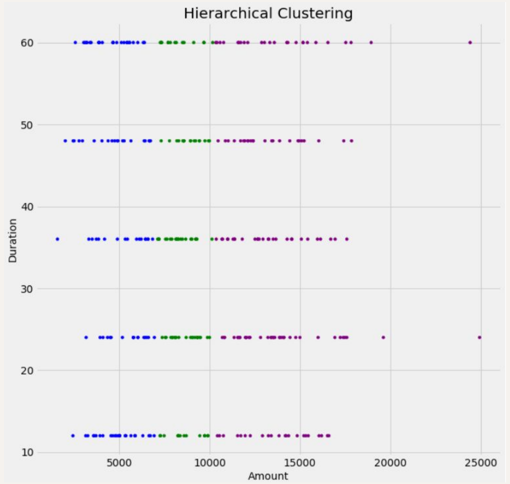


Figure 27 - Cluster Duration and Amount

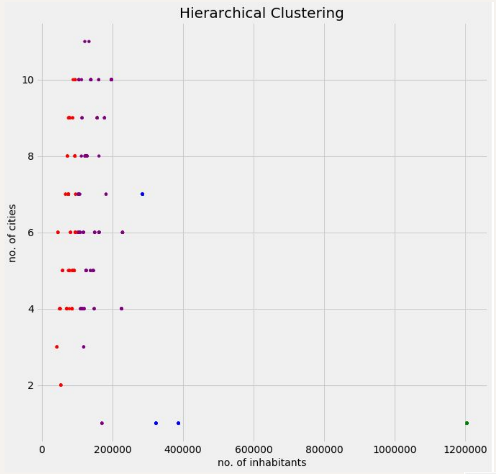


Figure 28 - Cluster Number of cities and Number of inhabitants

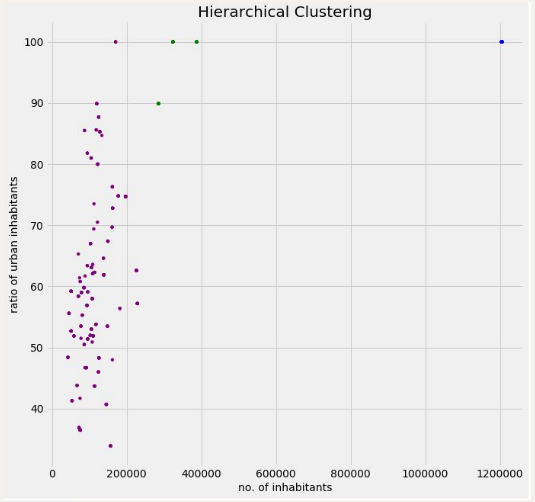


Figure 29 - Cluster Ratio of number of inhabitants and Number of inhabitants

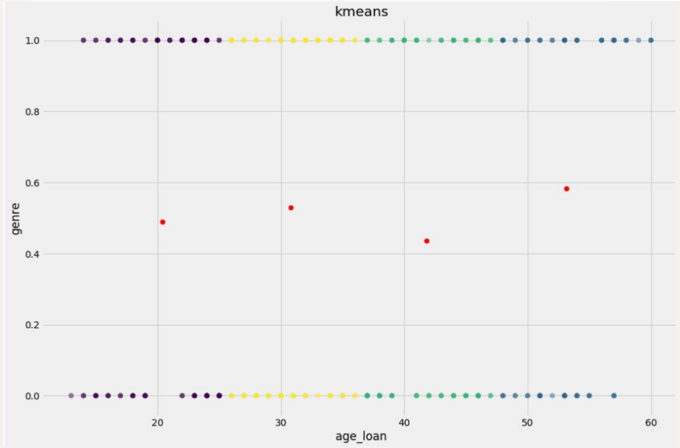


Figure 30 - Cluster Genre and Age at time of loan

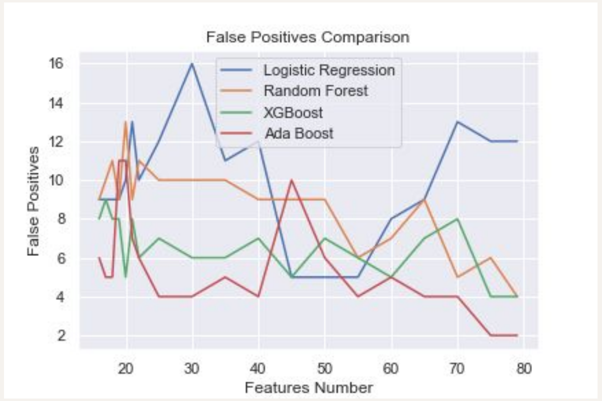


Figure 31 - False positives comparison

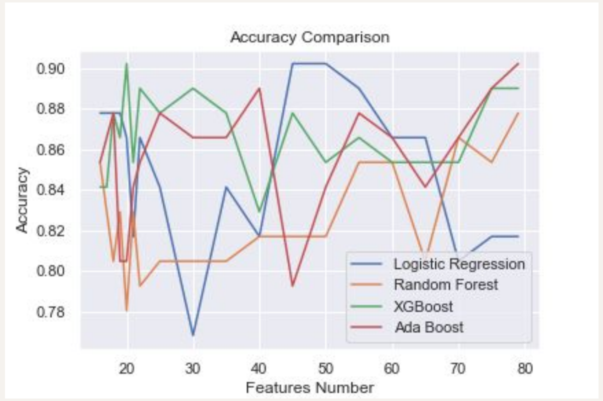


Figure 32 - Accuracy comparison

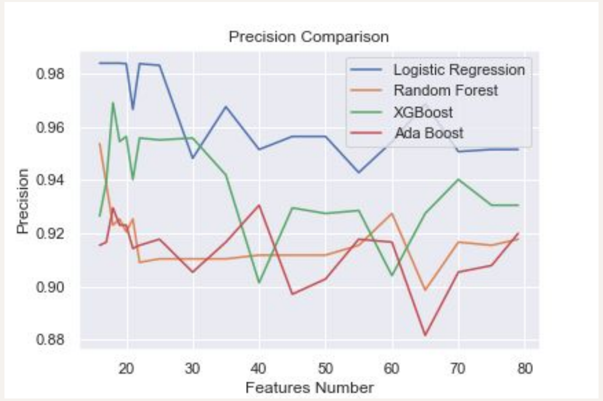


Figure 33 - Precision comparison

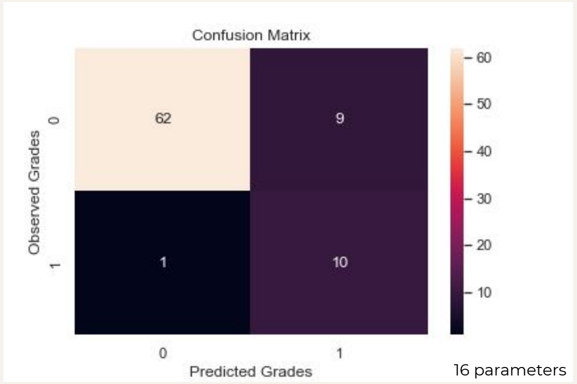


Figure 34 - COnfusion matrix 16 parameters

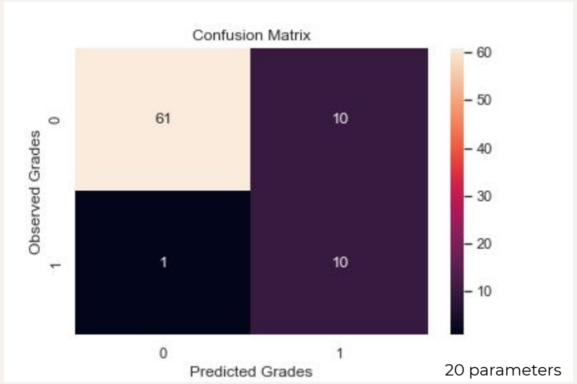


Figure 35 - Confusion matrix 20 parameters

Logistic Regression

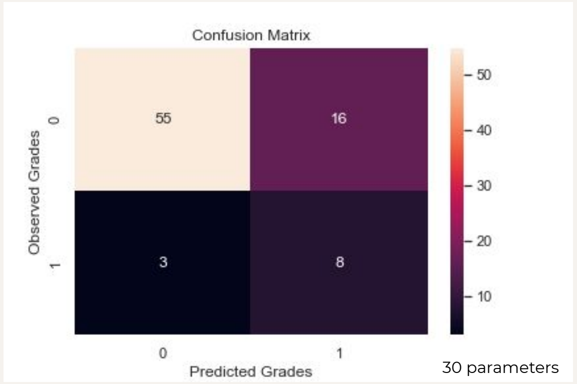


Figure 36 - Confusion matrix 30 parameters

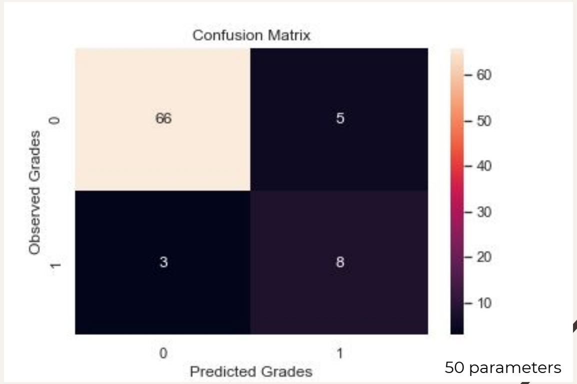


Figure 37 - Confusion matrix 50 parameters

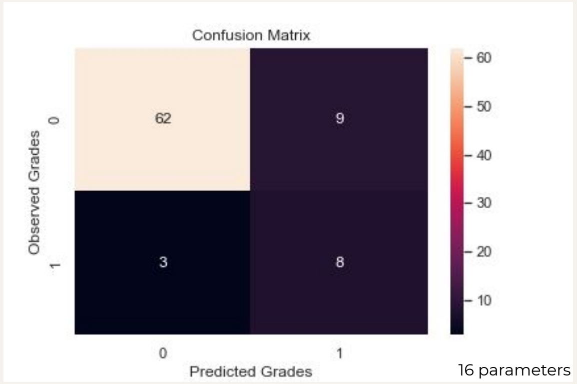


Figure 38 - Confusion matrix 16 parameters

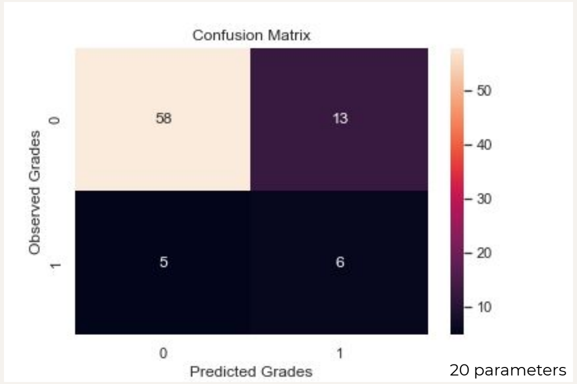


Figure 39 - COnfusion matrix 20 parameters

Random Forest

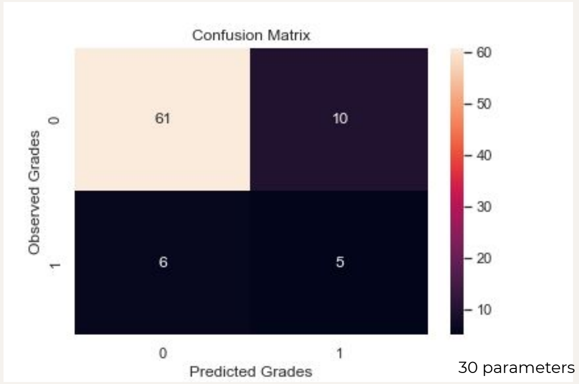


Figure 40 - Confusion matrix 30 parameters

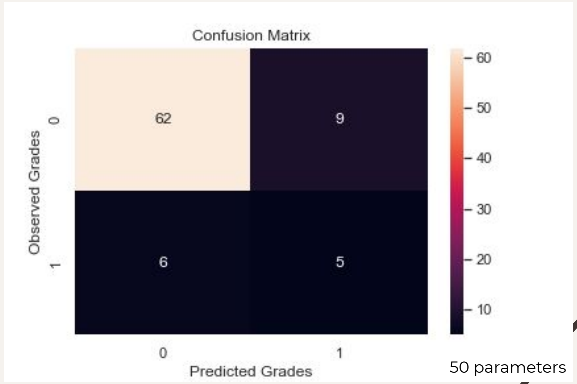


Figure 41 - Confusion matrix 50 parameters

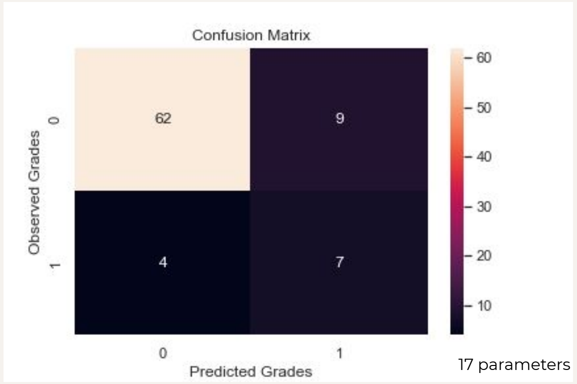


Figure 42 - Confusion matrix 17 parameters

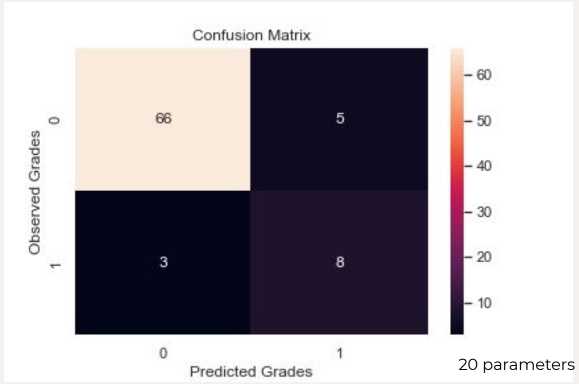


Figure 43 - Confusion matrix 20 parameters

XGBoost

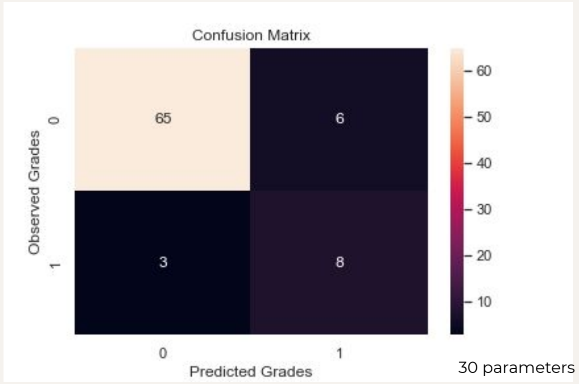


Figure 44 - Confusion matrix 30 parameters

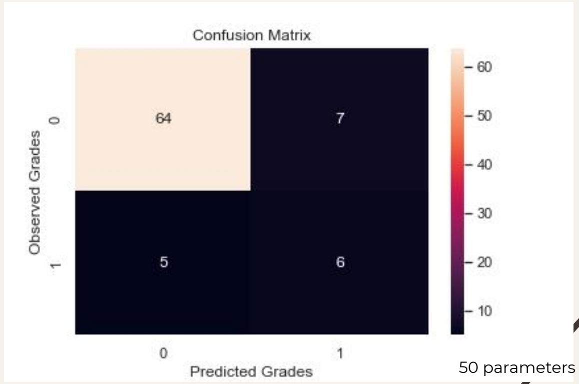


Figure 45 - Confusion matrix 50 parameters

AdaBoost

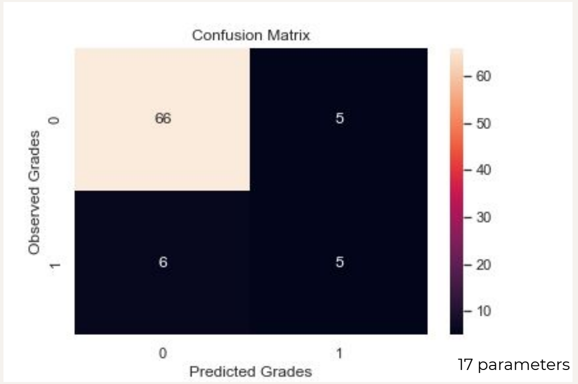


Figure 46 - Confusion matrix 17 parameters

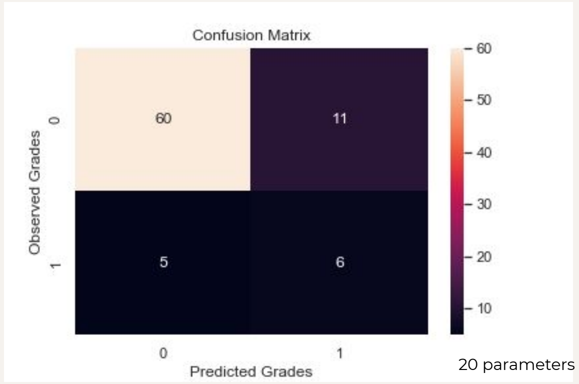


Figure 47 - Confusion matrix 20 parameters

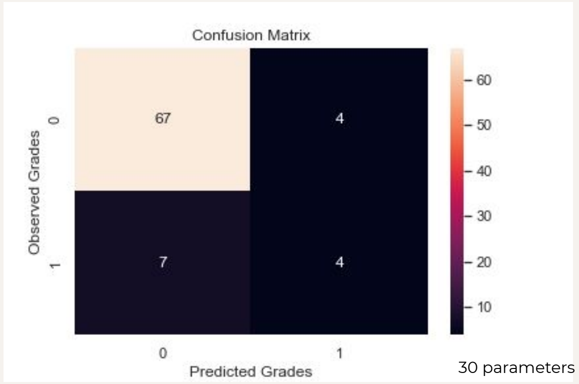


Figure 48 - Confusion matrix 30 parameters

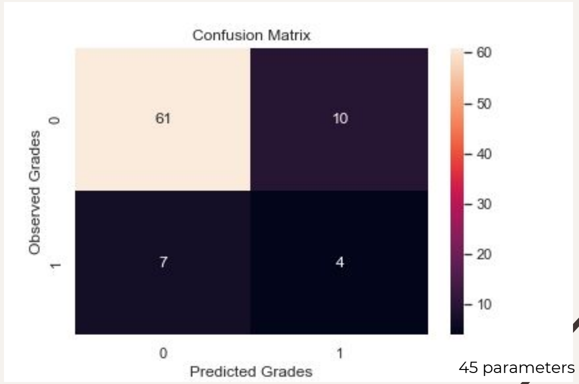


Figure 49 - Confusion matrix 45 parameters