Data Mining Project

Group 15 - AC 2021/2022
Iohan Sardinha - up201801011
João Vasconcelos - up201504397
Tiago Araújo - up202109481

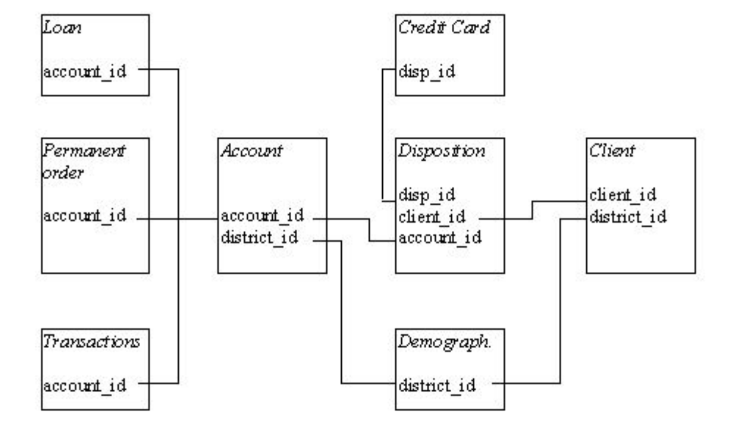
Domain Description

The domain of this task comes from a collection of relations from banking data.

The data is distributed in many datasets that were combined into the domain.

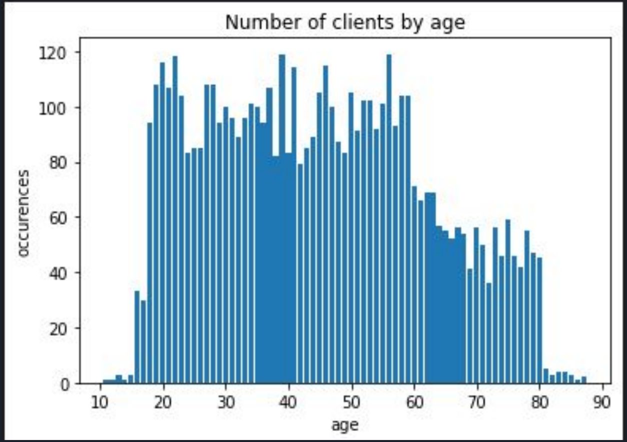
The information gathered was the description of previous loans given by the bank and the account information that asks for the loan.

In the end the domain consisted of 44 attributes about the account, the loan, the client that owns the account, the account transactions, and card information.

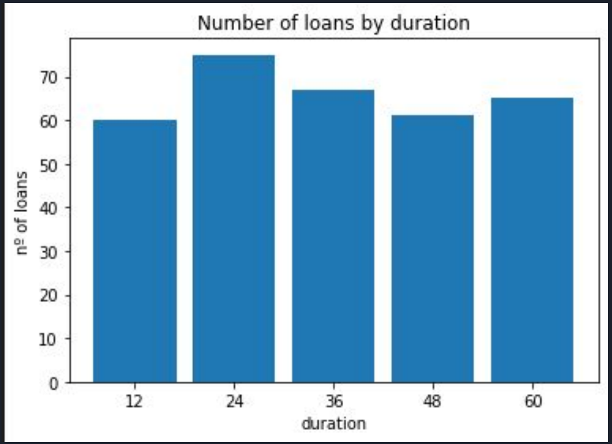


Exploratory Data Analysis

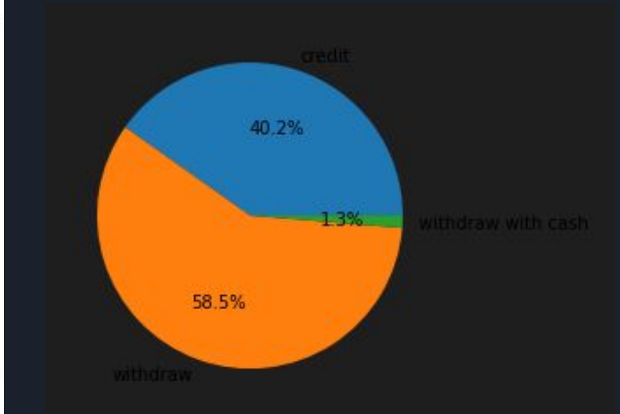
- Number of clients by age



- Number of loans by duration

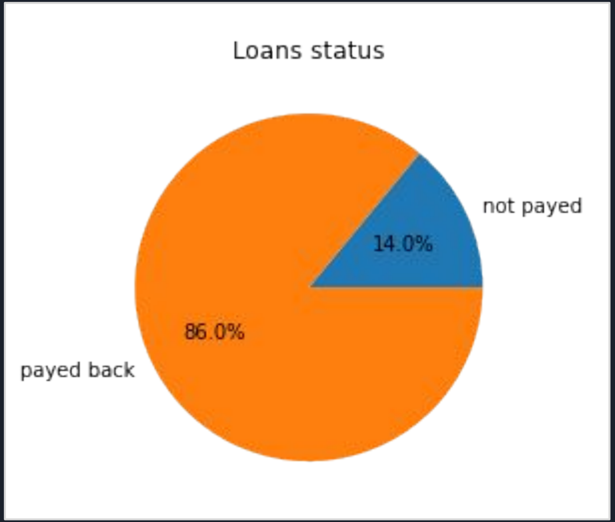


- Transaction type (credit, withdraw, withdraw with cash)

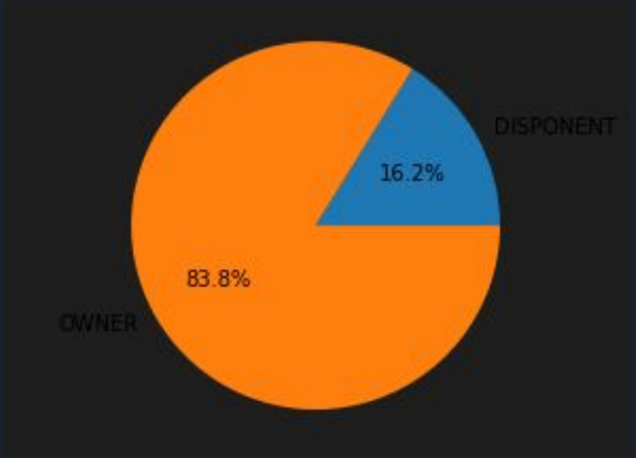


Exploratory Data Analysis

- Comparison between the number of Loans paid back and not payed



- Comparison between the number of (account owners and disponents)



Problem Definition

The given problem was to find the chance of loan requested by an account failing. In order for the bank the analyze the results and decide if it would give or not the loan.

That is, given the account and loan request data the model should tell the probability that the client will not pay back what it owes.

While minimizing the number of false negatives, which are the cases when the model predicts that a client will not fail to pay the loan but it does, since this are the cases that give the most loss for the bank.

Also very important was the certainty of the predictions, that is a model that would me the more sure that it's guess was true as possible. But a model which is always 100% like a binary classification is not desired since the bank will check the results before actually giving the loan and for that the degree of certainty is important.

Data Preparation

To be able to work with models the date needs to be united, organized and transformed, which was done in parts:

- Data Wrangling

  The datasets were merged into a single *loan* dataset, with the loan id, loan result, and other 44 columns from all the relations, in order to be able to be fitted in models.

- Missing Values

  There were many missing values in different datasets, but after the union there were only two attributes with a single entrance missing: *unemployment rate '95* and *no. of committed crimes '95*, for the first was assigned the value of the rate in 96 minus the average growth from 95 to 96, and for the second was assigned the value in 96, since the average difference was very small.

Data Preparation

- Feature engineering

  Some columns had unimportant data or data that could not be directly used, such as dates.

  From that new columns were created and some were deleted, like extracting the age, and gender of a client which were not originally attributes.

- Transaction data

  Since the transaction dataset had multiple entries for each account it was reduced into a description of each account transaction history.

  This history contains: the number of movements, the value of the lowest, biggest, and average values of transactions and balance.

Experimental Setup

Using Interactive Python with Scikit-Learn a variety of models were tested using different methods:

Oversampling, since the dataset had significantly fewer cases of failure then success, for that Synthetic Minority Oversampling Technique (SMOTE) was used

Feature selection, to measure the effect of each attribute in the model, for that the best method was Recursive Feature Elimination (RFE)

Cross validation, the methods above were applied inside a KFold Cross Validation, using the metric area under the curve (auc) in order to find the model which better differentiated between the classes of the problem

Experimental Setup

Parameter Tuning, alongside with the cross validation the models were tested with different parameters in order to find the ones that best suited the dataset

Before applying any method the data was normalized, for each categorical a binary column was created denoting it’s presence or not.

The models tested were: K Nearest Neighbours, Logistic Regression, Support Vector Machines, Naive Bayes, Gradient Boost, Bagging Classifier, Decision Tree and Random Forest

The model with best auc was selected.

Results

The best results were reached using a Random Forest model, although many times the Gradient Boost model got very similar to scores.

The preprocessing methods that yield the best results where:

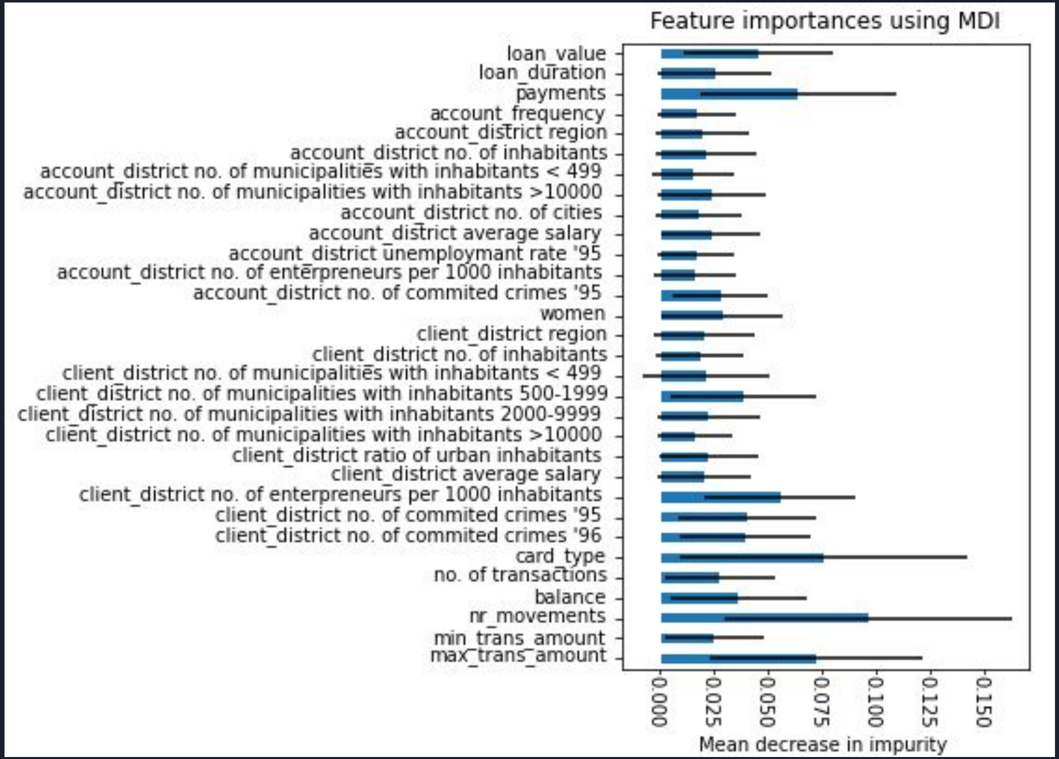
Normalizing data with MinMaxScaler and OneHotEncoder - that scale the date to a range from 0-1 and create binary columns for each categorical column.

Oversampling with SMOTE, selecting features with RFE

And a Random Forest with class_weight as balanced subsample and max_features as square root

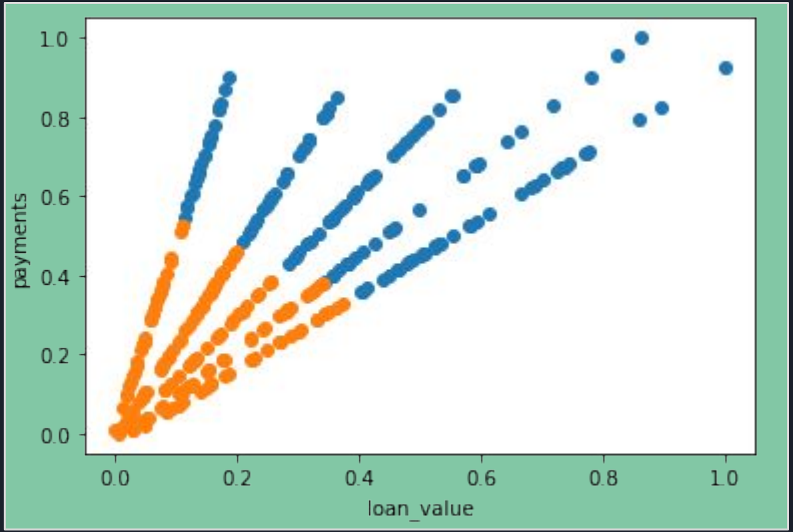
Results

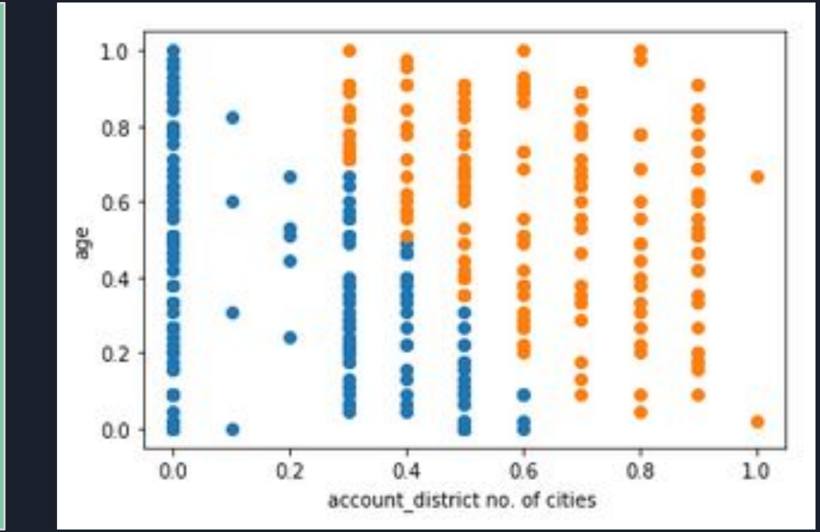
The final best score obtained in kaggle was 86% for private, with 92% for the public cases and estimation of 99% with Random Forest



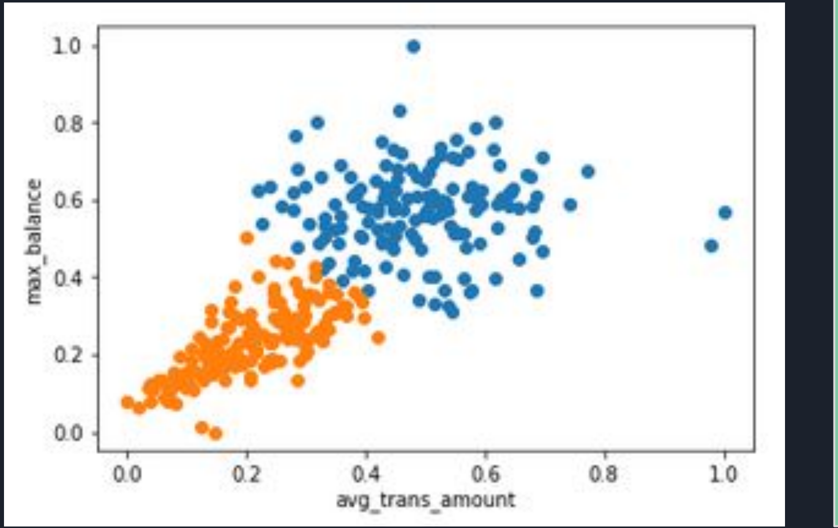
Although there was a submission from gradient boost that would get 89% in the private cases if it was between the selected ones

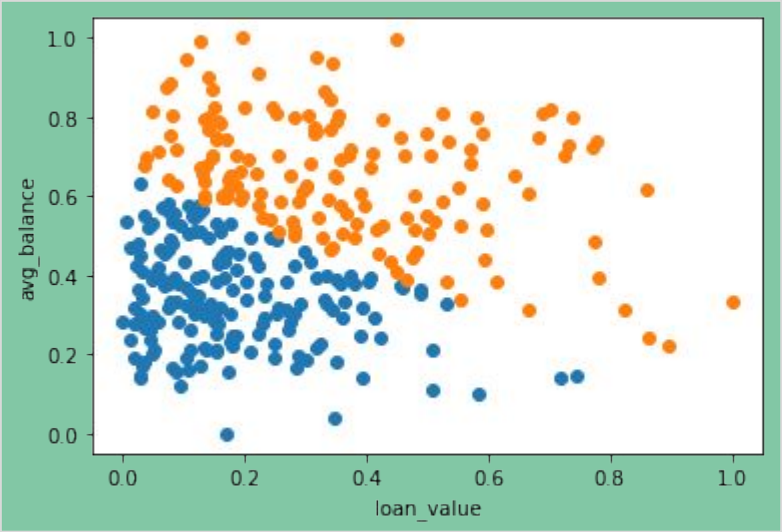
Data Clusters





Data Clusters





Conclusions

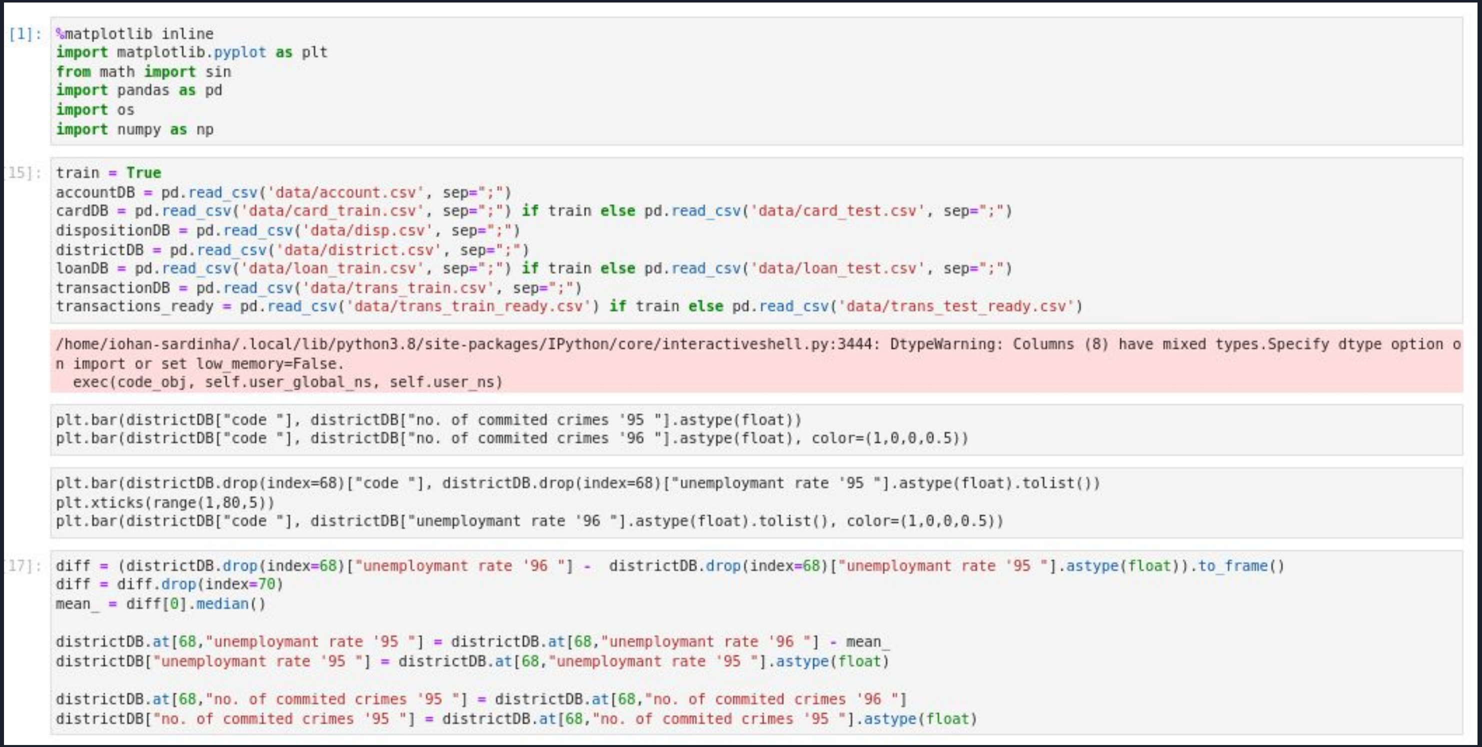
In the end the although the modeling part took some good amount of time and experimentation of different techniques the handling of the data was what required more attention and had the biggest effect on the final results.

The results on the training set not always reflected the real results.

Oversampling and feature selection made the results significantly better.

The Random Forest model turned out to be the best model.

Annex 1 - Uniting Data code



```
[1]: %matplotlib inline
     import matplotlib.pyplot as plt
     from math import sin
     import pandas as pd
     import os
     import numpy as np
```

```
15]: train = True
     accountDB = pd.read_csv('data/account.csv', sep=";")
     cardDB = pd.read_csv('data/card_train.csv', sep=";") if train else pd.read_csv('data/card_test.csv', sep=";")
     dispositionDB = pd.read_csv('data/disp.csv', sep=";")
     districtDB = pd.read_csv('data/district.csv', sep=";")
     loanDB = pd.read_csv('data/loan_train.csv', sep=";") if train else pd.read_csv('data/loan_test.csv', sep=";")
     transactionDB = pd.read_csv('data/trans_train.csv', sep=";")
     transactions_ready = pd.read_csv('data/trans_train_ready.csv') if train else pd.read_csv('data/trans_test_ready.csv')
```

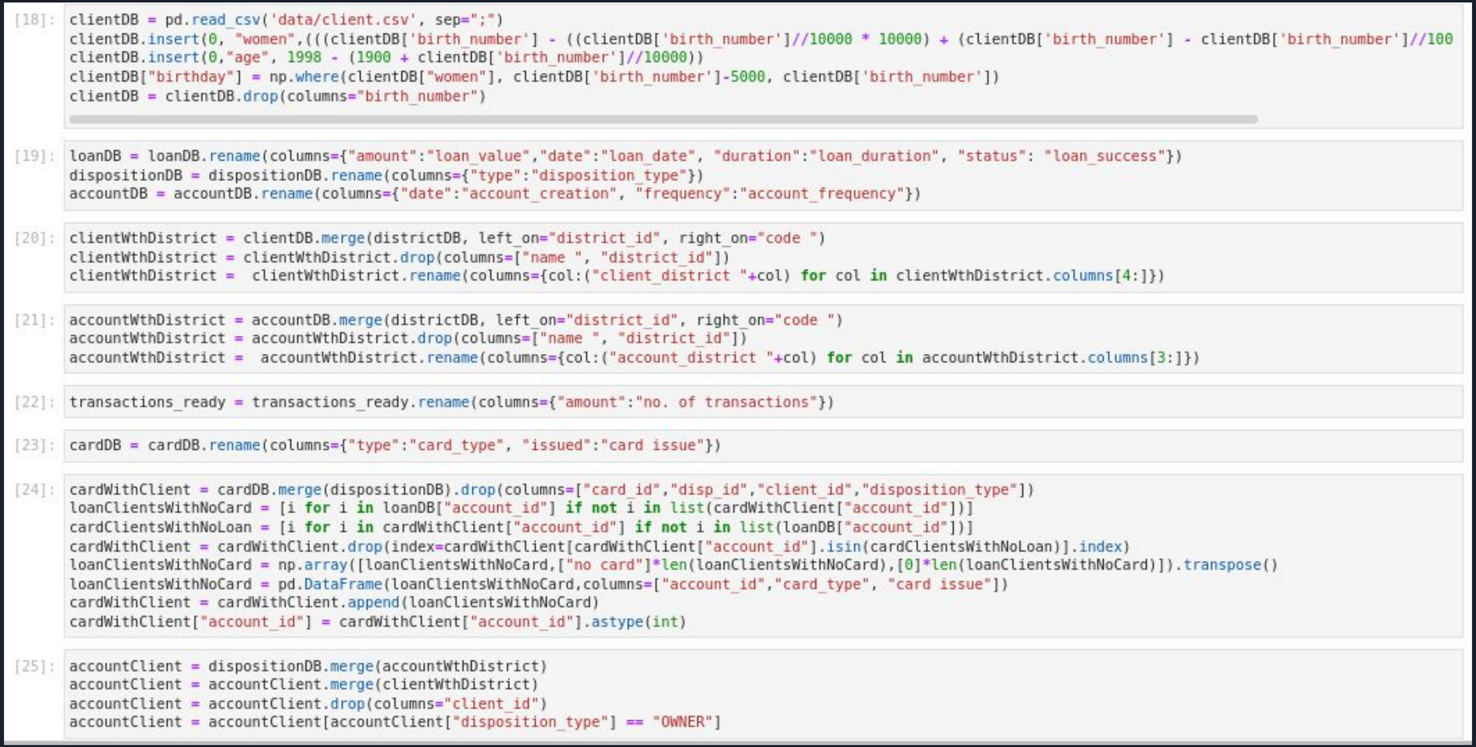
```
/home/iohan-sardinha/.local/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3444: DtypeWarning: Columns (8) have mixed types.Specify dtype option o
n import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
plt.bar(districtDB["code "], districtDB["no. of commited crimes '95 "].astype(float))
plt.bar(districtDB["code "], districtDB["no. of commited crimes '96 "].astype(float), color=(1,0,0,0.5))
```

```
plt.bar(districtDB.drop(index=68)["code "], districtDB.drop(index=68)["unemploymant rate '95 "].astype(float).tolist())
plt.xticks(range(1,80,5))
plt.bar(districtDB["code "], districtDB["unemploymant rate '96 "].astype(float).tolist(), color=(1,0,0,0.5))
```

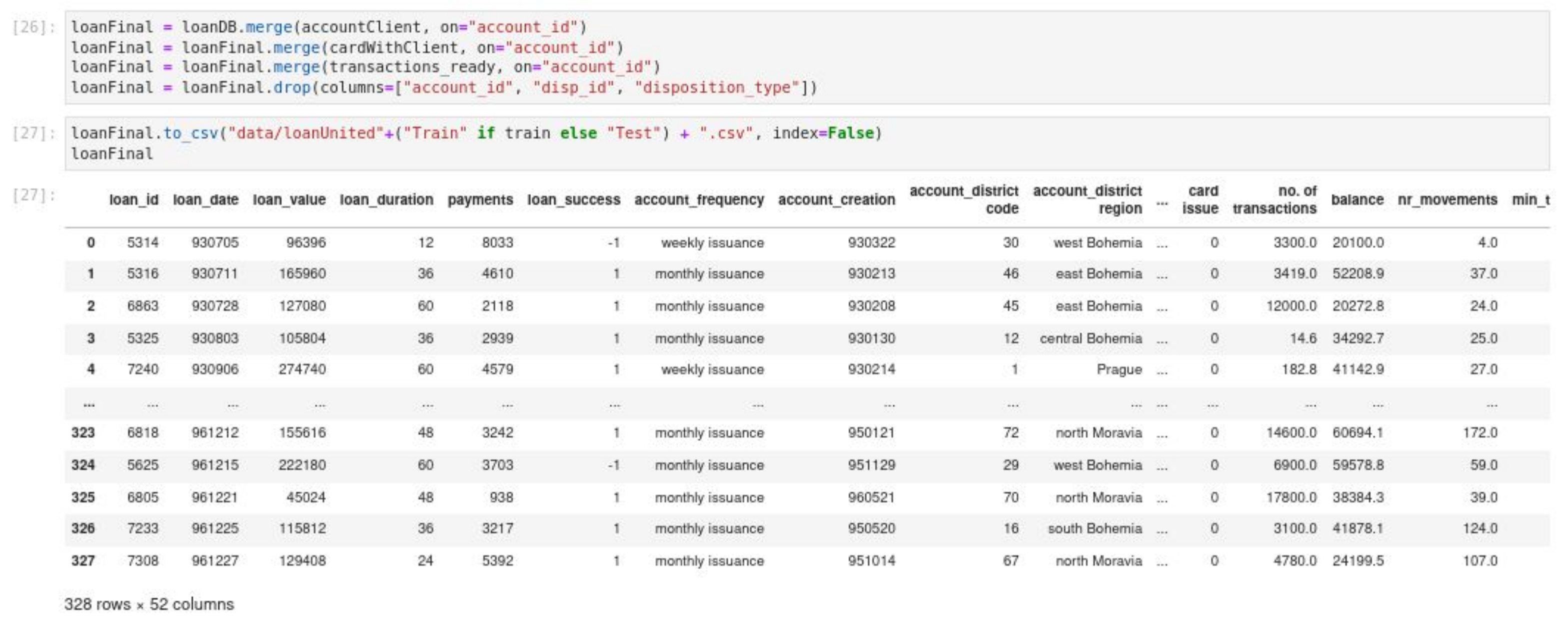
```
17]: diff = (districtDB.drop(index=68)["unemploymant rate '96 "] -  districtDB.drop(index=68)["unemploymant rate '95 "].astype(float)).to_frame()
     diff = diff.drop(index=70)
     mean_ = diff[0].median()

     districtDB.at[68,"unemploymant rate '95 "] = districtDB.at[68,"unemploymant rate '96 "] - mean_
     districtDB["unemploymant rate '95 "] = districtDB.at[68,"unemploymant rate '95 "].astype(float)

     districtDB.at[68,"no. of commited crimes '95 "] = districtDB.at[68,"no. of commited crimes '96 "]
     districtDB["no. of commited crimes '95 "] = districtDB.at[68,"no. of commited crimes '95 "].astype(float)
```

Annex 1 - Uniting Data code



```
[18]: clientDB = pd.read_csv('data/client.csv', sep=";")
      clientDB.insert(0, "women",(((clientDB['birth_number'] - ((clientDB['birth_number']//10000 * 10000) + (clientDB['birth_number'] - clientDB['birth_number']//100
      clientDB.insert(0,"age", 1998 - (1900 + clientDB['birth_number']//10000))
      clientDB["birthday"] = np.where(clientDB["women"], clientDB['birth_number']-5000, clientDB['birth_number'])
      clientDB = clientDB.drop(columns="birth_number")

[19]: loanDB = loanDB.rename(columns={"amount":"loan_value","date":"loan_date", "duration":"loan_duration", "status": "loan_success"})
      dispositionDB = dispositionDB.rename(columns={"type":"disposition_type"})
      accountDB = accountDB.rename(columns={"date":"account_creation", "frequency":"account_frequency"})

[20]: clientWthDistrict = clientDB.merge(districtDB, left_on="district_id", right_on="code ")
      clientWthDistrict = clientWthDistrict.drop(columns=["name ", "district_id"])
      clientWthDistrict =  clientWthDistrict.rename(columns={col:("client_district "+col) for col in clientWthDistrict.columns[4:]})

[21]: accountWthDistrict = accountDB.merge(districtDB, left_on="district_id", right_on="code ")
      accountWthDistrict = accountWthDistrict.drop(columns=["name ", "district_id"])
      accountWthDistrict =  accountWthDistrict.rename(columns={col:("account_district "+col) for col in accountWthDistrict.columns[3:]})

[22]: transactions_ready = transactions_ready.rename(columns={"amount":"no. of transactions"})

[23]: cardDB = cardDB.rename(columns={"type":"card_type", "issued":"card_issue"})

[24]: cardWithClient = cardDB.merge(dispositionDB).drop(columns=["card_id","disp_id","client_id","disposition_type"])
      loanClientsWithNoCard = [i for i in loanDB["account_id"] if not i in list(cardWithClient["account_id"])]
      cardClientsWithNoLoan = [i for i in cardWithClient["account_id"] if not i in list(loanDB["account_id"])]
      cardWithClient = cardWithClient.drop(index=cardWithClient[cardWithClient["account_id"].isin(cardClientsWithNoLoan)].index)
      loanClientsWithNoCard = np.array([loanClientsWithNoCard,["no card"]*len(loanClientsWithNoCard),[0]*len(loanClientsWithNoCard)]).transpose()
      loanClientsWithNoCard = pd.DataFrame(loanClientsWithNoCard,columns=["account_id","card_type", "card_issue"])
      cardWithClient = cardWithClient.append(loanClientsWithNoCard)
      cardWithClient["account_id"] = cardWithClient["account_id"].astype(int)

[25]: accountClient = dispositionDB.merge(accountWthDistrict)
      accountClient = accountClient.merge(clientWthDistrict)
      accountClient = accountClient.drop(columns="client_id")
      accountClient = accountClient[accountClient["disposition_type"] == "OWNER"]
```

Annex 1 - Uniting Data code



```
[26]: loanFinal = loanDB.merge(accountClient, on="account_id")
      loanFinal = loanFinal.merge(cardWithClient, on="account_id")
      loanFinal = loanFinal.merge(transactions_ready, on="account_id")
      loanFinal = loanFinal.drop(columns=["account_id", "disp_id", "disposition_type"])
```

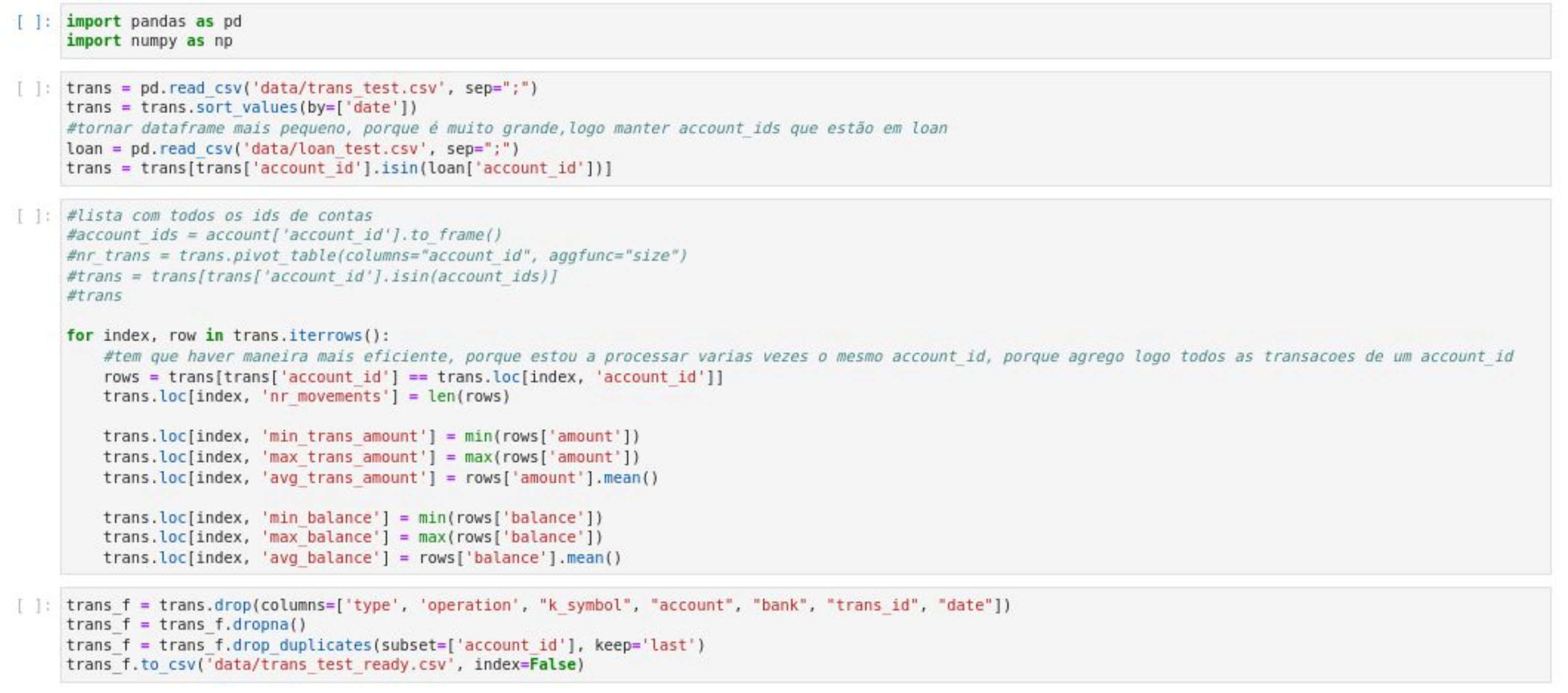
```
[27]: loanFinal.to_csv("data/loanUnited"+("Train" if train else "Test") + ".csv", index=False)
      loanFinal
```

[27]:

	loan_id	loan_date	loan_value	loan_duration	payments	loan_success	account_frequency	account_creation	account_district code	account_district region	...	card issue	no. of transactions	balance	nr_movements	min_t
0	5314	930705	96396	12	8033	-1	weekly issuance	930322	30	west Bohemia	...	0	3300.0	20100.0	4.0	
1	5316	930711	165960	36	4610	1	monthly issuance	930213	46	east Bohemia	...	0	3419.0	52208.9	37.0	
2	6863	930728	127080	60	2118	1	monthly issuance	930208	45	east Bohemia	...	0	12000.0	20272.8	24.0	
3	5325	930803	105804	36	2939	1	monthly issuance	930130	12	central Bohemia	...	0	14.6	34292.7	25.0	
4	7240	930906	274740	60	4579	1	weekly issuance	930214	1	Prague	...	0	182.8	41142.9	27.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
323	6818	961212	155616	48	3242	1	monthly issuance	950121	72	north Moravia	...	0	14600.0	60694.1	172.0	
324	5625	961215	222180	60	3703	-1	monthly issuance	951129	29	west Bohemia	...	0	6900.0	59578.8	59.0	
325	6805	961221	45024	48	938	1	monthly issuance	960521	70	north Moravia	...	0	17800.0	38384.3	39.0	
326	7233	961225	115812	36	3217	1	monthly issuance	950520	16	south Bohemia	...	0	3100.0	41878.1	124.0	
327	7308	961227	129408	24	5392	1	monthly issuance	951014	67	north Moravia	...	0	4780.0	24199.5	107.0	

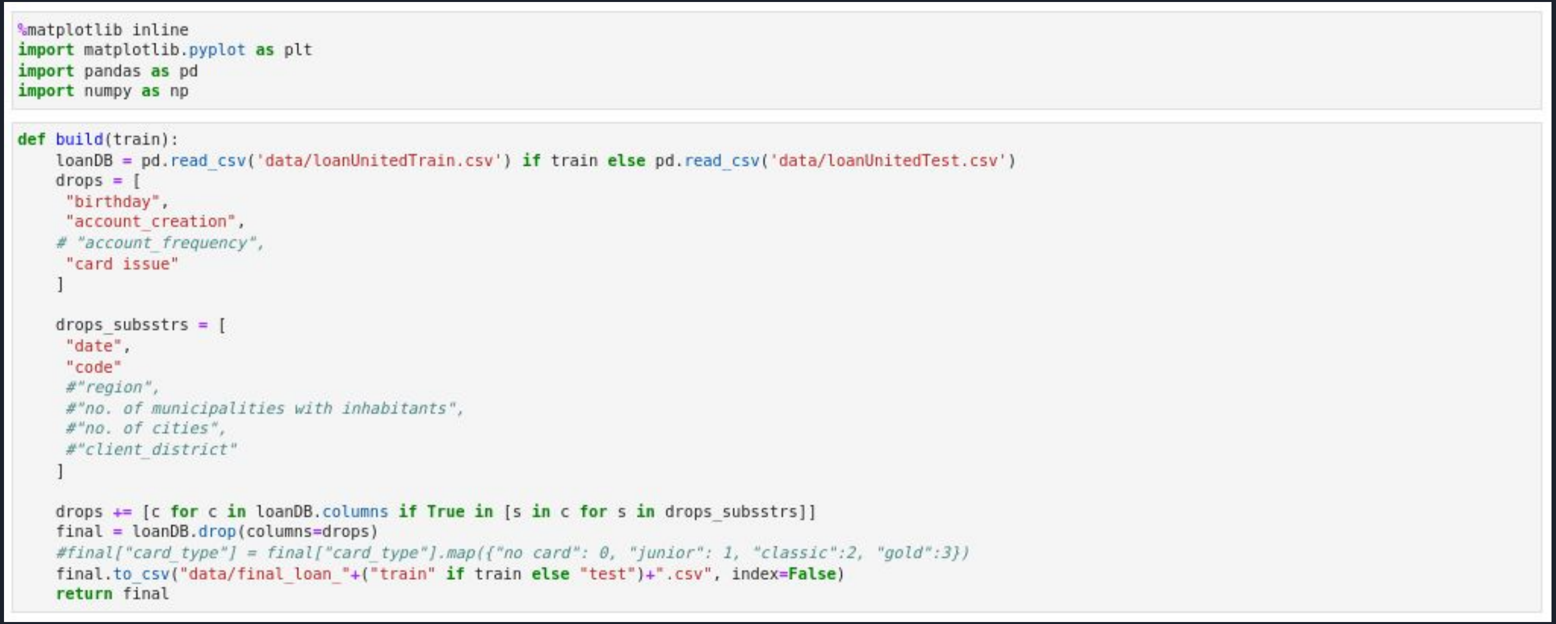
328 rows × 52 columns

Annex 2 - Transaction Reduction Code



```
[ ]: import pandas as pd
     import numpy as np

[ ]: trans = pd.read_csv('data/trans_test.csv', sep=";")
     trans = trans.sort_values(by=['date'])
     #tornar dataframe mais pequeno, porque é muito grande,logo manter account_ids que estão em loan
     loan = pd.read_csv('data/loan_test.csv', sep=";")
     trans = trans[trans['account_id'].isin(loan['account_id'])]

[ ]: #lista com todos os ids de contas
     #account_ids = account['account_id'].to_frame()
     #nr_trans = trans.pivot_table(columns="account_id", aggfunc="size")
     #trans = trans[trans['account_id'].isin(account_ids)]
     #trans

     for index, row in trans.iterrows():
         #tem que haver maneira mais eficiente, porque estou a processar varias vezes o mesmo account_id, porque agrego logo todos as transacoes de um account_id
         rows = trans[trans['account_id'] == trans.loc[index, 'account_id']]
         trans.loc[index, 'nr_movements'] = len(rows)

         trans.loc[index, 'min_trans_amount'] = min(rows['amount'])
         trans.loc[index, 'max_trans_amount'] = max(rows['amount'])
         trans.loc[index, 'avg_trans_amount'] = rows['amount'].mean()

         trans.loc[index, 'min_balance'] = min(rows['balance'])
         trans.loc[index, 'max_balance'] = max(rows['balance'])
         trans.loc[index, 'avg_balance'] = rows['balance'].mean()

[ ]: trans_f = trans.drop(columns=['type', 'operation', "k_symbol", "account", "bank", "trans_id", "date"])
     trans_f = trans_f.dropna()
     trans_f = trans_f.drop_duplicates(subset=['account_id'], keep='last')
     trans_f.to_csv('data/trans_test_ready.csv', index=False)
```

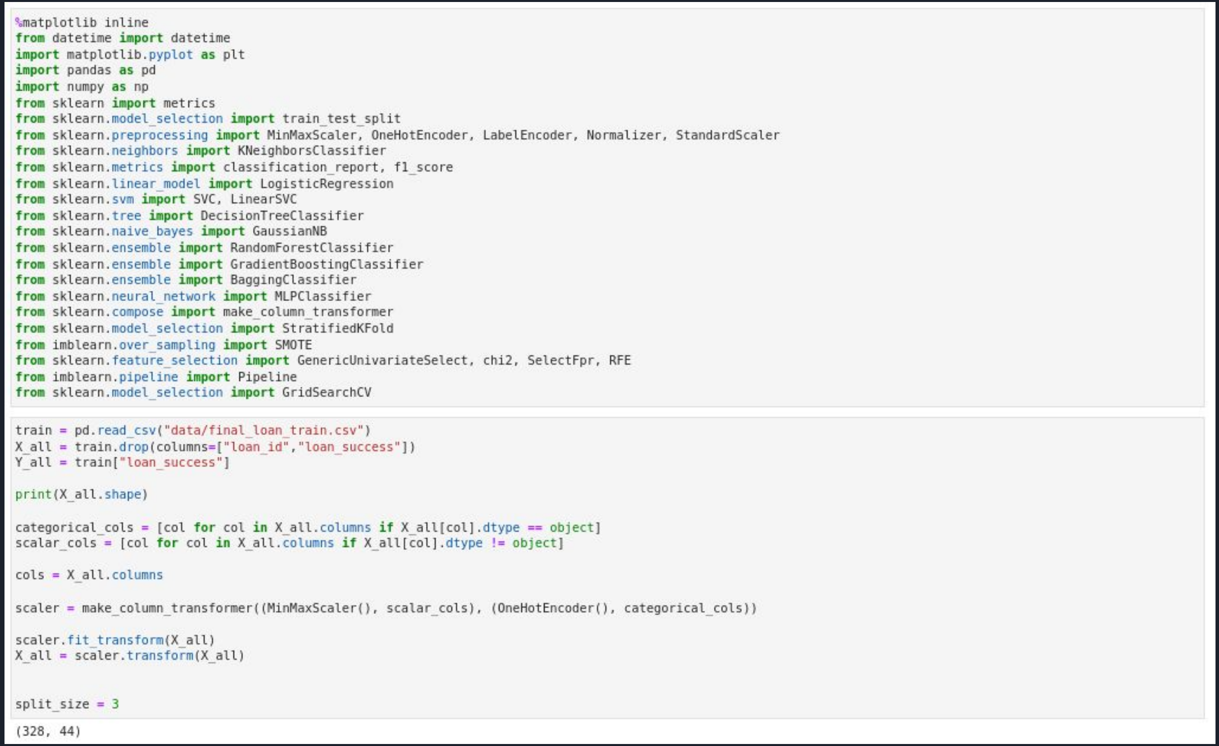
Annex 3 - Cleanup Code



```python
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```python
def build(train):
    loanDB = pd.read_csv('data/loanUnitedTrain.csv') if train else pd.read_csv('data/loanUnitedTest.csv')
    drops = [
     "birthday",
     "account_creation",
    # "account_frequency",
     "card issue"
    ]

    drops_subsstrs = [
     "date",
     "code"
     #"region",
     #"no. of municipalities with inhabitants",
     #"no. of cities",
     #"client_district"
    ]

    drops += [c for c in loanDB.columns if True in [s in c for s in drops_subsstrs]]
    final = loanDB.drop(columns=drops)
    #final["card_type"] = final["card_type"].map({"no card": 0, "junior": 1, "classic":2, "gold":3})
    final.to_csv("data/final_loan_"+("train" if train else "test")+".csv", index=False)
    return final
```

Annex 4 - Model Code

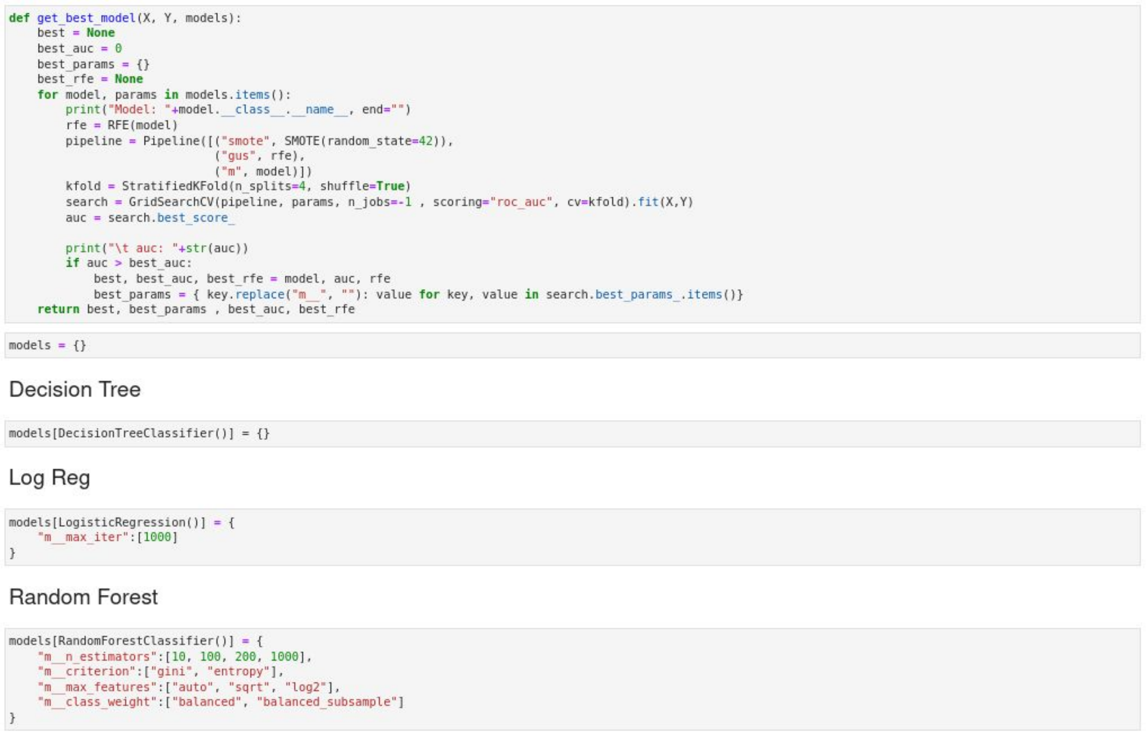


```
%matplotlib inline
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, LabelEncoder, Normalizer, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.compose import make_column_transformer
from sklearn.model_selection import StratifiedKFold
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import GenericUnivariateSelect, chi2, SelectFpr, RFE
from imblearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

```
train = pd.read_csv("data/final_loan_train.csv")
X_all = train.drop(columns=["loan_id","loan_success"])
Y_all = train["loan_success"]

print(X_all.shape)

categorical_cols = [col for col in X_all.columns if X_all[col].dtype == object]
scalar_cols = [col for col in X_all.columns if X_all[col].dtype != object]

cols = X_all.columns

scaler = make_column_transformer((MinMaxScaler(), scalar_cols), (OneHotEncoder(), categorical_cols))

scaler.fit_transform(X_all)
X_all = scaler.transform(X_all)


split_size = 3
```

```
(328, 44)
```

Annex 4 - Model Code



```
def get_best_model(X, Y, models):
    best = None
    best_auc = 0
    best_params = {}
    best_rfe = None
    for model, params in models.items():
        print("Model: "+model.__class__.__name__, end="")
        rfe = RFE(model)
        pipeline = Pipeline([("smote", SMOTE(random_state=42)),
                             ("gus", rfe),
                             ("m", model)])
        kfold = StratifiedKFold(n_splits=4, shuffle=True)
        search = GridSearchCV(pipeline, params, n_jobs=-1 , scoring="roc_auc", cv=kfold).fit(X,Y)
        auc = search.best_score_

        print("\t auc: "+str(auc))
        if auc > best_auc:
            best, best_auc, best_rfe = model, auc, rfe
            best_params = { key.replace("m__", ""): value for key, value in search.best_params_.items()}
    return best, best_params , best_auc, best_rfe
```

```
models = {}
```

Decision Tree

```
models[DecisionTreeClassifier()] = {}
```

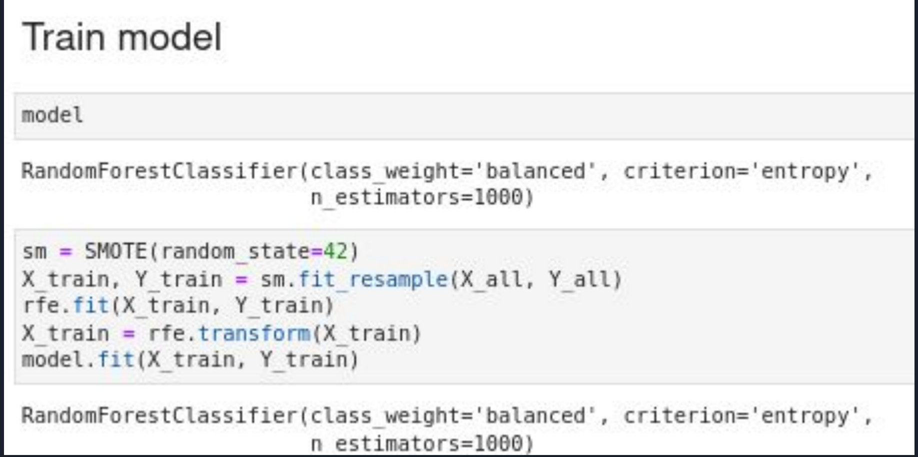
Log Reg

```
models[LogisticRegression()] = {
    "m__max_iter":[1000]
}
```

Random Forest

```
models[RandomForestClassifier()] = {
    "m__n_estimators":[10, 100, 200, 1000],
    "m__criterion":["gini", "entropy"],
    "m__max_features":["auto", "sqrt", "log2"],
    "m__class_weight":["balanced", "balanced_subsample"]
}
```

Annex 4 - Model Code



Train model

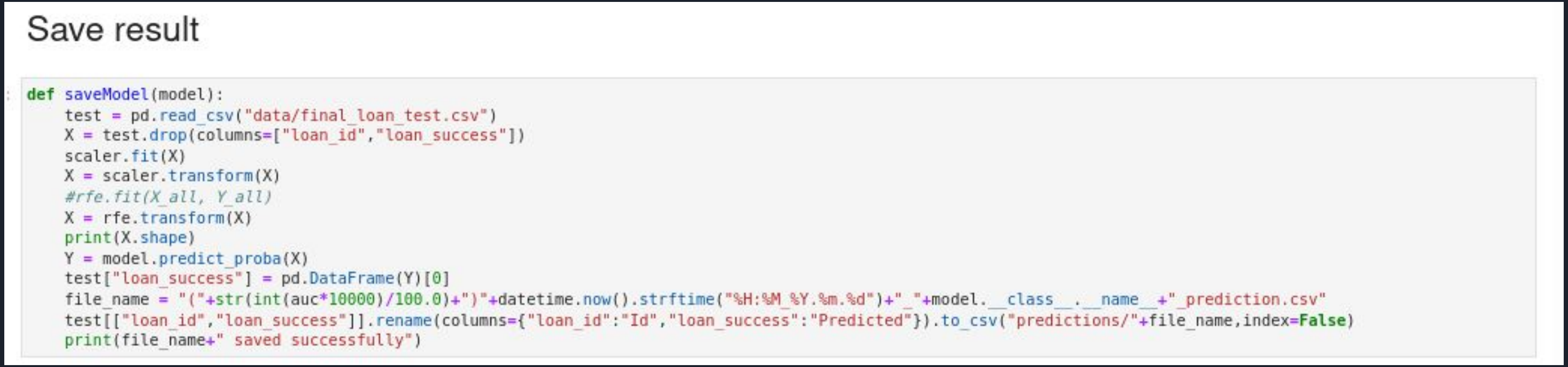
```
model
```

```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       n_estimators=1000)
```

```
sm = SMOTE(random_state=42)
X_train, Y_train = sm.fit_resample(X_all, Y_all)
rfe.fit(X_train, Y_train)
X_train = rfe.transform(X_train)
model.fit(X_train, Y_train)
```

```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       n_estimators=1000)
```

Annex 4 - Model Code



Save result

```python
def saveModel(model):
    test = pd.read_csv("data/final_loan_test.csv")
    X = test.drop(columns=["loan_id","loan_success"])
    scaler.fit(X)
    X = scaler.transform(X)
    #rfe.fit(X_all, Y_all)
    X = rfe.transform(X)
    print(X.shape)
    Y = model.predict_proba(X)
    test["loan_success"] = pd.DataFrame(Y)[0]
    file_name = "("+str(int(auc*10000)/100.0)+")"+datetime.now().strftime("%H:%M_%Y.%m.%d")+"_"+model.__class__.__name__+"_prediction.csv"
    test[["loan_id","loan_success"]].rename(columns={"loan_id":"Id","loan_success":"Predicted"}).to_csv("predictions/"+file_name,index=False)
    print(file_name+" saved successfully")
```