Data Mining Project

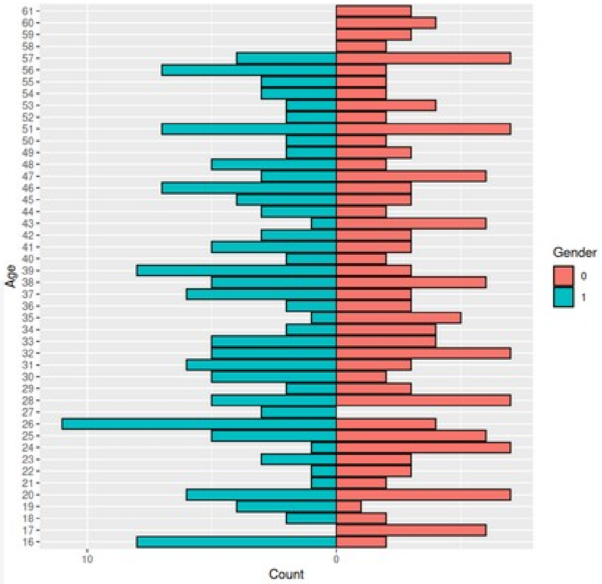
Miguel Amorim, up201907756
Rita Mendes, up201907877
Tiago Rodrigues, up201907021

Domain

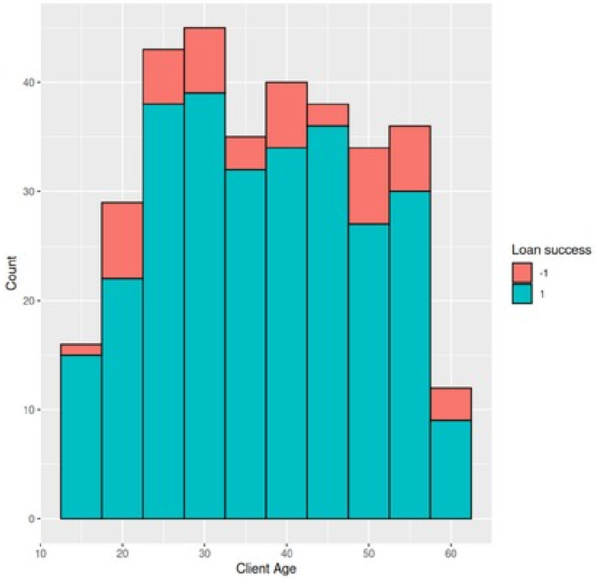
It is essential to first thoroughly understand what the customer really wants to accomplish. From a productivity standpoint, this implies that with a well defined goal, it is easy to go searching for the correct questions and, as follows, the correct answers. As such, there are a few key aspects to take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

Data Understanding

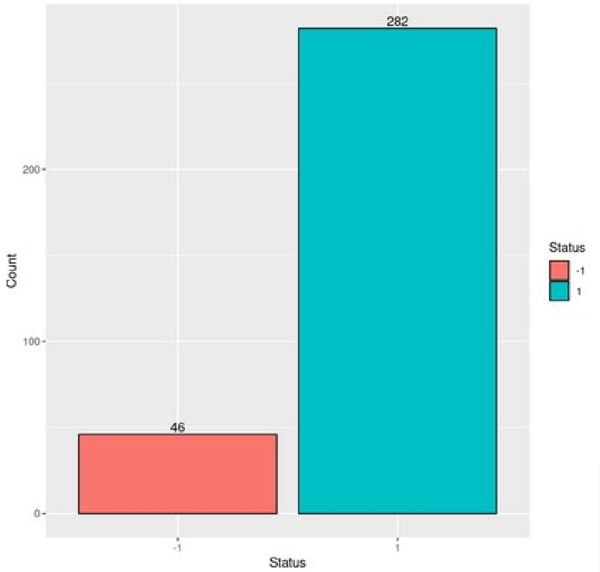
Here are some of the features that seem to show up more when exploring the data using only small transformations.



Dataset Population Characterization

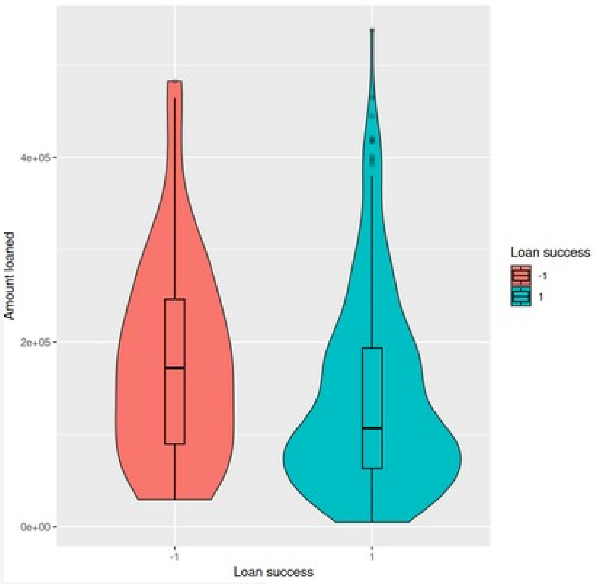


Loan success and client age

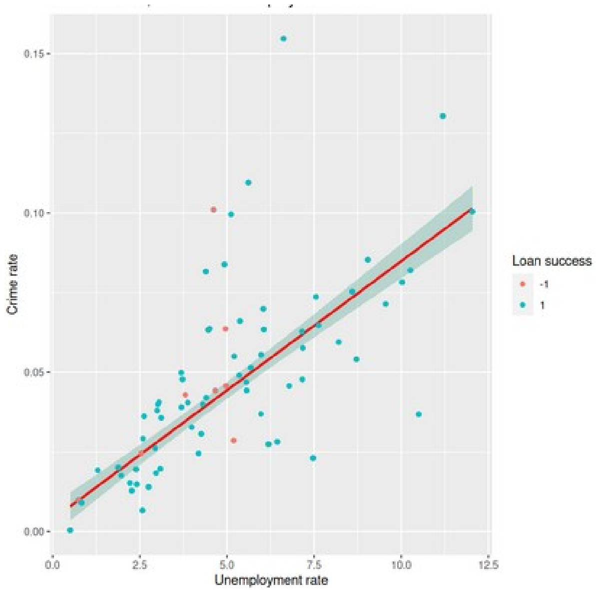


Successful and unsuccessful loans

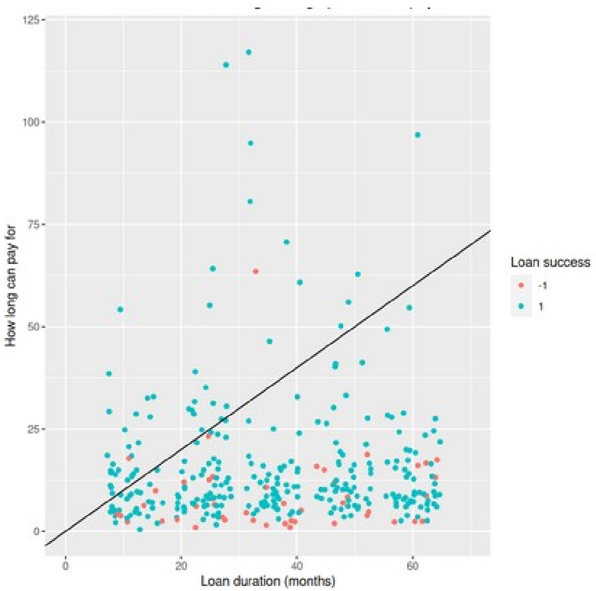
Data Understanding



Loan success and distribution of amount loaned



Loan success, crime and unemployment rates



Loan success and how long average person can pay it

Problem definition

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

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

Data preparation

In order to prepare the data to run the models and make predictions, we need to clean it and manipulate the attributes in order to improve it.

- Parse the date from the YYMMDD format to YYYY-MM-DD and calculate a new atribute **age_days**, which stores the age of account in days.
- Transform the attribute **birthnumber** into **gender** and **birthdate**, calculate the **client's age** and drop the **client's district**, only considering the account district.
- Rename some columns on district dataset, fill the empty values with the column average and transform the data from 1995 and 1996 into and average of both and an attribute that tells if those numbers grew between years.
- Remove columns with over 70% of null values.
- Change string variables to integers, helping later steps.

```
# Make date more readable
account_data <- transform(account_data, acc_creation_date = as.Date(
  paste(
    paste("19", date %/% 10000, sep = ""),
    (date %/% 100) %% 100,
    date %% 100,
    sep = "-"
  ),
  format = "%Y-%m-%d"
))
```

```
replace(client_data, (client_data == "" | client_data == " "), NA)

client_data <- transform(client_data,
  gender = ifelse(((birth_number %/% 100) %% 100) <= 12, 0, 1)
)

client_data <- transform(client_data, birthday = as.Date(
  paste(
    paste("19", birth_number %/% 10000, sep = ""),
    ifelse(((birth_number %/% 100) %% 100) <= 12,
      (birth_number %/% 100) %% 100,
      ((birth_number %/% 100) %% 100) - 50
    ),
    birth_number %% 100,
    sep = "-"
  ),
  format = "%Y-%m-%d"
))
```

```
remove_empty_cols <- function(data) {
  result <- data %>% select(where(~ mean(is.na(.)) < 0.7))
  return(result)
}
```

Data preparation

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

```r
data <- loan_data %>%
  rename(loan_date = date) %>%
  left_join(account_data, by = "account_id") %>%
  left_join(trans_data, by = "account_id") %>%
  mutate(transactions_net = transactions_net / age_days) %>%
  mutate(sanctions_rate = sanctions / age_days) %>%
  rename(daily_transactions_net = transactions_net) %>%
  left_join(disp_data, by = "account_id") %>%
  filter(type == "OWNER") %>%
  select(-age_days, -type, -sanctions) %>%
  left_join(card_data, "disp_id") %>%
  mutate(has_card = ifelse(!is.na(card_id), 1, 0)) %>%
  mutate(is_gold = ifelse((!is.na(type) & type == "gold"), 1, 0)) %>%
  select(-card_id, -type, -issued) %>%
  left_join(client_data, by = "client_id") %>%
```

- Aggregate the transactions dataframe by account_id, creating new variables to describe number of transactions per account (ex: **trans_count**), count credits and withdrawals (ex: **withdrawal_count**), amount stats (ex: **biggest_transaction**), balance stats (ex: **times_negative_balance**) and operation ratios (ex: **collection_bank_ratio**)
- Join every table and derive new attributes as **transactions_net** (is now the net amount of money transacted per day), **sanctions_rate** (number of sanctions per day), **can_afford_loan** (whether or not the client's district has an average salary that can cover the loan's monthly payments) or **acc_age_when_loan** (how old the account was when the loan was made)

Experimental Setup

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

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

```python
SPLITTER = TimeSeriesSplit()

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

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

    return grid_search.best_estimator_
```

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

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

    y_pred_normalized = np.argmax(model.predict_proba(X_test), axis=1)

    cm = metrics.confusion_matrix(y_test, y_pred_normalized)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap="Blues")

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

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

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

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

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

    submition_data = pd.DataFrame()

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

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

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

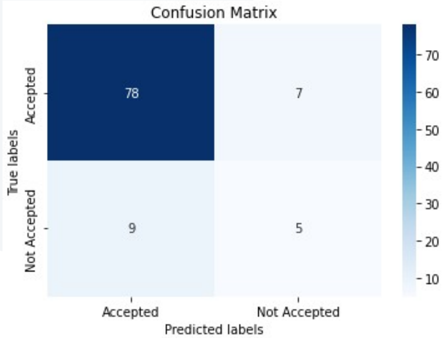
```python
def apply(model, params):
    best_model = train_model_grid(model, params)
    predicted_results = evaluate(best_model)
    export_results(best_model, best_model.__class__.__name__)

    return predicted_results
```

Results

Decision Tree

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

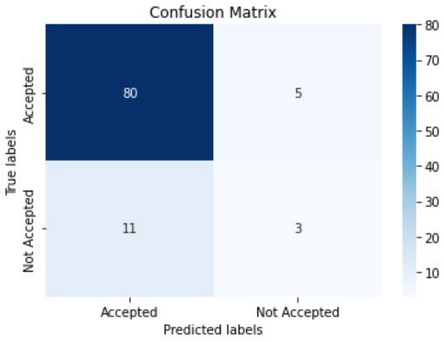
- Best params for DecisionTreeClassifier: {'criterion': 'gini', 'max_depth': 19, 'min_samples_leaf': 5, 'min_samples_split': 7}
- Best Score: 0.825073704073704
- AUC Score on Testing Data: 0.6516806722689076



Random Forest

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

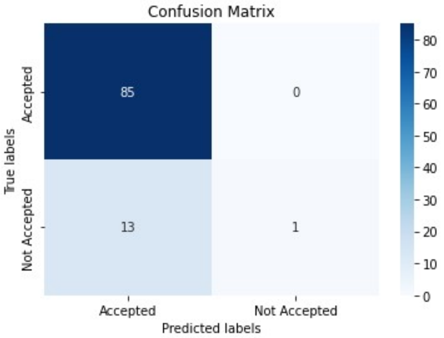
- Best params for RandomForestClassifier: {'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 150}
- Best Score: 0.8726257076257076
- AUC Score on Testing Data: 0.8529411764705882



Support Vector Machine

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

- Best params for SVC: {'C': 10, 'degree': 1, 'kernel': 'poly'}
- Best Score: 0.7356499056499056
- AUC Score on Testing Data: 0.7680672268907563

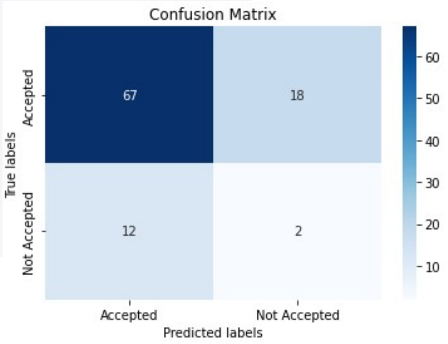


Results

MLP

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

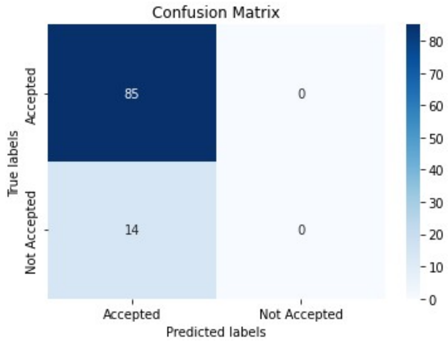
- Best params for MLPClassifier: {'alpha': 0.5, 'hidden_layer_sizes': 30}
- Best Score: 0.6599816294816295
- AUC Score on Testing Data: 0.4302521008403361



K-Nearest Neighbors

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

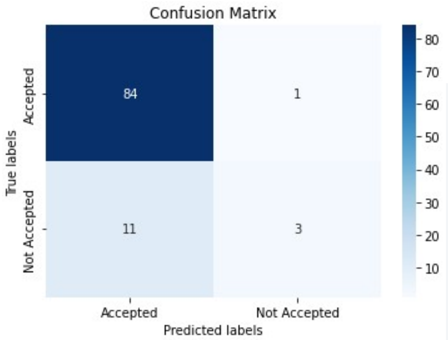
- Best params for RandomForestClassifier: KNeighborsClassifier: {'n_neighbors': 28, 'weights': 'distance'}
- Best Score: 0.6755937395937396
- AUC Score on Testing Data: 0.5785714285714285



Gradient Boosting

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

- Best params for GradientBoostingClassifier: {'max_depth': 5, 'max_features': 7, 'min_samples_leaf': 30, 'min_samples_split': 200}
- Best Score: 0.5
- AUC Score on Testing Data: 0.8050420168067227



Conclusions

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

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

Annexes

```
np.random.seed(2019)

def roc_curve_and_score(y_test, pred_proba):
    fpr, tpr, _ = metrics.roc_curve(y_test.ravel(), pred_proba.ravel())
    roc_auc = metrics.roc_auc_score(y_test.ravel(), pred_proba.ravel())
    return fpr, tpr, roc_auc


plt.figure(figsize=(10, 8))
matplotlib.rcParams.update({'font.size': 14})
plt.grid()

fpr, tpr, roc_auc = roc_curve_and_score(y_test, rf_results)
plt.plot(fpr, tpr, color='darkorange', lw=2, label='RFC({0:.3f})'.format(roc_auc))

fpr, tpr, roc_auc = roc_curve_and_score(y_test, svm_results)
plt.plot(fpr, tpr, color='red', lw=2, label='SVM({0:.3f})'.format(roc_auc))

fpr, tpr, roc_auc = roc_curve_and_score(y_test, knn_results)
plt.plot(fpr, tpr, color='crimson', lw=2, label='KNN({0:.3f})'.format(roc_auc))

fpr, tpr, roc_auc = roc_curve_and_score(y_test, dt_results)
plt.plot(fpr, tpr, color='yellow', lw=2, label='DTC({0:.3f})'.format(roc_auc))

fpr, tpr, roc_auc = roc_curve_and_score(y_test, mlb_results)
plt.plot(fpr, tpr, color='black', lw=2, label='MLP({0:.3f})'.format(roc_auc))

plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.legend(loc="lower right")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1 - Specificity')
plt.ylabel('Sensitivity')
plt.show()
```