Roy Madpis - 319091526
Rawaa Makhoul - 316114370
Alexandra Fatyeyeva - 336540950

Peer to peer lending project

Working Paper № 5 – Team A

Introduction

In the previous stages we formulated our modeling approach: first, we created a new feature showing whether the realized return is higher or lower than 2%. Then, we built a python script for generating the model, chose XGBoost as our algorithm and started selecting the best features and tuning the model's hyperparameters. In this stage we continue to fine-tune our model and evaluate its performance.

Fine-tune our model

1. Feature engineering

We tried to create several new features to improve our model's performance. We created a feature importance graph to decide which features we should manipulate (see in the appendix *Feature importance graph*). Some of those attempts did increase the accuracy and the NPV (Negative Predictive Value) and thus are used in our final model. All the attempts are summarized in the **research log** as well as in the appendix *Feature engineering*.

Note: we used cross-validation (with k = 5) to estimate our model's performance. Therefore, only if a new feature was able to increase the accuracy's or the NPV's mean – was regarded as a "contributing" new feature and, as a result, used in the final model.

2. Feature Selection

We performed backward feature selection[1] to find features that are not relevant for the model, i.e., lowers its accuracy. The feature selection was conducted twice – before and after the feature engineering; the goal was to check whether the new features indeed improved the model's performance. See in the appendix *Feature selection* for the output and the process of the work.

3. Parameter Optimization

XGBoost has many hyperparameters that can be adjusted to optimize the model's performance. Due to computation time, we couldn't use gridsearch (meaning, to try all the combinations of the hyperparameters altogether). Instead, we used HalvingGridSearch[2] - a function that enables frugally trying different combinations of hyperparameters: the function starts evaluating all the "candidates" with a small number of resources, then iteratively selects the best candidates of hyperparameters while using more and more resources.

Note: This method is less expensive in computation time. However, it can't guarantee getting the best combination of hyperparameters.

[1] Using the function SequentialFeatureSelector from mlxtend.feature_selection
[2] We used the function HalvingGridSearchCV from sklearn.model_selection.

Evaluating our model's performance

We can measure our model's performance from 2 main aspects:

1. **The data-science approach**: Measuring the Accuracy, Recall, F1-Score, NPV, and AUC of the model.
2. **The Business approach**: Measuring the weighted average realized return that the company would have earned if invested in the loans that our model suggests (the loans in the suggested portfolio).

In this stage, we only present an evaluation for the data-science approach.

Reminder: Our model classifies the loans into two classes.

- "0" – is for loans with a realized return higher than 2% - this is the "common class".
- "1" – is for loans with a realized return lower than 2%.

As we mentioned in *Working Paper #4,* the two main data-science metrics that are most important are Accuracy and NPV (Negative Predictive Value).
It is important to mention that even if our model has a smaller accuracy than the accuracy of a random classifier, it might still outperform the random model from the business aspect - it might generate a higher realized return. In this notion, we wish to maximize the NPV as it directly relates to the total weighted realized return gained.[3]

To evaluate how well our model performs in terms of accuracy and NPV, we chose to compare it to a random model that classifies all loans to be "realized return higher than 2%" (the common class).
The accuracy and NPV of the random classifier are simply the base probability of that common class, which is 71.76%. Therefore, these metrics of our model must be higher than that.
In fact, the **accuracy** and **NPV** of our model (the mean on 5 folds) are 71.95% and 72.19%, respectively – meaning our model slightly outperforms the random model.

See in the appendix the *Final performance of the model* for the final metrics values our best model yielded and the final ROC and Precision X Recall graphs.

To confirm that our results are valid and weren't influenced by a specific test set, we used a function we created *cross_val_RAR*[4]. The cross-validation results are quite constant, which means that the performance wasn't influenced by a particular test set and thus is valid.

Conclusion:

In this step we optimized and fine-tuned our XGBoost model, the model classifies loans into: "realized return higher than 2%" (0) or "realized return lower than 2%" (1). After performing the classification, our model returns a "suggested portfolio" - which is a list of filtered loans that our algorithm advises to invest in.
We also presented the model's performance regarding the data-science metrices.
In the next and final step, we will answer the company's questions and provide ways to apply the results.

[3] The higher the NPV, the more TN cases (more true "higher than 2%"cases) and less FN (less false "higher than 2%" cases), [we are most "afraid" of those FN cases – as they can be costly mistakes].

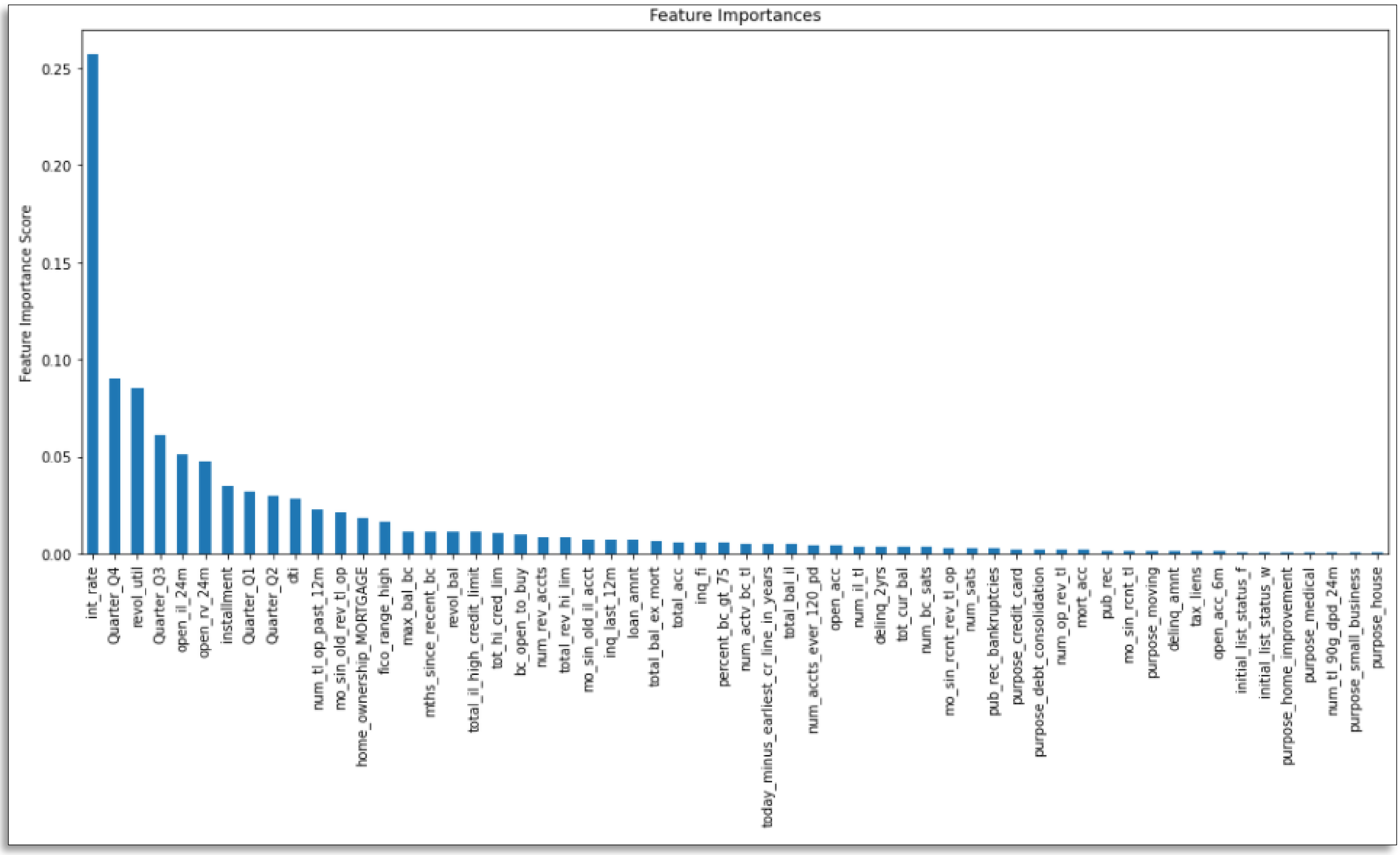
[4] Its output can be seen in the appendix in *"Using the Function: cros_val_RAR"*

Appendix

❖ Feature Importance Graph & Feature Engineering:

This section presents the importance graph we got from the XGBoost model before and after the feature engineering. In this way, we can observe whether the new features we created have importance according to the XGBoost.

Feature Importance Graph (before the feature engineering):



The Highest 20 Features regarding the feature importance:

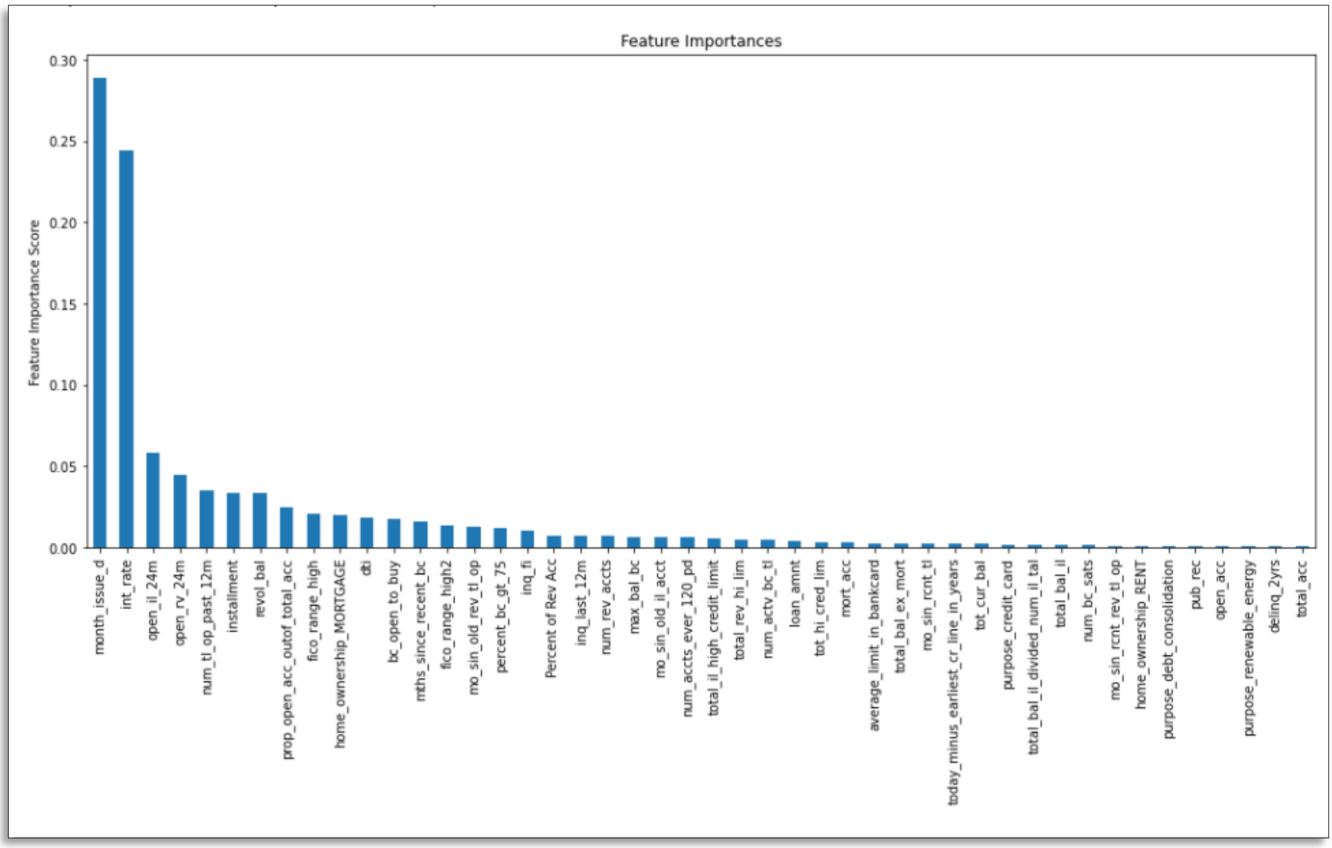
```
feat_imp.head(20)

int_rate                      0.257399
Quarter_Q4                    0.090001
revol_util                    0.085222
Quarter_Q3                    0.061200
open_il_24m                   0.051367
open_rv_24m                   0.047874
installment                   0.035176
Quarter_Q1                    0.032349
Quarter_Q2                    0.030088
dti                           0.028226
num_tl_op_past_12m            0.022541
mo_sin_old_rev_tl_op          0.021657
home_ownership_MORTGAGE       0.018390
fico_range_high               0.016041
max_bal_bc                    0.011661
mths_since_recent_bc          0.011640
revol_bal                     0.011399
total_il_high_credit_limit    0.011371
tot_hi_cred_lim               0.010464
bc_open_to_buy                0.010328
```

Feature Engineering:

In this stage, we tried to create several new features, most of which didn't yield a much better performing model. We present the feature engineering we tried - the ones that are marked with green are the features that increase the model's performance and thus are used in the final model (see the *Research Log* for additional information):

1. average_limit_in_banckard = The quotient of total_bc_limit and num_bc_tl (**total_bc_limit** = Total bankcard high credit/credit limit ; **num_bc_tl** = Number of bankcard accounts).
2. fico_range_high2 = fico_range_high^2
3. **Prop_annual_inc_total_hi_credit_limit** = The proportion of the annual income out of the total high credit limit
4. Prop_open_acc_outof_total_acc = The quotient of open_acc and total_acc **(open_acc** = The number of open credit lines in the borrower's credit file; **total_acc** = The total number of credit lines currently in the borrower's credit file)
5. **prop_cur_bal_outof_open_acc**= The quotient of tot_cur_bal and open_acc **(tot_cur_bal** = Total current balance of all accounts ; **open_acc**= The number of open credit lines in the borrower's credit file)
6. **All_accounts =** Combining num_il_tl, num_bc_tl and num_rev_accts (num_il_tl = Number of installment accounts; num_bc_tl = Number of bankcard accounts; num_rev_accts = Number of revolving accounts)
7. **Prop_satisfactory_bc_outof_active_bc =** The quotient of num_bc_sats and num_actv_bc_tl (num_bc_sats = Number of satisfactory bankcard accounts ; num_actv_bc_tl = Number of currently active bankcard accounts)
8. **The proportion of the current balance of all installment accounts and the Total installment limit:** total_bal_il / total_il_high_credit_limit (Total_bal_il = Total current balance of all installment accounts ; Total_il_high_credit_limit / = Total installment high credit/credit limit)
9. Total_bal_il_divided_num_il_tal - The proportion of the current balance of all installment accounts and the Number of installment accounts : **total_bal_il / num_il_tl** (**total_bal_il** = Total current balance of all installment accounts ; **num_il_tl** = Number of installment accounts)
10. **Average limit in an installment account** = total_il_high_credit_limit / num_il_tl (**total_il_high_credit_limit** = Total installment high credit/credit limit ; **num_il_tl** = Number of installment accounts)
11. **The product of open_il_24m and open_rv_24m** (**open_il_24m** = Number of installment accounts opened in past 24 months ; **open_rv_24m** = Number of revolving trades opened in past 24 months)
12. month_issue_date **-** Creating a column that contains the month of the issue date (extracting the month from the issue_d column).
13. emp_length **-** We "collapsed" the feature "emp_length" into 2 levels: "5 or lower" = emp_length that is equal or lower than 5, and "6 or higher" = emp_length that is equal or higher than 6. This action improved the model's performance.

Feature importance graph (after feature engineering):



<u>The Highest 20 Features regarding the feature importance:</u>

```
feat_imp.head(20)
```

```
month_issue_d                    0.2551
int_rate                         0.2193
open_il_24m                      0.0534
open_rv_24m                      0.0395
installment                      0.0376
revol_bal                        0.0340
num_tl_op_past_12m               0.0316
prop_open_acc_outof_total_acc    0.0281
dti                              0.0224
bc_open_to_buy                   0.0223
fico_range_high                  0.0204
home_ownership_MORTGAGE          0.0179
mo_sin_old_rev_tl_op             0.0175
mths_since_recent_bc             0.0155
fico_range_high2                 0.0137
percent_bc_gt_75                 0.0135
inq_fi                           0.0111
mo_sin_old_il_acct               0.0110
Percent of Rev Acc               0.0105
total_rev_hi_lim                 0.0088
```

❖ Feature Selection:

We performed the feature selection twice: before and after the feature engineering

- **Before the feature engineering;**

We performed feature selection Using the function *SequentialFeatureSelector* from *mlxtend.feature_selection*. As of high computation time, in the first three executions, we used only 2 cross-validation folds, used XGBoost with 50 estimators, and executed the feature selection in small steps. We now present the results we got in each execution of the function:

1st execution:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	column_reduced
74	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7185405718142809, 0.7186249457596066]	0.718583	(loan_amnt, int_rate, installment, dti, delinq...	0.000181516	4.2187e-05	4.2187e-05	0
73	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7186852128634106, 0.7186249457596066]	0.718655	(loan_amnt, int_rate, installment, dti, delinq...	0.000129654	3.01336e-05	3.01336e-05	mort_acc
72	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7189142278578661, 0.7186249457596066]	0.71877	(loan_amnt, int_rate, installment, dti, delinq...	0.00062234	0.000144641	0.000144641	purpose_moving
71	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7189142278578661, 0.7186852128634106]	0.7188	(loan_amnt, int_rate, installment, dti, delinq...	0.000492686	0.000114507	0.000114507	delinq_amnt

Total time of execution: 10.083 Hours

Number of features given: 74

The mean accuracy score of the validation with best features: 0.7188

The features to remove: **mort_acc, delinq_amnt**

2nd execution:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	column_reduced
72	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7186852128634106, 0.7186852128634106]	0.718685	(loan_amnt, int_rate, installment, dti, delinq...	0	0	0	0
71	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7189142278578661, 0.7186852128634106]	0.7188	(loan_amnt, int_rate, installment, dti, delinq...	0.000492686	0.000114507	0.000114507	purpose_moving
70	(0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	[0.7189142278578661, 0.7186972662841714]	0.718806	(loan_amnt, int_rate, installment, dti, fico_r...	0.000466755	0.000108481	0.000108481	delinq_2yrs
69	(0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	[0.7189142278578661, 0.7186972662841714]	0.718806	(loan_amnt, int_rate, installment, dti, fico_r...	0.000466755	0.000108481	0.000108481	initial_list_status_w

Total time of execution: 7.293 Hours

Number of features given: 72

The mean accuracy score of the validation with best features: 0.718806

The features to remove: **delinq_2yrs, initial_list_status**

3rd execution:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	column_reduced
69	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7186852128634106, 0.7186731594426499]	0.718679	(loan_amnt, int_rate, installment, dti, fico_r...	2.59308e-05	6.02671e-06	6.02671e-06	0
68	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7189142278578661, 0.7186731594426499]	0.718794	(loan_amnt, int_rate, installment, dti, fico_r...	0.000518617	0.000120534	0.000120534	purpose_moving
67	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7189142278578661, 0.7187093197049322]	0.718812	(loan_amnt, int_rate, installment, dti, fico_r...	0.000440824	0.000102454	0.000102454	num_accts_ever_120_pd
66	(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14...	[0.7188057470710187, 0.7188780675955836]	0.718842	(loan_amnt, int_rate, installment, dti, fico_r...	0.000155585	3.61603e-05	3.61603e-05	total_acc
65	(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14...	[0.7188780675955836, 0.7188780675955836]	0.718878	(loan_amnt, int_rate, installment, dti, fico_r...	0	0	0	purpose_house

Total time of execution: 9.7538 Hours

Number of features given: 69

The mean accuracy score of the validation with best features: 0.718878

The features to remove: **num_accts_ever_120_pd, total_acc**

4^{th} execution:

In the fourth execution, we used 2 cross-validation folds and used XGBoost with **250** estimators:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	column_reduced
71	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7192517236391688, 0.718661106021889]	0.718956	(loan_amnt, int_rate, installment, dti, delinq...	0.00127061	0.000295309	0.000295309	0
70	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7192878839014513, 0.7191914565353648]	0.71924	(loan_amnt, int_rate, installment, dti, delinq...	0.000207447	4.82137e-05	4.82137e-05	Quarter_Q1
69	(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14...	[0.7195410057374283, 0.7191552962730823]	0.719348	(loan_amnt, int_rate, installment, dti, delinq...	0.000829787	0.000192855	0.000192855	revol_util
68	(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14...	[0.7195168988959066, 0.7192999373222121]	0.719408	(loan_amnt, int_rate, installment, dti, delinq...	0.000466755	0.000108481	0.000108481	percent_bc_gt_75
67	(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14...	[0.7196494865242756, 0.7192999373222121]	0.719475	(loan_amnt, int_rate, installment, dti, delinq...	0.000751994	0.000174775	0.000174775	purpose_major_purchase
66	(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14...	[0.7197218070488405, 0.7192999373222121]	0.719511	(loan_amnt, int_rate, installment, dti, delinq...	0.000907579	0.000210935	0.000210935	home_ownership_ANY

Total time of execution: 2 days

Number of features given: 71

The mean accuracy score of the validation with best features: 0.7195

Important Note: There wasn't a significant improvement in the model's performance when we tried to remove those features (the features that the feature selection function outputted.)

- **After the feature engineering:**

We used **3** cross-validation folds and used XGBoost with **100** estimators:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err	column_reduced
70	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.719803269202951, 0.7189664213514637, 0.7195...	0.719444	(loan_amnt, int_rate, installment, dti, delinq...	0.000792	0.000352	0.000249	0
69	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7201106610733401, 0.7194003941919969, 0.719...	0.719601	(loan_amnt, int_rate, installment, dti, delinq...	0.000817	0.000363	0.000257	open_rv_24m
68	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7200925791986114, 0.7194003941919969, 0.719...	0.719607	(loan_amnt, int_rate, installment, dti, delinq...	0.000775	0.000345	0.000244	purpose_vacation
67	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7201106610733401, 0.7194003941919969, 0.719...	0.719613	(loan_amnt, int_rate, installment, dti, delinq...	0.000795	0.000353	0.00025	purpose_major_purchase
66	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7200925791986114, 0.7194003941919969, 0.719...	0.719607	(loan_amnt, int_rate, installment, dti, delinq...	0.000775	0.000345	0.000244	purpose_other
65	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7200925791986114, 0.7194003941919969, 0.719...	0.719607	(loan_amnt, int_rate, installment, dti, delinq...	0.000775	0.000345	0.000244	purpose_car
64	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7201106610733401, 0.7194003941919969, 0.719...	0.719613	(loan_amnt, int_rate, installment, dti, delinq...	0.000795	0.000353	0.00025	num_tl_90g_dpd_24m
63	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7200925791986114, 0.7194003941919969, 0.719...	0.719607	(loan_amnt, int_rate, installment, dti, delinq...	0.000775	0.000345	0.000244	home_ownership_ANY
62	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7200021698249675, 0.71967162721733, 0.71916...	0.719613	(loan_amnt, int_rate, installment, dti, delinq...	0.000774	0.000344	0.000243	max_bal_bc
61	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	[0.7199840879502387, 0.71967162721733, 0.71925...	0.719637	(loan_amnt, int_rate, installment, dti, delinq...	0.000671	0.000298	0.000211	purpose_medical

Total time of execution: ~3 days

Number of features given: 70

The mean accuracy score of the validation with best features: 0.7196

Note: The new features we created (in the feature engineering section) weren't found as "unnecessary" according to the feature selection function. This is an indication that the features we created indeed contribute to the model's performance.

❖ Final performance of the model:

In this section, we present the final results of the XGBoost model that predicts the label of the categorical target variable: realized_return_2% (predicting whether a loan will have a realized return that is lower (True) or Higher (False) than 2%).

- **The Metrics table** got from the function model_all for the best performing model – XGBoost (threshold = 0.75);

	XG Boost
Accuracy	0.7196
Precision	0.5477
Recall	0.0406
F1 Score	0.0757
Negative_Predictive_Value	0.7233
AUC	0.6354
Num of FN	19261.0000
Num of TN	50354.0000
Realized Return SP	2.3438

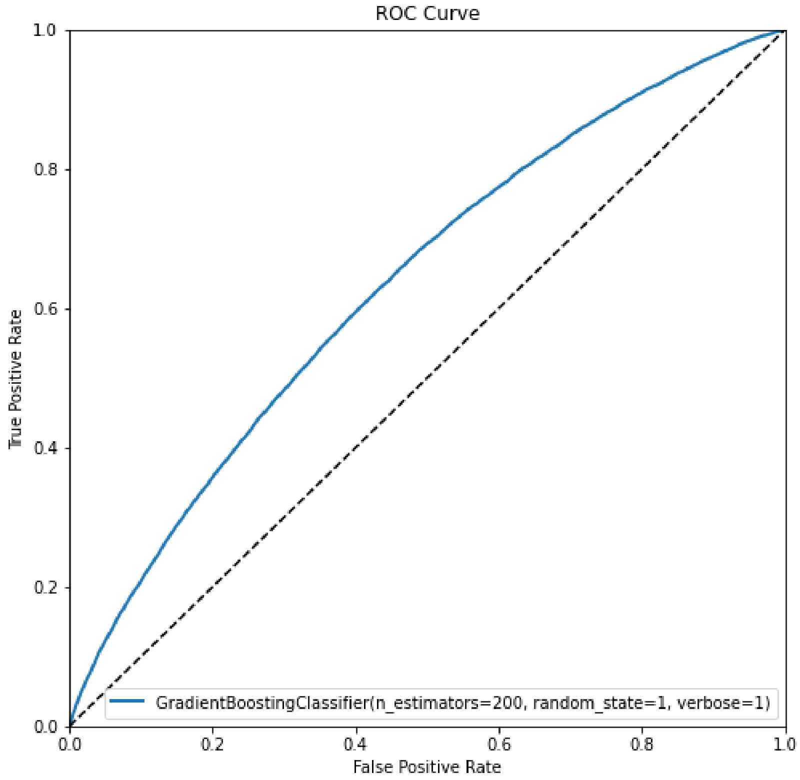
- **The Metrics table got from the function: cross_validation_RAR** for the best performing model – XGBoost (threshold = 0.75):

Statistics of the model for 5 folds with threshold of 0.75:

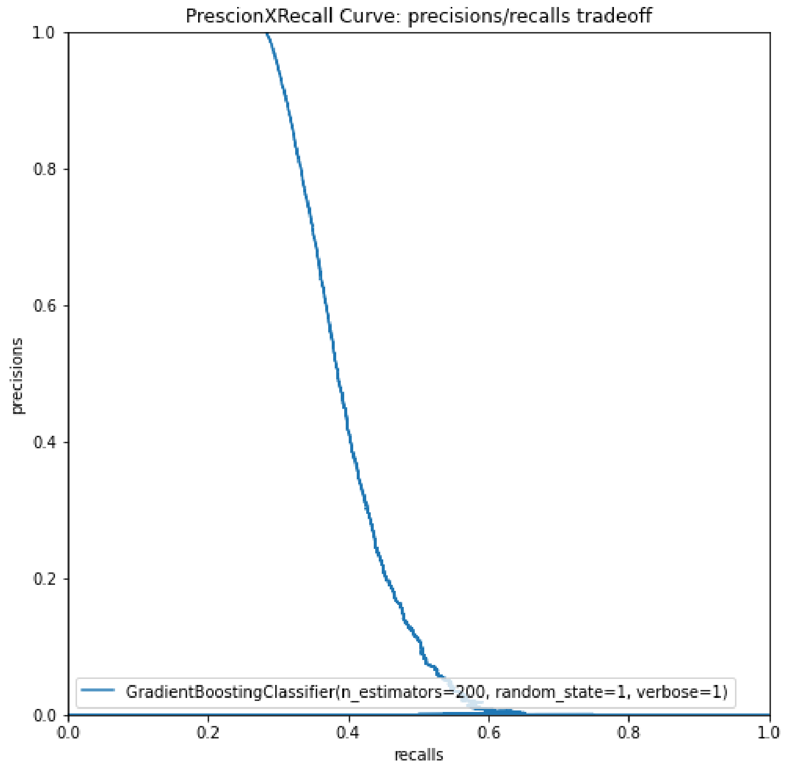
	Min	Max	Mean	Sd
Accuracy	0.718010	0.722380	0.719348	0.001752
Precision	0.521067	0.563877	0.540750	0.018515
Recall	0.037270	0.042249	0.040303	0.001939
F1 Score	0.069603	0.078346	0.075010	0.003472
Negative_Predictive_Value	0.721902	0.725701	0.723190	0.001543
AUC	0.631396	0.635464	0.632872	0.001726
Realized Return SP	2.301043	2.370801	2.348254	0.028843

- The ROC Curve got from the function: roc_curves_RAR

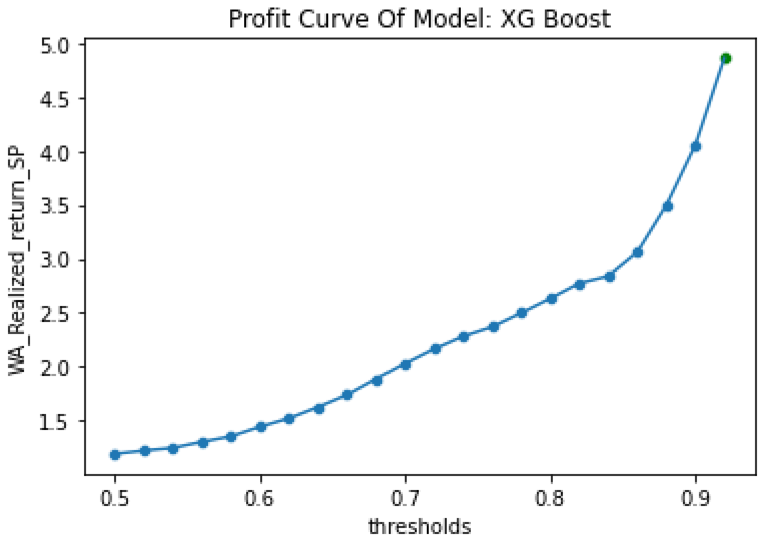
	AUC Value
GradientBoostingClassifier(n_estimators=200, random_state=1, verbose=1)	0.6354



- The Precision VS Recall Curve got from the function: Precision_VS_Recall_RAR



- **The Profit Curve of the XGBoost model** got from the function: [profit_curve]



The matching "threshold table" of the profit curve: (presenting for each threshold the weighted average realized return of the suggested portfolio as well as the number of loans, the total investment in thousands of dollars, the number of FN and TN yield):

	thresholds	WA_Realized_return_SP	number of loans	Total investment in thousands of $	Number of FN	Number of TN
0	0.5000	1.1894	69615.0000	910831.3570	19261.0000	50354.0000
1	0.5200	1.2195	68971.0000	901522.3070	18941.0000	50030.0000
2	0.5400	1.2448	68185.0000	890849.4820	18583.0000	49602.0000
3	0.5600	1.3007	67115.0000	876351.4820	18065.0000	49050.0000
4	0.5800	1.3513	65700.0000	857311.7570	17454.0000	48246.0000
5	0.6000	1.4405	63637.0000	830272.0250	16598.0000	47039.0000
6	0.6200	1.5194	60890.0000	795966.0000	15524.0000	45366.0000
7	0.6400	1.6236	57495.0000	754610.9750	14227.0000	43268.0000
8	0.6600	1.7364	53248.0000	702290.8000	12740.0000	40508.0000
9	0.6800	1.8846	48309.0000	641297.0250	11071.0000	37238.0000
10	0.7000	2.0301	42790.0000	572988.9250	9354.0000	33436.0000
11	0.7200	2.1649	37243.0000	503835.7750	7705.0000	29538.0000
12	0.7400	2.2829	31742.0000	434507.1750	6187.0000	25555.0000
13	0.7600	2.3703	26231.0000	365302.9500	4831.0000	21400.0000
14	0.7800	2.4988	20601.0000	292780.3500	3562.0000	17039.0000
15	0.8000	2.6326	14845.0000	216308.8250	2348.0000	12497.0000
16	0.8200	2.7738	9181.0000	139178.3750	1310.0000	7871.0000
17	0.8400	2.8396	4352.0000	70129.3750	539.0000	3813.0000
18	0.8600	3.0607	1336.0000	23543.5000	139.0000	1197.0000
19	0.8800	3.4959	171.0000	3188.2500	12.0000	159.0000
20	0.9000	4.0558	6.0000	125.8000	0.0000	6.0000
21	0.9200	4.8641	1.0000	3.6000	0.0000	1.0000

- **Final baseline grades table comparison:**

The baseline grade table with **all the loans**	The baseline grade table - of our XGBoost model with a threshold of 0.75

Base line Grades - all the loans

	weighted_average	weighted SD	Number of Loans
A	1.861573	4.514966	56753
B	1.532950	7.149496	92005
C	0.798874	9.438672	74610
D	-0.194200	11.545009	30543
E	-1.022335	13.340783	8798
F	-2.109506	14.559178	2114
G	-3.628924	15.255247	493
Total All loans Together	1.076391	8.414820	265316

Base line Grades - Suggested portfolio with threshold of: 0.75 XGBOOST

	weighted_average	weighted SD	Number of Loans
A	2.1611	4.3687	9006
B	2.3072	6.4866	13615
C	2.6130	8.1188	5728
D	3.5900	9.4213	548
E	6.1465	9.0121	70
F	3.2977	10.7355	3
Total All Suggested Portfolio	2.3438	6.3050	28970

- **Using the Function: cross_val_RAR**

Statistics of the model for 5 folds with threshold of 0.75:

	Min	Max	Mean	Sd
Accuracy	0.718010	0.722380	0.719348	0.001752
Precision	0.521067	0.563877	0.540750	0.018515
Recall	0.037270	0.042249	0.040303	0.001939
F1 Score	0.069603	0.078346	0.075010	0.003472
Negative_Predictive_Value	0.721902	0.725701	0.723190	0.001543
AUC	0.631396	0.635464	0.632872	0.001726
Realized Return SP	2.301043	2.370801	2.348254	0.028843

Note how the standard deviation is quite low for every metric, meaning the model performs quite the same for different test folds.

Cross-validation on the "baseline grades table":

Statistics of Realized return by grade for 5 folds with threshold of 0.75:

	Weighted Average all folds				Standard Deviation all folds				Number of Loans all folds			
	mean	min	max	sd	mean	min	max	sd	mean	min	max	sd
A	2.216684	2.153692	2.294094	0.062531	4.166257	3.939191	4.346233	0.194265	4234.6	4206	4296	39.227541
B	2.328109	2.254273	2.390866	0.059193	6.367769	6.271418	6.548693	0.119580	6299.6	6192	6412	78.697522
C	2.488098	2.280980	2.700446	0.164325	8.185954	7.794123	8.522023	0.274836	2683.2	2610	2764	63.888184
D	3.602997	3.028351	4.061545	0.481267	9.520894	9.091388	10.384430	0.537435	268.6	253	291	15.630099
E	3.800345	1.615217	6.142001	2.118742	11.959265	10.495964	13.778108	1.358434	36.8	28	47	7.049823
F	10.084597	6.949847	12.110476	2.001830	3.560709	1.550174	5.443511	1.761959	3.4	2	5	1.140175
Total All Suggested Portfolio	2.348254	2.301043	2.370801	0.028843	6.241215	6.182338	6.339383	0.059021	13526.4	13398	13656	103.910057