Credit Risk Predictive Modelling

ID/X Partners – Rakamin Academy Virtual Internship Experience Final Task

Created by:
Kenneth Wahyudi, S.Si.

kennethwahyudi48@gmail.com
https://www.linkedin.com/in/kenneth-wahyudi-80b886209/

I am an aspiring data scientist with interest in machine learning and paddling through the data lake. I have experiences as a research intern in BATAN, and several data science and machine learning projects under my belt. Specifically intermediate to advanced knowledge in Exploratory Data Analysis, Data Pre-processing, Supervised & Unsupervised Learning, as well as Data Visualization and Storytelling. And I am always looking forward to exploring new and uncharted territories that might allow myself to grow and improve.

Supported by:
Rakamin Academy
Career Acceleration School
www.rakamin.com

Overview

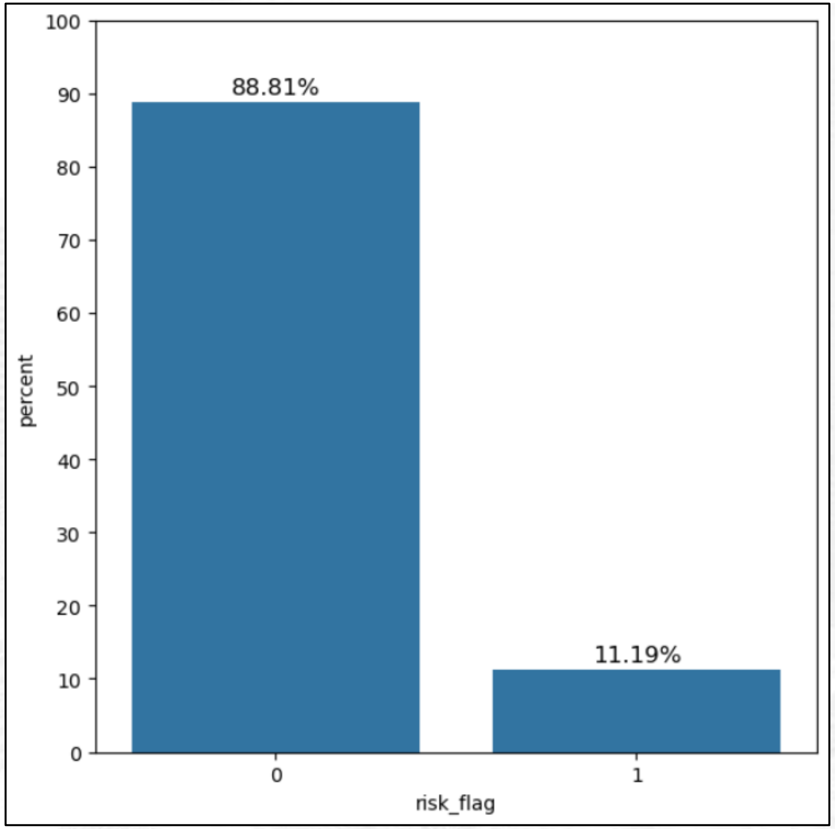
Welcome to a captivating journey into the world of modern finance. In the next few moments, we will explore the art and science behind developing a credit risk predictive model, offering insights to enhance your financial acumen. Fasten your seatbelts as we delve into the complexities of credit risk modeling, where numbers meet narratives, and the future of finance comes into sharper focus. The dataset contains 466285 rows, and customer's loan information from 2007-2014. The objective is to create a predictive model that would solve the problem of defaulting customers.

All results are from outputs of codes written in Python and its packages. JupyterLab was the GUI used in this project.

The dataset used in this project can be found in the following link.

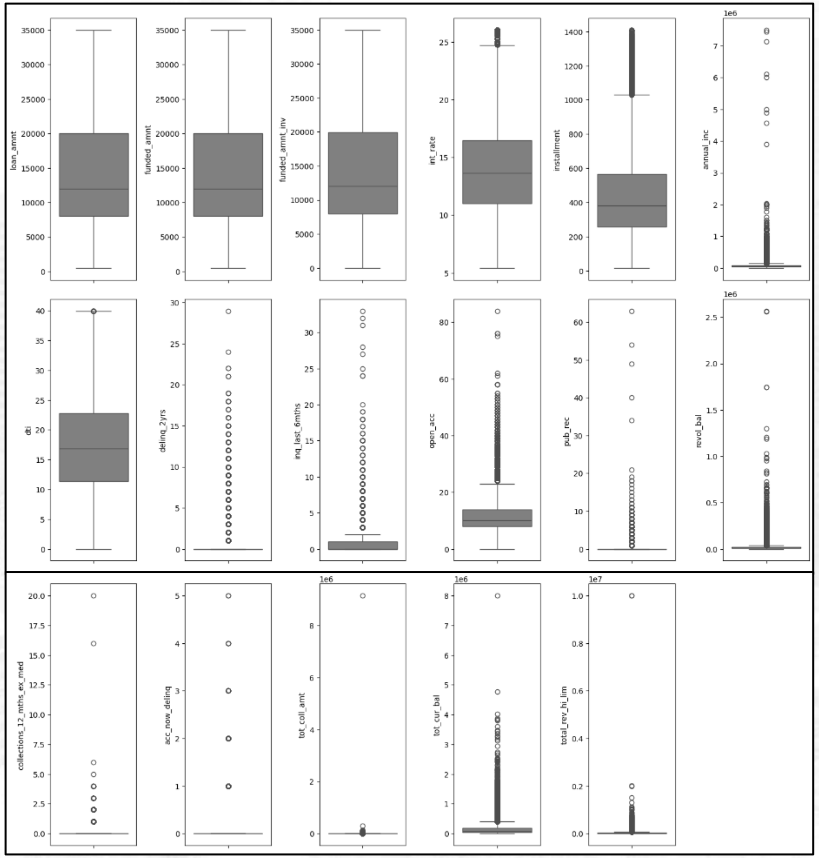
The full code in .ipynb file format can be found in the following link. And in .py file format can be found in the following link.

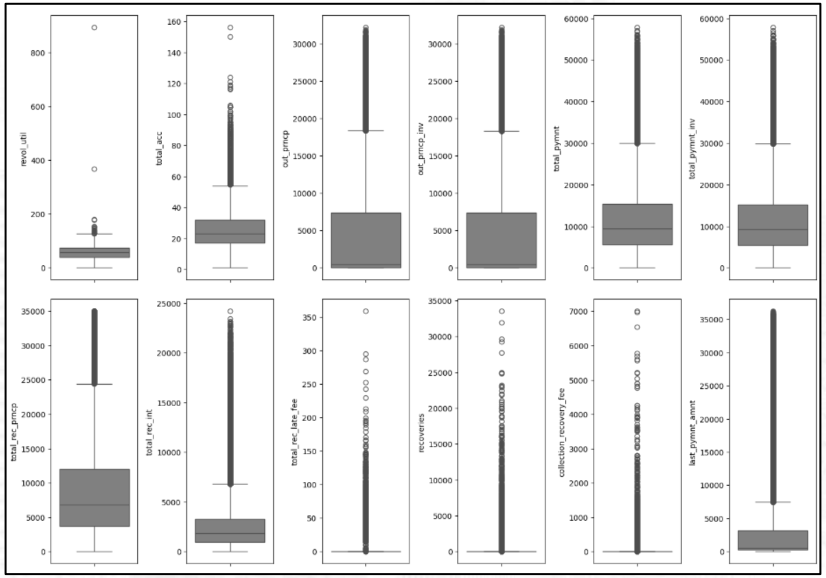
Exploratory Data Analysis



As we can see from the countplot to the left, the target classes are imbalanced. With at risk customers being only about 11.2% of the total dataset, and non-risk customers being an overwhelming 88.8% of the dataset. This means that we need to handle these imbalances later on before modelling.

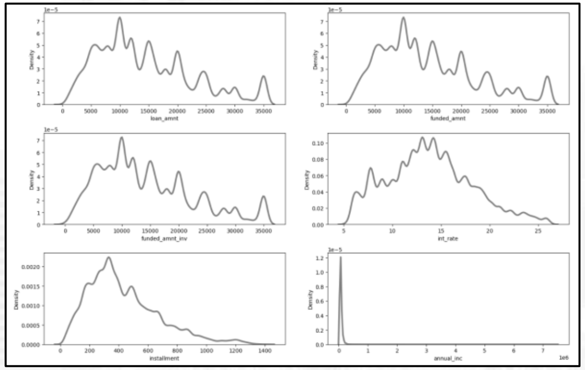
Exploratory Data Analysis

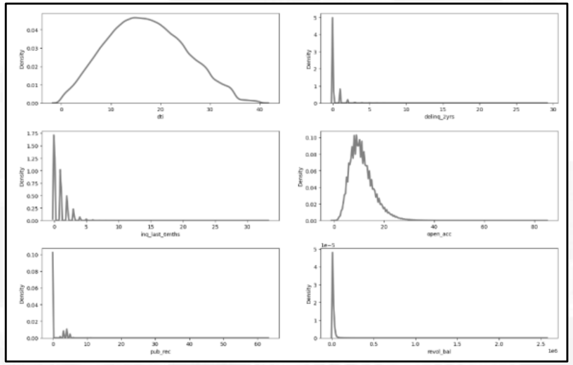


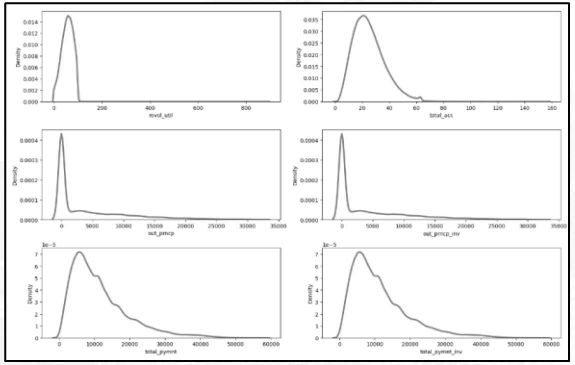


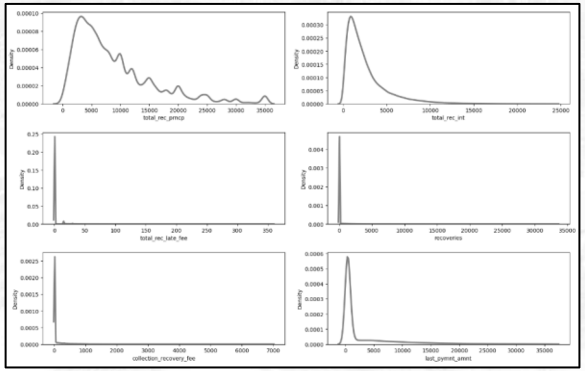
As we can see from the boxplots above, there are a lot of columns that have a lot of outliers within them. We might need to use tree-based algorithms later on since they are robust towards outliers, unlike linear models.

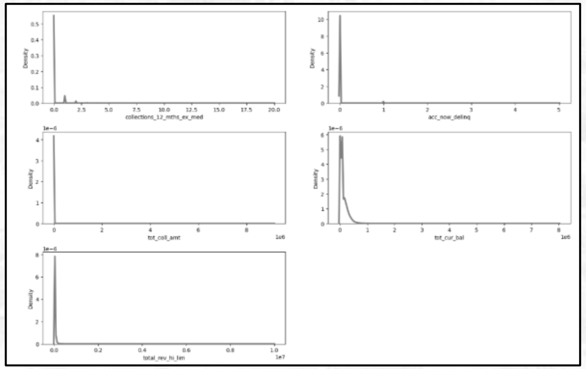
Exploratory Data Analysis





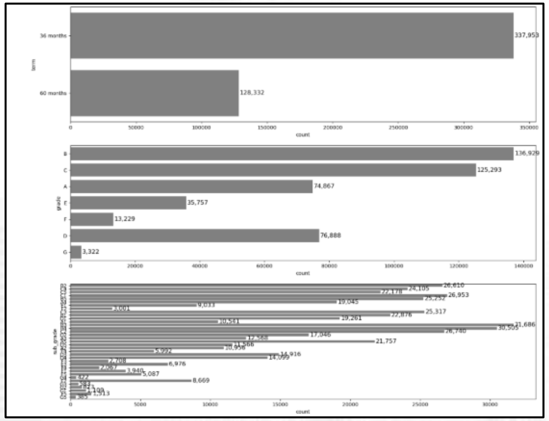


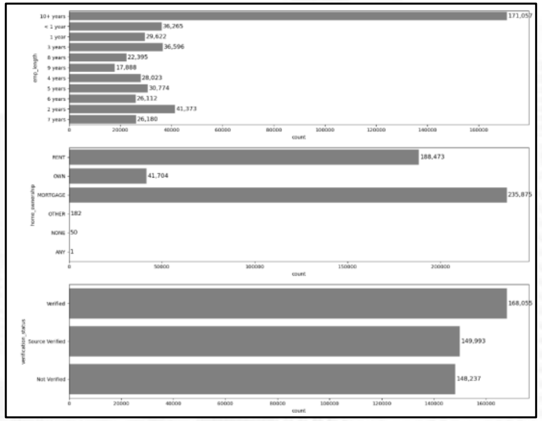


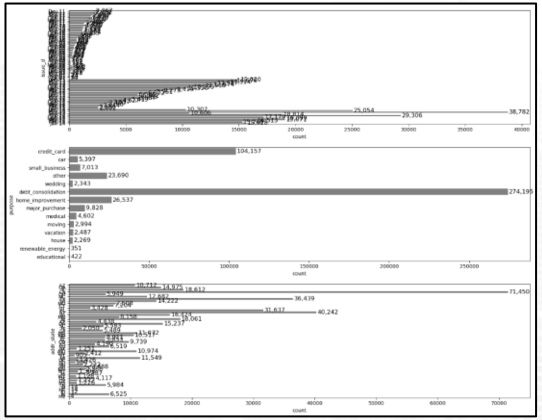


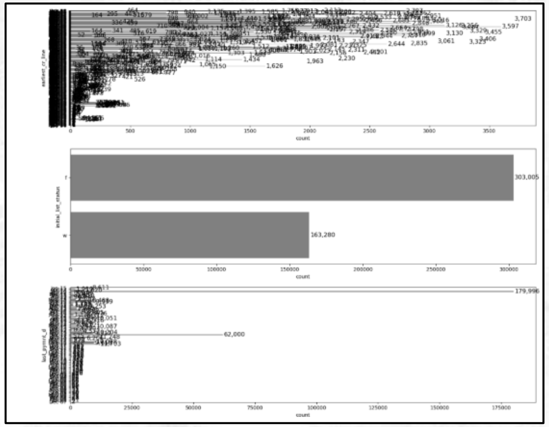
From the kdeplots above, we can see that the distributions of most of the columns are very skewed due to outliers. Most are skewed normal distributions, although a few are multi-modal.

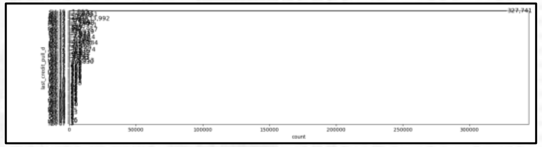
Exploratory Data Analysis







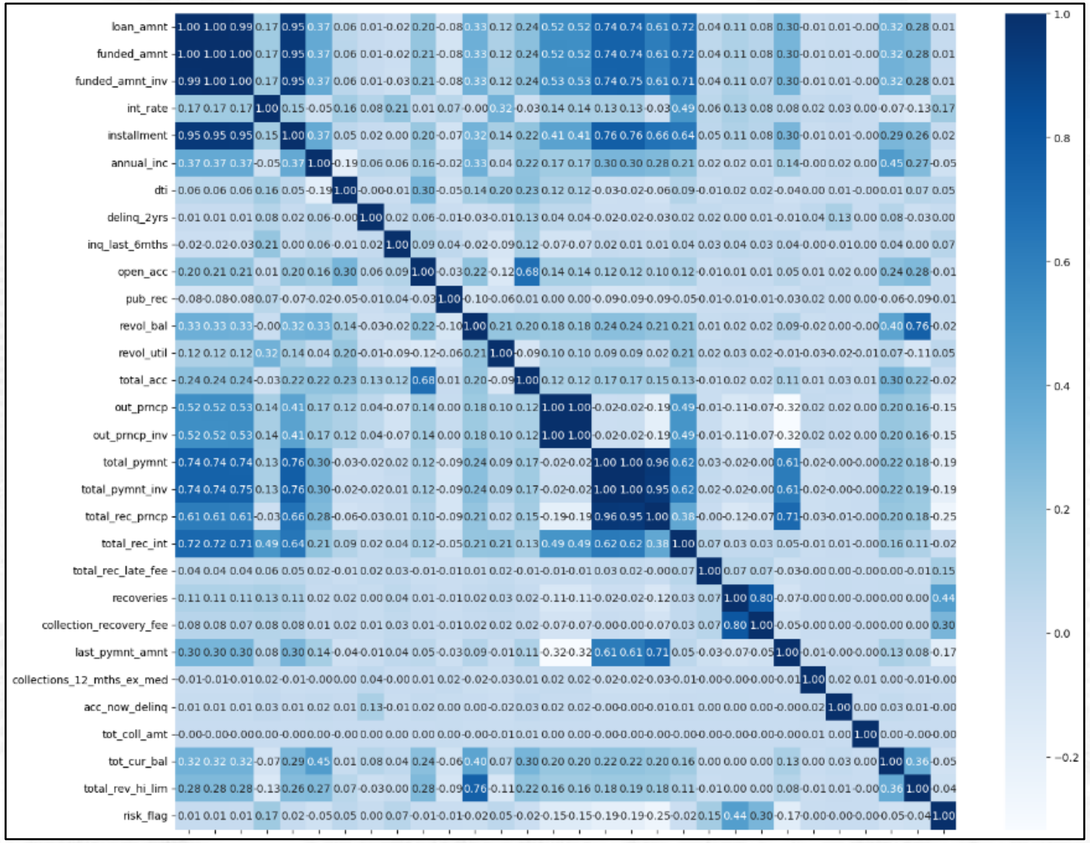




From the countplots above, it is clear that there is no column that has balanced classes within them.

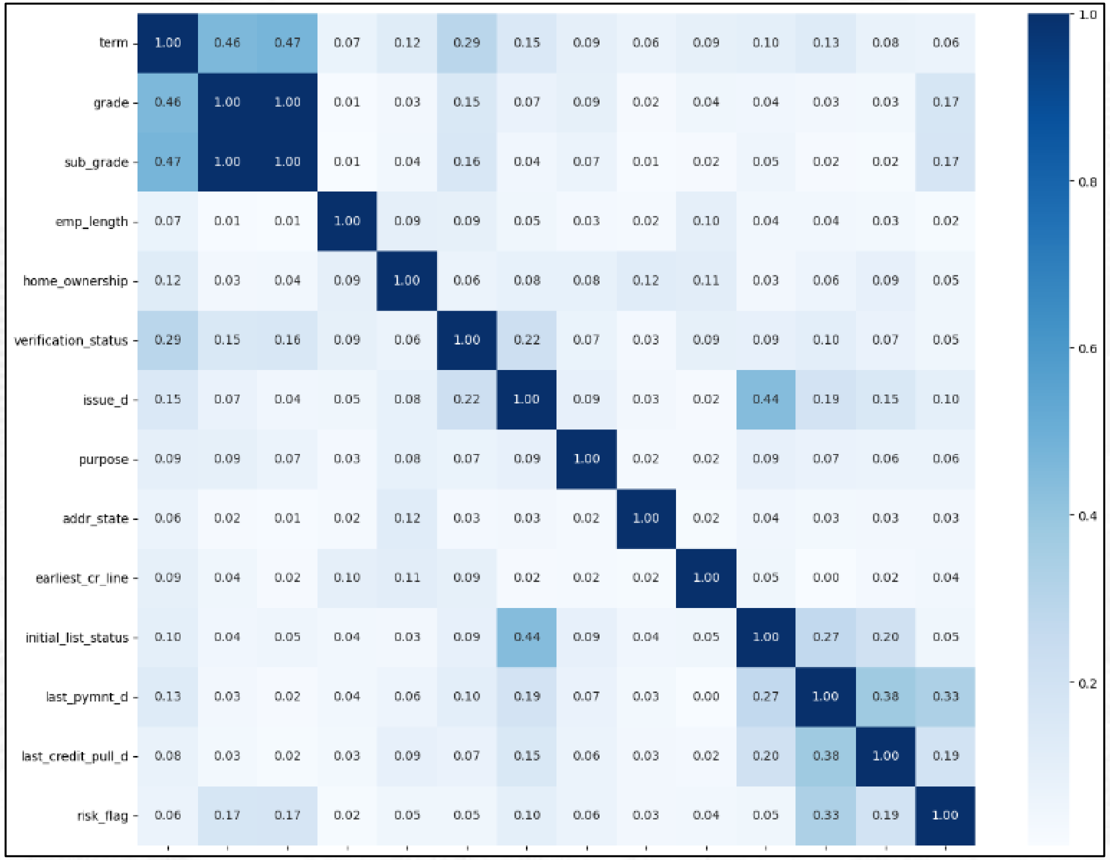
That being said, the imbalances are still somewhat manageable and not too severe. There a few columns that has way too many unique values in them, and this needs to be handled in the preprocessing later on.

Exploratory Data Analysis



As we can see from the correlation heatmap on the left, there are a few features that have quite strong correlation with the target. And there are quite a lot of features that have little to no correlation to the target. These features must be considered to be dropped. There are also a couple of features that correlate to each other very well. One of those features also must be dropped later on.

Exploratory Data Analysis

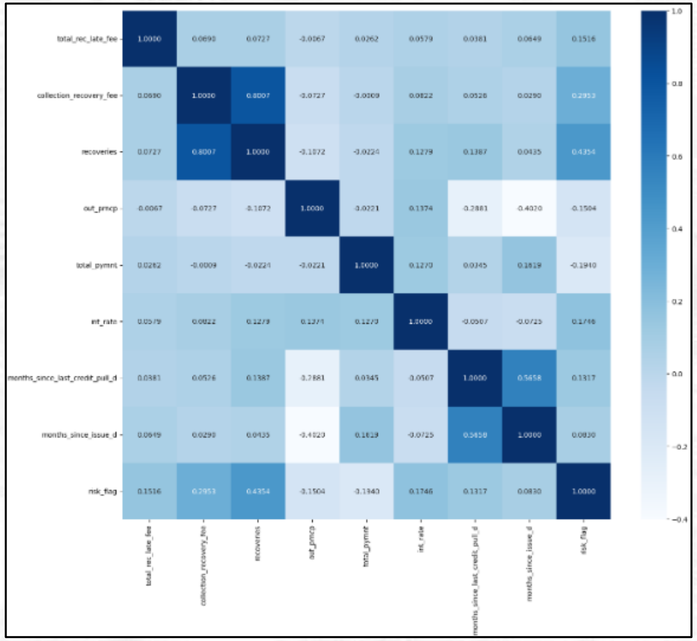


term	1.00	0.46	0.47	0.07	0.12	0.29	0.15	0.09	0.06	0.09	0.10	0.13	0.08	0.06
grade	0.46	1.00	1.00	0.01	0.03	0.15	0.07	0.09	0.02	0.04	0.04	0.03	0.03	0.17
sub_grade	0.47	1.00	1.00	0.01	0.04	0.16	0.04	0.07	0.01	0.02	0.05	0.02	0.02	0.17
emp_length	0.07	0.01	0.01	1.00	0.09	0.09	0.05	0.03	0.02	0.10	0.04	0.04	0.03	0.02
home_ownership	0.12	0.03	0.04	0.09	1.00	0.06	0.08	0.08	0.12	0.11	0.03	0.06	0.09	0.05
verification_status	0.29	0.15	0.16	0.09	0.06	1.00	0.22	0.07	0.03	0.09	0.09	0.10	0.07	0.05
issue_d	0.15	0.07	0.04	0.05	0.08	0.22	1.00	0.09	0.03	0.02	0.44	0.19	0.15	0.10
purpose	0.09	0.09	0.07	0.03	0.08	0.07	0.09	1.00	0.02	0.02	0.09	0.07	0.06	0.06
addr_state	0.06	0.02	0.01	0.02	0.12	0.03	0.03	0.02	1.00	0.02	0.04	0.03	0.03	0.03
earliest_cr_line	0.09	0.04	0.02	0.10	0.11	0.09	0.02	0.02	0.02	1.00	0.05	0.00	0.02	0.04
initial_list_status	0.10	0.04	0.05	0.04	0.03	0.09	0.44	0.09	0.04	0.05	1.00	0.27	0.20	0.05
last_pymnt_d	0.13	0.03	0.02	0.04	0.06	0.10	0.19	0.07	0.03	0.00	0.27	1.00	0.38	0.33
last_credit_pull_d	0.08	0.03	0.02	0.03	0.09	0.07	0.15	0.06	0.03	0.02	0.20	0.38	1.00	0.19
risk_flag	0.06	0.17	0.17	0.02	0.05	0.05	0.10	0.06	0.03	0.04	0.05	0.33	0.19	1.00

From the correlation heatmap on the left, we can see that there are few features that correlate to the target. But quite a lot of features that have little to no correlation to the target. Again, these features must be considered to be dropped. There are also a couple of features that correlate to each other very well. One of those features also must be dropped later on.

Data Preprocessing

Since there are so many columns, we risk of hitting the curse of dimensionality and expenses of computing. Therefore, we are going to do a feature selection where features that have less than 0.1 correlation to the target will be dropped. And we're also going to drop one of the features that have a correlation of 0.9 or higher with another feature, since one of those features is redundant.



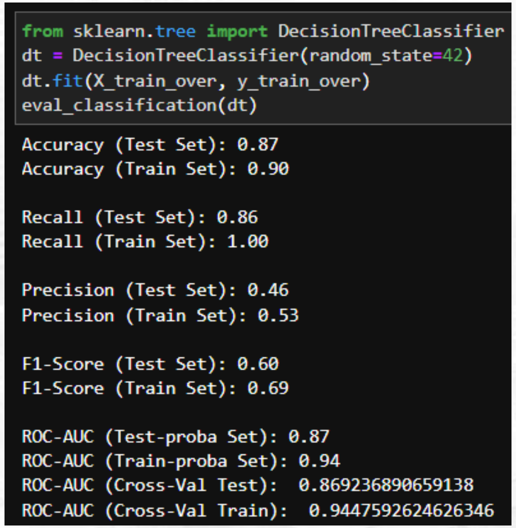
	total_rec_late_fee	collection_recovery_fee	recoveries	out_prncp	total_pymnt	int_rate	months_since_last_credit_pull_d	months_since_issue_d	risk_flag
total_rec_late_fee	1.0000	0.0690	0.0727	-0.0067	0.0262	0.0579	0.0381	0.0649	0.1516
collection_recovery_fee	0.0690	1.0000	0.8007	-0.0727	-0.0009	0.0822	0.0526	0.0290	0.2953
recoveries	0.0727	0.8007	1.0000	-0.1072	-0.0224	0.1279	0.1387	0.0435	0.4354
out_prncp	-0.0067	-0.0727	-0.1072	1.0000	-0.0221	0.1374	-0.2881	-0.4020	-0.1504
total_pymnt	0.0262	-0.0009	-0.0224	-0.0221	1.0000	0.1270	0.0345	0.1819	-0.1940
int_rate	0.0579	0.0822	0.1279	0.1374	0.1270	1.0000	-0.0507	-0.0725	0.1746
months_since_last_credit_pull_d	0.0381	0.0526	0.1387	-0.2881	0.0345	-0.0507	1.0000	0.5658	0.1317
months_since_issue_d	0.0649	0.0290	0.0435	-0.4020	0.1819	-0.0725	0.5658	1.0000	0.0830
risk_flag	0.1516	0.2953	0.4354	-0.1504	-0.1940	0.1746	0.1317	0.0830	1.0000

Since we're not going to use linear/distance based models, we don't have to scale the features, and can keep them as is. Now, the preprocessing is done. We have now narrowed the dataset to only 8 features and 1 target variable.

It is time to split the dataset. We are going to split the dataset into 3 parts. One training dataset, one test/validation dataset, and one validation set that will not be involved in the modelling (let's call it the back test set). The back test set will only be used in the final confusion matrix, so therefore the model will not be influenced by it whatsoever. Out of 466285 datapoints, we're going to use 66285 as the back test dataset. Leaving 400000 as the dataset that we will use for the modelling. We will also use random under sampler as a means of handling the dataset imbalance.

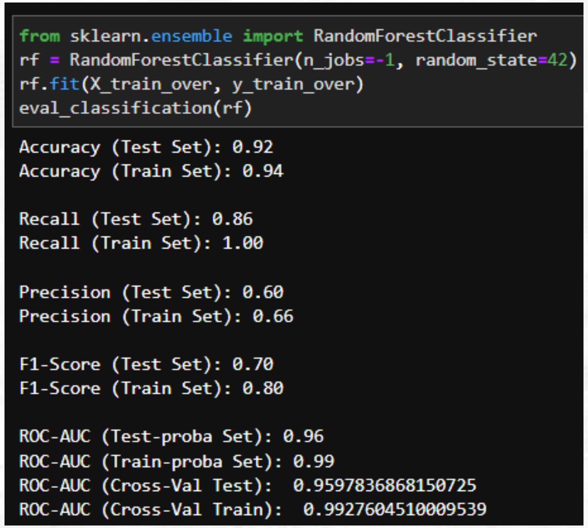
Modelling

Since our target is quite heavily imbalanced, we cannot solely use the accuracy metric alone. We need to account for other metrics such as ROC-AUC. And since it is imperative that we suppress the false negatives (since we don't want to give loans to defaulting customers more than we don't want to lose potential customers), we therefore should also look at the recall score. So as a summary, the metrics that we will look to maximize will be the recall and ROC-AUC metric. The rest are there as complimentary metrics.



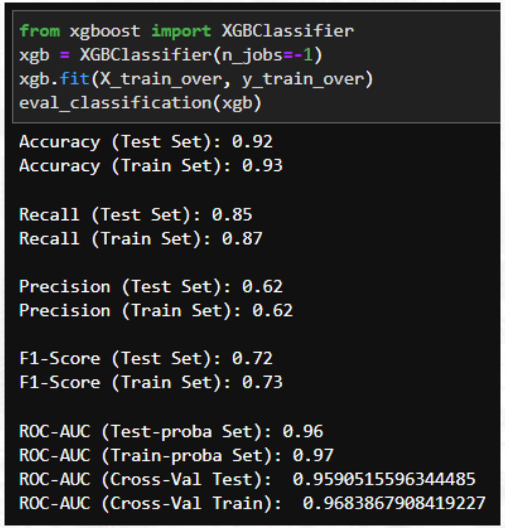
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_over, y_train_over)
eval_classification(dt)
```

```
Accuracy (Test Set): 0.87
Accuracy (Train Set): 0.90

Recall (Test Set): 0.86
Recall (Train Set): 1.00

Precision (Test Set): 0.46
Precision (Train Set): 0.53

F1-Score (Test Set): 0.60
F1-Score (Train Set): 0.69

ROC-AUC (Test-proba Set): 0.87
ROC-AUC (Train-proba Set): 0.94
ROC-AUC (Cross-Val Test):  0.869236890659138
ROC-AUC (Cross-Val Train):  0.9447592624626346
```



```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs=-1, random_state=42)
rf.fit(X_train_over, y_train_over)
eval_classification(rf)
```

```
Accuracy (Test Set): 0.92
Accuracy (Train Set): 0.94

Recall (Test Set): 0.86
Recall (Train Set): 1.00

Precision (Test Set): 0.60
Precision (Train Set): 0.66

F1-Score (Test Set): 0.70
F1-Score (Train Set): 0.80

ROC-AUC (Test-proba Set): 0.96
ROC-AUC (Train-proba Set): 0.99
ROC-AUC (Cross-Val Test):  0.9597836868150725
ROC-AUC (Cross-Val Train):  0.9927604510009539
```

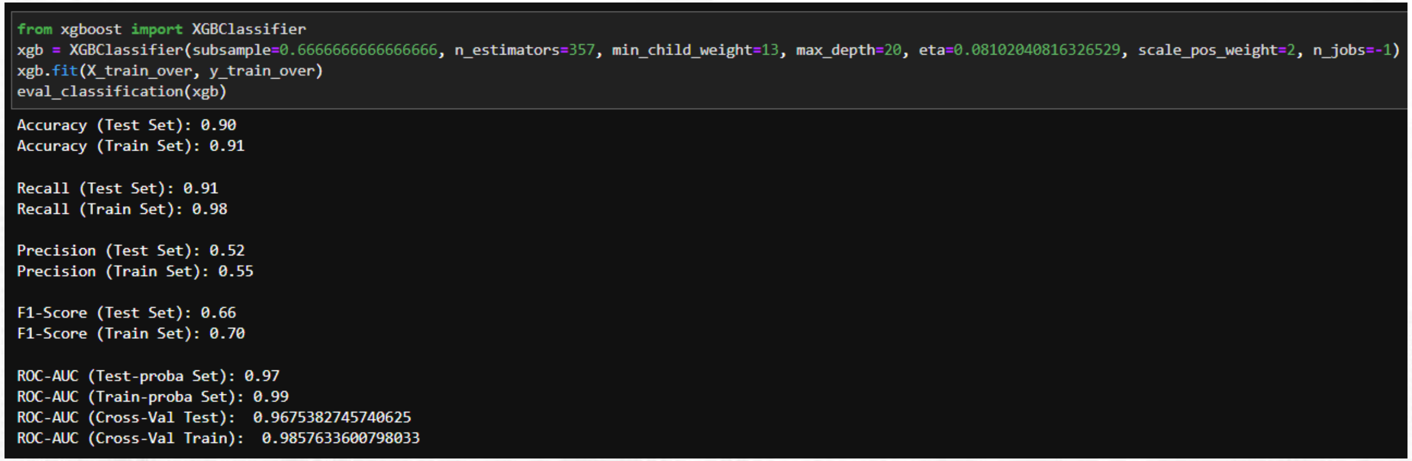


```
from xgboost import XGBClassifier
xgb = XGBClassifier(n_jobs=-1)
xgb.fit(X_train_over, y_train_over)
eval_classification(xgb)
```

```
Accuracy (Test Set): 0.92
Accuracy (Train Set): 0.93

Recall (Test Set): 0.85
Recall (Train Set): 0.87

Precision (Test Set): 0.62
Precision (Train Set): 0.62

F1-Score (Test Set): 0.72
F1-Score (Train Set): 0.73

ROC-AUC (Test-proba Set): 0.96
ROC-AUC (Train-proba Set): 0.97
ROC-AUC (Cross-Val Test):  0.9590515596344485
ROC-AUC (Cross-Val Train):  0.9683867908419227
```

From the results above, we can see that the best model is XGBoost, since it is the only model that does not overfit. We can now move on to the hyperparameter tuning.

Modelling



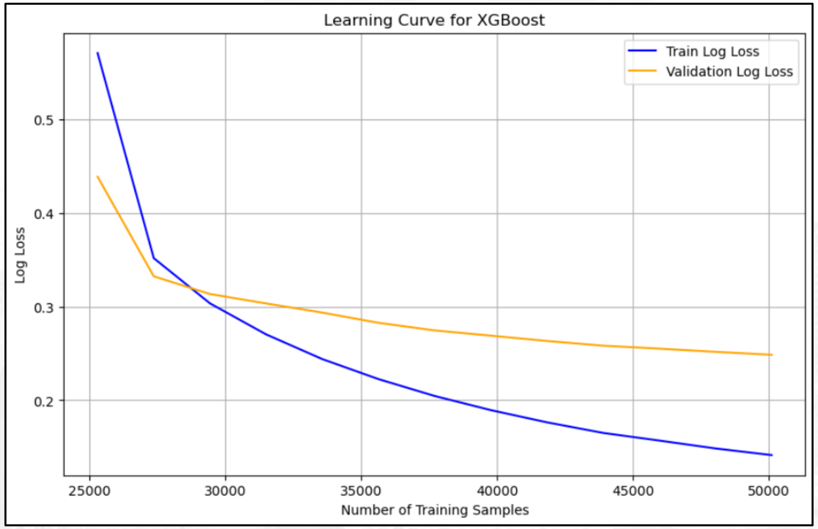
```
from xgboost import XGBClassifier
xgb = XGBClassifier(subsample=0.6666666666666666, n_estimators=357, min_child_weight=13, max_depth=20, eta=0.08102040816326529, scale_pos_weight=2, n_jobs=-1)
xgb.fit(X_train_over, y_train_over)
eval_classification(xgb)
```

```
Accuracy (Test Set): 0.90
Accuracy (Train Set): 0.91

Recall (Test Set): 0.91
Recall (Train Set): 0.98

Precision (Test Set): 0.52
Precision (Train Set): 0.55

F1-Score (Test Set): 0.66
F1-Score (Train Set): 0.70

ROC-AUC (Test-proba Set): 0.97
ROC-AUC (Train-proba Set): 0.99
ROC-AUC (Cross-Val Test):  0.9675382745740625
ROC-AUC (Cross-Val Train):  0.9857633600798033
```

As we can see above, the recall has improved significantly from 0.85 to 0.91 in the test set, whilst also maintaining a good fit to the model, and not sacrificing too much on accuracy.

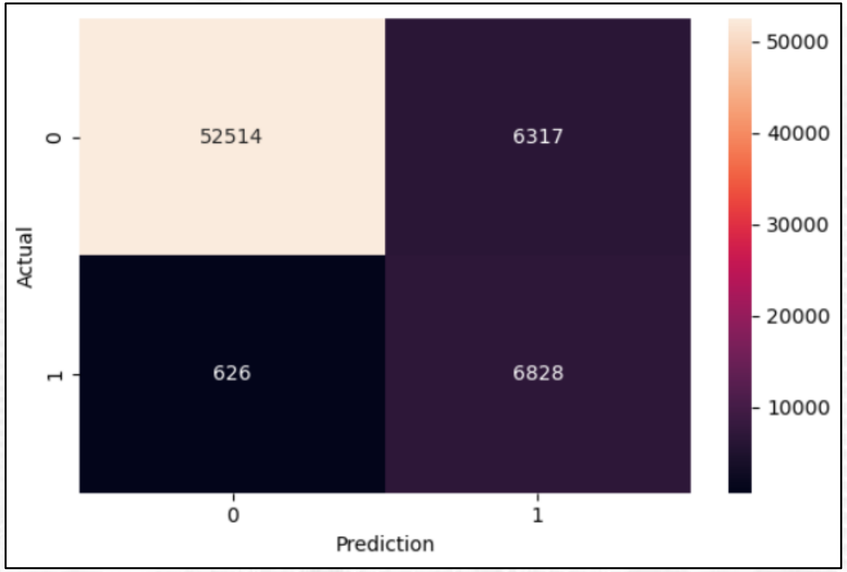
We can also check the learning curve to be sure that the model does not overfit.

Modelling



From the learning curve above, we can see that the train log loss does not differ by a lot compared to the validation log loss. This means that the model is a good fit. We can now proceed to the confusion matrix.

Modelling

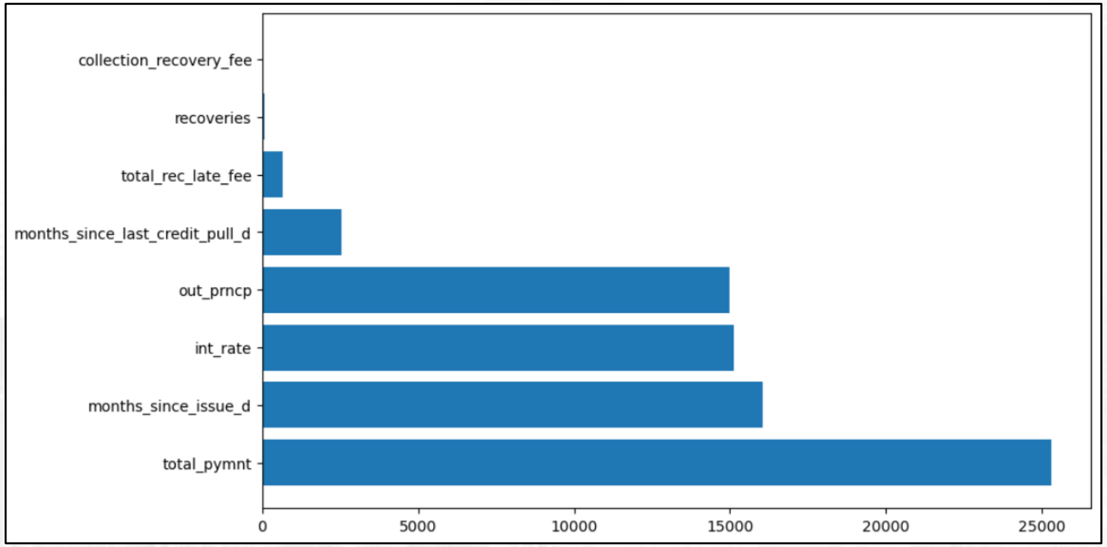


For the confusion matrix, we are going to use the back test dataset, just like we mentioned earlier. The back test dataset is a dataset that has not been touched in the modelling process, crucial for a fair and transparant review of the model's performance. From the confusion matrix above, we can calculate the accuracy, recall, and precision. And they are as follows :

1. **Accuracy** : (True Positive + True Negative) / Total. (6828 + 52514) / 66285 = **0.895**
2. **Recall** : True Positive / (True Positive + False Negative). 6828 / (6828 + 626) = **0.916**
3. **Precision** : True Positive / (True Positive + False Positive). 6828 / (6828 + 6317) = **0.519**

Let us now look at the feature importances of the model.

Modelling



From the feature importances above, we can see that the 'total_pymnt' feature is the most impactful, with the recovery related features not bringing as much impact compared to the others.

Business Simulation

Before we did the modelling, the default rate in the dataset was 11.19%. That means that out of all of the customers whose loan application were accepted, 11.19% of them eventually defaulted on their loan. However, with the predictive model, less than 1% (0.94%) defaulted on their loans. For further simulation, we need to make assumptions. Let's say, that on average, every loan is 1,000 dollars in amount, and the interest that we are going to get from that is 100 dollars. And let's say, that on average, when a customer defaults, we lose 50% or half of the amount that we loaned to them & its interest as well. Now that we get that out of the way, it's time to simulate.

Business Simulation

Before the modelling, there are 466285 customers, of which 52186 defaulted. That means that out of all the loan that we gave (466285 x 1000) = **466,285,000 dollars**, we receive back (414099 x 1100) + (52186 x 550) = **484,211,200 dollars** which is a **3.8% profit**.

Now after the modelling, looking at the confusion matrix in the previous section, out of 66285 applications, we only gave out loans to 53140 customers, of which 626 defaulted. That means that out of all the loan that we gave (53140 x 1000) = **53,140,000 dollars**, we receive back (52514 x 1100) + (626 x 550) = **58,109,700 dollars** which is a **9.35% profit**.

Business Simulation

We also need to address the fact that there are opportunity losses, since we are maximizing the recall score. If we do the first scenario (giving loans to all customers that applied) on the second back test dataset, than out of all the loan that we gave (66285 x 1000) = **66,285,000 dollars**, we receive back ((52514 + 6317) x 1100) + ((626 + 6828) x 550) = **68,813,800 dollars**. That's a profit ratio of **3.8%**, and a total profit of **2,528,800 dollars**, whilst the second scenario previously had a total profit of **4,969,700 dollars**.

Therefore in conclusion, there is a **146.05%** increase in profit margins after implementing the predictive model. And despite the opportunity loss, we still have a total profit that is higher, **96.52%** higher to be exact.