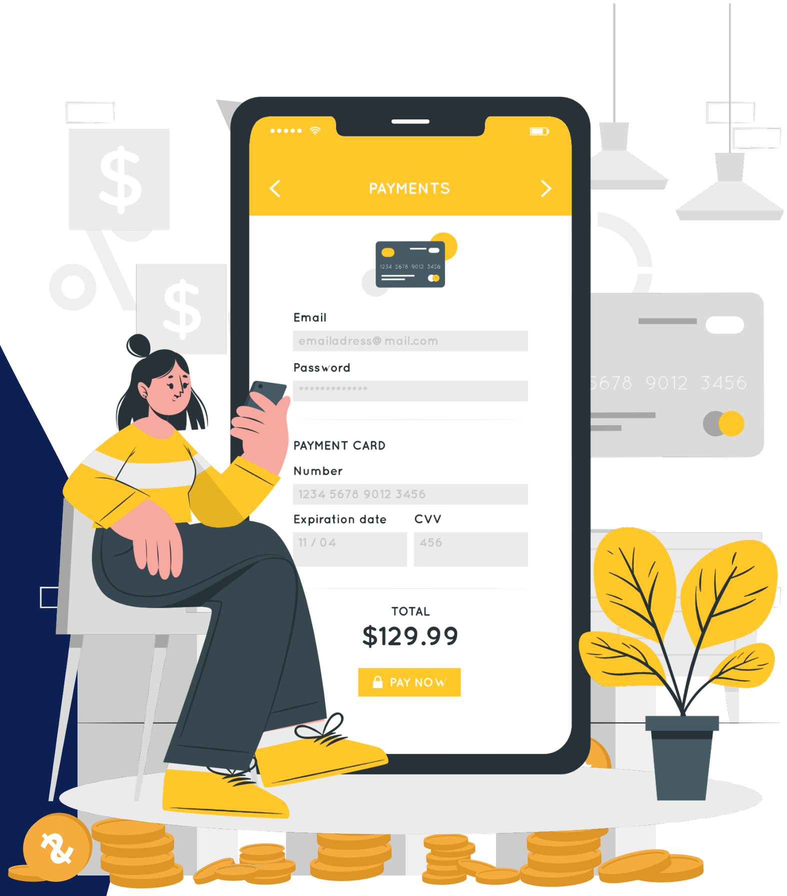
Loan Default Detection



The Problem

Financial institutions need to determine whether or not to approve someone's loan application

Why is it worth solving?

Banks want to increase their profit margins while also reducing their exposure to risk

01

Data Source

Credit Card Fraud Detection Dataset from Kaggle (https://www.kaggle.com/mishra5001/credit-card)

Loan Default Detection Raw Dataset

1

Shape

307,511 rows

2

Variable Types

Integers, Strings, Numerical, Logical

3

Predictors

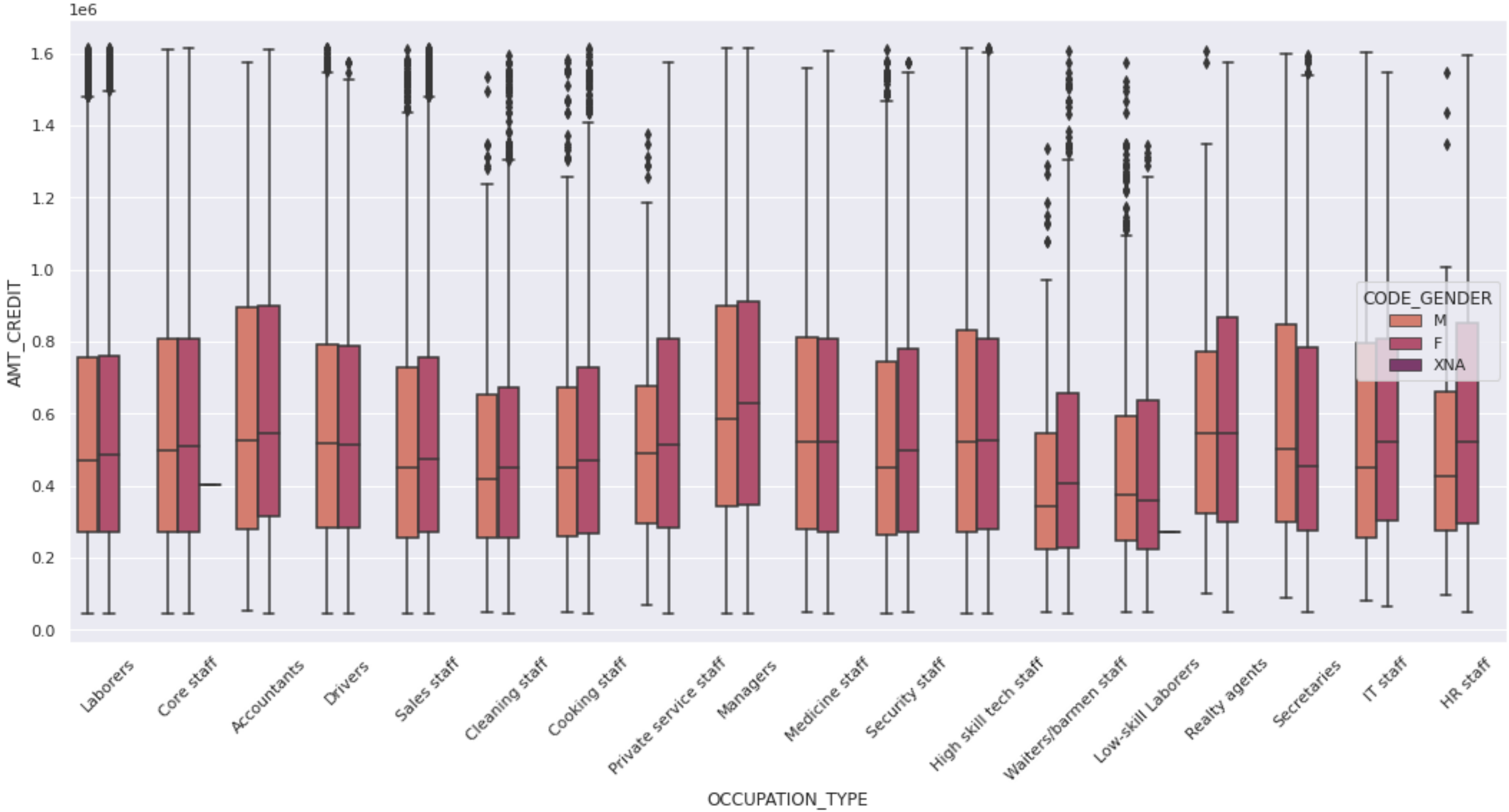
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AM
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	
...	...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157500.0	
307507	456252	0	Cash loans	F	N	Y	0	72000.0	
307508	456253	0	Cash loans	F	N	Y	0	153000.0	
307509	456254	1	Cash loans	F	N	Y	0	171000.0	
307510	456255	0	Cash loans	F	N	N	0	157500.0	

307511 rows × 122 columns

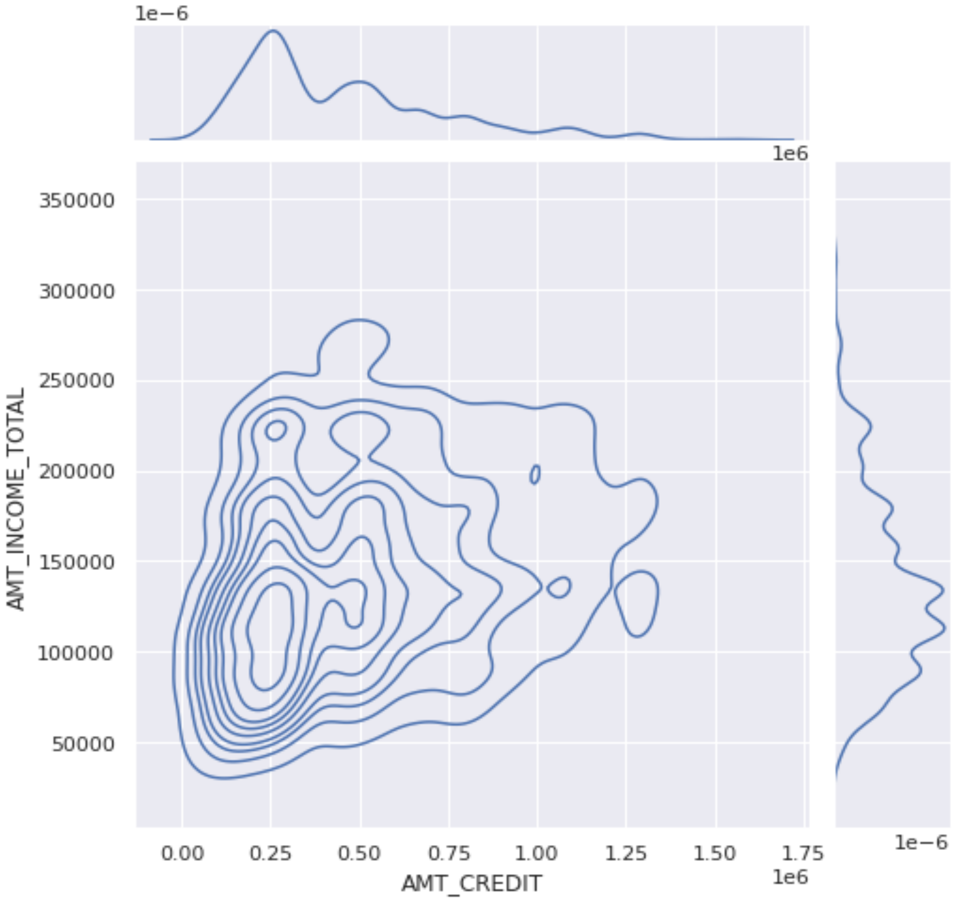
02

Data Introduction

Occupation and Loan Size



Concentration of Loan Amount vs. Income



03

Cleaning our Data

Dropping Columns

- Drop repetitive **mode and median** columns that outline living standard
- Dropped outliers with income greater than or equal to **$700,000**

Our resulting dataframe now has **306773 rows, 88 columns**

Adding Dummies

- Adding dummies for categorical columns using

  pd.get_dummies

NAME_CONTRACT_TYPE_Cash loans	NAME_CONTRACT_TYPE_Revolving loans
1	0
1	0
0	1
1	0
1	0
1	0
1	0
1	0

Resulting dataframe

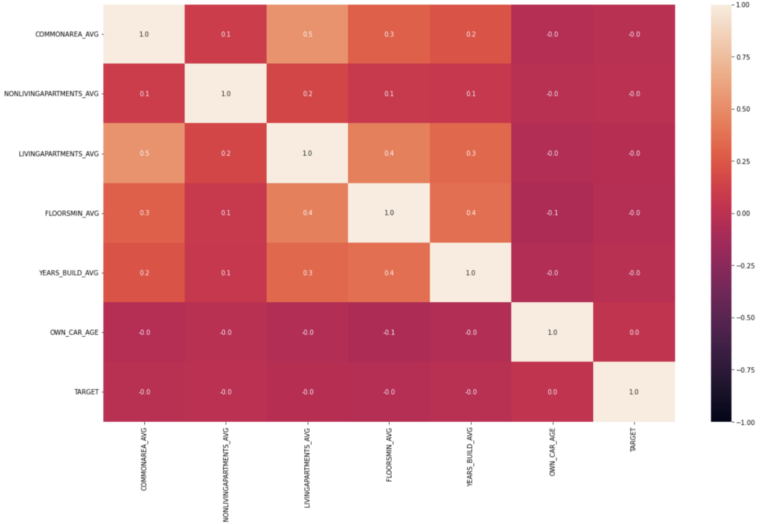
now has **306773 rows,**

200 columns

Dealing with NAN values

- Created a function to calculate the amount of **missing data** in each column
- Removed columns that had missing data greater than **60%**
- Resulting data frame has **306773 rows, 194 columns**
- Ensured that dropped data had relatively **no correlation** with the target variable

	Total	Percent
COMMONAREA_AVG	214462	69.909021
NONLIVINGAPARTMENTS_AVG	213119	69.471238
LIVINGAPARTMENTS_AVG	209813	68.393568
FLOORSMIN_AVG	208262	67.887982
YEARS_BUILD_AVG	204127	66.540080
OWN_CAR_AGE	202681	66.068722
LANDAREA_AVG	182270	59.415268
BASEMENTAREA_AVG	179658	58.563824



Filling in NaN Values

- Tried **K-Means Imputer** but the process was very **slow and impractical** to our workflow because we had a large quantity of NaN values
- Instead, we calculated the **means** for each column and filled missing values based on the mean value

```
for c in df_with_dummies_drop_nan.columns:
  df_with_dummies_drop_nan[c].fillna(value = df_with_dummies_drop_nan[c].mean(), inplace=True)
df_with_dummies_drop_nan.isna().sum()
```

Sampling Techniques/Split

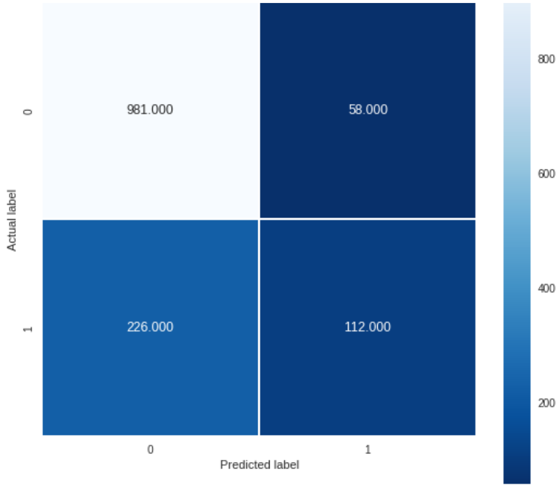
- Randomly subsampled for 20,000 rows
- Experimented with **RandomUnderSampler & NearMiss** (undersampling) and **SMOTE** (oversampling)
    - **SMOTE** generates **synthetic** samples for **minority** class
    - **SMOTE** had relatively **poor** performance - samples are too artificial
- Next tried **NearMiss** and tested out different ratios for the **sampling_strategy** parameter
    - We chose a **0.3** ratio between instances of default and no default
    - For every 3 defaults, there are 10 non-defaults
- After subsampling and undersampling, we have about **10,000 rows**
- **80/20** Train Test Split, Standardized and Normalized
- **Based on how we sampled, the baseline has an accuracy of 70%**

04

Descriptive Analysis & Predictive Models

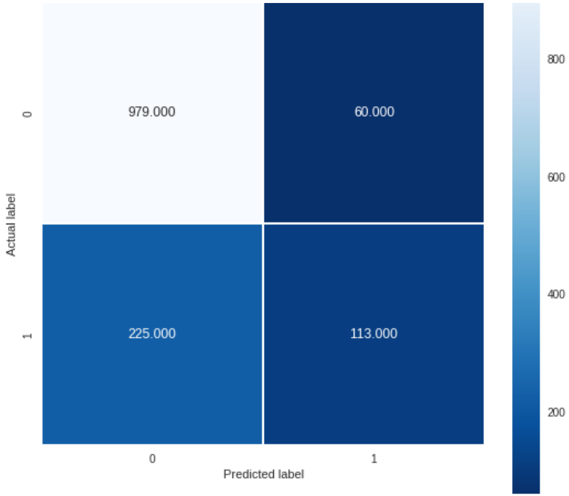
Logistic Regression

Default LR



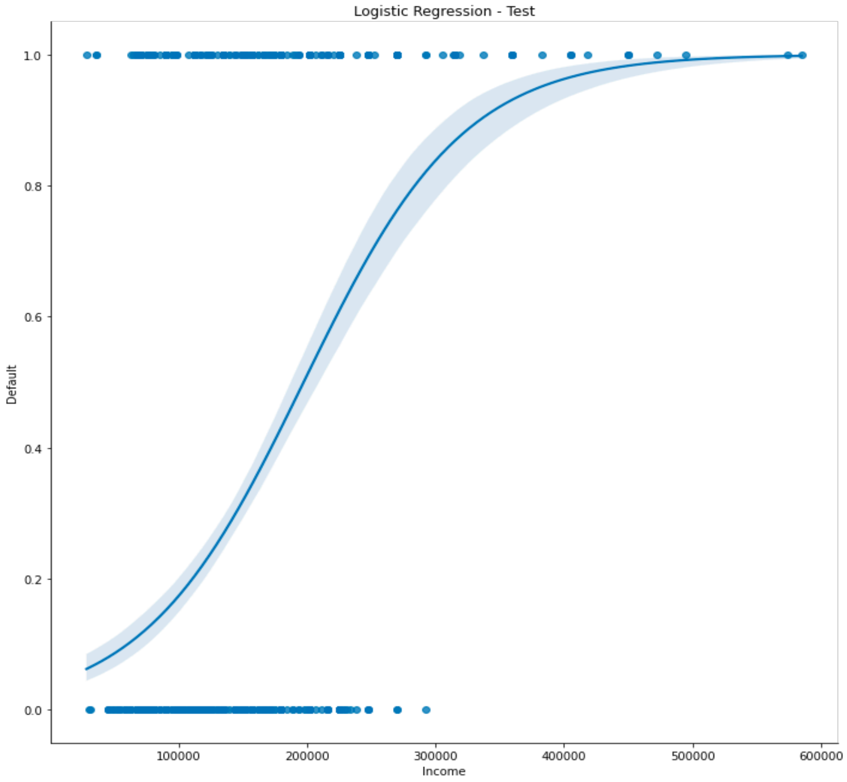
Training set accuracy: 80.90%
Testing set accuracy: 82.77%

CV LR



Training set accuracy: 80.90%
Testing set accuracy: 82.91%

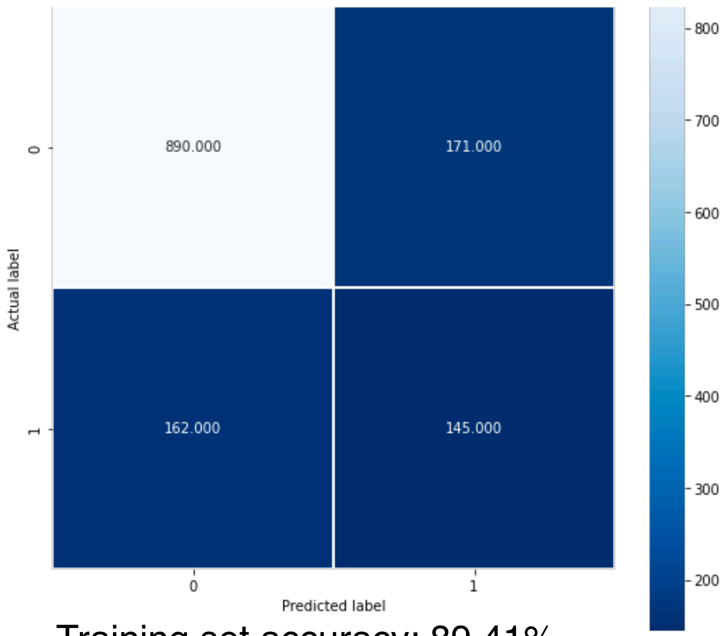
Logistic Regression



- The plot shows that the **higher the income**, the probability of **default increases**
  - We tried different parameters and all resulted in the same curve
  - We believe that this is due to **bias** within the dataset

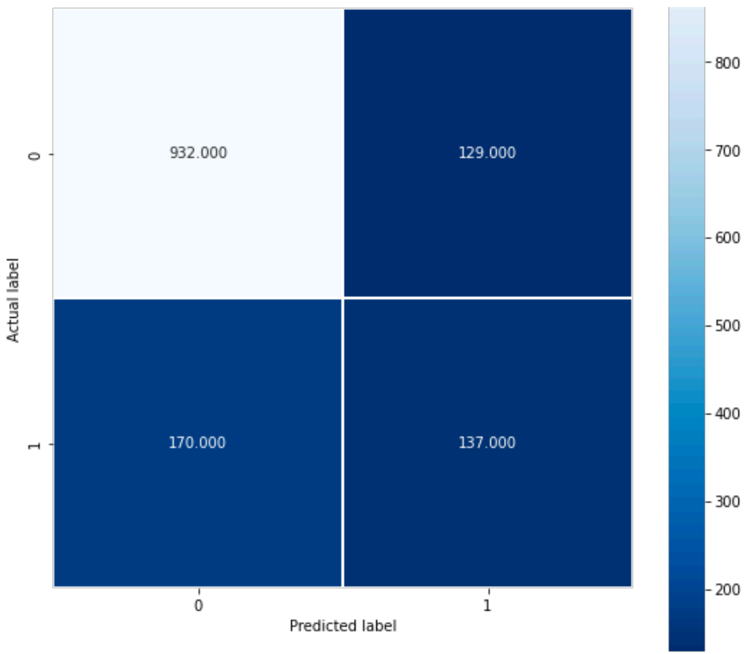
Bagging

Default Bagging



Training set accuracy: 89.41%
Testing set accuracy: 73.69%

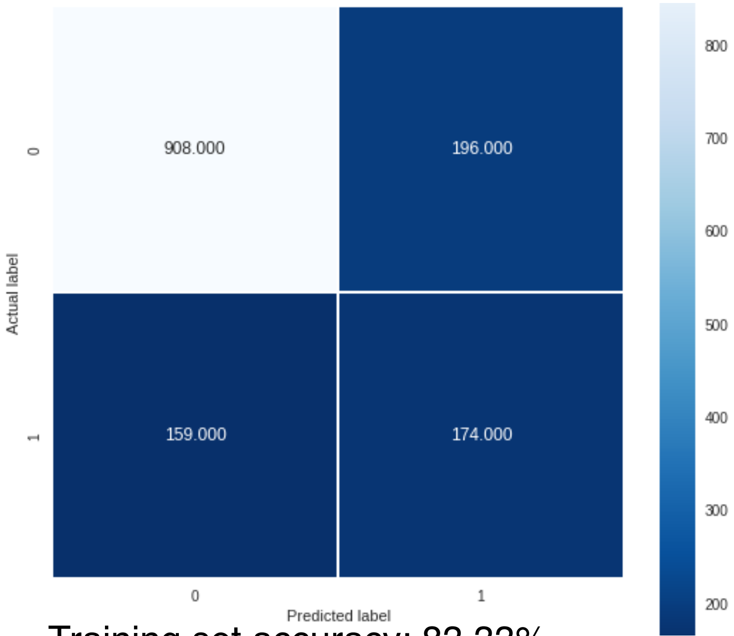
CV Bagging



Training set accuracy: 88.72%
Testing set accuracy: 75.82%

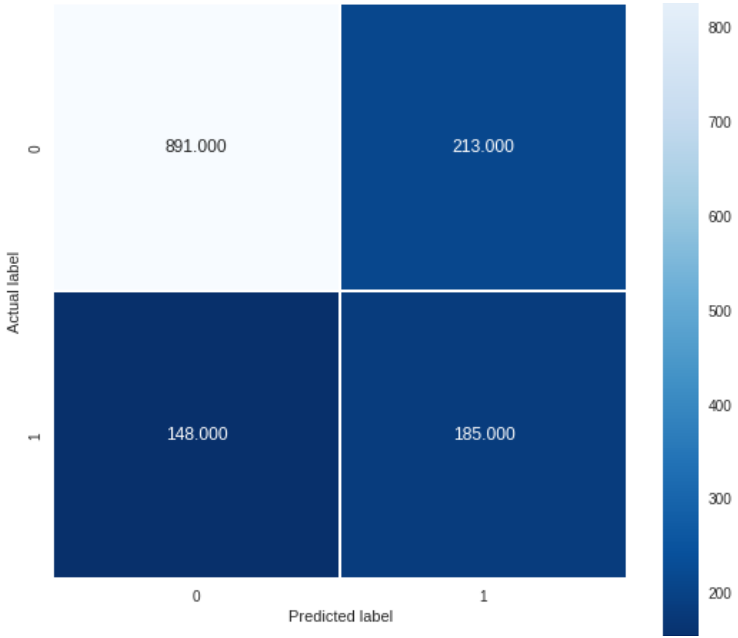
Boosting

Default Boosting



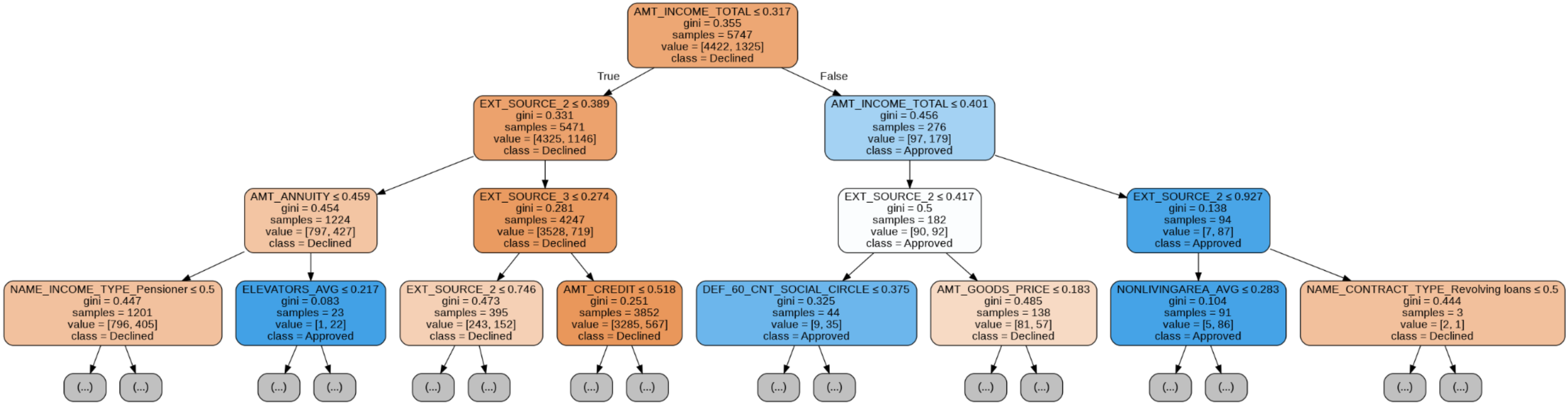
Training set accuracy: 83.33%
Testing set accuracy: 79.36%

CV Boosting

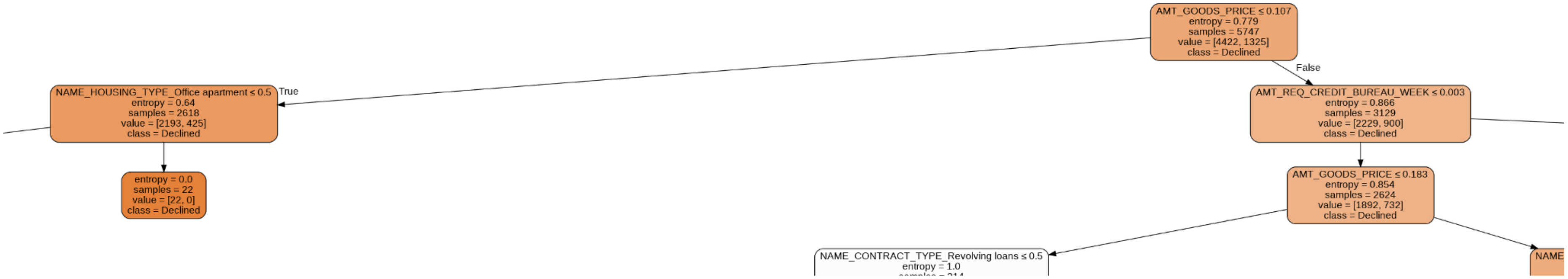


Training set accuracy: 83.74%
Testing set accuracy: 79.50%

Decision Tree

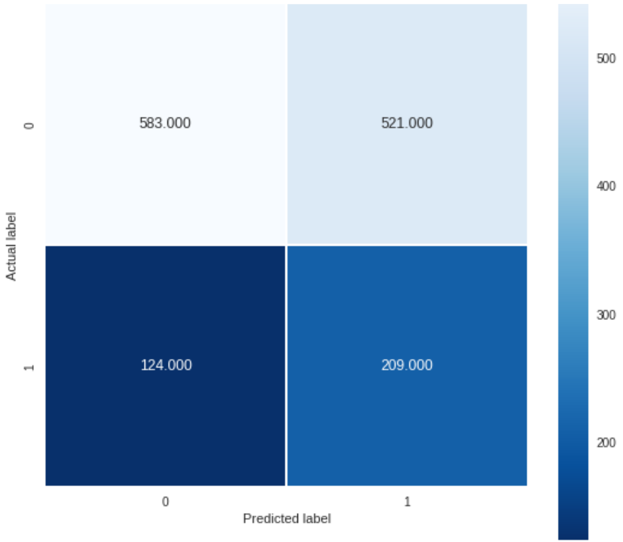


Decision Tree after CV (cropped)

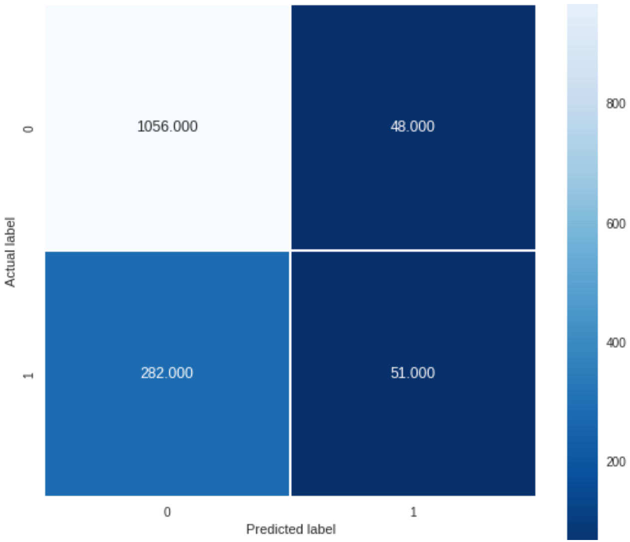


Decision Tree

Default Decision Tree



CV Decision Tree

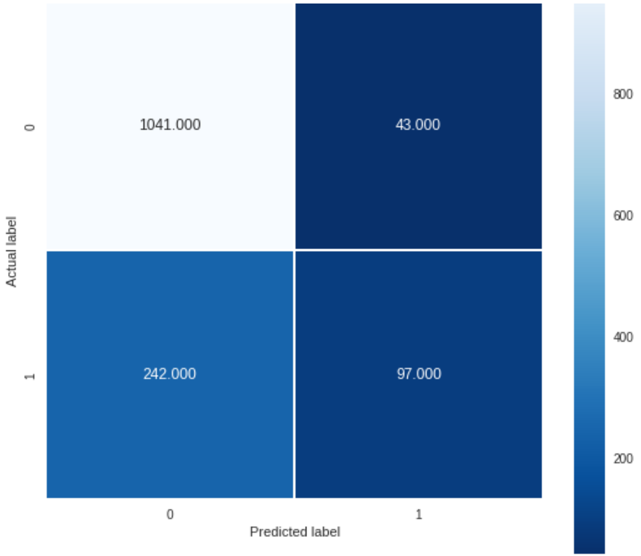


Training set accuracy: 100%
Testing set accuracy: 58.24%

Training set accuracy: 79.26%
Testing set accuracy: 77.52%

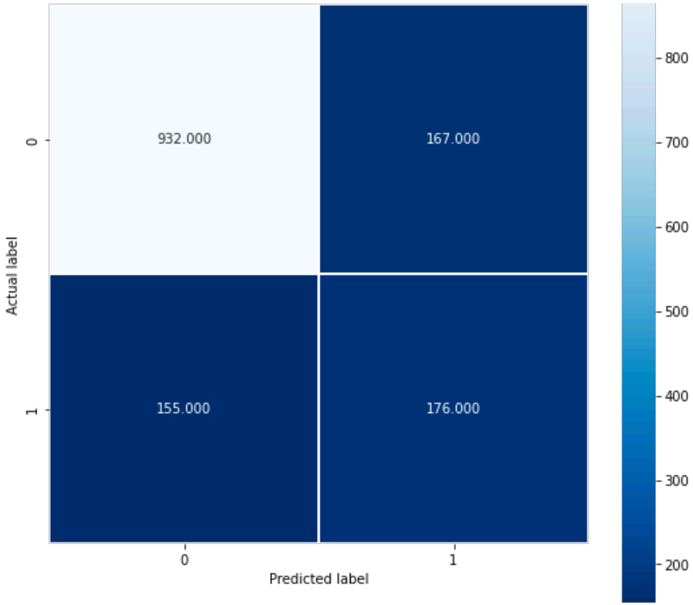
Lasso

Default Lasso



Training set accuracy: 76.48%
Testing set accuracy: 78.65%

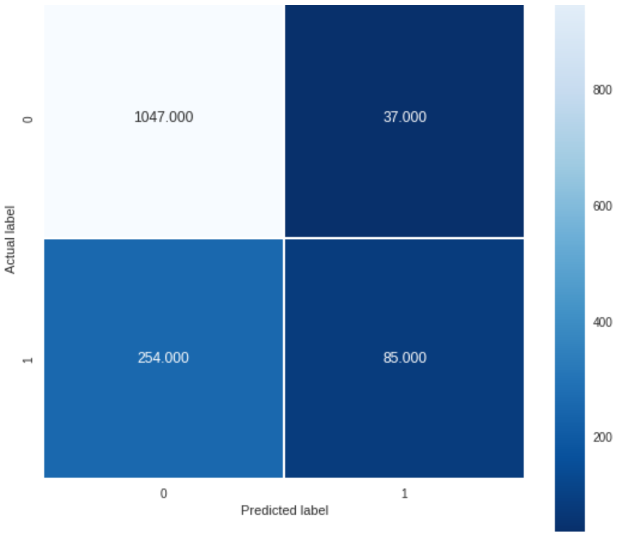
CV Lasso



Training set accuracy: 80.79%
Testing set accuracy: 82.48%

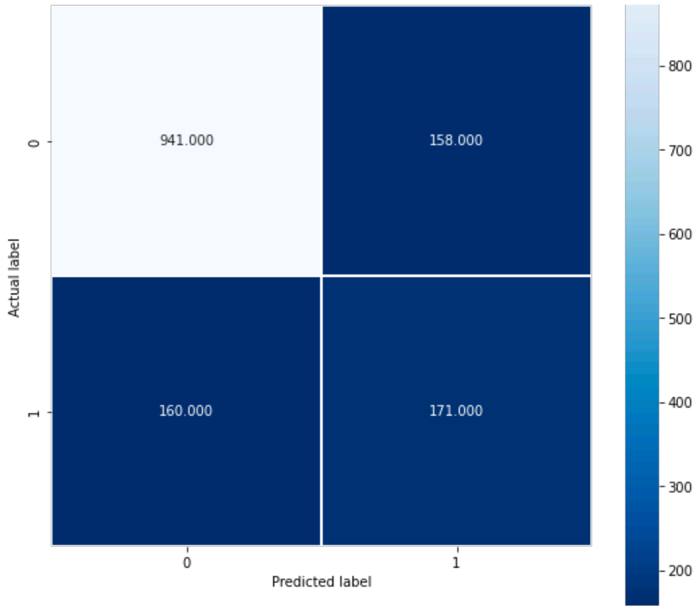
Ridge

Default Ridge



Training set accuracy: 80.95%
Testing set accuracy: 82.26%

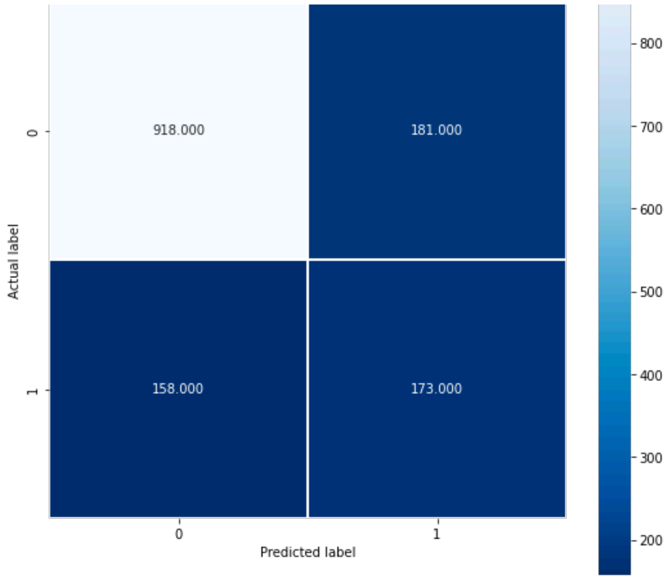
CV Ridge



Training set accuracy: 80.99%
Testing set accuracy: 82.12%

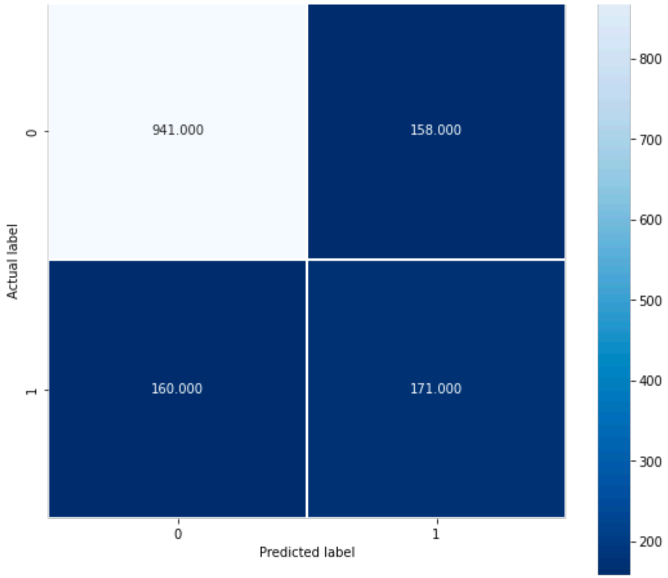
Elastic Net

Default Elastic Net



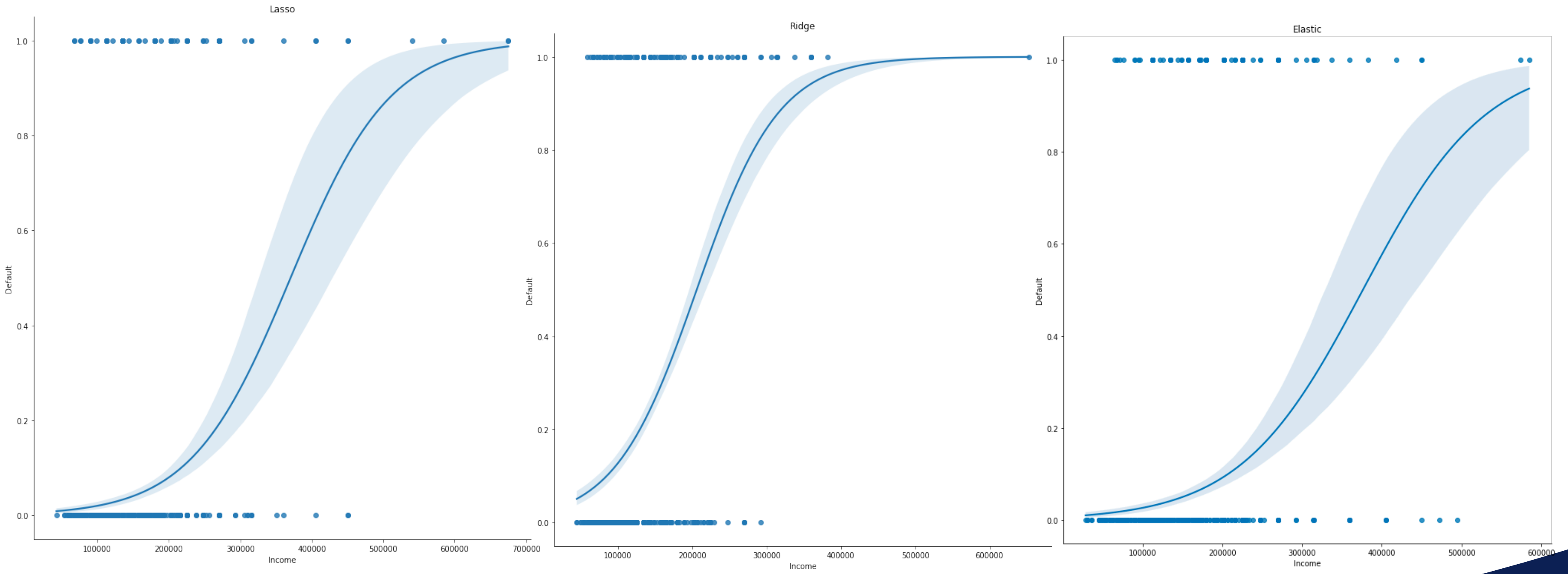
Training set accuracy: 79.98%
Testing set accuracy: 82.20%

CV Elastic Net



Training set accuracy: 79.86%
Testing set accuracy: 81.50%

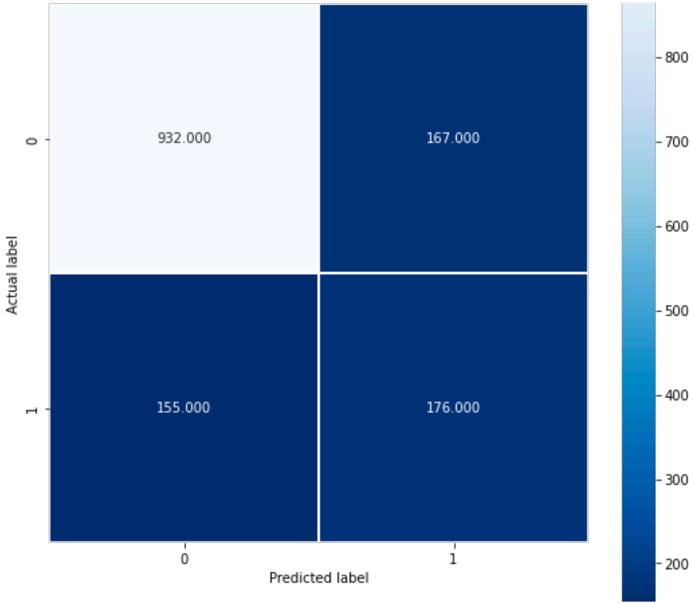
Lasso, Ridge, Elastic Net



Best Model based on accuracy

- **Lasso Net** is the most accurate model

- Why lasso has the highest accuracy
  - Lasso - some coefficients can become **zero** and **eliminate** the predictors from the model
  - Based on the heatmap, we have mostly columns that do not correlate to target



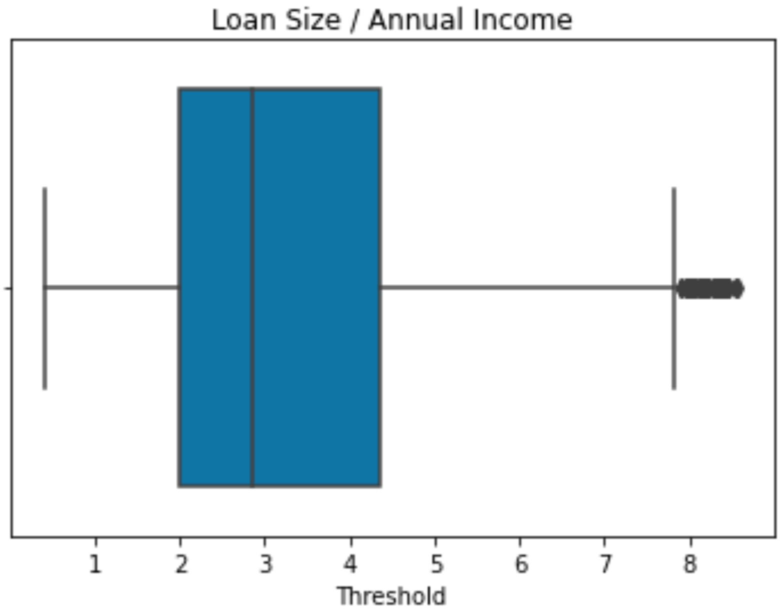
Training set accuracy: 80.79%
Testing set accuracy: 82.48%

05

Feature Engineering and Profit/Loss

Risk Rating and Profit/Loss

- Interest rate is from **U.S. Treasury Yield**
- Inflation Rate of **2%**
- Removed outliers in terms of **loan size** and **income**
- Created **Risk Rating column** for each individual
    - Based on ratio of **Loan Size** to **Income**



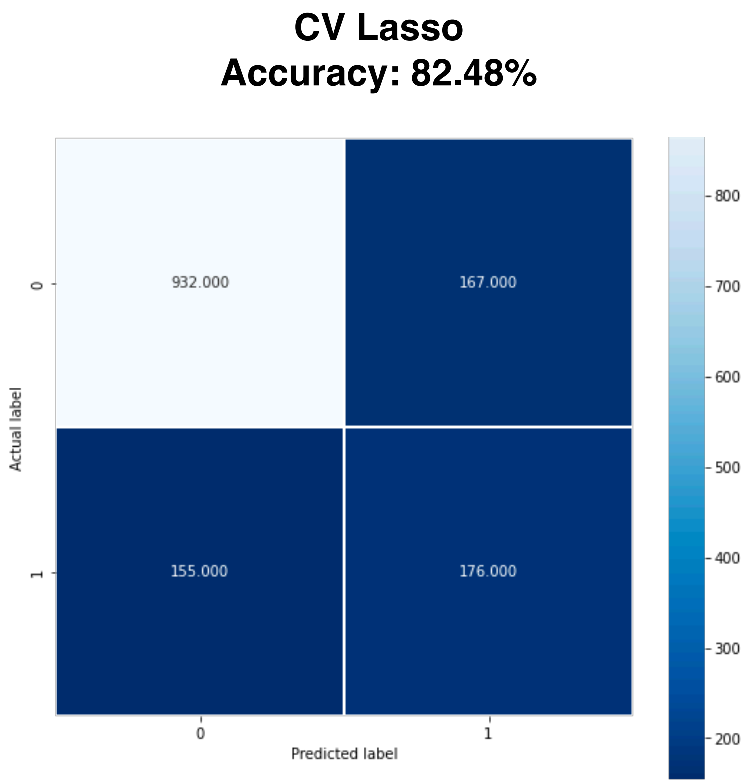
Feature Engineering Workflow

- Pass in each model's **predicted label** to our function
  - Using the predicted label of CV model
- Calculate **profit** on instances where the predicted is **no default** and true label indicate **no default**
- Calculate **loss** on instances where the prediction indicates **no default** but true label indicates **default**
- Calculate **opportunity cost** on instances where the prediction indicates **default** but true label indicates **no default**
- Aggregate **total profit and loss** to evaluate best model - the goal is to maximize profit

Profit/Loss Results

Model	Profit
Logistic Regression	$179,461,405
Bagging	$157,541,475
Boosting	$155,338,172
Lasso Regression	**$225,585,445**
Ridge Regression	$189,043,931
Elastic Net	$188,490,977
Decision Tree	$212,958,162

Lasso - Profit / Loss



- **Profit** - 932 people approved and did not default
- **Loss** - 155 people approved and defaulted
- **Opportunity cost** - 167 people rejected but would not default

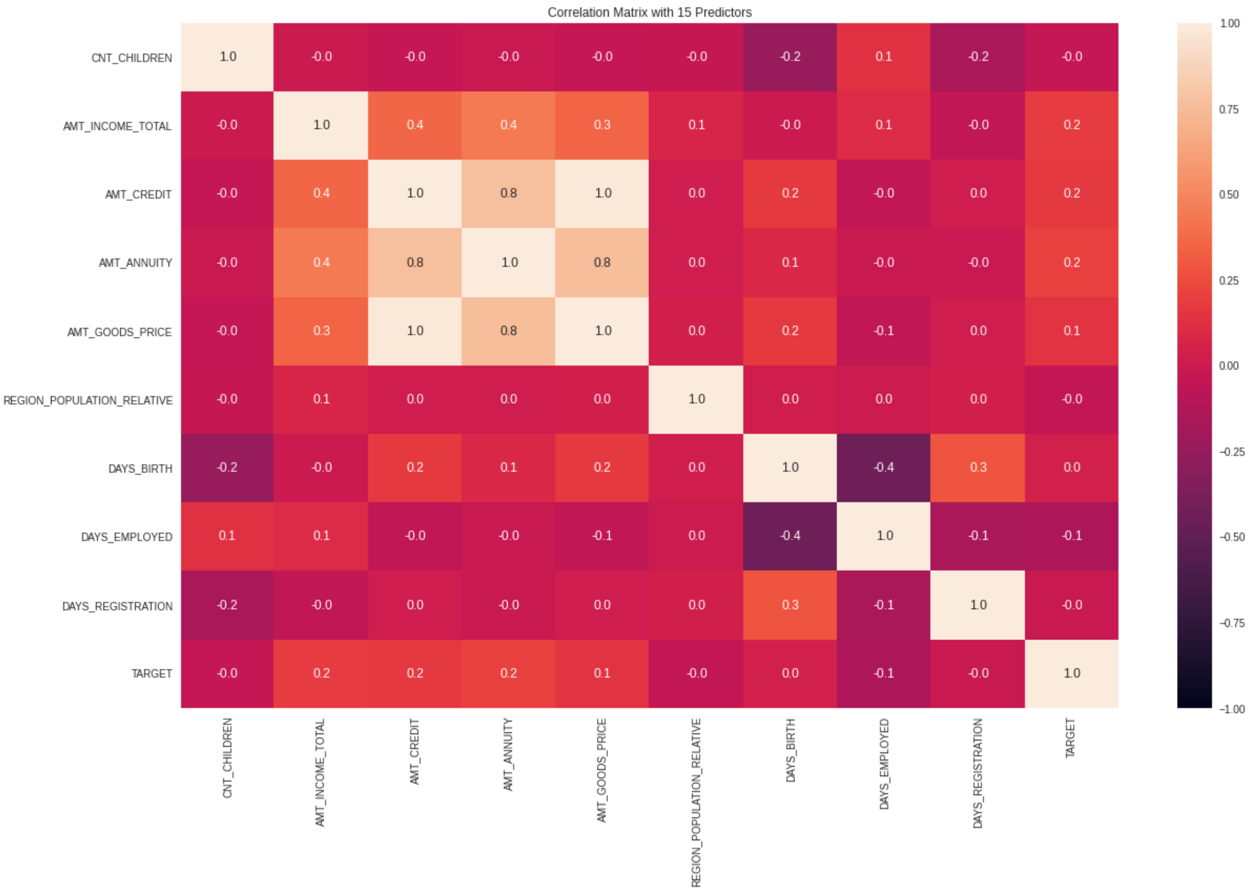
06

Obstacles

Challenges

- Extremely large amount of features (**122 columns** before dummies) and dataset (**300,000+ rows**)
    - With a large dataset, running the models and functions was time consuming - **random subsampling** results in a new dataset every run
- In our dataset, the number of defaulting instances were very small compared to instances that did not default - **undersampling** using **NearMiss**
- Data was **inherently bias** given that it is collected from **multiple banks**
- Could not find **correlation** between most of the variables with the **target** variable

Correlation Heatmap



Thank you