# Report on Credit Default Risk Analysis for a Fintech company

Neil Pradhan

11 June 2023

# Contents

# Chapter 1

# Problem Statement

The task is to predict the probability of default for the datapoints where the default variable is not set. The answer should contain the resulting predictions in a csv file with two columns, uuid and pd (probability of default == 1 ). Once done expose this model with an API Endpoint on a cloud provider of your choice. Bonus points if you use AWS. Given below is the Data columns and their types:

| Column Name | Column Type |
|---|---|
| uuid | text |
| default | categorical |
| account_amount_added_12_24m | numeric |
| account_days_in_dc_12_24m | numeric |
| account_days_in_rem_12_24m | numeric |
| account_days_in_term_12_24m | numeric |
| account_incoming_debt_vs_paid_0_24m | numeric |
| account_status | categorical |
| account_worst_status_0_3m | categorical |
| account_worst_status_12_24m | categorical |
| account_worst_status_3_6m | categorical |
| account_worst_status_6_12m | categorical |
| age | numeric |
| avg_payment_span_0_12m | numeric |
| avg_payment_span_0_3m | numeric |
| merchant_category | categorical |
| merchant_group | categorical |
| has_paid | boolean |
| max_paid_inv_0_12m | numeric |
| max_paid_inv_0_24m | numeric |
| name_in_email | categorical |
| num_active_div_by_paid_inv_0_12m | numeric |
| num_active_inv | numeric |
| num_arch_dc_0_12m | numeric |
| num_arch_dc_12_24m | numeric |
| num_arch_ok_0_12m | numeric |
| num_arch_ok_12_24m | numeric |
| num_arch_rem_0_12m | numeric |
| num_arch_written_off_0_12m | numeric |
| num_arch_written_off_12_24m | numeric |
| num_unpaid_bills | numeric |
| status_last_archived_0_24m | categorical |
| status_2nd_last_archived_0_24m | categorical |
| status_3rd_last_archived_0_24m | categorical |
| status_max_archived_0_6_months | categorical |
| status_max_archived_0_12_months | categorical |
| status_max_archived_0_24_months | categorical |
| recovery_debt | numeric |
| sum_capital_paid_account_0_12m | numeric |
| sum_capital_paid_account_12_24m | numeric |
| sum_paid_inv_0_12m | numeric |
| time_hours | numeric |
| worst_status_active_inv | categorical |

# Chapter 2

# Introduction

## 2.1 Background

This problem is technically defined as a credit risk problem and generally speaking estimating the credit risk has three main components namely

- Exposure This refers to a party that may possibly default or suffer an adverse change in its ability to perform.
- Likelihood The likelihood that this party will default on its obligations.
- Recovery rate How much can be retrieved if a default takes place

In this particular project, we will focus on creating a machine-learning model to focus to evaluate the probability of default corresponding to users.

# Chapter 3

# Approach

In this chapter, I will discuss the complete approach to the problem and the code that is implemented in the API deployment. Moreover, I will walk through every decision and choice I made and substantiate the reason behind those choices

## 3.1 Step by Step Implemented solution

1. load dataset remove the data without 'default' column and name it as test and use the remaining data and name is as 'train'

2. Exploratory Data Analytics

    - Plot percentage of NaN data per column
    - Visualizing Column Data Distribution for Numerical Columns

3. Understanding Features

    - Dealing with NaN values
        - Bar graphs for visualizing probability distribution for categorical columns with Nans
        - selecting mode for filling NaNs and reasoning behind the decision was very high frequency of mode relative to data, therefore the inherent data distribution is not much affected
        - selecting mean for other numerical columns

- Dealing with Categorical columns (explanation discussed in detail in next chapter)
    - strategy 1: Cols with less than or equal to 5 unique features we will go with label encoding and rest with one hot encoding
        - pros: dimensionality reduction
        - cons: label encoding may lead to bias as some features are treated more important than others
    - Strategy 2: all categorical cols will have one hot encoding, especially relevant when using XGboost.
        - pros: No bias in models, as all features are treated equally important
        - cons: dimensionality explosion

    **strategy 2 was used for the deployed code**

4. Feature Exploration: wetting the possibility of dimensionality reduction by exploring feature importance

    - feature importance using Shapely Additive Explanations (ShAP)

5. Training

    - Deal with Imbalanced data by using a combination of SMOTE-ENN (Synthetic Minority Oversampling Technique)-(Edited Nearest Neighbor)
    - Dimensionality Reduction: For the convenience of training 5 categorical and 5 numerical features are used, the numerical features are the top 5 from shapely analysis
    - Model Training (XGBOOST): Finding right parameters by GridsearchCV
    - Analysing Training Results using ROC AUC curve
    - Take all the code above and create functions to make it all in one file and make a Flask app or Fast API APP
    - Test the app in local environment
    - Make a Docker Container
    - set up ECR in AWS and do the deployment in AWS lambda

# Chapter 4

# Results, Experiments,and Reasoning behind decisions made in the Approach

## 4.1 Observations: Filling of Missing Values

The below are categorical columns that have NaN values we can observe the following:

Overall we see that these 6 columns have maximum frequency as 1.0, and the order of frequency points also go by 1,2,3,4 and we must remember that this data is only based on few data points that we have as most data points are NaNs

- account worst status 12 24m has 66 % missing values
- account status has 54 % missing values
- account worst status6 12m has 60 % missing values
- account worst status03m has 54 % missing values
- account worst status 3 6m has 57 % missing values
- worst status active inv has 70 % missing values

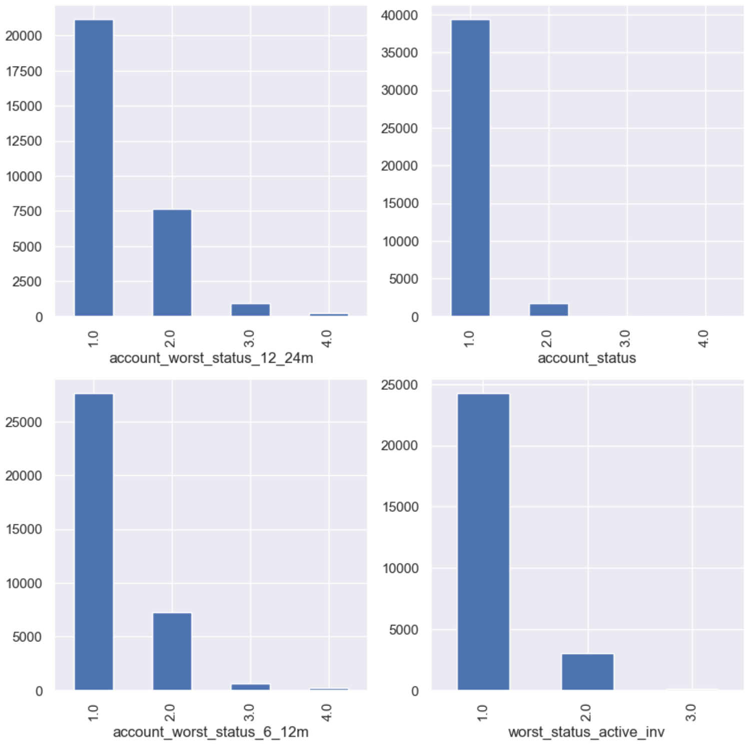


Figure 4.1: Bar graph for categorical columns that have NaN values (plot 1)

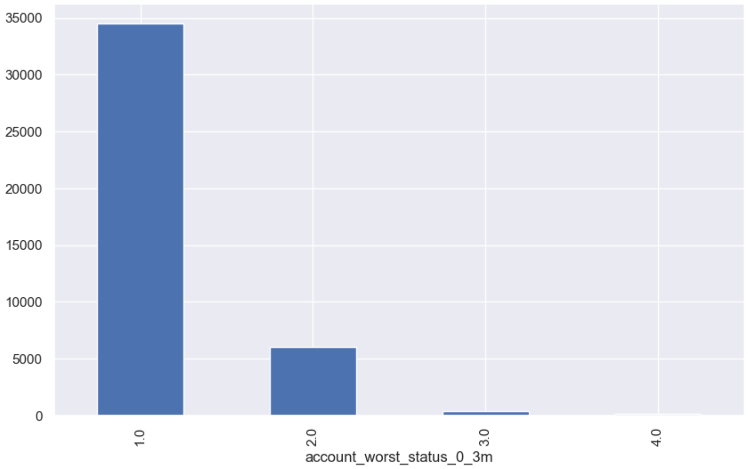


Figure 4.2: Bar Graph for the categorical column that has Nan Values (plot 2)

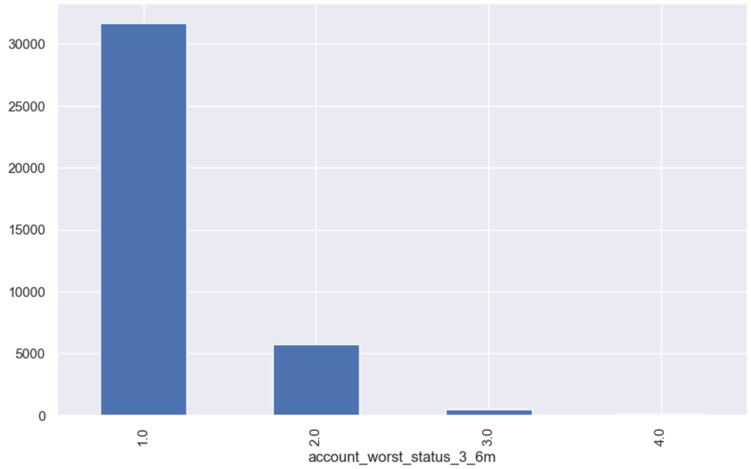


Figure 4.3: Bar Graph for the categorical column that has Nan Values (plot3)

The mode or the first highest frequency element is way higher in count than the second highest frequency, therefore I decided to fill the NaN values with mode as this preserves the data distribution moreover when plotting default and non-default users separately, we see a similar trend (below) therefore it is reasonable to substitute the NaNs with mode.

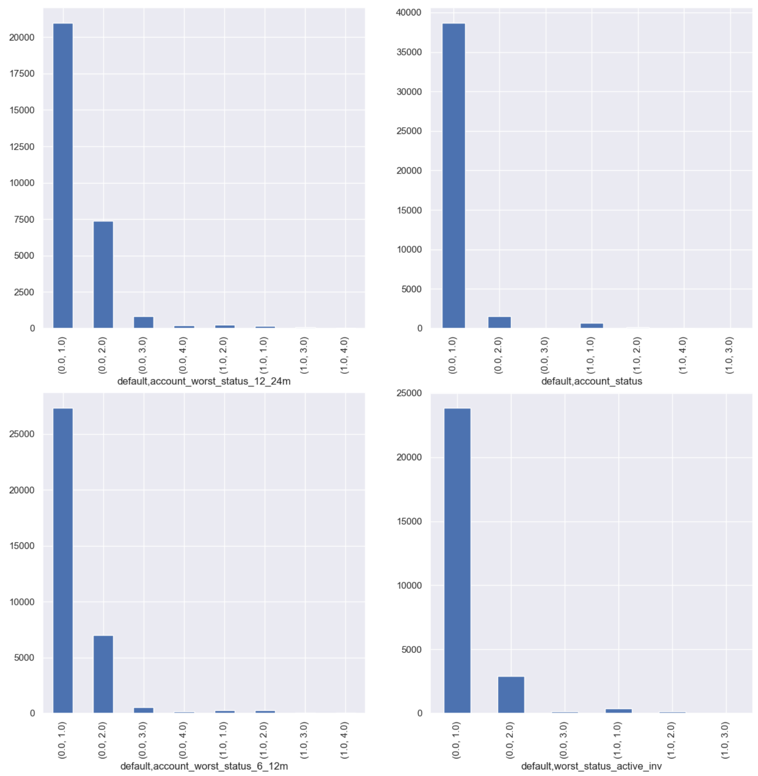


Figure 4.4: Bar Graph for the categorical column that has Nan Values this time distributed with default and non-default users (plot 1)

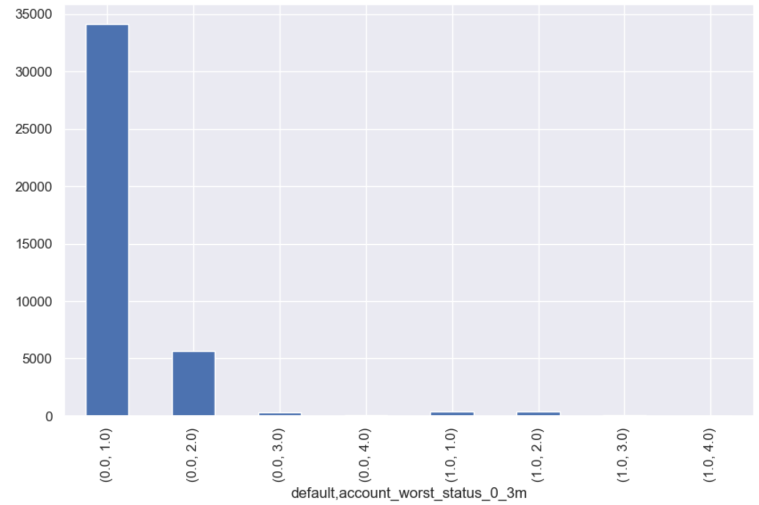


Figure 4.5: Bar Graph for the categorical column that has Nan Values this time distributed with default:1 and non-default:0 users (plot 2)

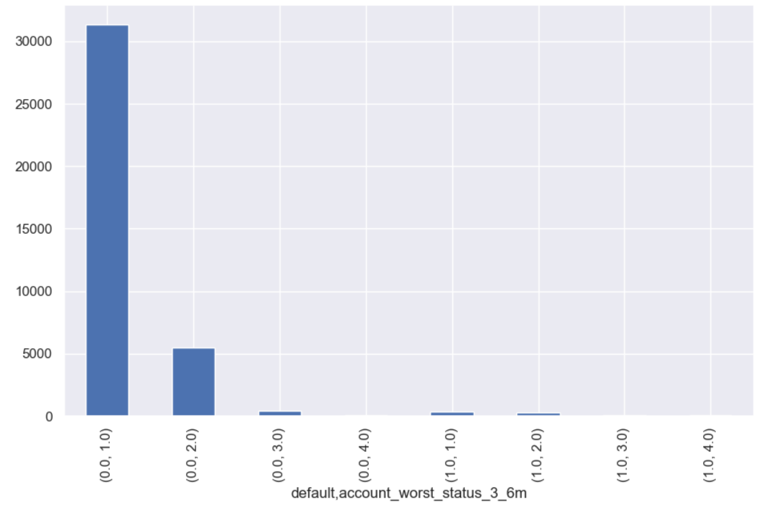


Figure 4.6: Bar Graph for the categorical column that has Nan Values this time distributed with default and non-default users (plot 3)

For Other columns, we see how much is the value counts or the weightage of the mode and we find 'account days in dc 12 24m', 'account days in rem 12 24m',' num arch written off 12 24m', 'account days in term 12 24m', 'num arch written off 0 12m', the mode represents 87%, 78%, 81.8%, 87% and 82% of the dataset for these columns respectively, therefore, we impute these columns with their respective modes.

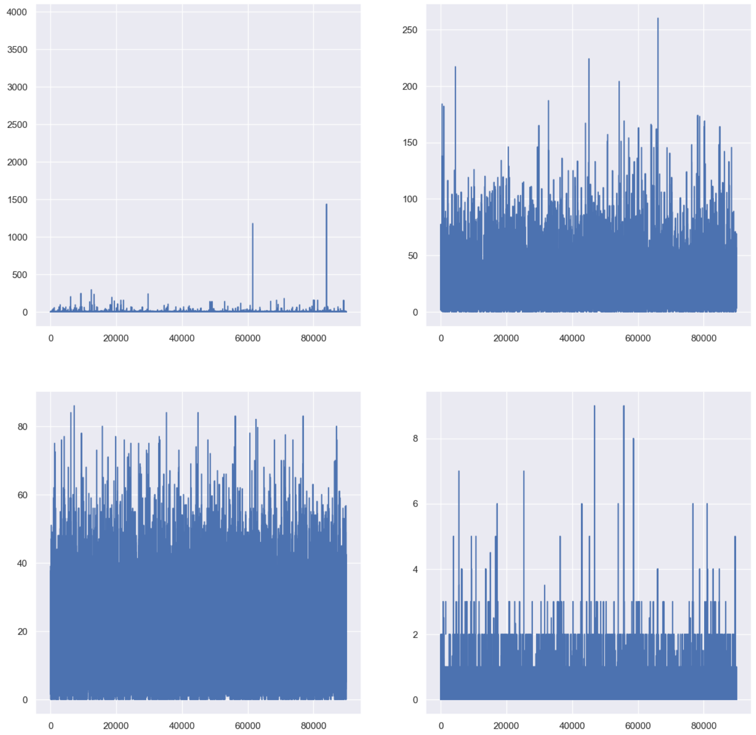


Figure 4.7: ploting value for every user for the 4 numerical columns: 'account incoming debt vs paid 0 24m', 'avg payment span 0 12m','avg payment span 0 3m','num active div by paid inv 0 12m'

For these 4 numerical columns, the mean looked like a reasonable substitution for Nan values

## 4.2 General Observations

- It is an Imbalanced dataset as expected as most people do not default and only few default, from the train dataset we have 88688 users (98.56%) with default:0 and 1288 users (1.43%) with default:1 and total of 89976.

- In total 43 columns, with 1 unique id ('uuid') for every customer which is text, 1 target column ('default') categorical and 15 categorical columns , 25 numerical columns and 1 boolean column

- Several columns with missing value data, here is the list with the corresponding percentage of missing values: missing_value_columns.txt

- for example some columns such as 'account worst status 3 6m' has around 57% missing values, 'account worst status 12 24m' has 66% and the entire list can be obtained in the txt file 'missing_value_columns.txt'

- train is the data with non-zero 'default' column as this is the only data that we can train on but note that for actually training we will use a subset of this data to do the validation and testing

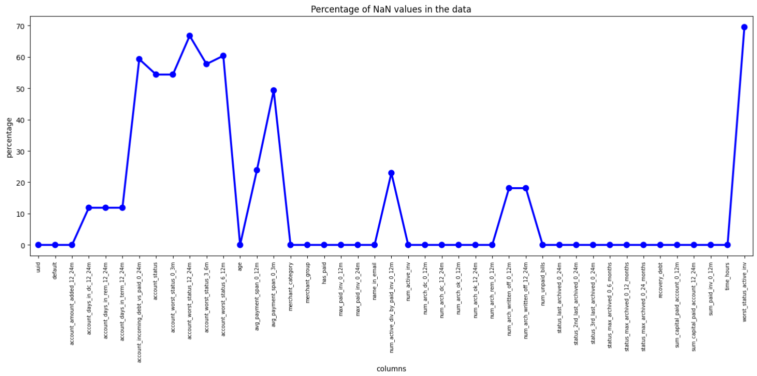


Figure 4.8: Percentage of NaN values in missing data

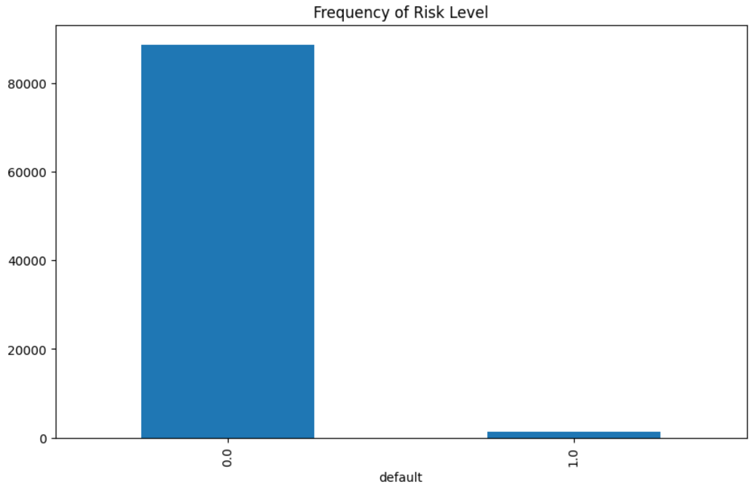


Figure 4.9: level of Data Imbalance

## 4.3 Observation: Shapley additive explanations

Only for the numerical values, I performed Shapley Additive explanations analysis which is found in the Jupiter notebook - shap_feature_importance.ipynb

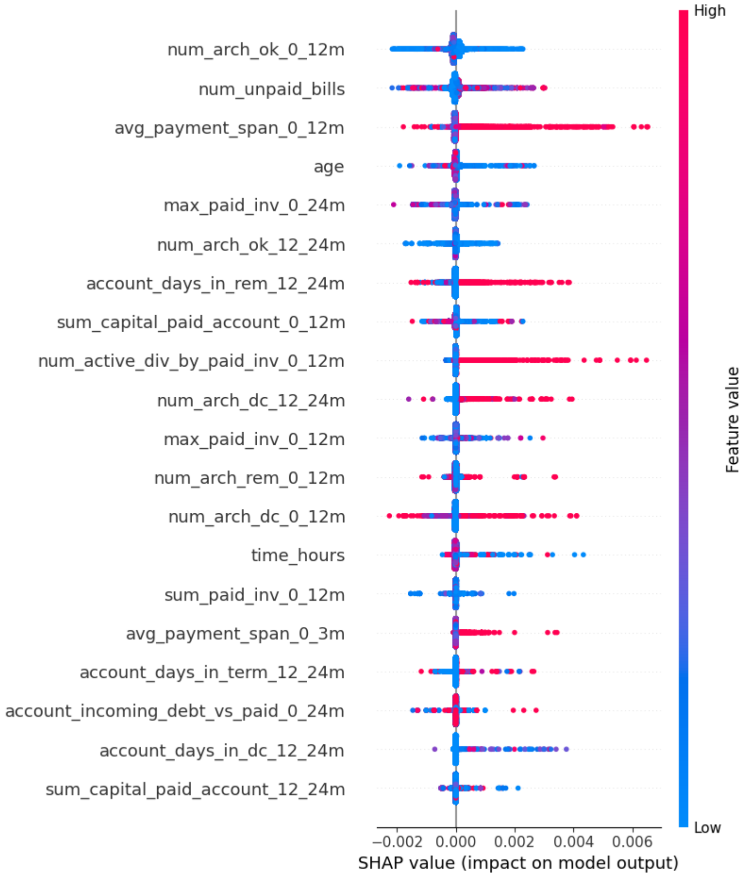


Figure 4.10: SHAP analysis (1)

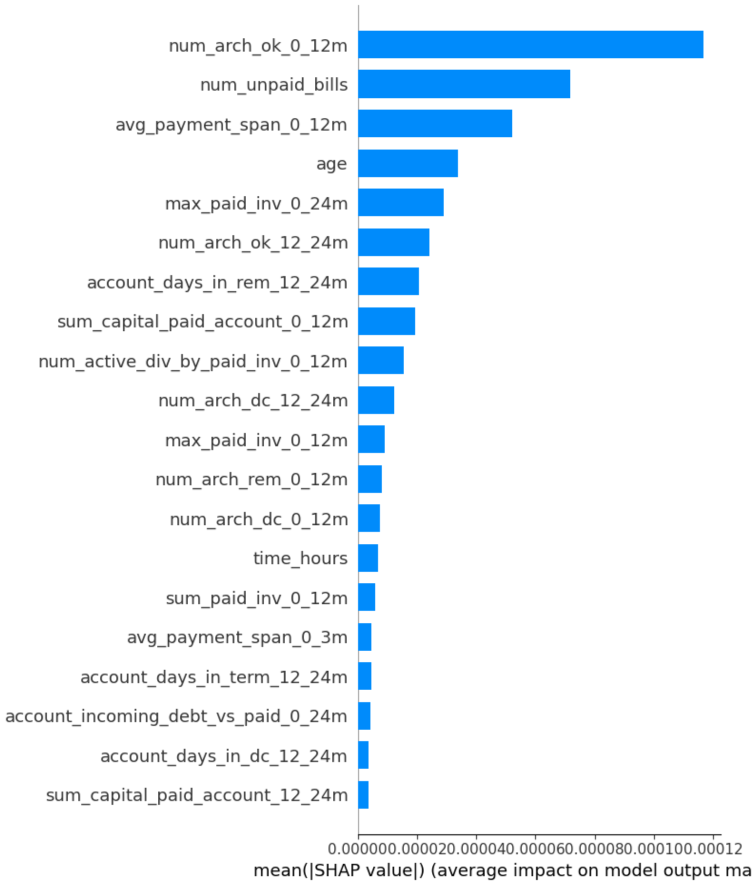


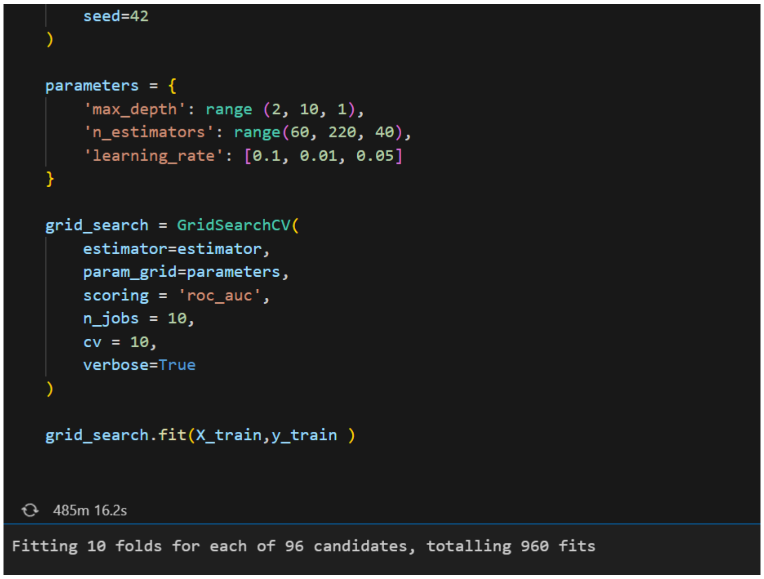
Figure 4.11: SHAP analysis feature importance bar (1)

## 4.4 Experiments using XGBOOST

I used Grid Search CV for finding right parameters but it was taking a long time but so I ditched that idea and simply did three experiments to save time

Firstly I took numerical features from the SHAP analysis and within categorical features choose amongst features that have less than 5 unique values inorder to save dimensionality explosion when using pd.get_dummies(). The experiments performed were as follows

1. Took top 2 numerical features from SHAP and 2 categorical features, trained xgboost and got ROC-AUC of 0.58
2. took top 3 numerical features from SHAP and 3 catgorical features, trained xgboost and got ROC-AUC of 0.64
3. took top 5 numerical features from SHAP and 5 categorical features, trained xgboost and got ROC-AUC of 0.81. (**This was the model deployed in AWS**)



```
    seed=42
)

parameters = {
    'max_depth': range (2, 10, 1),
    'n_estimators': range(60, 220, 40),
    'learning_rate': [0.1, 0.01, 0.05]
}

grid_search = GridSearchCV(
    estimator=estimator,
    param_grid=parameters,
    scoring = 'roc_auc',
    n_jobs = 10,
    cv = 10,
    verbose=True
)

grid_search.fit(X_train,y_train )
```

485m 16.2s

```
Fitting 10 folds for each of 96 candidates, totalling 960 fits
```

Figure 4.12: taking huge time:485 minutes

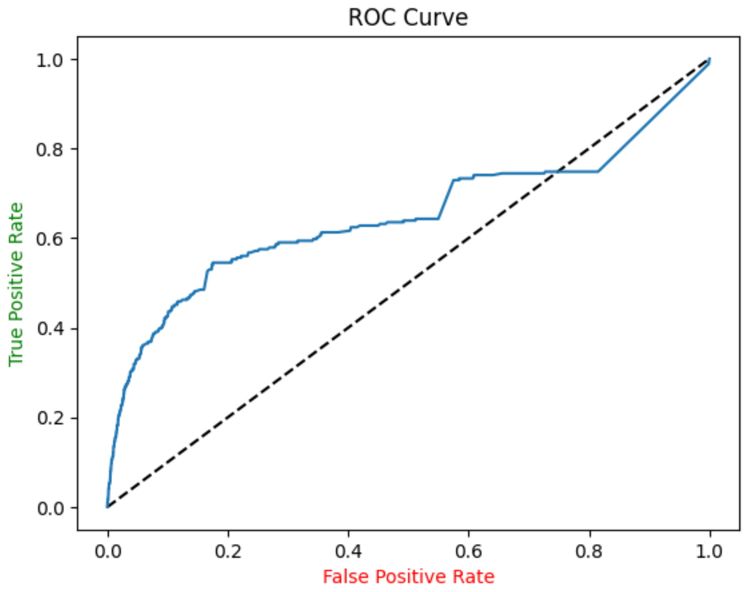


Figure 4.13: roc-auc for 3 categorical and 3 numerical columns

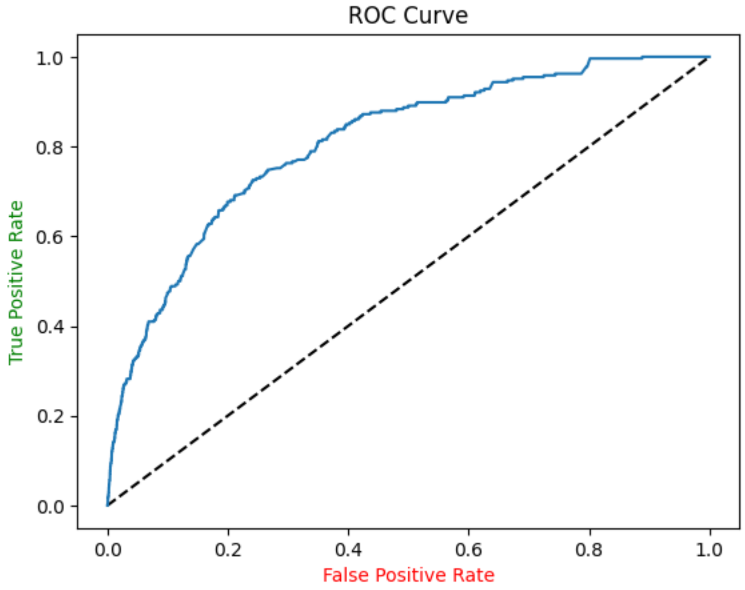


Figure 4.14: roc-auc for 5 categorical and 5 numerical columns

# Chapter 5

# Instructions on using Model exposed as API

### 5.0.1 Instructions

I have considered 10 columns which are as follows:

- ’num_arch_ok_0_12m’,
- ’num_unpaid_bills’,
- ’avg_payment_span_0_12m’,
- ’age’,
- ’max_paid_inv_0_24m’,
- ’account_status’,
- ’account_worst_status_0_3m’,
- ’account_worst_status_12_24m’,
- ’account_worst_status_3_6m’,
- ’account_worst_status_6_12m’

The solution is designed in a way that as in the case of training data, there were non-empty columns and some columns with NaN values, similarly the solution assumes full available data for non-empty columns, therefore columns such as

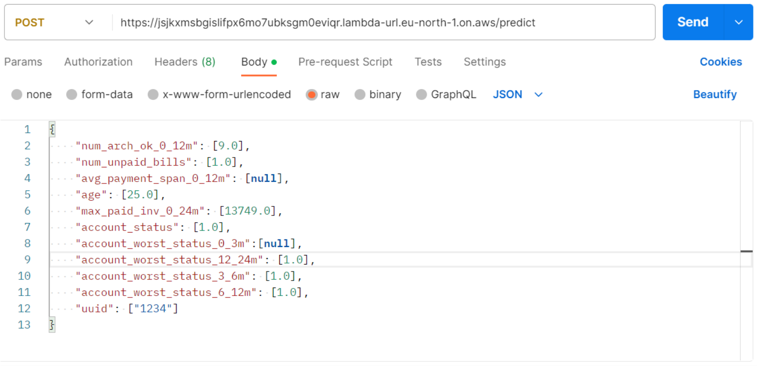
- 'num_arch_ok_0_12m'
- 'num_unpaid_bills',
- 'age',
- 'num_arch_ok_0_12m'
- uuid

are mandatory and other columns can be null.
**Remember that even if all columns are null Xgboost will still give a result but for a good estimate please consider above instructions**

The API was exposed within AWS lambda and the URL is as follows:

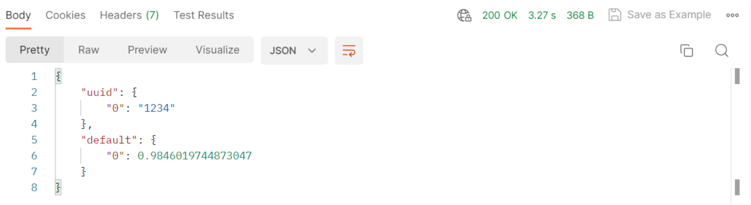
https://jsjkxmsbgislifpx6mo7ubksgm0eviqr.lambda-url.eu-north-1.on.aws/predict

Example Input and Output



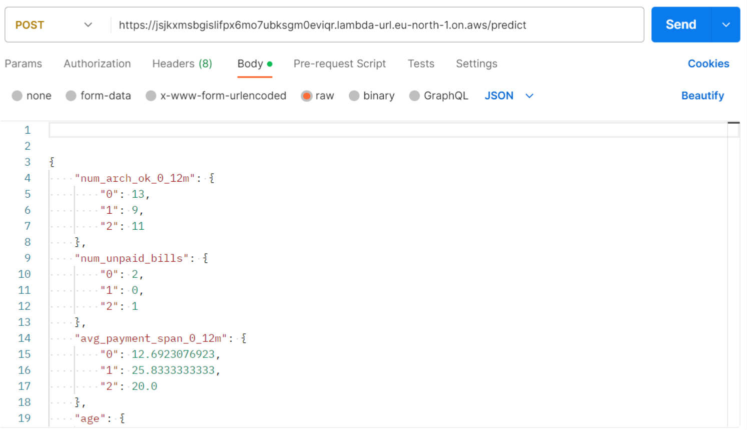
```json
{
    "num_arch_ok_0_12m": [9.0],
    "num_unpaid_bills": [1.0],
    "avg_payment_span_0_12m": [null],
    "age": [25.0],
    "max_paid_inv_0_24m": [13749.0],
    "account_status": [1.0],
    "account_worst_status_0_3m":[null],
    "account_worst_status_12_24m": [1.0],
    "account_worst_status_3_6m": [1.0],
    "account_worst_status_6_12m": [1.0],
    "uuid": ["1234"]
}
```

Figure 5.1: postman example single input



```
{
    "uuid": {
        "0": "1234"
    },
    "default": {
        "0": 0.9846019744873047
    }
}
```

Figure 5.2: Postman example output 1

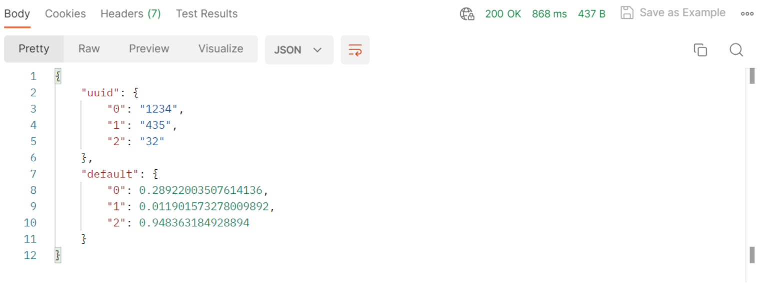


```
https://jsjkxmsbgislifpx6mo7ubksgm0eviqr.lambda-url.eu-north-1.on.aws/predict
```

```


{
    "num_arch_ok_0_12m": {
        "0": 13,
        "1": 9,
        "2": 11
    },
    "num_unpaid_bills": {
        "0": 2,
        "1": 0,
        "2": 1
    },
    "avg_payment_span_0_12m": {
        "0": 12.6923076923,
        "1": 25.8333333333,
        "2": 20.0
    },
    "age": {
```

Figure 5.3: postman example multi-input



```
{
    "uuid": {
        "0": "1234",
        "1": "435",
        "2": "32"
    },
    "default": {
        "0": 0.28922003507614136,
        "1": 0.011901573278009892,
        "2": 0.948363184928894
    }
}
```

Figure 5.4: Postman example output

# Chapter 6

# Discussion and Future Work

## 6.1 Ideas for Future work

Risk Bucketing is an idea of grouping borrowers with similar creditworthiness. As having a single machine-learning model for all the data could lead to bad predictions. Therefore bucketing similar groups of people together and then applying different models to different groups could lead to better results using machine learning.

In an attempt to find these similar groupings, I applied the Elbow method to find the number of clusters in this dataset for applying k means clustering. For the same purpose, I used the 25 numeric columns.
The Elbow method suggests that at no particular k at which the curve becomes flat and therefore in this data could be homogeneous or a part of a single cluster. Another validation is done by silhouette plot for k=2 and k=3 and we clearly see a very poor score, indicating that this data more or less belongs to the same cluster, however, further investigations should be made.

Moreover, several ways to optimize the model should be explored as a part of future work. Some of the other classification algorithms that can be applied are as follows

- Logistic Regression
- Bayesian Model
- Support Vector Machine

- Random Forests
- Deep learning

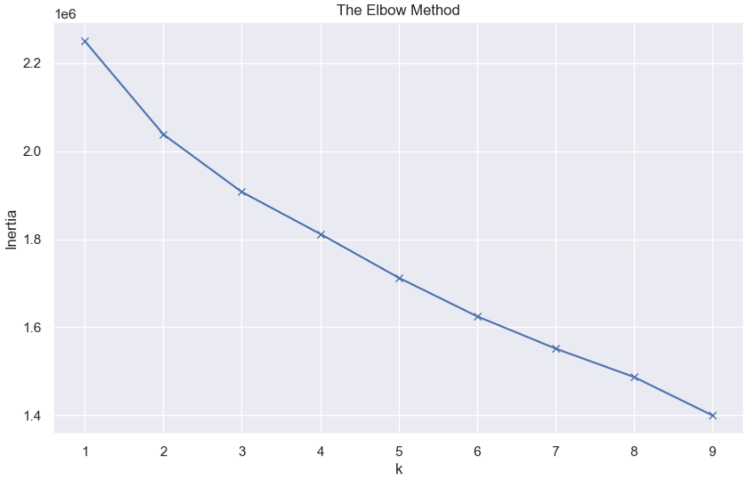


Figure 6.1: Elbow method to find the number of clusters

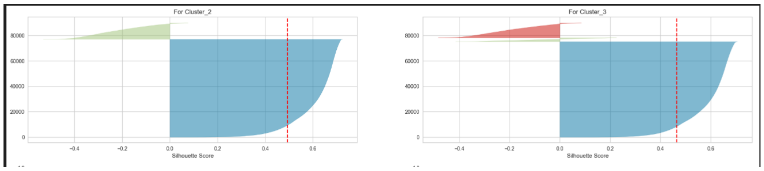


Figure 6.2: Silhouette Method to find the number of clusters