# ML Project Report: Financial Risk Profiling (Checkpoint 2)

Team Members

1. Priyanshu Tiwari (IMT2023010)
2. Siddharth Kini (IMT2023026)

**Github Link**: https://github.com/d3monviking/financial-risk-profiling

## Project Task

The primary objective of this project was to develop and evaluate various machine learning models for accurately predicting borrower default probability based on personal and Financial Indicators, thereby making it a binary classification problem. To achieve this, we did:

- **Data Pre-processing:** This involved cleaning the dataset, handling missing values, encoding categorical features, scaling numerical features, and feature engineering new features to prepare the data for model training.
- **Exploratory Data Analysis (EDA):** We conducted a thorough EDA to understand the distribution of features, identify correlations, and uncover insights into factors influencing the risk profiling.
- **Model Training and Evaluation:** We trained and evaluated various machine learning models suitable for classification tasks, including logistic regression, support vector machines, random forests, neural networks, and naive Bayes.

## Dataset and Features Description

The financial risk profiling dataset contains detailed borrower loan records capturing credit and loan-related attributes that contribute to assessing financial risks. Financial risk profiling is one of the important tasks undertaken by banks and money lenders, which helps them make informed decisions regarding loan disbursements. Therefore, risk profiling becomes of utmost importance. To achieve the most accurate and efficient prediction, we need to examine past data, i.e., the already evaluated loan applicants, along with the attributes of the applicants that support the evaluation. We begin the process by understanding the dataset, which comprises 204,277 rows and 18 columns. The columns include the applicant’s attributes, ranging from annual earnings to the qualification level.

## EDA and Preprocessing

Before training our models and making predictions, we need to analyse, understand and perform a sanity check of the training and testing data. We begin by conducting Exploratory Data Analysis (EDA), which helps us understand the quality of the features and their relationship to the target value.

### Importing Libraries and Data Overview

We begin our process by importing the necessary libraries, including NumPy, Pandas, Matplotlib, and Seaborn. These libraries are useful for loading the dataset into our program (using a Pandas DataFrame) and manipulating it for cleaning and evaluating the quality of the dataset and its features.

### Data Overview

After loading the dataset, we use the .info() method to get an overview of the training data:

- Dimensions: 204277 rows and 18 columns.
- Data Types: A mix of 2 float64, 8 int64, and 8 object (categorical) columns.

### Feature Engineering

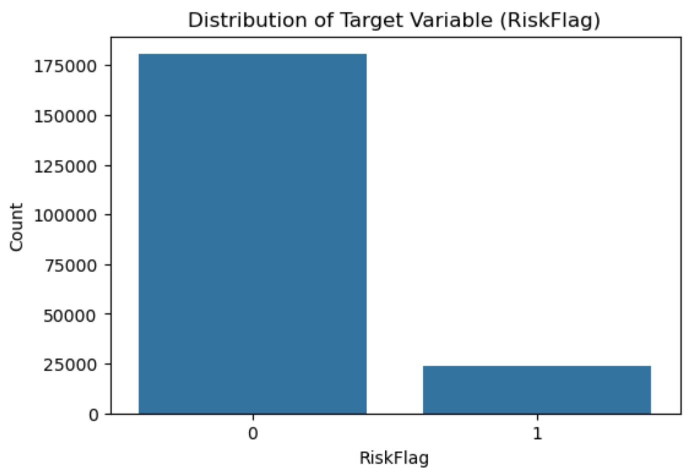
```
# Financial ratios
combined_df['Loan_to_Income'] = combined_df['RequestedSum'] /
combined_df['AnnualEarnings']
combined_df['EMI_Approx'] = combined_df['RequestedSum'] /
combined_df['RepayPeriod']
combined_df['EMI_to_Income'] = combined_df['EMI_Approx'] /
(combined_df['AnnualEarnings'] / 12)
combined_df['Debt_to_Loan'] = combined_df['DebtFactor'] /
(combined_df['RequestedSum'] + 1e-6)
combined_df['Earnings_per_Account'] = combined_df['AnnualEarnings'] /
(combined_df['ActiveAccounts'] + 1e-6)
combined_df['MonthlyIncome'] = combined_df['AnnualEarnings'] / 12
combined_df['Repayment_to_Earnings'] = combined_df['RequestedSum'] /
(combined_df['AnnualEarnings'] * combined_df['RepayPeriod'] / 12 + 1e-6)
combined_df['Margin_after_Debt'] = combined_df['MonthlyIncome'] -
(combined_df['DebtFactor'] * combined_df['MonthlyIncome'])

# Interaction features
combined_df['Trust_Qualification'] = combined_df['TrustMetric'] *
combined_df['QualificationLevel']
combined_df['WorkDuration_Qualification'] = combined_df['WorkDuration'] *
combined_df['QualificationLevel']

# Temporal and count features
combined_df['Log_WorkDuration'] = np.log1p(combined_df['WorkDuration'])
```

```
combined_df['Log_ActiveAccounts'] = np.log1p(combined_df['ActiveAccounts'])
combined_df['WorkDuration_Bin'] = pd.cut(combined_df['WorkDuration'],
bins=[-np.inf, 12, 60, np.inf], labels=[0,1,2])

# Age group bins: 18-30 young, 31-50 mid, 50+ senior
combined_df['Age_Group'] = pd.cut(combined_df['ApplicantYears'],
bins=[0,30,50,np.inf], labels=[0,1,2])

# Property and JointApplicant interaction
combined_df['Property_JointApplicant'] = combined_df['OwnsProperty'] *
combined_df['JointApplicant']
```

## Exploratory Data Analysis (EDA)

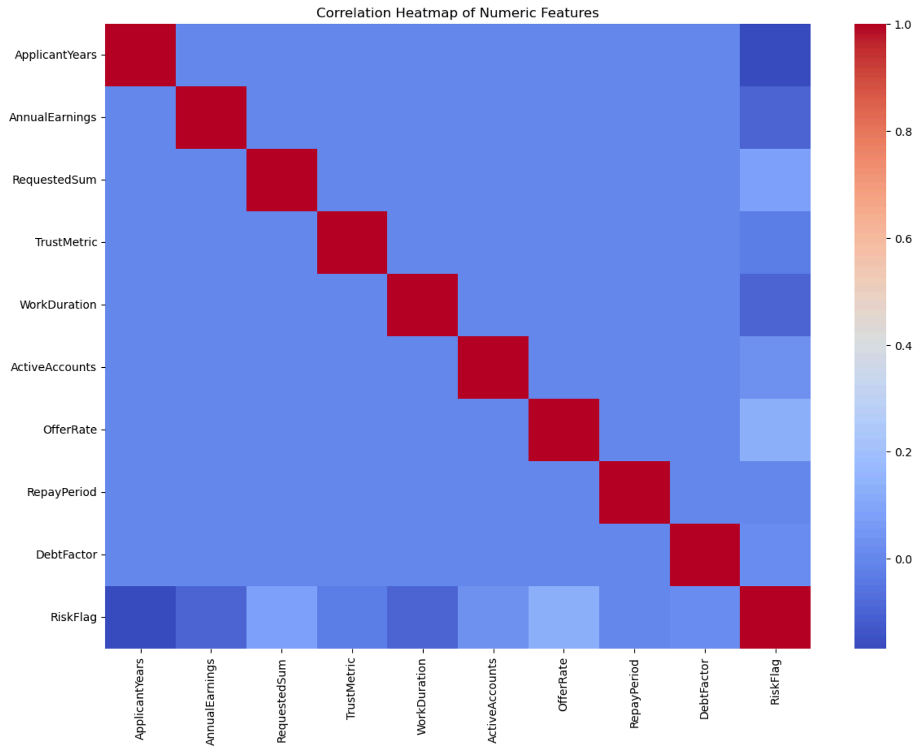
Various plots, including histograms, correlation heatmaps, scatterplots, and box plots, were generated to analyse the data distribution, detect outliers, and explore relationships among features. These visualisations provided critical insights that informed subsequent preprocessing and modelling decisions.

### Target Variable Histogram



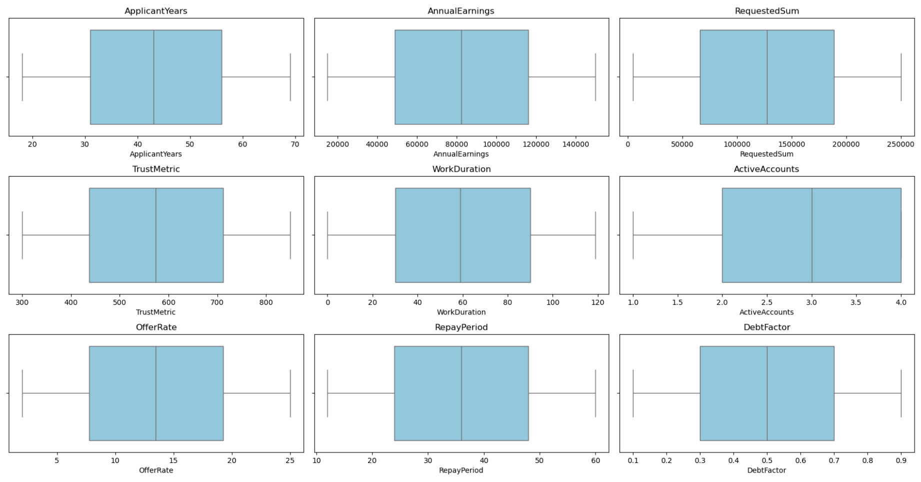
The histogram shows that the target variable RiskFlag is highly imbalanced, with a vast majority of samples labelled 0 (non-risky customers) and only a small fraction labelled 1 (risky customers). We infer that most customers in our training data are low-risk, and we need to preprocess our data and tune our model so that this bias is not observed when predicting the majority class.

## Correlation Matrix (Heatmap)



The correlation heatmap shows that the target variable has very weak correlations with almost all the numerical columns. A few features, such as RequestedSum, ActiveAccounts, and OfferRate, exhibit a slightly higher correlation with the target. We can also see that the features exhibit very low correlation with each other, indicating little multicollinearity.

## Numeric Features vs Target Value (Boxplots)



The boxplots show that most numerical features have distributions that are relatively symmetric with moderate spread. Only ActiveAccounts are skewed towards the higher value, suggesting that most applicants have around 2 to 4 accounts. Overall, the numerical features appear clean without any severe outliers, suggesting the dataset is relatively stable for modelling.

### Data Cleaning

Upon analysing our training and test datasets, we found that neither of them contained any null values. Therefore, we did not need to drop any columns or rows, nor did we need to impute any new values. All the features had the same amount of correlation with each other, so we decided not to drop any columns.

## Data Preprocessing

During our preprocessing phase, we:

- We did not find any null values in our training dataset.
- Applied ordinal encoding for quality-based features, target encoding for high-cardinality categorical columns, and one-hot encoding for low-cardinality ones.
- We performed feature engineering to create meaningful ratio-based features from existing ones, such as Loan-to-Income and EMI Approx, providing a clearer understanding of the loan application and the applicant's status.
- Finally, all numerical features were scaled using StandardScaler to ensure equal contribution across the model.

### Splitting The Dataset

For training our machine learning models, we split our dataset into training and testing sets. The split ratio is 80% for training and 20% for testing. We do this to evaluate the model's performance on unseen data and tune it accordingly.

## Models Used For Prediction

## Logistic Regression

- The preprocessing done on the training and testing data was the same as previously mentioned (in the EDA and Preprocessing section).
- We performed logistic regression using scikit-learn's LogisticRegression() model with class_weight='balanced', max_iter=1000.
- Our model was evaluated across thresholds from 0.5 to 0.95, showing a clear trade-off:
  - Low thresholds (0.5–0.6) - Higher recall for class 1, but lower accuracy.
  - Mid thresholds (0.7–0.8) - Balanced performance with strong F1-score for class 0 and moderate recall for class 1.
  - High thresholds (0.85–0.95) - Very high accuracy due to prioritising class 0, but severe decline in recall for class 1.
- We decided to keep the threshold = 0.9, because that's where our model reached the peak average accuracy of 0.89 on our validation data. The 0.9 threshold gave an accuracy of 0.887 on the test data.

```
---------------------------- Threshold: 0.9 -------------------------------------
               precision    recall  f1-score   support

           0       0.89      1.00      0.94     36105
           1       0.64      0.06      0.11      4751

    accuracy                           0.89     40856
   macro avg       0.76      0.53      0.52     40856
weighted avg       0.86      0.89      0.84     40856
```

## Naive Bayes

- The preprocessing done on the training and testing data was the same as previously mentioned (in the EDA and Preprocessing section).
- We used the Gaussian Naive Bayes model from the scikit-learn library and trained our data on it. Naive Bayes gave us an accuracy of 0.83 on our validation data and a score of 0.829 on the test data.

```
                Classification Report
            precision    recall  f1-score   support

           0       0.90      0.90      0.90     36073
           1       0.27      0.28      0.28      4783

    accuracy                           0.83     40856
   macro avg       0.59      0.59      0.59     40856
weighted avg       0.83      0.83      0.83     40856
```

## Random Forest

- For preprocessing, we decided to keep the feature engineering to a minimum, as the addition of new features didn't give us fruitful results. Apart from that, the encoding and scaling of the dataset are the same as above. The only new features used here were:

```
combined_df['Loan_to_Income'] = combined_df['RequestedSum'] /
combined_df['AnnualEarnings']
combined_df['EMI_Approx'] = combined_df['RequestedSum'] /
combined_df['RepayPeriod']
combined_df['EMI_to_Income'] = combined_df['EMI_Approx'] /
(combined_df['AnnualEarnings'] / 12)
```

- Upon applying SearchCV for the random forest, the best parameters for it are as follows:
  - Best params: {'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 403}

- Our random forest gave us an accuracy score of 0.88 on the validation set. We received a score of 0.885 on the test data.

```
                      Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     36105
           1       0.52      0.08      0.14      4751

    accuracy                           0.88     40856
   macro avg       0.71      0.54      0.54     40856
weighted avg       0.85      0.88      0.85     40856
```

## Support Vector Machine

- We trained our Neural Networks model with the data preprocessed similarly to the Random Forest.
- We trained our Support Vector Machine with different kernels, including the linear kernel, Sigmoid kernel, and RBF kernel. The sigmoid kernel yielded the worst accuracy on our validation data, whereas the RBF kernel performed the best with an accuracy of 0.7. When we ran this model on the test data, it gave an accuracy of 0.701.

```
                        Classification Report
               precision    recall  f1-score   support

           0       0.94      0.70      0.81     36105
           1       0.23      0.67      0.34      4751

    accuracy                           0.70     40856
   macro avg       0.59      0.69      0.57     40856
weighted avg       0.86      0.70      0.75     40856
```

## Neural Networks

- We trained our Neural Networks model with the data preprocessed similarly to the Random Forest.
- Upon applying SearchCV for the random forest, the best parameters for it are as follows:
  - {'activation': 'relu', 'alpha': 0.012151273301300586, 'batch_size': 128, ‘early_stopping ’: True, 'hidden_layer_sizes': (256, 128, 64), 'learning_rate': 'adaptive', 'max_iter': 300, 'solver': 'adam'}
- It achieved an accuracy of 0.77 on the validation data and 0.776 on the test data.

```
                    Classification Report
              precision    recall  f1-score   support

           0       0.91      0.82      0.86     36105
           1       0.23      0.42      0.30      4751

    accuracy                           0.77     40856
   macro avg       0.57      0.62      0.58     40856
weighted avg       0.84      0.77      0.80     40856
```

## Training on 20% of the training data - SVM vs Neural Networks

- When we trained our SVM and Neural Networks on a smaller dataset (20% of our original training dataset), we did not observe any significant change in our output. Theoretically, SVM should have performed better than Neural Networks on a smaller dataset, but Neural Networks still have a better accuracy.
- On the smaller dataset, SVM showed an improvement of 0.01 score on the validation data (with an accuracy of 0.71) and achieved an accuracy score of 0.712 on the test data.

```
                   Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.94      0.72      0.82    144420
           1       0.23      0.65      0.34     19002

    accuracy                           0.71    163422
   macro avg       0.59      0.68      0.58    163422
weighted avg       0.86      0.71      0.76    163422
```

- Neural Networks' performance dropped by 0.04 on the test data, and it gave an accuracy score of 0.772 on the test data.

```
                 Classification Report (Neural Networks)
              precision    recall  f1-score   support

          0        0.90      0.83      0.87    144420
          1        0.20      0.33      0.25     19002

accuracy                               0.77    163422
macro avg        0.55      0.58        0.56    163422
weighted avg     0.82      0.77        0.79    163422
```

## Performance and Interpretation of Different Models

- Logistic Regression performed the best overall, achieving the highest accuracy and most stable results, thanks to practical feature engineering.
- Random Forest was just behind logistic regression in terms of robust accuracy. Its performance is likely due to the low correlation/non-linear relation between the target variable and the features.
- Despite being a simple model, Naive Bayes outperformed SVM and Neural Networks. This is likely due to the low correlations between the features and the target, as well as weak and scattered signals.
- SVM and Neural Networks underperformed due to the weak and linear signal between the features. The strong imbalance in the target variable also contributed to the poor results.

## Interesting Observations

- The target variable RiskFlag is highly imbalanced, with nearly 88% of samples belonging to the non-risky class. This imbalance influences our models; therefore, we have to tune the parameters and thresholds to obtain accurate results.
- All the features in our dataset showed low correlation with the target variables.
- Logistic Regression outperformed more complex models, indicating that the engineered ratio-based and interaction features created a largely linear decision boundary.
- Neural Networks and SVMs struggled with this dataset, even after hyperparameter tuning, suggesting that the dataset’s predictive patterns are not highly nonlinear.
- Feature engineering played a key role; ratio-based features, such as Loan_to_Income, EMI_to_Income, and Debt_to_Loan, were among the stronger predictors, confirming their relevance in financial risk assessment.