T.C.

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

INTRODUCTION TO MACHINE LEARNING

CSE 4288

Term Project

Student

150121824 – Burak KARAYAĞLI

150121539 – Gülsüm Ece GÜNAY

150120027 – Mert MUSLU

150120051 – Erkut DÖNMEZ

150120026 – Ardacan ÖZENER

Lecturer

Assoc. Prof. Murat Can GANİZ

1. Description

The provided Python script focuses on splitting the **loan approval dataset** into three subsets: **training**, **validation**, and **testing** data. The split is carefully performed in a stratified manner based on the loan_status column, which indicates loan approval (1) or rejection (0).

1.1 Stratification Based on loan_status:

The dataset is split into two subsets:

- loan_status_yes: Contains rows where loan_status = 1.
- loan_status_no: Contains rows where loan_status = 0.
  These subsets are then reset using reset_index() for clean indexing.

1.2 Index Creation for Train, Validation, and Test Splits:

The indices of rows for each subset are split into train, validation, and test sets. The **train-validation-test** split ratios are as follows:

- **70%** for training data,
- **10%** for validation data,
- **20%** for test data.
- For loan_status_yes (loan status = 1), there are **10,000** rows:
  - 70% → 7,000 rows for training
  - 20% → 2,000 rows for testing
  - 10% → 1,000 rows for validation
- For loan_status_no (loan status = 0), there are **35,000** rows:
  - 70% → 24,500 rows for training
  - 20% → 7,000 rows for testing
  - 10% → 3,500 rows for validation

1.3 The train_test_split function is used twice for each subset to achieve the desired ratios:

- First, 30% of the data is extracted as a temporary set (temp), while 70% remains for training.

- The temporary set is further split into **test (20%)** and **validation (10%)** using a 2:3 ratio.

1.4 Reconstruction of Train, Validation, and Test Data:

- Rows corresponding to the split indices are selected using “.iloc”.
- Rows from the approved loans (loan_status_yes) and rejected loans (loan_status_no) are combined using pd.concat() to form the final **train_data**, **validation_data**, and **test_data**.
- Reconstructed datasets exported into **train_data.csv, validation_data.csv, test_data.csv** respectively.

1.5 Categorical Data Encoding

One-Hot Encoding: Used for discrete, nominal categories like *person_home_ownership* and *loan_intent*, where each category was converted into separate binary columns. This ensures all categories are treated independently.

Label Encoding: Applied to binary and ordinal categorical variables like *person_gender*, *person_education*, and *previous_loan_defaults_on_file*. For example, education levels were mapped to progressive numerical values to preserve their order, while binary variables were directly mapped to 0 and 1.

1.6 Normalization

Min-Max Scaling: Applied to numerical columns such as person_age, person_income, loan_amnt, and others. This scaled all values to a range of [0, 1] to ensure consistent feature scaling, preventing features with larger ranges from dominating the model.

1.7 Numerical Data Encoding

Binning: It is aimed to divide numerical values into categorical value ranges. *KBinsDiscretizer* utilizes it with *Equal Frequency Binning*

(Quantile Binning) method is applied to numerical columns such as *person_age, person_income, person_emp_exp, loan_amnt, loan_int_rate, loan_percent_income, cb_person_cred_hist_length* columns.

Equal Frequency Binning: It divides data into certain groups (bins). In this way, each bin consists of approximately the same number of data points.

The number of bins is determined dynamically based on each column's Information Gain (IG) values relative to the target variable. Columns with lower IG values are assigned more bins. This approach prioritizes the creation of more detailed bins for features that contribute more to the prediction of the target variable.

Figure 1, which is shown below, demonstrates the dynamic distribution of equal-frequency binning. Each bin is divided according to the values of the class label, which corresponds to the *loan_status* column.

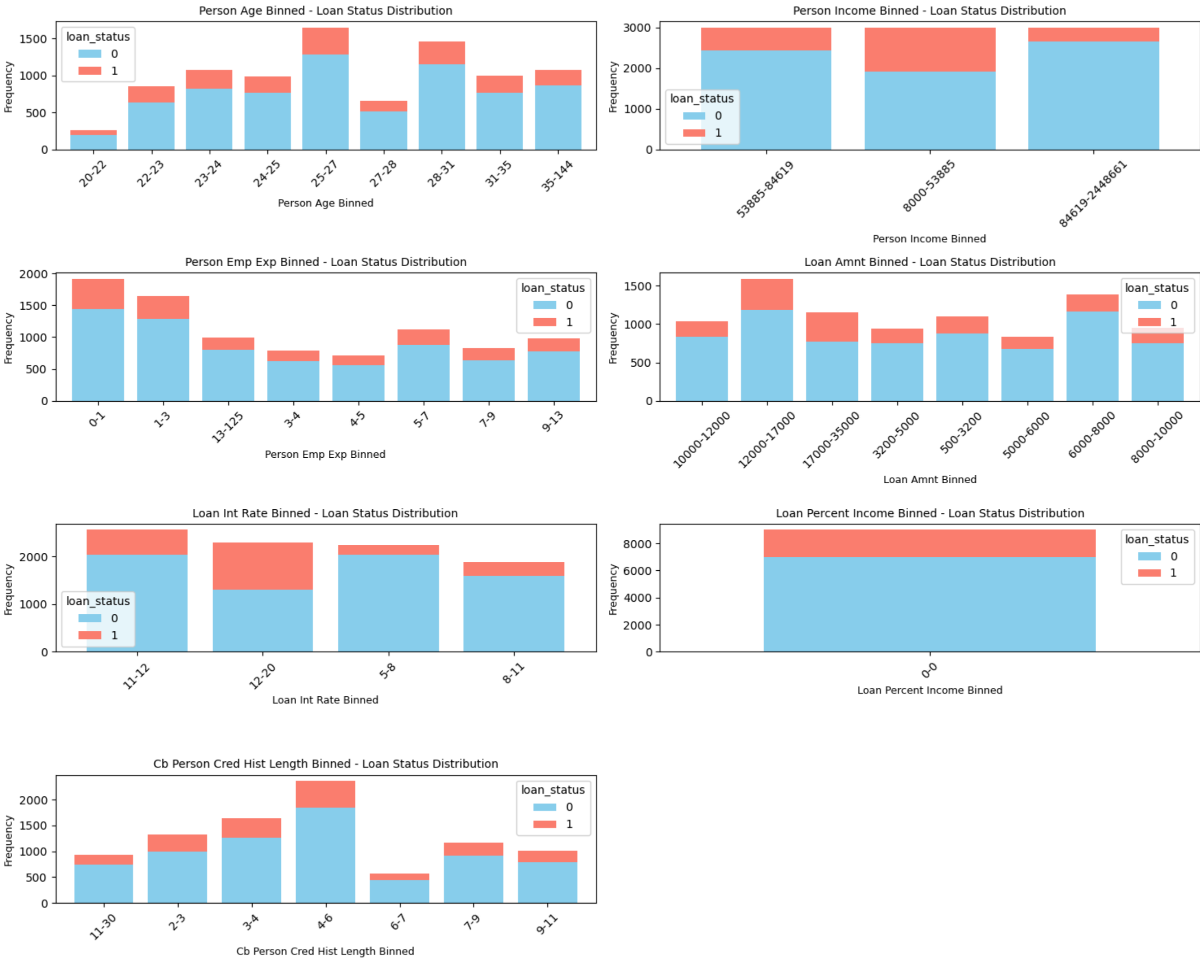


Figure 1- Histogram for Binned Columns with Loan Status Distribution

1.8 Trimming Outlier Values

The provided functions are used to detect and remove outliers based on the Interquartile Range (IQR) method, with adjustable thresholds for flexibility:

1. **determine_outlier_thresholds_iqr**:
   Calculates the lower and upper limits for outliers using the IQR, allowing customizable thresholds (**th1** and **th3**) for quartile calculations.
2. **check_outliers_iqr:**
   Identifies rows in the dataset where the values of a specified column fall outside the calculated lower and upper limits.
3. **remove_outliers_iqr:**
   Iterates over selected columns, detects the outliers then removes from the dataset. Also same method saves such values and displays them on a table:

- ○ Whether outliers were detected initially.
- ○ Whether any outliers remain after removal.
- ○ The number of removed outliers.
- ○ Column-specific lower and upper limits.

With the help of flexible threshold selection, it prevents over-trimming and corrupting the structure of the data set.

2. EDA Findings and Visualization

Figure 2 displays the graphical distribution of features from the [1] loan approval classification dataset. These visualizations in Figure 2 provide an exploratory data analysis (EDA) foundation, allowing us to observe trends, identify imbalances, and prepare the data for further processing and model training.

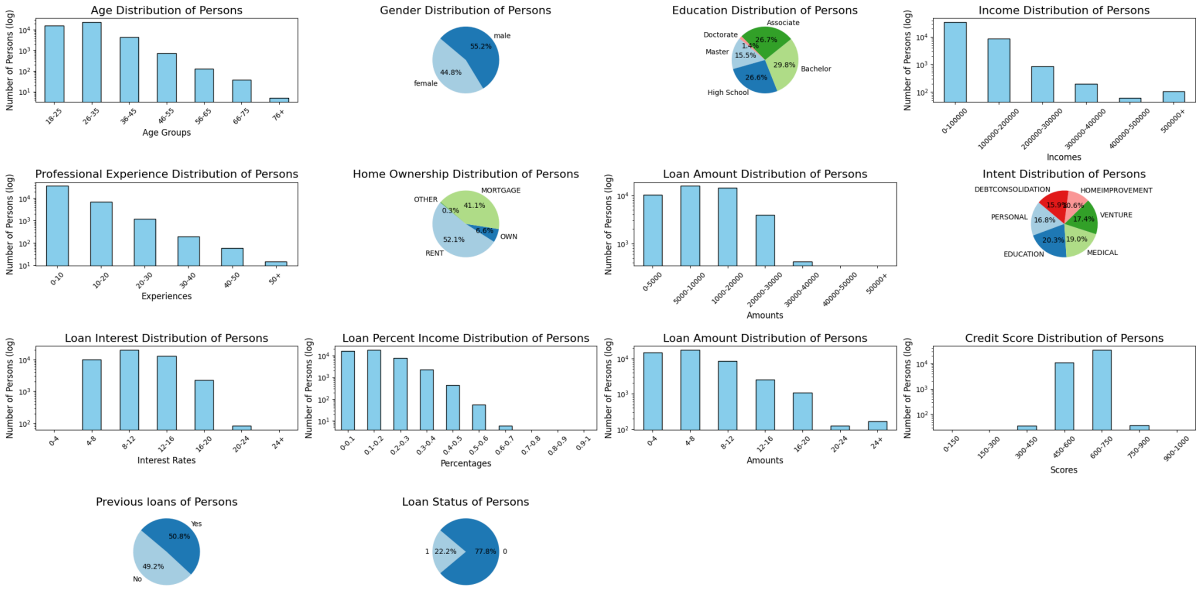
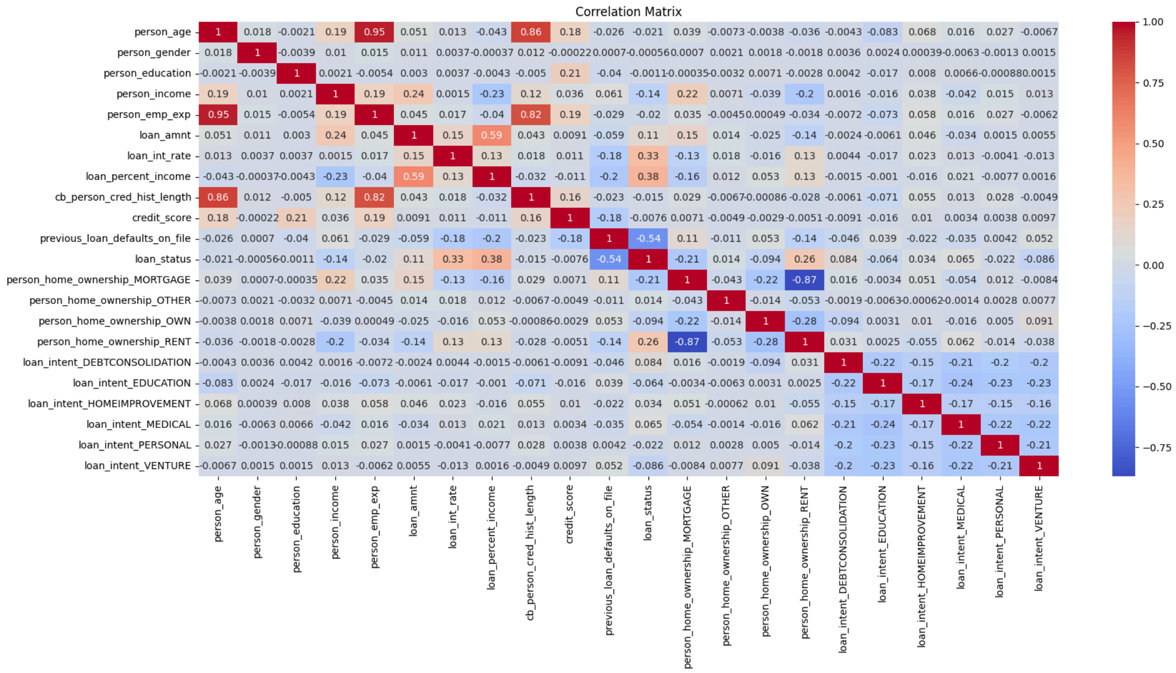


Figure 2 - Bar/Pie Charts of Features

After the data is preprocessed the correlation matrix is created for further understanding of the data.



Strong positively correlated features with loan status:

- **Loan Percent Income (loan_percent_income, correlation = 0.38):** The proportion of income dedicated to repaying a loan carries the most significant weight at 0.38. This shows that banks prioritize assessing how much of a borrower's income will be allocated to their loan payments, as it directly impacts their repayment capacity. Higher ratios suggest a greater financial burden, potentially lowering approval chances.
- **Loan Interest Rate (loan_int_rate, correlation = 0.33):** Loans with higher interest rates have a 0.33 weight, indicating that such loans are more likely to be approved. This reflects the banking sector's profit-driven approach—higher interest rates compensate for higher risk, making these loans more appealing to lenders despite the potential for borrower default.

- **Home Ownership - Renting (person_home_ownership_RENT, correlation = 0.26):** Interestingly, renting a home has a weight of 0.26, implying that renters might have a slight edge in loan approval compared to those who own homes or are paying a mortgage.

Weak Positively Correlated Features with Loan Status:

- **Loan Amount (loan_amnt, correlation = 0.11):** The loan amount exhibits a weak positive correlation with loan approval. This indicates that higher loan amounts slightly increase the likelihood of loan approval, but the effect is minimal.

Strong Negatively Correlated Features with Loan Status:

- **Previous Loan Defaults (previous_loan_defaults_on_file, correlation = -0.54):** This feature has the strongest negative correlation with loan approval. It highlights that having previous loan defaults significantly decreases the chances of loan approval.
- **Mortgage Ownership (person_home_ownership_MORTGAGE, correlation = -0.21):** Individuals paying for a mortgage tend to have a moderately lower likelihood of loan approval, suggesting that mortgage obligations negatively influence the decision.

Weak Negatively Correlated Features with Loan Status:

- **Personal Income (person_income, correlation = -0.14):** Contrary to common assumptions, higher personal income is weakly associated with a lower likelihood of loan approval. This indicates that higher salaries do not necessarily translate into higher approval chances and may, in some cases, slightly reduce them.
- **Home Ownership (person_home_ownership_OWN, correlation = -0.09):** Owning a home does not always favor the applicant in loan approval. This weak negative correlation suggests that home ownership may be associated with a slightly higher chance of rejection.

3. Handling Class Imbalance Problem with Data-level approaches

Class imbalance refers to a situation in a classification task where the distribution of instances across different classes is not uniform, resulting in one class being significantly underrepresented compared to others. The dataset that we use is 35000 to 10000 instances. This imbalance can lead to biased model performance, as the model may become more attuned to predicting the majority class, often at the expense of the minority class. In extreme cases, the model may fail to recognize the minority class, leading to poor generalization and misleading evaluation metrics. Handling class imbalance is crucial for improving the fairness and accuracy of machine learning models. One of the approaches is resampling. There are two methods for this approach:

3.1 Oversampling

Oversampling involves increasing the number of instances in the minority class by duplicating existing examples or generating synthetic data points. The goal is to balance the class distribution by making the minority class more prominent in the training data. This method can lead to overfitting due to repeated data.

- **SMOTE (Synthetic Minority Oversampling Technique)**: Generates synthetic samples for the minority class by interpolating between existing minority samples. It selects a random minority class sample, identifies its nearest neighbors, and creates a new synthetic sample along the line joining the original sample and its neighbor.

  Advantage: Reduces overfitting that can occur with simple oversampling.

  Limitation: May create synthetic samples in regions of feature space where they overlap with the majority class, potentially increasing noise.

- **ADASYN (Adaptive Synthetic Sampling):** ADASYN extends SMOTE by generating more synthetic samples for minority class examples that are harder to classify (i.e., those near the decision boundary or in regions with fewer similar samples).

Advantage: Focuses on hard-to-learn examples, making the decision boundary more robust.

Limitation: Could lead to overfitting if synthetic data creation is excessive.

Our training dataset has 31500 instances with class labels 24500-7000. By SMOTE, we increased 7000 instances to 24500. However, by ADASYN we increased 7000 instances to 23584. We used random sampling to increase 23584 to 24500. Each unique synthetic instance got a new ID starting from 50000.

3.2 Undersampling

Undersampling refers to reducing the number of instances in the majority class by randomly removing some examples. This technique aims to balance the class distribution by decreasing the influence of the majority class in the model's training process. This method may cause a loss of valuable information from the majority class.

- **Cluster-Based Undersampling**

Cluster-based undersampling reduces the size of the majority class by identifying representative samples using clustering algorithms such as k-means. Instead of randomly selecting instances, it retains those that summarize the data distribution. We take an instance that represents each cluster respectively.

Advantages:

- Retains the diversity and structure of the majority class, preserving important data patterns.

- Reduces the risk of losing significant information compared to random undersampling.

Limitations:

- Computationally intensive for large datasets due to the clustering process.
- Sensitive to outliers.

Application in Our Dataset:
Our training dataset has 31,500 instances with class labels distributed as 24,500 (majority) and 7,000 (minority). We applied k-means clustering to the majority class and reduced it to 7,000 instances. The centroids of the clusters were selected to form the reduced dataset, maintaining a 1:1 balance with the minority class.

- **NearMiss**

NearMiss is a family of techniques for undersampling the majority class by selecting samples based on their proximity to the minority class. This ensures that the retained majority class samples are informative for classification.

NearMiss Variants:

1. **NearMiss-1:** Selects majority class instances closest to the minority class samples based on average distance.
2. **NearMiss-2:** Selects majority class instances farthest from other majority class samples, ensuring diversity.
3. **NearMiss-3:** Retains majority class instances closest to a specific number of minority class samples, emphasizing relevance.

Advantages:

- Focuses on informative majority class instances near the decision boundary.
- Enhances the classifier's ability to differentiate between classes.

Limitations:

- May discard majority class samples that represent important patterns unrelated to the minority class.
- Sensitive to noise in the data, which can influence the proximity calculations.

Application in Our Dataset:
We applied NearMiss-1 to the dataset, retaining 7,000 majority class instances. Each retained instance was the closest minority class samples. This approach ensured that the remaining majority samples contributed to better class separability.

4. References

[1] https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data