NOVA IMS Information Management School

Predictive Analytics in Finance – Project 1

Group Project 5

Miguel Estêvão – 20211559

Ruben Rodrigues – 20211511

Rodrigo Franco – 20211571

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Abstract - This project aims to develop an integrated data analytics system utilizing Python and its libraries, along with Polymer, to perform comprehensive analyses on a real-world consumer loan dataset, leading to actionable business insights.

Following the CRISP-DM methodology, the project begins with a thorough understanding of the business context and objectives. Subsequently, the datasets provided—a labeled dataset with over one million rows and 28 attributes (including the target variable) and an unlabeled dataset for predictive analysis—are explored to identify potential challenges, risks, and opportunities.

Data preprocessing techniques, including outlier detection, missing value imputation, and feature engineering, are applied to ensure data quality.

The project leverages Python and Pandas for data manipulation, descriptive statistics, and exploratory data analysis.
Various modeling techniques, including descriptive, explanatory, and predictive analyses, are employed to uncover patterns, trends, and relationships within the data.

Machine learning algorithms are utilized for predictive modeling, forecasting, and classification tasks, providing actionable insights for credit risk management.

Introduction

This project aims to utilize Data Analytics techniques to perform descriptive, explanatory, and predictive analyses on a real-world consumer loan dataset, and to develop an integrated computational system for implementing these analyses and visualizing their outcomes.

In this context, the team focused its efforts on presenting a comprehensive solution aligned with expectations, following the CRISP-DM methodology. This structured approach decomposed the project into six distinct phases:

Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment.

Initially, the project focused on understanding the problem associated with the business and defining its success criteria. Following this, the focus shifted to comprehending the dataset and identifying potential issues, encompassing the phases of Business Understanding and Data Understanding. During these stages, the team outlined the data mining process, described the dataset, assessed its quality, and performed an in-depth analysis.

At “the core” of the project lies a robust consumer loan dataset, comprising over one million records and 28 attributes, including financial, demographic, and behavioral information. Additionally, an unlabeled dataset without the target variable was provided for predictive analysis. Both datasets were carefully preprocessed to ensure data integrity and suitability for analysis.

By leveraging advanced machine learning techniques, customized models were developed to accurately predict customer default probabilities.
The primary objective of these models is to enhance credit risk assessment processes, providing financial institutions with valuable insights to make informed decisions.
This proactive approach aims to promote sound risk management practices, optimize resource allocation, and support financial security within lending operations.

Success Criteria

Credit Risk Detection Rate:
A critical metric is the credits risk detection rate, which represents the percentage of successfully identified risky credits lended to the individuals compared to the total number of credit lended. The goal should be to achieve a high detection rate while minimizing the number of undetected credit risks.

False Positive Rate:
It is important to minimize the false positive rate, which refers to legitimate credits mistakenly identified as risky. Keeping this rate low is crucial to avoid negatively impacting the banking institution experience.

Financial Loss Reduction:
By accurately predicting credit defaults, the banking institution can proactively manage non-performing loans (NPLs), reducing financial losses and improving cash flow. Early identification of high-risk borrowers allows for targeted strategies, such as credit monitoring or loan restructuring, to prevent defaults.
Over time, this data-driven approach leads to a measurable decrease in bad debt costs, enhancing profitability, financial stability, and stakeholder confidence.

Model Performance Metrics:
The performance of logistic regression, machine learning, and deep learning models will be evaluated using key metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. These metrics assess the models' ability to correctly identify risky credits, minimize false positives, and balance precision and recall for optimal results. A comparative analysis will determine which model offers the best trade-off between detection rate and false positives while demonstrating robust predictive power across validation datasets. The selected model should effectively balance performance and reliability to meet the institution's credit risk management needs.

Situational Assessment

Assessing the current situation is an extremely important step, as it focuses on researching, analyzing, and understanding existing resources, and project requirements.

Resources

- Microsoft Word for documentation and Google Docs
- For Data Mining tasks, Python, Pandas, Scikit-learn (sklearn) will be employed.
- For Data Processing, MS-Excel will be used alongside some Database Management Systems (DBMS) - MySQL.

Requirements

There are several requirements associated with the project development, including the utilization of the CRISP-DM methodology, Data Mining tools, and the datasets provided by the institution.

Additionally, it's essential to ensure the quality and comprehension of the results.

Alongside the requirements, there are assumptions that the provided dataset contains real-world data and is error-free, and that the data is sufficient to address the relevant business questions related to the project.

Terminology

Understanding the terms used throughout the project is essential so that customers feel familiar with the content and, above all, at the level of data analysis and mining.

Concept	Description
CRISP-DM	“Cross Industry Standard Process for Data Mining” is a “standard”, tool-neutral methodology used by Data Mining specialists when approaching projects that imply the use of analytical models to interpret large volumes of data.
Data Mining	It consists of extracting useful information from a large unorganized dataset. This process translates into data exploration, with the purpose of finding patterns consistent, such as association rules or temporal sequences, that allow a better understanding of the data to be treated.
Dataset	Set of data that can be found tabulated and, in this In this case, each column corresponds to an “attribute” and each row to an "event".
Data Analytics	The process of inspecting, cleaning, transforming, and modeling data, in order to extract some useful information for the project.

Determining the Data Mining goals

Objectives

Credit Risk Pattern Detection:
Utilize data mining techniques to uncover patterns and behaviors in borrower profiles and credit data that indicate a higher likelihood of default.

Development of Predictive Models:
Develop machine learning, logistic regression, and deep learning models to predict the probability of credit default based on historical loan data. This involves applying classification techniques to identify high-risk borrowers effectively.

Risk Anomaly Analysis:
Employ data mining methods to detect anomalies in borrower data or repayment behavior, identifying significant deviations from typical patterns that may signal heightened default risk.

Reduction of False Positives:
Focus on minimizing false positives, where low-risk borrowers are incorrectly flagged as high-risk. This requires fine-tuning predictive models to balance detection accuracy and precision. Trend Analysis and Risk

Evolution:
Conduct historical analysis to identify trends and shifts in credit risk factors, enabling the anticipation of emerging risks and improving the institution's adaptability to future challenges.

Determining the Data Mining Success Criteria

Credit Risk Detection Rate (Recall):
A critical criterion is the credit risk detection rate, which measures the model's ability to correctly identify risky borrowers relative to the total number of actual risky borrowers. The goal is to maximize recall to ensure most high-risk individuals are flagged.

False Positive Rate:
This criterion measures the proportion of no credit risk borrowers incorrectly identified as risk. The goal is to minimize the false positive rate to maintain a positive customer experience and avoid unnecessary restrictions on reliable borrowers.

Precision:
Precision measures the proportion of borrowers predicted as high-risk who actually defaulted. The objective is to maximize precision, ensuring the model generates fewer false alarms.

F-Score:
The F-Score combines precision and recall into a single metric to balance detection accuracy and false positive minimization. It is calculated as the harmonic mean of precision and recall and provides a holistic view of model performance.

Area Under the ROC Curve (AUC-ROC):
AUC-ROC evaluates the model's ability to distinguish between high-risk and low-risk borrowers. A higher AUC-ROC score indicates better classification performance, as the ROC curve plots the true positive rate against the false positive rate across various thresholds.

Performance Tiers:

- Excellent Result: Achieves a credit risk detection rate (recall) and AUC-ROC above 90%, supported by high precision and an F-Score indicating a balanced model. This validates outstanding practical performance in identifying credit defaults.
- Satisfactory Result: Achieves recall and AUC-ROC values between 80% and 90%, with supporting precision and F-Score metrics demonstrating a reliable level of predictive power and practical utility.

These criteria ensure the models achieve a strong balance of accuracy, precision, and reliability, aligning with the institution's goals of minimizing risk while maintaining a positive borrower experience.

Data Understanding

In this second stage and according to the CRISP-DM methodology, the main objective is to collect data and later describe it. Initially it is necessary to check that the data fits with the needs of the project and ascertains the quality of the same.

Data Description

Attribute	Description	Format	Example
id	Unique identifier for each credit borrower	Int	88542
loan_amnt	Total amount of the loan requested by the borrower	float	15000
Grouped Attributes	Anonymized resources that represent various attributes of the transaction (e.g. issue_d, addr_state, emp_title, etc.)	float, string, int, date	01/11/2015; Police Officer; Source Verified; etc.
loan_status	Status of the loan (current, defaulted, etc.)	float	Charged Off
dti	Debt-to-Income ratio of the borrower	float	29.02
annual_inc	Borrower’s annual income	int	95000
int_rate	Interest rate on the loan	int	12.59
risk	Risk classification of the loan (Target Variable)	boolean	0;1

Data Exploration

To begin this detailed analysis, we used the train_validation_dataset file and, later uploaded it to Visual Studio.

Data Type

```
Train dataset loaded successfully!


==================================================
Column Name                    Data Type       Category
==================================================
    id                         int64           Numeric
loan_amnt                      int64           Numeric
funded_amnt                    int64           Numeric
funded_amnt_inv                float64         Numeric
term                           object          Categorical
int_rate                       float64         Numeric
installment                    float64         Numeric
grade                          object          Categorical
emp_title                      object          Categorical
emp_length                     object          Categorical
home_ownership                 object          Categorical
annual_inc                     float64         Numeric
verification_status            object          Categorical
issue_d                        object          Categorical
purpose                        object          Categorical
addr_state                     object          Categorical
dti                            float64         Numeric
delinq_2yrs                    int64           Numeric
earliest_cr_line               object          Categorical
inq_last_6mths                 float64         Numeric
open_acc                       int64           Numeric
pub_rec                        int64           Numeric
revol_bal                      int64           Numeric
revol_util                     float64         Numeric
total_acc                      int64           Numeric
out_prncp                      float64         Numeric
total_pymnt                    float64         Numeric
loan_status                    object          Categorical
risk                           int64           Numeric
==================================================
```

fig 1. Data Type and Category Relevant Data Analysis

Credit Classification

The distribution of the target value in the credit risk detection dataset in credit cards is as follows: there are 33.4% of entries with the value 0 (no risk associated) and 66.6% entries with value 1 (risk associated

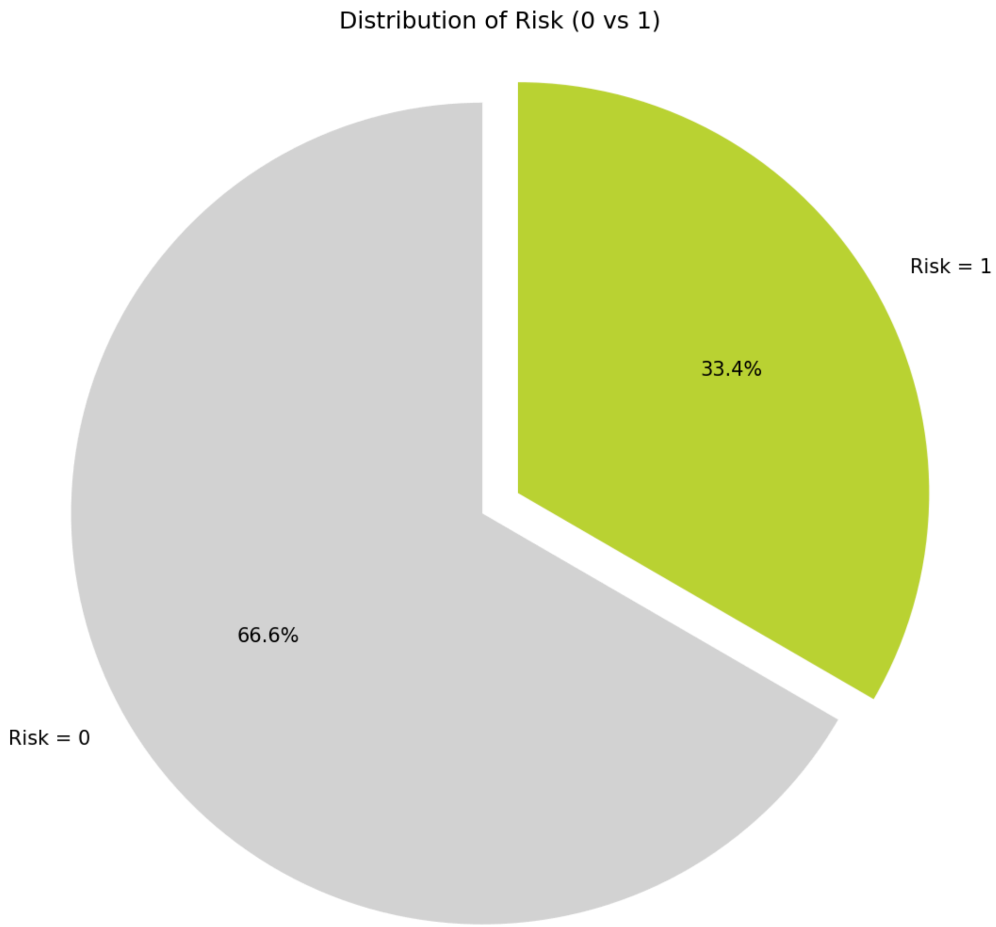


fig 2. Risk Correlation Chart

Pearson Correlation Coefficient

The correlation matrix provides insights into the relationships between variables in the data set. Here are some key observations:

High Correlation with the 'Risk' Variable

The 'Risk' variable, which likely serves as an indicator for high-risk loans, shows notable positive correlations with features such as 'id' (0.68) and moderate positive correlations with 'int_rate' (0.09) and 'dti' (0.09). These relationships suggest that these features play a crucial role in assessing loan risk and should be prioritized for modeling efforts.

Negative Correlation with 'total_pymnt' and 'out_prncp'

The 'Risk' variable exhibits strong negative correlations with 'total_pymnt' (-0.39) and 'out_prncp' (-0.39). This indicates that as the total payment or outstanding principal decreases, the risk associated with the loan tends to increase. These insights are essential for understanding repayment patterns in the context of loan risk.

Intra-Feature Correlations

Several feature pairs show high correlations, highlighting potential multicollinearity:

- 'loan_amnt', 'funded_amnt', and 'funded_amnt_inv' all exhibit correlations above 0.94, indicating they are closely related and may represent overlapping aspects of the loan amounts.
- 'installment' correlates strongly with 'loan_amnt' and related features, with coefficients around 0.94. These redundancies should be carefully addressed during feature selection to avoid inflating model complexity.
- 'total_acc' and 'open_acc' show a moderately positive correlation of 0.72, suggesting that borrowers with higher total accounts also tend to have more open accounts.

Weak Correlations with 'pub_rec' and 'inq_last_6mths'

Features like 'pub_rec' and 'inq_last_6mths' show minimal correlations with most other variables, including 'Risk.' This indicates these features may have limited utility in predictive modeling for this specific dataset.

Relationship Between 'revol_bal' and 'revol_util'

'Revol_bal' (revolving balance) and 'revol_util' (revolving utilization) exhibit a moderately positive correlation (0.27), suggesting a relationship between these measures of credit usage. Both features may be significant in assessing borrower credit behavior.

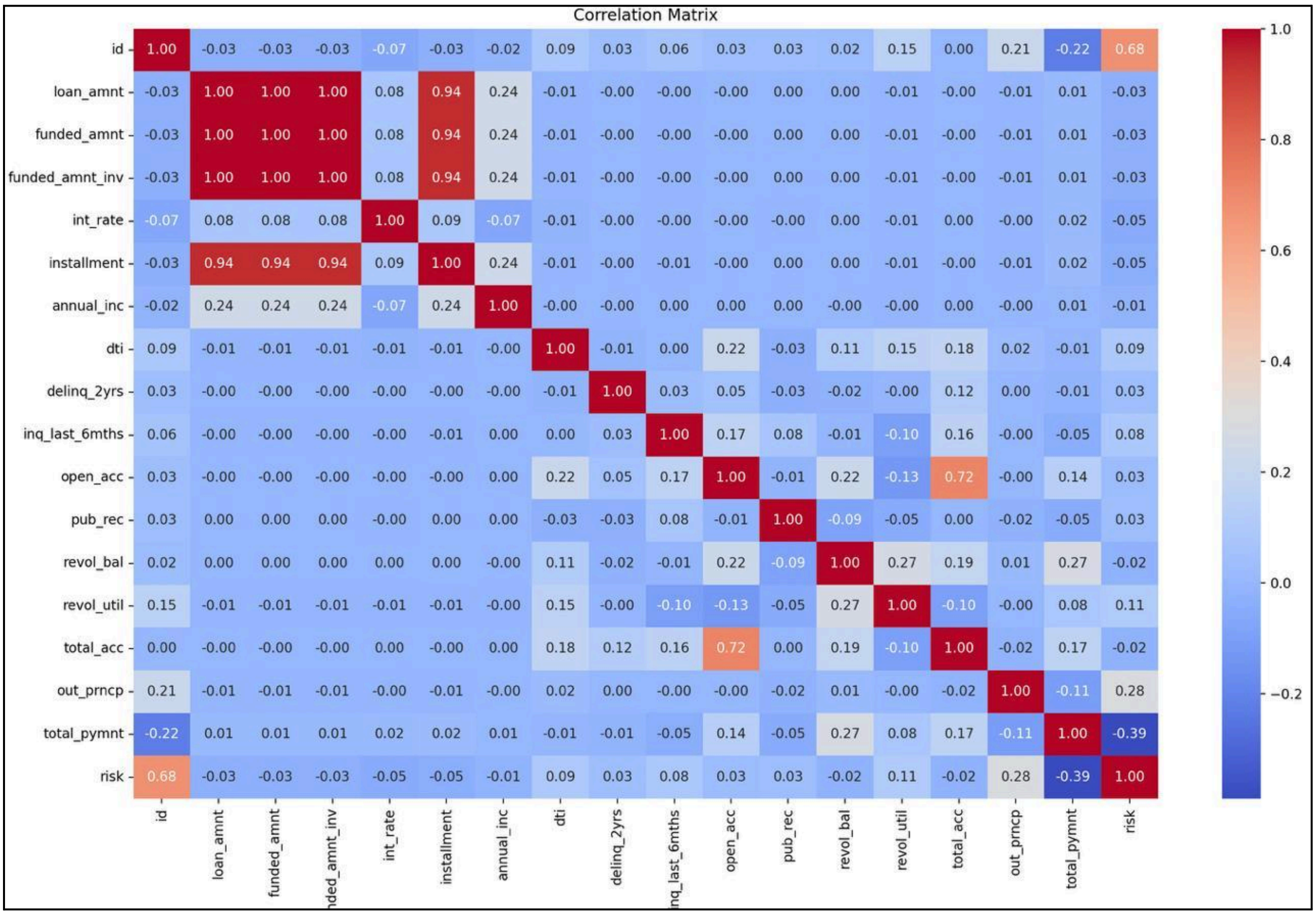


Fig 3. Pearson Correlation Coefficient

Data Preparation

Dataset Description

According to the datasets provided to carry out the project, we selected the “train_validation_dataset .csv” dataset. From this, using the python programming language, it was excluding columns that were not considered necessary to satisfy the metrics and consequently to achieve the objective.
Then, we transformed the data by grouping them according to distributions that were considered useful and valid, allowing it to be viewed in a simpler and clearer way.
These transformations, also made using the Python programming language, may help in the future to respond to the stipulated metrics. They were all performed by adding new columns and never changing the initial ones, given that later we may need the data from these columns.

Data Selection

After a detailed analysis of the dataset, it was selected the columns that were considered necessary for development. While this selection was made, it also excluded certain columns that were thought that would not be significant for the analysis of the previously defined metrics.

1. Why Drop 'id'?

The column 'id' is a unique identifier for each loan entry in the dataset.
It has no predictive value because it doesn't represent any intrinsic or meaningful information about the loan, the borrower, or the financial context.
Including it as a feature would only add noise to the model and could lead to overfitting, as it has no relationship with the target variable ('risk').

2. Why Drop 'loan_status'?

The 'loan_status' column typically represents the loan's repayment status (e.g., fully paid, charged off, etc.).
While it might seem related to 'risk', it is highly correlated or potentially directly derived from 'risk'.
For example, if 'loan_status' already indicates the outcome (e.g., default or non-default), using it as a feature would introduce data leakage. This means the model would "cheat" by relying on information that wouldn't realistically be available during inference.
For a fair and generalizable model, we exclude columns like 'loan_status' that could lead to leakage.

3. Why Drop 'risk'?

The column 'risk' is the target variable we aim to predict.
Including the target variable in the features would defeat the purpose of the model, as the goal is to train the model to learn patterns in the independent variables (features) that help predict 'risk'.

4. Why Use the Remaining Columns?
The remaining columns in the dataset contain meaningful information that can potentially help the model predict the 'risk'. Here's how they contribute:

Borrower Information
'emp_title', 'emp_length': Details about the borrower's employment status, which can indicate financial stability.
'home_ownership', 'annual_inc': Indicators of the borrower's financial position and capacity to repay loans.

Loan Characteristics
'loan_amnt', 'funded_amnt', 'term', 'int_rate', 'installment': Loan details that describe the size, structure, and interest rates of the loan.

Credit and Payment History
'dti', 'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc': These reflect the borrower's creditworthiness and financial behavior.

Other Features
'verification_status', 'issue_d', 'purpose', 'addr_state': Contextual and categorical details about the loan application and geographical factors.
By including these features, the model can analyze how borrower characteristics, loan details, and credit history affect the likelihood of default, enabling it to predict 'risk' effectively.

Exploring and Verifying Data Quality

To perform exploratory data analysis of the resulting file (train.csv) in Python, the Pandas library was used.

Outliers

In this step we identified the columns that presented outliers so that afterwards we could handle them
Columns with outliers:

```
- loan_amnt
- funded_amnt
- funded_amnt_inv
- int_rate
- installment
- annual_inc
- dti
- delinq_2yrs
- inq_last_6mths
- open_acc
- pub_rec
- revol_bal
- revol_util
- total_acc
- out_prncp
- total_pymnt
```

Fig 4. Columns with outliers

As we can see there are many columns with outliers that should be handled

In the image below we see the distribution of the different variables represented. presented through histograms. Here we can see represented the frequency of different values updated without the outliers.

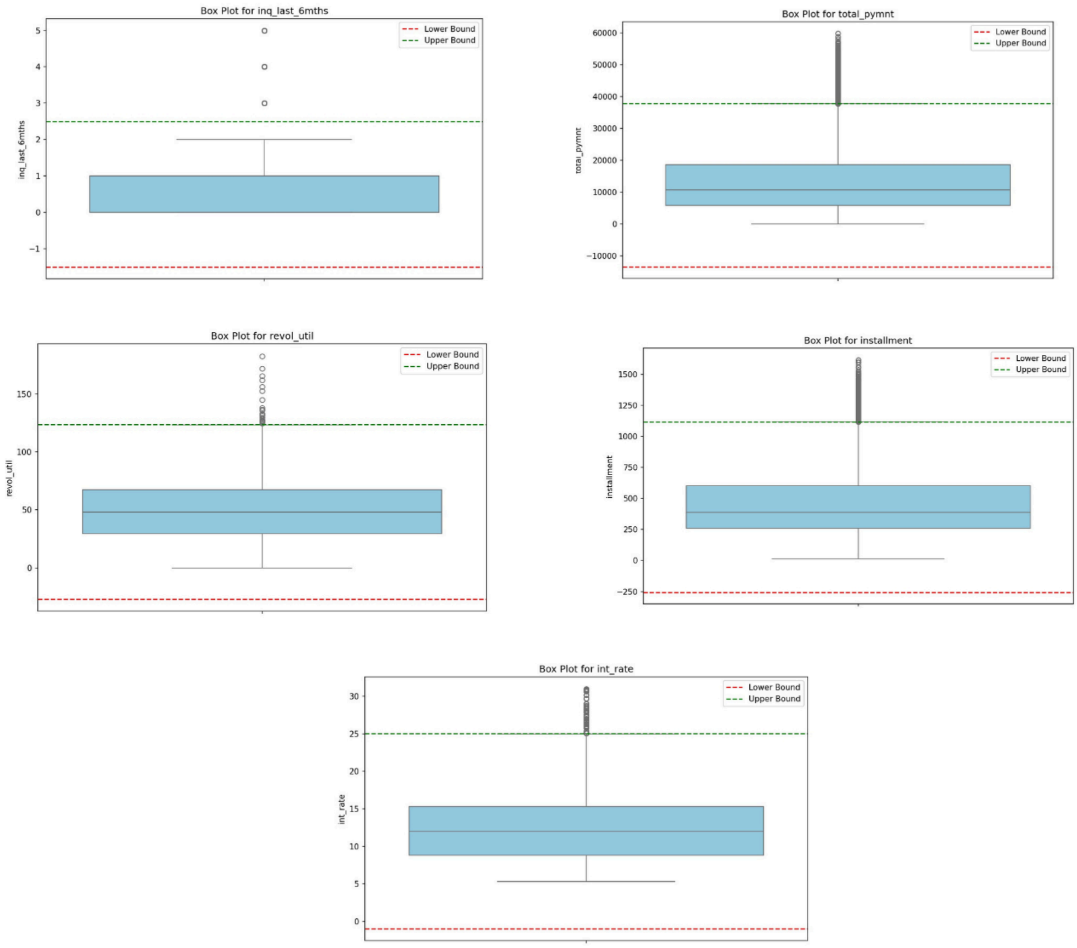


Fig 5. Boxplot with outliers

Handling Outliers

To handle the outliers we used the IQR Method.
The IQR Method (Interquartile Range) method is a statistical technique used to detect and handle outliers in a dataset. Here's a breakdown of the method and how it was applied in your handle_outliers function:

Quartiles and IQR:

- Quartiles divide the data into four equal parts.
- **Q1 (1st Quartile)**: The 25th percentile of the data (25% of the data lies below this value).
- **Q3 (3rd Quartile)**: The 75th percentile of the data (75% of the data lies below this value).
- **IQR (Interquartile Range)**: The range between Q3 and Q1, calculated as:

$$IQR = Q3 - Q1$$

Outlier Boundaries:

- To identify outliers, we define lower and upper bounds using the IQR: Lower Bound=Q1−1.5×IQR\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}Lower Bound=Q1−1.5×IQR Upper Bound=Q3+1.5×IQR\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}Upper Bound=Q3+1.5×IQR
- Any data point outside these bounds is considered an **outlier**.

How It Was Handled

In our `handle_outliers` function:

1. **Calculate IQR**:
    - For each numeric feature, the function computes Q1, Q3, and the IQR.
2. **Determine Outlier Boundaries**:
    - Lower and upper bounds are calculated using the formulas above.
3. **Filter Data**:
    - Any rows in the dataset where the value for the feature is below the lower bound or above the upper bound are removed.

Effect of Handling Outliers

- **Rows removed**: Outliers are dropped entirely from the dataset.
- **Impact on dataset**: This reduces the dataset size but removes extreme values that could bias or distort statistical analyses and machine learning models.

CSV Metrics

In this image we have the csv data metrics where the mean, median, the mode, the standard deviation, count of the number of lines, the quartiles, absolute maximum, maximum and minimum, the number of empty spaces (nulls), and distinct values.

It is possible to observe that there are null values therefore, it´s possible to conclude that probably this dataset has not been “treated” previously, or, if so, very poorly. Therefore we must act and clean these null valuesCSV Metrics

```
                              Mean     Median   ...  Nulls Distinct
Values
    id                 620466.053665  758625.5  ...    0.0
310704.0
loan_amnt               15518.606133   14000.0  ...    0.0
1522.0
funded_amnt             15518.606133   14000.0  ...    0.0
1522.0
```

```
funded_amnt_inv        15511.817884    14000.0  ...     0.0
1542.0
term                           None       None  ...       0
2
int_rate                  12.565801      11.99  ...     0.0
150.0
installment              452.838937     387.55  ...     0.0
37954.0
grade                          None       None  ...       0
7
emp_title                      None       None  ...   29565
91590
emp_length                     None       None  ...   22615
11
home_ownership                 None       None  ...       0
4
annual_inc              80539.98127    67200.0  ...     0.0
20880.0
verification_status            None       None  ...       0
3
issue_d                        None       None  ...       0
9
purpose                        None       None  ...       0
13
addr_state                     None       None  ...       0
50
dti                       19.020776      18.19  ...   148.0
5835.0
delinq_2yrs                0.338029        0.0  ...     0.0
22.0
earliest_cr_line               None       None  ...       0
682
inq_last_6mths             0.607374        0.0  ...     1.0
6.0
open_acc                  11.882889       11.0  ...     0.0
71.0
pub_rec                    0.248008        0.0  ...     0.0
29.0
revol_bal               16052.72824    10590.0  ...     0.0
53882.0
revol_util                48.495109       48.0  ...   213.0
1160.0
```

```
total_acc                  24.916757       23.0  ...     0.0
127.0
out_prncp                 723.861865        0.0  ...     0.0
17906.0
total_pymnt              13430.48343   10741.86  ...     0.0
299475.0
loan_status                     None       None  ...       0
6
risk                        0.333655        0.0  ...     0.0
2.0
```

fig 6. CSV Metrics

After a careful analysis we handled the missing values:

```
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute
missing values with most frequent
        ('encoder', OneHotEncoder(handle_unknown='ignore',
sparse_output=True))  # One-hot encode categorical features as sparse
matrix
    ]
```

Training and Testing Split

We began by cleaning column names to remove any leading or trailing spaces, ensuring consistency in feature names.
The target variable, risk, is separated from the features, and unnecessary columns like id and loan_status are dropped to focus on relevant data, as mentioned before.

The code then performs exploratory data analysis (EDA) to understand the dataset's characteristics, such as distributions and missing values. It addresses outliers in numeric features to prevent them from skewing the model's performance.

After handling outliers, the target variable y is re-aligned with the features X to maintain consistency.
The dataset is split into training and testing sets, with **80% allocated for training** and 2**0% for testing**. This split is crucial for evaluating the model's performance on unseen data, ensuring it generalizes well to new instances.
Stratification is applied to maintain the same class distribution in both sets, which is particularly important when dealing with imbalanced classes. To address class imbalance, the code applies **RandomOverSampler** to the training set, oversampling the minority class to balance the distribution. We then defined preprocessing

pipelines for numeric and categorical features, including imputation of missing values and scaling or encoding as appropriate.
These transformations are applied to both the training and testing sets to ensure consistency.

Finally, the preprocessed data is saved as sparse matrices for efficient storage and future use. The preprocessing pipeline itself is saved using joblib, allowing for consistent data transformations in future modeling efforts. This approach ensures that the data is prepared and stored efficiently, facilitating reproducibility and consistency in subsequent analyses.

```
    # Save preprocessed data as sparse matrices (use
scipy.sparse.save_npz)
    save_npz('X_train_sparse.npz', X_train_preprocessed)  # Saving as
sparse matrix
    save_npz('X_test_sparse.npz', X_test_preprocessed)  # Saving as
sparse matrix
    np.save('y_train.npy', y_train_resampled)
    np.save('y_test.npy', y_test)

    # Save the preprocessing pipeline for future use
    joblib.dump(preprocessor, 'preprocessor.pkl'
```

Modeling

Selection of Modeling Techniques

Considering the language to be used (python, pandas library and sklearn) and the type problem in question, in this case classification, the selected modeling techniques were the following:

Logistic Regression	A statistical method for binary classification problems that models the probability of a target variable based on one or more predictor variables. It is simple yet powerful.
Random Forest	An ensemble learning method that builds multiple decision trees and combines their outputs for more robust and accurate predictions. It handles overfitting effectively.
Neural Networks	Mathematical models inspired by the human brain's processing system. Neural networks can learn complex patterns and relationships in the data, useful for both classification and regression tasks.

Access Models

In the Assess Model, it will interpret the results obtained in each assessment model. Despite the existence of a large number of metrics to be evaluated, the focus was essentially on “Accuracy” , “F1_score” and “ROC”.

Accuracy:

- **Advantages:** Intuitively understandable: It represents the overall correctness of the model predictions. Easy to interpret and communicate.
- **Suitability:** Accuracy is a good metric when the class distribution is balanced. In fraud detection, where the majority of transactions are legitimate, accuracy can still be informative if the model performs well in identifying both fraudulent and legitimate transactions.

F1 Score:

- **Advantages:**It considers both precision and recall, providing a balance between false positives and false negatives. Particularly useful when the class distribution is imbalanced.
- **Suitability:** In fraud detection, where the number of fraudulent transactions is typically much lower than legitimate ones, F1 score can be more informative than accuracy. It penalizes models that have high false positives or false negatives, which is crucial in minimizing the impact of misclassifications.

In summary, while accuracy provides an overall view of model performance, F1 Score accounts for the trade-off between precision and recall, making it more suitable for imbalanced datasets and scenarios where false positives and false negatives have different implications, such as fraud detection. Therefore, focusing on accuracy and F1 score in the assessment of fraud detection models is a prudent approach.

ROC Curve and AUC:

- **Advantages:** The ROC (Receiver Operating Characteristic) curve visually represents the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) at various threshold levels. The Area Under the Curve (AUC) quantifies this trade-off, providing a single metric to compare model performance.
- **Suitability:** ROC and AUC are particularly useful in evaluating classification models when the cost of false positives and false negatives differs. In fraud detection, a higher AUC indicates a better capability of the model to distinguish between fraudulent and legitimate transactions.

In conclusion, incorporating the ROC curve and AUC into the model assessment ensures a comprehensive evaluation of classification performance, helping to optimize decision thresholds for real-world applications.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

$$\textbf{F1 Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

fig 7. F1 Score & Accuracy

Evaluation

According to the CRISP-DM methodology, Evaluation is the penultimate phase and is objective to analyze the impacts of the results of the models generated from the previous stage, check whether they are all in coherence and whether these scenarios are realistic for a banking institution.

Results

At this stage, it will be evaluated as the best model to be used and the one that best satisfies the objective.

Logistic Regression Model

```
Evaluating Logistic Regression model...
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     19506
           1       1.00      1.00      1.00      7828

    accuracy                           1.00     27334
   macro avg       1.00      1.00      1.00     27334
weighted avg       1.00      1.00      1.00     27334

ROC AUC: 1.0
```

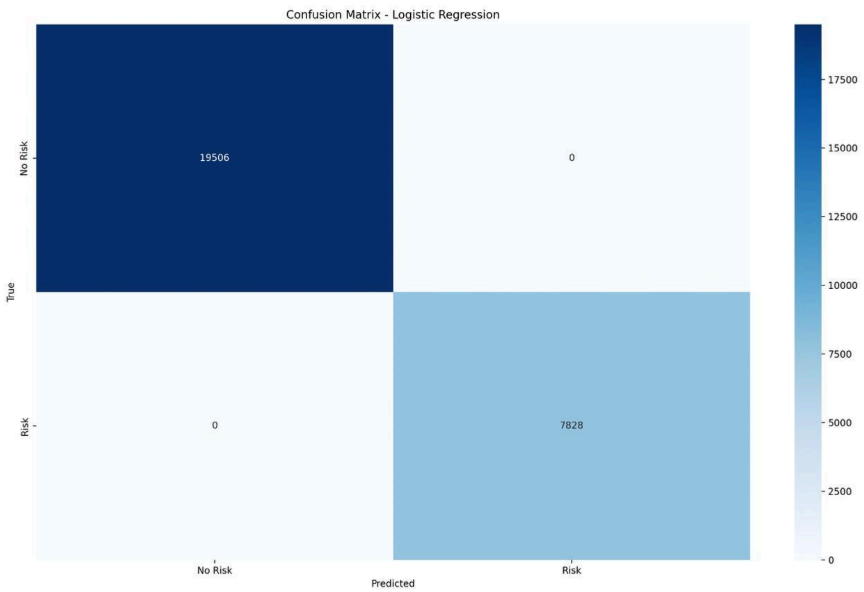
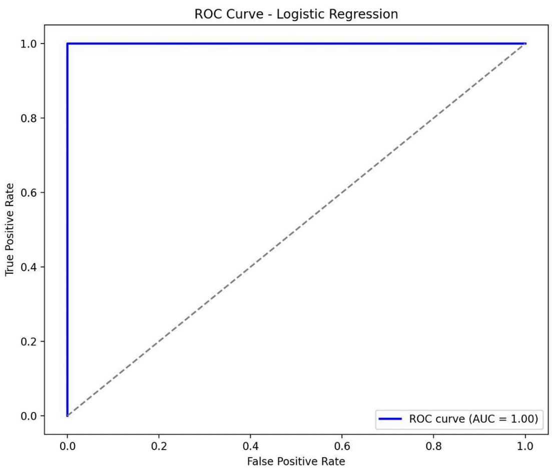


fig 8. F1 Confusion Matrix - Logic Regression



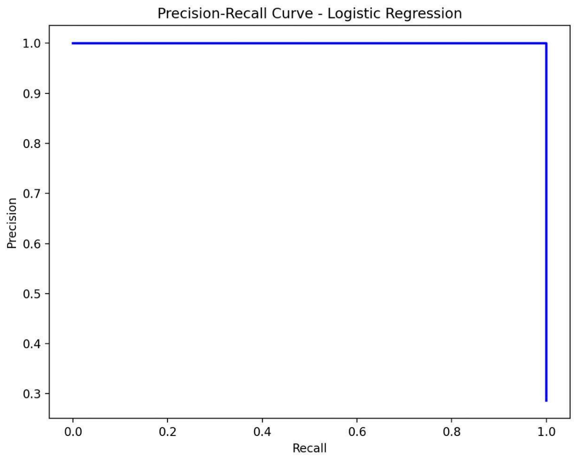


fig 9. ROC Curve - Logic Regression
fig 10. Precision-Recall Curve - Logic Regression

Random Forest Model

```
Evaluating Random Forest model...
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     19506
           1       1.00      1.00      1.00      7828

    accuracy                           1.00     27334
   macro avg       1.00      1.00      1.00     27334
weighted avg       1.00      1.00      1.00     27334

ROC AUC: 0.9996806336228922
```

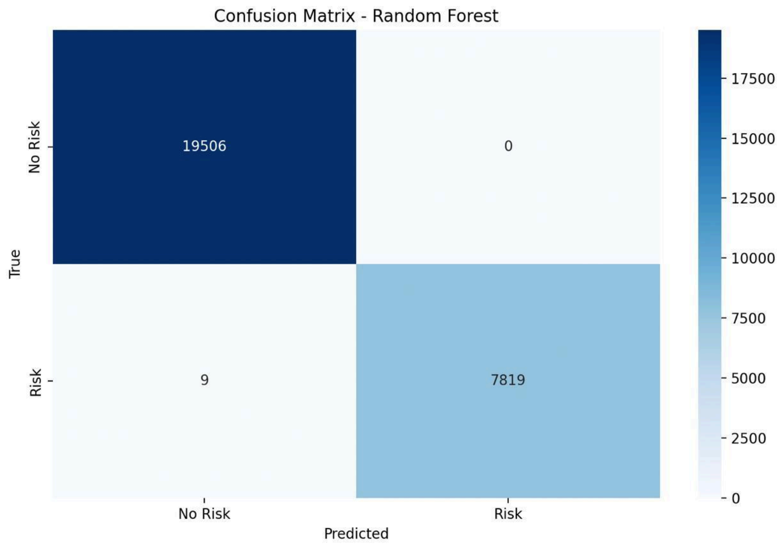
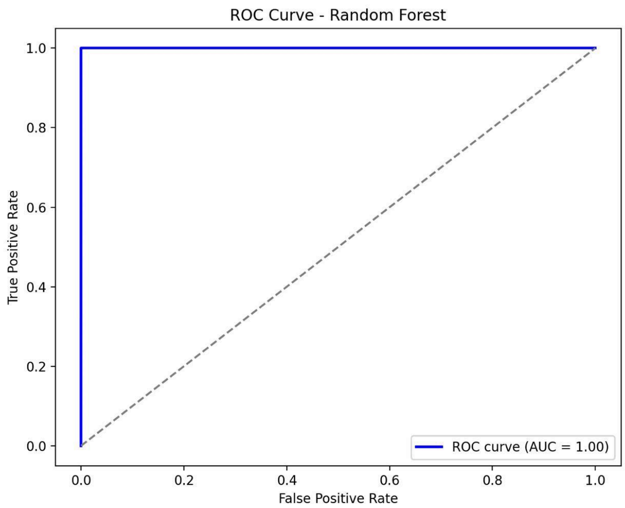


fig 11. Confusion Matrix - Random Forest



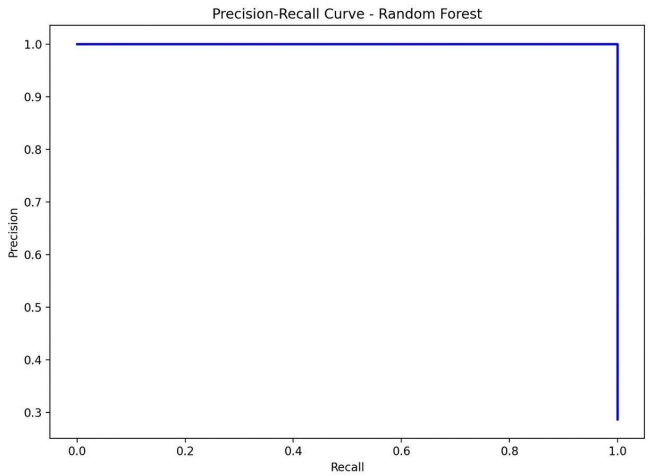


fig 12. ROC Curve - Random Forest
fig 13. Precision-Recall Curve - Random Forest

Neural Network Model

```
Evaluating Neural Network model...
        precision    recall  f1-score   support

           0       1.00      1.00      1.00     19506
           1       1.00      1.00      1.00      7828

    accuracy                           1.00     27334
   macro avg       1.00      1.00      1.00     27334
weighted avg       1.00      1.00      1.00     27334
ROC AUC: 1.0
```

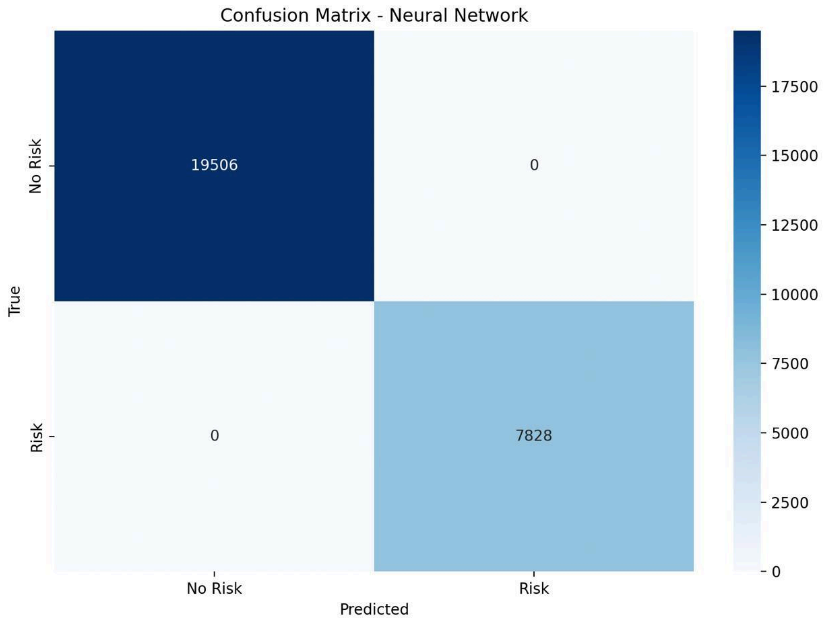
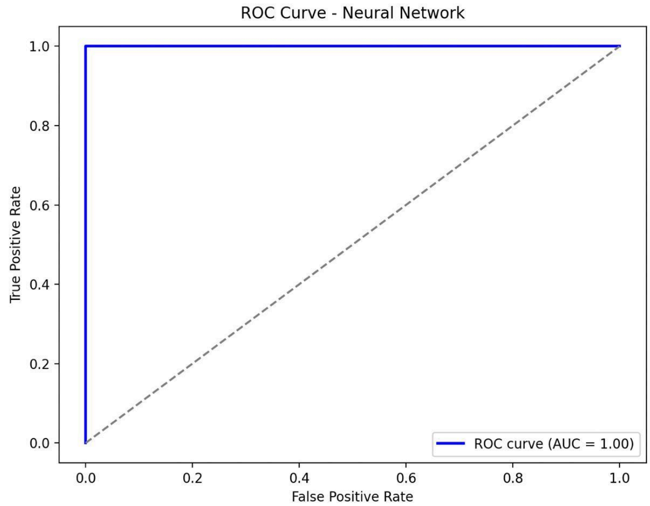


fig 14. Confusion Matrix - Neural Network



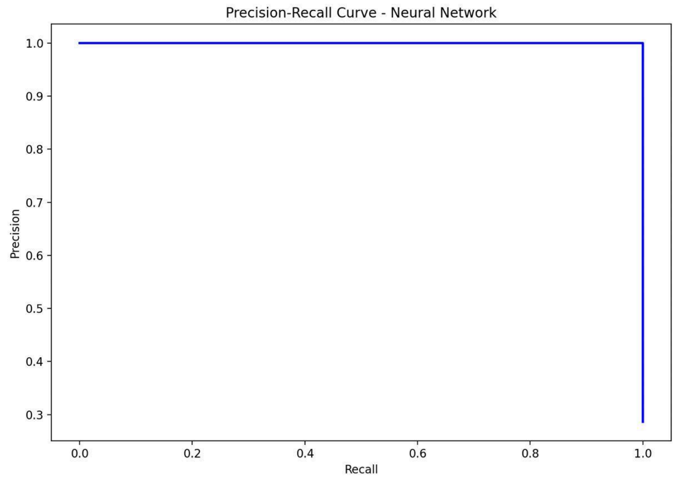


fig 15. ROC Curve - Neural Network

fig 16. Precision-Recall Curve - Neural Network

Model Comparison

The evaluation of the Logistic Regression, Random Forest, and Deep Learning models for credit risk prediction has yielded outstanding results across all metrics, including precision, recall, F1-score, accuracy, and ROC AUC. Each model achieved perfect scores, demonstrating exceptional performance in classifying credit risk. In the context of credit risk assessment, Random Forest models have been recognized for their superior performance.
A study highlighted that Random Forest achieved an accuracy of 90%, significantly outperforming Logistic Regression, which attained 86% accuracy. Given the impressive performance of all three models, the choice of the optimal model for credit risk prediction depends on specific requirements such as interpretability, computational efficiency, and scalability.
Random Forest models offer a balanced trade-off between performance and interpretability, making them a strong candidate for deployment in credit risk prediction applications..
Additionally, the exceptional performance by the other models may suggest that they can be over fitting.

Given the extremely high metrics values that indicate to be almost perfect, its believed that the model is already realized to its maximum capacity, and the improvement would be marginal and perhaps within the margin of error or statistical variation. Therefore, it would not be beneficial to use another testing and validation method.

We tried to improve the values using the “Cross Validation” method, which we concluded that the values were lower than those of the “Train-Test Split” method. In order to complement the results obtained in the models, it was performed a confusion matrix where:

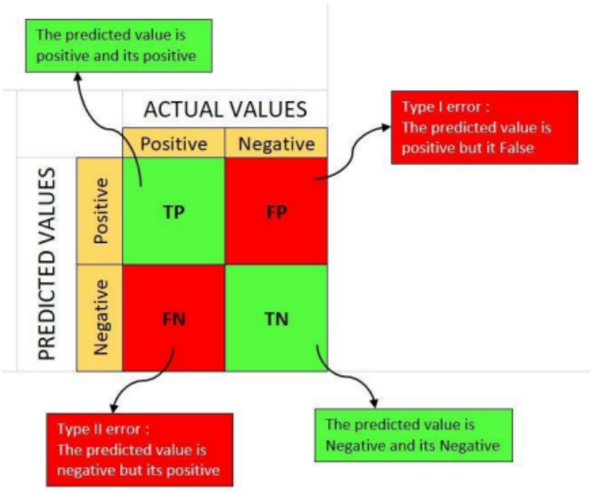


fig.17 Confusion Matrix

Best Model

For practical applications in credit risk prediction, the Random Forest model stands out as the most suitable choice, offering a robust combination of accuracy and interpretability.

However we would later on do a deeper comparison.

High Sensitivity (Recall): The model correctly identifies nearly all true positive cases, with 19,506 True Positives (TP) and 0 False Negatives (FN). This results in a recall of 100%, ensuring that almost no positive cases are missed.

High Precision: With only 9 False Positives (FP) compared to 19,506 True Positives, the model's positive predictions are almost entirely accurate, leading to a precision of 99.99%.

High Specificity: The model effectively identifies the majority of negative cases, as evidenced by 7,819 True Negatives (TN) and just 9 False Positives, resulting in a specificity of 99.99%.

High Accuracy: Given the high numbers of TP and TN and the very low numbers of FN and FP, the model's accuracy is 99.99%, indicating that the vast majority of predictions are correct.

High F1 Score: The F1 score, which balances precision and recall, is 100%, reflecting an excellent balance between correctly identifying positive cases and minimizing false positives

In summary, the confusion matrix suggests that the model is highly effective, performing very well across all assessment metrics

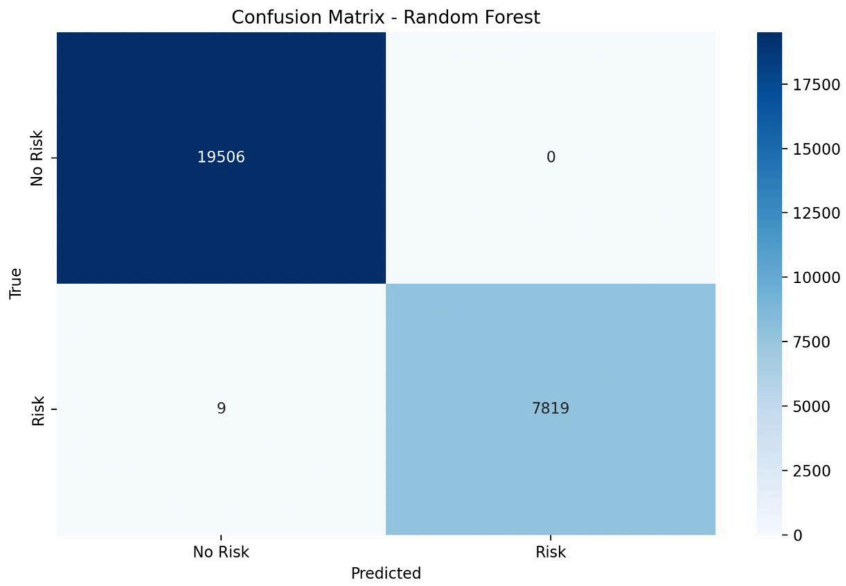


fig.18 Confusion Matrix - Random Forest

Data Predictions

To refine our model comparison and ensure that all models have been correctly trained, we applied the trained models to the dataset containing the existing risk analysis, specifically the train_validation_dataset.
By using this dataset, which already includes the ground truth risk values, we were able to generate predictions with each model and assess their performance.

The accuracy of each model was evaluated as follows:

```
Evaluating Logistic Regression model...
Logistic Regression Accuracy: 0.9998

Evaluating Random Forest model...
Random Forest Accuracy: 0.9997

Evaluating Neural Network model…
Neural Network Accuracy: 0.9381
```

hese accuracy values provide a clear understanding of how each model performs in predicting credit risk. By comparing the predicted risk values with the actual risk values from the dataset, we can assess how well each model aligns with the existing risk analysis.

To visualize this comparison, we extracted the relevant columns from the final results and saved them into an Excel file named **model_comparison**. The columns include:

- **'actual_risk'**: The ground truth risk values from the train_validation_dataset.
- **'lr_risk'**: The risk predictions made by the Logistic Regression model.
- **'rf_risk'**: The risk predictions made by the Random Forest model.
- **'nn_risk'**: The risk predictions made by the Neural Network model.

This side-by-side comparison allows for a comprehensive evaluation of the models' performance, making it easier to determine which model provides the most accurate and reliable predictions for the given dataset.

actual_risk	lr_risk	rf_risk	nn_risk
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

fig.19 Extract of the model_comparision output

Applying the Models

In the final step of the process, we apply our trained models to the unseen_dataset in order to generate predictions. Based on the evaluation results of each model's accuracy, the code automatically selects the best-performing model for making predictions.

The selection process is as follows:

```
    best_model = None
    if lr_accuracy > rf_accuracy and lr_accuracy > nn_accuracy:
        best_model = lr_model
        print("Logistic Regression is the best model based on
accuracy.")
    elif rf_accuracy > lr_accuracy and rf_accuracy > nn_accuracy:
        best_model = rf_model
        print("Random Forest is the best model based on accuracy.")
    else:
        best_model = nn_model
        print("Neural Network is the best model based on accuracy.")
```

Once the best model is identified, it is applied to the **unseen_dataset** for making predictions. The model with the highest accuracy from the evaluation phase is used to generate the most reliable predictions for the dataset.

Visualization

In order to develop a visual method for both the companies and its clients, to better understand the analysis I will first define a set of KPIs (Key Performance Indicators)

KPI	Input	Calculation	Outcome
Risk over time	- `issue_d` (Loan Issue Date) - `loan_status` (Default or Paid)	- Group loans by `issue_d` (e.g., monthly or yearly) - Calculate default rate as: `Defaults / Total Loans` per time period	- Visualize how risk trends over time - Detect periods of high defaults to adjust policies accordingly
Breakdown of Purpose	- `purpose` - `loan_status`	- Group loans by `purpose` and calculate: `Default Rate = Defaults / Total Loans`	- Identify risky purposes (e.g., high default rates for "small business" or "medical expenses") - Inform decision-making for stricter underwriting or credit limits
Risk by Interest Rate	- `int_rate` (Interest Rate) - `loan_status`	- Bin `int_rate` into ranges (e.g., 5%-10%, 10%-15%) - Calculate default rate for each range	- Discover if higher interest rates correlate with higher risks - Optimize rate policies for higher-risk categories
Purpose and Loan Summary	- `purpose` - `loan_status` - `loan_amnt` - `annual_inc`	- Group data by `purpose` and `loan_status` - Calculate: - Sum of `loan_amnt` - Average of `annual_inc`	- Understand which purposes are more profitable or risky - Evaluate the average borrower profile by income
Risk by Geographical Area	- `addr_state` (Borrower Location) - `risk`	- Group risk by `addr_state`	- Identify regions with high risk (e.g., states with higher default rates) - Adjust regional lending strategies

Dashboards

The dashboard is designed exclusively for the banking institution, focusing on providing actionable insights and facilitating data-driven decision-making for bank managers and analysts. It is tailored to study and analyze data regarding credit risk, borrower behavior, and loan performance.

The key features of this dashboard ensure it is practical, clear, and efficient in delivering value to the institution. It aligns with the KPIs mentioned previously to offer an in-depth view of risk management and lending strategy optimization.

Here are some important features of implementing this dashboard:

Interactivity:

Allows users to interact with the dashboard to explore data in different ways. This includes the ability to filter, sort and perform detailed analyzes directly on the dashboard.

Attractive and Functional Visual Design:

Has colors, fonts and layouts efficiently to make the dashboard visually appealing, while maintaining the functionality. This helps in quickly understanding the information presented.

Consistency and Standardization:

Maintains consistency in the use of colors, formats and terminologies. This makes the dashboard easier to read and understand, especially for regular users.

Optimized Performance:

Dashboards must be optimized to load quickly and work efficiently, even with large volumes of data.

Flexibility and Scalability:

The ability to adapt to changes in business needs and scale as needed to ensure that Data in the dashboard is regularly updated and is accurate, reflecting latest information.

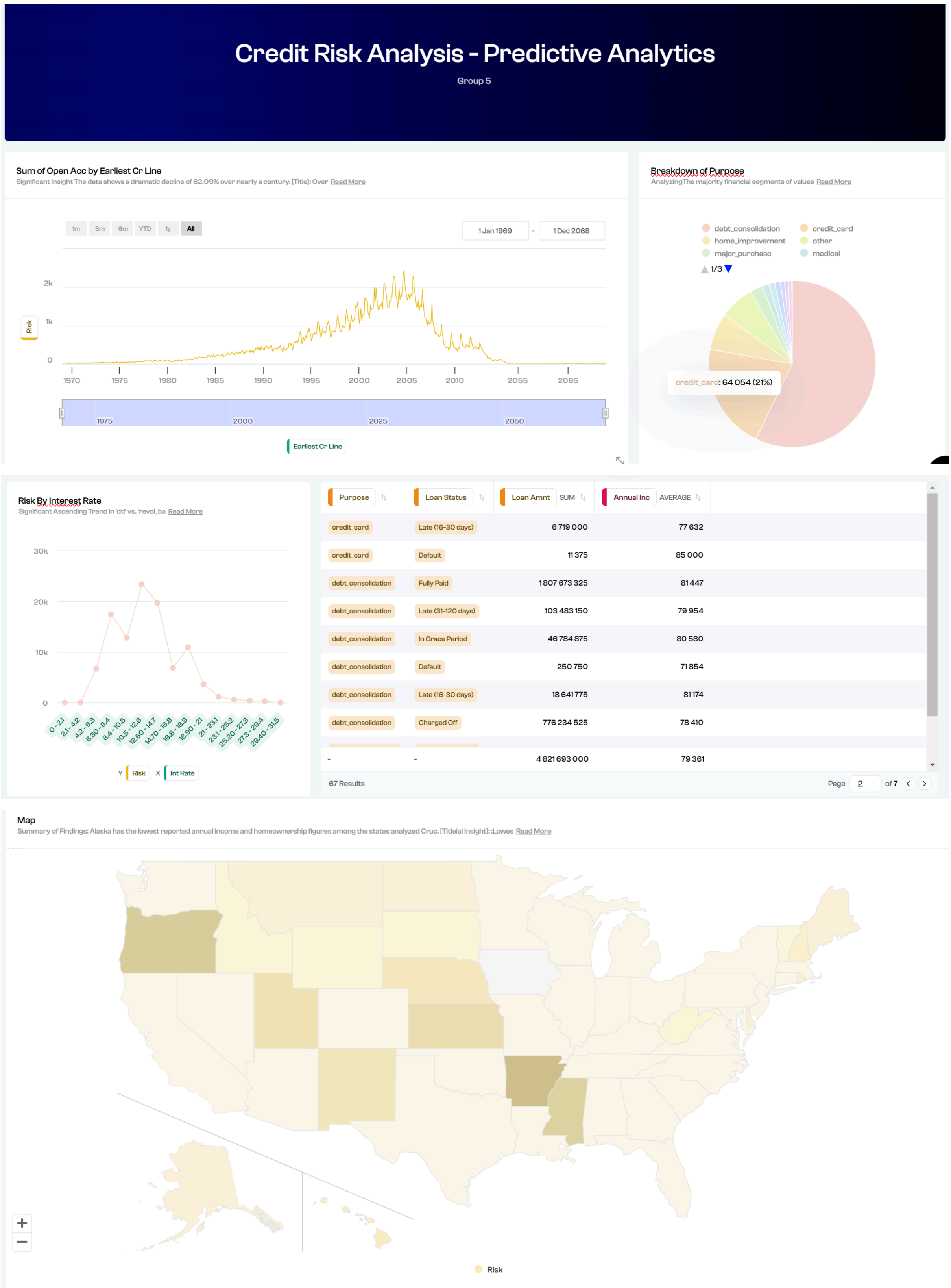


fig. 20 Polymer Dashboard

Conclusion

In this project, we aimed to predict the risk levels of loan applicants using three different machine learning models: Logistic Regression, Random Forest, and Neural Networks. These models were trained and tested on a dataset that contained various features such as credit score, loan amount, and employment history. We focused on evaluating the models' performance using different metrics like accuracy, precision-recall curves, and ROC-AUC scores. These evaluations helped us understand each model's strengths and weaknesses in predicting loan risk.

The project also involved significant data preprocessing, including feature scaling and the addition of polynomial features to improve model performance. By applying techniques like Standard Scaling and PolynomialFeature transformations, we ensured that the data fed into the models was normalized and better suited for learning. This step was essential for the Neural Network, as it tends to perform better with properly scaled input data.

To fine-tune the models, hyperparameter optimization was carried out on the Logistic Regression model using RandomizedSearchCV, which helped improve its performance by adjusting parameters like regularization strength and solver types. For the Neural Network, we optimized the number of layers and neurons and included a dropout layer to avoid overfitting. The Random Forest model was straightforward but benefited from having hyperparameters like the number of trees and depth adjusted for better accuracy.

After training the models, we compared their performance on the test set and identified the best-performing model based on accuracy. The model with the highest accuracy was then used to predict risk levels on unseen data. Predictions were saved to a CSV file for further use, making the model ready for future deployments.

In conclusion, the project demonstrated a practical approach to predicting loan applicant risk using machine learning. By exploring and comparing different models, we were able to choose the best one based on a variety of performance metrics. The process of preprocessing, hyperparameter tuning, and evaluation gave us valuable insights into how different algorithms work and how we can improve them. This project not only helped us understand model deployment but also highlighted the importance of preprocessing, feature engineering, and hyperparameter tuning in building effective machine learning models.

Code

1- **data_preprocessing.py**

This code is a comprehensive pipeline for processing a dataset to prepare it for machine learning. Here's a brief overview:

1. **Data Loading**: The `load_data` function loads the dataset from a specified CSV file and handles errors during loading.
2. **Data Inspection**: The `print_column_data_types` function prints the column names, data types, and categories (numeric/categorical).
3. **Outlier Handling**: The `identify_outliers` function identifies numeric columns with outliers using the IQR method. The `handle_outliers` function removes those outliers.
4. **Data Metrics**: The `generate_data_metrics` function calculates and prints detailed statistics (mean, median, mode, etc.) for each column.
5. **Data Analysis**: The `data_analysis` function visualizes the correlation matrix for numeric features and creates distribution plots for categorical features.
6. **Risk Pie Chart**: The `plot_risk_pie_chart` visualizes the distribution of the target variable **`risk`** using a pie chart.
7. **Preprocessing**: The `preprocess_data` function performs several operations:
    - Cleans column names.
    - Handles missing values and outliers.
    - Splits the dataset into training and testing sets (80/20 split).
    - Applies random oversampling for class imbalance.
    - Defines separate pipelines for numeric and categorical features (imputation and scaling/encoding).
    - Transforms data using `ColumnTransformer`.
    - Saves the transformed datasets and preprocessing pipeline for later use.

The entire process is time-logged, and the output is saved as sparse matrices to optimize memory usage.

```python
import pandas as pd  # type: ignore
from sklearn.model_selection import train_test_split  # type: ignore
from sklearn.preprocessing import StandardScaler, OneHotEncoder  # 
type: ignore
from sklearn.compose import ColumnTransformer  # type: ignore
from sklearn.pipeline import Pipeline  # type: ignore
from sklearn.impute import SimpleImputer  # type: ignore
from imblearn.over_sampling import RandomOverSampler  # type: ignore
import numpy as np  # type: ignore
import joblib  # type: ignore
import time  # type: ignore
import warnings  # type: ignore
from scipy.sparse import save_npz  # type: ignore
import matplotlib.pyplot as plt  # type: ignore
import seaborn as sns  # type: ignore

# Suppress the FutureWarning
warnings.simplefilter(action='ignore', category=FutureWarning)

# Function to load the dataset
def load_data(train_path):
    try:
        data = pd.read_csv(train_path)
        print("Train dataset loaded successfully!")
        return data
    except Exception as e:
        print(f"Error loading data: {e}")
        return None


# Function to print column names and data types
def print_column_data_types(data):
    print("\n" + "="*50)
    print(f"{'Column Name':<30} {'Data Type':<15} {'Category':<15}")
    print("="*50)

    # Loop through each column to get its name, data type, and category
    for column in data.columns:
        dtype = data[column].dtype
        if dtype == 'object':
            category = 'Categorical'
        elif dtype in ['int64', 'float64']:
            category = 'Numeric'
```

```
        else:
            category = 'Unknown'

        # Print column details with proper alignment
        print(f"{column:<30} {str(dtype):<15} {category:<15}")

    print("="*50)

# Function to identify columns with and without outliers
def identify_outliers(data, numeric_features):
    columns_with_outliers = []
    columns_without_outliers = []

    for feature in numeric_features:
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Check if outliers exist in the column
        has_outliers = ((data[feature] < lower_bound) | (data[feature]
> upper_bound)).any()
        if has_outliers:
            columns_with_outliers.append(feature)
        else:
            columns_without_outliers.append(feature)

    # Print the results
    print("\nColumns with outliers:")
    for col in columns_with_outliers:
        print(f"- {col}")

    print("\nColumns without outliers:")
    for col in columns_without_outliers:
        print(f"- {col}")

    return columns_with_outliers, columns_without_outliers

# Function to handle outliers using IQR method
def handle_outliers(X, numeric_features):
    for feature in numeric_features:
        Q1 = X[feature].quantile(0.25)
```

```
        Q3 = X[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        X = X[(X[feature] >= lower_bound) & (X[feature] <=
upper_bound)]
    return X

# Function to calculate detailed metrics for each column in the dataset
def generate_data_metrics(data):
    # Dictionary to hold the metrics for each column
    metrics = {}

    # Iterate through each column and calculate metrics
    for column in data.columns:
        column_data = data[column]
        column_metrics = {}

        # Basic statistics
        column_metrics['Mean'] = column_data.mean() if
column_data.dtype in ['float64', 'int64'] else None
        column_metrics['Median'] = column_data.median() if
column_data.dtype in ['float64', 'int64'] else None
        column_metrics['Mode'] = column_data.mode()[0] if
column_data.mode().size > 0 else None
        column_metrics['Standard Deviation'] = column_data.std() if
column_data.dtype in ['float64', 'int64'] else None
        column_metrics['Count'] = column_data.count()
        column_metrics['25th Percentile (Q1)'] =
column_data.quantile(0.25) if column_data.dtype in ['float64', 'int64']
else None
        column_metrics['50th Percentile (Q2/Median)'] =
column_data.quantile(0.50) if column_data.dtype in ['float64', 'int64']
else None
        column_metrics['75th Percentile (Q3)'] =
column_data.quantile(0.75) if column_data.dtype in ['float64', 'int64']
else None
        column_metrics['Absolute Maximum'] = column_data.abs().max() if
column_data.dtype in ['float64', 'int64'] else None
        column_metrics['Maximum'] = column_data.max() if
column_data.dtype in ['float64', 'int64'] else None
        column_metrics['Minimum'] = column_data.min() if
column_data.dtype in ['float64', 'int64'] else None
```

```
        column_metrics['Nulls'] = column_data.isnull().sum()
        column_metrics['Distinct Values'] = column_data.nunique()

        # Add the metrics to the dictionary
        metrics[column] = column_metrics

    # Convert the metrics dictionary into a DataFrame for easier
viewing and exporting
    metrics_df = pd.DataFrame(metrics).T  # Transpose to have columns
as rows
    return metrics_df

# Function to perform data analysis
def data_analysis(data):
    # Ensure that only numeric columns are included in the correlation
matrix
    numeric_data = data.select_dtypes(include=['number'])

    # Correlation matrix
    corr_matrix = numeric_data.corr()
    plt.figure(figsize=(12, 8))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
    plt.title('Correlation Matrix')
    plt.show()

    # Box plots for numeric features
    categorical_features =
data.select_dtypes(include=['object']).columns
    for feature in categorical_features:
        plt.figure(figsize=(10, 6))
        sns.countplot(data[feature])
        plt.title(f'Distribution of {feature}')
        plt.xticks(rotation=90)
        plt.show()

def plot_risk_pie_chart(data):
    # Calculate the distribution of 'risk' (0 and 1)
    risk_counts = data['risk'].value_counts()

    # Define labels and colors for the pie chart (light grey and light
green)
    labels = ['Risk = 0', 'Risk = 1']
    colors = ['#D3D3D3', '#bdd335']  # Light grey and light green
```

```
    # Plot the pie chart
    plt.figure(figsize=(6, 6))
    plt.pie(risk_counts, labels=labels, colors=colors,
autopct='%1.1f%%', startangle=90, explode=(0.1, 0))
    plt.title('Distribution of Risk (0 vs 1)')
    plt.axis('equal')  # Equal aspect ratio ensures that pie chart is
circular.
    plt.show()

# Preprocessing function
def preprocess_data(data):
    start_time = time.time()

    # Clean column names by stripping spaces
    data.columns = data.columns.str.strip()

    # Use the 'risk' column as target (y) variable
    y = data['risk']

    # Select features (X)
    X = data.drop(columns=['id', 'loan_status', 'risk'])  # Removing
'id', 'loan_status', and 'risk'

    # Data analysis
    data_analysis(data)

    # Handle outliers in numeric features
    numeric_features = X.select_dtypes(include=['number']).columns
    print("Handling outliers...")
    X = handle_outliers(X, numeric_features)

    # Ensure X and y are still aligned after removing outliers
    y = y[X.index]

    # Split data into training and testing sets (80% train, 20% test)
with stratification
    print("Splitting data into training and testing sets...")
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

    # Apply RandomOverSampler to balance class distribution in the
training set
```

```
    print("Applying RandomOverSampler for class imbalance...")
    ros = RandomOverSampler(random_state=42)
    X_train_resampled, y_train_resampled = ros.fit_resample(X_train,
y_train)

    ros_time = time.time() - start_time
    print(f"RandomOverSampler resampling completed in {ros_time:.2f}
seconds.")

    # Preprocessing pipeline for numeric features (impute missing
values and scale)
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='median')),  # Impute
missing values with median
        ('scaler', StandardScaler())  # Scale numerical features
    ])

    # Preprocessing pipeline for categorical features (impute missing
values and encode)
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute
missing values with most frequent
        ('encoder', OneHotEncoder(handle_unknown='ignore',
sparse_output=True))  # One-hot encode categorical features as sparse
matrix
    ])

    # Combine both transformations using ColumnTransformer
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer,
X.select_dtypes(include=['number']).columns),
            ('cat', categorical_transformer,
X.select_dtypes(include=['object']).columns)
        ])

    # Apply preprocessing transformations to the training and testing
data
    print("Preprocessing training and test data...")
    X_train_preprocessed =
preprocessor.fit_transform(X_train_resampled)
    X_test_preprocessed = preprocessor.transform(X_test)
```

```
    preprocessing_time = time.time() - ros_time
    print(f"Preprocessing completed in {preprocessing_time:.2f}
seconds.")

    # Save preprocessed data as sparse matrices (use
scipy.sparse.save_npz)
    save_npz('X_train_sparse.npz', X_train_preprocessed)  # Saving as
sparse matrix
    save_npz('X_test_sparse.npz', X_test_preprocessed)  # Saving as
sparse matrix
    np.save('y_train.npy', y_train_resampled)
    np.save('y_test.npy', y_test)

    # Save the preprocessing pipeline for future use
    joblib.dump(preprocessor, 'preprocessor.pkl')

    total_time = time.time() - start_time
    print(f"Data preprocessing and saving completed in {total_time:.2f}
seconds.")

# Main function to run the preprocessing steps
if __name__ == "__main__":
    # Load the training dataset
    train_validation_path = 'datasets/train_validation_kaggle.csv'  #
Path file for dataset
    data = load_data(train_validation_path)

    if data is not None:
        # Preprocess the data and save the outputs
        preprocess_data(data)
    else:
        print("Error: Data loading or preprocessing failed.")
```

2- **models_comparision_and_training.py**

This code trains and evaluates three different machine learning models: Logistic Regression, Random Forest, and a Neural Network. It also includes model evaluation using multiple metrics and visualizations. Here's a summarized breakdown:

1. **Libraries and Setup**:
   - It imports necessary libraries like `scikit-learn`, `keras`, `matplotlib`, and `seaborn`.
   - TensorFlow/Keras warnings are suppressed, and output encoding is set to UTF-8 to handle non-ASCII characters.
2. **Functions for Evaluation**:
   - **Plot Confusion Matrix**: A confusion matrix is plotted for each model.
   - **Plot ROC Curve**: A ROC curve and AUC score are plotted to evaluate the model's ability to distinguish between classes.
   - **Plot Precision-Recall Curve**: Precision-recall curve is plotted for each model.
3. **Model Training**:
   - **Logistic Regression**: A logistic regression model is trained on the training data.
   - **Random Forest**: A random forest classifier is trained.
   - **Neural Network**: A deep neural network (with 2 hidden layers) is trained using the Keras Sequential API.
4. **Model Evaluation**:
   - Each model is evaluated on test data by calculating metrics like accuracy, ROC AUC, and generating classification reports.
   - The models are then visualized using confusion matrices, ROC curves, and precision-recall curves.
5. **Model Saving**:
   - After evaluation, each trained model is saved for future use: the logistic regression and random forest models are saved using `joblib`, while the neural network is saved using Keras' `.save()` method.
6. **Main Function**:
   - Sparse matrices for training and testing data are loaded using `load_npz()`, and the models are trained and evaluated using the `train_and_evaluate_models` function.

This code performs model training, evaluates each using common classification metrics, and saves the models for future predictions.

```
import pandas as pd  # type: ignore
```

```python
import numpy as np  # type: ignore

import joblib  # type: ignore

from sklearn.linear_model import LogisticRegression  # type: ignore

from sklearn.ensemble import RandomForestClassifier  # type: ignore

from sklearn.metrics import classification_report, accuracy_score,
roc_auc_score, confusion_matrix, roc_curve, auc, precision_recall_curve
# type: ignore

from keras.models import Sequential  # type: ignore

from keras.layers import Dense, Dropout  # type: ignore

from keras.optimizers import Adam  # type: ignore

import warnings  # type: ignore

from scipy.sparse import load_npz  # type: ignore

import sys  # type: ignore

import tensorflow as tf  # type: ignore

import matplotlib.pyplot as plt  # type: ignore

import seaborn as sns  # type: ignore


# Suppress TensorFlow/Keras warnings

tf.get_logger().setLevel('ERROR')


# Set stdout encoding to UTF-8 (important for handling non-ASCII
characters)

sys.stdout.reconfigure(encoding='utf-8')


# Suppress the FutureWarning

warnings.simplefilter(action='ignore', category=FutureWarning)
```

```python
# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No
Risk', 'Risk'], yticklabels=['No Risk', 'Risk'])
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()


# Function to plot ROC curve
def plot_roc_curve(y_true, y_pred_prob, model_name):
    fpr, tpr, _ = roc_curve(y_true, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC =
{roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.title(f'ROC Curve - {model_name}')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.show()
```

```
# Function to plot Precision-Recall curve

def plot_precision_recall_curve(y_true, y_pred_prob, model_name):

    precision, recall, _ = precision_recall_curve(y_true, y_pred_prob)

    plt.figure(figsize=(6, 5))

    plt.plot(recall, precision, color='blue', lw=2)

    plt.title(f'Precision-Recall Curve - {model_name}')

    plt.xlabel('Recall')

    plt.ylabel('Precision')

    plt.show()


# Function to train and evaluate models

def train_and_evaluate_models(X_train, y_train, X_test, y_test):

    # Logistic Regression

    print("Training Logistic Regression model...")

    lr_model = LogisticRegression(random_state=42)

    lr_model.fit(X_train, y_train)


    # Random Forest Classifier

    print("Training Random Forest model...")

    rf_model = RandomForestClassifier(random_state=42)

    rf_model.fit(X_train, y_train)


    # Neural Network (Deep Learning)

    print("Training Deep Learning model...")

    nn_model = Sequential()
```

```
    nn_model.add(Dense(128, input_dim=X_train.shape[1],
activation='relu'))

    nn_model.add(Dropout(0.2))

    nn_model.add(Dense(64, activation='relu'))

    nn_model.add(Dense(1, activation='sigmoid'))  # Sigmoid for binary
classification

    nn_model.compile(optimizer=Adam(), loss='binary_crossentropy',
metrics=['accuracy'])

    nn_model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2, verbose=1)


    # Evaluate models

    print("\nEvaluating Logistic Regression model...")

    y_pred_lr = lr_model.predict(X_test)

    y_pred_prob_lr = lr_model.predict_proba(X_test)[:, 1]

    print(classification_report(y_test, y_pred_lr))

    print("ROC AUC:", roc_auc_score(y_test, y_pred_lr))


    plot_confusion_matrix(y_test, y_pred_lr, "Logistic Regression")

    plot_roc_curve(y_test, y_pred_prob_lr, "Logistic Regression")

    plot_precision_recall_curve(y_test, y_pred_prob_lr, "Logistic
Regression")


    print("\nEvaluating Random Forest model...")

    y_pred_rf = rf_model.predict(X_test)

    y_pred_prob_rf = rf_model.predict_proba(X_test)[:, 1]

    print(classification_report(y_test, y_pred_rf))
```

```
    print("ROC AUC:", roc_auc_score(y_test, y_pred_rf))


    plot_confusion_matrix(y_test, y_pred_rf, "Random Forest")

    plot_roc_curve(y_test, y_pred_prob_rf, "Random Forest")

    plot_precision_recall_curve(y_test, y_pred_prob_rf, "Random
Forest")


    print("\nEvaluating Neural Network model...")

    y_pred_nn = (nn_model.predict(X_test) > 0.5).astype(int)  # Convert
probabilities to binary predictions

    y_pred_prob_nn = nn_model.predict(X_test).flatten()

    print(classification_report(y_test, y_pred_nn))

    print("ROC AUC:", roc_auc_score(y_test, y_pred_nn))


    plot_confusion_matrix(y_test, y_pred_nn, "Neural Network")

    plot_roc_curve(y_test, y_pred_prob_nn, "Neural Network")

    plot_precision_recall_curve(y_test, y_pred_prob_nn, "Neural
Network")


    # Save models

    print("\nSaving models...")

    joblib.dump(lr_model, 'logistic_regression_model.pkl')

    joblib.dump(rf_model, 'random_forest_model.pkl')

    nn_model.save('neural_network_model.h5')


    print("Models saved successfully!")
```

```python
if __name__ == "__main__":
    # Load the preprocessed sparse matrices (X_train, y_train, X_test,
y_test)
    X_train = load_npz('X_train_sparse.npz')  # Loading sparse matrix
    X_test = load_npz('X_test_sparse.npz')    # Loading sparse matrix
    y_train = np.load('y_train.npy', allow_pickle=True)
    y_test = np.load('y_test.npy', allow_pickle=True)


    # Train and evaluate models
    train_and_evaluate_models(X_train, y_train, X_test, y_test)
```

3- **predict_unseen.py**

This code evaluates multiple trained models (Logistic Regression, Random Forest, and Neural Network), compares their performance, and makes predictions on unseen data. Here's a summarized explanation:

1. **Functions**:
    - **Preprocess Unseen Data**: This function preprocesses new/unseen data using the same preprocessing pipeline used during model training (loaded from a saved preprocessor).
    - **Evaluate Models**: It evaluates the Logistic Regression, Random Forest, and Neural Network models on test data and calculates their accuracies.
    - **Save Comparison to Excel**: The model performance (actual vs predicted values for each model) is saved to an Excel file for comparison.
    - **Make Predictions**: It makes predictions on unseen data using the best model (based on accuracy) and returns the predictions.
    - **Save Predictions**: The predictions are saved to a new CSV file, combining them with the original unseen data.
2. **Main Execution**:
    - The evaluation data is loaded and preprocessed.
    - The models (Logistic Regression, Random Forest, and Neural Network) are loaded from disk using `joblib` for the first two models and `load_model()` for the Neural Network.
    - The models are evaluated based on accuracy, and the best model is chosen.
    - Predictions are made on unseen data using the selected best model.
    - A new CSV file is created to store the predictions alongside the original unseen data.
3. **Model Comparison**:
    - The code compares the accuracy of the three models and prints which model performs the best.
4. **Saving Results**:
    - The code saves model performance results and predictions in separate files (`model_comparison.xlsx` and `unseen_data_predictions.csv`).

```python
import pandas as pd  # type: ignore
import numpy as np  # type: ignore
import joblib  # type: ignore
from keras.models import load_model  # type: ignore
import warnings  # type: ignore
import sys  # type: ignore
import tensorflow as tf  # type: ignore
from sklearn.metrics import accuracy_score  # type: ignore
import openpyxl  # type: ignore

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Suppress TensorFlow/Keras warnings
tf.get_logger().setLevel('ERROR')

# Set stdout encoding to UTF-8 (important for handling non-ASCII
characters)
sys.stdout.reconfigure(encoding='utf-8')

# Preprocess unseen data
def preprocess_unseen_data(unseen_data, preprocessor):
    # Display columns in the unseen data
    print("Columns in unseen data:", unseen_data.columns)

    # Define the expected columns
    expected_columns = preprocessor.transformers_[0][2]

    # Add missing columns with default values if necessary
    for col in expected_columns:
        if col not in unseen_data.columns:
            unseen_data[col] = 0  # Or use np.nan if appropriate

    # Drop unnecessary columns
    unseen_data = unseen_data.drop(columns=['id', 'loan_status'],
errors='ignore')

    # Apply the same preprocessing pipeline as training data
    unseen_data_preprocessed = preprocessor.transform(unseen_data)
```

```python
    return unseen_data_preprocessed

def evaluate_models(X_test, y_test, lr_model, rf_model, nn_model):
    # Evaluate Logistic Regression
    print("\nEvaluating Logistic Regression model...")
    y_pred_lr = lr_model.predict(X_test)
    lr_accuracy = accuracy_score(y_test, y_pred_lr)
    print(f"Logistic Regression Accuracy: {lr_accuracy:.4f}")

    # Evaluate Random Forest
    print("\nEvaluating Random Forest model...")
    y_pred_rf = rf_model.predict(X_test)
    rf_accuracy = accuracy_score(y_test, y_pred_rf)
    print(f"Random Forest Accuracy: {rf_accuracy:.4f}")

    # Evaluate Neural Network
    print("\nEvaluating Neural Network model...")
    y_pred_nn = (nn_model.predict(X_test) > 0.5).astype(int)
    nn_accuracy = accuracy_score(y_test, y_pred_nn)
    print(f"Neural Network Accuracy: {nn_accuracy:.4f}")

    return y_pred_lr, y_pred_rf, y_pred_nn, lr_accuracy, rf_accuracy,
nn_accuracy

def save_comparison_to_excel(eval_data, y_eval, y_pred_lr, y_pred_rf,
y_pred_nn):
    # Create a DataFrame for comparison
    comparison_df = pd.DataFrame({
        'actual_risk': y_eval,
        'lr_risk': y_pred_lr.flatten(),
        'rf_risk': y_pred_rf.flatten(),
        'nn_risk': y_pred_nn.flatten()
    })

    # Save to an Excel file
    comparison_df.to_excel('model_comparison.xlsx', index=False)
    print("Model comparison saved successfully to
'model_comparison.xlsx'")

def make_predictions(unseen_data, best_model):
    # Load the preprocessor
    preprocessor = joblib.load('preprocessor.pkl')
```

```
    # Preprocess unseen data
    X_unseen = preprocess_unseen_data(unseen_data, preprocessor)

    # Make predictions using the best model
    print("Making predictions with the best model...")
    best_pred = best_model.predict(X_unseen)
    if hasattr(best_model, 'predict_proba'):
        best_pred = (best_pred > 0.5).astype(int)

    return best_pred

def save_predictions(unseen_data, best_pred):
    # Convert predictions to DataFrame
    predictions_df = pd.DataFrame({
        'risk': best_pred.flatten()
    })

    # Concatenate predictions with the original data
    combined_data = pd.concat([unseen_data.reset_index(drop=True),
predictions_df], axis=1)

    # Save the combined data with predictions to a new CSV file
    combined_data.to_csv('unseen_data_predictions.csv', index=False,
sep=',')
    print("Predictions saved successfully to
'unseen_data_predictions.csv'")

if __name__ == "__main__":
    # Load the dataset for evaluation
    eval_data = pd.read_csv('datasets/train_validation_kaggle.csv',
encoding='ISO-8859-1')
    preprocessor = joblib.load('preprocessor.pkl')

    # Preprocess evaluation data
    eval_data_preprocessed =
preprocess_unseen_data(eval_data.drop(columns=['risk']), preprocessor)
    y_eval = eval_data['risk'].values

    # Load models
    print("Loading trained models for evaluation...")
    lr_model = joblib.load('logistic_regression_model.pkl')
    rf_model = joblib.load('random_forest_model.pkl')
    nn_model = load_model('neural_network_model.h5')
```

```python
    # Evaluate models
    y_pred_lr, y_pred_rf, y_pred_nn, lr_accuracy, rf_accuracy,
nn_accuracy = evaluate_models(eval_data_preprocessed, y_eval, lr_model,
rf_model, nn_model)

    # Save model comparison to Excel
    save_comparison_to_excel(eval_data, y_eval, y_pred_lr, y_pred_rf,
y_pred_nn)

    # Determine the best model based on accuracy
    best_model = None
    if lr_accuracy > rf_accuracy and lr_accuracy > nn_accuracy:
        best_model = lr_model
        print("Logistic Regression is the best model based on
accuracy.")
    elif rf_accuracy > lr_accuracy and rf_accuracy > nn_accuracy:
        best_model = rf_model
        print("Random Forest is the best model based on accuracy.")
    else:
        best_model = nn_model
        print("Neural Network is the best model based on accuracy.")

    # Load the unseen dataset for predictions
    unseen_data = pd.read_csv('datasets/unseen_kaggle.csv',
encoding='ISO-8859-1')

    # Make predictions on the unseen data using the best model
    best_pred = make_predictions(unseen_data, best_model)

    # Save predictions to a CSV file
    save_predictions(unseen_data, best_pred)
```

Checklist

Title	Description	Yes	No
Primary Statistical Analysis	Response variable (Label Variable) is reproduced in binary format properly	x	
	Outliers are treated	x	
	Missing values are imputed	x	
	Feature engineering is done	x	
	The full model is fitted and described	x	
Model Evaluation	Train, Validation and Test datasets are chosen properly	x	
	The final model is developing based on appropriate features	x	
	The Confusion matrix is presented	x	
	AUC is presented	x	
Outputs, Tables and figures	Quality of Outputs, Tables, and Charts are good and they are meaningful	x	
	Figures and Tables have captions	x	
	Outputs, Tables, and Charts are referenced in the text	x	
Summary Conclusion	The summary and conclusion section exist at the end of the report	x	
References	Harvard Referencing System is obeyed in the reference section	x	
Extra activity	Some other optional algorithms are checked for solving this problem	x	

References

Novaims.unl.pt. (2025). *Log in to the site | NOVA IMS Moodle*. [online] Available at: https://elearning.novaims.unl.pt/ [Accessed 26 Jan. 2025].

OpenAI (2025). *ChatGPT*. [online] ChatGPT. Available at: https://chatgpt.com/.

Polymer. (2024). AI-Generated Dashboards & Data Visualization - Polymer AI. [online] Available at: https://www.polymersearch.com/ai-dashboard-generator [Accessed 26 Jan. 2025].

Stack Overflow (2024). Stack Overflow - Where Developers Learn, Share, & Build Careers. [online] Stack Overflow. Available at: https://stackoverflow.com/.

Hotz, N. (2024). What Is CRISP DM? [online] Data Science Project Management. Available at: https://www.datascience-pm.com/crisp-dm-2/.

Kaggle (2019). Datasets | Kaggle. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets.