Machine Learning Project

Phase 1: Data Exploration and Preprocessing (25%)

1. Data Selection

We chose a dataset about credit risk to fit a real-life problem. As we saw at the beginning of the Machine Learning course, AI is commonly used by financial institutions. By using machine learning for credit risk analysis, we want to **predict the likelihood that a borrower will default on their loan or credit obligation**. This helps those institutions and lenders make informed decisions and reduce financial losses.

The features of our dataset include age, annual income, home ownership, employment length (in years), loan intent, loan grade, loan amount, interest rate, loan status (0 is non default 1 is default), percent income, historical default and credit history length.

The binary feature `loan_status` reinforces this classification methodology.

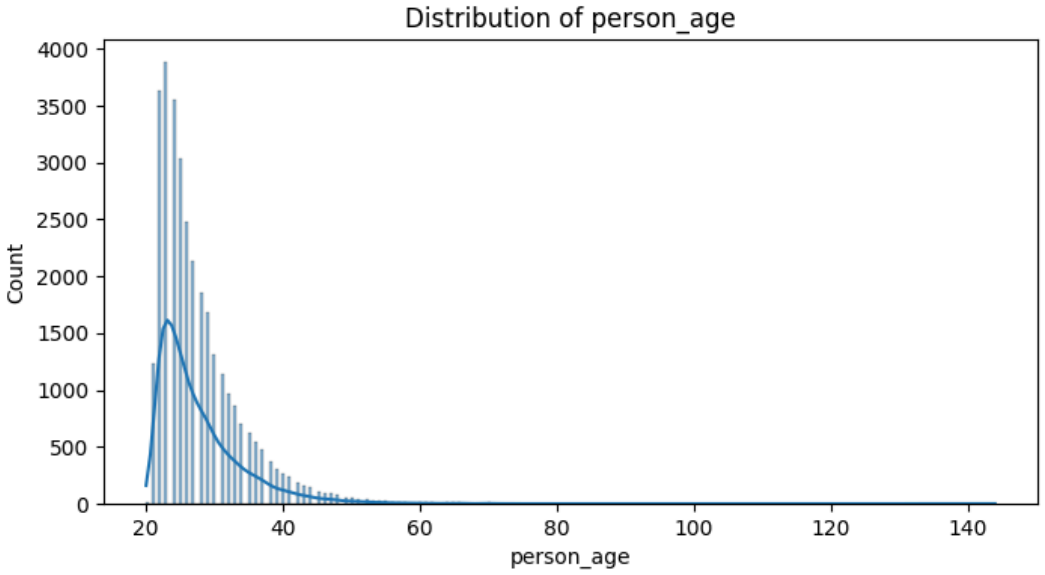
Moreover, our dataset focuses on the likelihood of default based on customer features, so we conclude the datasets can help solve classification problems.

Feature Name	General Data Type	Detailed Data Type
Age	Numerical	Integer
Income	Numerical	Integer
Person home ownership	Categorical	Categorical
Person employment length	Numerical	Float
Loan intent	Categorical	Categorical
Loan grade	Categorical	Categorical
Loan amount	Numerical	Integer
Loan interest rate	Numerical	Float
Loan status	?	Binary (Outcome)
Loan percentage of income	Numerical	Float
History of defaults	Categorical	Binary
Length if credit history	Numerical	Integer

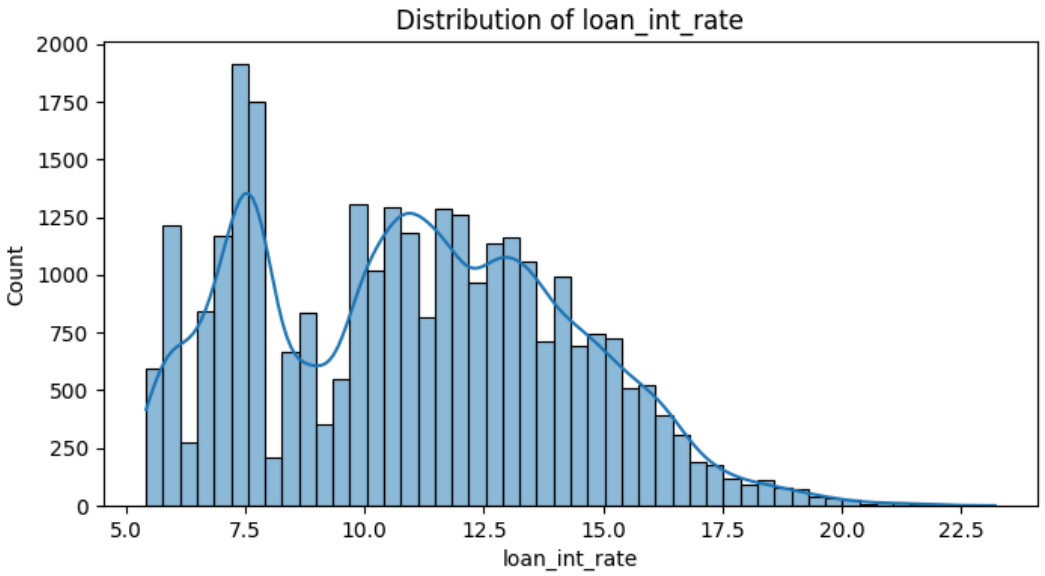
2. Exploration

Thanks to the Python library `Panda`, we can perform exploratory data analysis and obtain information about outliers and missing values.

We have 2 features that miss a non-negligible number of values. Here is a distribution graph of `person_emp_length` and `loan_int_rate`, the 2 lacking features of our dataset:



In the histogram of `person_emp_length`, we have a highly skewed distribution. The data is strongly right-skewed with a lot of values concentrated between 20 and 40 years.



In the histogram of `loan_int_rate`, the peak of the distribution lies between 7% and 10%, suggesting that most loans have interest rates within this range. The data is right-skewed, with the majority of the `loan_int_rate` values concentrated between 5% and 15%.

A smaller portion of loans has higher interest rates (15%–22%), indicating possibly fewer high-risk loans.

There are 895 missing values inside the employment length, which represents around 2.747% of the whole column. There are also 3116 missing values inside the interest rate column, which represents about 10.457% of the whole column. Missing values in key features (e.g., `loan_int_rate` and `person_emp_length`) need to be addressed via imputation or exclusion in the next step of the phase 1. We will manage that problem later.

For outliers, we can see that the `person_age` feature has some suspicious inputs: we can acknowledge 3 persons with 144 years of age, and 2 persons with 123 years of age. These results are outliers that we need to address. Excluding the 94 years old person is also arguable since we can imagine that no banks would borrow money in this situation.

Before imputation, the summary statistics are the following:

Feature	Mean	Median	Variance	Standard Deviation
person_age	27.735	26.00	40.298	6.348
person_income	66074.848	55000.00	3841907194.836	61983.119
person_emp_length	4.790	4.00	17.161	4.143
loan_amnt	9589.371	8000.00	39968781.627	6322.087
loan_int_rate	11.012	10.99	10.501	3.240
loan_status	0.218	0.00	0.170	0.413
loan_percent_income	0.170	0.15	0.011	0.107
cb_person_cred_hist_length	5.804	4.00	16.443	4.055

3. Preprocessing

First and foremost, we decided to rule out the dropping data method since it can cause bias and lead to the loss of valuable information that could improve model accuracy. Moreover, according to established best practices, dropping data is acceptable if the percentage of missing values is very low (e.g., <5%) and spread randomly. However, in our case, we have about 10% of the values of a whole feature that are missing, which exceeds the recommended threshold.

Then, we rejected the mean or median imputation methods. In fact, both mean and median imputation replace missing values with a **single central value** (either the mean or median), which means that the variability or spread of the data is lost. This "centralization" mathematically influences other values such as variance, standard deviation and possibly existing correlations.

At the end, we opted for KNN imputation, a widely recognized method for handling missing values. This imputer utilizes the **k-Nearest Neighbors** method to replace the missing values in the datasets with the mean value from nearest neighbors found in the training set. Using neighboring data points allows us to preserve the local structure and relationships within the dataset.

To do so, we used `KNNImputer` from the `sklearn` library of Python. This method helped us to maintain general statistics of the dataset (e.g., mean, median…) while filling values and preserving relationships between the features. We also deleted the 6 rows including persons of unrealistic ages.

Here are the new outputs on the completed dataset:

Feature	Mean	Median	Variance	Standard Deviation	Missing Values
person_age	27.716	26.00	38.368	6.194	0
person_income	65886.614	55000.00	2759824083.981	52534.030	0
person_emp_length	4.806	4.00	16.416	4.052	0
loan_amnt	9589.378	8000.00	39951793.586	6320.743	0
loan_int_rate	11.085	11.12	9.782	3.128	0
loan_status	0.218	0.00	0.171	0.413	0
loan_percent_income	0.170	0.15	0.011	0.107	0
cb_person_cred_hist_length	5.804	4.00	16.433	4.054	0

The next step involved scaling the data, as required in the assessment. For this, we used the `StandardScaler` function from the `sklearn` library. This method standardizes numerical features by removing the mean and scaling the data to unit variance. Addressing outliers before scaling like we did was important, since `StandardScaler` is sensitive to outliers.

By deleting outliers rows, we removed or mitigated their impact to prevent them from disproportionately influencing the mean and variance, the statistics that influence scaling.

Phase 2: Model Implementation and Evaluation (50%)

1. Problem Type Identification:

The dataset is suited for classification because it contains labeled data, where each observation is associated with a predefined class (category, features…) . These labels allow for supervised learning, enabling a model to learn patterns and predict the correct class for new, future data.

2. Model Selection and Implementation:

We will implement three different machine learning algorithms:

-Decision Tree

-Random Forest

-KNN

2.1- General Code Part

link to google collab: Projet_ML.ipynb

As part of the implementation of machine learning algorithms, we used several key libraries:

Pandas enables efficient data manipulation and analysis, particularly for managing Excel and performing transformations.

NumPy, with its tools for mathematical calculations and table manipulation, is indispensable for optimizing performance.

Matplotlib offers a clear way of representing data and modeling decision boundaries graphically for visualizing results.

Finally, Scikit-learn is used for its comprehensive functionality, from data pre-processing to the application of algorithms such as KNN or Decision Tree. All that stuff gives to us a large panel of applications for data preparation to interpretation of results.

Let's start with what we have in common:

Firstly, we initialized our dataset using the Pandas library by reading the Excel file. Then, we separated the data into two distinct variables: X for the features and y for the labels.

At this step, we preprocess the data by converting all features into numerical values (using scikit-learn to create binary columns), performing imputation to handle missing or irrelevant features, and then concatenating the processed parts.

example of the variable encoder:

cb_person_default_on_file		cb_person_default_on_file_N	cb_person_default_on_file_Y
N	->	1	0
Y		0	1
N		1	0

Imputation…

Next, we standardized the data so it’s ready to be used.

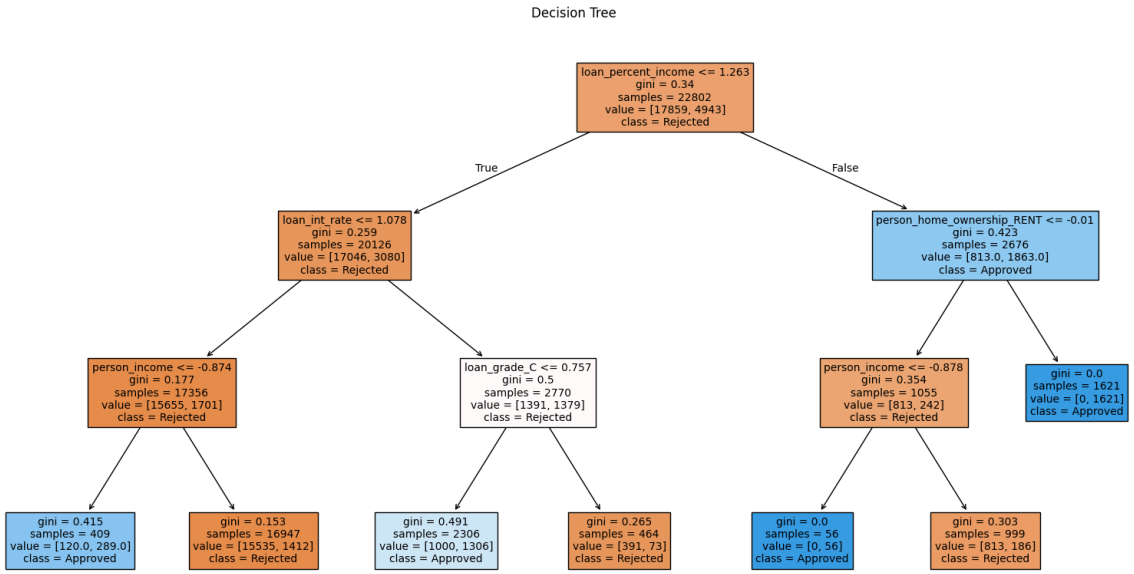
We decided to split our data into two datasets: one for training the machine learning models, which is very important for allowing the model to learn patterns and relationships within the data, and the other for testing, which will be used to evaluate the model's performance.

2.2- Decision Tree

A Decision Tree is a supervised machine learning algorithm (used for both classification and regression tasks). It works by recursively splitting the data into subsets based on feature values, resulting in a tree-like structure of decisions and their possible consequences. Each internal node represents a decision based on a feature, while each leaf node corresponds to a predicted class or value. The aim is to create a model that accurately predicts outcomes by learning decision rules from the training data.

In this implementation, we use a DecisionTreeClassifier with a maximum depth of 3 to avoid overfitting. The model is trained on the features X_train and the corresponding labels y_train. Once trained, predictions are made on the test set X_test using the predict() method. To evaluate the model's performance, we calculate the confusion matrix, which compares the predicted values (y_pred) with the true values (y_test). This allows us to assess the accuracy of the model's predictions.

Finally, a graphical representation of the decision tree is created using the plot_tree() function from scikit learn. In the plot, each leaf is color-coded according to the predicted class (either "Approved" or "Rejected"), providing a clear visualization of the decisions made by the model at each node. This tree structure helps in understanding how the model classifies data based on different feature thresholds.



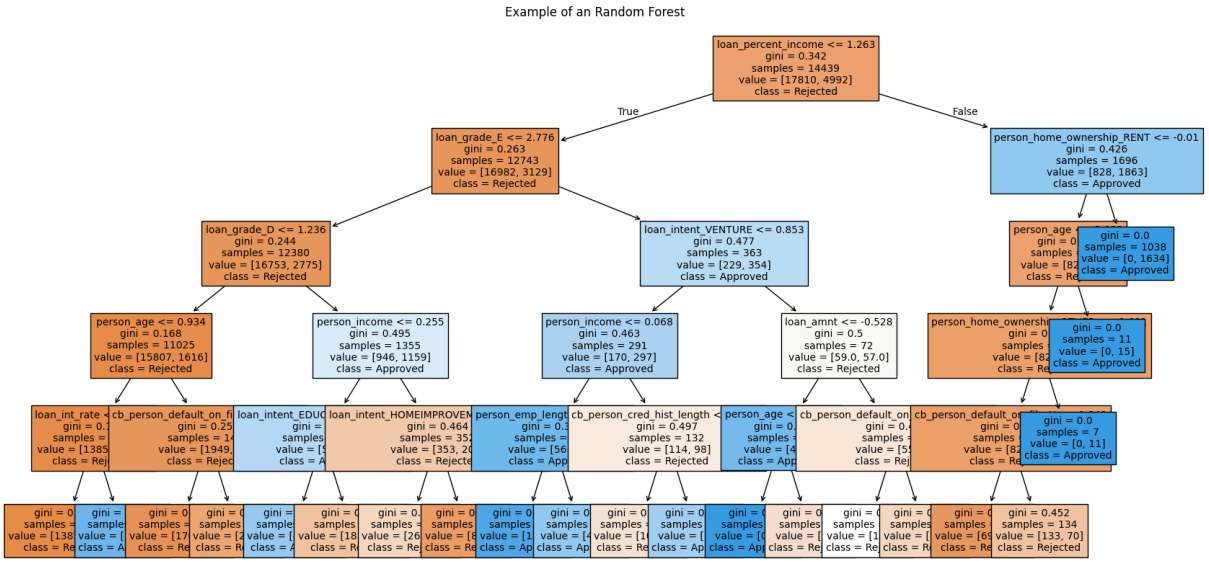
2.3- Random Forest

Random Forest is an ensemble learning algorithm that builds multiple decision trees during training and merges them to improve the accuracy and robustness of predictions. Each tree in the forest is trained on a random subset of the data and features, and during prediction, the class is decided by the majority vote of all trees.

In this code, after preprocessing and standardizing the data, we split it into training and testing sets using train_test_split. We then train a RandomForestClassifier with 100 trees (n_estimators=100) and a maximum tree depth of 5 (max_depth=5) to prevent overfitting.

Once the model is trained, we make predictions on the test set using the predict() method, and these predictions are compared with the true labels (y_test) to evaluate the performance of the model.

To visualize how a Random Forest model works, we display a single decision tree from the forest using the plot_tree() function. This function plots the first decision tree in the forest (rf.estimators_[0]), showing the decision rules at each node and the predicted class (either "Rejected" or "Approved") at the leaves. The tree is filled with colors representing the predicted classes.

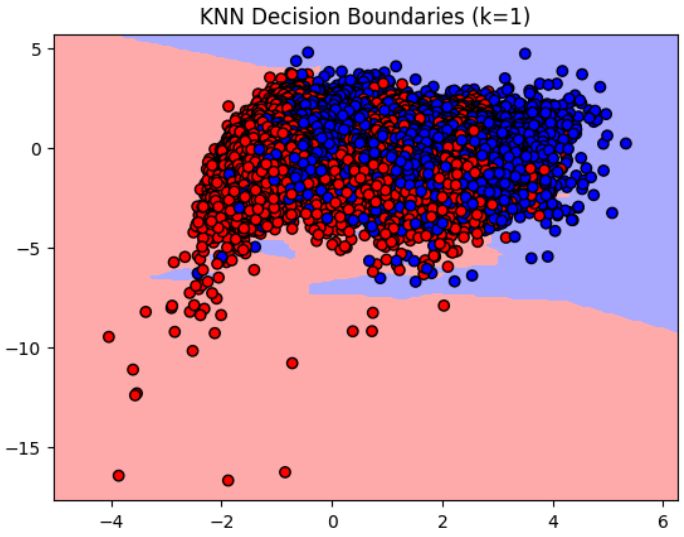


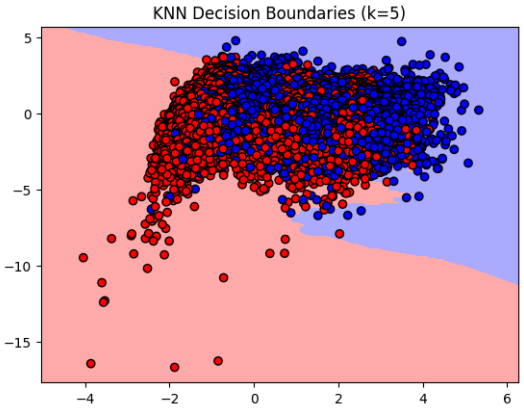
2.4- KNN

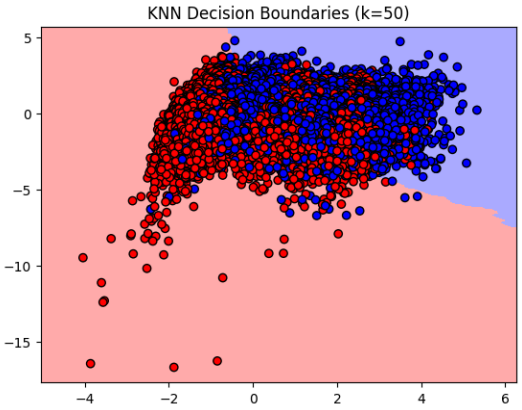
K-Nearest Neighbors (KNN) works by assigning a class to a data point based on the majority class of its k nearest neighbors in the feature space. In this part of the code, we first apply Principal Component Analysis (PCA) to reduce the dimensionality of the dataset to two components, making it easier to visualize the decision boundaries in a 2D space (Calculate covariance matrix -> Calculation of eigenvectors (principal components) and eigenvalues -> The data is transformed into a new space defined by the principal components.).

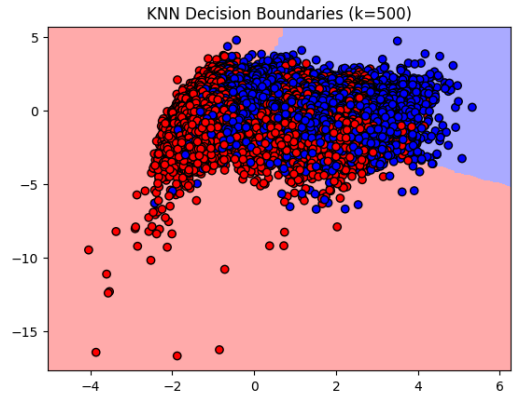
We train the KNeighborsClassifier with k=5, meaning the model will use the 5 nearest neighbors to classify each data point. Once trained, the model is used to predict the class labels for the test set.

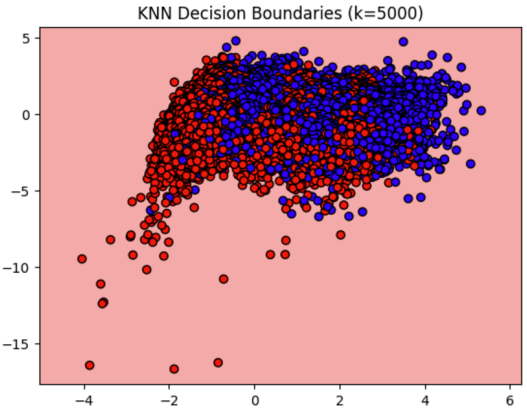
To visualize the model's decision boundaries, we create a meshgrid over the 2D PCA space. The classifier predicts the class for every point in the grid, and these predictions are reshaped to match the meshgrid's shape. We then use contourf to plot the decision regions, coloring them according to the predicted class. The data points themselves are overlaid on the contour plot with different colors representing the actual class labels (loan_status).











2.5- Evaluate models

In this part of the code, we evaluate the performance of the model using several classification metrics: precision, recall and F1 score, as well as the ROC curve and confusion matrix.

The classification report provides detailed information on the model's performance, indicating the precision, recall and F1 score for each class (Rejected and Approved). These metrics are particularly useful for assessing the balance between false positives and false negatives, especially in the case of unbalanced classes.

The formulas for the main metrics are as follows:

$$Precision = \frac{TP}{TP+FP}$$

where TP is the number of true positives and FP is the number of false positives. Precision measures the proportion of positive predictions that are actually correct.

$$Recall = \frac{TP}{TP+FN}$$

where FN is the number of false negatives. Recall measures the model's ability to identify all positive instances.

$$F1\,Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The F1 score is the harmonic mean of precision and recall, and it is used to combine these two measures into a single indicator, especially when the classes are imbalanced.

The ROC curve shows the model's ability to discriminate between classes, by plotting the relationship between the rate of false positives and the rate of true positives. The AUC (Area Under the Curve) quantifies this performance: an AUC of 1 indicates perfect classification, while an AUC close to 0.5 suggests that the model is random…

Finally, the confusion matrix shows the errors in the model by comparing the predictions with the true values. It displays the number of true positives, false positives, true negatives and false negatives, providing a detailed view of where the model is wrong.

These tools are applied in a similar way to the three models (Decision Tree, Random Forest and KNN), providing essential measures for adjusting and improving their performance.

3. Model Evaluation:

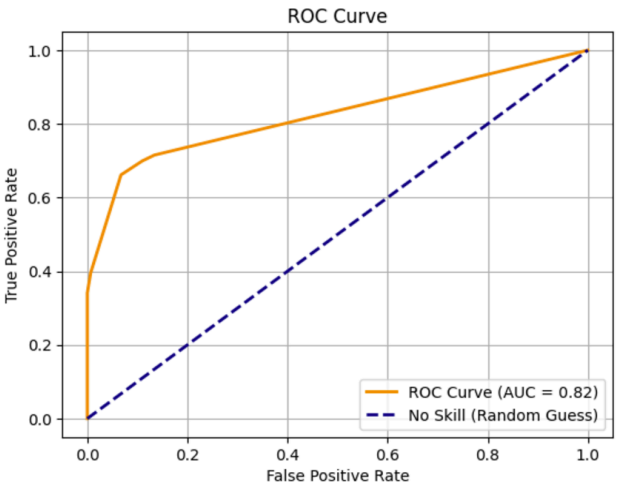
3.1- Decision Tree

The Decision Tree model obtained an overall precision of 88%. For the Rejected class, the results are very good with a precision of 91%, a recall of 94%, and an F1-score of 92%. This indicates that the model effectively identifies rejected loans.

However, for the Approved class, performance is much lower. Precision is 75%, recall 67%, and F1-score 71%. This indicates that the model has difficulty correctly identifying approved loans, which could lead to a relatively high number of false negatives.

Concerning the ROC curve, the AUC is 0.83. This indicates a good ability to discriminate between the two classes, although the model may still confuse some instances between Rejected and Approved.

	precision	recall	f1-score	support
Rejected	0.90	0.93	0.92	7540
Approved	0.74	0.66	0.70	2233
accuracy			0.87	9773
macro avg	0.82	0.80	0.81	9773
weighted avg	0.87	0.87	0.87	9773



3.2- Random Forest

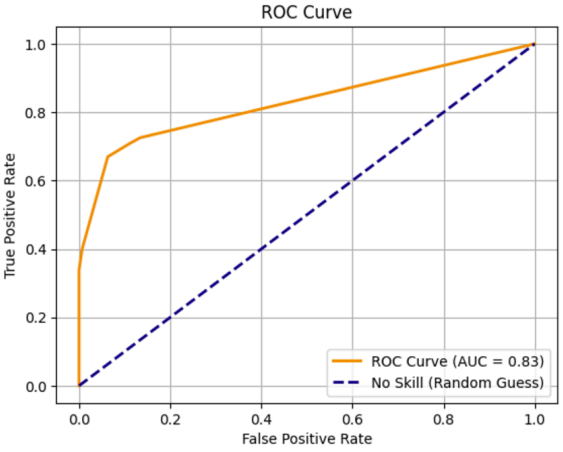
The Random Forest model is the best performing of the three. Its overall precision is anexcellent 90%. For the Rejected class, performance is very good with 89% accuracy, 99%

recall, and an F1-score of 94%. This means that the model recognises rejected loans very well, with very few false positives for this category.

For the Approved class, the results are better than for the other two models, with a precision of 96%, a recall of 55%, and an F1-score of 70%. Although recall is still average, it remains higher than that obtained with the Decision Tree or KNN.

The AUC of the ROC curve is 0.83, indicating a very good ability to discriminate between classes, similar to that of the Decision Tree. However, the Random Forest model shows better robustness and generalisation.

	precision	recall	f1-score	support
Rejected	0.88	0.99	0.94	7563
Approved	0.96	0.55	0.70	2210
accuracy			0.89	9773
macro avg	0.92	0.77	0.82	9773
weighted avg	0.90	0.89	0.88	9773



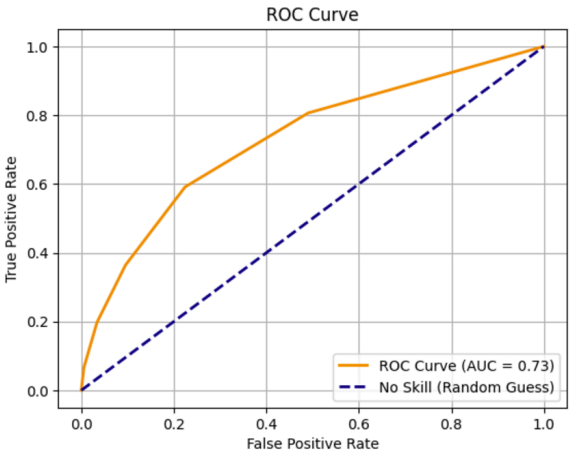
3.3- KNN

The KNN model performs much less well than the others. Overall precision is a relatively low 67%. For the Rejected class, performance remains acceptable, with precision of 78%, recall of 80%, and an F1-score of 79%. However, for the Approved class, the results are very poor with a precision of only 21%, a recall of 19%, and an F1-score of 20%.

These low scores for the Approved class indicate that the model has difficulty recognising this category, which could lead to a large number of false positives.

The ROC curve shows an AUC of 0.72, reflecting poor performance in terms of separation between the two classes. This shows that KNN is not well suited to this particular problem.

	precision	recall	f1-score	support
Rejected	0.79	0.87	0.83	7718
Approved	0.20	0.12	0.15	2055
accuracy			0.71	9773
macro avg	0.49	0.50	0.49	9773
weighted avg	0.66	0.71	0.68	9773



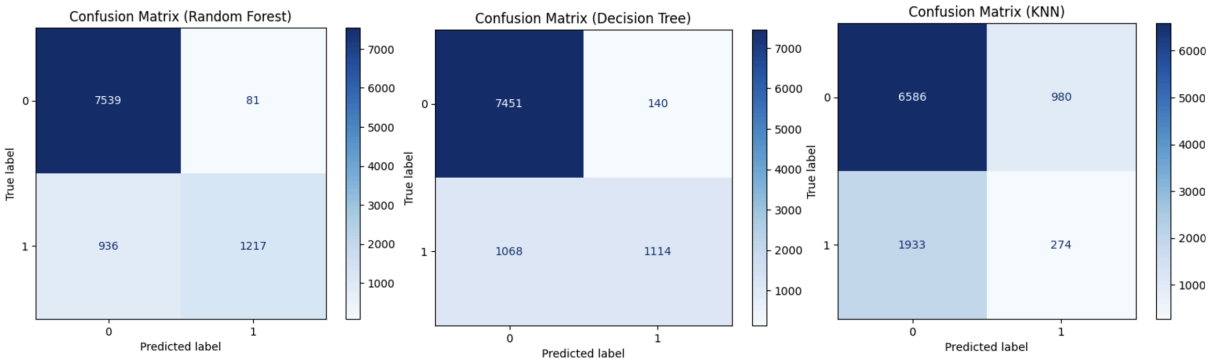
3.4- Overall comparison

The Random Forest model is clearly the best performing, offering high precision and balanced scores for both classes. Although it still shows a weakness in recall for the Approved class, it far outperforms the other two models in terms of overall performance.

The Decision Tree model is a good alternative, but it is less robust than Random Forest and shows similar difficulties in managing the Approved class well.

Finally, the KNN model performs the worst. It suffers from poor overall performance, particularly for the Approved class, which makes it unsuitable for this problem.

Verification with confusion matrix:



As we said, the Random Forest model performed best, with 7539 true negatives, 1217 true positives and few errors (81 false positives and 936 false negatives), making it well suited to discriminating between the two classes. In comparison, the Decision Tree performed well but less accurately, with 7451 true negatives, 1114 true positives and more errors (140 false positives and 1068 false negatives), making it less reliable. Finally, the KNN model performed worst, with only 6586 true negatives and 274 true positives, and a large number of errors (980 false positives and 1933 false negatives), making it unsuitable for this problem.