Introduction to AI and ML

CSL236

Project Report

Faculty Name: Mrs. Ankita Gupta

Student Name: Piyush Gambhir, Harsh Pratap, Deepshika

Roll No.: 21CSU349, 21CSU311, 21CSU287

Semester: V

Group: AIMLB (AL-3)

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2022-23

Table of Contents

Project Description

In today's fast-paced world, financial institutions are under increasing pressure to make accurate and quick decisions regarding loan applications. Traditional methods can be time-consuming and sometimes lead to errors. The Loan Approval Prediction project aims to harness the power of machine learning to predict the approval of loan applications based on a variety of criteria. By integrating data-driven insights, we can significantly improve the accuracy and efficiency of the loan approval process.

The main objective of this project is to build a machine learning model that can predict the likelihood of a loan application being approved. The model will take into account various factors like applicant's credit history, income, employment status, property area, and several others, to render a decision.

Project Statement

The Loan Approval Prediction project is an initiative aimed at reduce the time for the loan approval process within financial institutions. This project involves the creation and implementation of a sophisticated machine learning model designed to predict the outcome of loan applications with greater accuracy and speed. The model will analyze a comprehensive set of criteria, including the applicant's credit history, income, employment status, and property area, among others. The primary objective is to replace slower, error-prone traditional methods with a data-driven, efficient approach. This innovation is expected to significantly improve the decision-making process in loan approvals, benefiting both lenders and applicants by offering a more reliable and expedient service.

Analysis

Hardware Requirements

1. Processor (CPU):

- **Type:** A multi-core processor, like an Intel Core i5 or i7, or an equivalent AMD processor.
- **Purpose:** Provides the computational power necessary for processing large datasets and running complex machine learning algorithms efficiently.
- **Note:** For smaller datasets or less complex models, an Intel Core i3 processor may suffice.

2. Memory (RAM):

- **Minimum Requirement:** At least 8 GB of RAM.
- **Recommended:** 16 GB or more for handling larger datasets and more complex models.
- **Purpose:** Ensures smooth data processing and model training, especially when working with large datasets.

3. Storage:

- **Type:** SSD (Solid State Drive) is highly recommended.
- **Capacity:** Minimum 256 GB, with 512 GB or more being preferable for larger datasets and additional resources.
- **Purpose:** Faster data access and storage of large datasets, model files, and other necessary resources.

Software Requirements

1. **Operating System**:
   - **Options**: Windows, macOS, or Linux.
   - **Note**: The choice may depend on personal preference or availability.
2. **Python**:
   - **Version**: Python 3.x, with the latest stable release being preferable.
   - **Purpose**: The primary programming language for scripting and running machine learning models.
3. **Data Processing Libraries**:
   - **Primary Libraries**: Pandas for data manipulation and NumPy for numerical operations.
   - **Purpose**: Essential for data preprocessing, cleaning, and transformation.
4. **Machine Learning Libraries**:
   - **Examples**: scikit-learn for implementing machine learning algorithms, TensorFlow or PyTorch if deep learning models are required.
   - **Purpose**: To develop and train machine learning models.
5. **Development Environment**:
   - **Tool**: Jupyter Notebook or JupyterLab.
   - **Purpose**: Provides an interactive environment for writing and testing Python code, data analysis, and visualizing results.
6. **Version Control**:
   - **Tool**: Git and GitHub or similar.
   - **Purpose**: For code versioning, collaboration, and backup.
7. **Additional Tools (Optional)**:

- **Visualization Libraries**: Matplotlib, Seaborn for data visualization.
- **Advanced Data Processing**: SQL databases if dealing with structured data querying.

Design

Data/Input Output Description:

Dataset Overview: The project utilizes a dataset named **loan_approval_dataset.csv**, which contains various features related to loan applications. These features include personal and financial information of applicants, such as income, credit history, loan amount, and others.

Input Features: The dataset comprises several columns, representing different attributes that influence loan approval. Key features include:

- Applicant's income
- Co-applicant's income
- Loan amount
- Loan amount term
- Credit history
- Property area
- Marital status
- Education level

Output Variable: The target variable is the loan approval status, which is a binary value indicating whether a loan application was approved or not.

Data Processing: The data undergoes several preprocessing steps, including handling missing values, encoding categorical variables, and scaling numerical features.

Algorithmic Approach /Algorithm

Data Exploration:

- The initial step involves thoroughly examining the dataset to understand its structure.
- Key activities include identifying missing values, exploring the distribution of various features, and understanding the relationships between different variables.

Data Preprocessing:

1. **Cleaning**:

- This step focuses on removing any inconsistencies in the data, such as trailing spaces.
- Missing values are identified and appropriately handled, either by imputation or removal, depending on the context and the nature of the data.

2. **Transformation**:

    - Categorical variables are encoded using the **LabelEncoder**, which converts categorical text data into model-understandable numerical data.
    - Numerical features are standardized using the **StandardScaler** to ensure that the feature values are on a similar scale. This is crucial for models that are sensitive to the magnitude of data, such as distance-based algorithms.

Model Training:

variety of machine learning algorithms are tested to identify the one that performs the best for this specific dataset. Key algorithms include:

1. **Random Forest Classifier**:

- Random Forest is an ensemble learning technique that constructs multiple decision trees during the training phase.
- For classification tasks, it outputs the mode of the classes (the class selected by most trees) or mean prediction (regression) of the individual trees.
- Random Forest corrects for decision trees' habit of overfitting to their training set, making it more robust and accurate.
- It handles various types of data, be it categorical or numerical, without requiring much data pre-processing.

2. **Logistic Regression**:

   - Logistic Regression is a statistical model that predicts the probability of a binary outcome based on one or more predictor variables.
   - Unlike linear regression, it uses a logistic function to model a binary dependent variable, which ensures that the predictions are bounded between 0 and 1.
   - It's widely used for binary classification problems, such as spam detection or predicting whether a loan will default.

3. **Support Vector Machine (SVM)**:

   - SVM is a supervised machine learning algorithm mainly used for classification tasks but can also be used for regression.
   - It works by finding a hyperplane in an N-dimensional space that distinctly classifies data points.
   - SVM offers a powerful way of maximizing the margin between data points of different classes, making the classifier robust.

4. **K-Nearest Neighbors (KNN)**:

   - KNN is a simple, instance-based learning algorithm.
   - The algorithm stores all available cases and classifies new cases based on a similarity measure (e.g., Euclidean distance).

- It's a non-parametric method, meaning it makes no underlying assumptions about the distribution of data, making it quite versatile in handling various types of data.

5. **Gaussian Naive Bayes**:

- Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data.
- It's based on Bayes' theorem and assumes that the predictors are independent, which is a naive assumption (hence the name).
- It's particularly useful in large datasets and performs well in cases of text classification and spam filtering.

6. **Decision Tree**:

- A Decision Tree is a flowchart-like tree structure where an internal node represents a feature (or attribute), a branch represents a decision rule, and each leaf node represents the outcome.
- It's a simple to understand and interpret model, making it useful for decision analysis.
- Decision Trees can handle both categorical and numerical data and are capable of solving both regression and classification problems.

Training Process: The dataset is divided into training, validation, and testing subsets. Each model is trained on the training subset and its parameters are fine-tuned.

Model Testing and Evaluation:

- **Validation**: Each model's performance is evaluated on a validation subset. This step is crucial for understanding the model's ability to generalize to unseen data.

Evaluation Metrics:

- **Accuracy**: Measures the proportion of correctly predicted instances to the total instances. It is a straightforward metric for binary classification tasks.
- **Precision and Recall**: Important in scenarios where the cost of false positives and false negatives varies significantly. Precision measures the accuracy of positive predictions, while recall measures the ability of a classifier to find all positive instances.
- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two in cases where one may be more important than the other.
- **ROC-AUC Score**: A plot of the true positive rate against the false positive rate at various threshold settings. The area under the curve (AUC) represents a measure of separability achieved by the model.

Implementation / Output (Screenshots)

Loan Approval Prediction

In today's fast-paced world, financial institutions are under increasing pressure to make accurate and quick decisions regarding loan applications. Traditional methods can be time-consuming and sometimes lead to errors. The Loan Approval Prediction project aims to harness the power of machine learning to predict the approval of loan applications based on a variety of criteria. By integrating data-driven insights, we can significantly improve the accuracy and efficiency of the loan approval process.

The main objective of this project is to build a machine learning model that can predict the likelihood of a loan application being approved. The model will take into account various factors like applicant's credit history, income, employment status, property area, and several others, to render a decision.

Dataset Link:

https://www.kaggle.com/datasets/architsharma01/loan-approval-prediction-dataset

Dataset Description:

- **loan_id**: A unique identifier for each loan application.
- **no_of_dependents**: The number of dependents of the loan applicant.
- **education**: The education level of the loan applicant. Values include:
  - Graduate
  - Not Graduate
- **self_employed**: Indicates whether the applicant is self-employed. Possible values:
  - Yes
  - No
- **income_annum**: The annual income of the loan applicant.
- **loan_amount**: The amount of loan requested by the applicant.
- **loan_term**: The duration (in years) for which the loan is requested.
- **cibil_score**: The credit score of the loan applicant.
- **residential_assets_value**: The value of the residential assets owned by the applicant.
- **commercial_assets_value**: The value of the commercial assets owned by the applicant.
- **luxury_assets_value**: The value of luxury assets owned by the applicant.
- **bank_asset_value**: The value of assets held in banks by the applicant.
- **loan_status**: The status of the loan application. Possible values:
  - Approved
  - Rejected

Installing Dependencies

In [ ]:
```
! pip install numpy pandas matplotlib seaborn scikit-learn tabulate -q
```

Code

Importing Required Libraries

In [ ]:

```
# importing the necessary libraries

# importing pandas library and renaming it as 'pd' for convenience
import pandas as pd
# importing numpy library and renaming it as 'np' for convenience
import numpy as np
# importing matplotlib's pyplot module and renaming it as 'plt' for convenience
import matplotlib.pyplot as plt
# importing seaborn library and renaming it as 'sns' for convenience
import seaborn as sns
# importing the 'warnings' module from the Python Standard Library
import warnings


# preprocessing

# importing label encoder from sklearn
from sklearn.preprocessing import LabelEncoder
# importing standard scaler from sklearn
from sklearn.preprocessing import StandardScaler
# importing train_test_split from sklearn
from sklearn.model_selection import train_test_split


# classification models

# importing the Logistic Regression algorithm from sklearn
from sklearn.linear_model import LogisticRegression
# importing the K-Nearest Neighbors algorithm from sklearn
from sklearn.neighbors import KNeighborsClassifier
# importing the Decision Tree algorithm from sklearn
from sklearn.tree import DecisionTreeClassifier
# importing the Random Forest algorithm from sklearn
from sklearn.ensemble import RandomForestClassifier
# importing the Support Vector Machine algorithm from sklearn
from sklearn.svm import SVC
# importing the Gaussian Naive Bayes algorithm from sklearn
from sklearn.naive_bayes import GaussianNB


# performance metrics

# importing the k-fold cross validation from sklearn
from sklearn.model_selection import cross_val_score
# importing the accuracy_score metric from sklearn
from sklearn.metrics import accuracy_score
# importing the confusion_matrix metric from sklearn
from sklearn.metrics import confusion_matrix
# importing the classification_report metric from sklearn
from sklearn.metrics import classification_report
# importing the recall score metric from sklearn
from sklearn.metrics import recall_score
# importing the precision score metric from sklearn
```

```
from sklearn.metrics import precision_score
# importing the f1 score metric from sklearn
from sklearn.metrics import accuracy_score
# importing the f1 score metric from sklearn
from sklearn.metrics import f1_score
# importing confusion matrix display from sklearn
from sklearn.metrics import ConfusionMatrixDisplay
# import the roc_auc_score metric from sklearn
from sklearn.metrics import roc_curve, auc
```

Suppressing Warnings

In [ ]:
```
# setting a filter for warning messages to 'ignore,' which will suppress warning mess
warnings.filterwarnings('ignore')
```

Analyzing the Dataset

In [ ]:
```
# reading the csv file into a pandas dataframe
dataset = pd.read_csv('loan_approval_dataset.csv')
```

In [ ]:
```
# printing the first five rows of the dataset
print(dataset.head().to_markdown())
```

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value	loan_status
0	1	2	Graduate	No	9600000	29900000	12	778	2400000	17600000	22700000	8000000	Approved
1	2	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000	8800000	3300000	Rejected
2	3	3	Graduate	No	9100000	29700000	20	506	7100000	4500000	33300000	12800000	Rejected
3	4	3	Graduate	No	8200000	30700000	8	467	18200000	3300000	23300000	7900000	Rejected
4	5	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000	29400000	5000000	Rejected

In [ ]:
```
# printing the number of rows and columns in the dataset
print("\nNumber of rows and columns in the dataset: ", dataset.shape)
```

```
Number of rows and columns in the dataset:  (4269, 13)
```

In [ ]:
```
# printing the number of missing values in each column
print("\nNumber of missing values in each column: \n", dataset.isnull().sum())
```

```
Number of missing values in each column:
 loan_id                    0
 no_of_dependents           0
 education                  0
 self_employed              0
 income_annum               0
 loan_amount                0
 loan_term                  0
 cibil_score                0
 residential_assets_value   0
 commercial_assets_value    0
 luxury_assets_value        0
 bank_asset_value           0
 loan_status                0
dtype: int64
```

In [ ]:
```
# printing the descriptive statistics of the dataset
print("\nDescriptive statistics of the dataset: \n", dataset.describe().to_markdown()
```

Descriptive statistics of the dataset:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value
count	4269	4269	4269	4269	4269	4269	4269	4269	4269	4269
mean	2135	2.49871	5.05912e+06	1.51335e+07	10.9004	599.936	7.47262e+06	4.97316e+06	1.51263e+07	4.97669e+06
std	1232.5	1.69591	2.80684e+06	9.04336e+06	5.70919	172.43	6.50364e+06	4.38897e+06	9.10375e+06	3.25019e+06
min	1	0	200000	300000	2	300	-100000	0	300000	0
25%	1068	1	2.7e+06	7.7e+06	6	453	2.2e+06	1.3e+06	7.5e+06	2.3e+06
50%	2135	3	5.1e+06	1.45e+07	10	600	5.6e+06	3.7e+06	1.46e+07	4.6e+06
75%	3202	4	7.5e+06	2.15e+07	16	748	1.13e+07	7.6e+06	2.17e+07	7.1e+06
max	4269	5	9.9e+06	3.95e+07	20	900	2.91e+07	1.94e+07	3.92e+07	1.47e+07

In [ ]:
```
# printing the data types of each column
print("\nData types of each column: \n", dataset.dtypes)
```

```
Data types of each column:
 loan_id                    int64
 no_of_dependents           int64
 education                 object
 self_employed             object
 income_annum               int64
 loan_amount                int64
 loan_term                  int64
 cibil_score                int64
 residential_assets_value   int64
 commercial_assets_value    int64
 luxury_assets_value        int64
 bank_asset_value           int64
 loan_status               object
dtype: object
```

In [ ]:
```
# number of unique values in categorical columns
print("\nNumber of unique values in categorical columns: \n", dataset.select_dtypes(i
```

```
Number of unique values in categorical columns:
  education        2
 self_employed     2
 loan_status       2
dtype: int64
```

In [ ]:
```
# removing trailing spaces in column names
dataset.columns = dataset.columns.str.strip()
```

In [ ]:
```
# number of unique values in all columns
print("\nNumber of unique values in all columns: \n", dataset.nunique())
```

```
Number of unique values in all columns:
 loan_id                   4269
no_of_dependents              6
education                     2
self_employed                 2
income_annum                 98
loan_amount                 378
loan_term                    10
cibil_score                 601
residential_assets_value    278
commercial_assets_value     188
luxury_assets_value         379
bank_asset_value            146
loan_status                   2
dtype: int64
```

In [ ]:
```
# number of approved and rejected loan applications
print("\nNumber of approved and rejected loan applications: \n", dataset['loan_status
```

```
Number of approved and rejected loan applications:
 loan_status
 Approved    2656
 Rejected    1613
Name: count, dtype: int64
```

Data Pre-processing

Data Cleaning

In [ ]:
```
# removing trailing spaces in all the columns
dataset = dataset.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

In [ ]:
```
# strip the spaces in categorical columns
dataset[dataset.select_dtypes(include=['object']).columns] = dataset.select_dtypes(in
```

Data Transformation

In [ ]:
```
# identifying the categorical columns
categorical_columns = dataset.select_dtypes(include=['object']).columns

# identifying the the unique values in each categorical column
unique_values = {col: dataset[col].unique() for col in categorical_columns}

# printing the unique values in each categorical column
for key, value in unique_values.items():
    print("Unique values in column", key, "are: ", value)
```

```
Unique values in column education are:  ['Graduate' 'Not Graduate']
Unique values in column self_employed are:  ['No' 'Yes']
Unique values in column loan_status are:  ['Approved' 'Rejected']
```

In [ ]:
```
# reversing the order of the unque values in the 'education' array
unique_values['education'] = np.flip(unique_values['education'])

# reversing the order of the unique values in the 'loan_status' array
unique_values['loan_status'] = np.flip(unique_values['loan_status'])

# printing the unique values in each categorical column
for key, value in unique_values.items():
    print("Unique values in column", key, "are: ", value)
```

```
Unique values in column education are:  ['Not Graduate' 'Graduate']
Unique values in column self_employed are:  ['No' 'Yes']
Unique values in column loan_status are:  ['Rejected' 'Approved']
```

In [ ]:
```
# encoding the categorical columns
for col in categorical_columns:
    le = LabelEncoder()
    dataset[col] = le.fit_transform(dataset[col])
```

In [ ]:
```
# columns to scale

# removing the 'loan_id' and 'loan_status' columns from the list of columns to scale
columns_to_scale = dataset.columns.difference(['loan_id', 'loan_status'])
# removing all the categorical columns from the list of columns to scale
columns_to_scale = columns_to_scale.difference(categorical_columns)

# scaling the columns
scaler = StandardScaler()
dataset[columns_to_scale] = scaler.fit_transform(dataset[columns_to_scale])
```

Data Reduction

In [ ]:
```
# dropping the 'loan_id' column
dataset.drop('loan_id', axis=1, inplace=True)
```

In [ ]:
```
# printing the first five rows of the updated dataset
print("\nUpdated dataset copy:\n", dataset.head().to_markdown())
```

```
Updated dataset copy:
 |    |   no_of_dependents |   education |   self_employed |   income_annum |   loan_a
mount |   loan_term |   cibil_score |   residential_assets_value |   commercial_assets
_value |   luxury_assets_value |   bank_asset_value |   loan_status |
|---:|-------------------:|------------:|----------------:|---------------:|---------
----:|------------:|--------------:|---------------------------:|--------------------
-----:|----------------------:|-------------------:|--------------:|
|  0 |          -0.294102 |           0 |               0 |        1.61798 |      1.63
305  |    0.192617 |      1.03279  |                -0.780058  |                 2.8
7729  |             0.832028 |         0.930304   |              0 |
|  1 |          -1.47355  |           1 |               1 |       -0.34175 |     -0.32
4414 |   -0.508091 |     -1.06105  |                -0.733924  |                -0.6
31921 |            -0.694993 |        -0.515936   |              1 |
|  2 |           0.295621 |           0 |               0 |        1.43982 |      1.61
093  |    1.59403  |     -0.54484  |                -0.0573003 |                -0.1
07818 |             1.99652  |         2.40732    |              1 |
|  3 |           0.295621 |           0 |               0 |        1.11914 |      1.72
152  |   -0.508091 |     -0.771045 |                 1.64964   |                -0.3
81263 |             0.897943 |         0.899533   |              1 |
|  4 |           1.47507  |           1 |               1 |        1.68924 |      1.00
268  |    1.59403  |     -1.26406  |                 0.757724  |                 0.7
35304 |             1.56807  |         0.00717199 |              1 |
```

Training and Testing the Machine Learning Models on Various Algorithms

In [ ]:
```
# splitting the dataset into features and target variable
X = dataset.drop(columns=['loan_status'])
y = dataset['loan_status']
```

In [ ]:
```
# splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=

# printing the number of rows and columns in the training set
print("Number of rows and columns in the training set: ", X_train.shape)

# printing the number of rows and columns in the testing set
print("Number of rows and columns in the testing set: ", X_test.shape)
```

```
Number of rows and columns in the training set:  (3415, 11)
Number of rows and columns in the testing set:  (854, 11)
```

In [ ]:
```
# creating an empty dictionary to store the performance metrics of each model
performance_metrics = {}
```

Logistic Regression Model

In [ ]:
```
# applying the Logistic Regression classifier to the training set
log_reg = LogisticRegression()

# training the Logistic Regression classifier
log_reg.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

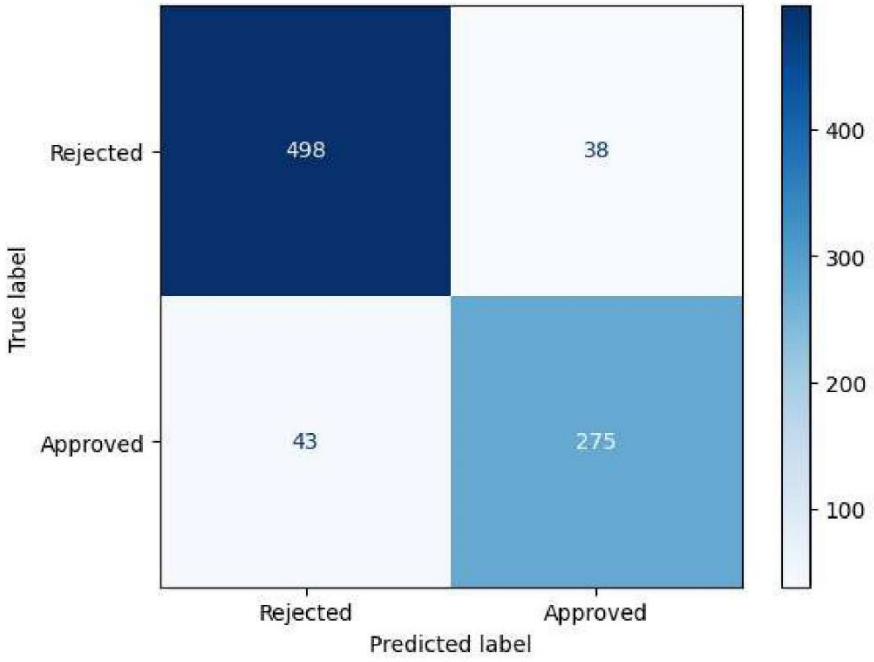
In [ ]:
```
# predicting the target variable for the testing set
y_test_pred = log_reg.predict(X_test)
```

In [ ]:
```
# evaluating the model

# confusion matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_test_pred))

# accuracy of the model
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy: ", accuracy)

# precision of the model
precision = precision_score(y_test, y_test_pred, average='macro')
print("Precision: ", precision)

# recall of the model
recall = recall_score(y_test, y_test_pred, average='macro')
print("Recall: ", recall)

# f1 score of the model
f1 = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score: ", f1)

# classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_test_pred))

#ROC curve
if hasattr(log_reg, "decision_function"):
        probas = log_reg.decision_function(X_test)
else:
    probas = log_reg.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

#saving logistic regression performance metrics
performance_metrics["Logistic Regression"] = {"Accuracy":accuracy, "Precision":precis
```

```
Confusion Matrix:
[[498  38]
 [ 43 275]]
Accuracy:  0.905152224824356
Precision:  0.8995559046376076
Recall:  0.8969421759128884
F1 Score:  0.898211707993237

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.92       536
           1       0.88      0.86      0.87       318

    accuracy                           0.91       854
   macro avg       0.90      0.90      0.90       854
weighted avg       0.90      0.91      0.90       854
```

In [ ]:
```
# printing the colored confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Rejected', 'Approved'])
cmd.plot(cmap='Blues')
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a67eece50>



K-Nearest Neighbours Model

In [ ]:
```
# applying the K-Nearest Neighbors classifier to the training set
knn = KNeighborsClassifier()

# training the K-Nearest Neighbors classifier
knn.fit(X_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

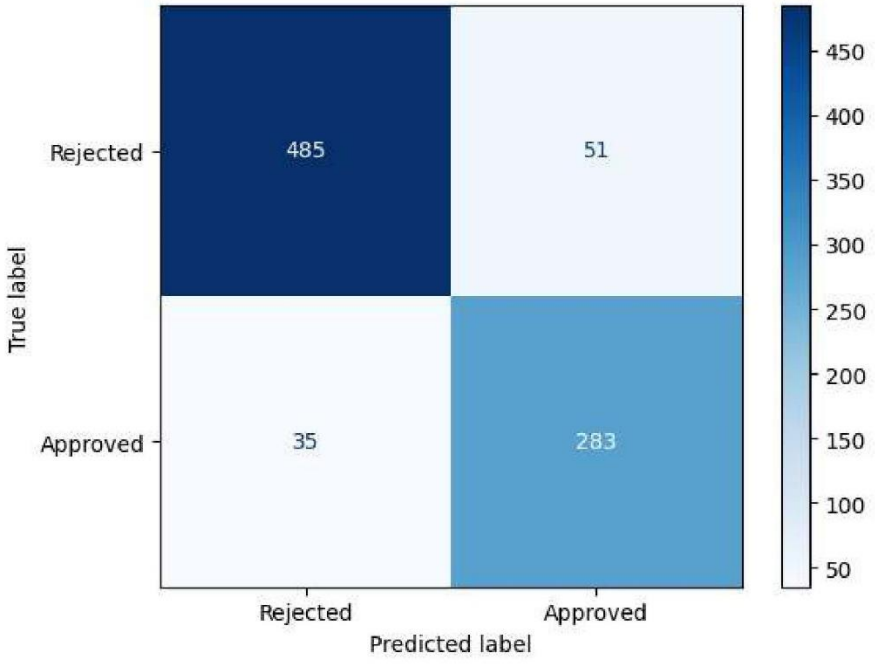
In [ ]:
```
# predicting the target variable for the testing set
y_test_pred = knn.predict(X_test)
```

In [ ]:
```
# evaluating the model

# confusion matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_test_pred))

# accuracy of the model
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy: ", accuracy)

# precision of the model
precision = precision_score(y_test, y_test_pred, average='macro')
print("Precision: ", precision)

# recall of the model
recall = recall_score(y_test, y_test_pred, average='macro')
print("Recall: ", recall)

# f1 score of the model
f1 = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score: ", f1)

# classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_test_pred))

#ROC curve
if hasattr(knn, "decision_function"):
        probas = knn.decision_function(X_test)
else:
    probas = knn.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

#saving K nearest neighbours performance metrics
performance_metrics["K Nearest Neighbour"] = {"Accuracy":accuracy, "Precision":precis
```

```
Confusion Matrix:
[[485  51]
 [ 35 283]]
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.90      0.92       536
           1       0.85      0.89      0.87       318

    accuracy                           0.90       854
   macro avg       0.89      0.90      0.89       854
weighted avg       0.90      0.90      0.90       854
```

In [ ]:
```
# printing the colored confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Rejected', 'Approved'])
cmd.plot(cmap='Blues')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a6da59510>
```



Decision Tree Model

In [ ]:
```
# applying the Decision Tree classifier to the training set
decision_tree = DecisionTreeClassifier()

# training the Decision Tree classifier
decision_tree.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

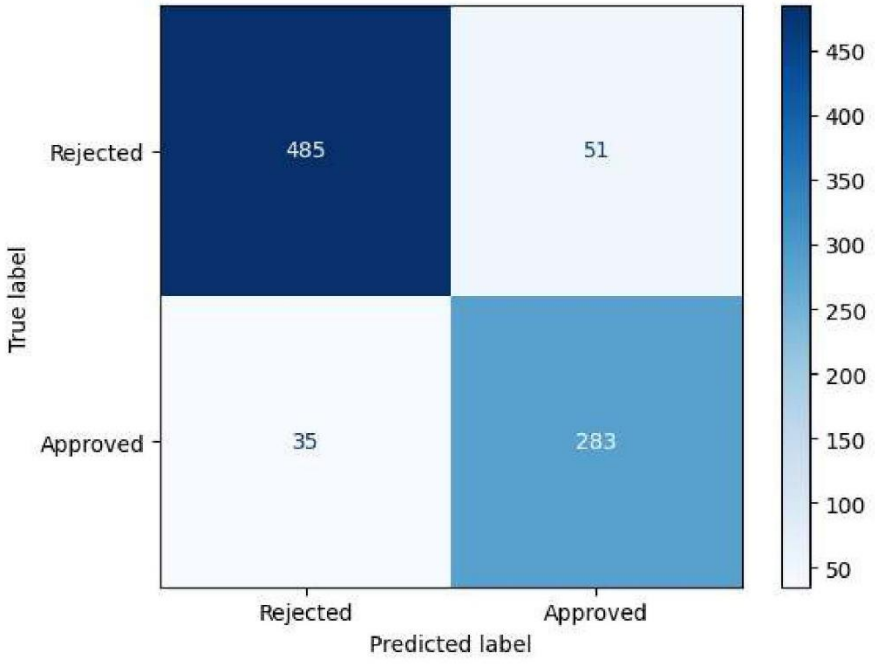
In [ ]:
```
# predicting the target variable for the testing set
y_test_pred = knn.predict(X_test)
```

In [ ]:
```
# evaluating the model

# confusion matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_test_pred))

# accuracy of the model
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy: ", accuracy)

# precision of the model
precision = precision_score(y_test, y_test_pred, average='macro')
print("Precision: ", precision)

# recall of the model
recall = recall_score(y_test, y_test_pred, average='macro')
print("Recall: ", recall)

# f1 score of the model
f1 = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score: ", f1)

# classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_test_pred))

#ROC curve
if hasattr(decision_tree, "decision_function"):
        probas = decision_tree.decision_function(X_test)
else:
    probas = decision_tree.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

#saving Decision Tree performance metrics
performance_metrics["Decision Tree"] = {"Accuracy":accuracy, "Precision":precision, "
```

```
Confusion Matrix:
[[485  51]
 [ 35 283]]
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.90      0.92       536
           1       0.85      0.89      0.87       318

    accuracy                           0.90       854
   macro avg       0.89      0.90      0.89       854
weighted avg       0.90      0.90      0.90       854
```

In [ ]:
```
# printing the colored confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Rejected', 'Approved'])
cmd.plot(cmap='Blues')
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a6d908a50>



Support Vector Machine Model

In [ ]:
```
# applying the Support Vector Machine classifier to the training set
svm = SVC()

# training the Support Vector Machine classifier
svm.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC()
```

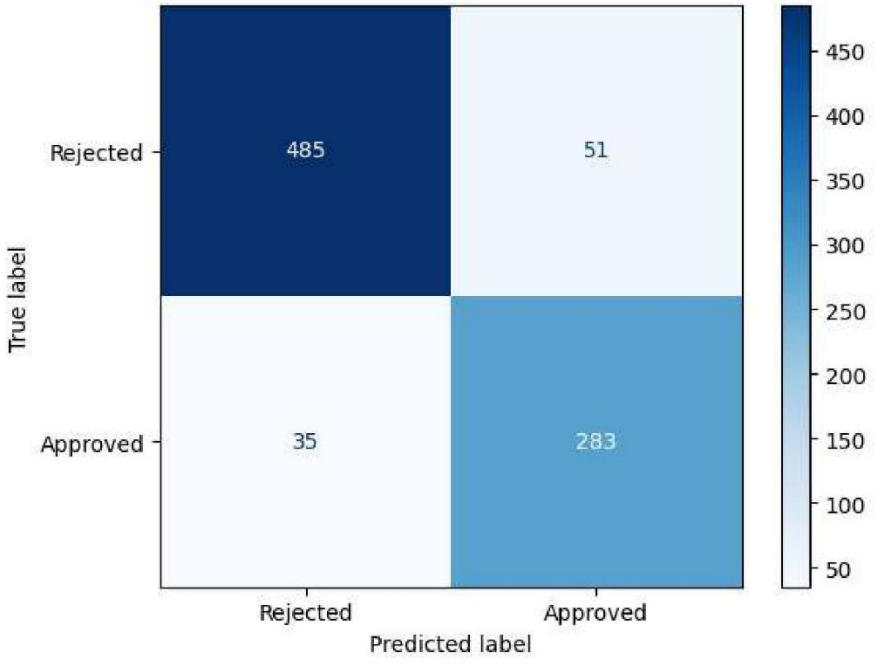
In [ ]:
```
# predicting the target variable for the testing set
y_test_pred = knn.predict(X_test)
```

In [ ]:
```
# evaluating the model

# confusion matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_test_pred))

# accuracy of the model
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy: ", accuracy)

# precision of the model
precision = precision_score(y_test, y_test_pred, average='macro')
print("Precision: ", precision)

# recall of the model
recall = recall_score(y_test, y_test_pred, average='macro')
print("Recall: ", recall)

# f1 score of the model
f1 = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score: ", f1)

# classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_test_pred))

#ROC curve
if hasattr(svm, "decision_function"):
        probas = svm.decision_function(X_test)
else:
    probas = svm.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

#saving support vector machine performance metrics
performance_metrics["Support Vector Machine"] = {"Accuracy":accuracy, "Precision":pre
```

```
Confusion Matrix:
[[485  51]
 [ 35 283]]
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.90      0.92       536
           1       0.85      0.89      0.87       318

    accuracy                           0.90       854
   macro avg       0.89      0.90      0.89       854
weighted avg       0.90      0.90      0.90       854
```

In [ ]:
```
# printing the colored confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Rejected', 'Approved'])
cmd.plot(cmap='Blues')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a6dbfea10>
```



Naive Bayes Model

In [ ]:
```
# applying the Gaussian Naive Bayes classifier to the training set
gaussian_nb = GaussianNB()

# training the Gaussian Naive Bayes classifier
gaussian_nb.fit(X_train, y_train)
```

Out[ ]:
```
▾ GaussianNB
GaussianNB()
```

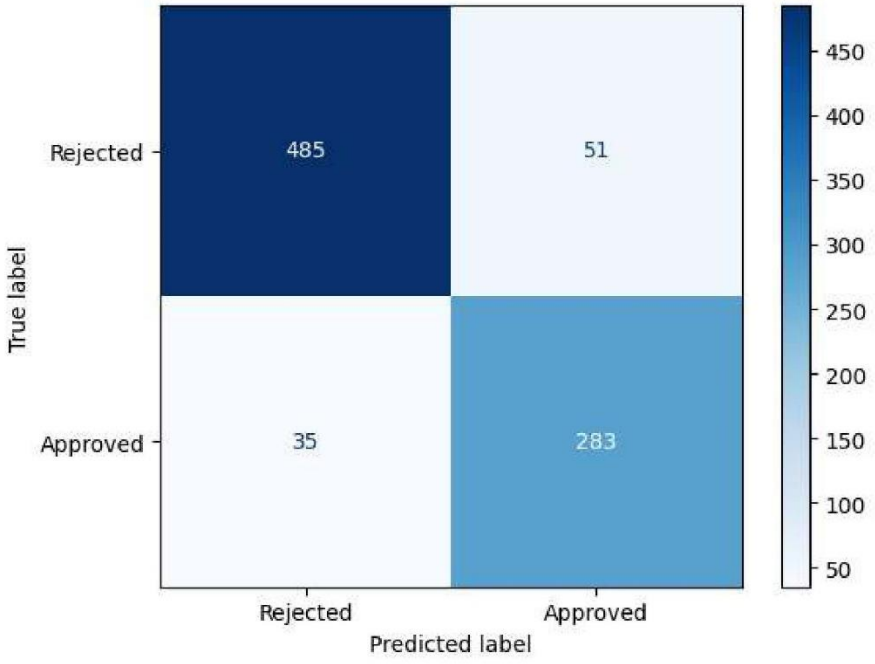
In [ ]:
```
# predicting the target variable for the testing set
y_test_pred = knn.predict(X_test)
```

In [ ]:
```
# evaluating the model

# confusion matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_test_pred))

# accuracy of the model
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy: ", accuracy)

# precision of the model
precision = precision_score(y_test, y_test_pred, average='macro')
print("Precision: ", precision)

# recall of the model
recall = recall_score(y_test, y_test_pred, average='macro')
print("Recall: ", recall)

# f1 score of the model
f1 = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score: ", f1)

# classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_test_pred))

#ROC curve
if hasattr(gaussian_nb, "decision_function"):
        probas = gaussian_nb.decision_function(X_test)
else:
    probas = gaussian_nb.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

#saving Gaussian NB performance metrics
performance_metrics["Gaussian NB"] = {"Accuracy":accuracy, "Precision":precision, "Re
```

```
Confusion Matrix:
[[485  51]
 [ 35 283]]
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.90      0.92       536
           1       0.85      0.89      0.87       318

    accuracy                           0.90       854
   macro avg       0.89      0.90      0.89       854
weighted avg       0.90      0.90      0.90       854
```

In [ ]:
```
# printing the colored confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Rejected', 'Approved'])
cmd.plot(cmap='Blues')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a6c835450>
```



Random Forest Classifier Model

In [ ]:
```
# applying random forest classifier on the training set
rf_classifier = RandomForestClassifier(random_state=42)

# training the random forest classifier
rf_classifier.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

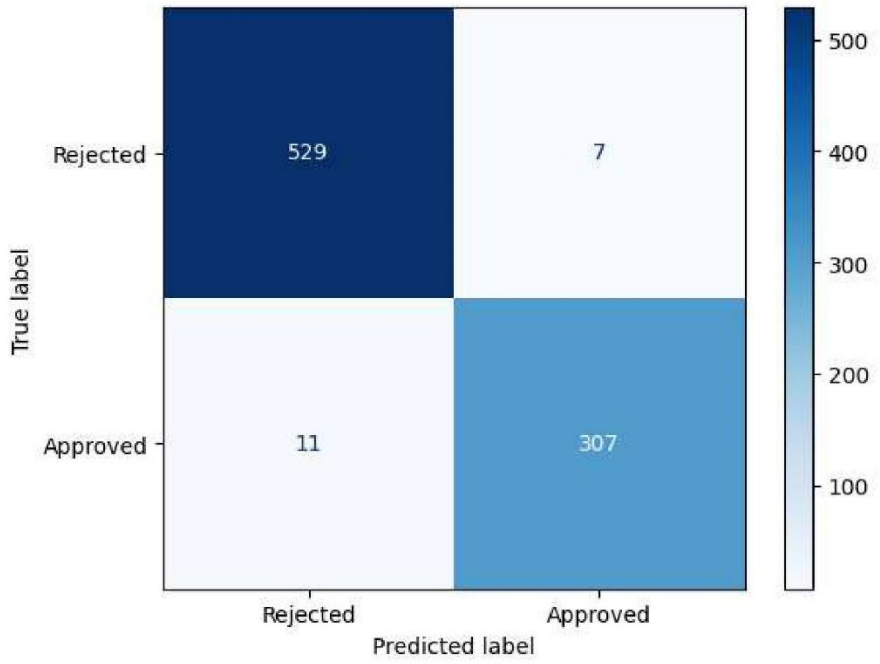
In [ ]:
```
# predicting the target variable for the testing set
y_test_pred = rf_classifier.predict(X_test)
```

In [ ]:
```
# evaluating the model

# confusion matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_test_pred))

# accuracy of the model
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy: ", accuracy)

# precision of the model
precision = precision_score(y_test, y_test_pred, average='macro')
print("Precision: ", precision)

# recall of the model
recall = recall_score(y_test, y_test_pred, average='macro')
print("Recall: ", recall)

# f1 score of the model
f1 = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score: ", f1)

# classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_test_pred))

#ROC curve
if hasattr(rf_classifier, "decision_function"):
        probas = rf_classifier.decision_function(X_test)
else:
    probas = rf_classifier.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, probas)
roc_auc = auc(fpr, tpr)

#saving Random Forest Classifier performance metrics
performance_metrics["Random Forest Classifier"] = {"Accuracy":accuracy, "Precision":p
```

```
Confusion Matrix:
[[529   7]
 [ 11 307]]
Accuracy:  0.9789227166276346
Precision:  0.9786683179995281
Recall:  0.9761745517694547
F1 Score:  0.977395181403228

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       536
           1       0.98      0.97      0.97       318

    accuracy                           0.98       854
   macro avg       0.98      0.98      0.98       854
weighted avg       0.98      0.98      0.98       854
```

In [ ]:
```
# printing the colored confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Rejected', 'Approved'])
cmd.plot(cmap='Blues')
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a6f0e7590>



Comparing the Models and Selecting the Best One

In [ ]:
```
# printing the performance metrics of each model
print("\nPerformance metrics of each model: \n")
for key, value in performance_metrics.items():
    print("Model: ", key)
    print("Accuracy: ", value["Accuracy"])
```

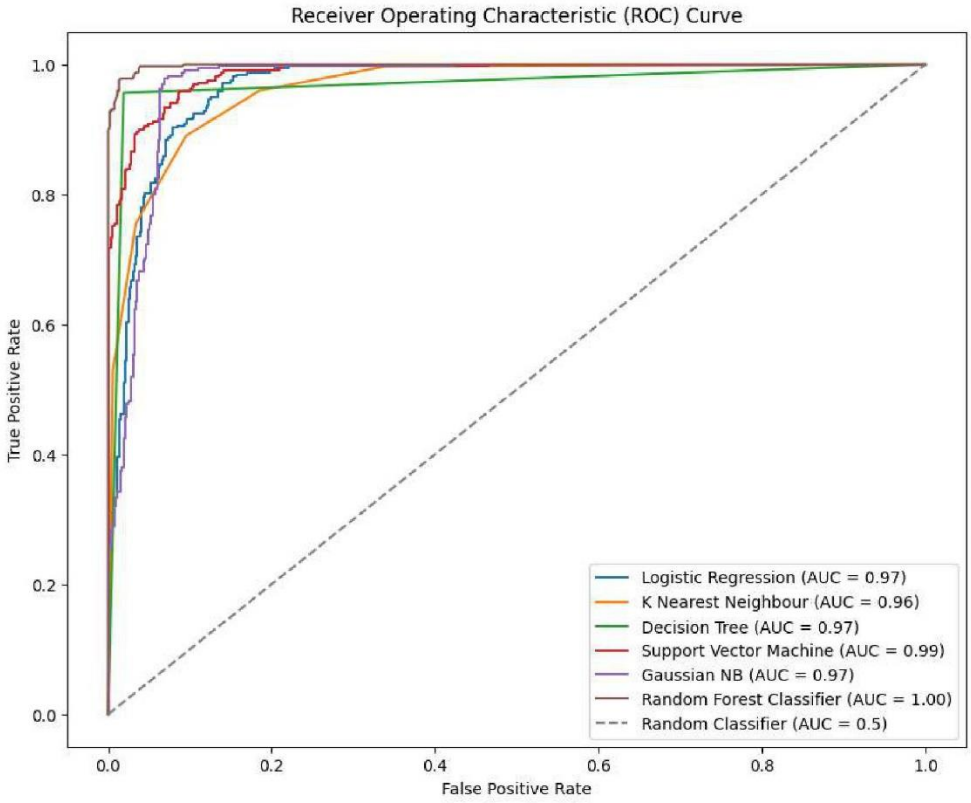
```
    print("Precision: ", value["Precision"])
    print("Recall: ", value["Recall"])
    print("F1 Score: ", value["F1 Score"])

    print("\n")
```

```
Performance metrics of each model:

Model:  Logistic Regression
Accuracy:  0.905152224824356
Precision:  0.8995559046376076
Recall:  0.8969421759128884
F1 Score:  0.898211707993237


Model:  K Nearest Neighbour
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043


Model:  Decision Tree
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043


Model:  Support Vector Machine
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043


Model:  Gaussian NB
Accuracy:  0.8992974238875878
Precision:  0.8899988484569323
Recall:  0.8973939265934479
F1 Score:  0.8933293827849043


Model:  Random Forest Classifier
Accuracy:  0.9789227166276346
Precision:  0.9786683179995281
Recall:  0.9761745517694547
F1 Score:  0.977395181403228
```

ROC Curve

In [ ]:
```
# plotting the ROC curve for each model
plt.figure(figsize=(10, 8))
for key, value in performance_metrics.items():
    fpr = value["FPR"]
    tpr = value["TPR"]
    roc_auc = value["ROC"]
    plt.plot(fpr, tpr, label=f'{key} (AUC = {roc_auc:.2f})')
```

```
# Plot the ROC curve for a random classifier (baseline)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray',
         label='Random Classifier (AUC = 0.5)')

# Set labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Making Predictions

In [ ]:
```
# sample input in the form of a dictionary
sample_input = {
    'no_of_dependents': 2,
    'education': "Graduate",
    'self_employed': 'No',
    'income_annum': 500,
    'loan_amount': 20000000,
    'loan_term': 15,
    'cibil_score': 750,
    'residential_assets_value': 1000,
    'commercial_assets_value': 5000,
    'luxury_assets_value': 250,
    'bank_asset_value': 5000
}
```

```
# converting the dictionary to DataFrame
sample_df = pd.DataFrame([sample_input])

# encoding the categorical columns
le = LabelEncoder()
sample_df['education'] = le.fit_transform(sample_df['education'])
sample_df['self_employed'] = le.fit_transform(sample_df['self_employed'])


# # scale numerical features
sample_df[columns_to_scale] = scaler.transform(sample_df[columns_to_scale])

# printing the sample input
print("Sample input:\n", sample_df.to_markdown())
```

```
Sample input:
 |    |   no_of_dependents |   education |   self_employed |   income_annum |   loan_a
mount |   loan_term |   cibil_score |   residential_assets_value |   commercial_assets
_value |   luxury_assets_value |   bank_asset_value |
|---:|-------------------:|------------:|----------------:|---------------:|----------
----:|------------:|-------------:|---------------------------:|--------------------
-----:|----------------------:|-------------------:|
|  0 |          -0.294102 |           0 |               0 |       -1.80246 |      0.53
8198 |    0.718147 |      0.870389 |                -1.14897 |                     -
1.1321 |              -1.66171 |           -1.52984 |
```

In [ ]:
```
# predicting the target variable for the sample input
sample_prediction = rf_classifier.predict(sample_df)

# printing the prediction
print("Prediction: ", "Approved" if sample_prediction[0] == 0 else "Rejected")
```

```
Prediction:  Approved
```

Conclusion

Through the Loan Approval Prediction project, we successfully implemented a machine learning solution to predict the approval of loan applications based on the provided features. The main steps undertaken in this project were:

- **Data Exploration**: We began by understanding the structure of the dataset, checking for missing values, and analyzing the statistics of each feature.
- **Data Pre-processing**: This involved data cleaning where trailing spaces were removed. Data transformation was then performed wherein categorical variables were encoded using the LabelEncoder and numerical features were scaled using the StandardScaler.
- **Model Training**: We used the Random Forest Classifier, which is known for its robustness and capability to handle a variety of data types. The dataset was split into training, validation, and testing subsets. The model was then trained on the training subset.
- **Model Testing**: We validated the model on a validation subset and checked its accuracy. This step helps in understanding how well the model will generalize to unseen data.

The achieved accuracy on the validation set indicates the model's ability to predict loan application approvals. However, there are several ways this project can be further improved:

- **Feature Engineering**: Creating new features or transforming existing features might enhance the model's predictive power.
- **Hyperparameter Tuning**: The performance of the Random Forest Classifier can be enhanced by tuning its hyperparameters.
- **Use of Advanced Models**: More sophisticated algorithms like Gradient Boosting or Neural Networks can be tried to possibly achieve better results.
- **Handling Imbalanced Data**: If the dataset is imbalanced, techniques like oversampling, undersampling, or using the Synthetic Minority Over-sampling Technique (SMOTE) can be applied.

Conclusion

In the Loan Approval Prediction project, we successfully implemented and evaluated multiple machine learning algorithms to predict the approval of loan applications. The key achievements of this project are:

1. **Comprehensive Data Analysis**: Through detailed exploration and preprocessing of the loan application dataset, we gained valuable insights into the factors affecting loan approval.
2. **Robust Model Selection and Training**: We applied a range of algorithms, including Random Forest Classifier, Logistic Regression, SVM, KNN, Gaussian Naive Bayes, and Decision Trees. Each algorithm was carefully selected for its relevance to the dataset and problem at hand.
3. **Effective Evaluation**: The models were rigorously tested and evaluated using various metrics like accuracy, precision, recall, F1 score, and ROC-AUC score. This comprehensive evaluation ensured that the chosen model accurately predicts loan approvals while minimizing errors.
4. **Insights and Knowledge Gained**: The project provided significant insights into the domain of financial analytics and machine learning, demonstrating how data-driven approaches can aid in decision-making processes.

Future Scope

Looking forward, there are several avenues for enhancing and expanding this project:

1. **Advanced Machine Learning Techniques**: Implementing more sophisticated algorithms such as deep learning and ensemble methods might yield better predictive performance.
2. **Feature Engineering**: Further analysis and engineering of features, including the creation of new features or transformation of existing ones, could enhance the models' ability to capture complex patterns in the data.
3. **Hyperparameter Optimization**: A more extensive search for optimal hyperparameters could improve model performance.
4. **Handling Imbalanced Data**: Employing advanced techniques to address class imbalance in the dataset, such as SMOTE or advanced undersampling methods, could lead to more robust models.
5. **Real-Time Application Development**: Developing a real-time application or API that integrates the model for instant loan approval predictions could be a practical next step.
6. **Cross-Domain Adaptation**: Exploring the model's adaptability to similar problems in different domains, like credit card approval or insurance underwriting, could broaden the project's impact.
7. **Explainability and Fairness in AI**: Incorporating model explainability tools and fairness metrics to ensure that the predictions are transparent and unbiased.
8. **Integration with Alternative Data Sources**: Incorporating additional data sources like social media profiles, transaction history, or behavioral data could provide a more holistic view of an applicant's financial health.

This project's success lays a strong foundation for future research and development in the field of predictive analytics in finance, opening pathways for more innovative and impactful solutions.