Project Report
Classification models on Bank Marketing Dataset

1 Dataset Introduction

In the competitive banking industry, accurately identifying customers who are most likely to respond positively to marketing campaigns is essential for optimizing resource allocation and boosting conversion rates. Effective targeting not only enhances the efficiency of marketing efforts but also minimizes operational costs and improves customer satisfaction through personalized engagement.

This project focuses on the Bank Marketing Dataset[1], a publicly available dataset from the UC Irvine Machine Learning Repository, which captures information on the direct marketing campaigns of a Portuguese bank. This project aims to evaluate and compare various predictive models to determine which approach can most effectively enhance the accuracy of customer targeting in future marketing campaigns.

The dataset encompasses a diverse array of attributes, including customer demographics, campaign-specific details, and macroeconomic indicators, making it an invaluable resource for building predictive models to identify high-potential customers. Key features such as age, occupation, marital status, prior campaign outcomes, and economic variables provide crucial insights into customer behavior and the effectiveness of marketing campaigns. The detailed descriptions of the variables are presented in Table 1. For this project, we focus on the 'bank-full.csv'[2] dataset, which contains 45,211 observations, 16 features, and 1 target variable.

2 Data Analysis[3]

After loading the dataset, the initial step was to check for missing values, as their presence could compromise the accuracy of predictions or classifications. Fortunately, no missing values were found in the dataset. However, a challenge arose during data preparation due to the presence of mixed data types across the columns. The dataset contains both numeric and categorical variables, with the target variable y being binary.

To address this, quantitative values were retained in numeric format, while categorical variables were processed using one-hot encoding. This technique transformed each categorical variable into a set of binary columns, with each column representing a unique category. One-hot encoding ensures compatibility with machine learning models, which typically cannot process categorical data directly while also avoiding the imposition of an ordinal relationship between categories.

The distributions and count plots of the variables are presented in the following Figures 2. From the final plot, which depicts the target variable y, it is evident that the dataset is imbalanced, with a higher proportion of people declining to subscribe to a term deposit compared to those who agree.

3 Overview of Some Classification Methods

In this project, we applied a range of supervised machine learning techniques to develop predictive models for the Bank Marketing Dataset. Each algorithm brings a unique perspective to the classification task, and their performance was evaluated to determine the most effective approach for predicting customer subscription to term deposits. Below is an overview of the machine learning techniques used:

[1]Bank Marketing Dataset
[2]This dataset is located within the folder named 'bank'.
[3]This project is coding in Python 3.

- **Logistic Regression**
  Logistic Regression is a statistical method widely used for binary classification problems. Logistic Regression builds upon a linear model, which fits the training data using the equation: $y = w^{\intercal}x = w_0 + w_1x_1 + w_2x_2 + \ldots$, where y represents the target variable, w_0 is the bias, $x_1, x_2, x_3, \ldots$ are the feature vectors, and $w_1, w_2, w_3, \ldots$ are the corresponding weights. And Logistic Regression extends this linear model by applying the sigmoid function, defined as: $P(X) = 1/(1 + e^{-y})$. This transformation is critical for classification problems, as a linear model produces outputs in the range $[-\infty, +\infty]$, which is unsuitable for probabilities that must lie in the range $[0, 1]$. The sigmoid function maps the linear model's output to a probability, making it ideal for binary classification tasks.

  In terms of coding, Python provides an implementation through the `LogisticRegression` class in the `sklearn.linear_model` module.

- **Decision Tree**
  A Decision Tree is a versatile machine learning model commonly used for both classification and regression tasks. It works by recursively splitting the dataset into smaller subsets based on specific feature thresholds, creating a tree-like structure. Each internal node of the tree represents a decision based on a feature, branches correspond to the possible outcomes of that decision, and the leaf nodes represent the predicted class labels or regression values. The splits are determined using criteria such as Gini Impurity or Information Gain for classification tasks, aiming to partition the data in a way that minimizes impurity or maximizes information gain. For classification, the final prediction is the majority class in a leaf node, while for regression, it is the mean value of the target variable in that node. The recursive splitting process continues until stopping criteria are met, such as reaching a maximum tree depth, having all samples in a node belong to the same class, or having a node contain fewer samples than a specified threshold.

  In Python, Decision Trees can be implemented using the `DecisionTreeClassifier` class in the `sklearn.tree` module.

- **Random Forest**
  Random Forest is an ensemble learning method that builds multiple decision trees during training and combines their outputs for classification. It operates by constructing a "forest" of decision trees during training and aggregates their predictions to improve performance and robustness. Each tree in the forest is trained on a randomly sampled subset of the training data (using bootstrapping) and considers a random subset of features for splitting at each node. This randomness introduces diversity among the trees, which helps reduce overfitting and enhances generalization. For classification tasks, the model predicts the class based on majority voting across all trees, while for regression, it averages the predictions of all trees.

  Random Forest offers several advantages, such as handling large datasets with many features effectively, reducing overfitting compared to a single decision tree, and performing well with both numerical and categorical data. However, it can be computationally expensive when dealing with very large datasets and is less interpretable than a single decision tree.

  In Python, Random Forest can be implemented using the `RandomForestClassifier` class from the `sklearn.ensemble` module.

- **Support Vector Machine (SVM)**
  SVM aims to find the hyperplane that best separates the classes in a high-dimensional space. The use of kernels allows SVM to handle non-linear relationships effectively. For this project, we used the probabilistic variant of SVM for better interpretability of predictions.

  SVM can handle both linear and non-linear classification problems. For linearly separable data, it identifies a linear decision boundary. For non-linear data, SVM uses kernel functions (such as polynomial, sigmoid) to transform the data into a higher-dimensional space where a linear separator can be found. This makes SVM highly effective for complex datasets with non-linear relationships.

  The key advantages of SVM include its ability to handle high-dimensional spaces and its effectiveness in scenarios where the number of dimensions exceeds the number of samples. However, SVM can be computationally expensive for large datasets and may require careful tuning of parameters like the regularization parameters.

  In Python, SVM can be implemented using the `SVC` (Support Vector Classifier) class from the `sklearn.svm` module.

- **Naive Bayes**
  Naive Bayes is a probabilistic classification algorithm that is based on Bayes' Theorem and assumes that the features of the dataset are conditionally independent. Despite this simplifying assumption, Naive Bayes often performs well in practice, particularly in applications like text classification, spam filtering, and sentiment analysis. The algorithm uses Bayes' Theorem to compute the posterior probability $P(Y|X)$ of a class Y given a feature vector X, expressed as: $P(C|X) = \frac{P(X|C)P(C)}{P(X)}$. Here, $P(Y|X)$ is the posterior probability of the class Y given the features X, $P(X|Y)$ is the likelihood of the features given the class, $P(Y)$ is the prior probability of the class, and $P(X)$ is the marginal probability of the features. Since $P(X)$ is constant for all classes, the algorithm focuses on maximizing $P(X|Y)P(Y)$ to predict the most likely class.

  Naive Bayes classifiers are typically divided into three types based on the nature of the feature distribution. Gaussian Naive Bayes assumes that continuous data follows a Gaussian (normal) distribution. Multinomial Naive Bayes is suitable for discrete data, such as word counts in text classification. Bernoulli Naive Bayes is designed for binary or boolean features. The simplicity of Naive Bayes, combined with its efficiency and effectiveness for certain tasks, makes it a popular choice for many classification problems.

  In Python, it can be implemented using the `GaussianNB` class from the `sklearn.naive_bayes` module in our case.

- ***k*-Nearest Neighbors (*k*-NN)**
  The k-Nearest Neighbors (k-NN) algorithm is a simple and intuitive supervised learning method used for both classification and regression tasks. It is a non-parametric and lazy learning algorithm, meaning it makes no assumptions about the underlying data distribution and does not train a model explicitly. Instead, k-NN predicts the output for a new data point based on the k nearest data points in the training dataset.

  For classification, the predicted class of a new data point is determined by the majority class among its k nearest neighbors, where k is a user-defined parameter. For regression, the prediction is typically the average (or weighted average) of the values of the k nearest neighbors. The distance metric used to determine "nearest" neighbors is typically Euclidean distance, but other metrics such as Manhattan or Minkowski distances can also be used. The choice of k is crucial for the performance of the algorithm. A small k may lead to overfitting, while a large k may smooth the decision boundary too much, leading to underfitting.

  Regarding the coding, k-NN can be implemented using the `KNeighborsClassifier` class from the `sklearn.neighbors` module.

Each model was trained and evaluated on the dataset using standard metrics such as accuracy, precision, recall, the F1-score, and AUC to compare their performance. The goal was to identify the model that best balances predictive accuracy and interpretability for practical use in marketing decision-making.

4 Results Comparison

In this section, we evaluated six machine learning models—Logistic Regression, Decision Tree, Random Forest, SVM, Naive Bayes, and k-NN—on their ability to predict whether customers would subscribe to a term deposit. The models were assessed using key metrics such as accuracy, AUC (Area Under the Curve), precision, recall, and F1-score, see the following. These metrics provided a comprehensive view of their overall performance and their ability to handle the imbalanced nature of the dataset. I also provided a visualized plot of these results in Figure 3 and draw the ROC curves for each model in Figure 4.

Model: Logistic Regression
Accuracy: 0.90
AUC: 0.90
Classification Report:

```
              precision    recall  f1-score   support

       False       0.92      0.98      0.95     11977
        True       0.65      0.35      0.45      1587

    accuracy                           0.90     13564
   macro avg       0.79      0.66      0.70     13564
weighted avg       0.89      0.90      0.89     13564
```

Model: Decision Tree
Accuracy: 0.87
AUC: 0.69
Classification Report:

```
              precision    recall  f1-score   support

       False       0.93      0.93      0.93     11977
        True       0.45      0.46      0.45      1587

    accuracy                           0.87     13564
   macro avg       0.69      0.69      0.69     13564
weighted avg       0.87      0.87      0.87     13564
```

Model: Random Forest
Accuracy: 0.91
AUC: 0.93
Classification Report:

```
              precision    recall  f1-score   support

       False       0.92      0.98      0.95     11977
        True       0.68      0.39      0.50      1587

    accuracy                           0.91     13564
   macro avg       0.80      0.68      0.72     13564
weighted avg       0.90      0.91      0.90     13564
```

Model: SVM
Accuracy: 0.90
AUC: 0.89
Classification Report:

```
              precision    recall  f1-score   support

       False       0.92      0.98      0.95     11977
        True       0.67      0.33      0.44      1587

    accuracy                           0.90     13564
   macro avg       0.79      0.65      0.69     13564
weighted avg       0.89      0.90      0.89     13564
```

Model: Naive Bayes
Accuracy: 0.86
AUC: 0.81
Classification Report:

```
              precision    recall  f1-score   support

       False       0.93      0.91      0.92     11977
        True       0.42      0.47      0.44      1587

    accuracy                           0.86     13564
   macro avg       0.67      0.69      0.68     13564
weighted avg       0.87      0.86      0.87     13564
```

Model: k-NN
Accuracy: 0.90
AUC: 0.83
Classification Report:

```
              precision    recall  f1-score   support

       False       0.92      0.97      0.94     11977
        True       0.60      0.33      0.43      1587

    accuracy                           0.90     13564
   macro avg       0.76      0.65      0.68     13564
weighted avg       0.88      0.90      0.88     13564
```

Among the models, Random Forest demonstrated the best overall performance, achieving the highest accuracy (0.91) and AUC (0.93). This indicates its superior ability to differentiate between the two classes, even in the presence of class imbalance. Logistic Regression and SVM closely followed, both achieving an accuracy of 0.90, with AUC values of 0.90 and 0.89, respectively. k-NN also performed well, achieving an accuracy of 0.90 and an AUC of 0.83, though its ability to handle the minority class was more limited. On the other hand, Decision Tree and Naive Bayes exhibited lower overall effectiveness, with accuracies of 0.87 and 0.86 and AUC values of 0.69 and 0.81, respectively.

For the majority class (`False`), most models performed strongly. Random Forest, Logistic Regression, SVM, and k-NN all achieved high precision and recall values, with Random Forest leading the group. Decision Tree and Naive Bayes also performed well for the majority class, though their overall metrics were slightly lower. However, when focusing on the minority class (`True`), the challenges posed by class imbalance became apparent. Random Forest emerged as the best model for this class, with a precision of 0.68, recall of 0.39, and an F1-score of 0.50. Logistic Regression and SVM showed moderate performance for the minority class, both achieving an F1-score of approximately 0.45. Naive Bayes, while having the lowest precision for the minority class (0.42), had a relatively higher recall (0.47), leading to an F1-score of 0.44. k-NN showed better precision for the minority class (0.60) but struggled with recall (0.33), resulting in an F1-score of 0.43.

Examining the macro and weighted averages further highlights the strengths and weaknesses of the models. Random Forest led in macro averages, with precision, recall, and F1-scores of 0.80, 0.68, and 0.72, respectively, indicating its balanced performance across both classes. Logistic Regression and SVM also performed well, with macro F1-scores of 0.70 and 0.69, respectively. Naive Bayes and k-NN had lower macro averages due to their limited handling of the minority class. Weighted averages, which consider class sizes, showed that Random Forest, Logistic Regression, and SVM had nearly identical performance, with precision, recall, and F1-scores around 0.89–0.91. Naive Bayes and k-NN had weighted averages of approximately 0.86–0.88, reflecting their slightly lower overall effectiveness.

5 Conclusion

Overall, Random Forest emerged as the most robust model for this task, excelling in both overall and class-specific performance metrics. Its ability to effectively handle class imbalance makes it the strongest candidate for deployment. Logistic Regression and SVM also proved to be competitive options, especially for scenarios requiring simpler models or faster predictions. Simpler models like Decision Tree and Naive Bayes were less effective, struggling to balance performance across the imbalanced classes. k-NN, while showing reasonable accuracy, was limited by its low recall for the minority class.

The results underscore the challenge posed by class imbalance in the Bank Marketing Dataset, as most models performed better for the majority class than the minority class. Addressing this imbalance through techniques such as oversampling or undersampling could further enhance model performance. Nevertheless, Random Forest's superior performance highlights its suitability for predicting customer subscriptions to term deposits, with Logistic Regression and SVM offering viable alternatives for specific use cases.

6 Lessons Learned and Impact

This project deepened my understanding of machine learning, particularly the importance of addressing class with imbalance and evaluating models using metrics beyond accuracy. Comparing models like Random Forest, Logistic Regression, and SVM highlighted their strengths and limitations, teaching me the value of balancing performance and interpretability.

The hands-on experience enhanced my skills in data preprocessing, feature engineering, and model evaluation while fostering a systematic approach to problem-solving.

Overall, this project not only improved my technical expertise but also prepared me to tackle real-world data challenges more effectively.

Variable	**Type**	**Data Type**	**Description**
age	Feature	Integer	Age
job	Feature	Categorical	Type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
marital	Feature	Categorical	Marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed)
education	Feature	Categorical	Education level (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')
default	Feature	Binary	Has credit in default?
balance	Feature	Integer	Average yearly balance
housing	Feature	Binary	Has housing loan?
loan	Feature	Binary	Has personal loan?
contact	Feature	Categorical	Contact communication type (categorical: 'cellular', 'telephone')
day_of_week	Feature	Date	Last contact day of the week
month	Feature	Date	Last contact month of year (categorical: 'jan', 'feb', ..., 'nov', 'dec')
duration	Feature	Integer	Last contact duration, in seconds (numeric). Important note: this attribute highly affects the target (e.g., if duration=0, then y='no'). Yet, the duration is not known before a call is performed.
campaign	Feature	Integer	Number of contacts performed during this campaign and for this client (numeric, includes last contact).
pdays	Feature	Integer	Number of days that passed by after the client was last contacted from a previous campaign (-1 means client was not previously contacted).
previous	Feature	Integer	Number of contacts performed before this campaign and for this client (numeric).
poutcome	Feature	Categorical	Outcome of the previous marketing campaign (categorical: 'failure', 'none', 'success').
y	Target	Binary	Has the client subscribed to a term deposit?

Table 1: Detailed Description of Variables in the Bank Marketing Dataset

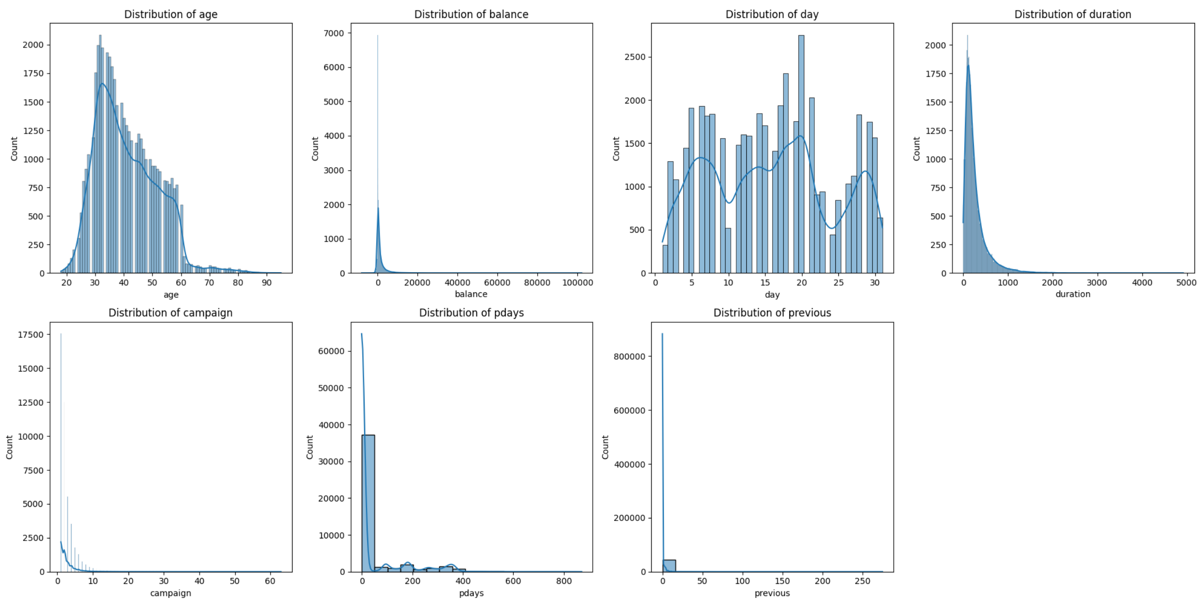

Figure 1: Distribution of numerical variables

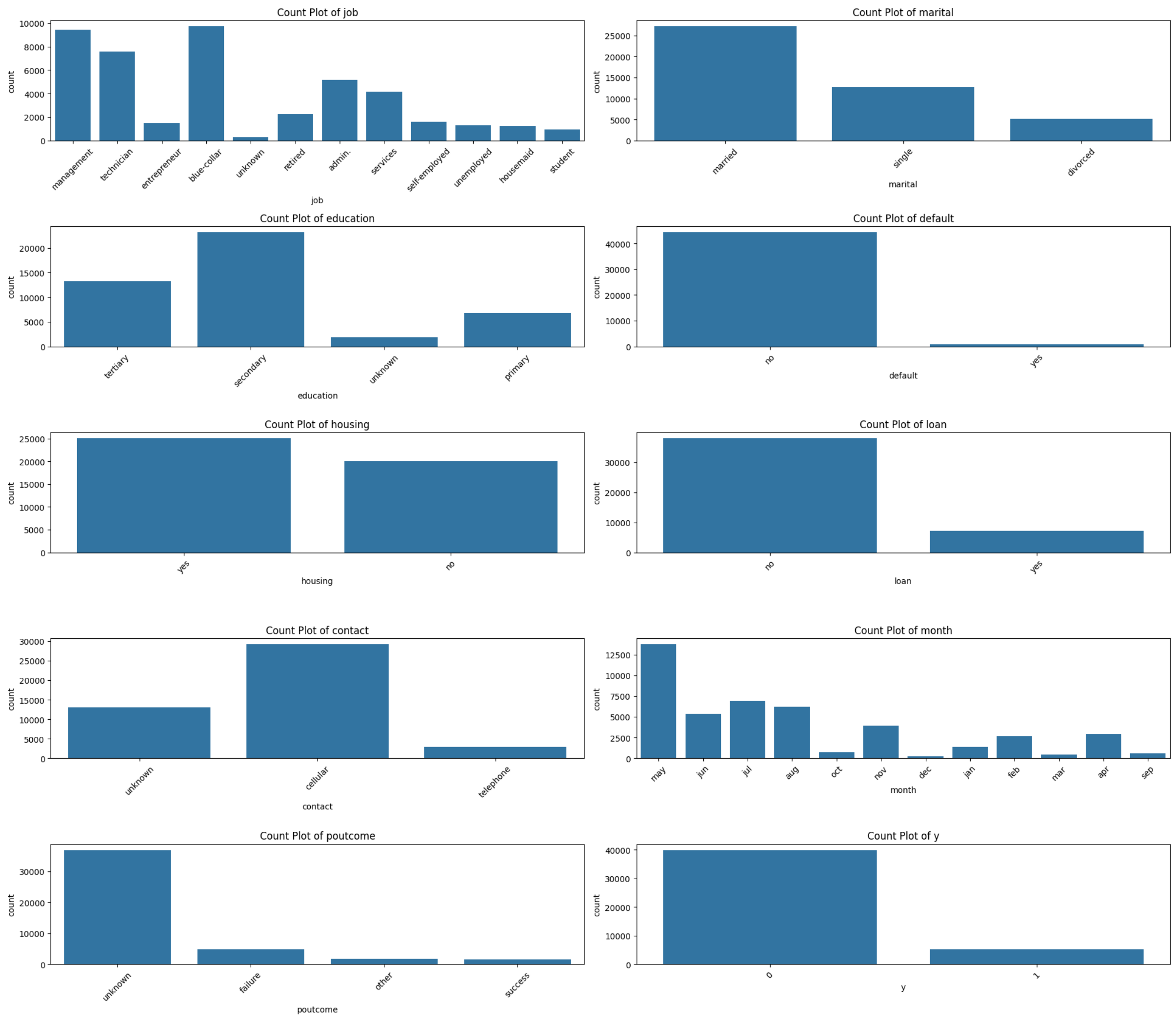

Figure 2: Count plots of categorical variables

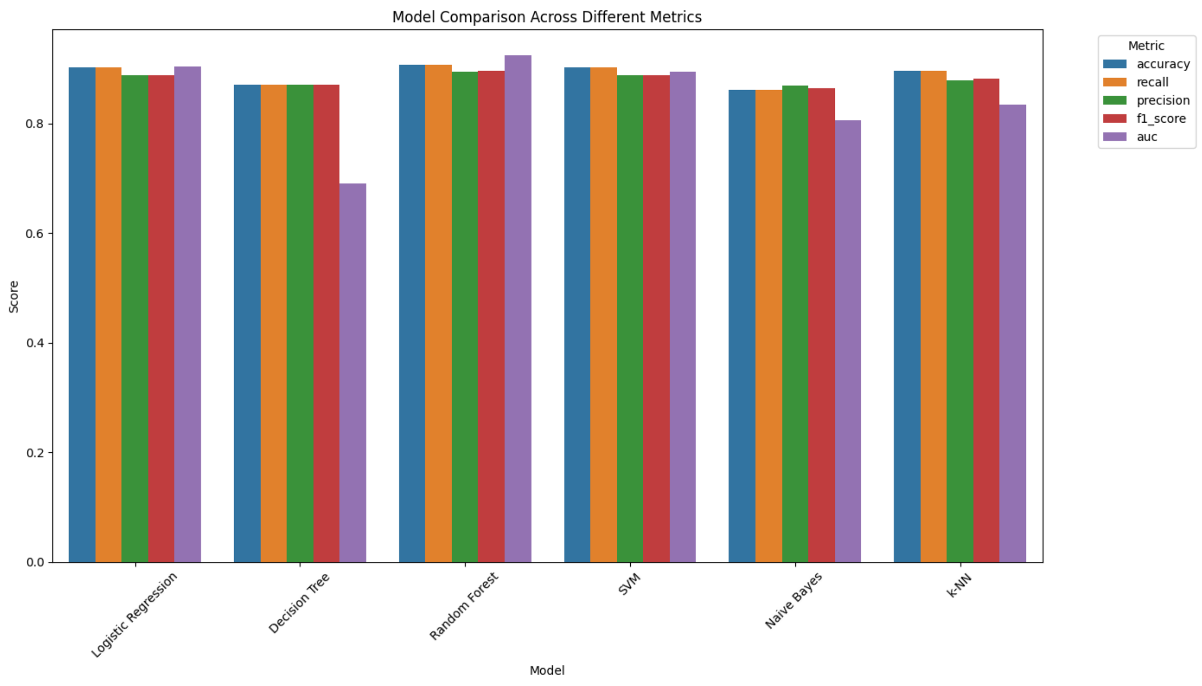

Figure 3: Models comparison across different metrics

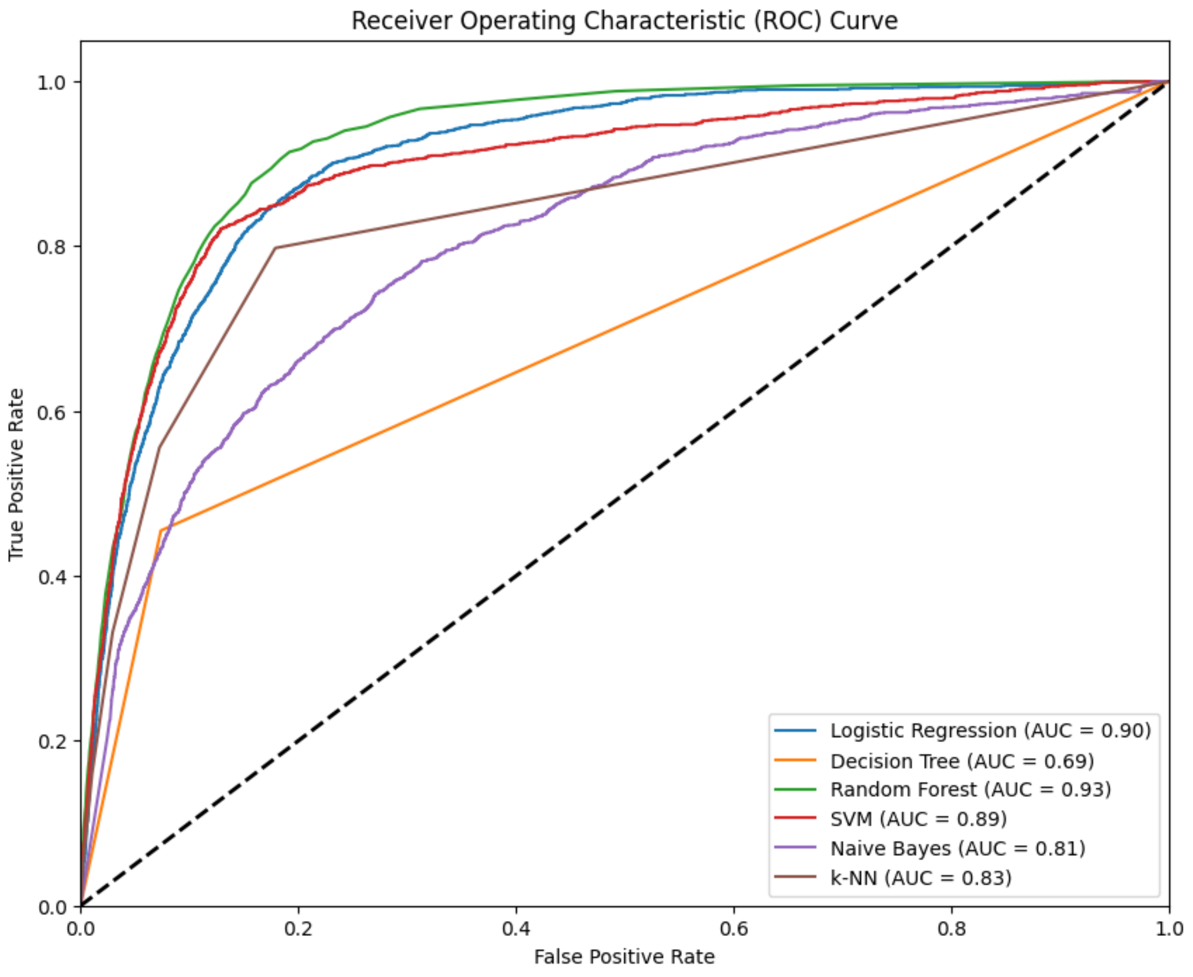

Figure 4: ROC Curve