Bank Telemarketing Success Prediction

Course Project: Machine Learning Practice (BSCS2008),

Prof. Ashish Tendulkar

Name: Aditya Rajput

Roll Number: 23F1002621

Date of Submission: 30/11/24

IITM Online BS Degree Program,

Indian Institute of Technology, Madras,

Chennai Tamil Nadu, India, 600036

Abstract

This project focused on a Kaggle competition involving direct marketing campaigns of a banking institution, where the goal was to predict whether clients would subscribe to a term deposit based on campaign data. The dataset contained features related to client demographics, financial information, and contact history. The problem was framed as a binary classification task, evaluated using the macro F1-score.

Through rigorous data preprocessing, feature engineering, and experimentation with multiple machine learning models, the project achieved a macro F1-score of 0.7748, securing a rank of 54 among 1200+ participants. Key steps included handling categorical and numerical variables, optimizing model hyperparameters, and evaluating performance on unseen test data.

The results of this project have significant implications for targeted marketing strategies. By accurately predicting customer responses, financial institutions can optimize resource allocation, personalize client outreach, and improve campaign efficiency. This approach has the potential to be applied across various industries to enhance customer engagement, reduce costs, and increase product adoption.

Introduction to Dataset

The dataset used in this project pertains to a direct marketing campaign by a banking institution aimed at promoting a term deposit product. The campaign was carried out through phone calls, with multiple contacts often being necessary to determine whether a client would subscribe to the bank's term deposit product. The dataset is structured with features representing various attributes of the clients and their interactions with the marketing campaign, along with an indicator of whether the client eventually subscribed to the product.

The dataset consists of three main files:

- **train.csv**: This file contains the training data used to build and train machine learning models.
- **test.csv**: This file contains the test data used to evaluate the performance of the trained model.
- **sample_submission.csv**: This file provides a template for submitting predictions in the correct format.

The dataset includes 16 input variables, which describe both demographic and behavioral characteristics of the clients, as well as the specifics of their interactions during the marketing campaign. These variables include both numerical and categorical data, such as:

- **Age**: The age of the client (numeric).
- **Job**: The type of job the client holds (categorical).
- **Marital Status**: The marital status of the client (categorical).
- **Education**: The education level of the client (categorical).
- **Balance**: The average yearly balance in euros (numeric).
- **Housing Loan**: Whether the client has a housing loan (binary: "yes", "no").
- **Personal Loan**: Whether the client has a personal loan (binary: "yes", "no").
- **Previous Outcome**: The outcome of previous marketing campaigns (categorica),

Additionally, the dataset includes behavioral data related to the campaign:

- **Duration**: The duration of the last contact, in seconds (numeric).
- **Campaign**: The number of contacts made during this campaign (numeric).
- **Pdays**: The number of days since the client was last contacted in a previous campaign (numeric, with -1 indicating the client was not previously contacted).
- **Previous**: The number of contacts made before this campaign (numeric).
- **Poutcome**: The outcome of the previous marketing campaign (categorical).

The target variable is whether or not the client subscribed to the term deposit, represented as a binary variable:

- **Target**: Whether the client subscribed to the term deposit (binary: "yes", "no").

This dataset provides valuable insights into customer demographics, loan status, and past campaign interactions, which can be leveraged to predict the likelihood of a client subscribing to a term deposit. The analysis of these features will help in building a predictive model for identifying high-potential clients for future marketing campaigns.

Exploratory Data Analysis

Get the Data

For the initial stages of the project, the following essential libraries were imported into the Kaggle Notebook to assist with data analysis and preprocessing:

- **NumPy**: Used for performing numerical operations, such as mathematical calculations on arrays and matrices.
- **Pandas**: A key library for data manipulation and analysis, providing easy-to-use data structures like DataFrames for handling the dataset.
- **Matplotlib.pyplot**: Used to create various visualizations, such as graphs and plots, to explore and present the data.
- **Seaborn**: A powerful statistical data visualization library built on top of Matplotlib, which allows for more advanced plotting capabilities, including heatmaps, pair plots, etc.

Once the libraries were imported, the training and test datasets were loaded into separate Pandas DataFrames for analysis.

- The **training data** consisted of 39,211 rows and 16 columns.
- The **test data** contained 10,000 rows and 15 columns, with the target variable missing (as it was not available in the test set).

To gain an initial understanding of the dataset, the first few rows of the training data were inspected using the .head() method in Pandas, which provides a snapshot of the top entries in the DataFrame. This allowed for a quick inspection of the structure and content of the data.

Next, the .info() method was applied to get an overview of the dataset, including the number of non-null entries for each column and the data types of the columns. The results provided valuable insights into the completeness and structure of the dataset, such as identifying which columns contained missing values.

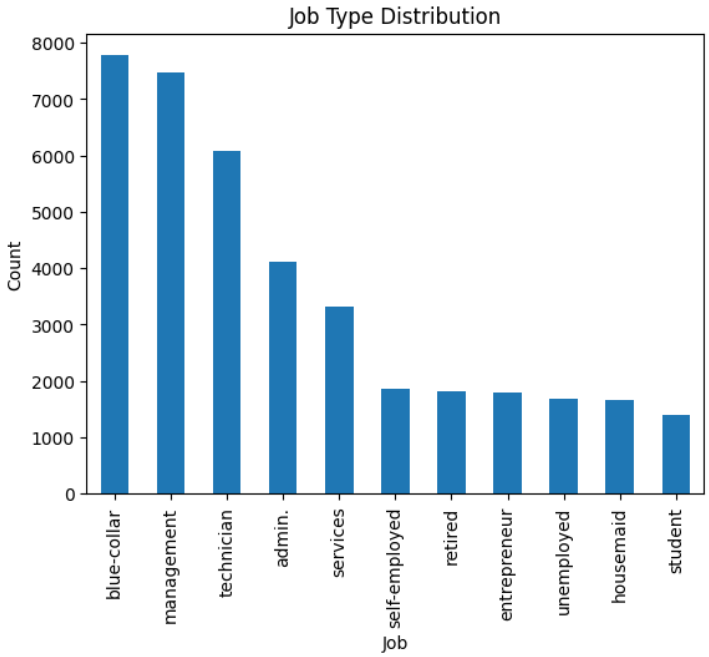
To quantify missing data, the .isna().sum() method was used to calculate the total number of null values in each column. This helped pinpoint columns with missing data that might require imputation or further handling.

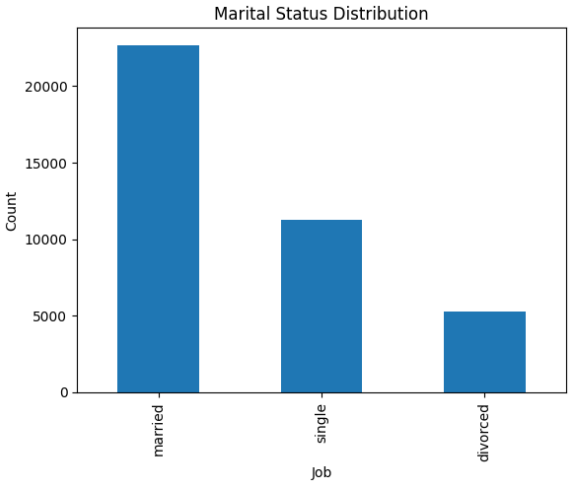
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39211 entries, 0 to 39210
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   last contact date  39211 non-null  object
 1   age                39211 non-null  int64
 2   job                38982 non-null  object
 3   marital            39211 non-null  object
 4   education          37744 non-null  object
 5   default            39211 non-null  object
 6   balance            39211 non-null  int64
 7   housing            39211 non-null  object
 8   loan               39211 non-null  object
 9   contact            28875 non-null  object
 10  duration           39211 non-null  int64
 11  campaign           39211 non-null  int64
 12  pdays              39211 non-null  int64
 13  previous           39211 non-null  int64
 14  poutcome           9760 non-null   object
 15  target             39211 non-null  object
dtypes: int64(6), object(10)
memory usage: 4.8+ MB
```

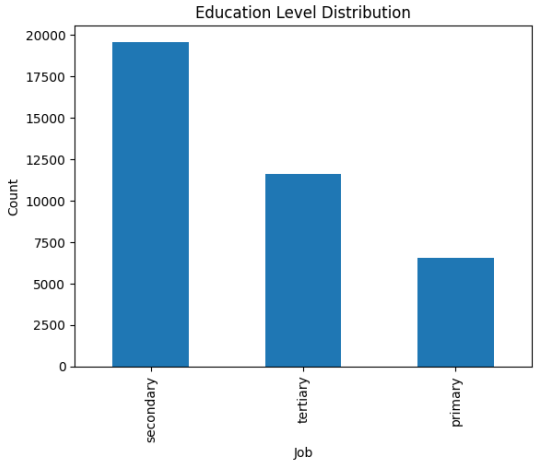
```
last contact date        0
age                      0
job                    229
marital                  0
education             1467
default                  0
balance                  0
housing                  0
loan                     0
contact              10336
duration                 0
campaign                 0
pdays                    0
previous                 0
poutcome             29451
target                   0
dtype: int64
```

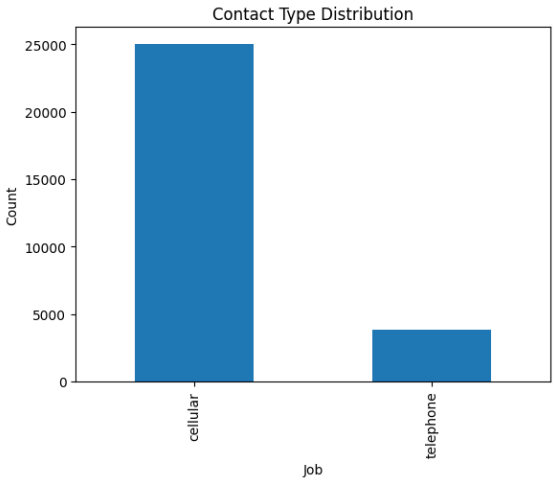
Exploratory Data Analysis (EDA) on Categorical Variables

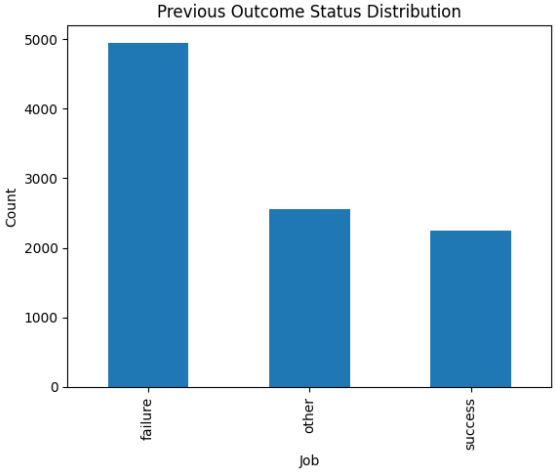
The first step in EDA involved examining the distribution of categorical variables. The results were as follows:











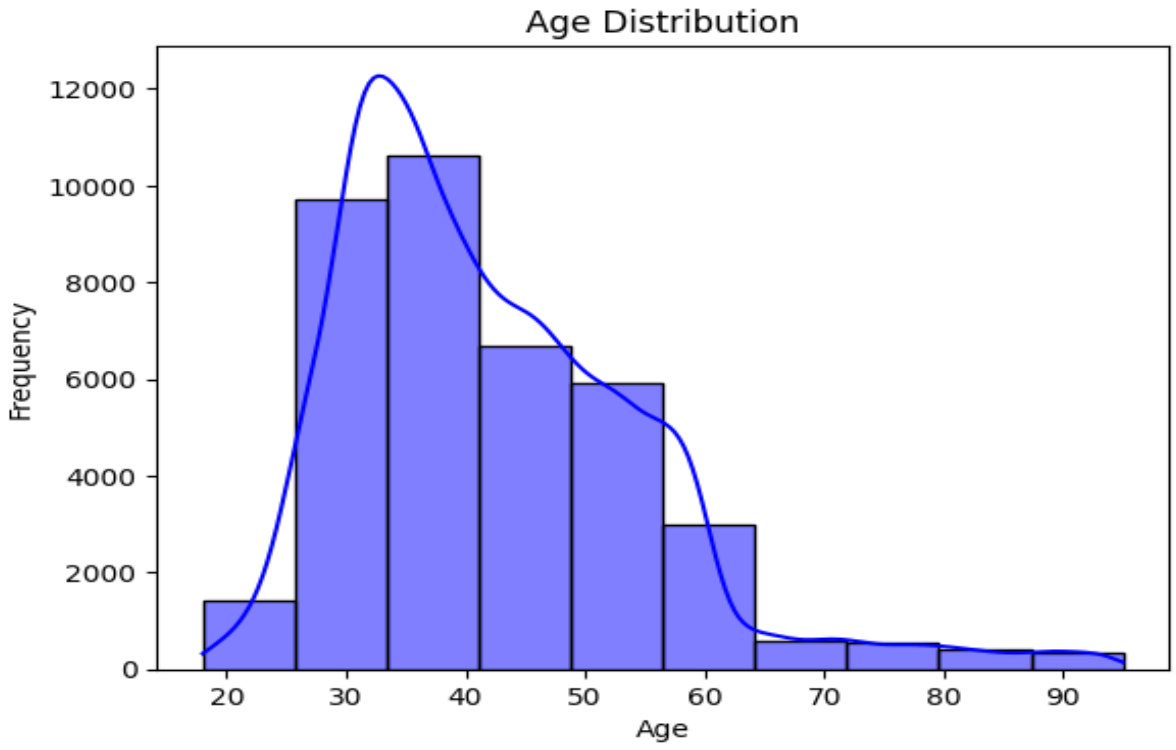
- **Default**: This column indicated whether the client had credit in default. The distribution was:
  - "no": 36,954
  - "yes": 2,257
- **Housing**: This column denoted whether the client had a housing loan. The distribution was:
  - "yes": 21,657
  - "no": 17,554
- **Loan**: This column represented whether the client had a personal loan. The distribution was:
  - "no": 31,820
  - "yes": 7,391

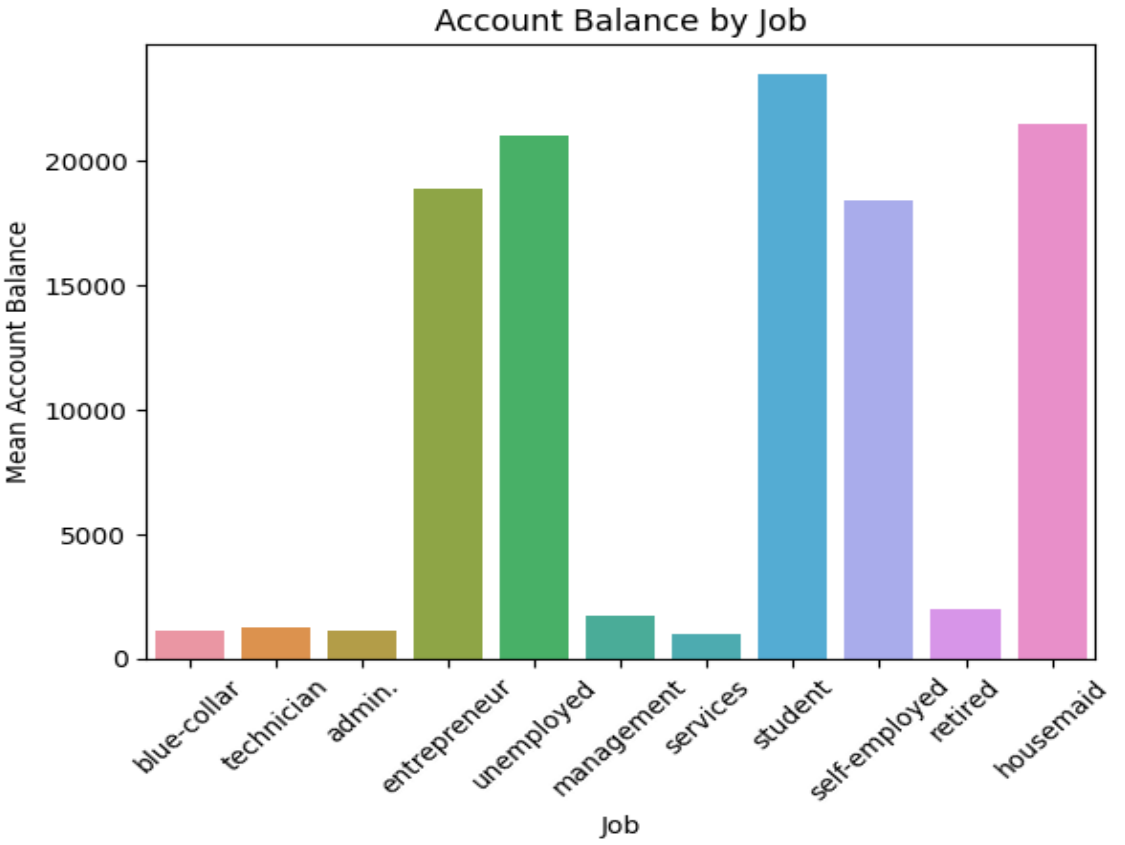
Exploratory Data Analysis (EDA) on Numerical Variables

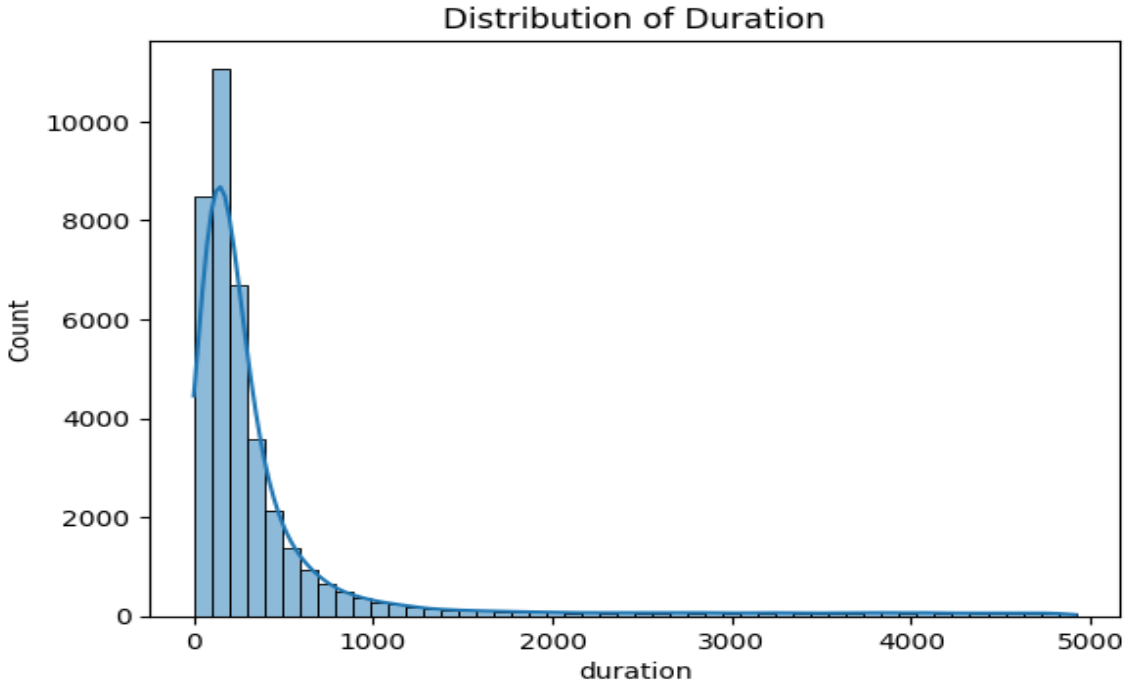
The .describe() method was used to gain a quick numerical summary of each numerical column. The results obtained were as follows:

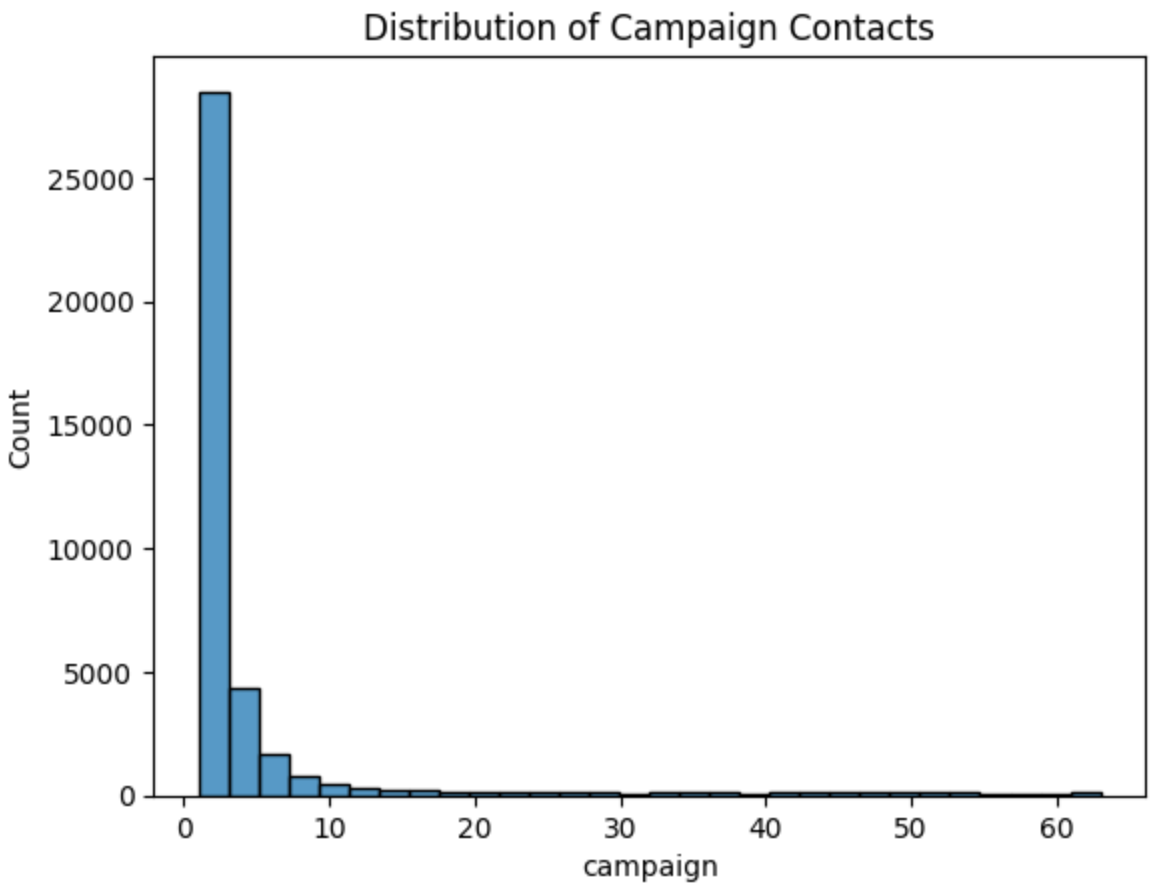
	age	balance	duration	campaign	pdays	previous
count	39211.000000	39211.000000	39211.000000	39211.000000	39211.000000	39211.000000
mean	42.120247	5441.781719	439.062789	5.108770	72.256051	11.826171
std	12.709352	16365.292065	769.096291	9.890153	160.942593	44.140259
min	18.000000	-8019.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	98.000000	109.000000	1.000000	-1.000000	0.000000
50%	40.000000	549.000000	197.000000	2.000000	-1.000000	0.000000
75%	50.000000	2030.000000	380.000000	4.000000	-1.000000	0.000000
max	95.000000	102127.000000	4918.000000	63.000000	871.000000	275.000000

Some Histograms were plotted as follows:




Account Balance by Job
Mean Account Balance
0
5000
10000
15000
20000
blue-collar
technician
admin.
entrepreneur
unemployed
management
services
student
self-employed
retired
housemaid
Job


Distribution of Duration
Count
0
2000
4000
6000
8000
10000
0
1000
2000
3000
4000
5000
duration



Convert ‘Last Contact Date’.

Next the ‘Last Contact Date’ was converted to three separate columns ‘Year’, ‘Month’ and ‘Date’ using pd.to_datetime. This ensures proper handling of date-related operations.

- Extract the Month, Date, and Year from the last contact date and stores them in new columns: Month, Date, and Year.
- The original last contact date column is removed from both datasets using drop, as it is now redundant.
- The newly created Month, Date, and Year columns are cast to object type to treat them as categorical features, which might be more suitable for machine learning models.
- This preprocessing step ensures that date information is retained in a structured format, facilitating further feature engineering and analysis.

Separate Features and Labels

Split Features (X_train) and Target (y_train):

- The target column, which represents the label to be predicted, is separated from the training_data dataset.
- X_train contains all columns except target and serves as the feature set for training. y_train contains only the target column, which is the output variable.

Prepare Test Features (X_test):

The test_data dataset is directly assigned to X_test, as it does not include the target column. This dataset will be used for predictions once the model is trained. This setup establishes the feature matrix (X_train, X_test) and target vector (y_train), which are essential for supervised learning tasks.

The training data is further divided into two subsets: training and validation data. The model is trained on the training subset, while the validation subset is used to evaluate the model's performance, as the labels for the actual test data are unavailable. The model that achieves the best performance on the validation data will be selected for making predictions on the actual test data.

The train_test_split function from sklearn.model_selection is used to perform this split. The test size is set to 0.3, meaning 30% of the data is allocated to validation, and 70% is used for training. This results in two dataframes: X_train with 27,448 records and X_val with 11,763 records. Corresponding target variables are split into y_train and y_val in the same manner.

Data Preprocessing

This section sets up a preprocessing pipeline for handling both categorical and numerical features in a dataset, ensuring the data is ready for model training.

Importing Required Libraries

The necessary modules for creating pipelines and handling preprocessing are imported from sklearn, including Pipeline, ColumnTransformer, and transformers for categorical and numerical data.

Feature Categorization

- categorical_features: Lists categorical columns like job, education, and poutcome.
- numeric_features: Lists numerical columns like age, balance, and duration.
- columns_to_impute: Specifies the subset of categorical columns requiring missing value imputation.

Categorical Feature Pipeline

A pipeline for categorical features is created with two steps:

1. **Imputation**: Missing values are filled using the most frequent value in each column (SimpleImputer with most_frequent strategy).
2. **Encoding**: Categorical features are converted into one-hot encoded format (OneHotEncoder with sparse_output=False).

Numerical Feature Pipeline

A pipeline for numerical features is created with a single step:

1. **Scaling**: Features are standardized to have a mean of 0 and a standard deviation of 1 using StandardScaler.

Combining Pipelines with ColumnTransformer

The ColumnTransformer combines both pipelines:

- The categorical pipeline is applied to the categorical_features.
- The numerical pipeline is applied to the numeric_features.
- Any remaining columns in the dataset are passed through unchanged (remainder='passthrough').

Transforming the Data

- **Training Data (X_train)**: The fit_transform method of the preprocessor fits the transformations to the training data and applies them.

- **Validation Data (X_val)**: The transform method applies the transformations (fit on training data) to the validation data.

Converting Transformed Data to DataFrames

The transformed datasets (X_train_transformed and X_val_transformed) are converted back into pandas DataFrames for compatibility with further analysis or modeling.

This code efficiently preprocesses the dataset, ensuring consistency and readiness for machine learning model training and evaluation.

Model Building

A total of 11 different machine learning models were trained on the training data and evaluated on the validation data. For each model, various F1-scores were recorded, and hyperparameter tuning was performed to optimize their performance. The results obtained from all 11 models are summarized as follows:

1. RidgeClassifier

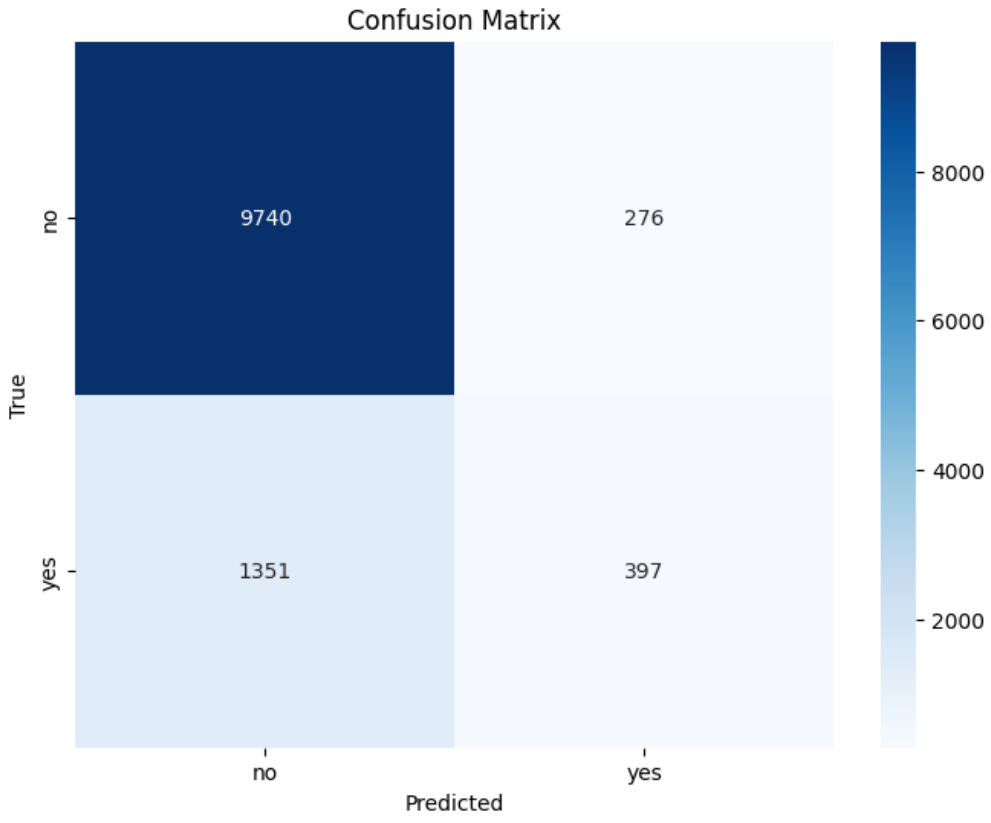
1. **Model Selection**: The Ridge Classifier is chosen as a linear classification model that applies L2 regularization, which helps prevent overfitting by penalizing large coefficients in the model.
2. **Model Initialization**:
    - `alpha=0.1`: Sets the regularization strength. Smaller values allow the model to fit more closely to the data, while larger values add more penalty to large coefficients.
    - `solver='lsqr'`: Specifies the LSQR (Least Squares QR decomposition) solver, which is efficient for handling large datasets.
    - `fit_intercept=True`: Ensures that the model includes an intercept term.
3. **Model Training**:
   The Ridge Classifier is trained using the `fit` method, with the transformed training data (`X_train`) and corresponding labels (`y_train`).
4. **Prediction**:
   After training, the `predict` method is used to make predictions on the validation dataset (`X_val`). The predictions are stored in `y_pred_ridge` for evaluation against the true labels.

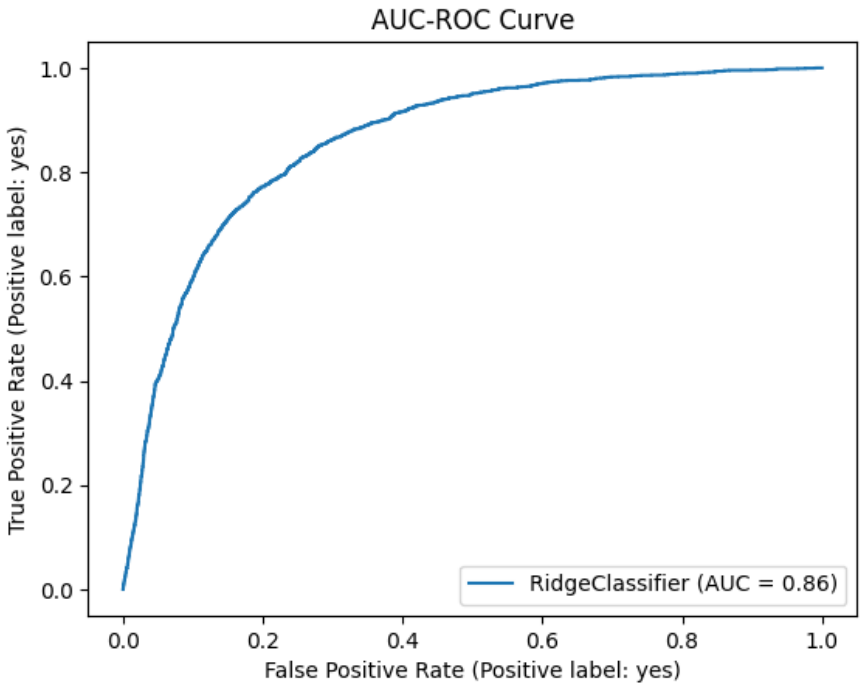
The f1-score obtained was 0.6254. The Results were as follows:

F1 Score: 0.6254

Classification Report:

	precision	recall	f1-score	support
no	0.88	0.97	0.92	10016
yes	0.59	0.23	0.33	1748
accuracy			0.86	11764
macro avg	0.73	0.60	0.63	11764
weighted avg	0.84	0.86	0.83	11764





2. SGDClassifier

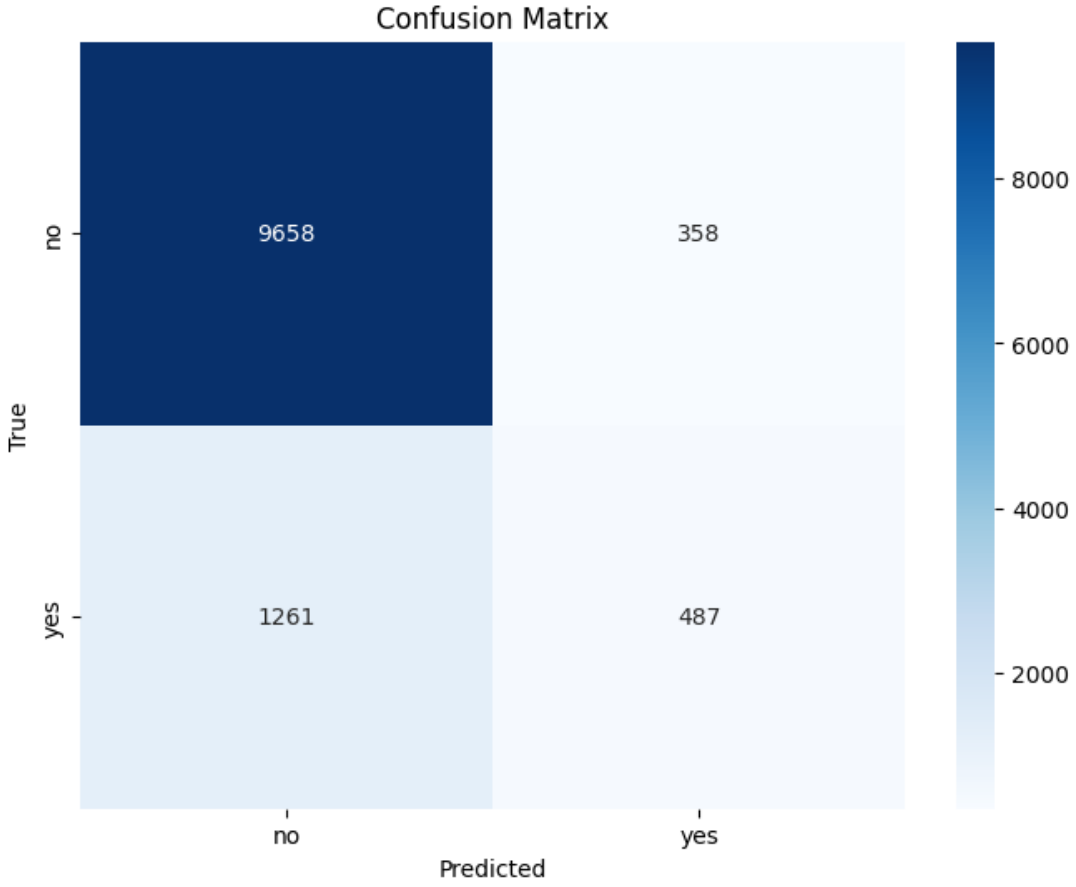
1. **Model Selection**:
   The SGDClassifier is selected as it provides a highly efficient, iterative approach to training linear models, suitable for large datasets. It optimizes the model using stochastic gradient descent, which updates model parameters incrementally for each training example.
2. **Model Initialization**:
   - loss='log_loss': Specifies the loss function to be used. The log_loss function makes the classifier operate as a linear model for logistic regression.
   - random_state=60: Ensures reproducibility by setting a seed for the random number generator.
3. **Model Training**:
   The classifier is trained using the fit method with the preprocessed training data (X_train) and corresponding labels (y_train). This step updates the model parameters iteratively using stochastic gradient descent.
4. **Prediction**:
   After training, the predict method is used to generate predictions on the validation dataset (X_val). The predicted labels are stored in y_pred_sgd for evaluation against the ground truth.

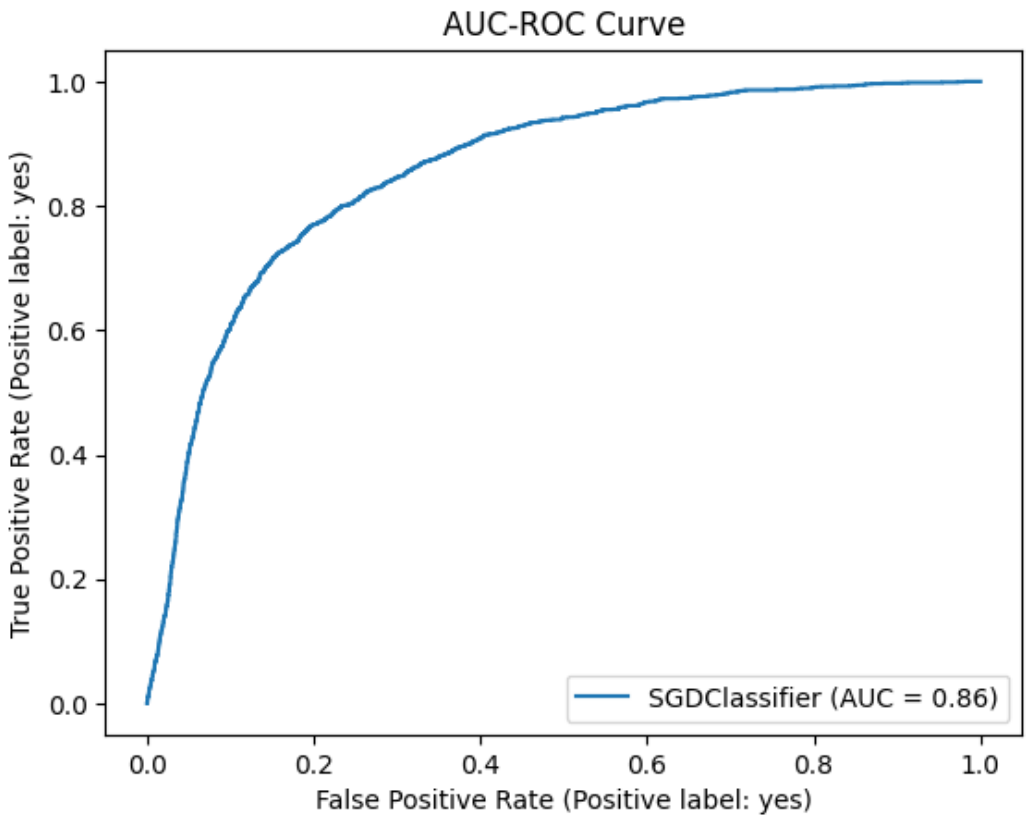
The f1-score obtained was 0.6491. The Results were as follows:

F1 Score: 0.6491

Classification Report:

	precision	recall	f1-score	support
no	0.88	0.96	0.92	10016
yes	0.58	0.28	0.38	1748
accuracy			0.86	11764
macro avg	0.73	0.62	0.65	11764
weighted avg	0.84	0.86	0.84	11764





3. KNeighborsClassifier

1. **Model Selection**:
   The KNeighborsClassifier is selected as a non-parametric model that classifies data points based on the majority class of their nearest neighbors in the feature space. It is particularly effective for datasets where class separation is based on proximity.
2. **Model Initialization**:
   - n_neighbors=5: Specifies that the classifier will consider the 5 nearest neighbors when determining the class of a data point.
   - weights='distance': Ensures that nearer neighbors have a higher influence on the prediction by weighting their contribution based on inverse distance.
   - algorithm='auto': Lets the classifier automatically choose the most appropriate algorithm for computing nearest neighbors based on the dataset.
3. **Model Training**:
   The model is trained using the fit method, which takes the preprocessed

training data (X_train) and their corresponding labels (y_train). This step memorizes the dataset for use during predictions.

4. **Prediction**:
   The predict method is used to classify data points in the validation dataset (X_val). The resulting predictions are stored in y_pred_knn for later evaluation against the actual labels.

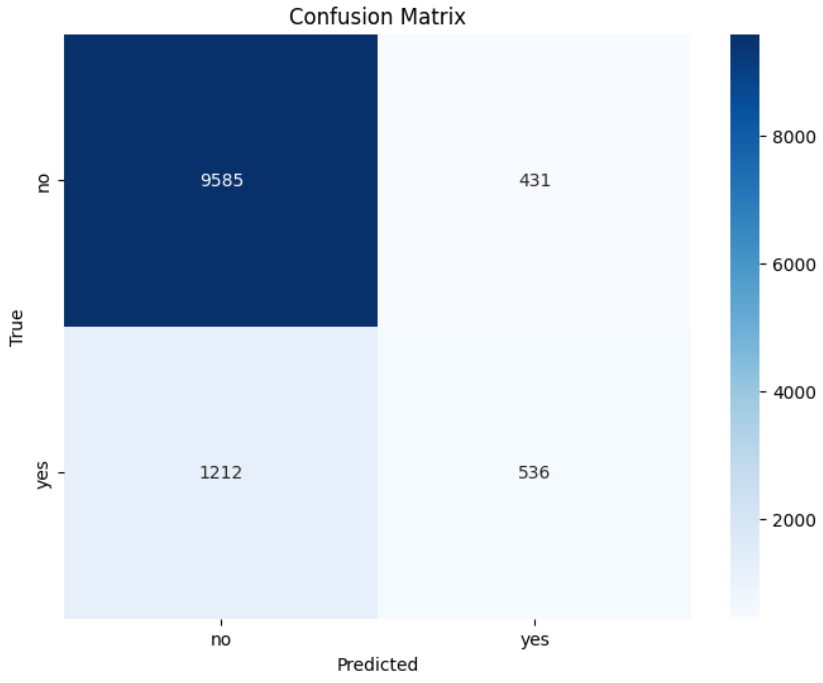
The f1-score obtained was 0.6580. The Results were as follows:

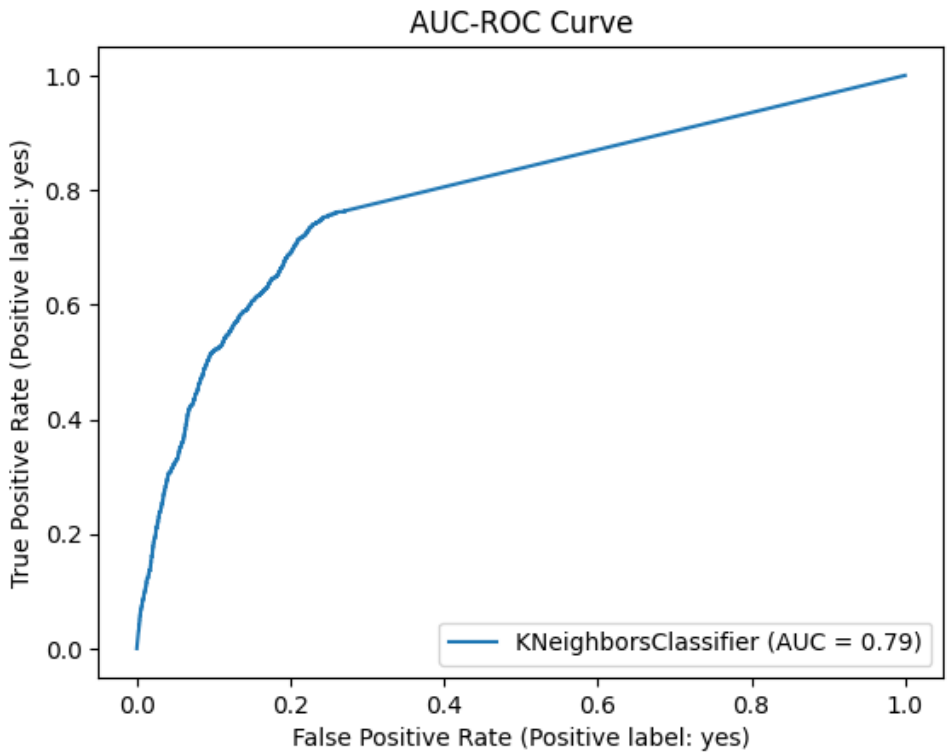
```
F1 Score: 0.6580

Classification Report:
              precision    recall  f1-score   support

          no       0.89      0.96      0.92     10016
         yes       0.55      0.31      0.39      1748

    accuracy                           0.86     11764
   macro avg       0.72      0.63      0.66     11764
weighted avg       0.84      0.86      0.84     11764
```





4. DecisionTreeClassifier

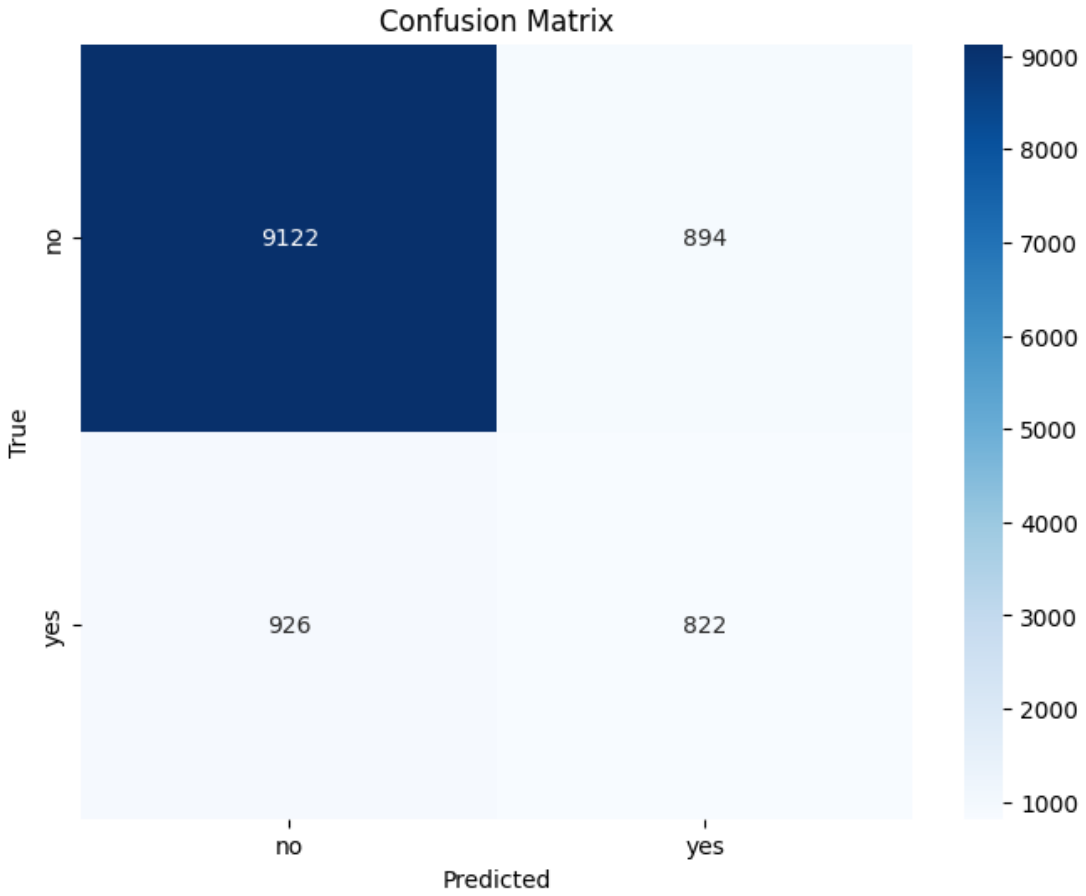
1. **Model Selection**:
   The DecisionTreeClassifier is chosen as a versatile, non-parametric model that predicts the target variable by learning simple decision rules inferred from the data features. It is well-suited for handling both categorical and numerical data.
2. **Model Initialization**:
   - The classifier is initialized with default parameters, allowing the model to automatically determine the best splits based on the data.
3. **Model Training**:
   The fit method is used to train the model on the preprocessed training data (X_train) and their corresponding labels (y_train). The decision tree learns a series of if-else rules to split the data and classify it accurately.
4. **Prediction**:
   After training, the predict method is used to generate predictions for the validation dataset (X_val). The predicted labels are stored in y_pred_dt for evaluation against the true labels.

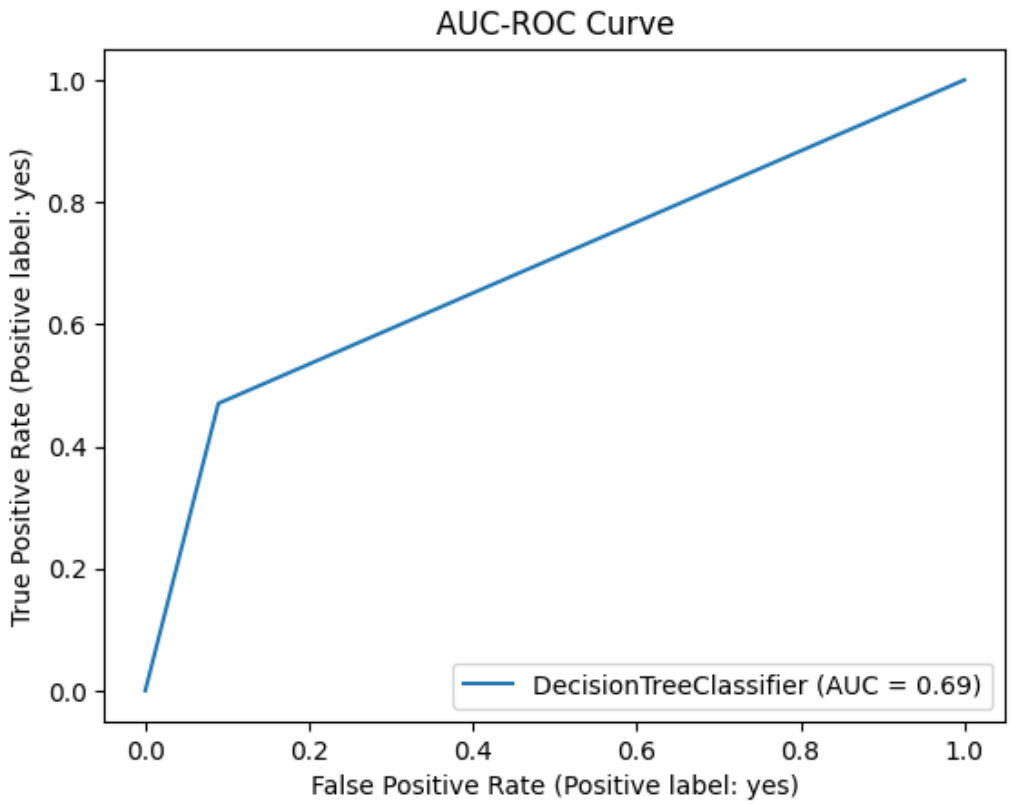
The F1 Score obtained was 0.6919. The results were as follows:

F1 Score: 0.6919

Classification Report:

	precision	recall	f1-score	support
no	0.91	0.91	0.91	10016
yes	0.48	0.47	0.47	1748
accuracy			0.85	11764
macro avg	0.69	0.69	0.69	11764
weighted avg	0.84	0.85	0.84	11764





5. GaussianNB

1. **Model Selection**:
   The GaussianNB classifier is chosen as a probabilistic model based on Bayes' Theorem. It assumes that the features follow a Gaussian (normal) distribution, making it suitable for numerical data.
2. **Model Initialization**:
   - The model is initialized with default parameters, leveraging its simple yet effective approach to classification.
3. **Model Training**:
   The fit method is used to train the model with the preprocessed training data (X_train) and corresponding labels (y_train). The classifier learns the parameters of the Gaussian distribution for each feature and each class.
4. **Prediction**:
   After training, the predict method is used to classify the validation dataset (X_val). The predicted labels are stored in y_pred_gnb for further evaluation against the true labels.

The f1-score obtained was 0.6962. The results were as follows:

```
F1 Score: 0.6962

Classification Report:
              precision    recall  f1-score   support

          no       0.92      0.87      0.90     10016
         yes       0.44      0.57      0.50      1748

    accuracy                           0.83     11764
   macro avg       0.68      0.72      0.70     11764
weighted avg       0.85      0.83      0.84     11764
```

6. Perceptron

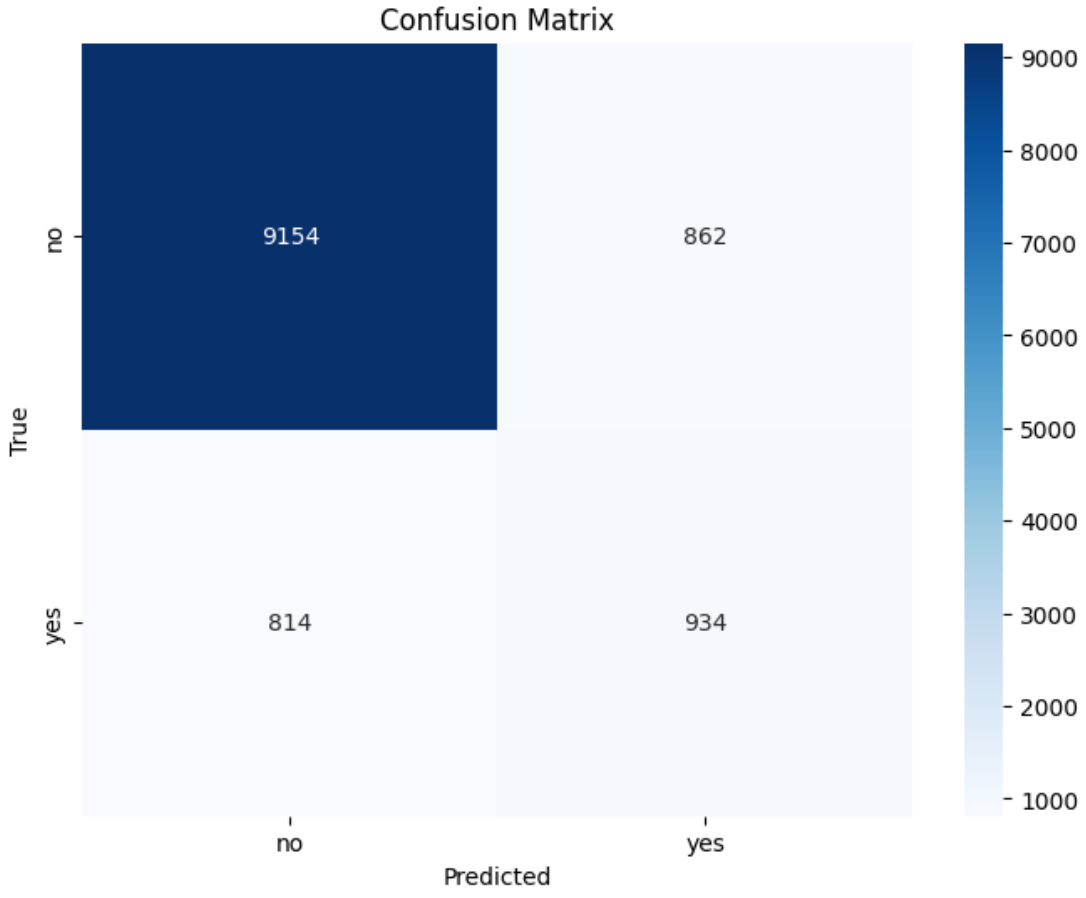
1. **Model Selection**:
   The Perceptron classifier is selected as a linear model that uses a single-layer neural network for binary classification. It applies the perceptron learning algorithm to find a hyperplane that separates the data into distinct classes.
2. **Model Initialization**:
   - The classifier is initialized with default parameters, providing a straightforward implementation of the perceptron algorithm.
3. **Model Training**:
   The fit method is used to train the model on the preprocessed training data (X_train) and their corresponding labels (y_train). The algorithm iteratively adjusts the weights to minimize classification errors.
4. **Prediction**:
   The predict method is used to generate predictions for the validation dataset (X_val). These predictions are stored in y_pred_perceptron for evaluation against the true labels.

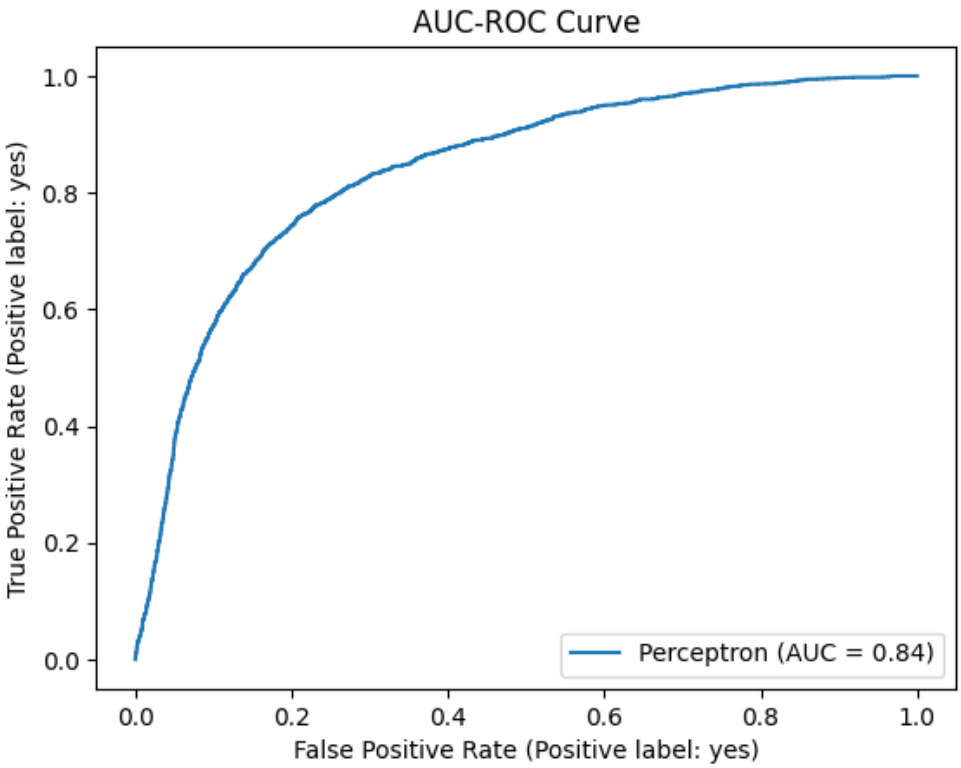
The f1-score obtained was 0.7216. The results were as follows:

F1 Score: 0.7216

Classification Report:

	precision	recall	f1-score	support
no	0.92	0.91	0.92	10016
yes	0.52	0.53	0.53	1748
accuracy			0.86	11764
macro avg	0.72	0.72	0.72	11764
weighted avg	0.86	0.86	0.86	11764





7. Logistic Regression

1. **Model Selection**:
   The LogisticRegression classifier is chosen as a widely used linear model for classification tasks. It predicts probabilities for each class using the logistic function and selects the class with the highest probability.
2. **Model Initialization**:
   - penalty=None: Specifies that no regularization is applied, allowing the model to fit the data without penalizing large coefficients.
   - solver='newton-cg': Uses the Newton-Conjugate Gradient solver, which is efficient for optimizing the logistic regression loss function.
   - class_weight='balanced': Adjusts the weights for each class inversely proportional to their frequencies in the training data, handling imbalanced datasets effectively.
3. **Model Training**:
   The fit method is used to train the model with the preprocessed training data (X_train) and corresponding labels (y_train). The algorithm optimizes the weights to minimize the logistic loss function.
4. **Prediction**:
   The predict method generates predictions for the validation dataset (X_val).

The predicted labels are stored in y_pred_logit for subsequent evaluation against the true labels.

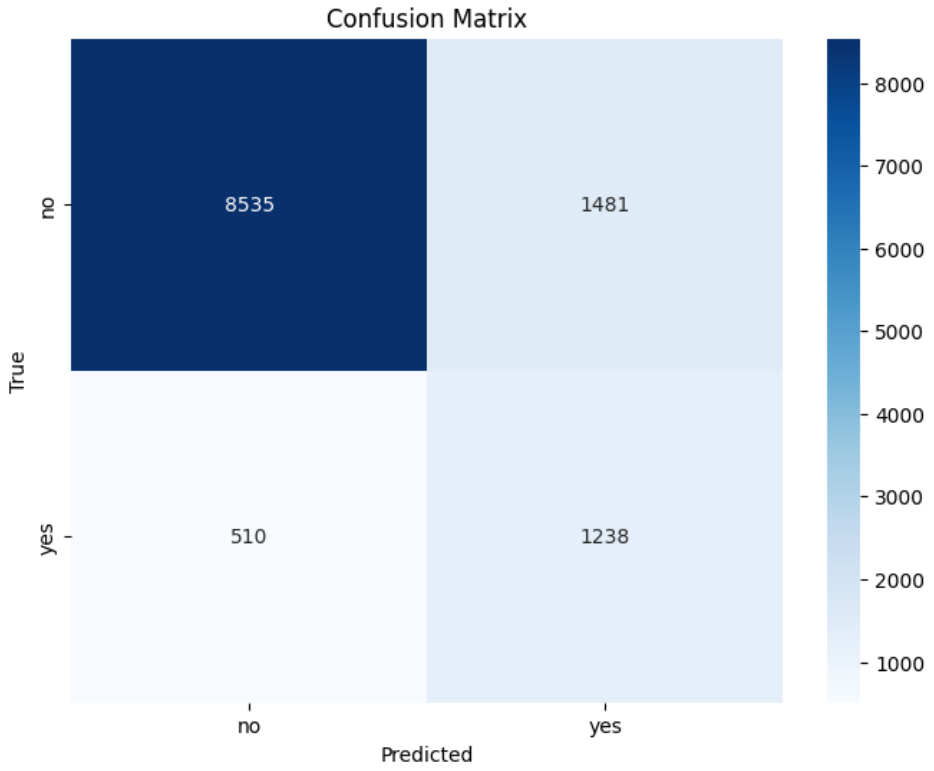
The f1-score obtained was 0.7249. The results were as follows:

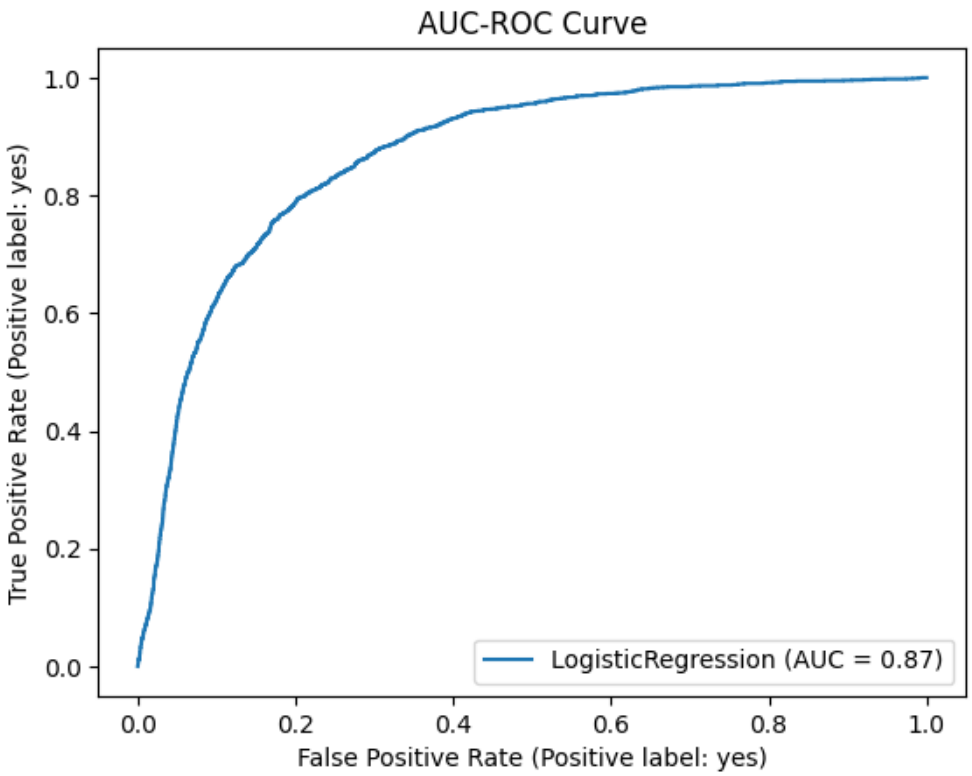
```
F1 Score: 0.7249

Classification Report:
              precision    recall  f1-score   support

          no       0.94      0.85      0.90     10016
         yes       0.46      0.71      0.55      1748

    accuracy                           0.83     11764
   macro avg       0.70      0.78      0.72     11764
weighted avg       0.87      0.83      0.84     11764
```





8. MLPClassifier with RandomizedSearchCV

1. **Model Selection**:
   The MLPClassifier is chosen as a neural network-based model for classification. It learns complex patterns through multiple layers of nodes, making it effective for capturing non-linear relationships in the dataset.
2. **Hyperparameter Tuning**:
   To improve model performance, hyperparameter tuning is performed using RandomizedSearchCV, which samples from a defined set of hyperparameters and selects the best combination based on cross-validation:
   - hidden_layer_sizes: Specifies different configurations for the hidden layers, including single-layer and multi-layer architectures (e.g., (50,), (100,), (50, 50)).
   - activation: Defines the activation function to be used in each layer, with options such as 'identity', 'logistic', 'tanh', and 'relu'.
   - solver: Chooses the optimization algorithm, either 'adam' or 'sgd'.
   - learning_rate_init: Specifies the initial learning rate for training, with values 0.001, 0.01, and 0.1.
   - max_iter: Sets the maximum number of iterations for the solver, with values 200, 500, and 1000.

RandomizedSearchCV performs 20 iterations of random sampling with 3-fold cross-validation and optimizes for the f1_macro score, which balances precision and recall across all classes.

3. **Model Training**:
   The model is trained using the fit method, where the RandomizedSearchCV instance fits the MLP classifier on the training data (X_train) and corresponding labels (y_train). The best hyperparameter combination is identified during this process.
4. **Prediction**:
   After training, the best-performing MLP model (best_mlp) is used to predict the labels for the validation dataset (X_val). The predictions are stored in y_pred_mlp for further evaluation.

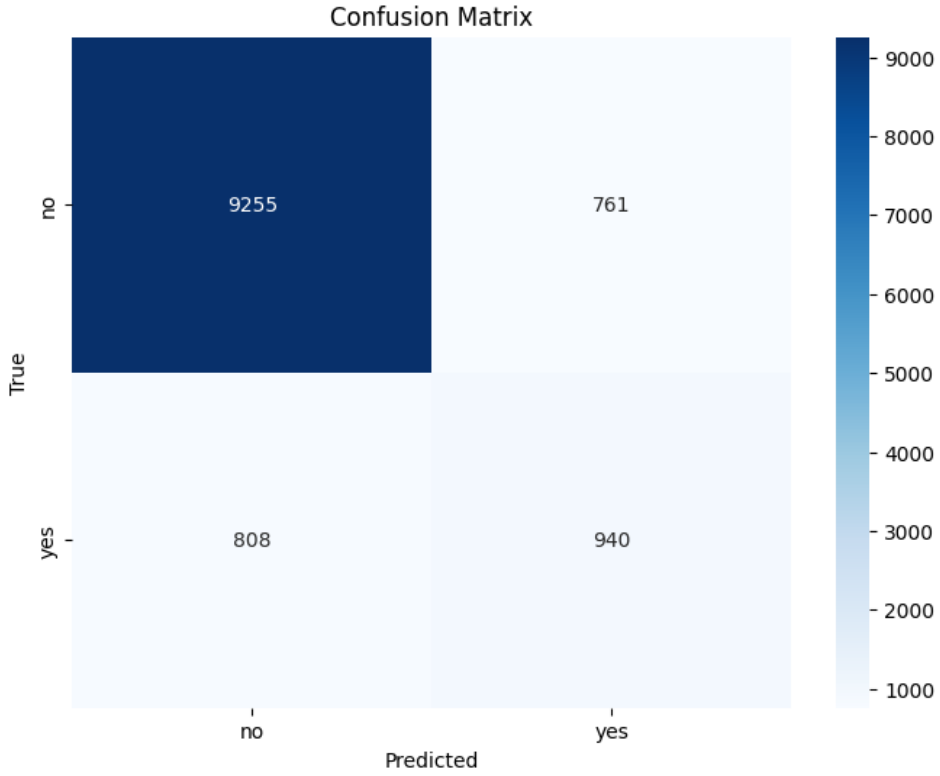
The f1-score obtained was 0.7335. The results were as follows:

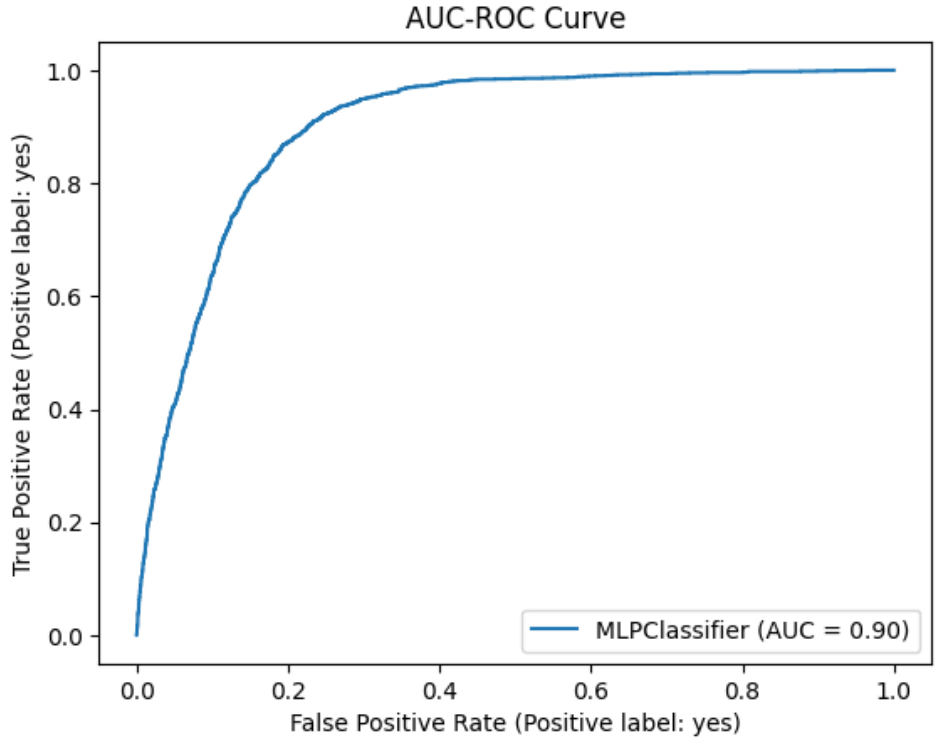
```
F1 Score: 0.7335

Classification Report:
              precision    recall  f1-score   support

          no       0.92      0.92      0.92     10016
         yes       0.55      0.54      0.55      1748

    accuracy                           0.87     11764
   macro avg       0.74      0.73      0.73     11764
weighted avg       0.87      0.87      0.87     11764
```


Confusion Matrix
9255
761
808
940
no
yes
True
no
yes
Predicted
9000
8000
7000
6000
5000
4000
3000
2000
1000


AUC-ROC Curve
True Positive Rate (Positive label: yes)
False Positive Rate (Positive label: yes)
MLPClassifier (AUC = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0

9. SVC with RandomizedSearchCV

1. **Model Selection**:
   The SVC classifier is chosen for its ability to perform well in high-dimensional spaces and its effectiveness in classification tasks. It constructs a hyperplane that maximizes the margin between classes, and is particularly powerful for non-linear decision boundaries.
2. **Hyperparameter Tuning**:
   Hyperparameter tuning is performed using RandomizedSearchCV, which searches through a defined set of hyperparameters to find the best model configuration. The parameters tuned include:
   - C: The regularization parameter, with values 0.1, 1, 10, and 100. Larger values of C allow less regularization, making the model more complex.
   - gamma: The kernel coefficient, which defines the influence of a single training sample. It is tuned with values 'scale', 'auto', 0.1, and 0.01.
   - kernel: The kernel type used to transform the data into a higher-dimensional space. Options include 'linear', 'rbf' (Radial Basis Function), and 'poly' (polynomial kernel).

   RandomizedSearchCV performs 10 iterations of random hyperparameter sampling with 3-fold cross-validation, optimizing for the f1_macro score, which balances precision and recall for multi-class classification.

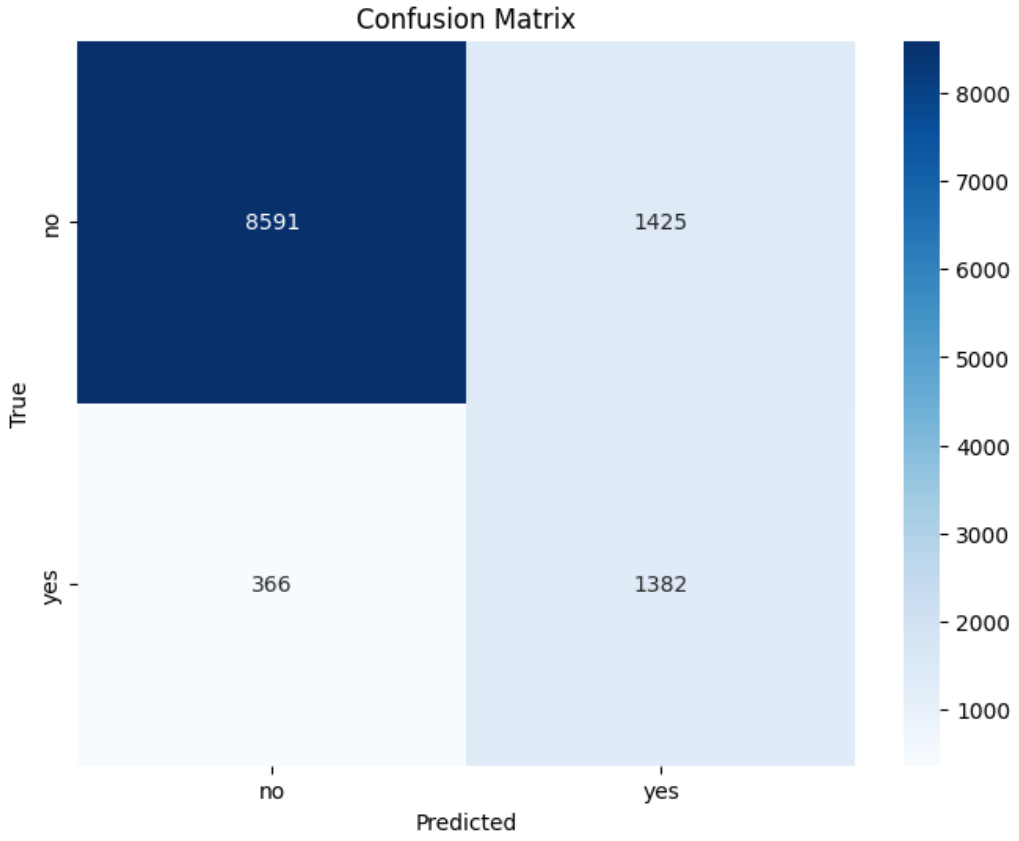
3. **Model Training**:
   The fit method is used to train the model on the preprocessed training data (X_train) and labels (y_train). During training, the RandomizedSearchCV selects the best combination of hyperparameters based on performance.
4. **Prediction**:
   After training, the best-performing model (best_svc) is used to predict the labels for the validation dataset (X_val). The predictions are stored in y_pred_svc for evaluation against the true labels.

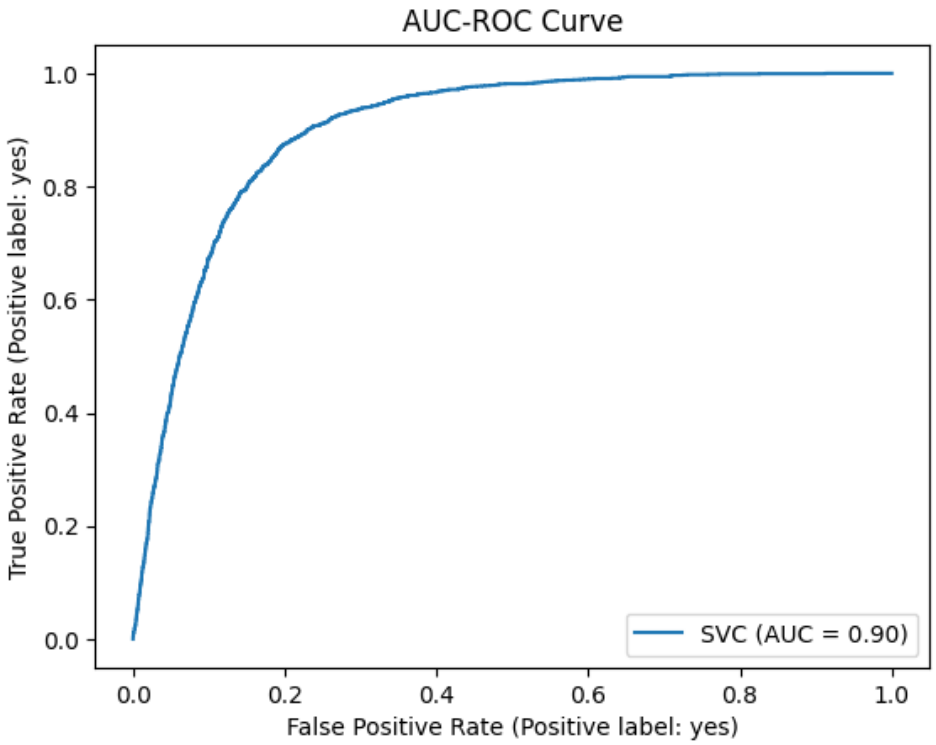
The f1-score obtained was 0.7562. The results were as follows:

F1 Score: 0.7562

Classification Report:

	precision	recall	f1-score	support
no	0.96	0.86	0.91	10016
yes	0.49	0.79	0.61	1748
accuracy			0.85	11764
macro avg	0.73	0.82	0.76	11764
weighted avg	0.89	0.85	0.86	11764





10. XGBClassifier with RandomizedSearchCV

1. **Model Selection**:
   The XGBClassifier is chosen for its powerful gradient boosting capabilities. XGBoost is an ensemble learning algorithm that combines multiple weak learners (decision trees) to form a strong learner, making it effective for complex classification problems, particularly with imbalanced datasets.
2. **Data Transformation**:
   Since the XGBClassifier requires binary labels for classification, the target labels (y_train and y_val) are transformed using LabelBinarizer. The fit_transform method is applied to both training and validation labels, creating a binary matrix representation of the target variable (y_train_xgb and y_val_xgb).
3. **Hyperparameter Tuning**:
   The model is tuned using RandomizedSearchCV, which samples from a wide range of hyperparameters to find the best configuration. The parameters tuned include:
   - n_estimators: The number of boosting rounds (trees), with values 50, 100, 200, and 300.
   - max_depth: The maximum depth of each tree, which helps control overfitting, with values 3, 6, 10, and 15.

- learning_rate: The learning rate that controls the contribution of each tree to the final model, with values 0.01, 0.1, 0.2, and 0.3.
- subsample: The fraction of samples used to train each tree, with values 0.6, 0.8, and 1.0.
- colsample_bytree: The fraction of features used to train each tree, with values 0.6, 0.8, and 1.0.
- min_child_weight: The minimum sum of instance weight needed in a child, with values 1, 3, and 5.
- gamma: The minimum loss reduction required to make a further partition, with values 0, 0.1, 0.2, and 0.5.
- reg_alpha and reg_lambda: Regularization parameters to avoid overfitting, with values 0, 0.1, 1, and 1.5.
- scale_pos_weight: A parameter to help handle class imbalance, set to 1, pos_weight, or 2 * pos_weight (calculated as the ratio of the majority to minority class).

RandomizedSearchCV uses 5-fold stratified cross-validation to evaluate model performance and selects the best combination of hyperparameters based on the f1_macro score, which balances precision and recall across classes.

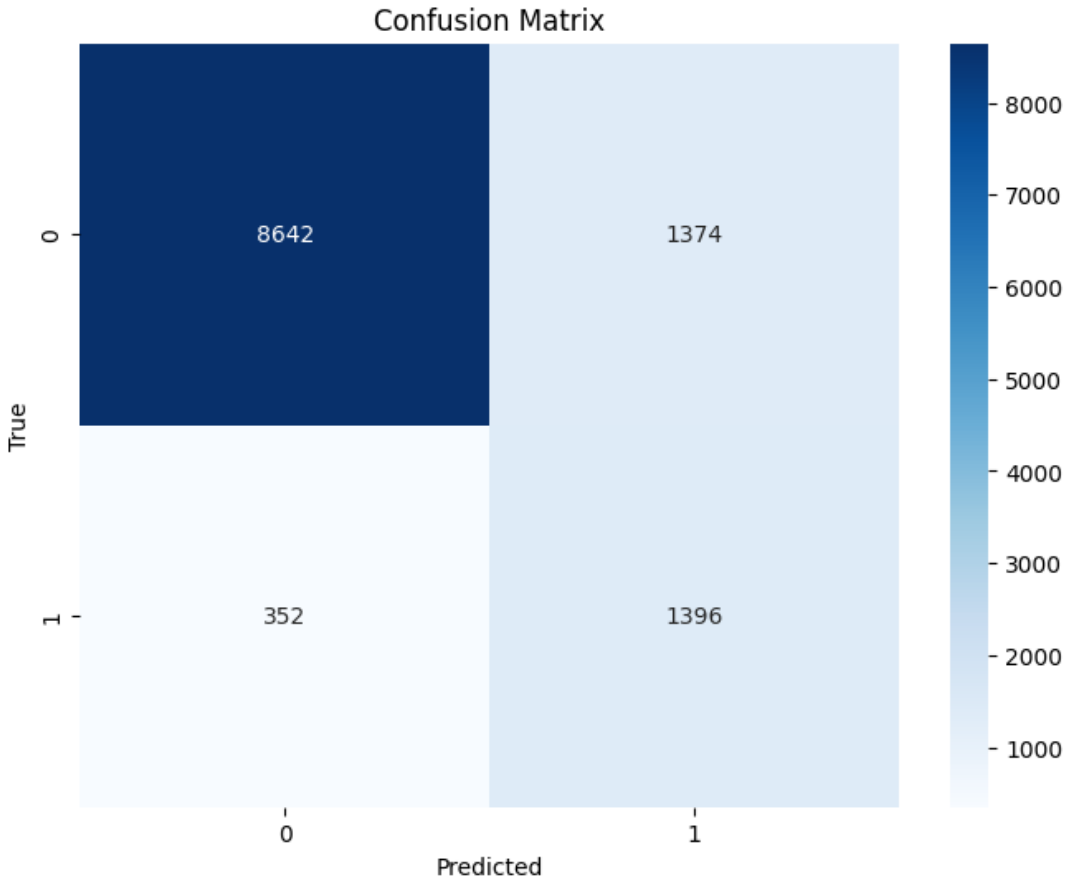
4. **Model Training**:
   The fit method is used to train the model on the preprocessed training data (X_train) and transformed labels (y_train_xgb). The hyperparameter search is performed during this training process.
5. **Prediction**:
   After training, the best-performing XGBoost model (best_xgb) is used to predict the labels for the validation dataset (X_val). The predictions are stored in y_pred_xgb for evaluation.

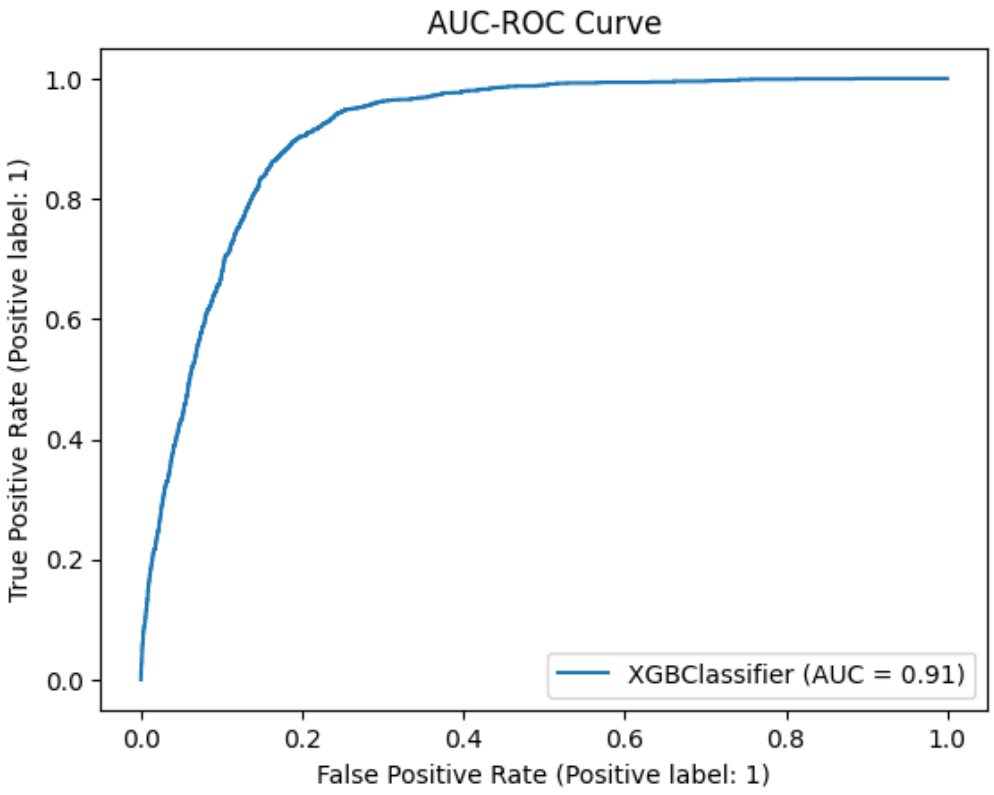
The f1-score obtained was 0.7636. The results were as follows:

F1 Score: 0.7636

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.86	0.91	10016
1	0.50	0.80	0.62	1748
accuracy			0.85	11764
macro avg	0.73	0.83	0.76	11764
weighted avg	0.89	0.85	0.87	11764





11. RandomForestClassifier with RandomizedSearchCV

1. **Model Selection**:
   The RandomForestClassifier is chosen for its robustness, flexibility, and ability to handle both regression and classification tasks effectively. Random forests work by constructing multiple decision trees during training and outputting the class that is the mode of the classes predicted by individual trees. It is particularly well-suited for handling large datasets and capturing complex relationships.
2. **Hyperparameter Tuning**:
   To optimize the performance of the Random Forest model, RandomizedSearchCV is used. This method samples hyperparameters from a specified range and evaluates the model to identify the best configuration. The following parameters are tuned:
   - n_estimators: The number of trees in the forest, with values 50, 100, 200, and 300.
   - max_depth: The maximum depth of each tree, controlling the model's complexity, with options None, 10, 20, and 30.

- min_samples_split: The minimum number of samples required to split an internal node, with values 2, 5, and 10.
- min_samples_leaf: The minimum number of samples required to be at a leaf node, with values 1, 2, and 4.
- max_features: The number of features to consider when looking for the best split, with options 'sqrt', 'log2', and None.
- bootstrap: Whether bootstrap sampling is used when building trees, with values True and False.

RandomizedSearchCV performs 20 iterations of random hyperparameter sampling and evaluates the model using 3-fold cross-validation. The model's performance is optimized for the f1_macro score, which ensures a balanced evaluation across all classes.

3. **Model Training**:
   The fit method is used to train the model on the preprocessed training data (X_train) and corresponding labels (y_train). The hyperparameter search is conducted during this training process to identify the best-performing model.
4. **Prediction**:
   After training, the best model (best_rf) is used to make predictions on the validation dataset (X_val). The predicted labels are stored in y_pred_rf for evaluation.

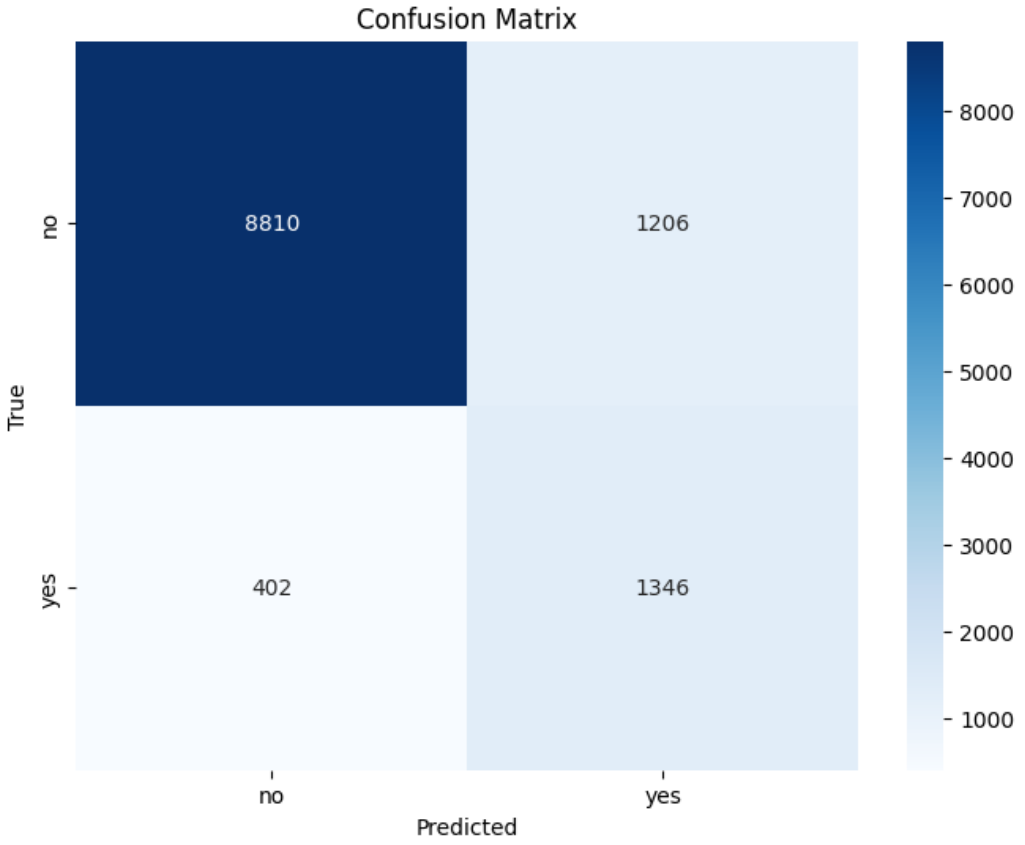
The F1 Score obtained was 0.7712. The results were as follows:

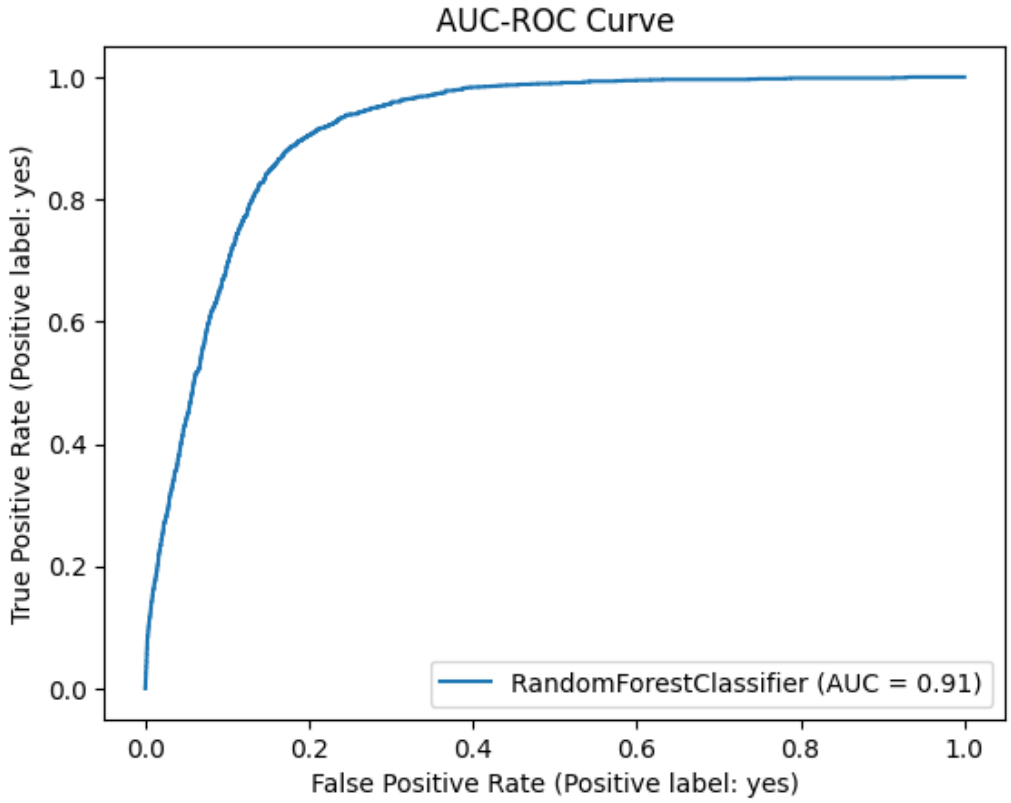
```
F1 Score: 0.7712

Classification Report:
              precision    recall  f1-score   support

          no       0.96      0.88      0.92     10016
         yes       0.53      0.77      0.63      1748

    accuracy                           0.86     11764
   macro avg       0.74      0.82      0.77     11764
weighted avg       0.89      0.86      0.87     11764
```


Confusion Matrix
8810
1206
402
1346
no
yes
True
no
yes
Predicted
8000
7000
6000
5000
4000
3000
2000
1000


AUC-ROC Curve
True Positive Rate (Positive label: yes)
False Positive Rate (Positive label: yes)
RandomForestClassifier (AUC = 0.91)
0.0
0.2
0.4
0.6
0.8
1.0

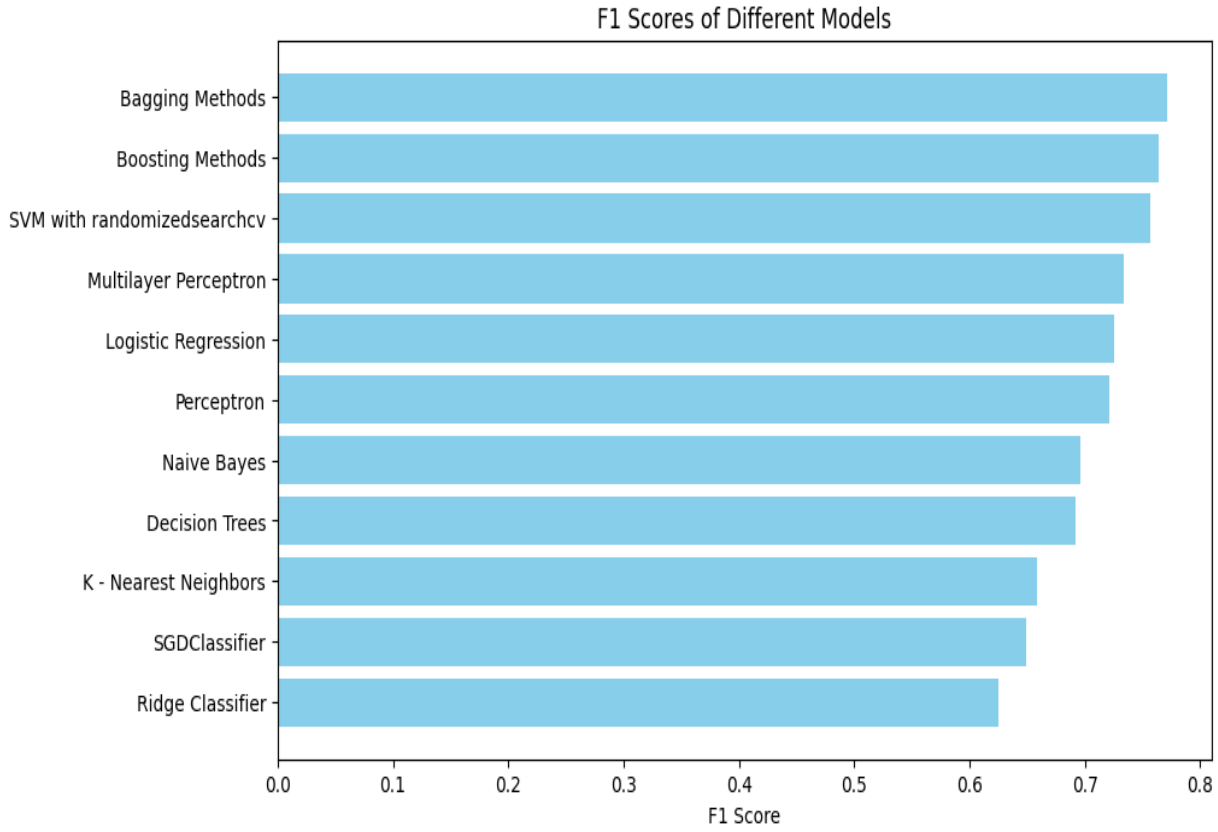
Model Comparison

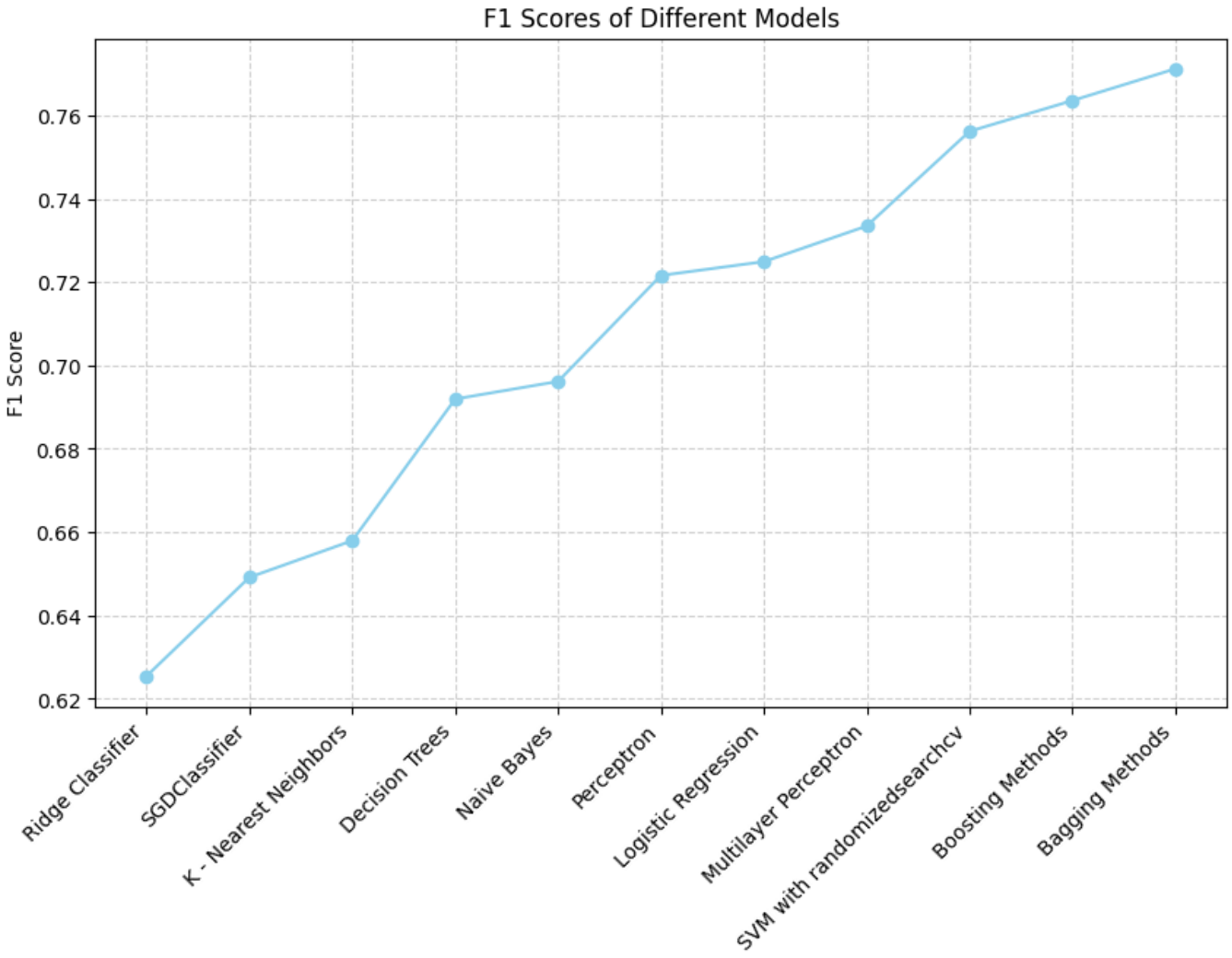
This section visualizes the performance of various machine learning models using their F1 scores. The F1 score is chosen as the evaluation metric to balance precision and recall, particularly for imbalanced datasets. The following steps outline the process:

Using matplotlib, a horizontal bar chart is created to visualize the F1 scores of the models. The chart is configured with:

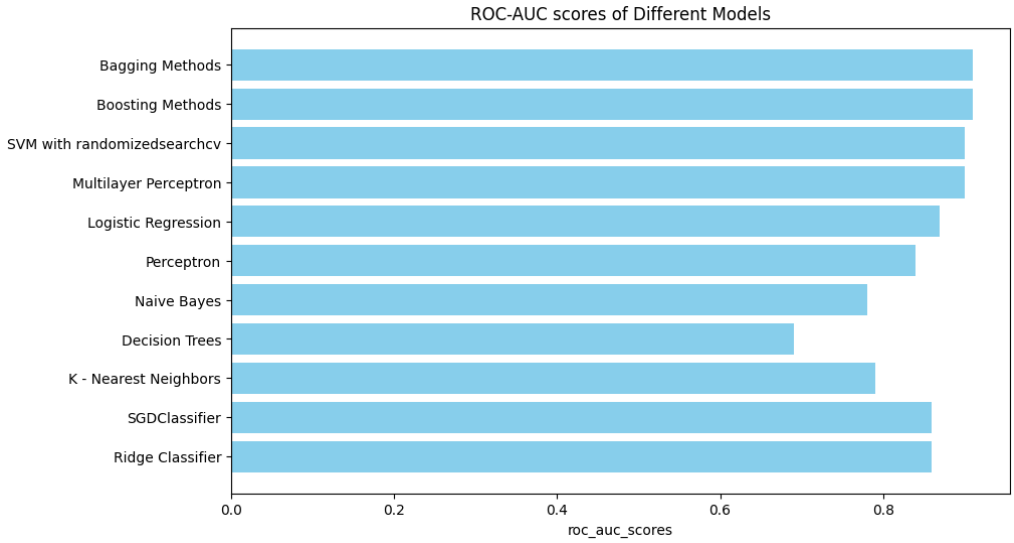
- models on the y-axis, representing each model.
- f1_scores on the x-axis, indicating the performance of each model.
- A sky-blue color is used for the bars to enhance readability.

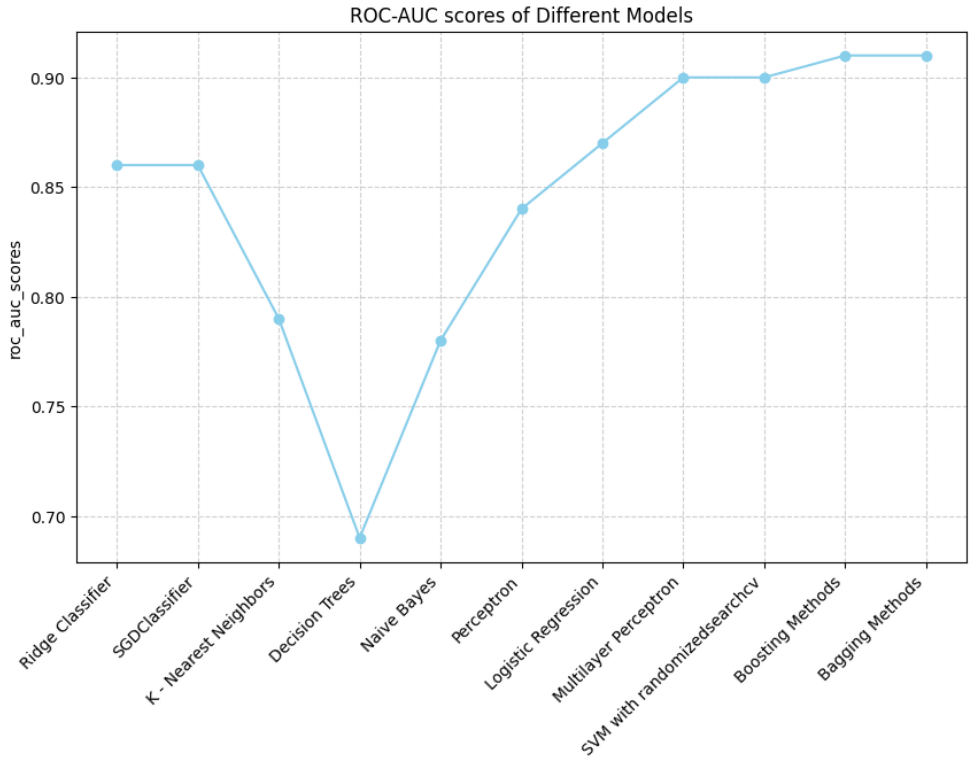
This visualization allows for an easy comparison of how each model performed based on the F1 score, helping to identify the best-performing models for the classification task.





Similar plots can be drawn for the ROC-AUC Scores of models.





Conclusion

After training 11 different models, the RandomForestClassifier with RandomizedSearchCV achieved the highest F1 score on the validation set. To further improve the model's performance, the training and validation datasets were combined, resulting in a total of 39,211 records. The Random Forest model was retrained on this combined dataset, and the best-performing estimator was selected for final predictions. These predictions were then applied to the test dataset, and the results were submitted in a submission.csv file.

The final F1 score obtained on the test data was 0.76921, demonstrating the model's effectiveness in handling the classification task.

The model ranked 54th out of over 1,200 submissions, showcasing its strong performance in comparison to other models in the competition.

References

1. **Scikit-learn Documentation**
   Scikit-learn is a popular machine learning library in Python used for implementing various algorithms and techniques such as classification, regression, and model evaluation.
   - Scikit-learn Documentation. (n.d.). *Scikit-learn*. Retrieved from https://scikit-learn.org/stable/
2. **Random Forest Algorithm**
   For an overview of the Random Forest algorithm, which is widely used in machine learning for classification tasks.
   - Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324
3. **XGBoost Algorithm**
   XGBoost is a highly efficient implementation of gradient boosting that is used for classification and regression problems.
   - Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM. https://doi.org/10.1145/2939672.2939785
4. **Deep Learning with Neural Networks**
   For the implementation of neural networks and MLP (Multilayer Perceptron) classifiers, which are common in complex classification tasks.
   - Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. Retrieved from https://www.deeplearningbook.org/
5. **Google Colab**
   Google Colab is a popular platform for running machine learning experiments using cloud-based resources.
   - Google Colab. (n.d.). *Google Colaboratory*. Retrieved from https://colab.research.google.com/
6. **ChatGPT (OpenAI)**
   A reference to ChatGPT, which can be cited for assistance with code explanations, model suggestions, and general knowledge in machine learning.
   - OpenAI. (2023). *ChatGPT*. Retrieved from https://chat.openai.com/
7. **Google**