National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Snovy Lopes
Student ID: 23242221

School of Computing
National College of Ireland

Supervisor: Cristina Hava Muntean

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Snovy Lopes
Student ID:	23242221
Programme:	MSc in Data Analytics **Year:** 2024-25
Module:	MSc Research Project
Lecturer:	Cristina Hava Muntean
Submission Due Date:	12 December 2024
Project Title:	Advanced Resampling Techniques and Ensemble Methods for Improved Detection of Imbalanced Medicare Fraud Cases

Word Count: 653 words **Page Count:** 8 pages

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Snovy Lopes

Date: 12-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Snovy Lopes
Student ID: 23242221

1 Introduction

The configuration manual provides a organised way for handling, monitoring and maintaining key components and outcomes related to the project "Advanced Resampling Techniques and Ensemble Methods for Improved Detection of Imbalanced Medicare Fraud Cases". This manual discusses the system specification that is the software and the hardware used for implementing the project, techniques, dataset, tools, etc that ensures that the project objectives are met. The coding for this research is conducted in jupyter notebook.

2 System Configuration

2.1 Software Specification

- Jupyter Notebook: This open source application was used to clean, preprocess, handle class imbalance, fit the model and evaluate it.
- A google drive account is used to upload the video presentation.

2.2 Hardware Specification

- Dell 13th Gen Intel(R) Core (TM), i5-1335U, 1.30 GHz 16 GB Ram

3 Methodology

- STEP 1: Import all the libraries.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek, SMOTEENN
from imblearn.under_sampling import EditedNearestNeighbours
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

- STEP 2: Load the downloaded dataset (Medicare Part D)

```
df = pd.read_csv(r'C:\Users\snovy\Downloads\Medicare Part D Prescribers - by Provider and Drug\Medicare Part D Prescribers - by P
```

```
leie_df = pd.read_csv(r'Downloads\UPDATED.csv')
```

- STEP 3: Load the LEIE dataset and integrate required columns to the Part D dataset to help label the providers as fraud or non-frauds.

```
leie_df = leie_df.rename(columns={'NPI': 'Prscrbr_NPI'})
```

```
leie_df['Exclusion_End_Year'] = pd.to_datetime(leie_df['EXCLDATE'], format='%Y%m%d').dt.year
```

```
merged_df = pd.merge(df, leie_df[['Prscrbr_NPI', 'Exclusion_End_Year']], on='Prscrbr_NPI', how='left')
```

```
# Obtain a subset from the original dataset
MS_df = merged_df[merged_df['Prscrbr_State_Abrvtn'] == 'MS']

# Display the first few rows of the MS subset
MS_df.head(10)
```

- STEP 4: Create a subset of the original dataset based on a particular state called ‘MS’. The dataset is now called MS_df.

- STEP 5: Clean the data by handling the missing values and removing the unnecessary variables.

- STEP 6: Label the providers that are excluded from participating in Medicare program the year after the active year as fraud.

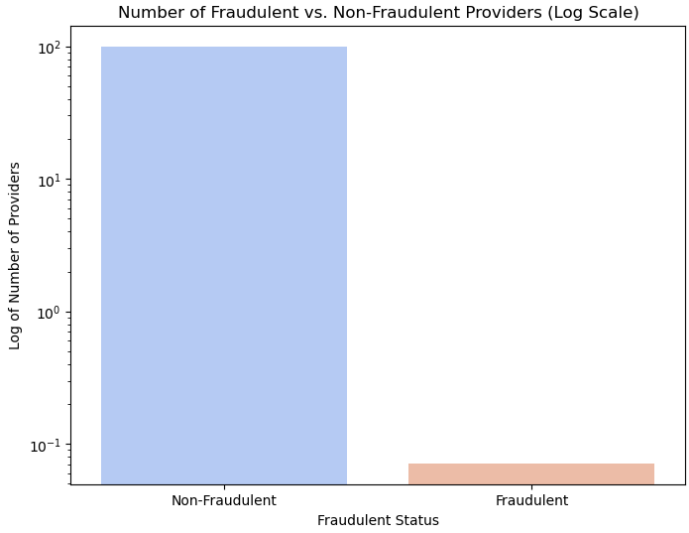
```
# Flag providers as fraudulent if they were excluded in 2023, with an active year of 2022
MS_df['Fraudulent'] = MS_df.apply(lambda row: 1 if row['Year'] == 2022 and row['Exclusion_End_Year'] == 2023 else 0, axis=1)
```

```
MS_df.drop("GE65_Tot_30day_Fills",axis=1,inplace=True)
MS_df.drop("GE65_Tot_Drug_Cst",axis=1,inplace=True)
MS_df.drop("GE65_Tot_Day_Suply",axis=1,inplace=True)
MS_df.drop("GE65_Sprsn_Flag",axis=1,inplace=True)
MS_df.drop("GE65_Bene_Sprsn_Flag",axis=1,inplace=True)
MS_df.drop("GE65_Tot_Clms",axis=1,inplace=True)
MS_df.drop("GE65_Tot_Benes",axis=1,inplace=True)
MS_df.drop("Tot_Benes",axis=1,inplace=True)
```

STEP 5: Removing unnecessary columns

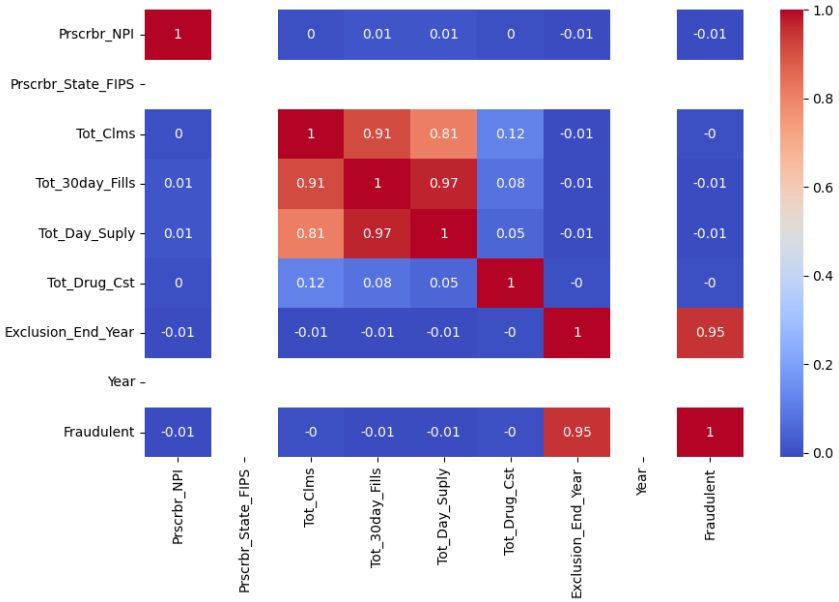
```
# Calculate percentages
total_providers = len(MS_df)
fraud_percent = (class_percentages[1] / total_providers) * 100
non_fraud_percent = (class_percentages[0] / total_providers) * 100

# Display results
print(f"Fraudulent Providers: {class_percentages[1]} ({fraud_percent:.6f}%)")
print(f"Non-Fraudulent Providers: {class_percentages[0]} ({non_fraud_percent:.6f}%)")
```

```
Fraudulent Providers: 0.07147301060615038 (0.000026%)
Non-Fraudulent Providers: 99.92852698939384 (0.037006%)
```



PERCENTAGE DISTRIBUTION

- STEP 7: The percentage of the fraud and non-fraud claims are counted and visualized.
- STEP 8: Correlation matrix: This matrix is obtained to check for multicollinearity between the features. The scale ranges from -1 (negative correlation) to 1 (positive correlation). The highly correlated variables are dropped.



CORRELATION MATRIX

- STEP 8: Encode the categorical features using the label encoder from the scikit library and test the features using the Chi-square test.

```
Chi-Square Feature Scores:
```

	Feature	Chi2 Score
8	Exclusion_End_Year	3.508663e+08
7	Tot_Drug_Cst	3.734618e+05
6	Tot_Day_Suply	7.508178e+04
5	Tot_30day_Fills	2.829834e+03
0	Prscrbr_City	1.423673e+03
4	Tot_Clms	1.200174e+03
1	Prscrbr_Type	8.669963e+02
3	Gnrc_Name	1.578926e+01
2	**Brnd_Name**	**5.415881e+00**

CLASS IMBALANCE

- STEP 9: To better understand the class imbalance, the dataset goes under RUS (RandomUnderSampler) to generate three undersampled datasets of varying ratios (1% , 5%, 20%).

In [76]:
```python
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek, SMOTEENN
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import EditedNearestNeighbours
import pandas as pd

# Step 1: Split data into training and testing sets
X = data.drop(columns=['Fraudulent'])
y = data['Fraudulent']

# Ensure the test set remains untouched
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Define dataset characteristics
sampling_ratios = {
    "Part D Full": None,  # No resampling
    "RUS-1": 0.01,        # 1% minority of the majority
    "RUS-5": 0.05,        # 5% minority of the majority
    "RUS-20": 0.2         # 20% minority of the majority
}
```

```python
# Step 2: Undersample training data
undersampled_datasets = {}

for dataset_name, ratio in sampling_ratios.items():
    if ratio is None:  # No resampling for "Part D Full"
        undersampled_datasets[dataset_name] = pd.concat(
            [pd.DataFrame(X_train, columns=X.columns), pd.DataFrame(y_train, columns=['Fraudulent'])], axis=1
        )
    else:
        rus = RandomUnderSampler(sampling_strategy=ratio, random_state=42)
        X_resampled, y_resampled = rus.fit_resample(X_train, y_train)
        undersampled_datasets[dataset_name] = pd.concat(
            [pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['Fraudulent'])], axis=1
        )
    # Save undersampled training dataset
    undersampled_datasets[dataset_name].to_csv(
        f"C:/Users/snovy/Desktop/THESIS/Solution Artefact/final_dfs/{dataset_name}_undersampled.csv", index=False
    )
    print(f"{dataset_name} undersampled dataset saved. Size: {undersampled_datasets[dataset_name].shape}")
```

To avoid any leakage of training dataset to test dataset, the sampling method are applied only to the train dataset and the test dataset remains untouched which is used during model training to make predictions.

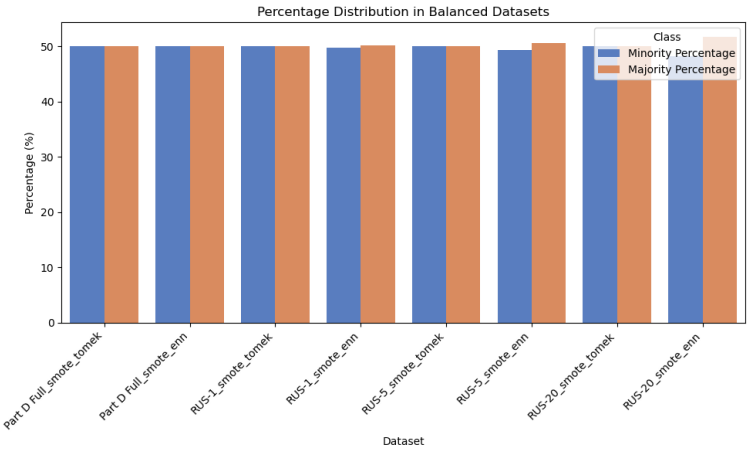
```python
# Step 3: Apply SMOTE-Tomek and SMOTE-ENN to training data
def balance_dataset(dataset, feature_columns, target_column, method, smote_k=3, enn_k=1, random_state=42):
    if method == 'smote_tomek':
        balancer = SMOTETomek(random_state=random_state)
    elif method == 'smote_enn':
        smote = SMOTE(random_state=random_state, k_neighbors=smote_k)
        enn = EditedNearestNeighbours(n_neighbors=enn_k)
        balancer = SMOTEENN(smote=smote, enn=enn)
    else:
        raise ValueError("Method must be 'smote_tomek' or 'smote_enn'")

    X = dataset[feature_columns]
    y = dataset[target_column]
    X_balanced, y_balanced = balancer.fit_resample(X, y)
    return pd.concat([pd.DataFrame(X_balanced, columns=feature_columns), pd.DataFrame(y_balanced, columns=[target_column])], axis

balanced_datasets = {}
for name, dataset in undersampled_datasets.items():
    for method in ['smote_tomek', 'smote_enn']:
        balanced_name = f"{name}_{method}"
        print(f"Balancing {name} dataset using {method.upper()}...")
        # Adjust SMOTE-ENN parameters for better balance
        if method == 'smote_enn':
            balanced_datasets[balanced_name] = balance_dataset(
                dataset, X.columns, 'Fraudulent',
                method=method, smote_k=10, enn_k=8)
        else:
            balanced_datasets[balanced_name] = balance_dataset(
                dataset, X.columns, 'Fraudulent', method=method)
        # Save balanced training dataset
        balanced_datasets[balanced_name].to_csv(
            f"C:/Users/snovy/Desktop/THESIS/Solution Artefact/final_dfs/{balanced_name}.csv", index=False
```

- STEP 10: Next we apply the hybrid resampling methods to the undersampled train sets to balance them.

```python
# Step 3: Apply both SMOTE-Tomek and SMOTE-ENN to each dataset and save results
balanced_datasets = {}

for name, dataset in undersampled_datasets.items():
    for method in ['smote_tomek', 'smote_enn']:
        balanced_name = f"{name}_{method}"
        print(f"Balancing {name} dataset using {method.upper()}...")
        # Adjust SMOTE-ENN parameters for better balance
        if method == 'smote_enn':
            balanced_datasets[balanced_name] = balance_dataset(
                dataset, X.columns, 'Fraudulent',
                method=method, smote_k=10, enn_k=8
            )
        else:
            balanced_datasets[balanced_name] = balance_dataset(
                dataset, X.columns, 'Fraudulent', method=method
            )
        # Save balanced dataset
        balanced_datasets[balanced_name] = balance_dataset(dataset, X.columns, 'Fraudulent', method=method)
        balanced_datasets[balanced_name].to_csv(f"C:/Users/snovy/Desktop/THESIS/final_dfs/{balanced_name}.csv", index=False)
        print(f"{balanced_name} balanced dataset saved. Size: {balanced_datasets[balanced_name].shape}")
        print(f"Class distribution:\n{balanced_datasets[balanced_name]['Fraudulent'].value_counts()}")
        print("-" * 50)
```

Dataset	Minority Count	Majority Count	Total Count	Minority Imbalance
Part D Full	135	188887	189022	0.07%
RUS-1	135	13500	13635	0.99%
RUS-5	135	2700	2835	4.76%

RUS-20	135	675	810	16.67%



MODEL DEVELOPMENT AND EVALUATION

- STEP 11: Train both Random Forest and XGBoost on the balanced training datasets. Fit the model and then make predictions using the original test dataset.

```
# Function to train and evaluate Random Forest
def train_random_forest(X_train, X_test, y_train, y_test):
    # Train the Random Forest model
    rf_model = RandomForestClassifier(class_weight='balanced', random_state=42)
    rf_model.fit(X_train, y_train)

    # Predictions and evaluation
    y_pred = rf_model.predict(X_test)
    metrics = {
        "confusion_matrix": confusion_matrix(y_test, y_pred),
        "classification_report": classification_report(y_test, y_pred, zero_division=0),
        "accuracy_score": accuracy_score(y_test, y_pred),
    }

    return {"model": rf_model, "metrics": metrics}
```

```
# Train and evaluate Random Forest for all balanced datasets
rf_results = {}

for name, dataset in balanced_datasets.items():
    print(f"Training Random Forest on {name} dataset...")

    # Extract the training set from the balanced dataset
    X_balanced_train = dataset[X_train.columns]
    y_balanced_train = dataset['Fraudulent']

    # Ensure test features match training features
    X_test_filtered = X_test[X_balanced_train.columns]

    # Train and evaluate the model
    rf_results[name] = train_random_forest(X_balanced_train, X_test_filtered, y_balanced_train, y_test)
```

```python
# Function to train and evaluate XGBoost
def train_xgboost(X_train, X_test, y_train, y_test, random_state=42):
    # Train the XGBoost model
    xgb_model = XGBClassifier(
        use_label_encoder=False,
        eval_metric='logloss',
        scale_pos_weight=len(y_train[y_train == 0]) / len(y_train[y_train == 1]),  # Handle class imbalance
        random_state=random_state
    )
    xgb_model.fit(X_train, y_train)

    # Predictions and evaluation
    y_pred = xgb_model.predict(X_test)
    metrics = {
        "confusion_matrix": confusion_matrix(y_test, y_pred),
        "classification_report": classification_report(y_test, y_pred, zero_division=0),
        "accuracy_score": accuracy_score(y_test, y_pred),
    }

    return {"model": xgb_model, "metrics": metrics}
```

```python
# Train XGBoost using pre-split test data
xgb_results = {}

for name, dataset in balanced_datasets.items():
    print(f"Training XGBoost on {name} dataset...")

    # Extract the training set from the balanced dataset
    X_balanced_train = dataset[X_train.columns]
    y_balanced_train = dataset['Fraudulent']

    # Use the untouched original test set
    X_test_filtered = X_test[X_balanced_train.columns]

    # Train and evaluate the model
    xgb_results[name] = train_xgboost(X_balanced_train, X_test_filtered, y_balanced_train, y_test)
```

```python
# Define a function to plot confusion matrix
def plot_confusion_matrix(cm, dataset_name):
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, annot_kws={"size": 14})
    plt.title(f'Confusion Matrix - {dataset_name}', fontsize=14)
    plt.xlabel('Predicted', fontsize=12)
    plt.ylabel('Actual', fontsize=12)
    plt.xticks(ticks=[0.5, 1.5], labels=['Non-Fraudulent', 'Fraudulent'], fontsize=10)
    plt.yticks(ticks=[0.5, 1.5], labels=['Non-Fraudulent', 'Fraudulent'], fontsize=10, rotation=0)
    plt.tight_layout()
    plt.show()

# Feature and target column names
feature_columns = [
    'Prscrbr_City', 'Prscrbr_Type', 'Tot_Clms',
    'Tot_Day_Suply', 'Tot_Drug_Cst',
]
target_column = 'Fraudulent'
```

- STEP 12: obtain the confusion matrix and classification reports for each of the datasets to evaluate the models. Given below are the results of the models

Dataset	Accuracy	Precision (0)	Recall (0)	F1-Score (0)	Precision (1)	Recall (1)	F1-Score (1)
Part D Full (SMOTE-Tomek)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Part D Full (SMOTE-ENN)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
RUS-1 (SMOTE-Tomek)	0.9999	1.00	1.00	1.00	0.92	0.98	0.95
RUS-1 (SMOTE-ENN)	0.9999	1.00	1.00	1.00	0.92	0.98	0.95
RUS-5 (SMOTE-Tomek)	0.9999	1.00	1.00	1.00	0.87	1.00	0.93
RUS-5 (SMOTE-ENN)	0.9999	1.00	1.00	1.00	0.87	1.00	0.93
RUS-20 (SMOTE-Tomek)	0.9999	1.00	1.00	1.00	0.87	1.00	0.93
RUS-20 (SMOTE-ENN)	0.9999	1.00	1.00	1.00	0.87	1.00	0.93

RESULTS OF XGBOOST CLASSIFIER

Dataset	Accuracy	Precision (0)	Recall (0)	F1-Score (0)	Precision (1)	Recall (1)	F1-Score (1)
Part D Full (SMOTE-Tomek)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Part D Full (SMOTE-ENN)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
RUS-1 (SMOTE-Tomek)	1.00	1.00	1.00	1.00	0.94	1.00	0.97
RUS-1 (SMOTE-ENN)	1.00	1.00	1.00	1.00	0.95	1.00	0.97
RUS-5 (SMOTE-Tomek)	0.9999	1.00	1.00	1.00	0.92	0.98	0.95
RUS-5 (SMOTE-ENN)	0.9999	1.00	1.00	1.00	0.89	1.00	0.94
RUS-20 (SMOTE-Tomek)	0.9999	1.00	1.00	1.00	0.87	1.00	0.93
RUS-20 (SMOTE-ENN)	0.9999	1.00	1.00	1.00	0.87	1.00	0.93

	Prscrbr_City	Prscrbr_Type	Tot_Clms	Tot_Day_Suply	Tot_Drug_Cst	Fr
0	166	12	13	28	19.38	
1	166	12	21	148	113.40	
2	113	61	20	141	135.65	
3	113	61	95	708	294.65	
4	113	61	45	73	64.21	

RESULTS OF RANDOM FOREST CLASSIFIER

- STEP 13: Plot the AUC-ROC and AUPRC curves for each of the models.

```python
# Function to compute and plot AUPRC curve
def plot_rf_auprc(rf_results, feature_columns, target_column):
    plt.figure(figsize=(10, 8))
    auprc_scores = []
    # Loop through each dataset and plot AUPRC curve
    for name, result in rf_results.items():
        model = result['model']
        dataset = balanced_datasets[name]

        # Prepare the test set
        X = dataset[feature_columns]
        y = dataset[target_column]
        _, X_test, _, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        # Predict probabilities for the positive class
        y_prob = model.predict_proba(X_test)[:, 1]

        # Compute Precision-Recall curve and AUPRC
        precision, recall, _ = precision_recall_curve(y_test, y_prob)
        auprc = average_precision_score(y_test, y_prob)
        auprc_scores.append(auprc)
        # Plot the Precision-Recall curve
        plt.plot(recall, precision, label=f'{name} (AUPRC = {auprc:.4f})')

    # Add plot formatting
    plt.title('Precision-Recall Curve for Random Forest Models')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.legend(loc='lower left')
    plt.grid(alpha=0.3)
    plt.show()
    return auprc_scores
```

```python
# Function to compute and plot ROC curve
def plot_roc_curves(datasets, rf_results, feature_columns, target_column):
    plt.figure(figsize=(10, 8))
    auc_scores = []
    for name, dataset in datasets.items():
        # Get the trained model and test set
        model = rf_results[name]['model']

        # Split the dataset into train and test sets
        X = dataset[feature_columns]
        y = dataset[target_column]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        # Predict probabilities
        y_prob = model.predict_proba(X_test)[:, 1]

        # Compute ROC curve and AUC
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        auc = roc_auc_score(y_test, y_prob)
        auc_scores.append(auc)  # Store the AUC score

        # Plot the ROC curve
        plt.plot(fpr, tpr, label=f'{name} (AUC = {auc:.4f})')

    # Plot formatting
    plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
    plt.title('AUC-ROC Curve Comparison')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.grid()
    plt.show()
```