MACHINE LEARNING PROJECT: 3

(Classification of Consumer Data using Cross-Validation and Ensemble Methods)

BANK MARKETING CLASSIFICATION COMPARATIVE ANALYSIS

Name: Megha Garg
Roll no. 045030

REPORT

1. OBJECTIVE OF THE PROJECT

The main objective of the project is to develop and implement a robust classification model for consumer data analysis, leveraging various techniques such as cross-validation or ensemble methods, with the aim of enhancing segmentation, clustering, or class prediction in the context of the bank marketing dataset.

1. **Classification of Consumer Data into Segments using Decision Tree:** Utilize Decision Tree technique to partition the bank marketing dataset into segments or classes, ensuring reliable model evaluation and validation.
2. **Classification of Consumer Data into Segments using Cross Validation & Ensemble Methods:** Utilize cross-validation techniques and ensemble methods such as random forests and gradient boosting to partition the bank marketing dataset into segments or classes, ensuring reliable model evaluation, validation, and enhanced classification accuracy.
3. **Determination of an Appropriate Classification Model:** Compare and contrast different classification models, including default approaches and those optimized through cross-validation or ensemble techniques, to identify the most suitable model for segmenting or clustering consumer data effectively.
4. **Identification of Important Features for Classification:** Conduct feature selection and analysis to identify significant variables or features within the bank marketing dataset, along with their respective thresholds, contributing most to accurate segmentation or clustering of consumer data.

2. DATA DESCRIPTION

2.1. DATA SOURCE, SIZE, SHAPE

2.1.1 DATA SOURCE:

https://www.kaggle.com/datasets/hariharanpavan/bank-marketing-dataset-analysis-classification

2.1.2 DATA SIZE:

Size of Dataset Used - 6.4 MB

2.1.3 DATA DIMENSIONS:

Dataset consists of '19 columns and 45211 rows.
It contains preprocessed data from Project 2.

Columns: 'srno','age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan','contact','date','month','duration','campaign','pdays', 'previous' 'poutcome ','y','cluster'.

Click here to jump to the code cell.

2.2 DESCRIPTION OF VARIABLES

This is the classic marketing bank dataset uploaded originally in the UCI Machine Learning Repository. This dataset comprises the results of a marketing campaign conducted by a financial institution, wherein a diverse range of bank customers were surveyed. The aim of this survey was to gather insights into the behaviors and preferences of existing customers. These insights are instrumental in analyzing and identifying avenues for enhancing future marketing campaigns, thereby facilitating the bank in refining its strategies to better serve its clientele.

- **Srno:** Index Variable.
- **Age:** Age of the individual (18-95year old).
- **Job:** Type of job (e.g., technician, management, blue-collar).
- **Marital:** Marital status of the individual (Single, Married, Divorced)
- **Education:** Level of education attained.
- **Default:** Whether the individual has credit in default (yes or no).
- **Balance:** Average yearly balance in euros.
- **Housing:** Housing status.
- **Loan:** Whether the individual has a personal loan (yes or no).
- **Contact:** Type of communication contact (cellular, telephone).
- **Day:** Last contact day of the week.
- **Month:** Last contact month of the year.
- **Duration:** Duration of last contact in seconds.
- **Campaign:** Number of contacts performed during this campaign for this client.
- **Pdays:** Number of days since the client was last contacted from a previous campaign.
- **Previous:** Number of contacts performed before this campaign for this client.
- **Poutcome:** Outcome of the previous marketing campaign (success, failure, non-existent).
- **y:** Indicator of whether the client subscribed to a term deposit (yes or no).
- **cluster:** Representing the cluster to which the row belongs.

TYPES OF VARIABLES

1. Categorical Variables:

Categorical variables are variables that represent categories or groups. They are qualitative in nature and can take on a limited number of distinct values that belong to a specific category or group. Categorical variables can be further classified into two main types:

- **Ordinal Column - Education:** This variable represents categories with a natural ordering or ranking, such as education level (e.g., primary, secondary, tertiary).
- **Nominal Columns - Job, Marital Status, Housing, Loan, Contact, Month, y:** These variables represent categories without any inherent order or ranking. Examples include job type, marital status, housing status, loan status, contact type, month of contact, and the target variable (y) indicating subscription to a term deposit.

2. Non-Categorical Variables:

Non-categorical variables, also known as numerical variables or quantitative variables, represent quantities or measurements. Unlike categorical variables, which represent categories or groups, non-categorical variables are numeric in nature and can take on a range of numerical values.They include: **Age, Balance, Duration, Campaign, Pdays, Previous.**

Non-categorical variables are numeric in nature and can take on a range of numerical values.

3. Index Variables:
Index variable are the variables used to uniquely identify each row of the dataset. **Srno** is the index variable.

3. DATA ANALYSIS

3.1.1. DECISION TREE

Decision Tree is a popular supervised learning algorithm used for classification and regression tasks. It constructs a tree-like structure where each internal node represents a decision based on a feature, leading to a split in the data, and each leaf node represents the final output or class label. The algorithm recursively splits the data based on the most significant feature at each step, aiming to maximize information gain or minimize impurity. It's interpretable, handles both numerical and categorical data, and can capture nonlinear relationships.

Click here to jump to the code cell.

- **DecisionTreeClassifier(criterion='gini', random_state=45030,max_depth = 3):** This creates an instance of the DecisionTreeClassifier class with the specified parameters. The criterion parameter defines the criterion to measure the quality of a split, which can be 'gini', 'entropy', or 'log_loss'. The random_state parameter sets the random seed for reproducibility.max_depth is used to define the maximum depth of the tree to be formed.

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

- **dtc_model = dtc.fit(train_bank_inputs, train_bank_output):** This line fits the Decision Tree model to the training data. It learns patterns from train_bank_inputs (the features) and train_bank_output (the target variable or labels) using the fit method.
- **dtc_model.predict(test_bank_inputs):** This line invokes the predict method of the dtc_model object, which takes the test input data (test_bank_inputs) as input and returns the predicted labels for the corresponding inputs.
- To generate a visual representation of the trained Decision Tree Classifier (dtc_model) we use the plot_tree function. It displays the structure of the decision tree, including nodes, splits, and leaf nodes, making it easier to interpret the model's decision-making process.

NOTE:- After implementing a supervised machine learning algorithm utilizing all input variables, it becomes evident that the target variable's output predominantly hinges on a single variable, namely 'job.' Consequently, precision, accuracy, and F1 scores yield a value of 1, indicating an expected outcome. To assess the significance of other variables, we can prune the evident variable 'job' and analyze the remaining ones, subsequently recalculating accuracy, F1-score, and precision accordingly.

Click here to jump to the code cell.

3.1.2 CROSS VALIDATION METHOD (K-FOLD ANALYSIS)

Cross-Validation is a resampling technique used to evaluate machine learning models by partitioning the dataset into subsets, training the model on a portion of the data, and then evaluating its performance on the remaining data. The main goal of cross-validation is to assess how well a model will generalize to new, unseen data. One of the most common cross validation technique used is **K-fold.**

K-fold cross-validation is a technique for evaluating predictive models. The dataset is divided into k subsets or folds. The model is trained and evaluated k times, using a different fold as the validation set each time. Performance metrics from each fold are averaged to estimate the model's generalization performance. This method aids in model assessment, selection, and hyperparameter tuning, providing a more reliable measure of a model's effectiveness.

Click here to jump to the code cell.

In each set (fold) training and the test would be performed precisely once during this entire process. It helps us to avoid overfitting.

- A pipeline is created that first scales the data using StandardScaler and then applies on the relevant algorithm.
- A scorer for balanced accuracy is defined using make_scorer.
- Cross-validation is performed using cross_val_score with 5 folds, scoring each fold based on the balanced accuracy.
- Another scorer is defined for precision using make_scorer with weighted averaging.
- Cross-validation for the F1 score is performed with weighted averaging, and F1 scores for each fold are printed along with the average F1 score.

NOTE:- Similarly, after implementing K-Fold algorithm utilizing all input variables, it becomes evident that the target variable's output predominantly hinges on a single variable, namely 'job.' Consequently, precision, accuracy, and F1 scores yield a value of 1, indicating an expected outcome. To assess the significance of other variables, we can prune the evident variable 'job' and analyze the remaining ones, subsequently recalculating accuracy, F1-score, and precision accordingly.

Click here to jump to the code cell.

3.1.3 ENSEMBLE METHOD (RANDOM FOREST AND XGBOOST)

Ensemble Learning is a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.

3.1.3.1 RANDOM FOREST

Random Forest is an ensemble learning method that builds multiple decision trees during training. Each tree is trained on a random subset of the data and a random selection of features, resulting in a diverse set of trees. During prediction, the final outcome is determined by aggregating the predictions of all individual trees, typically using a majority vote for classification tasks.

This approach reduces overfitting and improves generalization performance. Random Forest is known for its high accuracy, robustness to noisy data, and capability to handle large datasets with high dimensionality. Additionally, it provides insights into feature importance, aiding in feature selection and interpretation.

Click here to jump to the code cell.

- RandomForestClassifier from the sklearn.ensemble module is used to initializes a Random Forest classifier with 100 decision trees and sets a random state for reproducibility.
- The classifier is trained using the fit method on the training data.

- Train and test scores are calculated using the score method on both the training and test datasets.
- The classifier predicts labels for the test data using the predict method and computes the accuracy score using accuracy_score.
- rf_classifier.estimators - returns the individual decision trees in the Random Forest ensemble classifier.

3.1.3.2 XGBOOST

XGBoost, short for eXtreme Gradient Boosting, is a powerful ensemble learning algorithm known for its efficiency and performance in both regression and classification tasks. It builds an ensemble of weak learners (decision trees) sequentially, where each subsequent tree corrects the errors made by the previous ones. XGBoost employs a gradient boosting framework, optimizing a predefined objective function through gradient descent.

It incorporates regularization techniques to prevent overfitting and offers various hyperparameters for fine-tuning model performance. XGBoost's key features include handling missing values, built-in cross-validation, and support for parallel and distributed computing, making it widely used in machine learning competitions and industry applications.

Click here to jump to the code cell.

- Import XGBoost library (xgboost).
- xgb.XGBClassifier(n_estimators=100, random_state=45030): Used to create instance of XGBoost Classifier.
- Trained the XGBoost classifier on the training data (train_bank_inputs, train_bank_output).
- Calculated the accuracy score of the trained model on both the training and test datasets using the score method.
- Used the trained classifier to make predictions on the test data (test_bank_inputs) and stored the predictions in xgb_predictions.

3.2.1. MODEL PERFORMANCE EVALUATION - CONFUSION MATRIX (DECISION TREE)

In machine learning algorithm learning, a 'Confusion Matrix' is a table that visualizes the performance of a classification algorithm by comparing predicted and actual labels for a dataset. It organizes predictions into four categories: true positives (correctly predicted positive instances), true negatives (correctly predicted negative instances), false positives (incorrectly predicted positive instances), and false negatives (incorrectly predicted negative instances). This matrix helps evaluate the model's accuracy, precision, recall, and F1 score.

Confusion matrix is calculated using 'confusion_matrix' function, which compares the actual target values (test_bank_output) with the predicted values (dtc_predict) from the decision tree model. Then, it generates a classification report using 'classification_report' function, providing various metrics such as precision, recall, F1-score, and support for each class in the target variable. Finally, it prints the classification report.

Click here to jump to the code cell.

```
      0     1    2
0  2737   409  132
1   632  2213   64
2  2078   648  130
              precision    recall  f1-score   support

         0.0       0.50      0.83      0.63      3278
         1.0       0.68      0.76      0.72      2909
         2.0       0.40      0.05      0.08      2856

    accuracy                           0.56      9043
   macro avg       0.53      0.55      0.48      9043
weighted avg       0.53      0.56      0.48      9043
```

OBSERVATION:

The decision tree algorithm achieves an accuracy of 0.56, demonstrating moderate performance. However, it exhibits imbalanced precision-recall scores, particularly for class 2, indicating potential issues with class imbalance and misclassification. Overall, while the model provides some predictive capability, its effectiveness may be limited by these imbalances.

3.2.2. MODEL PERFORMANCE EVALUATION - CONFUSION MATRIX (K-FOLD)

- **make_scorer** is used to create a scoring function from a metric or callable, allowing customization of evaluation metrics for model performance during cross-validation.

- **cross_val_score** computes the cross-validated scores for an estimator, providing a standardized way to evaluate model performance across multiple folds of the dataset.

  Click here to jump to the code cell.

```
K-Fold Cross Validation Results:
  Fold  Balanced Accuracy  Precision  Cross-Validation Score
0     1           0.537679   0.518995               0.476201
1     2           0.544408   0.394949               0.458876
2     3           0.541402   0.512158               0.475180
3     4           0.539256   0.393681               0.455049
4     5           0.544399   0.517652               0.481053

AVERAGE SCORES:
  Balanced Accuracy: 0.5414
  Precision: 0.4675
  Cross-Validation Score: 0.4693
```

OBSERVATION:

The K-Fold cross-validation approach yields an average balanced accuracy of 0.5414, indicating relatively consistent performance across folds. However, precision scores vary, with an average of 0.4675, suggesting some inconsistency in correctly identifying positive instances. Overall, the model demonstrates moderate predictive ability, with room for improvement in precision.

3.2.3. MODEL PERFORMANCE EVALUATION - CONFUSION MATRIX (ENSEMBLE METHOD)

- **RANDOM FOREST:**

  The **classification_report** function computes and displays various metrics such as precision, recall, F1-score, and support for each class in the classification task. It compares the predicted labels (rf_predictions) with the true labels (test_bank_output) to generate these metrics.

  Click here to jump to the code cell.

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.56      0.62      0.59      3278
         1.0       0.67      0.74      0.70      2909
         2.0       0.49      0.38      0.42      2856

    accuracy                           0.58      9043
   macro avg       0.57      0.58      0.57      9043
weighted avg       0.57      0.58      0.57      9043
```

  OBSERVATION:

  The random forest algorithm achieves an accuracy of 0.58, demonstrating its capability to classify instances correctly. Precision scores vary across classes, with class 1 having the highest precision at 0.67, followed by class 0 at 0.56, and class 2 at 0.49. The macro average F1-score of 0.57 indicates a reasonable balance between precision and recall across classes.

- **XGBOOST:**

  The **classification_report** function provides metrics such as precision, recall, F1-score, and support for each class in the classification task, comparing the predicted labels (xgb_predictions) with the true labels (test_bank_output).

  Click here to jump to the code cell.

```
XGBoost Accuracy: 0.5952670573924582
Classification Report:
              precision    recall  f1-score   support

         0.0       0.57      0.65      0.61      3278
         1.0       0.68      0.75      0.71      2909
         2.0       0.51      0.37      0.43      2856

    accuracy                           0.60      9043
   macro avg       0.59      0.59      0.58      9043
weighted avg       0.59      0.60      0.59      9043
```

OBSERVATION:

XGBoost demonstrates superior performance with an accuracy of 0.60, outperforming both decision tree and random forest algorithms. It achieves higher precision and recall scores across all classes, indicating its effectiveness in correctly classifying instances. The weighted average F1-score of 0.59 reflects a good balance between precision and recall, making XGBoost a promising choice for classification tasks.

3.3.1. MODEL PERFORMANCE EVALUATION - TIME & MEMORY STATISTICS (DECISION TREE)

- Libraries Used - 'time' and 'psutil'.
- time.time() is used to record the current time.
- 'psutil' library is used to obtain memory usage information (memory_usage) of the process in KB.

NOTE: As the time taken and memory utilised is different each time we execute the code, we executed the code 10 times and average of all the values are observed.

Click here to jump to the code cell.

```
Time taken: 0.06720709800720215 seconds
Memory used: 248.796875 MB
```

3.3.2. MODEL PERFORMANCE EVALUATION - TIME & MEMORY STATISTICS (K-FOLD)

Click here to jump to the code cell.

```
Time taken by k-fold cross-validation: 1.2523643970489502 seconds
Max memory usage: 755.6953125 MB
```

3.3.3. MODEL PERFORMANCE EVALUATION - TIME & MEMORY STATISTICS (ENSEMBLE METHOD)

- **RANDOM FOREST:**

  Click here to jump to the code cell.

  ```
  Time taken by Random Forest: 8.584991455078125 seconds
  Max memory usage: 474.6953125 MB
  ```

- **XGBOOST:**

  Click here to jump to the code cell.

  ```
  Time taken by Random Forest: 1.7145826816558838 seconds
  Max memory usage: 609.5859375 MB
  ```

3.4.1. VARIABLE OR FEATURE ANALYSIS (DECISION TREE)

The **'feature_importances_'** attribute of the trained Decision Tree model is used to obtain these importance scores.

NOTE: Initially, the decision-making process solely relied on the 'job' variable. However, we conducted a feature selection process to identify additional important variables that contribute to the decision-making process.

Click here to jump to the code cell.

- **3.4.1.1. List of Relevant Features:**

```
        feature    importance
1       education   0.802
10      age         0.183
0       marital     0.015
```

OBSERVATION:

Each feature's importance is determined based on its contribution to the model's predictive performance. Features such as 'education', 'age' and 'marital' have the most importance scores above the threshold, indicating their significant impact on the model's decision-making process. These features are crucial in determining the outcome of the model and play a significant role in predicting the target variable accurately.

- **3.4.1.2. List of Irrelevant Features:**

```
      feature    importance
  5    contact    0.000
  3    housing    0.000
  15   previous   0.000
  4    loan       0.000
  9    y          0.000
  8    poutcome   0.000
  2    default    0.000
  6    day        0.000
  7    month      0.000
  11   balance    0.000
  12   duration   0.000
  13   campaign   0.000
  14   pdays      0.000
```

OBSERVATION:

All features have an importance score of 0.000, indicating they have no discriminatory power in the model. This suggests that the model does not rely on these features for making predictions.
Given their lack of importance, these features may be candidates for removal from the dataset.

3.4.2. VARIABLE OR FEATURE ANALYSIS (K-FOLD)

Feature importances are calculated using the 'feature_importances_' attribute of the DecisionTreeClassifier estimator from scikit-learn. The importance values for each feature are computed during the training of the decision tree model.
Additionally, the cross_val_score() function from scikit-learn is used to perform k-fold cross-validation, allowing for robust evaluation of the model's performance and feature importances across multiple folds of the data.
Click here to jump to the code cell.

- **3.4.2.1. List of Relevant Features:**

```
        feature    importance
1       education   0.7965132464491809
10      age         0.19794831178477237
0       marital     0.005538441766046787
```

OBSERVATION:

The results from k-fold cross-validation indicate that the most influential features for classification are education, age, and marital status, with education being the most significant factor contributing to the model's performance.

- **3.4.2.2. List of Irrelevant Features:**

```
      feature    importance
  5    contact    0.000
```

```
3    housing    0.000
15   previous   0.000
4    loan       0.000
9    y          0.000
8    poutcome   0.000
2    default    0.000
6    day        0.000
7    month      0.000
11   balance    0.000
12   duration   0.000
13   campaign   0.000
14   pdays      0.000
```

OBSERVATION:

All features have an importance score of 0.000, indicating they have no discriminatory power in the model. This suggests that the model does not rely on these features for making predictions.
Given their lack of importance, these features may be candidates for removal from the dataset.

3.4.3. VARIABLE OR FEATURE ANALYSIS (ENSEMBLE METHOD)

- **3.4.3.1. RANDOM FOREST**

  rf_classifier.feature_importances_: This attribute of the trained Random Forest classifier provides the importance scores of each feature. It helps identify which features are most influential in making predictions.
  Click here to jump to the code cell.

- List Of Relevant Features:

```
        feature    importance
12   duration    0.161675
11    balance    0.160574
1   education    0.157556
10        age    0.153608
6         day    0.107518
13   campaign    0.067370
```

OBSERVATION:

The relevant features identified by the random forest algorithm suggest that the duration of the call, client's balance, education level, age, and day of contact play significant roles in the classification process. Interestingly, the duration of the call and client's balance are among the top contributors, indicating their strong predictive power in determining the outcome.

- List Of Irrelevant Features:

```
        feature    importance
7       month    0.047396
14      pdays    0.025646
0     marital    0.024170
3     housing    0.019308
5     contact    0.017295
15   previous    0.017036
4        loan    0.016061
8    poutcome    0.010488
9           y    0.010103
2     default    0.004196
```

OBSERVATION:

The list highlights several features such as marital status, housing, contact method, and loan status, among others, with notably low importance scores. This suggests that these attributes have minimal impact on the classification outcome, indicating their relatively low relevance in predicting the target variable.

- **3.4.3.2. XGBOOST**

  The **feature_importances_** attribute is used to retrieve the importance scores of each feature from a trained XGBoost classifier. These importance scores indicate the contribution of each feature to the predictions made by the XGBoost model.
  Click here to jump to the code cell.

  - List Of Relevant Features:

    ```
          feature    importance
    1   education    0.55675
    3     housing    0.08368
    10        age    0.06168
    0     marital    0.04121
    7       month    0.03808
    ```

    OBSERVATION:

    The list showcases the importance of various features in XGBoost classification, with education being the most significant predictor followed by housing, age, and marital status. This underscores the influence of demographic and socio-economic factors in determining the target variable, highlighting their relevance in the classification process.

  - List Of Irrelevant Features:

    ```
            feature    importance
    14        pdays    0.02215
    11      balance    0.02159
    5       contact    0.02135
    4          loan    0.02104
    6           day    0.01999
    12     duration    0.01979
    9             y    0.01967
    2       default    0.01909
    13     campaign    0.01866
    8      poutcome    0.01811
    15     previous    0.01716
    ```

    OBSERVATION:

    The list provides the importance of various features in the context of their relevance for the classification task. However, none of the features show significant importance, as their importance scores are relatively low and comparable. This suggests that these features may not contribute substantially to the classification process, hence considered as irrelevant.

4. RESULT

4.1. CLASSIFICATION MODEL PARAMETER

To assess and evaluate the performance of different models, we will utilize multiple metrics, including Accuracy, Precision, Recall, and F1-score.

Accuracy measures the overall correctness of predictions.

Precision quantifies the proportion of true positive predictions among all positive predictions.

Recall assesses the proportion of true positive predictions identified correctly.

F1-score balances Precision and Recall, providing a harmonic mean between the two metrics.

- **DECISION TREE:**
  Click here to jump to the code cell.

```
      0     1    2
0  2737   409  132
1   632  2213   64
2  2078   648  130
              precision    recall  f1-score   support

         0.0       0.50      0.83      0.63      3278
         1.0       0.68      0.76      0.72      2909
         2.0       0.40      0.05      0.08      2856

    accuracy                           0.56      9043
   macro avg       0.53      0.55      0.48      9043
weighted avg       0.53      0.56      0.48      9043
```

OBSERVATION:

- The confusion matrix provides a breakdown of the model's predictions for each class: 0, 1, and 2. It indicates the number of true positives, false positives, and false negatives for each class.
- Class 0 (label 0.0) has a precision of 0.50, recall of 0.83, and F1-score of 0.63, suggesting relatively good performance in correctly identifying instances of this class. However, there are some misclassifications, as indicated by the false positives and false negatives.
- Class 1 (label 1.0) exhibits higher precision (0.68) and recall (0.76) compared to class 2, indicating better predictive ability for this class. The F1-score for class 1 is also relatively high at 0.72.
- Class 2 (label 2.0) shows lower precision (0.40) and recall (0.05) compared to the other classes, indicating challenges in accurately predicting instances of this class. The F1-score for class 2 is the lowest among all classes, at 0.08.
- The overall accuracy of the model is 0.56, indicating the proportion of correctly classified instances out of all instances.
- Both macro avg and weighted avg metrics provide insights into the average performance across all classes, considering class imbalances. In this case, they offer similar values due to class distribution similarities.

- **CROSS-VALIDATION(K-FOLD):**
  Click here to jump to the code cell.

```
K-Fold Cross Validation Results:
  Fold  Balanced Accuracy  Precision  Cross-Validation Score
0     1           0.537679   0.518995                0.476201
1     2           0.544408   0.394949                0.458876
2     3           0.541402   0.512158                0.475180
3     4           0.539256   0.393681                0.455049
4     5           0.544399   0.517652                0.481053

AVERAGE SCORES:
  Balanced Accuracy: 0.5414
  Precision: 0.4675
  Cross-Validation Score: 0.4693
```

OBSERVATION:

- The balanced accuracy scores across the folds range from approximately 0.538 to 0.544, with an average score of 0.5414 demonstrating moderate consistency in correctly classifying instances across different folds.
- Precision scores vary notably between folds, ranging from approximately 0.395 to 0.518, with an average precision of 0.4675 suggesting potential instability in predicting positive instances.
- The model's overall performance, as measured by cross-validation ranges from approximately 0.455 to 0.481 across folds, with an average score of 0.4693 and 5-fold having the best CV Score suggesting moderate performance.
- The average scores indicate moderate performance in terms of balanced accuracy and cross-validation, but a relatively lower precision score.

- **RANDOM FOREST:**
  Click here to jump to the code cell.

  ```
  Classification Report:
                precision    recall  f1-score   support

          0.0       0.56      0.62      0.59      3278
          1.0       0.67      0.74      0.70      2909
          2.0       0.49      0.38      0.42      2856

      accuracy                           0.58      9043
     macro avg       0.57      0.58      0.57      9043
  weighted avg       0.57      0.58      0.57      9043
  ```

  OBSERVATION:

  - The precision, recall, and F1-score for class 0 are 0.56, 0.62, and 0.59, respectively showing moderate precision and recall in identifying instances of class 0, with a balanced F1-score.
  - For class 1, the precision, recall, and F1-score are 0.67, 0.74, and 0.70, respectively demonstrating relatively high precision and recall in classifying instances of class 1, resulting in a well-balanced F1-score.
  - Class 2 exhibits a precision of 0.49, recall of 0.38, and F1-score of 0.42 demonstrating relatively weaker compared to other classes, with lower precision, recall, and F1-score.
  - The overall accuracy of the model is 0.58, indicating the proportion of correctly classified instances across all classes.
  - The macro-average F1-score is 0.57, representing the average of the F1-scores for each class.
  - The weighted average F1-score, also 0.57, considers the class imbalance by weighting the F1-score of each class by its support (the number of true instances).

- **XGBOOST:**
  Click here to jump to the code cell.

  ```
  XGBoost Accuracy: 0.5952670573924582
  Classification Report:
                precision    recall  f1-score   support

          0.0       0.57      0.65      0.61      3278
          1.0       0.68      0.75      0.71      2909
          2.0       0.51      0.37      0.43      2856

      accuracy                           0.60      9043
     macro avg       0.59      0.59      0.58      9043
  weighted avg       0.59      0.60      0.59      9043
  ```

  OBSERVATION:

  - The XGBoost model achieves an accuracy of approximately 59.53%, indicating the proportion of correctly classified instances across all classes.
  - For class 0, the precision, recall, and F1-score are 0.57, 0.65, and 0.61, respectively. This suggests that the model demonstrates moderate precision and recall in identifying instances of class 0, resulting in a well-balanced F1-score.
  - Class 1 exhibits a precision of 0.68, recall of 0.75, and F1-score of 0.71, indicating relatively high precision and recall in classifying instances of class 1. This contributes to a balanced F1-score for this class.
  - The precision, recall, and F1-score for class 2 are 0.51, 0.37, and 0.43, respectively. These metrics suggest that the model's performance in identifying instances of class 2 is relatively weaker compared to other classes, with lower precision, recall, and F1-score.
  - The macro-average F1-score is approximately 0.58, representing the average of the F1-scores for each class. This indicates that the model's performance is relatively balanced across all classes, with similar contributions from each class to the overall F1-score.

  - The weighted average F1-score, approximately 0.59, considers the class imbalance by weighting the F1-score of each class by its support. This metric suggests that the model's performance, when accounting for class imbalance, remains consistent with the macro-average F1-score.

4.2. CLASSIFICATION MODEL PERFORMANCE: TIME & MEMORY STATISTICS

Time Statistics: The time taken by a model to train and make predictions directly impacts its practical usability. Models with shorter training and prediction times are preferred, especially in real-time or time-sensitive applications.

Memory Statistics: Memory usage is crucial for understanding the resource requirements of a model. Models that consume excessive memory may not be suitable for deployment on resource-constrained environments like mobile devices or embedded systems.

As the time taken and memory utilised is different each time we execute the code, we executed the code 10 times and average of all the values are observed.

- **DECISION TREE:**
  Click here to jump to the code cell.

  ```
  Time taken: 0.06720709800720215 seconds
  Memory used: 248.796875 MB
  ```

  OBSERVATION:

  - The Decision Tree algorithm executed swiftly, requiring approx. 0.07 seconds to complete the classification task.
  - Its memory usage was moderate, consuming approximately 248.8 MB.
  - Despite its simplicity, the Decision Tree algorithm can provide reasonable classification results and is suitable for scenarios where interpretability is crucial. However, its performance might be limited in handling complex relationships within the data compared to ensemble methods.

- **K-FOLD:**
  Click here to jump to the code cell.

  ```
  Time taken by k-fold cross-validation: 1.2523643970489502 seconds
  Max memory usage: 755.6953125 MB
  ```

  OBSERVATION:

  - The K-fold cross-validation process took longer to execute, averaging around 1.25 seconds.
  - It utilized a significant amount of memory, reaching up to 755.7 MB during execution.
  - K-fold cross-validation is essential for assessing the model's performance and generalization ability, but its computational cost can be relatively high.

- **RANDOM FOREST:**
  Click here to jump to the code cell.

  ```
  Time taken by Random Forest: 8.584991455078125 seconds
  Max memory usage: 474.6953125 MB
  ```

  OBSERVATION:

  - The Random Forest algorithm demonstrated a longer execution time, taking around 8.58 seconds to complete the classification task.
  - Despite its longer runtime, it consumed a relatively lower amount of memory, approx. 474.7 MB.
  - Random Forest combines multiple decision trees to improve predictive accuracy and generalization, making it suitable for more complex classification tasks.

- **XGBOOST:**
  Click here to jump to the code cell.

```
Time taken by Random Forest: 1.7145826816558838 seconds
Max memory usage: 609.5859375 MB
```

OBSERVATION:

- XGBoost exhibited a moderate execution time, requiring approximately 1.71 seconds to complete.
- It utilized a reasonable amount of memory, reaching up to 609.6 MB.
- Its balanced performance in terms of time and memory usage makes it a preferred choice for various classification tasks, especially with structured/tabular data.

4.3. VARIABLE OR FEATURE ANALYSIS

Variable or feature analysis is crucial for selecting relevant attributes, optimizing model performance, and gaining insights into data patterns, thereby enhancing model interpretability and domain understanding.

NOTE: Initially, the decision-making process solely relied on the 'job' variable. However, we conducted a feature selection process to identify additional important variables that contribute to the decision-making process.

4.3.1.DECISION TREE:

Click here to jump to the code cell.

- Relevant or Important Variables

```
        feature    importance
1       education   0.802
10      age         0.183
0       marital     0.015
```

OBSERVATION:

- Education: The feature "education" has the highest importance score of 0.802, indicating that it significantly influences the decision tree's predictions. This suggests that the level of education plays a crucial role in determining the outcome or target variable.
- Age: Although less influential compared to education, the "age" feature still holds importance with a score of 0.183. Age likely contributes to decision-making processes in the model, implying that different age groups may exhibit distinct patterns or behaviors relevant to the target variable.
- Marital Status: The feature "marital" has the lowest importance score of 0.015, suggesting it has relatively minor influence compared to education and age. However, it still contributes to the decision-making process, indicating that marital status may have some predictive value in the model, albeit to a lesser extent.

- Irrelevant or Non-Important Variables

```
     feature    importance
5    contact    0.000
3    housing    0.000
15   previous   0.000
4    loan       0.000
9    y          0.000
8    poutcome   0.000
2    default    0.000
6    day        0.000
7    month      0.000
11   balance    0.000
12   duration   0.000
13   campaign   0.000
14   pdays      0.000
```

OBSERVATION:

    - None of these features considered in the analysis show any importance, as indicated by their importance scores of 0. This suggests that these features do not contribute to the predictive power of the model and are essentially ignored in the decision-making process.
    - The absence of importance for all features implies that they do not provide meaningful information for making predictions or classifications. It could be that these features are irrelevant or poorly correlated with the target variable.

4.3.2.CROSS VALIDATION METHOD(K-FOLD):

Click here to jump to the code cell.

- Relevant or Important Variables

```
        feature    importance
1       education   0.7965132464491809
10      age         0.19794831178477237
0       marital     0.005538441766046787
```

  OBSERVATION:

    - Education (0.7965): Education level appears to be the most significant feature in predicting the target variable suggesting that individuals with different educational backgrounds may exhibit varying behaviors or responses, making it a crucial factor for classification.
    - Age (0.1979): Age is identified as the second most important feature. This indicates that age plays a significant role in determining the outcome, with different age groups likely demonstrating distinct patterns or behaviors in the dataset.
    - Marital Status (0.0055): Marital status has the lowest importance among the three features considered. While it may still contribute to the classification process to some extent, its impact seems relatively minor compared to education and age.

- Irrelevant or Non-Important Variables

```
    feature    importance
5    contact     0.000
3    housing     0.000
15   previous    0.000
4    loan        0.000
9    y           0.000
8    poutcome    0.000
2    default     0.000
6    day         0.000
7    month       0.000
11   balance     0.000
12   duration    0.000
13   campaign    0.000
14   pdays       0.000
```

  OBSERVATION:

    - No Importance: All features listed have an importance value of 0. This suggests that these features do not contribute significantly to the classification process and have minimal impact on predicting the target variable.
    - Potential Removal: Since these features have no discernible influence on the classification outcome, they could be considered for removal from the dataset to simplify the model and potentially improve performance and computational efficiency.
    - Focus on Relevant Features: The absence of importance underscores the need to focus on features that have a meaningful impact on the target variable. By identifying and prioritizing relevant features, the model can be optimized to better capture the underlying patterns in the data.

4.3.3.RANDOM FOREST:

Click here to jump to the code cell.

- Relevant or Important Variables

```
       feature    importance
12    duration    0.161675
11     balance    0.160574
1    education    0.157556
10         age    0.153608
6          day    0.107518
13    campaign    0.067370
```

OBSERVATION:

- The duration of the call, customer's balance, and level of education are highly influential in predicting campaign outcomes. These factors collectively indicate the quality of customer engagement, financial stability, and educational background, which significantly impact response rates and campaign success.
- Customer age and the day of the call also play substantial roles in determining campaign effectiveness.Different age demographics exhibit varying financial behaviors, and certain days of the month may coincide with key financial events.
- The frequency of contacts made during the campaign contributes moderately to prediction accuracy.

- Irrelevant or Non-Important Variables

```
       feature    importance
7        month    0.047396
14       pdays    0.025646
0      marital    0.024170
3      housing    0.019308
5      contact    0.017295
15    previous    0.017036
4         loan    0.016061
8     poutcome    0.010488
9            y    0.010103
2      default    0.004196
```

OBSERVATION:

- The month of the last contact, number of days since the last contact (pdays), and marital status have minimal importance. This suggests that these variables may not significantly influence the segmentation process.
- Variables related to housing, type of contact, and previous campaign interactions hold little importance.
- Factors such as loan status, outcome of the previous campaign, and the target variable (subscription) demonstrate low importance.These variables exhibit negligible influence on clustering process. While they may provide contextual information, their minimal importance suggests that other factors play more significant roles.

4.3.4. XGBOOST:

Click here to jump to the code cell.

- Relevant or Important Variables

```
       feature    importance
1    education    0.55675
3      housing    0.08368
10         age    0.06168
0      marital    0.04121
7        month    0.03808
```

OBSERVATION:

- Education level holds the highest importance in the cluster making process, with an importance score of 0.55675.Higher education levels may correlate with specific behaviors or preferences, influencing their placement within distinct clusters.
- Variables related to housing status and age demonstrate moderate importance, with scores of 0.08368 and 0.06168, respectively indicating that individuals with similar housing arrangements or age groups may exhibit comparable characteristics or responses to

certain stimuli.

- Marital status and the month of contact show relatively lower but still notable importance in the cluster making process.These factors likely contribute additional context to understanding customer segmentation and behavior patterns.

- Irrelevant or Non-Important Variables

```
       feature    importance
14       pdays    0.02215
11     balance    0.02159
5      contact    0.02135
4         loan    0.02104
6          day    0.01999
12    duration    0.01979
9            y    0.01967
2      default    0.01909
13    campaign    0.01866
8     poutcome    0.01811
15    previous    0.01716
```

OBSERVATION:

- Variables such as 'pdays' and 'balance' exhibit slightly higher importance compared to other non-relevant features, but their importance scores are still relatively low.While these features may have some minor impact on the clustering process, their overall contribution is limited.
- 'Contact' and 'loan' status demonstrate similar low importance levels in the clustering process.Despite being included as variables, the choice of contact and whether an individual has a loan do not significantly influence the clustering outcomes.
- Other variables such as 'day', 'duration', 'poutcome' (outcome of the previous marketing campaign), and 'campaign' (number of contacts performed during this campaign) also exhibit low importance. These factors do not substantially contribute to the clustering process.

5. MANAGERIAL INSIGHTS

5.1. Determination Of Appropriate Classification Model

5.1.1. Decision Tree Vs Cross Validation(K-Fold)

- On the Basis of Classification Model Parameters:
  - Performance Metrics:
    - Decision Tree: The decision tree classifier achieved an accuracy of 56% on the test data, with varying precision, recall, and F1-score across different classes. The weighted average precision, recall, and F1-score were 0.53, 0.56, and 0.48, respectively.
    - K-Fold Cross Validation: The K-Fold classification method yielded an average balanced accuracy of 54.14% and an average precision of 46.75%. However, the cross-validation scores were slightly lower, with an average of 46.93%.
  - Balanced Accuracy:
    - Both methods achieved similar balanced accuracy scores, with the decision tree classifier showing a balanced accuracy of approximately 55%, and the K-Fold cross-validation averaging at 54.14%. This indicates that both methods performed comparably in terms of correctly classifying each class, considering imbalanced data distribution.
  - Precision and Cross-Validation Score:
    - The decision tree classifier exhibited higher precision values for individual folds compared to the K-Fold cross-validation method. However, the average precision score from K-Fold was slightly lower than that of the decision tree. Similarly, the cross-validation scores for K-Fold were marginally lower than the precision values obtained from the decision tree classifier.

Overall, we can say that **Decision tree classifier outperforms K-Fold cross-validation in terms of precision and interpretability, showcasing a clearer decision-making process and higher individual precision values.**

5.1.2. Decision Tree Vs Ensemble Methods(Random Forest & XGBoost)

- On the Basis of Classification Model Parameters:
    - Accuracy:
        - XGBoost achieves the highest accuracy of 0.60, followed by Random Forest with 0.58, and Decision Tree with 0.56. This indicates that XGBoost generally provides better predictive performance.
    - Precision and Recall:
        - XGBoost and Random Forest exhibit balanced precision and recall scores across different classes, whereas the Decision Tree shows imbalanced scores, particularly for class 2.
        - XGBoost (0.58) and Random Forest (0.57) achieve higher precision and recall for class 2 compared to the Decision Tree, suggesting better classification of this class.
    - F1-Score:
        - XGBoost and Random Forest achieve higher F1-scores across all classes compared to the Decision Tree, indicating better balance between precision and recall.
    - Generalization:
        - XGBoost and Random Forest demonstrate better generalization performance, as evidenced by their higher accuracy and balanced precision-recall scores.
          Decision Tree may suffer from overfitting, as indicated by its lower accuracy and imbalanced precision-recall scores.

  In summary, **XGBoost and Random Forest outperform the Decision Tree** in terms of accuracy, balanced precision-recall scores, and generalization ability, with XGBoost achieving the highest overall performance.

CONCLUSION:

Ensemble methods, exemplified by **XGBoost followed by Random Forest, outshine default decision trees and k-fold analysis** in predictive accuracy, precision, and recall. Their balanced performance across diverse datasets, coupled with efficient resource utilization, positions them as superior options for classification tasks. With better scalability and generalization, ensemble methods offer enhanced predictive power and reliability in real-world applications compared to traditional techniques.

5.2. Relevant or Important Variables or Features

- **Job (Occupation):**
  Understanding the influence of occupation on customer behavior is crucial for targeted marketing strategies. Tailoring campaigns based on occupation-specific preferences and needs can enhance engagement and conversion rates. For example, provide exclusive mortgage rates for healthcare professionals or retirement planning seminars for educators.
- **Education:**
  Education level serves as a key determinant of financial literacy and decision-making. Designing educational content or financial products tailored to different educational backgrounds can improve customer understanding and satisfaction. for example, organize educational workshops or webinars focused on financial literacy targeted at various education levels. Offer topics like budgeting basics for students, investment strategies for professionals, and retirement planning for seniors.
- **Housing:**
  Housing status reflects individual financial stability and lifestyle preferences. Offering housing-related products or services, such as mortgages or rental assistance, can be tailored based on homeownership status to address specific customer needs. Provide incentives like lower interest rates for first-time homebuyers or renovation loans for existing homeowners.
- **Age:**
  Age segmentation allows for customized marketing approaches catering to different life stages and priorities. Developing age-specific promotions or services can resonate better with target demographics and increase customer engagement. Offer retirement savings plans for older customers and education savings accounts for younger demographics.
- **Marital Status:**
  Marital status influences household dynamics and financial decision-making. Crafting marketing campaigns that resonate with married or single individuals shared or individual financial goals can improve campaign effectiveness and customer satisfaction. Develop joint banking accounts or family savings plans targeting married couples. For example, creating individualized financial planning services for singles focusing on personal goals like homeownership or travel.

- **Month:**
  Seasonality and temporal factors impact consumer behavior and purchasing patterns. Adapting marketing strategies to leverage seasonal trends or capitalize on specific months' consumer behaviors can optimize campaign performance and ROI.

Recognizing the significance of these features allows marketers to personalize strategies, tailor offerings, and optimize resource allocation to maximize customer engagement, satisfaction, and retention. Strategic utilization of these insights can drive more effective and targeted marketing campaigns, leading to improved business outcomes and customer relationships.

5.3. Managerial Implications

- **Customized Financial Products:** Utilize the education level and age of customers to design customized financial products and services that cater to their specific life stages and financial goals. For example, offering educational loans or investment plans targeted towards younger customers with higher education levels, while providing retirement planning options for older customers.
- **Targeted Marketing Campaigns:** Segment marketing campaigns based on marital status and age demographics to ensure relevant and personalized messaging. For instance, promoting mortgage or family-oriented financial planning services to married individuals, while offering retirement savings options to older customers.
- **Financial Education Programs:** Develop financial education programs or workshops tailored to the educational background of customers. Focus on providing resources and guidance that align with the financial literacy levels of different education groups, empowering them to make informed financial decisions.
- **Customer Engagement Strategies:** Implement targeted customer engagement strategies based on demographic characteristics such as age and marital status. This could involve personalized communication channels or incentives aimed at increasing customer satisfaction and loyalty within each cluster.
- **Risk Management:** Assess the risk profiles of customers within each cluster based on their demographic attributes. Use this information to adjust lending criteria, interest rates, or credit limits to mitigate risks associated with different customer segments.

REFERENCES

- Dataset: Kaggle (https://www.kaggle.com/datasets/hariharanpavan/bank-marketing-dataset-analysis-classification)
- Code Reference: medium.com, geekforgeeks, notes
- Google: analytixlabs.co.in, scikit-learn.org
- Other References: chatgpt, gemini

CODE

```
pip install memory-profiler
```

```
Collecting memory-profiler
  Downloading memory_profiler-0.61.0-py3-none-any.whl (31 kB)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from memory-profiler) (5.9.5)
Installing collected packages: memory-profiler
Successfully installed memory-profiler-0.61.0
```

```
# Required Libraries

import pandas as pd, numpy as np # For Data Manipulation
import matplotlib.pyplot as plt, seaborn as sns # For Data Visualization
from sklearn.model_selection import train_test_split # For Splitting Data into Training & Testing Sets
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree  # For Decision Tree Model
from sklearn.metrics import confusion_matrix, classification_report # For Decision Tree Model Evaluation

import warnings

warnings.filterwarnings("ignore") # Ignore the warnings
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

```

```
Mounted at /content/drive
```

```python

file_path = 'bank.csv'

# Read CSV file
df_ppd = pd.read_csv(file_path)
```

```python
df_ppd
```

	srno	job	marital	education	default	housing	loan	contact	day	month	poutcome	y	cluster	age	balance	duration	campaign	pdays
0	0	4.0	1.0	2.0	0.0	1.0	0.0	2.0	4.0	8.0	3.0	0.0	1.0	0.519481	0.092259	0.053070	0.000000	0.000000
1	1	9.0	2.0	1.0	0.0	1.0	0.0	2.0	4.0	8.0	3.0	0.0	2.0	0.337662	0.073067	0.030704	0.000000	0.000000
2	2	2.0	1.0	1.0	0.0	1.0	1.0	2.0	4.0	8.0	3.0	0.0	0.0	0.194805	0.072822	0.015453	0.000000	0.000000
3	3	1.0	1.0	3.0	0.0	1.0	0.0	2.0	4.0	8.0	3.0	0.0	0.0	0.376623	0.086476	0.018707	0.000000	0.000000
4	4	11.0	2.0	3.0	0.0	0.0	0.0	2.0	4.0	8.0	3.0	0.0	2.0	0.194805	0.072812	0.040260	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	45206	9.0	1.0	2.0	0.0	0.0	0.0	0.0	16.0	9.0	3.0	1.0	2.0	0.428571	0.080293	0.198658	0.032258	0.000000
45207	45207	5.0	0.0	0.0	0.0	0.0	0.0	0.0	16.0	9.0	3.0	1.0	1.0	0.688312	0.088501	0.092721	0.016129	0.000000
45208	45208	5.0	1.0	1.0	0.0	0.0	0.0	0.0	16.0	9.0	2.0	1.0	1.0	0.701299	0.124689	0.229158	0.064516	0.212156
45209	45209	1.0	1.0	1.0	0.0	0.0	0.0	1.0	16.0	9.0	3.0	0.0	0.0	0.506494	0.078868	0.103294	0.048387	0.000000
45210	45210	2.0	1.0	1.0	0.0	0.0	0.0	0.0	16.0	9.0	1.0	0.0	0.0	0.246753	0.099777	0.073404	0.016129	0.216743

45211 rows × 19 columns

```python
df_ppd.shape
```

```
(45211, 19)
```

```python
df_ppd.columns
```

```
Index(['srno', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'day', 'month', 'poutcome', 'y', 'cluster', 'age', 'balance',
       'duration', 'campaign', 'pdays', 'previous'],
      dtype='object')
```

```python
# We have to consider all the features as input variables.
bank_inputs = df_ppd[['job', 'marital', 'education','default','housing','loan','contact','day','month','poutcome','y',
                      'age','balance','duration','campaign','pdays','previous']]; bank_inputs

# Output variable - cluster
bank_output = df_ppd[['cluster']]; bank_output
```

	cluster
0	1.0
1	2.0
2	0.0
3	0.0
4	2.0
...	...
45206	2.0
45207	1.0
45208	1.0
45209	0.0
45210	0.0

45211 rows × 1 columns

```python
# saving the names/labels of all the input and output variables.

bank_inputs_names = bank_inputs.columns; print("Input Names: ",bank_inputs_names)
bank_output_labels = bank_output['cluster'].unique().astype(str); print("Output Label: ", bank_output_labels)
```

```
Input Names:  Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'day', 'month', 'poutcome', 'y', 'age', 'balance', 'duration',
       'campaign', 'pdays', 'previous'],
      dtype='object')
Output Label:  ['1.0' '2.0' '0.0']
```

```python
# Splitting the dataset into test & train
train_bank_inputs, test_bank_inputs, train_bank_output, test_bank_output = train_test_split(bank_inputs, bank_output, test_size=0.20, stratify=bank_o

```

DECISION TREE

```
# Decision Tree
import time
import psutil


start_time = time.time()
dtc = DecisionTreeClassifier(criterion='gini', random_state=45030,max_depth = 3) # Other Criteria : Entropy,  Log Loss
dtc_model = dtc.fit(train_bank_inputs, train_bank_output); dtc_model
end_time = time.time()

execution_time = end_time - start_time
print("Time taken:", execution_time, "seconds")

# Memory usage
process = psutil.Process()
memory_usage = process.memory_info().rss /1024  # in KB
print("Memory used:", memory_usage/1024, "MB")

print("Model:",dtc_model)
```

```
Time taken: 0.04039573669433594 seconds
Memory used: 236.96875 MB
Model: DecisionTreeClassifier(max_depth=3, random_state=45030)
```

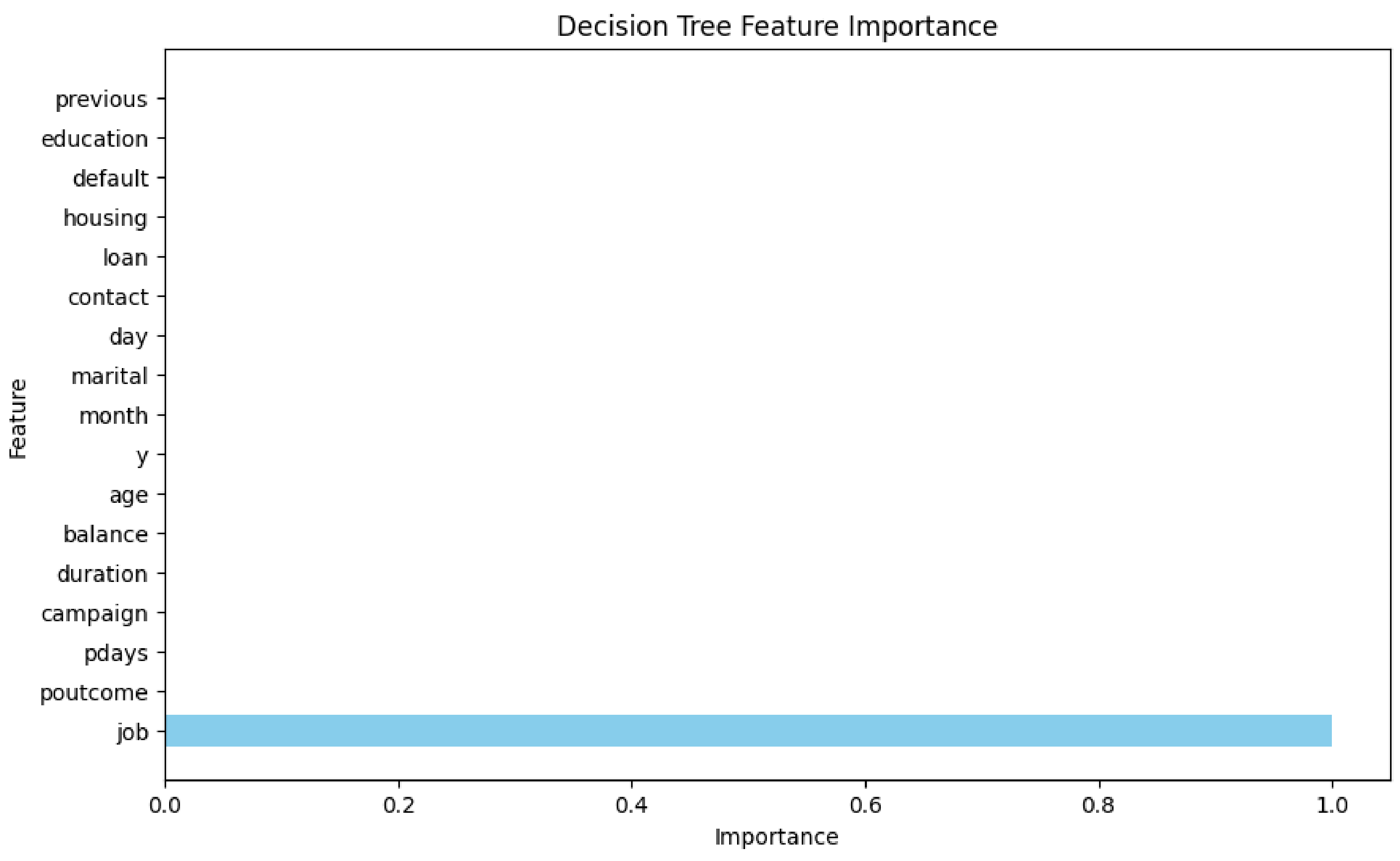
```
# Decision Tree : Model Rules
dtc_model_rules = export_text(dtc_model, feature_names = list(bank_inputs_names)); print(dtc_model_rules)

```

```
|--- job <= 2.50
|   |--- class: 0.0
|--- job >  2.50
|   |--- job <= 6.50
|   |   |--- class: 1.0
|   |--- job >  6.50
|   |   |--- class: 2.0
```

```
# Decision Tree : Feature Importance
dtc_imp_features = pd.DataFrame({'feature': bank_inputs_names, 'importance': np.round(dtc_model.feature_importances_, 3)})
dtc_imp_features.sort_values('importance', ascending=False, inplace=True); dtc_imp_features
```

	feature	importance
0	job	1.0
9	poutcome	0.0
15	pdays	0.0
14	campaign	0.0
13	duration	0.0
12	balance	0.0
11	age	0.0
10	y	0.0
8	month	0.0
1	marital	0.0
7	day	0.0
6	contact	0.0
5	loan	0.0
4	housing	0.0
3	default	0.0
2	education	0.0
16	previous	0.0

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.barh(dtc_imp_features['feature'], dtc_imp_features['importance'], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importance')
plt.show()

```



```
bank_inputs_new_dt = df_ppd[['marital', 'education','default','housing','loan','contact','day','month','poutcome','y',
                      'age','balance','duration','campaign','pdays','previous']];
bank_inputs_names = bank_inputs_new_dt.columns; print("Input Names: ",bank_inputs_names)
```

```
Input Names:  Index(['marital', 'education', 'default', 'housing', 'loan', 'contact', 'day',
       'month', 'poutcome', 'y', 'age', 'balance', 'duration', 'campaign',
       'pdays', 'previous'],
      dtype='object')
```

```
# Splitting the dataset into test & train
train_bank_inputs, test_bank_inputs, train_bank_output, test_bank_output = train_test_split(bank_inputs_new_dt, bank_output, test_size=0.20, stratify

```

```
# Decision Tree : Model Without 'Job'
import time
import psutil


start_time = time.time()
dtc = DecisionTreeClassifier(criterion='gini', random_state=45030,max_depth = 3) # Other Criteria : Entropy,  Log Loss
dtc_model = dtc.fit(train_bank_inputs, train_bank_output); dtc_model
end_time = time.time()

execution_time = end_time - start_time
print("Time taken:", execution_time, "seconds")

# Memory usage
process = psutil.Process()
memory_usage = process.memory_info().rss /1024  # in KB
print("Memory used:", memory_usage/1024, "MB")

print("Model:",dtc_model)
```

```
Time taken: 0.058518648147583 01 seconds
Memory used: 254.328125 MB
Model: DecisionTreeClassifier(max_depth=3, random_state=45030)
```

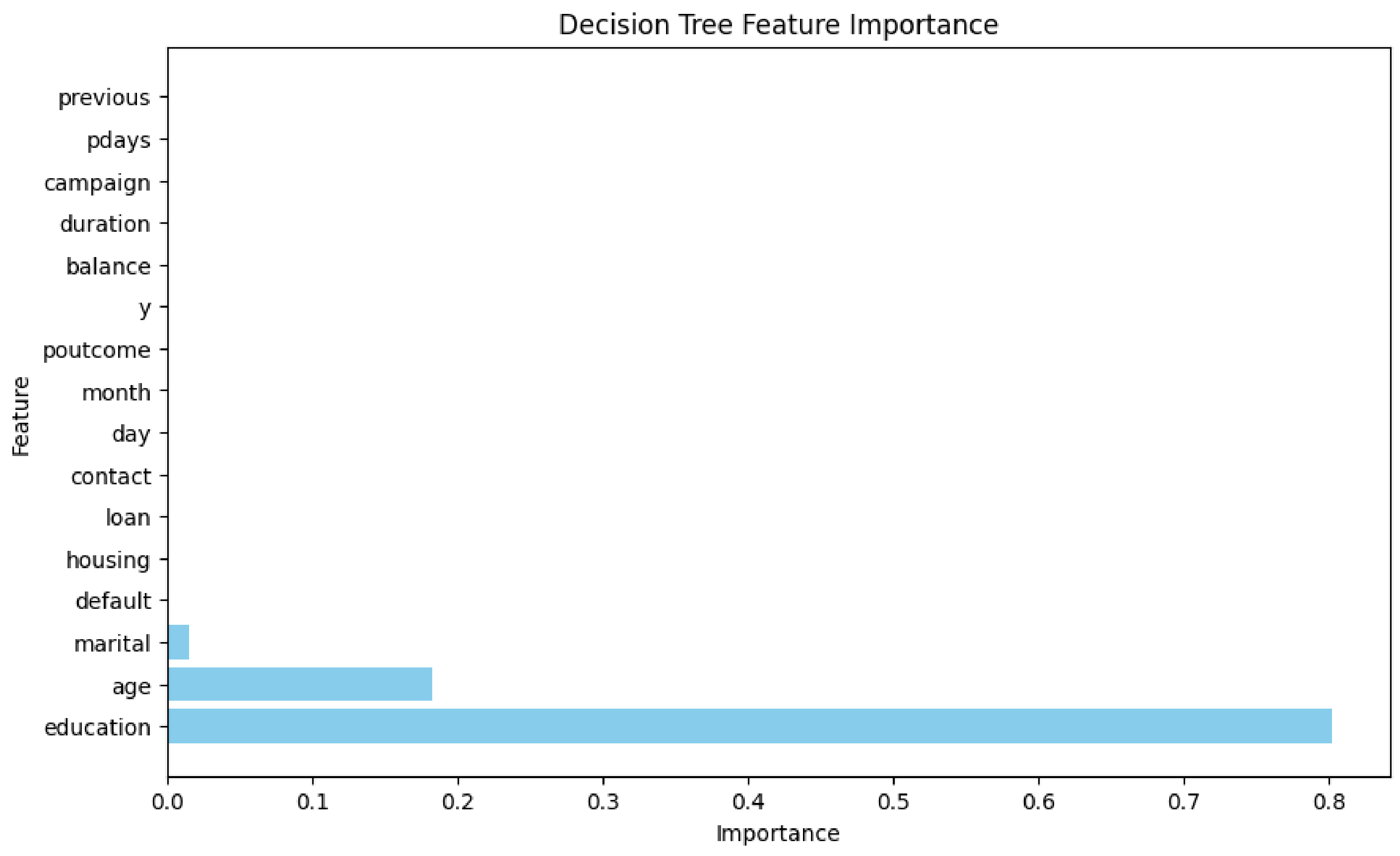
```
# Decision Tree : Model Rules
dtc_model_rules = export_text(dtc_model, feature_names = list(bank_inputs_names)); print(dtc_model_rules)

```

```
|--- education <= 1.50
|   |--- age <= 0.50
|   |   |--- education <= 0.50
|   |   |   |--- class: 0.0
|   |   |--- education >  0.50
|   |   |   |--- class: 0.0
|   |--- age >  0.50
|   |   |--- age <= 0.55
|   |   |   |--- class: 1.0
|   |   |--- age >  0.55
|   |   |   |--- class: 1.0
|--- education >  1.50
|   |--- education <= 2.50
|   |   |--- marital <= 1.50
|   |   |   |--- class: 1.0
|   |   |--- marital >  1.50
|   |   |   |--- class: 1.0
|   |--- education >  2.50
|   |   |--- age <= 0.53
|   |   |   |--- class: 2.0
|   |   |--- age >  0.53
|   |   |   |--- class: 1.0
```

```
# Decision Tree : Feature Importance
dtc_imp_features = pd.DataFrame({'feature': bank_inputs_names, 'importance': np.round(dtc_model.feature_importances_, 3)})
dtc_imp_features.sort_values('importance', ascending=False, inplace=True); dtc_imp_features
```

	feature	importance
1	education	0.802
10	age	0.183
0	marital	0.015
2	default	0.000
3	housing	0.000
4	loan	0.000
5	contact	0.000
6	day	0.000
7	month	0.000
8	poutcome	0.000
9	y	0.000
11	balance	0.000
12	duration	0.000
13	campaign	0.000
14	pdays	0.000
15	previous	0.000

```
import matplotlib.pyplot as plt


plt.figure(figsize=(10, 6))
plt.barh(dtc_imp_features['feature'], dtc_imp_features['importance'], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importance')
plt.show()

```



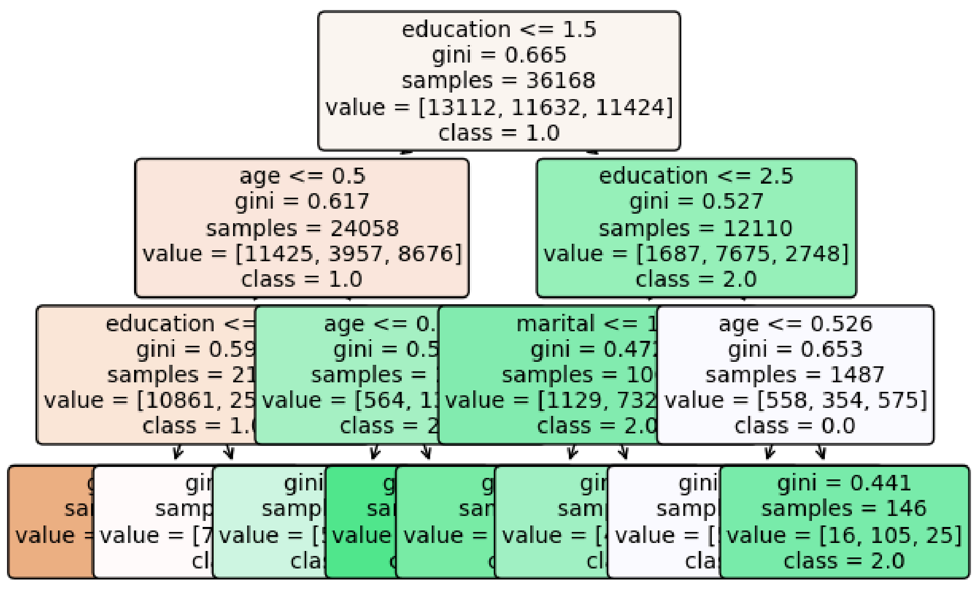
```
# Decision Tree : Prediction (Testing Subset)
dtc_predict = dtc_model.predict(test_bank_inputs); dtc_predict

```

```
array([1., 0., 0., ..., 0., 0., 0.])
```

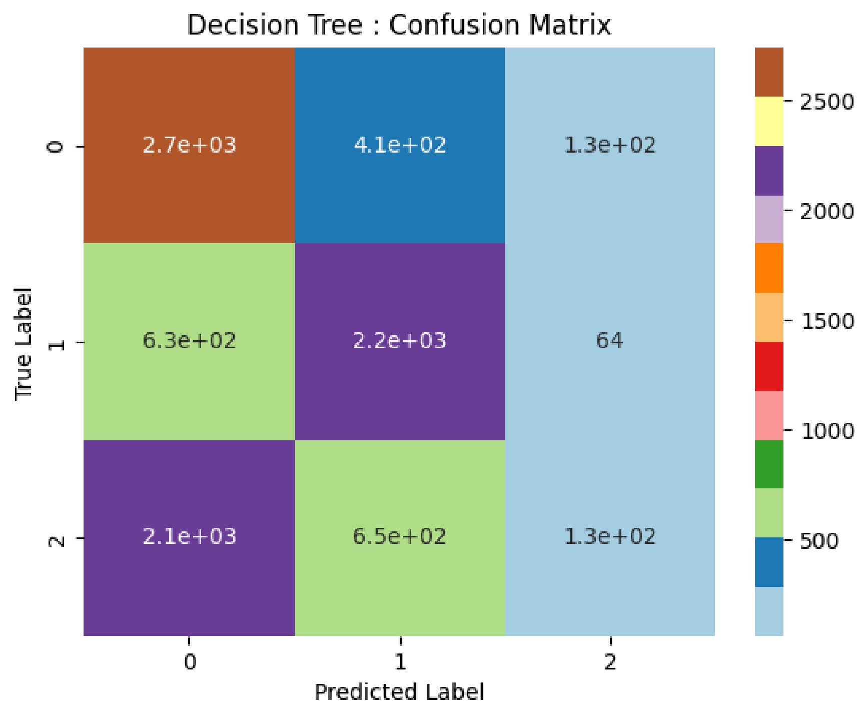
```
# Decision Tree : Prediction Evaluation (Testing Subset)
dtc_predict_conf_mat = pd.DataFrame(confusion_matrix(test_bank_output, dtc_predict)); print(dtc_predict_conf_mat)
dtc_predict_perf = classification_report(test_bank_output, dtc_predict); print(dtc_predict_perf)

```

```
      0     1    2
0  2737   409  132
1   632  2213   64
2  2078   648  130
              precision    recall  f1-score   support

         0.0       0.50      0.83      0.63      3278
         1.0       0.68      0.76      0.72      2909
         2.0       0.40      0.05      0.08      2856

    accuracy                           0.56      9043
   macro avg       0.53      0.55      0.48      9043
weighted avg       0.53      0.56      0.48      9043
```

```
# Decision Tree : Plot [Training Subset]
train_subset_dtc_plot = plot_tree(dtc_model, feature_names=bank_inputs_names, class_names=bank_output_labels, rounded=True, filled=True, fontsize=10)
plt.show()

```



```
# Confusion Matrix : Plot [Testing Subset]
ax = plt.axes()
sns.heatmap(dtc_predict_conf_mat, annot=True, cmap='Paired')
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Decision Tree : Confusion Matrix')
plt.show()

```



K-FOLD ANALYSIS

```
# We have to consider all the features as input variables.
bank_inputs = df_ppd[['job', 'marital', 'education','default','housing','loan','contact','day','month','poutcome','y',
                      'age','balance','duration','campaign','pdays','previous']]; bank_inputs

# Output variable - cluster
bank_output = df_ppd[['cluster']]; bank_output
```

```
       cluster
    0      1.0
    1      2.0
    2      0.0
    3      0.0
    4      2.0
  ...      ...
45206      2.0
45207      1.0
45208      1.0
45209      0.0
45210      0.0

45211 rows × 1 columns
```

```python
# saving the names/labels of all the input and output variables.

bank_inputs_names = bank_inputs.columns; print("Input Names: ",bank_inputs_names)
bank_output_labels = bank_output['cluster'].unique().astype(str); print("Output Label: ", bank_output_labels)
```

```
Input Names:  Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'day', 'month', 'poutcome', 'y', 'age', 'balance', 'duration',
       'campaign', 'pdays', 'previous'],
      dtype='object')
Output Label:  ['1.0' '2.0' '0.0']
```

```python
# Splitting the dataset into test & train
train_bank_inputs, test_bank_inputs, train_bank_output, test_bank_output = train_test_split(bank_inputs, bank_output, test_size=0.20, stratify=bank_o
```

```python
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import make_scorer, balanced_accuracy_score, precision_score, f1_score
from sklearn import preprocessing
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier

clf = make_pipeline(preprocessing.StandardScaler(), DecisionTreeClassifier(criterion='gini', random_state=45030, max_depth=3))

# Define a scorer for balanced accuracy
scorer = make_scorer(balanced_accuracy_score)

# Perform k-fold cross-validation for balanced accuracy
k_fold_balanced_accuracy = cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=scorer)

# Print the balanced accuracy for each fold
print("Balanced accuracy for each fold:", k_fold_balanced_accuracy)

# Calculate the average balanced accuracy
average_balanced_accuracy = np.mean(k_fold_balanced_accuracy)
print("Average balanced accuracy:", average_balanced_accuracy)

# Perform k-fold cross-validation for precision
precision_scorer = make_scorer(precision_score, average='weighted')
k_fold_precision = cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=precision_scorer)
print("\nPrecision for each fold:", k_fold_precision)
average_precision = np.mean(k_fold_precision)
print("Average precision:", average_precision)

# Perform k-fold cross-validation for F1 score
f1_scorer = make_scorer(f1_score, average='weighted')
k_fold_f1 = cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=f1_scorer)
print("\nCross Validation for each fold:", k_fold_f1)
average_f1 = np.mean(k_fold_f1)
print("Average Cross-Validation score:", average_f1)

```

```
Balanced accuracy for each fold: [1. 1. 1. 1. 1.]
Average balanced accuracy: 1.0

Precision for each fold: [1. 1. 1. 1. 1.]
Average precision: 1.0

Cross Validation for each fold: [1. 1. 1. 1. 1.]
Average Cross-Validation score: 1.0
```

```python
import pandas as pd

# Create a DataFrame to store the scores
results = pd.DataFrame({
    'Fold': range(1, 6),
    'Balanced Accuracy': k_fold_balanced_accuracy,
    'Precision': k_fold_precision,
    'Cross-Validation Score': k_fold_f1
})

# Calculate the average scores
average_scores = {
    '   Balanced Accuracy': average_balanced_accuracy,
    '   Precision': average_precision,
    '   Cross-Validation Score': average_f1
}

# Print the DataFrame with results
print("K-Fold Cross Validation Results:")
print(results)


print("\nAVERAGE SCORES:")
for metric, score in average_scores.items():
    print(f"{metric}: {score:.4f}")

```

```
K-Fold Cross Validation Results:
   Fold  Balanced Accuracy  Precision  Cross-Validation Score
0     1                1.0        1.0                     1.0
1     2                1.0        1.0                     1.0
2     3                1.0        1.0                     1.0
3     4                1.0        1.0                     1.0
4     5                1.0        1.0                     1.0

AVERAGE SCORES:
   Balanced Accuracy: 1.0000
   Precision: 1.0000
   Cross-Validation Score: 1.0000
```

```python
# Train the decision tree classifier
clf.fit(train_bank_inputs, train_bank_output)

# Get feature importances
importances = clf.named_steps['decisiontreeclassifier'].feature_importances_

# Create a DataFrame to display the importance of each feature
feature_importance_df = pd.DataFrame({'Feature': train_bank_inputs.columns, 'Importance': importances})

# Sort the DataFrame by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Display the feature importance
print(feature_importance_df)

```

```
        Feature  Importance
  0         job         1.0
  9    poutcome         0.0
  15      pdays         0.0
  14   campaign         0.0
  13   duration         0.0
  12    balance         0.0
  11        age         0.0
  10          y         0.0
  8       month         0.0
  1     marital         0.0
  7         day         0.0
  6     contact         0.0
  5        loan         0.0
  4     housing         0.0
  3     default         0.0
  2   education         0.0
  16   previous         0.0
```

```python
bank_inputs_new_dt = df_ppd[['marital', 'education','default','housing','loan','contact','day','month','poutcome','y',
                      'age','balance','duration','campaign','pdays','previous']];
bank_inputs_names = bank_inputs_new_dt.columns; print("Input Names: ",bank_inputs_names)
```

```
  Input Names:  Index(['marital', 'education', 'default', 'housing', 'loan', 'contact', 'day',
         'month', 'poutcome', 'y', 'age', 'balance', 'duration', 'campaign',
         'pdays', 'previous'],
        dtype='object')
```

```python
# Splitting the dataset into test & train
train_bank_inputs, test_bank_inputs, train_bank_output, test_bank_output = train_test_split(bank_inputs_new_dt, bank_output, test_size=0.20, stratify
```

```python
import time
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import make_scorer, balanced_accuracy_score, precision_score, f1_score
from sklearn import preprocessing
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier
from memory_profiler import memory_usage

# Start the timer
start_time = time.time()

clf = make_pipeline(preprocessing.StandardScaler(), DecisionTreeClassifier(criterion='gini', random_state=45030, max_depth=3))

# Define a scorer for balanced accuracy
scorer = make_scorer(balanced_accuracy_score)

# Perform k-fold cross-validation for balanced accuracy
k_fold_balanced_accuracy = cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=scorer)

# Print the balanced accuracy for each fold
print("Balanced accuracy for each fold:", k_fold_balanced_accuracy)

# Calculate the average balanced accuracy
average_balanced_accuracy = np.mean(k_fold_balanced_accuracy)
print("Average balanced accuracy:", average_balanced_accuracy)

# Perform k-fold cross-validation for precision
precision_scorer = make_scorer(precision_score, average='weighted')
k_fold_precision = cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=precision_scorer)
print("\nPrecision for each fold:", k_fold_precision)
average_precision = np.mean(k_fold_precision)
print("Average precision:", average_precision)

# Perform k-fold cross-validation for F1 score
f1_scorer = make_scorer(f1_score, average='weighted')
k_fold_f1 = cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=f1_scorer)
print("\nCross Validation for each fold:", k_fold_f1)
average_f1 = np.mean(k_fold_f1)
print("Average Cross-Validation score:", average_f1)

# Calculate the time taken
end_time = time.time()
time_taken = end_time - start_time
print("\n\nTime taken by k-fold cross-validation:", time_taken, "seconds")

def get_memory_usage():
    mem_usage = memory_usage(-1, interval=0.1)
    return max(mem_usage)

max_memory_usage = get_memory_usage()
print("Max memory usage:", max_memory_usage, "MB")

```

```
Balanced accuracy for each fold: [0.5376794  0.54440802 0.54140191 0.53925643 0.5443991 ]
Average balanced accuracy: 0.5414289723759836

Precision for each fold: [0.51899544 0.39494912 0.51215763 0.3936807  0.51765212]
Average precision: 0.4674870017106018

Cross Validation for each fold: [0.47620088 0.45887649 0.47517968 0.45504886 0.4810529 ]
Average Cross-Validation score: 0.46927176269738846


Time taken by k-fold cross-validation: 2.5566556453704834 seconds
Max memory usage: 278.1875 MB
```

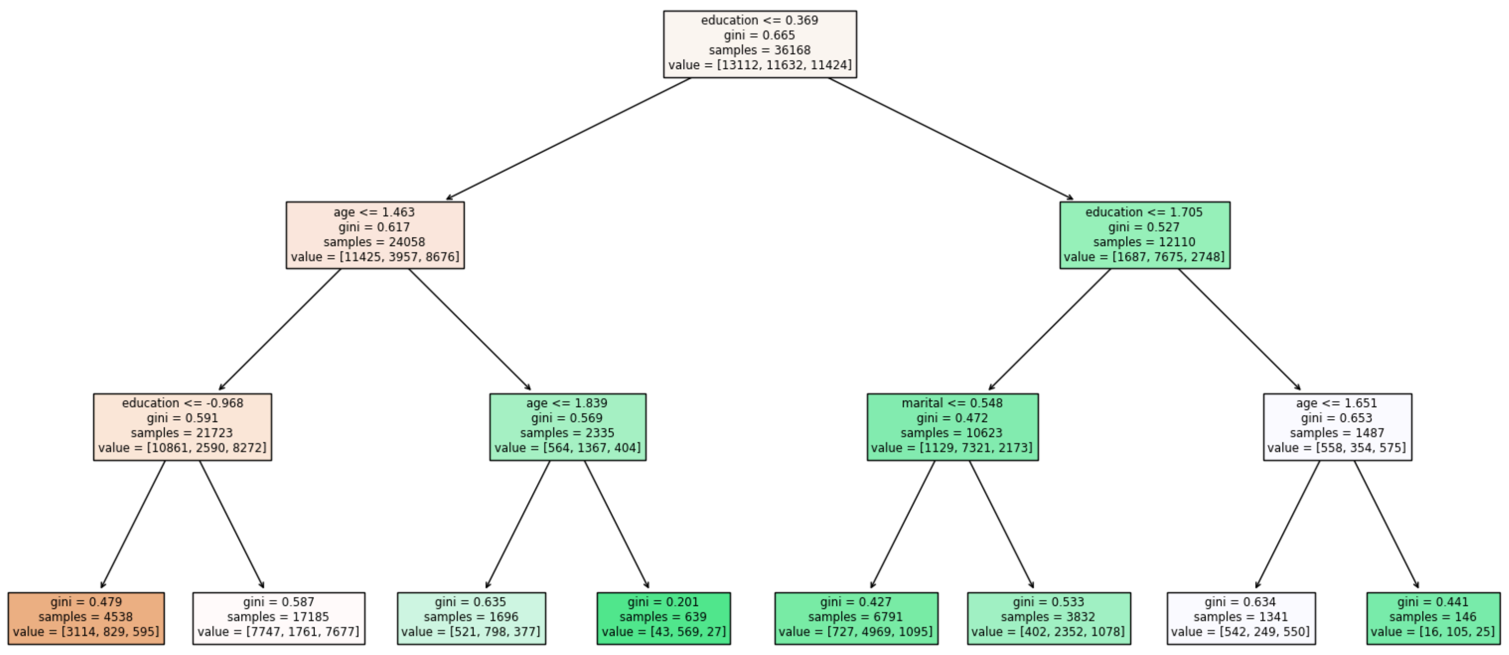
```python
import pandas as pd

# Create a DataFrame to store the scores
results = pd.DataFrame({
    'Fold': range(1, 6),
    'Balanced Accuracy': k_fold_balanced_accuracy,
    'Precision': k_fold_precision,
    'Cross-Validation Score': k_fold_f1
})

# Calculate the average scores
average_scores = {
    '   Balanced Accuracy': average_balanced_accuracy,
    '   Precision': average_precision,
    '   Cross-Validation Score': average_f1
}

# Print the DataFrame with results
print("K-Fold Cross Validation Results:")
print(results)

# Print the average scores
print("\nAVERAGE SCORES:")
for metric, score in average_scores.items():
    print(f"{metric}: {score:.4f}")

```

```
K-Fold Cross Validation Results:
   Fold  Balanced Accuracy  Precision  Cross-Validation Score
0     1           0.537679   0.518995                0.476201
1     2           0.544408   0.394949                0.458876
2     3           0.541402   0.512158                0.475180
3     4           0.539256   0.393681                0.455049
4     5           0.544399   0.517652                0.481053

AVERAGE SCORES:
   Balanced Accuracy: 0.5414
   Precision: 0.4675
   Cross-Validation Score: 0.4693
```

```python
from sklearn.model_selection import cross_val_score
import numpy as np

# Initialize an empty dictionary to store feature importances for each fold
feature_importances_per_fold = {feature: [] for feature in train_bank_inputs.columns}

# Define a custom scoring function to compute feature importance for each fold
def compute_feature_importances(clf, X, y):
    clf.fit(X, y)
    importances = clf.named_steps['decisiontreeclassifier'].feature_importances_
    for feature, importance in zip(X.columns, importances):
        feature_importances_per_fold[feature].append(importance)
    return 0.0  # Return a placeholder value

# Perform k-fold cross-validation and compute feature importances for each fold
cross_val_score(clf, train_bank_inputs, train_bank_output, cv=5, scoring=compute_feature_importances)

# Calculate the mean importance of each feature across all folds
mean_feature_importances = {feature: np.mean(importances) for feature, importances in feature_importances_per_fold.items()}

# Sort the mean feature importances in descending order
sorted_feature_importances = dict(sorted(mean_feature_importances.items(), key=lambda item: item[1], reverse=True))

# Display the mean importance of each feature in descending order
for feature, importance in sorted_feature_importances.items():
    print(f"{feature}: {importance}")

```

```
education: 0.7965132464491809
age: 0.19794831178477237
marital: 0.005538441766046787
default: 0.0
housing: 0.0
loan: 0.0
contact: 0.0
day: 0.0
month: 0.0
poutcome: 0.0
y: 0.0
balance: 0.0
duration: 0.0
campaign: 0.0
pdays: 0.0
previous: 0.0
```

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Fit the decision tree classifier to the data
clf.fit(train_bank_inputs, train_bank_output)

# Plot the decision tree
plt.figure(figsize=(20,10))
plot_tree(clf.named_steps['decisiontreeclassifier'], filled=True, feature_names=train_bank_inputs.columns)
plt.show()

```



RANDOM FOREST

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import time
from memory_profiler import memory_usage

def get_memory_usage():
    mem_usage = memory_usage(-1, interval=0.1)
    return max(mem_usage)

# Create and train the Random Forest classifier
start_time = time.time()
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=45030)
rf_classifier.fit(train_bank_inputs, train_bank_output)
end_time = time.time()

# Calculate the train and test scores
train_score = rf_classifier.score(train_bank_inputs, train_bank_output)
test_score = rf_classifier.score(test_bank_inputs, test_bank_output)

# Print the train and test scores
print("Train Score {:.2f} & Test Score {:.2f}".format(train_score, test_score))

rf_predictions = rf_classifier.predict(test_bank_inputs)
""
# Calculate the accuracy of the classifier
rf_accuracy = accuracy_score(test_bank_output, rf_predictions)
print("\n Random Forest Accuracy:", rf_accuracy)

time_taken = end_time - start_time
print("Time taken by Random Forest:", time_taken, "seconds")

max_memory_usage = get_memory_usage()
print("Max memory usage:", max_memory_usage, "MB")

```

```
Train Score 1.00 & Test Score 0.58

 Random Forest Accuracy: 0.581223045449519
Time taken by Random Forest: 17.022558212280273 seconds
Max memory usage: 451.1796875 MB
```

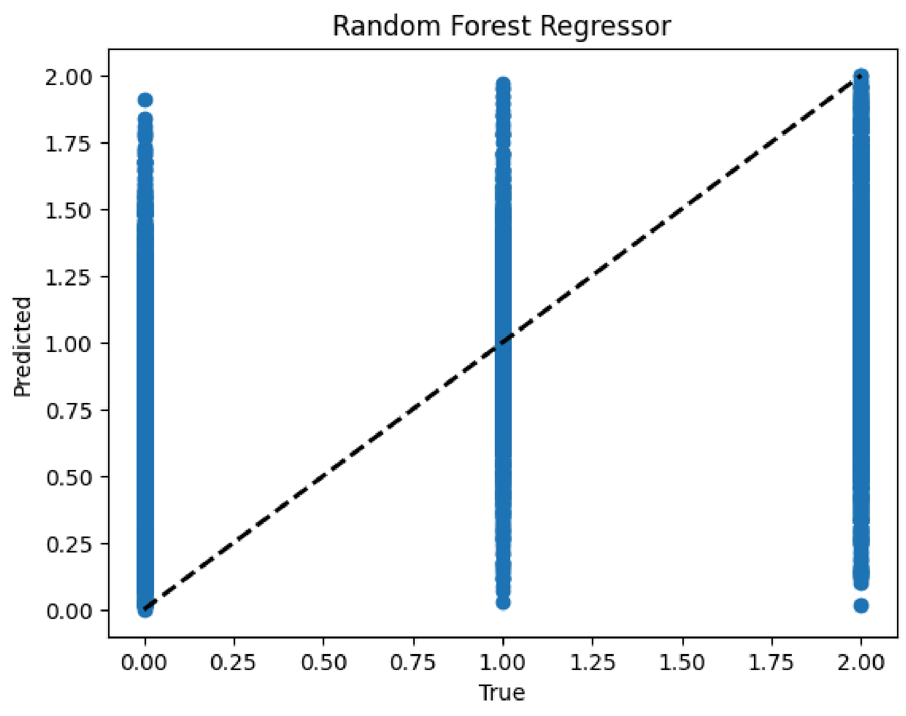
```python
print("Classification Report:")
print(classification_report(test_bank_output, rf_predictions))
```

```
Classification Report:
              precision    recall  f1-score   support
```

```
         0.0       0.56      0.62      0.59      3278
         1.0       0.67      0.74      0.70      2909
         2.0       0.49      0.38      0.42      2856

    accuracy                           0.58      9043
   macro avg       0.57      0.58      0.57      9043
weighted avg       0.57      0.58      0.57      9043
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Train the Random Forest Regressor
rfr = RandomForestRegressor(n_estimators=100, random_state=45030)
rfr.fit(train_bank_inputs, train_bank_output)

# Make predictions
y_pred_rfr = rfr.predict(test_bank_inputs)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(test_bank_output, y_pred_rfr))
mse = mean_squared_error(test_bank_output, y_pred_rfr)
mae = mean_absolute_error(test_bank_output, y_pred_rfr)
r2 = r2_score(test_bank_output, y_pred_rfr)

# Print evaluation metrics
print("Model\t\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")
print("Random Forest Regressor \t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}".format(rmse, mse, mae, r2))

# Plot predicted vs true values
plt.scatter(test_bank_output, y_pred_rfr)
plt.plot([test_bank_output.min(), test_bank_output.max()], [test_bank_output.min(), test_bank_output.max()], 'k--', lw=2)

plt.xlabel("True")
plt.ylabel("Predicted")

plt.title("Random Forest Regressor")

plt.show()

```

```
Model                            RMSE            MSE             MAE             R2
Random Forest Regressor          0.79            0.63            0.66            0.07
```

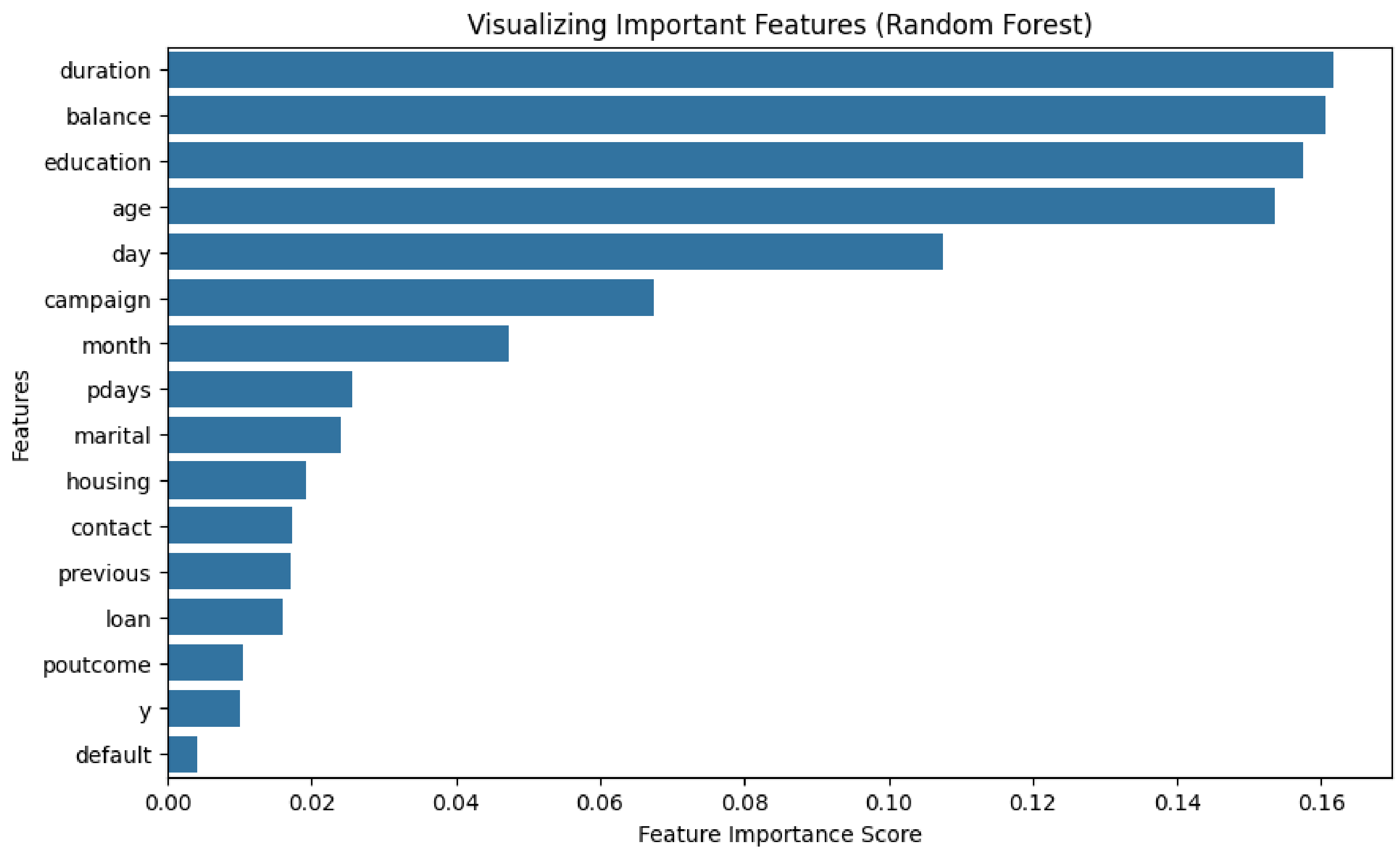


```
# Get feature importances
feature_importance = rf_classifier.feature_importances_

# Create a DataFrame to store feature importance along with feature names
feature_importance_df = pd.DataFrame({'Feature': train_bank_inputs.columns, 'Importance': feature_importance})

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print feature importance"
print("Feature Importance:")
print(feature_importance_df)

```

```
Feature Importance:
      Feature  Importance
12   duration    0.161675
11    balance    0.160574
1   education    0.157556
10        age    0.153608
6         day    0.107518
13   campaign    0.067370
7       month    0.047396
14      pdays    0.025646
0     marital    0.024170
3     housing    0.019308
5     contact    0.017295
15   previous    0.017036
4        loan    0.016061
8    poutcome    0.010488
9           y    0.010103
```

```
    2     default    0.004196
```

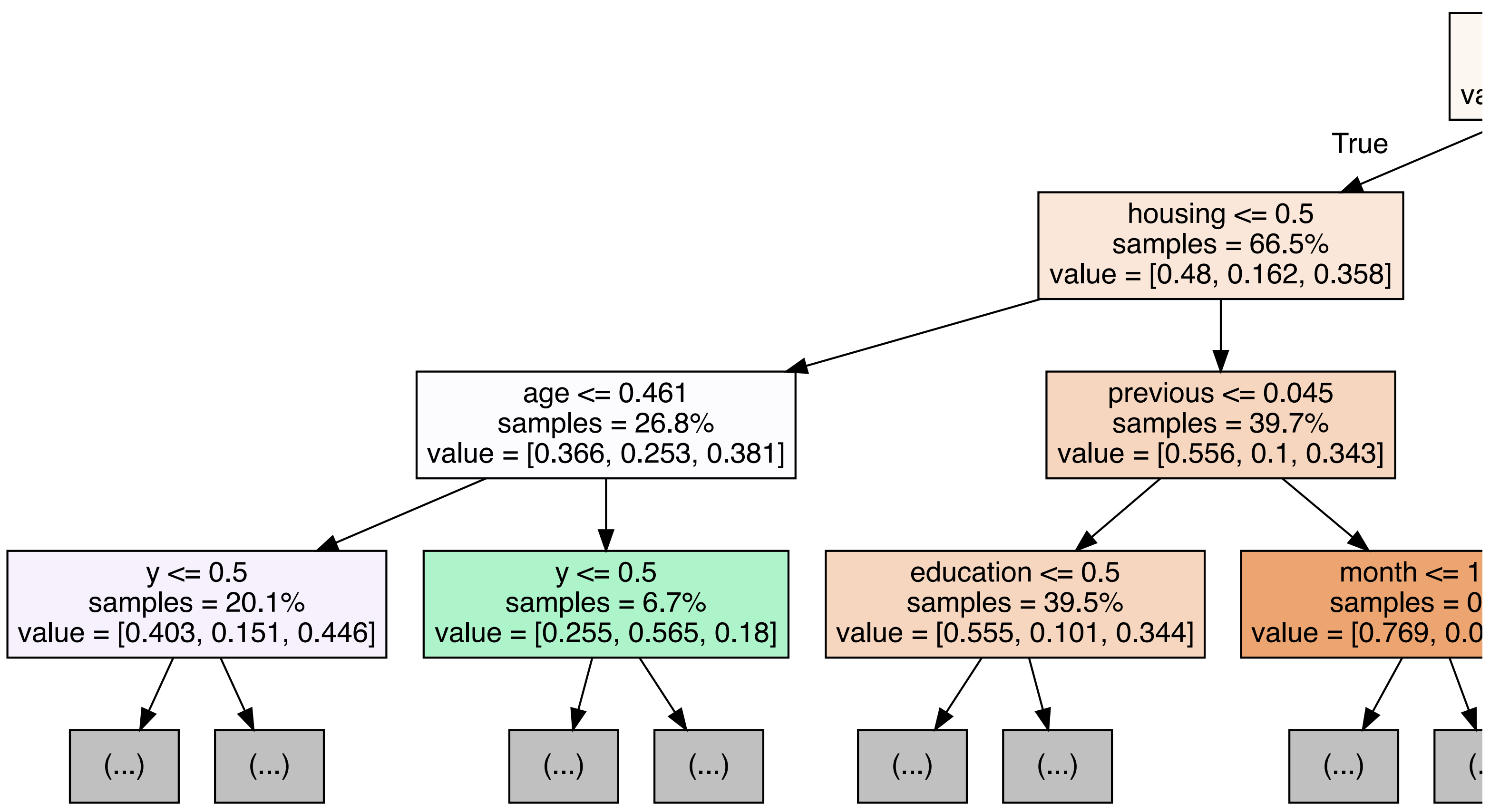
```
# Create a Series for feature importance with corresponding feature names
feature_imp = pd.Series(rf_classifier.feature_importances_, index=train_bank_inputs.columns).sort_values(ascending=False)

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features (Random Forest)")
plt.show()

```

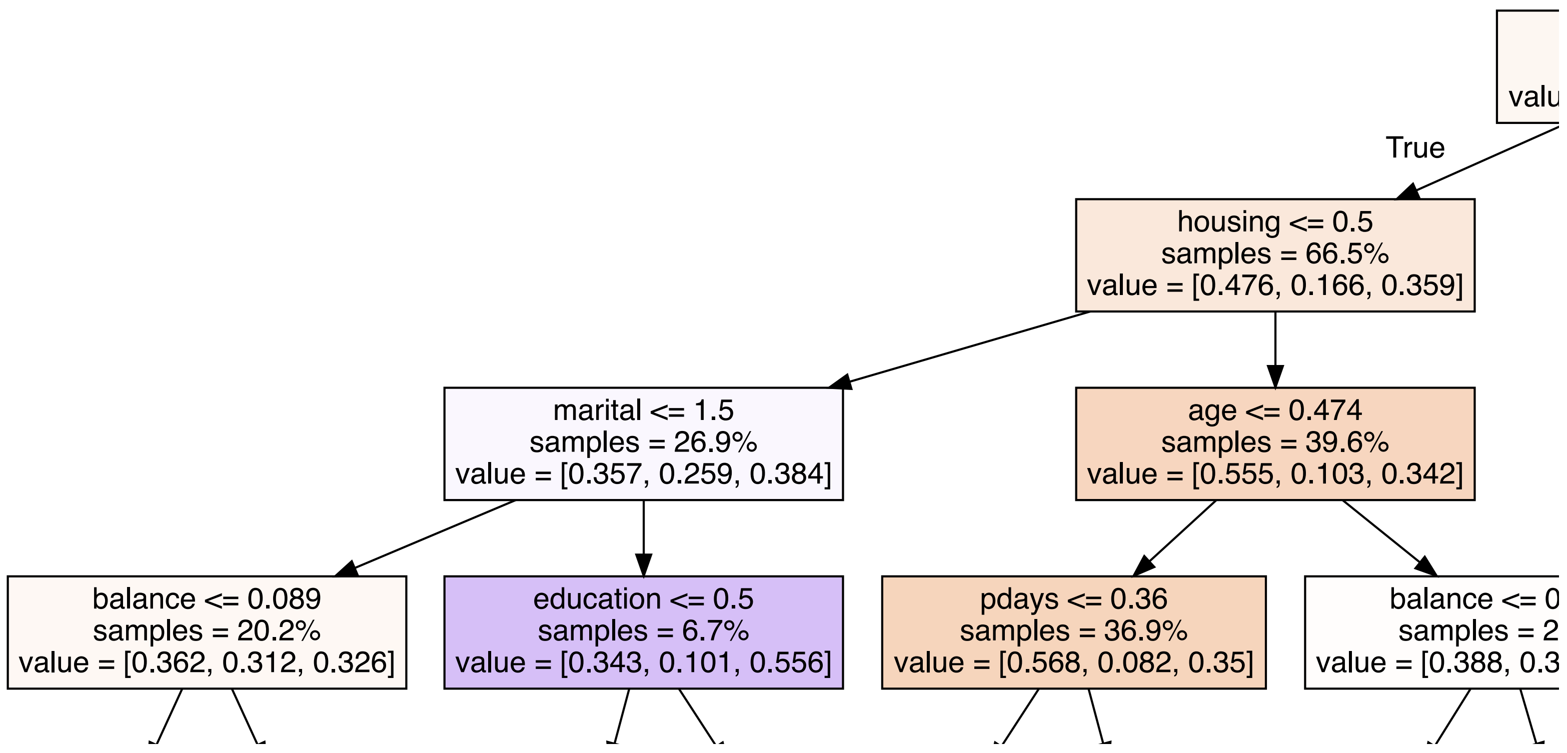


```
# Initialize a dictionary to store tree frequencies and rules
from collections import Counter
tree_frequency = Counter()
tree_rules = {}

for i in range(len(rfr.estimators_)):
    tree = rf_classifier.estimators_[i]
    tree_str = export_text(tree, feature_names=list(train_bank_inputs.columns))
    tree_frequency[tree_str] += 1
    tree_rules[tree_str] = tree

# Get the three most frequent trees
top_trees = tree_frequency.most_common(3)


for tree_str, frequency in top_trees:
    print(f"Tree Frequency: {frequency}")
    print(tree_str)
    print("\n")
```

```
Tree Frequency: 1
|--- education <= 1.50
|   |--- housing <= 0.50
|   |   |--- age <= 0.46
|   |   |   |--- y <= 0.50
|   |   |   |   |--- month <= 4.50
|   |   |   |   |   |--- education <= 0.50
|   |   |   |   |   |   |--- age <= 0.34
|   |   |   |   |   |   |   |--- campaign <= 0.01
|   |   |   |   |   |   |   |   |--- day <= 2.50
|   |   |   |   |   |   |   |   |   |--- day <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- age <= 0.18
|   |   |   |   |   |   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |   |   |   |   |   |--- age >  0.18
|   |   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |   |--- day >  1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |--- day >  2.50
|   |   |   |   |   |   |   |   |   |--- age <= 0.15
|   |   |   |   |   |   |   |   |   |   |--- day <= 8.00
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 3
|   |   |   |   |   |   |   |   |   |   |--- day >  8.00
|   |   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |   |--- age >  0.15
|   |   |   |   |   |   |   |   |   |   |--- day <= 6.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |--- day >  6.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |--- campaign >  0.01
|   |   |   |   |   |   |   |   |--- balance <= 0.07
|   |   |   |   |   |   |   |   |   |--- marital <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- day <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |   |   |--- day >  3.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |   |--- marital >  1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |   |   |--- balance >  0.07
|   |   |   |   |   |   |   |   |   |--- age <= 0.25
|   |   |   |   |   |   |   |   |   |   |--- duration <= 0.07
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |   |   |   |--- duration >  0.07
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |--- age >  0.25
|   |   |   |   |   |   |   |   |   |   |--- marital <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |--- marital >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 2.0
|   |   |   |   |   |   |--- age >  0.34
|   |   |   |   |   |   |   |--- marital <= 0.50
|   |   |   |   |   |   |   |   |--- month <= 3.50
|   |   |   |   |   |   |   |   |   |--- age <= 0.36
|   |   |   |   |   |   |   |   |   |   |--- class: 1.0
|   |   |   |   |   |   |   |   |   |--- age >  0.36
|   |   |   |   |   |   |   |   |   |   |--- campaign <= 0.07
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- campaign >  0.07
|   |   |   |   |   |   |   |   |   |   |   |--- class: 2.0
```

```python
# Initialize a dictionary to store tree frequencies
import graphviz
from sklearn.tree import export_graphviz
tree_frequency = Counter()

# Loop through the trees and count their frequency
for i in range(len(rfr.estimators_)):
    tree = rf_classifier.estimators_[i]
    tree_str = export_graphviz(tree, feature_names=train_bank_inputs.columns,
                               filled=True, max_depth=3, impurity=False, proportion=True)
    tree_frequency[tree_str] += 1

# Get the three most frequent trees
top_trees = tree_frequency.most_common(3)


for tree_str, frequency in top_trees:
    graph = graphviz.Source(tree_str)
    display(graph)
```





XGBOOST

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

```

```
import time
from memory_profiler import memory_usage
# Function to get memory usage
def get_memory_usage():
    mem_usage = memory_usage(-1, interval=0.1)
    return max(mem_usage)

# Start the timer
start_time = time.time()

xgb_classifier = xgb.XGBClassifier(n_estimators=100, random_state=45030)
xgb_classifier.fit(train_bank_inputs, train_bank_output)


# End the timer
end_time = time.time()
time_taken = end_time - start_time
print("Time taken by Random Forest:", time_taken, "seconds")

# Get memory usage
max_memory_usage = get_memory_usage()
print("Max memory usage:", max_memory_usage, "MB")
```

```
Time taken by Random Forest: 3.273855686187744 seconds
Max memory usage: 593.63671875 MB
```

```
train_score = xgb_classifier.score(train_bank_inputs, train_bank_output)
test_score = xgb_classifier.score(test_bank_inputs, test_bank_output)
print("Train Score: {:.2f} & Test Score: {:.2f}".format(train_score, test_score))

```

```
Train Score: 0.71 & Test Score: 0.60
```

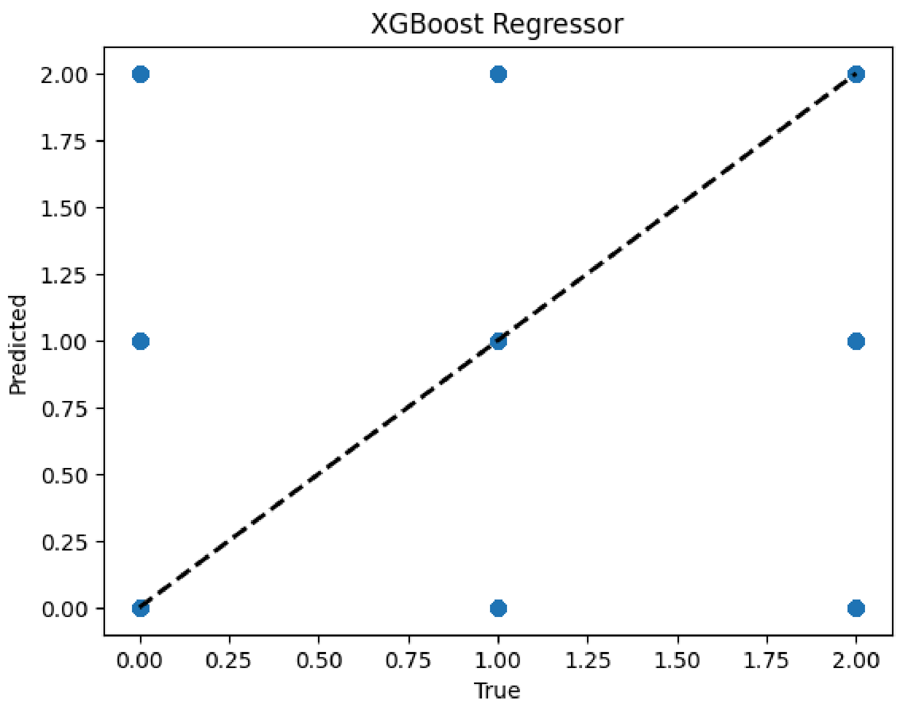
```
xgb_predictions = xgb_classifier.predict(test_bank_inputs)

```

```
xgb_accuracy = accuracy_score(test_bank_output, xgb_predictions)
print("XGBoost Accuracy:", xgb_accuracy)

print("Classification Report:")
print(classification_report(test_bank_output, xgb_predictions))
```

```
XGBoost Accuracy: 0.5952670573924582
Classification Report:
              precision    recall  f1-score   support

         0.0       0.57      0.65      0.61      3278
         1.0       0.68      0.75      0.71      2909
         2.0       0.51      0.37      0.43      2856

    accuracy                           0.60      9043
   macro avg       0.59      0.59      0.58      9043
weighted avg       0.59      0.60      0.59      9043
```

```
from xgboost import XGBClassifier

# Train the XGBoost Regressor
xgb = XGBClassifier(n_estimators=100, random_state=45030)
xgb.fit(train_bank_inputs, train_bank_output)

# Make predictions
y_pred_xgb = xgb.predict(test_bank_inputs)

# Calculate evaluation metrics
rmse_xgb = np.sqrt(mean_squared_error(test_bank_output, y_pred_xgb))
mse_xgb = mean_squared_error(test_bank_output, y_pred_xgb)
mae_xgb = mean_absolute_error(test_bank_output, y_pred_xgb)
r2_xgb = r2_score(test_bank_output, y_pred_xgb)

# Print evaluation metrics
print("Model\t\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")
print("XGBoost Regressor \t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}".format(rmse_xgb, mse_xgb, mae_xgb, r2_xgb))

plt.scatter(test_bank_output, y_pred_xgb)
plt.plot([test_bank_output.min(), test_bank_output.max()], [test_bank_output.min(), test_bank_output.max()], 'k--', lw=2)

plt.xlabel("True")
plt.ylabel("Predicted")

plt.title("XGBoost Regressor")

plt.show()

```

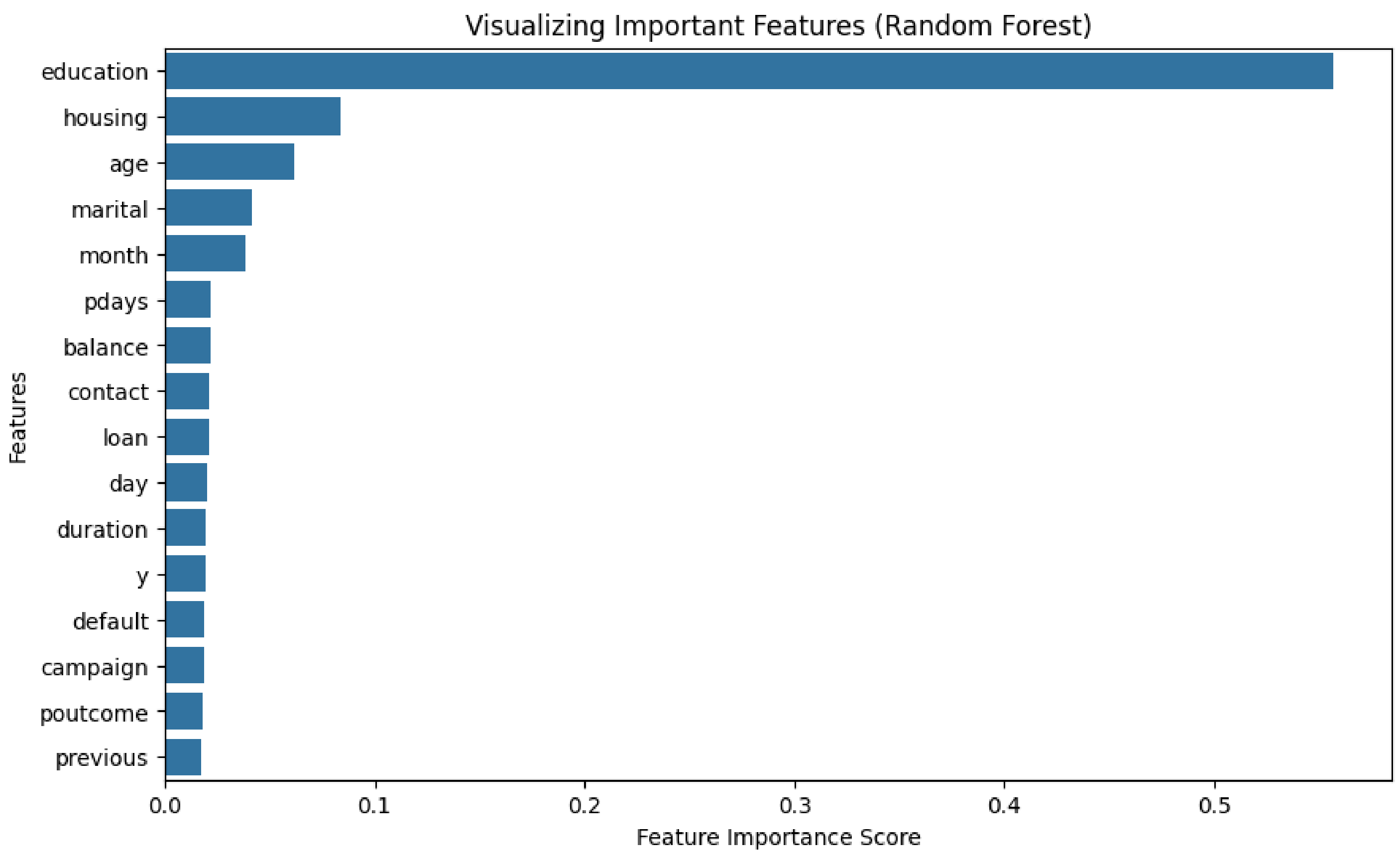
```
Model                          RMSE           MSE            MAE            R2
XGBoost Regressor        1.02           1.03           0.61           -0.53
```



```
feature_importance = xgb_classifier.feature_importances_

# Create a DataFrame to store feature importance along with feature names
feature_importance_df = pd.DataFrame({'Feature': train_bank_inputs.columns, 'Importance': feature_importance})

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print feature importance
print("Feature Importance:")
for idx, row in feature_importance_df.iterrows():
    print(f"{row['Feature']}: {row['Importance']:.5f}")
```

```
Feature Importance:
education: 0.55675
housing: 0.08368
age: 0.06168
marital: 0.04121
month: 0.03808
pdays: 0.02215
balance: 0.02159
contact: 0.02135
loan: 0.02104
day: 0.01999
duration: 0.01979
y: 0.01967
default: 0.01909
campaign: 0.01866
poutcome: 0.01811
previous: 0.01716
```

```
# Create a Series for feature importance with corresponding feature names
feature_imp = pd.Series(xgb_classifier.feature_importances_, index=train_bank_inputs.columns).sort_values(ascending=False)

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features (Random Forest)")
plt.show()

```



```python
import xgboost as xgb
import matplotlib.pyplot as plt


xgb.plot_tree(xgb_classifier, num_trees=0, rankdir='LR')

plt.rcParams['figure.figsize'] = [10, 10]

plt.show()

```