MACHINE LEARNING PROJECT: 2 (SUPERVISED LEARNING)

BANK MARKETING CLASSIFICATION COMPARITIVE ANALYSIS

Name: Megha Garg
Roll no. 045030

REPORT

1. OBJECTIVE OF THE PROJECT

The main objective of this project is to leverage supervised learning classification algorithms to analyze consumer data extracted from the banking marketing dataset. The goal is to segment customers into distinct clusters or classes based on various demographic and behavioral attributes. Through this analysis, the project aims to derive actionable insights that can inform targeted marketing strategies, enhance customer segmentation, and optimize resource allocation for improved engagement and conversion rates in banking marketing campaigns.

1. **Classification of Consumer Data into Segments:** Employ supervised learning classification algorithms to categorize consumer data into distinct segments or classes based on demographic information, banking history, and marketing interactions.

2. **Determination of an Appropriate Classification Model:** Evaluate multiple classification models, such as decision trees, logistic regression(LR), k-nearest neighbors (KNN), and support vector machines (SVM), to identify the most suitable one for accurately classifying consumer data in the context of your banking marketing dataset.

3. **Identification of Important Features for Classification:** Conduct feature selection or extraction techniques to identify the most relevant variables or features that significantly contribute to the classification of consumer data, enabling targeted marketing strategies and personalized services.

2. DATA DESCRIPTION

2.1. DATA SOURCE, SIZE, SHAPE

2.1.1 DATA SOURCE:

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

2.1.2 DATA SIZE:

Size of Dataset Used - 6.89 MB

Click here to jump to the code cell.

2.1.3 DATA DIMENSIONS:

Dataset consists of '19 columns and 45211 rows.
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.

Click here to jump to the code cell.

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.

2.3 DESCRIPTIVE STATISTICS

2.3.1 FREQUENCY DISTRIBUTION OF CATEGORICAL VARIABLES

These frequency statistics provide insights into the distribution of categorical variables within the dataset, which can inform targeted marketing strategies and customer segmentation efforts.

Click here to jump to the code cell.

1. **Job Distribution:**
   - The **most common** job categories among the dataset's respondents are **blue-collar, management, and technician, with frequencies of 9732, 9458, and 7597, respectively**.
   - The least common job category is unknown, with a frequency of 288.
2. **Marital Status:**
   - The majority of respondents are **married (27214), followed by single (12790) and divorced (5207)** individuals.
3. **Housing Status:**
   - Most respondents have **housing (25130)**, while a smaller portion do not (20081).
4. **Loan Status:**
   - The **majority** of respondents **do not have a loan (37967)**, while a smaller portion do (7244).
5. **Contact Type:**
   - The most common contact type is cellular (29285), followed by unknown (13020) and telephone (2906).
6. **Month of Last Contact:**
   - The most common month for the last contact is **May (13766)**, followed by July (6895) and August (6247).
7. **Outcome of Marketing Campaign (y):**
   - The majority of respondents did not subscribe to the term deposit (39922), while a smaller portion did (5289).
8. **Education Level:**
   - The most common education level among respondents is **secondary (23202), followed by tertiary (13301), primary (6851), and unknown (1857).**
9. **Day of Last Contact:**
   - The most common day for the last contact is the **20th (2752)**, followed by the 1
10. **Cluster:**
    - Most datapoints belongs to **cluster 0** followed by cluster 1 and then cluster 2.

RELATIVE FREQUENCY OF CATEGORICAL VARIABLES

Click here to jump to the code cell.

- **Job Distribution:** The most common job categories among the dataset's respondents are blue-collar (21.53%), management (20.92%), and technician (16.80%).The least common job category is unknown (0.64%).
- **Marital Staus:** The majority of respondents are married (60.19%), followed by single (28.29%) and divorced (11.52%) individuals.
- **Housing status:** Most respondents have housing (55.58%), while a smaller portion do not (44.42%).
- **Loan Status:** The majority of respondents do not have a loan (83.98%), while a smaller portion(16.0%) does.
- **Contact Type:** The most common contact type is cellular (64.77%), followed by unknown (28.80%) and telephone (6.43%).
- **Month of Last Contact:** The most common month for the last contact is May (30.45%), followed by July (15.25%) and August (13.82%).

- **Outcome of Markting Campaign (y):** The majority of respondents did not subscribe to the term deposit (88.30%), while a smaller portion did (11.7%).
- **Education Level:** The most common education level among respondents is secondary (51.32%), followed by tertiary (29.42%), primary (15.15%), and unknown (4.11).
- **Day of Last Contact:** The most common day for the last contact is the 20th (6.09%), followed by the 18th (5.11%) and the 21st (4.48%).

2.3.2 DESCRIPTIVE STATISTICS OF NON-CATEGORICAL VARIABLES

- Measure Of Central Tendency
- Measure Of Dispersion
- Correlation Statistics

These descriptive statistics provide insights into the distribution and variability of non-categorical variables in the dataset:

MEASURE OF CENTRAL TENDENCY AND DISPERSION

Click here to jump to the code cell.

- **Age:** The **average age** of the individuals is around **41 years**, with a standard deviation of approximately 10.6 years. The **ages range from 18 to 95 years**.
- **Balance:** The **mean balance amount is approximately 1362**, with a considerable **standard deviation of 3044**. The balance amounts vary widely, ranging from negative values (indicating debt) to a maximum of 102,127.
- **Duration:** The **mean duration** of the last contact in seconds is about **258 seconds**, with a standard deviation of approximately 257 seconds. The duration varies from 0 seconds to a maximum of 4918 seconds.
- **Campaign:** On **average, each individual was contacted about 2.76 times** during the campaign, with a standard deviation of around 3.1. The number of contacts ranges from 1 to a maximum of 63.
- **Pdays:** The average number of days since the client was last contacted is approximately 40 days, with a standard deviation of approximately 100 days. The values range from -1 (indicating the client was not previously contacted) to a maximum of 871 days.
- **Previous: On average, clients were contacted about 0.58 times** before the current campaign, with a standard deviation of around 2.3. The number of previous contacts varies widely, ranging from 0 to a maximum of 275.

CORRELATION MATRIX

Click here to jump to the code cell.

- **Age shows a very weak positive correlation** with **balance** (0.097783) and a weak negative correlation with pdays (-0.023758).
- **Balance exhibits a weak positive correlation** with **age** (0.097783) and a very weak positive correlation with previous (0.016674).
- **Duration demonstrates negligible correlations with other variables.**
- **Campaign** and **pdays** have **weak negative correlations** with **duration** (-0.084570 and -0.001565, respectively) and weak positive correlations with each other (0.454820).
- Previous displays very weak positive correlations with balance (0.016674) and pdays (0.454820).

P-VALUES OF CORRELATION MATRIX:

- **Age has significant correlations with balance (p-value = 0.0) and pdays (p-value = 0.0)**.
- **Balance has significant correlations with all** other variables except duration (p-values < 0.05).
- Campaign and pdays have significant correlations with each other (p-value = 0.0) and with all other variables (p-values < 0.05).
- **Previous** has significant correlations with **balance and pdays**(p-values < 0.05).

3. ANALYSIS OF DATA

3.1 DATA PRE-PROCESSING

3.1.1 Missing Data Statistics and Treatment.

Click here to jump to the code cell.

Data Imputing: Data imputation is a statistical technique used to fill in missing values in a dataset. When data is collected or recorded, it is common for some observations to have missing values due to various reasons such as human error, equipment failure, or simply incomplete data. Data imputation methods aim to estimate or predict the missing values based on the available data. This process helps to ensure that datasets are complete and suitable for analysis, as many statistical algorithms and machine learning models cannot handle missing values.

Since there are no missing values present in the selected dataset, there is no need for imputation procedures.
Also, no Duplicate values are present in the dataset.

3.1.2 Numerical Encoding of Categorical Variables or Features

Click here to jump to the code cell.

Need For Numeric Encoding:
This allows clustering algorithms to process and analyze the data effectively, as most algorithms work with numerical inputs. Encoding ensures that categorical variables contribute meaningfully to the clustering process by representing them as numerical values while preserving the inherent characteristics of the original data.
Ordinal encoding converts categorical variables into numerical values while preserving their ordinal relationships, assigning unique integers based on the order of categories. It's suitable for variables with inherent order, like "low," "medium," and "high," ensuring compatibility with machine learning algorithms.

Among the categorical variables, all entries, with the exception of the 'day' column, are alphanumeric in nature. The 'day' column, in contrast, comprises numerical values.

The categorical columns specified in categorical_columns are transformed into numerical representations, where each unique category is assigned an ordinal value based on its order of appearance in the dataset. All the categorical data is encoded into numerical form and as there are no missing values, there is no use of creating dummy data.

job: admin.: 0, blue-collar: 1, entrepreneur: 2, housemaid: 3, management: 4, retired: 5, self-employed: 6, services: 7, student: 8, technician: 9, unemployed: 10, unknown: 11

marital : divorced: 0, married: 1, single: 2

education: primary: 0, secondary: 1, tertiary: 2, unknown: 3

default: no: 0,yes: 1

Housing: no: 0, yes 1

loan: no:0, yes: 1

contact: cellular: 0, telephone: 1, unknown: 2

day: 1: 0, 2: 1, 3: 2, 4: 3, 5: 4, 6: 5, 7: 6, 8: 7, 9: 8, 10: 9, 11: 10, 12: 11, 13: 12, 14: 13, 15: 14, 16: 15, 17: 16, 18: 17, 19: 18, 20: 19, 21: 20, 22: 21, 23: 22, 24: 23, 25: 24, 26: 25, 27: 26, 28: 27, 29: 28, 30: 29, 31: 30

month: apr: 0, aug: 1, dec: 2, feb: 3, jan: 4, jul: 5, jun: 6, mar: 7, may: 8, nov: 9, oct: 10, sep: 11

poutcome: failure: 0, other: 1, success : 2, unknown:3

3.1.3 Data Transformation & Rescaling [Treatment of Outliers] for Non-caregorical Data

Need For Data Transformation & Rescaling:
To ensure that **numerical features have similar scales**, preventing bias in clustering algorithms. This enhances algorithm convergence, improves interpretability, and mitigates the influence of outliers, ultimately leading to more accurate and reliable clustering results.

Non-Categorical variables: 'age','balance','duration','campaign','pdays' and previous.

BOX PLOT

Click here to jump to the code cell.

- A box plot is a graphical representation used for visualizing the **distribution of non-categorical data and identifying potential outliers**. It consists of a box that spans the interquartile range (IQR), with a line representing the median. Outliers are data points that fall beyond the whiskers and are represented as individual points on the plot.
- The primary use of a box plot in data transformation and rescaling is to **assess the distribution of the data and detect any outliers that may skew the analysis.** This information is crucial for making informed decisions about whether to treat outliers through various methods such as removing, transforming, or imputing them, depending on the context of the analysis.
- Rescaling techniques such as **min-max scaling or normalization(0-1) can be applied to transform numerical data into a comparable range**. Box plots help in visualizing the effectiveness of these transformations in reducing the influence of outliers and improving the distribution of the data.
- Through boxplot it can be observed that number of outliers for each non-categorical variable (age, balance, duration, campaign, pdays, previous) are 487, 4729, 3235, 3064, 8257 and 8257 respectively.

OUTLIER TREATMENT USING MIN-MAX SCALING

Click here to jump to the code cell.

Min-Max scaling transforms numerical features to a uniform range between 0 and 1, ensuring consistent scaling across variables. This prevents dominance by features with larger ranges and facilitates fair comparison. It preserves relative differences, crucial for maintaining dataset integrity. Additionally, it can enhance model performance, especially for algorithms sensitive to feature scale, by mitigating bias towards larger-magnitude features.

3.1.4 DATA BIFURCATION: TRAINING AND TESTING SETS

Data bifurcation into training and testing sets is essential in machine learning for two main reasons:

Training Set: This subset is used to train the model by exposing it to labeled data. The model learns patterns and relationships from this data.

Testing Set: The testing set is kept separate from the training data and is used to evaluate the model's performance. By assessing the model's predictions on unseen data, we can determine how well it generalizes to new instances.

Splitting the data allows us to objectively assess the model's performance, prevent overfitting, tune parameters, and gain confidence in its deployment.

The 'train_test_split' function randomly divides the data, allocating 80% to training and 20% to testing. Additionally, stratify=bank_output ensures that the class distribution is preserved in both subsets, while random_state=45030 sets a seed for reproducibility.

Click here to jump to the code cell.

3.2 DATA ANALYSIS

3.2.1.1 Decision Tree Algorithm

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.

- 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.

Click here to jump to the code cell.

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.

  Click here to jump to the code cell.
- 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.

  Click here to jump to the code cell.

  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.

3.2.1.2 Logistic Regression | K Nearest Neighbour | Support Vector Machine

1. **Logistic Regression**

   It is a supervised machine learning algorithm used for binary classification tasks, where the goal is to predict the probability that an instance belongs to a particular class. It models the probability of the binary outcome using a logistic function, which maps the input features to the probability of the output belonging to one of the two classes.It differs from other algorithms in its modeling approach, interpretability, and assumptions about the relationship between features and outcomes.
   Click here to jump to the code cell.

   - lr = LogisticRegression(random_state=45030): Here, a logistic regression model object is initialized with a specified random state for reproducibility.
   - lr_model = lr.fit(train_bank_inputs, train_bank_output): This line trains the logistic regression model using the training data (train_bank_inputs and train_bank_output). The fit method fits the model to the training data.
   - lr_predict = lr_model.predict(test_bank_inputs): It predicts the output labels for the testing subset (test_bank_inputs) using the trained logistic regression model.
     Click here to jump to the code cell.

2. **K-Nearest Neighbor**

   The k-nearest neighbors (KNN) algorithm is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the k nearest data points to a given input data point based on a distance metric (such as Euclidean distance) in the feature space. For classification, the output is determined by a majority vote among the k nearest neighbors, while for regression, it's the average of the target values of those neighbors.
   Click here to jump to the code cell.

   - knn = KNeighborsClassifier(n_neighbors=k): This line creates a KNN classifier object knn with the specified number of neighbors k. Each iteration of the loop uses a different k value.
   - knn.fit(train_bank_inputs, train_bank_output): Trains the KNN classifier (knn) using the training data (train_bank_inputs as features and train_bank_output as labels).
   - knn_pred = knn.predict(test_bank_inputs): This line makes predictions on the test data (test_bank_inputs) using the trained KNN classifier (knn) and assigns the predicted labels to knn_pred.
     Click here to jump to the code cell.

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

   Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. In classification, SVM finds the optimal hyperplane that best separates different classes in the feature space by maximizing the margin between the classes. It can handle both linear and non-linear data using different kernel functions such as linear, polynomial, or radial basis function (RBF). SVM aims to maximize the margin between support vectors (data points closest to the decision boundary) and is effective in high-dimensional spaces.
   Click here to jump to the code cell.

   - from sklearn.svm import SVC: Imports the Support Vector Classification (SVC) class from the svm module of the sklearn library.
   - svm_model = SVC(kernel='linear', C=1): Creates an SVM model object with a linear kernel and a regularization parameter C=1. This sets up the SVM model to learn a linear decision boundary with regularization strength equal to 1.
   - svm_model.fit(train_bank_inputs, train_bank_output): Trains the SVM model on the training dataset train_bank_inputs with corresponding target labels train_bank_output. This line fits the model to the provided data, allowing it to learn patterns and relationships between input features and output labels.
   - svm_predict = svm_model.predict(test_bank_inputs): Makes predictions on the testing subset test_bank_inputs using the trained SVM model. This line generates predicted labels for the test data based on the learned patterns from the training data.
     Click here to jump to the code cell.

3.2.2.1.1 Model Performance Evaluation - Confusion Matrix (Decision Tree)

In supervised 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.

Accuracy: Proportion of correctly classified instances out of total instances.
Precision: Proportion of true positive predictions among all positive predictions.
Recall: Proportion of true positive predictions among all actual positive instances.
F1 Score: Harmonic mean of precision and recall, balancing both metrics.

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.

OBSERVATION:

```
      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
```

- Precision & Recall:
  - Class 0.0 has moderate precision (50%) and high recall (83%).
  - Class 1.0 shows relatively high precision (68%) and recall (76%).
  - Class 2.0 has low precision (40%) and very low recall (5%).
- F1-score:
  - F1-score for Class 0.0 and Class 1.0 indicates balanced performance.
  - Class 2.0 has a very low F1-score, suggesting poor classification.
- Accuracy & Macro Average:
  - The overall accuracy is 56%, indicating moderate classification performance.
  - Macro average precision, recall, and F1-score hover around 0.53, indicating fair performance across all classes.

3.2.2.1.2 Model Performance Evaluation - Time and 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.

Click here to jump to the code cell.

OBSERVATION: 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.

```
Time taken: 0.06112265586853027 seconds
Memory used: 294.86328125 MB
```

3.2.2.2.1 Model Performance Evaluation - Confusion Matrix (Logistic Regression | Support Vector Machine | K-Nearest Neighbor)

- **Logistic Regression:**

  Click here to jump to the code cell.

```
      0     1    2
0  2298   574  406
1   615  1726  568
2  1363   767  726
        precision    recall  f1-score   support

   0.0       0.54      0.70      0.61      3278
   1.0       0.56      0.59      0.58      2909
   2.0       0.43      0.25      0.32      2856

accuracy                           0.53      9043
macroavg       0.51      0.52      0.50      9043
 wtd avg       0.51      0.53      0.51      9043
```

OBSERVATION:

- Class 0 has the highest number of true positives (2298), followed by class 1 (1726), and class 2 (726). - Precision measures the proportion of true positive predictions among all positive predictions. Class 0 has the highest precision (0.54), followed by class 1 (0.56), and class 2 (0.43). - F1-score is the harmonic mean of precision and recall, providing a balanced measure of model performance. Class 0 has the highest F1-score (0.61), followed by class 1 (0.58), and class 2 (0.32). - The overall accuracy of the model is 0.53, indicating that 53% of all predictions are correct.

- **K-Nearest Neighbor:**

  Click here to jump to the code cell.

```
      0     1    2
0  2016   466  796
1   657  1861  391
2  1264   655  937

         precision    recall  f1-score   support

    0.0       0.51      0.62      0.56      3278
    1.0       0.62      0.64      0.63      2909
    2.0       0.44      0.33      0.38      2856

accuracy                          0.53      9043
macroavg       0.53      0.53      0.52      9043
 wtd avg       0.53      0.53      0.52      9043
```

OBSERVATION:

- Precision, recall, and F1-score metrics vary across classes, with class 0 having precision of 0.51, recall of 0.62, and F1-score of 0.56, class 1 with precision of 0.62, recall of 0.64, and F1-score of 0.63, and class 2 with precision of 0.44, recall of 0.33, and F1-score of 0.38.
- The overall accuracy of the model is 0.53, indicating that approximately 53% of instances are correctly classified.
- Macro and weighted averages of precision, recall, and F1-score are around 0.53, indicating the overall performance of the model across all classes.

Click here to jump to the code cell.

```
Accuracy for k=7: 0.5415238305871946
Accuracy for k=9: 0.5473847174610196
Accuracy for k=11: 0.5486011279442663
Accuracy for k=13: 0.5525821077076192
```

OBSERVATION:

The accuracy of the K-Nearest Neighbors (KNN) algorithm was evaluated with different values of k. Starting from k=7 with an accuracy of approximately 54.15%, the accuracy increased gradually with higher values of k. At k=13, the accuracy reached its highest point at around 55.25%. This observation suggests that a larger number of neighbors considered in the algorithm contributes to improved predictive performance.

- **Support Vector Machine (SVM):**

  Click here to jump to the code cell.

```
      0     1    2
0  2289   596  393
1   625  2134  150
2  1354   864  638
              precision    recall  f1-score   support

         0.0       0.54      0.70      0.61      3278
         1.0       0.59      0.73      0.66      2909
         2.0       0.54      0.22      0.32      2856

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

OBSERVATION:

- From the confusion matrix, it can be observed that the SVM algorithm achieved an accuracy of 56%.
- Class 0 has the highest precision and recall values, indicating that it was well predicted.
- However, Class 2 exhibits lower precision and recall, suggesting that it was more challenging to classify accurately. Overall, while the model achieved a reasonable accuracy, there is room for improvement in predicting Class 2.

3.2.2.2.2 Model Performance Evaluation - Time and Memory Statistics (Logistic Regression | Support Vector Machine | K Nearest Neighbour)

- 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.

LOGISTIC REGRESSION:

Click here to jump to the code cell.

```
Time taken: 1.0147788524627686 seconds
Memory used: 590.57421875 MB
```

K-NEAREST NEIGHBOR:

Click here to jump to the code cell.

```
Time taken: 0.011163473129272461 seconds
Memory used: 591.38671875 MB
```

SVM:

Click here to jump to the code cell.

```
Time taken: 183.97964334487915 seconds
Memory used: 819.13671875 MB
```

OBSERVATION:

It is evident that Logistic Regression and K-Nearest Neighbors (KNN) algorithms are relatively faster and consume less memory compared to Support Vector Machine (SVM). Logistic Regression took approximately 1 second with memory usage around 590.57 MB, while KNN was significantly faster, taking only about 0.01 seconds with similar memory consumption. In contrast, SVM took considerably longer, around 184 seconds, and utilized more memory, approximately 819.14 MB. Therefore, in terms of computational efficiency and memory utilization, Logistic Regression and KNN outperform SVM.

3.2.3.1 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.

- 3.2.3.1.1. List of Relevant or Important Variables or Features and their Thresholds

  Click here to jump to the code cell.

```
        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.

- 3.2.3.1.2. List of Irrelevant or Non-Important Variables or Features

```
   feature    importance
  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
```

  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.

3.2.3.2. Variable or Feature Analysis (Logistic Regression | Support Vector Machine | K Nearest Neighbour)

3.2.3.2.1. List of Relevant or Important Variables or Features and their Thresholds.

- **LOGISTIC REGRESSION:**

  Click here to jump to the code cell.

```
      feature  coefficient  abs_coefficient
10        age    -0.820251         0.820251
1   education    -0.643992         0.643992
14      pdays     0.490540         0.490540
3     housing     0.300373         0.300373
9           y    -0.129067         0.129067
4        loan     0.129060         0.129060
5     contact     0.103359         0.103359
2     default     0.102407         0.102407
```

  OBSERVATION:

  Features such as 'age', 'education', 'pdays', 'housing', 'y', 'loan', 'contact', and 'default' have coefficients above the threshold, indicating their significance in determining the target variable. These features contribute significantly to the model's predictive power and decision-making process. Their coefficients signify the strength and direction of their influence on the outcome, with negative coefficients implying a negative correlation and positive coefficients indicating a positive correlation with the target variable.

- **K-NEAREST NEIGHBOR:**

  Click here to jump to the code cell.

```
      feature  accuracy_change  abs_accuracy_change
1   education         0.116775             0.116775
6         day        -0.015592             0.015592
10        age         0.012496             0.012496
9           y        -0.006524             0.006524
12   duration        -0.003981             0.003981
0     marital         0.002875             0.002875
3     housing        -0.002875             0.002875
5     contact        -0.002875             0.002875
14      pdays         0.002654             0.002654
```

  OBSERVATION:

  Features such as 'education', 'day', 'age', 'y', 'duration', 'marital', 'housing', 'contact', and 'pdays' exhibit significant impacts on accuracy, as indicated by their absolute accuracy change values exceeding the threshold. Notably, 'education' demonstrates the most substantial influence on accuracy, followed by 'day' and 'age'. These features play crucial roles in determining the model's predictive performance, highlighting their importance in accurately predicting the target variable.

- **SUPPORT VECTOR MACHINE (SVM):**

  Click here to jump to the code cell.

```
      feature  coefficient  abs_coefficient
10        age    -3.443606         3.443606
11    balance    -1.616925         1.616925
1   education    -1.489817         1.489817
14      pdays     0.717645         0.717645
3     housing     0.425993         0.425993
15   previous    -0.405033         0.405033
4        loan     0.201516         0.201516
0     marital    -0.106145         0.106145
```

  OBSERVATION:

  These selected features have coefficients indicating their influence on the target variable. Features such as 'age', 'balance', and 'education' exhibit relatively higher coefficients, suggesting stronger impact on the target. Conversely, features like 'marital' and 'loan' have coefficients closer to zero, indicating lesser influence. The absolute coefficient values highlight the magnitude of impact, with higher absolute values signifying greater importance.

3.2.3.2.2. List of Irrelevant or Non-Important Variables or Features

- **LOGISTIC REGRESSION:**

  Click here to jump to the code cell.

```
     feature  coefficient  abs_coefficient
12  duration     0.086105         0.086105
13  campaign     0.049499         0.049499
0    marital    -0.035187         0.035187
8   poutcome    -0.020067         0.020067
7      month     0.015740         0.015740
11   balance    -0.007260         0.007260
15  previous     0.006640         0.006640
6        day    -0.004239         0.004239
```

OBSERVATION:

Features such as 'duration', 'campaign', and 'marital' exhibit relatively small coefficients, suggesting less influence on the model's predictions. Conversely, 'month' and 'balance' have coefficients closer to zero, indicating minimal impact on the model's output. These observations imply that certain features contribute less to the predictive power of the model and may be considered less significant in making accurate predictions.

- **K-NEAREST NEIGHBOR:**

  Click here to jump to the code cell.

```
     feature  accuracy_change  abs_accuracy_change
8   poutcome        -0.002322             0.002322
4       loan        -0.001438             0.001438
11   balance        -0.001327             0.001327
13  campaign         0.001327             0.001327
7      month        -0.000885             0.000885
2    default         0.000332             0.000332
15  previous         0.000221             0.000221
```

OBSERVATION:
The listed features show minimal changes in accuracy, with 'poutcome', 'loan', and 'balance' exhibiting relatively small variations. Conversely, 'campaign' has a slightly higher impact on accuracy. Features such as 'month', 'default', and 'previous' contribute negligibly to accuracy changes. These observations suggest that these variables have limited influence on the model's predictive performance and may not significantly affect its accuracy.

- **SUPPORT VECTOR MACHINE (SVM):**

  Click here to jump to the code cell.

```
     feature  coefficient  abs_coefficient
5    contact     0.099003         0.099003
9          y    -0.098059         0.098059
13  campaign     0.060813         0.060813
8   poutcome     0.039008         0.039008
12  duration    -0.028575         0.028575
2    default     0.016498         0.016498
7      month     0.016070         0.016070
6        day    -0.003112         0.003112
```

OBSERVATION:

The listed features exhibit coefficients below the predefined threshold, suggesting they have minimal importance in the model. Notably, 'contact' and 'y' display relatively higher coefficients but still fall below the threshold. 'Campaign' and 'poutcome' have moderate coefficients, while 'duration', 'default', 'month', and 'day' contribute minimally to the model's predictive ability. These observations indicate that these variables may not significantly influence the model's output and could potentially be disregarded in feature selection processes.

4. RESULTS | OBSERVATIONS

4.1. Classification Model Parameters

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.

- **LOGISTIC REGRESSION:**

  Click here to jump to the code cell.

```
      0     1    2
0  2298   574  406
1   615  1726  568
2  1363   767  726
              precision    recall  f1-score   support

         0.0       0.54      0.70      0.61      3278
         1.0       0.56      0.59      0.58      2909
         2.0       0.43      0.25      0.32      2856

    accuracy                           0.53      9043
   macro avg       0.51      0.52      0.50      9043
weighted avg       0.51      0.53      0.51      9043
```

OBSERVATION:

The confusion matrix shows the performance of the logistic regression model across different classes:

- Class 0 has 2298 true negatives, 574 false positives, and 406 false negatives.
- Class 1 has 1726 true negatives, 615 false positives, and 568 false negatives.
- Class 2 has 726 true negatives, 1363 false positives, and 767 false negatives.

The precision, recall, and F1-score for each class indicate the model's performance:

- Class 0 has a precision of 0.54, recall of 0.70, and F1-score of 0.61.
- Class 1 has a precision of 0.56, recall of 0.59, and F1-score of 0.58.
- Class 2 has a precision of 0.43, recall of 0.25, and F1-score of 0.32.

The overall accuracy of the model is 0.53, indicating the proportion of correct predictions out of all predictions made. The macro avg and weighted avg provide the average precision, recall, and F1-score across all classes. In this case, both macro avg and weighted avg are close, indicating a balanced dataset.

- **K-NEAREST NEIGHBOR:**

  Click here to jump to the code cell.

```
      0     1    2
0  2016   466  796
1   657  1861  391
2  1264   655  937
              precision    recall  f1-score   support

         0.0       0.51      0.62      0.56      3278
        1.0       0.62      0.64      0.63      2909
        2.0       0.44      0.33      0.38      2856

    accuracy                           0.53      9043
  macro avg       0.53      0.53      0.52      9043
weighted avg       0.53      0.53      0.52      9043
```

OBSERVATION:

The confusion matrix for k-nearest neighbors (KNN) reveals the model's performance across different classes:

- Class 0 has 2016 true negatives, 466 false positives, and 796 false negatives.
- Class 1 has 1861 true negatives, 657 false positives, and 391 false negatives.
- Class 2 has 937 true negatives, 1264 false positives, and 655 false negatives.

The precision, recall, and F1-score for each class indicate the model's performance:

- Class 0 has a precision of 0.51, recall of 0.62, and F1-score of 0.56.
- Class 1 has a precision of 0.62, recall of 0.64, and F1-score of 0.63.
- Class 2 has a precision of 0.44, recall of 0.33, and F1-score of 0.38.

The overall accuracy of the KNN model is 0.53, representing the proportion of correct predictions out of all predictions made. Both the macro avg and weighted avg provide the average precision, recall, and F1-score across all classes, showing consistency in performance evaluation.

- **SUPPORT VECTOR MACHINE(SVM):**

  Click here to jump to the code cell.

```
      0     1    2
0  2289   596  393
1   625  2134  150
2  1354   864  638
              precision    recall  f1-score   support

         0.0       0.54      0.70      0.61      3278
         1.0       0.59      0.73      0.66      2909
         2.0       0.54      0.22      0.32      2856

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

OBSERVATION:

The confusion matrix for Support Vector Machine (SVM) illustrates its performance across different classes:

- Class 0 has 2289 true negatives, 596 false positives, and 393 false negatives.
- Class 1 has 2134 true negatives, 625 false positives, and 150 false negatives.
- Class 2 has 638 true negatives, 1354 false positives, and 864 false negatives.

The precision, recall, and F1-score for each class showcase the model's performance:

- Class 0 has a precision of 0.54, recall of 0.70, and F1-score of 0.61.
- Class 1 has a precision of 0.59, recall of 0.73, and F1-score of 0.66.
- Class 2 has a precision of 0.54, recall of 0.22, and F1-score of 0.32.

The overall accuracy of the SVM model is 0.56, indicating the proportion of correct predictions out of all predictions made. Both the macro avg and weighted avg provide the average precision, recall, and F1-score across all classes, demonstrating consistency in performance evaluation.

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: 0.038233041763305664 seconds
Memory used: 272.6875 MB
```

OBSERVATION:

- The decision tree model is relatively fast to train and make predictions.
- The memory usage is moderate, consuming 272.6875 MB, which indicates that decision trees are generally lightweight models suitable for deployment in memory-constrained environments.

- **LOGISTIC REGRESSION:**

  Click here to jump to the code cell.

  ```
  Time taken: 1.0147788524627686 seconds
  Memory used: 590.57421875 MB
  ```

  OBSERVATION:

  - Logistic regression takes considerably more time compared to decision trees, with a training time of 1.015 seconds.
  - The memory usage is relatively high, consuming 590.57 MB, indicating that logistic regression may require more computational resources compared to decision trees.
  - These time and memory statistics suggest that logistic regression may be suitable for datasets of moderate size but may not be as efficient for larger datasets or real-time applications.

- **K-NEAREST NEIGHBOR (KNN):**

  Click here to jump to the code cell.

  ```
  Time taken: 0.011163473129272461 seconds
  Memory used: 591.38671875 MB
  ```

  OBSERVATION:

  - K-Nearest Neighbor (KNN) demonstrates a relatively low training time of 0.011 seconds, suggesting its efficiency for model training.
  - However, the memory usage is relatively high, consuming 591.39 MB, indicating that KNN may require substantial memory resources, particularly as the dataset size increases.
  - While KNN offers fast training time, its memory usage may limit its scalability, especially for large datasets. Consideration should be given to memory constraints when deploying KNN models in memory-limited environments.

- **SUPPORT VECTOR MACHINE (SVM):**

  Click here to jump to the code cell.

  ```
  Time taken: 183.97964334487915 seconds
  Memory used: 819.13671875 MB
  ```

  OBSERVATION:

  - Support Vector Machine (SVM) exhibits a considerably longer training time of 183.98 seconds compared to other models, indicating relatively slower performance.
  - The memory usage is also relatively high, consuming 819.14 MB, suggesting that SVM may demand significant memory resources, particularly for large datasets.
  - The longer training time and higher memory usage of SVM could potentially limit its scalability, especially for large datasets or memory-constrained environments.

4.3. Variable or Feature Analysis

Variable or feature analysis in supervised machine learning 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. Click here to jump to the code cell.

- **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.

- **LOGISTIC REGRESSION:**

  Click here to jump to the code cell.

    - Relevant or Important Variables

```
        feature  coefficient  abs_coefficient
10          age    -0.820251         0.820251
1     education    -0.643992         0.643992
14        pdays     0.490540         0.490540
3       housing     0.300373         0.300373
9             y    -0.129067         0.129067
4          loan     0.129060         0.129060
5       contact     0.103359         0.103359
2       default     0.102407         0.102407
```

    OBSERVATION:

    All the features listed have coefficients above the threshold, indicating their relevance in predicting the target variable according to logistic regression.

        - Age and Education: These features have the highest absolute coefficients, indicating their significant impact on the logistic regression model's predictions. Negative coefficients for age and education suggest a negative correlation with the target variable.
        - Pdays and Housing: These features also exhibit notable absolute coefficients, suggesting a moderate impact on the model's predictions. Positive coefficients for pdays and housing indicate a positive correlation with the target variable.
        - Y, Loan, Contact, Default: These features have relatively lower absolute coefficients compared to the others, indicating a lesser impact on the model's predictions. However, they still contribute to the predictive power of the logistic regression model.

    - Irrelevant or Non-Important Variables

```
        feature  coefficient  abs_coefficient
12     duration     0.086105         0.086105
13     campaign     0.049499         0.049499
0       marital    -0.035187         0.035187
8      poutcome    -0.020067         0.020067
7         month     0.015740         0.015740
11      balance    -0.007260         0.007260
15     previous     0.006640         0.006640
6           day    -0.004239         0.004239
```

    OBSERVATION:

    While these features may still contribute to the logistic regression model's performance, their coefficients indicate that they have less influence on predicting the target variable compared to other features with higher absolute coefficients.

        - Marital Status and Day: These features have relatively low absolute coefficients, suggesting a minimal impact on the logistic regression model's predictions. The coefficients are close to zero, indicating little to no correlation with the target variable.

    - Balance and Previous: Although these features have small absolute coefficients, they still contribute slightly to the model's predictions. However, their impact is negligible compared to other more influential features.
    - Duration, Campaign, Poutcome, Month: Despite having coefficients above the threshold, these features are considered less important as their absolute coefficients are relatively low. Their impact on the model's predictions is minor compared to other more significant features.

- **K-NEAREST NEIGHBOR:**

  Click here to jump to the code cell.

  - Relevant or Important Variables

    ```
            feature  accuracy_change  abs_accuracy_change
    1   education         0.116775             0.116775
    6         day        -0.015592             0.015592
    10        age         0.012496             0.012496
    9           y        -0.006524             0.006524
    12   duration        -0.003981             0.003981
    0     marital         0.002875             0.002875
    3     housing        -0.002875             0.002875
    5     contact        -0.002875             0.002875
    14      pdays         0.002654             0.002654
    ```

    OBSERVATION:

    - Education: This feature has the highest positive impact on accuracy change, indicating that changes in education level significantly affect the model's accuracy.
    - Day and Age: While day and age also contribute to accuracy change, their impact is relatively small compared to education.
    - Marital Status, Housing, Contact, and Pdays: These features have minimal impact on accuracy change, with coefficients close to zero. Changes in these features are unlikely to significantly affect the model's accuracy.

  - Irrelevant or Non-Important Variables

    ```
            feature  accuracy_change  abs_accuracy_change
    8    poutcome        -0.002322             0.002322
    4        loan        -0.001438             0.001438
    11    balance        -0.001327             0.001327
    13   campaign         0.001327             0.001327
    7       month        -0.000885             0.000885
    2     default         0.000332             0.000332
    15   previous         0.000221             0.000221
    ```

    OBSERVATION:

    - Poutcome, Loan, and Month: These features exhibit negligible impact on accuracy change, with coefficients very close to zero. Changes in these features are unlikely to affect the model's accuracy significantly.
    - Balance: Despite having a small negative coefficient, the impact of balance on accuracy change is minimal.
    - Campaign and Default: These features have small positive coefficients, indicating a slight impact on accuracy change. However, the effect is relatively minor compared to other features.
    - Previous: This feature has the smallest positive coefficient, suggesting a minimal influence on accuracy change.

- **SUPPORT VECTOR MACHINE (SVM):**

  Click here to jump to the code cell.

  - Relevant or Important Variables

    ```
            feature  coefficient  abs_coefficient
    10        age    -3.443606         3.443606
    11    balance    -1.616925         1.616925
    1   education    -1.489817         1.489817
    14      pdays     0.717645         0.717645
    3     housing     0.425993         0.425993
    15   previous    -0.405033         0.405033
    4        loan     0.201516         0.201516
    0     marital    -0.106145         0.106145
    ```

    OBSERVATION:

    - Age, Balance, and Education: These features have the highest absolute coefficients, indicating their significant impact on the SVM model's predictions. Age and education exhibit negative coefficients, suggesting a negative correlation with the target variable, while balance has a positive coefficient, indicating a positive correlation.
    - Pdays and Housing: These features also have relatively high absolute coefficients, indicating notable influence on the model's predictions. Pdays has a positive coefficient, suggesting a positive correlation, while housing exhibits a similar pattern.

        - Previous, Loan, and Marital: Although these features have coefficients with smaller magnitudes compared to others, they still contribute to the SVM model's predictions. Previous and marital status exhibit negative coefficients, indicating a negative correlation, while loan shows a positive correlation.

    - Irrelevant or Non-Important Variables

```
        feature  coefficient  abs_coefficient
5       contact     0.099003         0.099003
9             y    -0.098059         0.098059
13     campaign     0.060813         0.060813
8      poutcome     0.039008         0.039008
12     duration    -0.028575         0.028575
2       default     0.016498         0.016498
7         month     0.016070         0.016070
6           day    -0.003112         0.003112
```

      OBSERVATION:

        - Contact and Y: These features have the highest absolute coefficients among the non-important features. While they contribute slightly to the SVM model's predictions, their impact is not as significant as other features.
        - Campaign, Duration, and Default: These features also have coefficients with moderate magnitudes but are still considered non-important. They contribute to the SVM model's predictions, albeit to a lesser extent than other features.
        - Month and Day: These features have the smallest absolute coefficients among the non-important features, indicating minimal impact on the SVM model's predictions. They are considered the least influential in the model's decision-making process.

5. MANAGERIAL INSIGHTS

5.1. Appropriate Model: Compare and Contrast

In our comparison process, we will employ a range of metrics such as accuracy, precision, recall, and F1-score. Additionally, we will analyze time and memory statistics to gain insights into the efficiency and computational requirements of each algorithm with respect to the given dataset. This comprehensive evaluation will help us identify the most suitable algorithm for our specific task.

- **ON THE BASIS OF CONFUSION MATRIX (ACCURACY, PRECISION, RECALL AND F1 SCORE):**
    1. **Accuracy: It measures the overall correctness of the model's predictions.**
        - Decision Tree: 0.56
        - Logistic Regression: 0.53
        - K-Nearest Neighbors (KNN): 0.53
        - Support Vector Machine (SVM): 0.56

       The SVM and Decision Tree model has the highest accuracy of 0.56, followed by K-Nearest Neighbor (0.53) and Logistic Regression (0.53) meaning they correctly classify a higher percentage of instances compared to logistic regression and KNN.
    2. **Precision: It indicates the proportion of correctly identified instances among all instances predicted by the model as belonging to a certain class.**
        - Decision Tree: 0.50, 0.68, 0.40 (for classes 0, 1, and 2 respectively)
        - Logistic Regression: 0.54, 0.56, 0.43
        - KNN: 0.51, 0.62, 0.44
        - SVM: 0.54, 0.59, 0.54

       SVM exhibits the highest precision across all classes, indicating that it has fewer false positive predictions compared to other models followed by Logistic Regression. This means that when SVM predicts a certain class, it is more likely to be correct.
    3. **Recall: It measures the ability of the model to correctly identify all relevant instances from a class.**
        - Decision Tree: 0.83, 0.76, 0.05
        - Logistic Regression: 0.70, 0.59, 0.25
        - KNN: 0.62, 0.64, 0.33
        - SVM: 0.70, 0.73, 0.22

       Decision Tree achieves the highest recall for classes 0 and 1, indicating that it captures a higher proportion of true positives compared to other models. However, it has a lower recall for class 2, suggesting it may miss some instances of this class.
    4. **F1-score: It is the harmonic mean of precision and recall, providing a balance between the two metrics.**
        - Decision Tree: 0.63, 0.72, 0.08
        - Logistic Regression: 0.61, 0.58, 0.32
        - KNN: 0.56, 0.63, 0.38
        - SVM: 0.61, 0.66, 0.32

       Decision Tree achieves the highest F1-score for class 1, while SVM exhibits the highest F1-score for class 0, indicating a balance between precision and recall for these classes.

  Overall, Decision Tree emerges as the top-performing model, followed by SVM, KNN, and Logistic Regression, respectively on the basis of these metrics.
- **ON THE BASIS OF TIME AND MEMORY STATISTICS:**
    1. **K-Nearest Neighbors (KNN):**
        - KNN has the shortest time taken, requiring only 0.011 seconds.

        - It also has the lowest memory usage compared to Logistic Regression, KNN, and SVM.
        - KNN is the most efficient in terms of both time and memory usage.
    2. **Decision Tree:**
        - Decision Tree has a moderate time taken of 0.038 seconds, which is relatively faster than Logistic Regression and SVM but slower than KNN.
        - It has lower memory usage compared to Logistic Regression and SVM but slightly higher than KNN.
        - Decision Tree ranks second in terms of efficiency, being faster than Logistic Regression and SVM.
    3. **Logistic Regression:**
        - Logistic Regression takes 1.014 seconds, making it the slowest model among the four.
        - It has moderate memory usage, lower than SVM but higher than Decision Tree and KNN.
        - Logistic Regression ranks third in terms of efficiency, being slower than Decision Tree and KNN.
    4. **Support Vector Machine (SVM):**
        - SVM has the longest time taken, requiring 183.98 seconds.
        - It also has the highest memory usage among the four models.
        - SVM is the least efficient in terms of both time and memory usage.

APPROPRIATE MODEL:- DECISION TREE

- Decision Tree exhibits better performance in terms of precision, recall, F1-score, and accuracy, comparable to other models.
- It achieves a balanced trade-off between precision, recall, and accuracy across all classes, indicating reliable predictions.
- Decision Tree demonstrates moderate computational efficiency, with a relatively low time taken and memory usage compared to Logistic Regression and SVM.
- While not the fastest, it strikes a good balance between classification performance and computational resources.

5.2. Relevant or Important Variables or Features

Initially, the decision-making process heavily depended on the **'job'** variable. However, we refined our approach by conducting feature selection and pruning the 'job' variable. Through this process, we identified additional important features that significantly contribute to the decision-making process for segmenting or classifying customer data using the decision tree model of supervised learning.

These important features, along with their respective importance scores, are as follows:

- **Job: Dominant**
- **Education: 0.802**
- **Age: 0.183**
- **Marital Status: 0.015**

  Explanation of DTC Rules: Click [here](#scrollTo=U_bgswCX6ccn&line=3&uniqifier=1) to jump to the code cell.
- Root Node: Education
- Splitting Criteria:
    - **Education Split:**
        - The initial split is based on the education feature.
        - If education is less than or equal to 1.50, it follows the left branch of the tree. Otherwise, it follows the right branch.
    - **Age Split:**
        - Within the left branch (when education is less than or equal to 1.50), there is a further split based on the age feature.
        - If age is less than or equal to 0.50, it follows the left branch again. Otherwise, it follows the right branch.
    - **Education Split (Again):**
        - Within the left branch (when age is less than or equal to 0.50), there is another split based on the education feature.
        - This split further refines the decision based on different levels of education.
    - **Marital Status and Age Splits:**
        - Within the right branch (when education is greater than 1.50), there are splits based on marital and age.
        - These splits further segment the data based on marital status and age to make more specific predictions.
- **Interpretation:**
    - **The decision tree segments the data based on the values of education, age, and possibly marital.**
    - **It makes predictions based on these splits, assigning each observation to one of the classes.**

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.

  By leveraging the insights gained from these important features, the bank can enhance its customer-centric approach and tailor its offerings to better meet the diverse needs and preferences of individuals within each cluster, ultimately fostering stronger relationships and driving long-term customer satisfaction and loyalty.

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

-- CODE --

LIBRARIES USED

```
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
#memory profiler for finding the memory of the clusters
warnings.filterwarnings("ignore") # Ignore the warnings
```

2. DATA DESCRIPTION

2.1. DATA SOURCE, SIZE, SHAPE

DATA SOURCE

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

DATA DIMENSIONS

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

```

```
   Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```

# Specify file path
file_path1 = '/content/drive/My Drive/bank-full.csv'  # Update with the actual path of your CSV file
file_path2 = '/content/drive/My Drive/index_cluster.csv'

# Read CSV file
import pandas as pd
df1 = pd.read_csv(file_path1)
df2 = pd.read_csv(file_path2)
```

```

df1.insert(0, 'srno', range(0, len(df1))) # Adding Index Variable

df1.head(10)
```

	srno	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	p
0	0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	
1	1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	
2	2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	
3	3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	
4	4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	
5	5	35	management	married	tertiary	no	231	yes	no	unknown	5	may	139	1	
6	6	28	management	single	tertiary	no	447	yes	yes	unknown	5	may	217	1	
7	7	42	entrepreneur	divorced	tertiary	yes	2	yes	no	unknown	5	may	380	1	
8	8	58	retired	married	primary	no	121	yes	no	unknown	5	may	50	1	
9	9	43	technician	single	secondary	no	593	yes	no	unknown	5	may	55	1	

```
import pandas as pd

# Assuming df1 and df2 are your dataframes
df = pd.merge(df1, df2, on='srno', how='inner')
df
```

	srno	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campai
0	0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	
1	1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	
2	2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	
3	3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	
4	4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
45206	45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	
45207	45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	
45208	45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127	
45209	45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	
45210	45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361	

45211 rows × 19 columns

```
import pandas as pd
```

```
df.shape
```

```
(45211, 19)
```

Dataset consists of 19 columns and 45211 rows

DATA SIZE

```
import pandas as pd

data_size_bytes = df.memory_usage().sum()
data_size_mb = data_size_bytes / (1024 * 1024)

print("Size of dataset (MB):", data_size_mb)
```

```
Size of dataset (MB): 6.553840637207031
```

OBSERVATION

Size of Dataset Used - 6.89 MB

```
df.columns
```

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

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:** Number of clusters after applying k-means clustering algorithm(k=3).

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, cluster:** 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:

Srno is index variable uniquely identifying each row in the dataset.

2.3 DESCRIPTIVE STATISTICS

FREQUENCY DISTRIBUTION OF CATEGORICAL VARIABLES

```

categorical_vars = ['job', 'marital', 'housing', 'loan', 'contact', 'month', 'y', 'education','day','cluster']

for var in categorical_vars:
    # Calculate frequency statistics
    freq_stats = df[var].value_counts()

    # Display frequency statistics
    print(f"\nCount | Frequency Statistics for {var}:")
    print(freq_stats)
```

```
Count | Frequency Statistics for job:
job
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: count, dtype: int64

Count | Frequency Statistics for marital:
marital
married     27214
single      12790
divorced     5207
Name: count, dtype: int64

Count | Frequency Statistics for housing:
housing
yes    25130
no     20081
Name: count, dtype: int64

Count | Frequency Statistics for loan:
loan
no     37967
```

```
yes     7244
Name: count, dtype: int64

Count | Frequency Statistics for contact:
contact
cellular     29285
unknown      13020
telephone     2906
Name: count, dtype: int64

Count | Frequency Statistics for month:
month
may    13766
jul     6895
aug     6247
jun     5341
nov     3970
apr     2932
feb     2649
jan     1403
oct      738
sep      579
mar      477
dec      214
Name: count, dtype: int64
```

RELATIVE FREQUENCY OF CATEGORICAL VARIABLES

```python
# Displaying frequency of each category in tabular form
for var in categorical_vars:
    print(f"\nFrequency of categories in '{var}':\n")
    print(df[var].value_counts(normalize=True).to_frame().rename(columns={var: 'Relative Frequency'}))
    print("\n=====================================\n")
```

```

Frequency of categories in 'job':

               proportion
job
blue-collar      0.215257
management       0.209197
technician       0.168034
admin.           0.114375
services         0.091880
retired          0.050076
self-employed    0.034925
entrepreneur     0.032890
unemployed       0.028820
housemaid        0.027427
student          0.020747
unknown          0.006370

=====================================


Frequency of categories in 'marital':

          proportion
marital
married     0.601933
single      0.282896
divorced    0.115171

=====================================


Frequency of categories in 'housing':

         proportion
housing
yes        0.555838
no         0.444162

=====================================


Frequency of categories in 'loan':

      proportion
loan
no      0.839774
yes     0.160226

=====================================


Frequency of categories in 'contact':

           proportion
contact
cellular     0.647741
unknown      0.287983
```

These frequency tables provide insights into the distribution of categorical variables in the dataset:

- Job: The most common job categories are blue-collar, management, and technician, together accounting for around 59% of the dataset.
- Marital Status: The majority of individuals are married (approximately 60%), followed by singles (around 28%) and divorced individuals (about 12%).
- Housing Loan: Approximately 56% of individuals have a housing loan.
- Personal Loan: The majority of individuals (around 84%) do not have a personal loan.
- Contact Method: The most common contact method is cellular, used for approximately 65% of the contacts.

- Month: The most frequent month for contact is May, accounting for approximately 30% of the contacts, followed by July and August.
- Subscription to Term Deposit: Around 11.7% of individuals subscribed to a term deposit.
- Education: The majority of individuals have a secondary education (around 51%), followed by tertiary education (approximately 29%).
- Day of Month: The distribution of contacts throughout the month is relatively uniform, with no single day significantly standing out.

DESCRIPTIVE STATISTICS OF NON-CATEGORICAL VARIABLES

- Measure Of Central Tendency
- Measure Of Dispersion
- Correlation Statistics

```
# Non-categorical variables
non_categorical_vars = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']

# Calculate descriptive statistics
non_categorical_stats = df[non_categorical_vars].describe()

# Calculate additional measures of dispersion
# Variance
variance = df[non_categorical_vars].var()

# Range
data_range = df[non_categorical_vars].max() - df[non_categorical_vars].min()

# Interquartile Range (IQR)
Q1 = df[non_categorical_vars].quantile(0.25)
Q3 = df[non_categorical_vars].quantile(0.75)
IQR = Q3 - Q1

# Display measures of dispersion
print("Measures of Dispersion for Non-Categorical Variables:")
print("\nDescriptive Statistics:")
print(non_categorical_stats)
print("\nVariance:")
print(variance)
print("\nRange:")
print(data_range)
print("\nInterquartile Range (IQR):")
print(IQR)
```

```
Measures of Dispersion for Non-Categorical Variables:

Descriptive Statistics:
                age        balance      duration      campaign         pdays  \
count  45211.000000   45211.000000  45211.000000  45211.000000  45211.000000
mean      40.936210    1362.272058    258.163080      2.763841     40.197828
std       10.618762    3044.765829    257.527812      3.098021    100.128746
min       18.000000   -8019.000000      0.000000      1.000000     -1.000000
25%       33.000000      72.000000    103.000000      1.000000     -1.000000
50%       39.000000     448.000000    180.000000      2.000000     -1.000000
75%       48.000000    1428.000000    319.000000      3.000000     -1.000000
max       95.000000  102127.000000   4918.000000     63.000000    871.000000

           previous
count  45211.000000
mean       0.580323
std        2.303441
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max      275.000000

Variance:
age         1.127581e+02
balance     9.270599e+06
duration    6.632057e+04
campaign    9.597733e+00
pdays       1.002577e+04
previous    5.305841e+00
dtype: float64

Range:
age             77
balance     110146
duration      4918
campaign        62
pdays          872
previous       275
dtype: int64

Interquartile Range (IQR):
age           15.0
balance     1356.0
duration     216.0
campaign       2.0
pdays          0.0
previous       0.0
dtype: float64
```

OBSERVATION

These descriptive statistics provide insights into the distribution and variability of non-categorical variables in the dataset:

- **Age:** The average age of the individuals is around 41 years, with a standard deviation of approximately 10.6 years. The ages range from 18 to 95 years.

- **Balance:** The mean balance amount is approximately 1362, with a considerable standard deviation of 3044. The balance amounts vary widely, ranging from negative values (indicating debt) to a maximum of 102,127.
- **Duration:** The mean duration of the last contact in seconds is about 258 seconds, with a standard deviation of approximately 257 seconds. The duration varies from 0 seconds to a maximum of 4918 seconds.
- **Campaign:** On average, each individual was contacted about 2.76 times during the campaign, with a standard deviation of around 3.1. The number of contacts ranges from 1 to a maximum of 63.
- **Pdays:** The average number of days since the client was last contacted is approximately 40 days, with a standard deviation of approximately 100 days. The values range from -1 (indicating the client was not previously contacted) to a maximum of 871 days.
- **Previous:** On average, clients were contacted about 0.58 times before the current campaign, with a standard deviation of around 2.3. The number of previous contacts varies widely, ranging from 0 to a maximum of 275.

CORRELATION MATRIX

```
import pandas as pd
from scipy.stats import pearsonr

# Calculate correlation matrix
correlation_matrix = df[non_categorical_vars].corr()

p_values = pd.DataFrame(columns=correlation_matrix.columns, index=correlation_matrix.columns)

# Perform correlation test and calculate p-values
for col1 in non_categorical_vars:
    for col2 in non_categorical_vars:
        if col1 != col2:
            corr, p_value = pearsonr(df[col1], df[col2])
            p_values.loc[col1, col2] = p_value

# Display correlation matrix and p-values
print("Correlation Matrix:")
print(correlation_matrix)
print("\nP-values:")
print(p_values)

```

```
Correlation Matrix:
               age   balance  duration  campaign     pdays  previous
age       1.000000  0.097783 -0.004648  0.004760 -0.023758  0.001288
balance   0.097783  1.000000  0.021560 -0.014578  0.003435  0.016674
duration -0.004648  0.021560  1.000000 -0.084570 -0.001565  0.001203
campaign  0.004760 -0.014578 -0.084570  1.000000 -0.088628 -0.032855
pdays    -0.023758  0.003435 -0.001565 -0.088628  1.000000  0.454820
previous  0.001288  0.016674  0.001203 -0.032855  0.454820  1.000000

P-values:
               age   balance  duration  campaign     pdays  previous
age            NaN       0.0  0.322973  0.311463       0.0  0.784141
balance        0.0       NaN  0.000005  0.001936  0.465127  0.000392
duration  0.322973  0.000005       NaN       0.0  0.739356  0.798107
campaign  0.311463  0.001936       0.0       NaN       0.0       0.0
pdays          0.0  0.465127  0.739356       0.0       NaN       0.0
previous  0.784141  0.000392  0.798107       0.0       0.0       NaN
```

Observations from the correlation matrix:

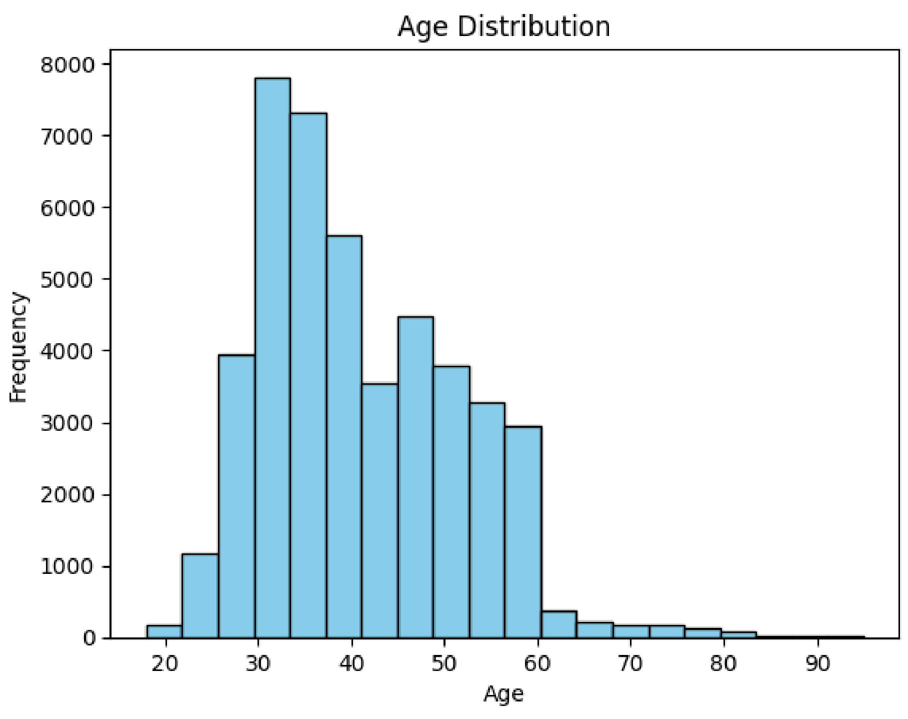
- Age shows a very weak positive correlation with balance (0.097783) and a weak negative correlation with pdays (-0.023758).
- Balance exhibits a weak positive correlation with age (0.097783) and a very weak positive correlation with previous (0.016674).
- Duration demonstrates negligible correlations with other variables.
- Campaign and pdays have weak negative correlations with duration (-0.084570 and -0.001565, respectively) and weak positive correlations with each other (0.454820).
- Previous displays very weak positive correlations with balance (0.016674) and pdays (0.454820).

Observations from p-values:

- Age has significant correlations with balance (p-value = 0.0) and pdays (p-value = 0.0).
- Balance has significant correlations with all other variables except duration (p-values < 0.05).
- Campaign and pdays have significant correlations with each other (p-value = 0.0) and with all other variables (p-values < 0.05).
- Previous has significant correlations with balance and pdays (p-values < 0.05).

DATA VISUALISATION

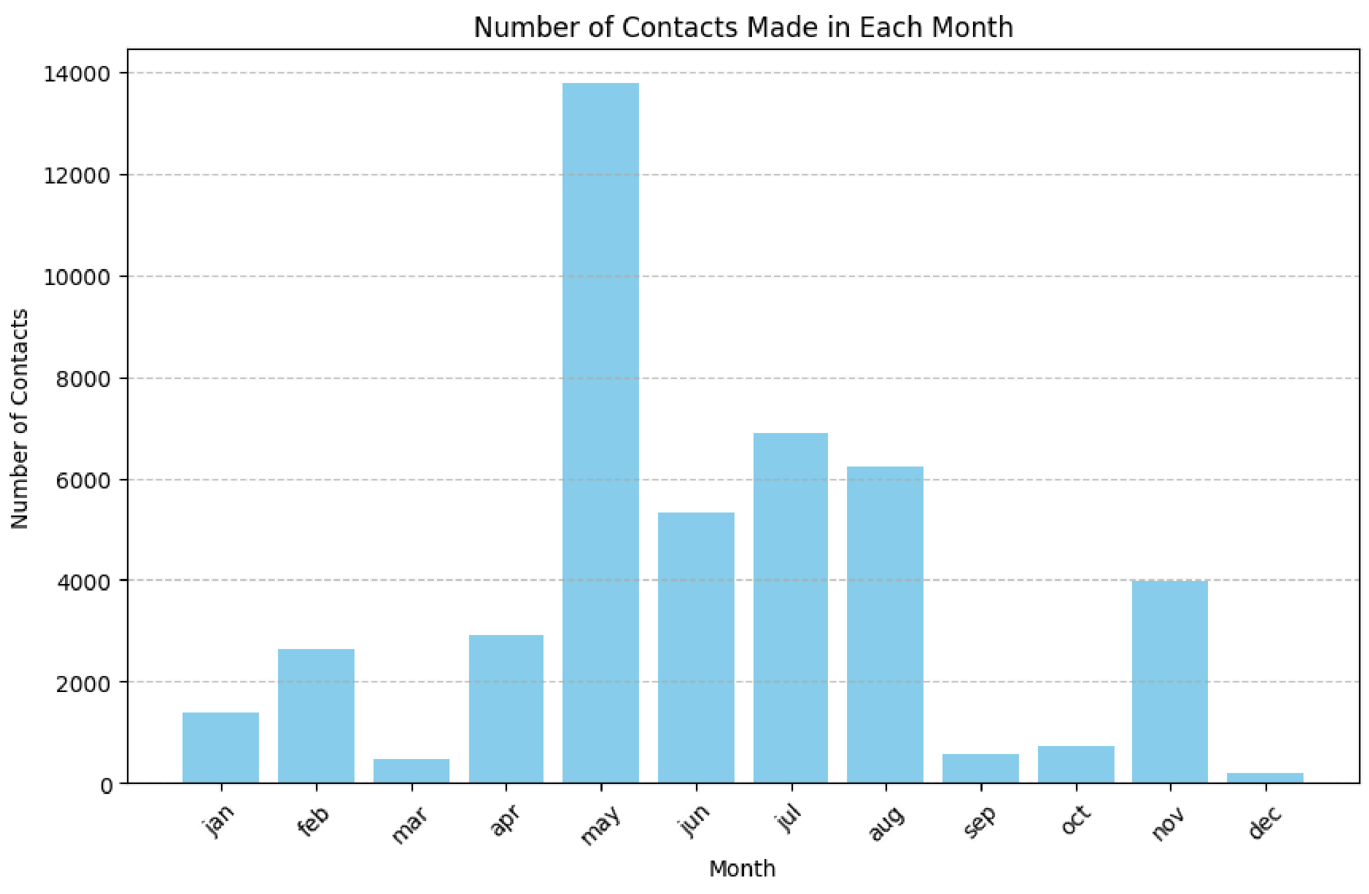
```
# Histogram of age distribution
import matplotlib.pyplot as plt

plt.hist(df['age'], bins=20, color='skyblue', edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

```



OBSERVATION:

The concentration of customers between the ages of 30 to 40 may be attributed to this demographic's peak earning and spending years, as individuals in this age group typically have established careers, higher disposable incomes, and increased financial responsibilities such as mortgage payments and family expenses. Additionally, this age range often coincides with major life events such as marriage, homeownership, and parenthood, prompting individuals to engage more actively with banking services for savings, investments, and loan products.

```
# Number of contacts made in each month.

# Define the desired order of months
month_order = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']
monthly_contacts = df['month'].value_counts().reindex(month_order)

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(monthly_contacts.index, monthly_contacts.values, color='skyblue')
plt.title('Number of Contacts Made in Each Month')
plt.xlabel('Month')
plt.ylabel('Number of Contacts')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

```

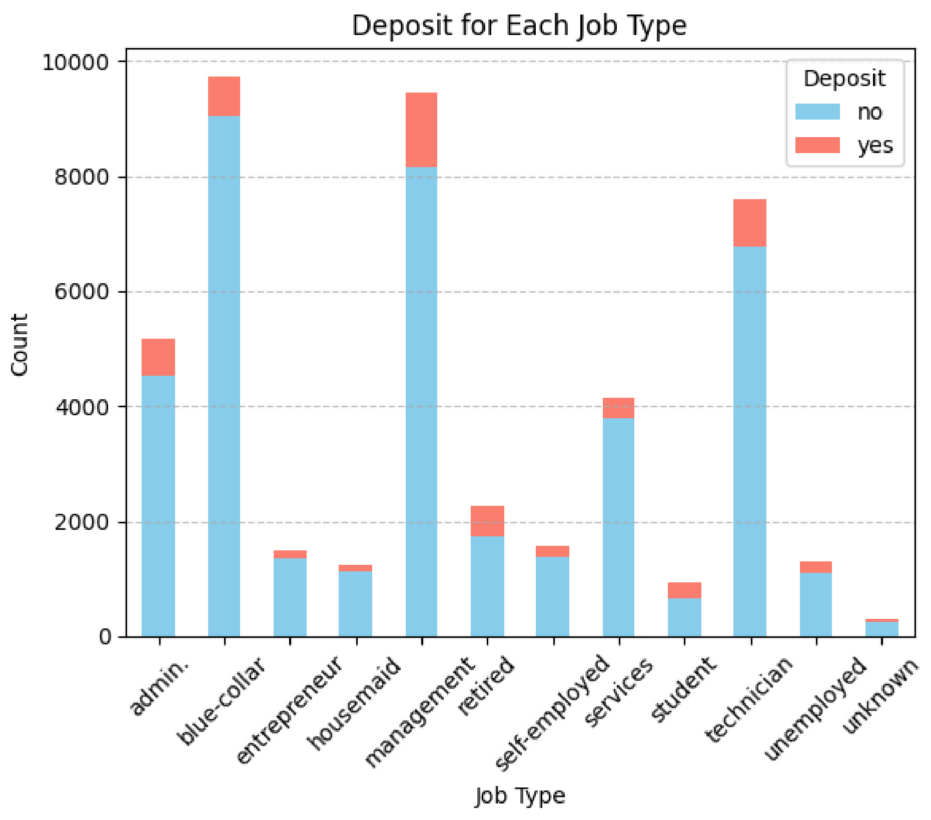


OBSERVATION:

The higher number of contacts made by the bank during May, June, and July may coincide with seasonal trends such as tax season, summer vacations, and mid-year financial reviews, prompting increased financial activity and communication with customers. Conversely, the decrease in contacts during October, September, and December could be influenced by holidays, year-end financial planning, and reduced business operations during these periods.

```
# Create a crosstab of job and deposit

import matplotlib.pyplot as plt

job_deposit_cross = pd.crosstab(df['job'], df['y'])

# Plot the stacked bar plot
plt.figure(figsize=(16, 8))
job_deposit_cross.plot(kind='bar', stacked=True, color=['skyblue', 'salmon'])
plt.title('Deposit for Each Job Type')
plt.xlabel('Job Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Deposit', loc='upper right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

```

```
<Figure size 1600x800 with 0 Axes>
```



OBSERVATION:

The stacked bar plot showcases the distribution of deposit subscriptions across various job types in the bank marketing dataset. It reveals that certain professions, such as admin, services, and retired individuals, exhibit a higher count of deposit subscriptions, while blue-collar, management, and technician roles show a higher proportion of non-subscriptions. This indicates a potential correlation between job type and deposit subscription behavior among bank customers.

3. ANALYSIS OF DATA

3.1 DATA PRE-PROCESSING

3.1.1 Missing Data Statistics and Treatment.

```
# Checking Missing Values
variable_missing_data = df.isna().sum(); variable_missing_data
```

```
srno         0
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
cluster      0
dtype: int64
```

OBSERVATION

Since there are no missing values present in the selected dataset, there is no need for imputation procedures.

```
'''
# Excluding Empty Records (If Any)
df_cat.dropna(axis=0, how='all', inplace=True) # Categorical Data Subset
df_noncat_mdi_si.dropna(axis=0, how='all', inplace=True) # Non-Categorical Data Subset

# Excluding Empty Variables (If Any)
df_cat_mdi.dropna(axis=1, how='all', inplace=True) # Categorical Data Subset
df_noncat_mdi_si.dropna(axis=1, how='all', inplace=True) # Non-Categorical Data Subset

df_cat_mdt = df_cat_mdi.copy() # Missing Categorical Treated Dataset
df_noncat_mdt = df_noncat_mdi_si.copy() # Missing Non-Categorical Treated Dataset

'''
```

```
'\n# Excluding Empty Records (If Any)\ndf_cat.dropna(axis=0, how='all', inplace=True) # Categorical Data Subset\ndf_noncat_mdi_si.dropna(axis=0, how
='all', inplace=True) # Non-Categorical Data Subset\n\n# Excluding Empty Variables (If Any)\ndf_cat_mdi.dropna(axis=1, how='all', inplace=True) # Ca
tegorical Data Subset\ndf_noncat_mdi_si.dropna(axis=1, how='all', inplace=True) # Non-Categorical Data Subset\n\ndf_cat_mdt = df_cat_mdi.copy() # Mi
ssing Categorical Treated Dataset\ndf_noncat_mdt = df_noncat_mdi_si.copy() # Missing Non-Categorical Treated Dataset\n\n'
```

```
'''
Impute Missing Categorical Data [Nominal | Ordinal] using Central Tendency (Mode)
si_cat = SimpleImputer(missing_values=np.nan, strategy='most_frequent') # Strategy = median [When Odd Number of Categories Exists]
si_cat_fit = si_cat.fit_transform(df_cat)
df_cat_mdi = pd.DataFrame(si_cat_fit, columns=df_cat.columns); df_cat_mdi # Missing Categorical Data Imputed Subset
df_cat_mdi.info()
'''
```

```
'\nImpute Missing Categorical Data [Nominal | Ordinal] using Central Tendency (Mode)\nsi_cat = SimpleImputer(missing_values=np.nan, strategy='most_f
requent') # Strategy = median [When Odd Number of Categories Exists]\nsi_cat_fit = si_cat.fit_transform(df_cat)\ndf_cat_mdi = pd.DataFrame(si_cat_fi
t, columns=df_cat.columns); df_cat_mdi # Missing Categorical Data Imputed Subset\ndf_cat_mdi.info()\n'
```

```
'''
Impute Missing Non-Categorical Data using Central Tendency(median)

si_noncat = SimpleImputer(missing_values=np.nan, strategy='median')
si_noncat_fit = si_noncat.fit_transform(df_noncat)
df_noncat_mdi_si = pd.DataFrame(si_noncat_fit, columns=df_noncat.columns); df_noncat_mdi_si # Missing Non-Categorical Data Imputed Subset using Simpl
df_noncat_mdi_si.info()

'''
```

```
'\nImpute Missing Non-Categorical Data using Central Tendency(median)\n\nsi_noncat = SimpleImputer(missing_values=n
p.nan, strategy='median')\nsi_noncat_fit = si_noncat.fit_transform(df_noncat)\ndf_noncat_mdi_si = pd.DataFrame(si_no
ncat_fit, columns=df_noncat.columns); df_noncat_mdi_si # Missing Non-Categorical Data Imputed Subset using Simple Im
puter\ndf_noncat_mdi_si.info()\n\n'
```

```
# checking number of duplicated values.

num_duplicate_rows = df.duplicated().sum()

print("Number of duplicate rows:", num_duplicate_rows)
```

```
Number of duplicate rows: 0
```

OBSERVATION

No Duplicate values present in the dataset.

Data Bifurcation Into Categorical and Non-Categorical Data

```
# Categorical Data
df_cat = df[['srno', 'job', 'marital','education','default','housing','loan','contact','day','month','poutcome','y','cluster']]
# Non Categorical Data
df_noncat = df[['srno', 'age', 'balance','duration','campaign','pdays','previous']]
```

3.1.2 Numerical Encoding of Categorical Variables or Features

Need For Numeric Encoding:

This allows clustering algorithms to process and analyze the data effectively, as most algorithms work with numerical inputs. Encoding ensures that categorical variables contribute meaningfully to the clustering process by representing them as numerical values while preserving the inherent characteristics of the original data.

Ordinal encoding converts categorical variables into numerical values while preserving their ordinal relationships, assigning unique integers based on the order of categories. It's suitable for variables with inherent order, like "low," "medium," and "high," ensuring compatibility with machine learning algorithms.

Categorical Data

```
df_cat # Dataframe containing Categorical variables.
```

	srno	job	marital	education	default	housing	loan	contact	day	month	poutcome	y	cluster
0	0	management	married	tertiary	no	yes	no	unknown	5	may	unknown	no	1
1	1	technician	single	secondary	no	yes	no	unknown	5	may	unknown	no	2
2	2	entrepreneur	married	secondary	no	yes	yes	unknown	5	may	unknown	no	0
3	3	blue-collar	married	unknown	no	yes	no	unknown	5	may	unknown	no	0
4	4	unknown	single	unknown	no	no	no	unknown	5	may	unknown	no	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	45206	technician	married	tertiary	no	no	no	cellular	17	nov	unknown	yes	2
45207	45207	retired	divorced	primary	no	no	no	cellular	17	nov	unknown	yes	1
45208	45208	retired	married	secondary	no	no	no	cellular	17	nov	success	yes	1
45209	45209	blue-collar	married	secondary	no	no	no	telephone	17	nov	unknown	no	0
45210	45210	entrepreneur	married	secondary	no	no	no	cellular	17	nov	other	no	0

45211 rows × 13 columns

OBSERVATION:

Among the categorical variables, all entries, with the exception of the 'day' column, are alphanumeric in nature. The 'day' column, in contrast, comprises numerical values.

Alphanumeric Encoding

```
from sklearn.preprocessing import OrdinalEncoder # Using OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
categorical_columns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact','day', 'month', 'poutcome','y','cluster']
noncategorical_columns = ['age', 'balance','duration','campaign','pdays','previous']
# Fit and transform the categorical columns
df_cat[categorical_columns] = ordinal_encoder.fit_transform(df_cat[categorical_columns])

print(df_cat)

```

```
        srno   job  marital  education  default  housing  loan  contact   day  \
0          0   4.0      1.0        2.0      0.0      1.0   0.0      2.0   4.0
1          1   9.0      2.0        1.0      0.0      1.0   0.0      2.0   4.0
2          2   2.0      1.0        1.0      0.0      1.0   1.0      2.0   4.0
3          3   1.0      1.0        3.0      0.0      1.0   0.0      2.0   4.0
4          4  11.0      2.0        3.0      0.0      0.0   0.0      2.0   4.0
...      ...   ...      ...        ...      ...      ...   ...      ...   ...
45206  45206   9.0      1.0        2.0      0.0      0.0   0.0      0.0  16.0
45207  45207   5.0      0.0        0.0      0.0      0.0   0.0      0.0  16.0
45208  45208   5.0      1.0        1.0      0.0      0.0   0.0      0.0  16.0
45209  45209   1.0      1.0        1.0      0.0      0.0   0.0      1.0  16.0
45210  45210   2.0      1.0        1.0      0.0      0.0   0.0      0.0  16.0

       month  poutcome    y  cluster
0        8.0       3.0  0.0      1.0
1        8.0       3.0  0.0      2.0
2        8.0       3.0  0.0      0.0
3        8.0       3.0  0.0      0.0
4        8.0       3.0  0.0      2.0
...      ...       ...  ...      ...
45206    9.0       3.0  1.0      2.0
45207    9.0       3.0  1.0      1.0
45208    9.0       2.0  1.0      1.0
45209    9.0       3.0  0.0      0.0
45210    9.0       1.0  0.0      0.0

[45211 rows x 13 columns]
```

OBSERVATION

The categorical columns specified in categorical_columns are transformed into numerical representations, where each unique category is assigned an ordinal value based on its order of appearance in the dataset. All the categorical data is encoded into numerical form and as there are no missing values, there is no use of creating dummy data.

Encoded Values

```
encoded_categories = ordinal_encoder.categories_

# To Print the categories along with their corresponding encoded values
for col_index, col_name in enumerate(categorical_columns):
    print(f"{col_name}:")
    category_strings = [f"{category}: {category_index}" for category_index, category in enumerate(encoded_categories[col_index])]
    print(", ".join(category_strings))

```

```
job:
admin.: 0, blue-collar: 1, entrepreneur: 2, housemaid: 3, management: 4, retired: 5, self-employed: 6, services: 7, student: 8, technician: 9, unempl
marital:
divorced: 0, married: 1, single: 2
education:
primary: 0, secondary: 1, tertiary: 2, unknown: 3
default:
no: 0, yes: 1
housing:
no: 0, yes: 1
loan:
```

```
no: 0, yes: 1
contact:
cellular: 0, telephone: 1, unknown: 2
day:
1: 0, 2: 1, 3: 2, 4: 3, 5: 4, 6: 5, 7: 6, 8: 7, 9: 8, 10: 9, 11: 10, 12: 11, 13: 12, 14: 13, 15: 14, 16: 15, 17: 16, 18: 17, 19: 18, 20: 19, 21: 20,
month:
apr: 0, aug: 1, dec: 2, feb: 3, jan: 4, jul: 5, jun: 6, mar: 7, may: 8, nov: 9, oct: 10, sep: 11
poutcome:
failure: 0, other: 1, success: 2, unknown: 3
y:
no: 0, yes: 1
cluster:
0: 0, 1: 1, 2: 2
```

3.1.3 Data Transformation & Rescaling [Treatment of Outliers] for Non-Categorical Data

Need For Data Transformation & Rescaling:
To ensure that **numerical features have similar scales**, preventing bias in clustering algorithms. This enhances algorithm convergence, improves interpretability, and mitigates the influence of outliers, ultimately leading to more accurate and reliable clustering results.

```
df_noncat # Dataframe containing non-categorical variables.
```

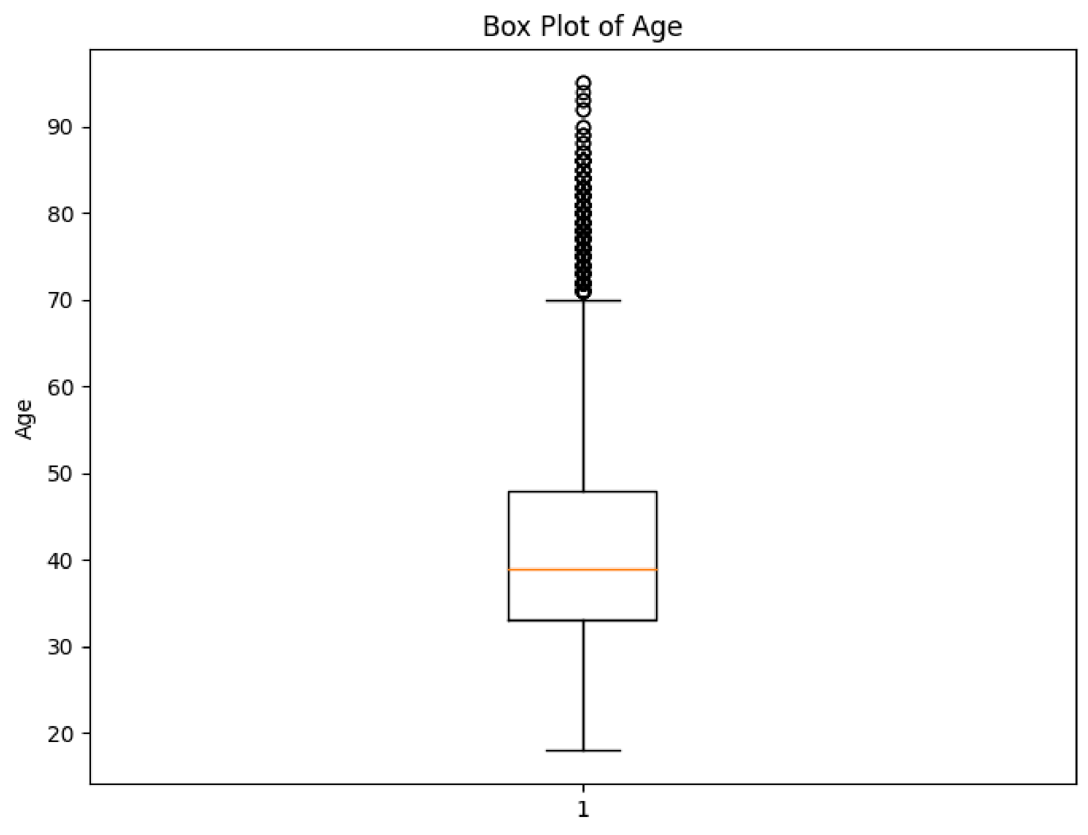
	srno	age	balance	duration	campaign	pdays	previous
0	0	58	2143	261	1	-1	0
1	1	44	29	151	1	-1	0
2	2	33	2	76	1	-1	0
3	3	47	1506	92	1	-1	0
4	4	33	1	198	1	-1	0
...	...	...	...	...	...	...	...
45206	45206	51	825	977	3	-1	0
45207	45207	71	1729	456	2	-1	0
45208	45208	72	5715	1127	5	184	3
45209	45209	57	668	508	4	-1	0
45210	45210	37	2971	361	2	188	11

45211 rows × 7 columns

BOX PLOT

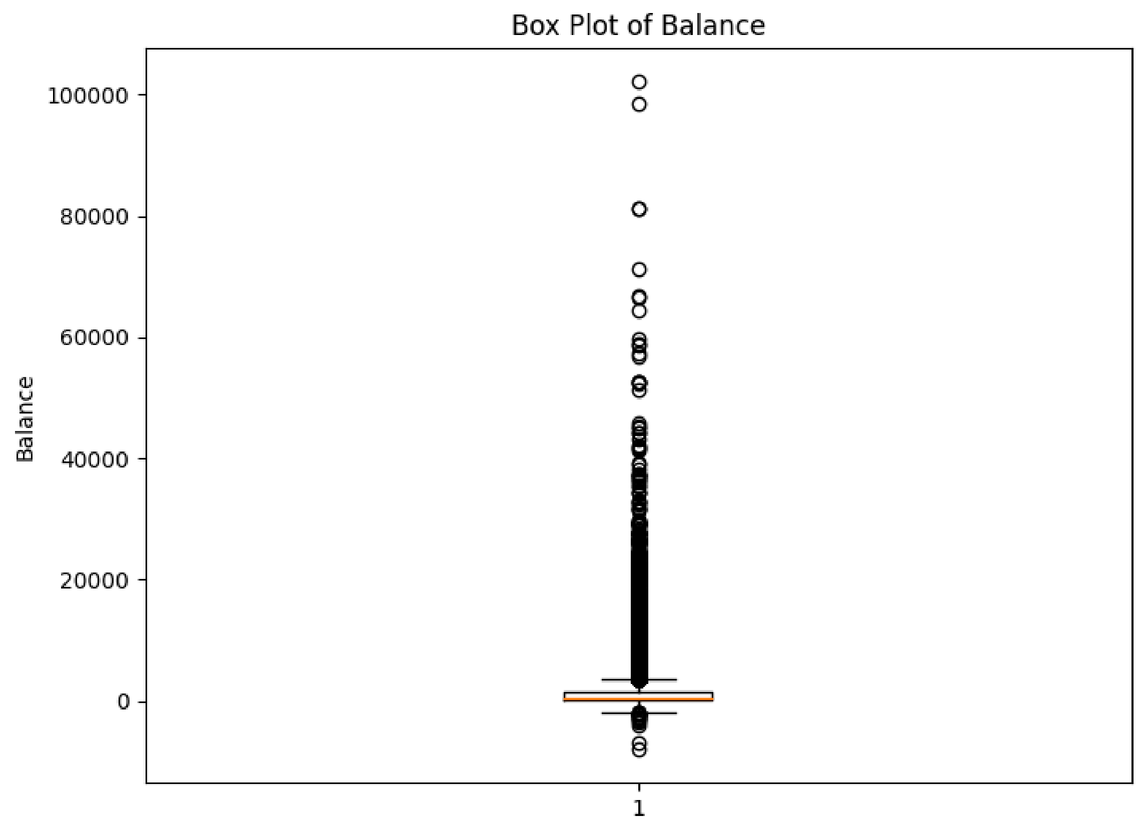
- A box plot is a graphical representation used for visualizing the **distribution of non-categorical data and identifying potential outliers**. It consists of a box that spans the interquartile range (IQR), with a line representing the median. Outliers are data points that fall beyond the whiskers and are represented as individual points on the plot.
- The primary use of a box plot in data transformation and rescaling is to **assess the distribution of the data and detect any outliers that may skew the analysis.** This information is crucial for making informed decisions about whether to treat outliers through various methods such as removing, transforming, or imputing them, depending on the context of the analysis.
- Rescaling techniques such as **min-max scaling or normalization(0-1) can be applied to transform numerical data into a comparable range**. Box plots help in visualizing the effectiveness of these transformations in reducing the influence of outliers and improving the distribution of the data.

```
# Box plot for 'age'
plt.figure(figsize=(8, 6))
plt.boxplot(df['age'])
plt.title('Box Plot of Age')
plt.ylabel('Age')
plt.show()

q1_campaign = np.percentile(df['age'], 25)
q3_campaign = np.percentile(df['age'], 75)
iqr_campaign = q3_campaign - q1_campaign
lower_bound_campaign = q1_campaign - 1.5 * iqr_campaign
upper_bound_campaign = q3_campaign + 1.5 * iqr_campaign
outliers_campaign = [x for x in df['age'] if x < lower_bound_campaign or x > upper_bound_campaign]

print("Outlier Statistics for Age:")
print("Q1:", q1_campaign)
print("Q3:", q3_campaign)
print("IQR:", iqr_campaign)
print("Lower Bound:", lower_bound_campaign)
print("Upper Bound:", upper_bound_campaign)
print("Number of Outliers:", len(outliers_campaign))

plt.show()

```



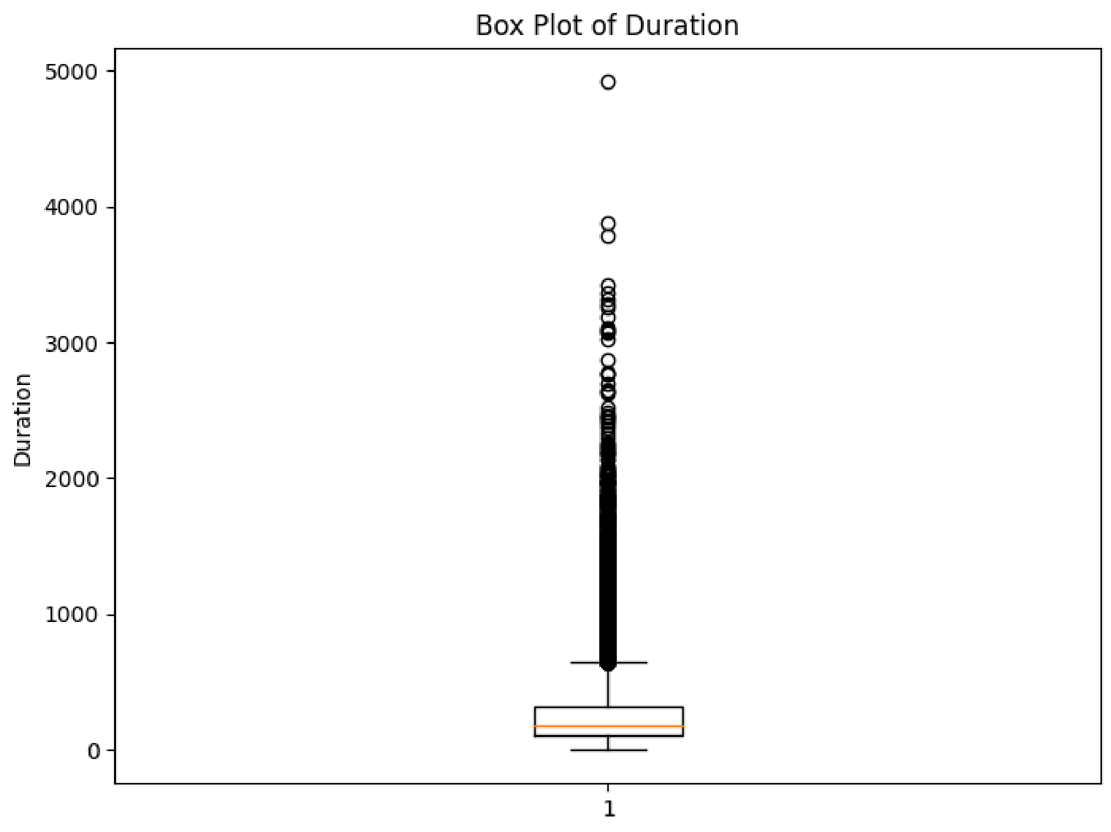
```
Outlier Statistics for Age:
Q1: 33.0
Q3: 48.0
IQR: 15.0
Lower Bound: 10.5
Upper Bound: 70.5
Number of Outliers: 487
```

```python
# Box plot for 'balance'
plt.figure(figsize=(8, 6))
plt.boxplot(df['balance'])
plt.title('Box Plot of Balance')
plt.ylabel('Balance')
plt.show()

q1_campaign = np.percentile(df['balance'], 25)
q3_campaign = np.percentile(df['balance'], 75)
iqr_campaign = q3_campaign - q1_campaign
lower_bound_campaign = q1_campaign - 1.5 * iqr_campaign
upper_bound_campaign = q3_campaign + 1.5 * iqr_campaign
outliers_campaign = [x for x in df['balance'] if x < lower_bound_campaign or x > upper_bound_campaign]

print("Outlier Statistics for Balance:")
print("Q1:", q1_campaign)
print("Q3:", q3_campaign)
print("IQR:", iqr_campaign)
print("Lower Bound:", lower_bound_campaign)
print("Upper Bound:", upper_bound_campaign)
print("Number of Outliers:", len(outliers_campaign))

plt.show()

```



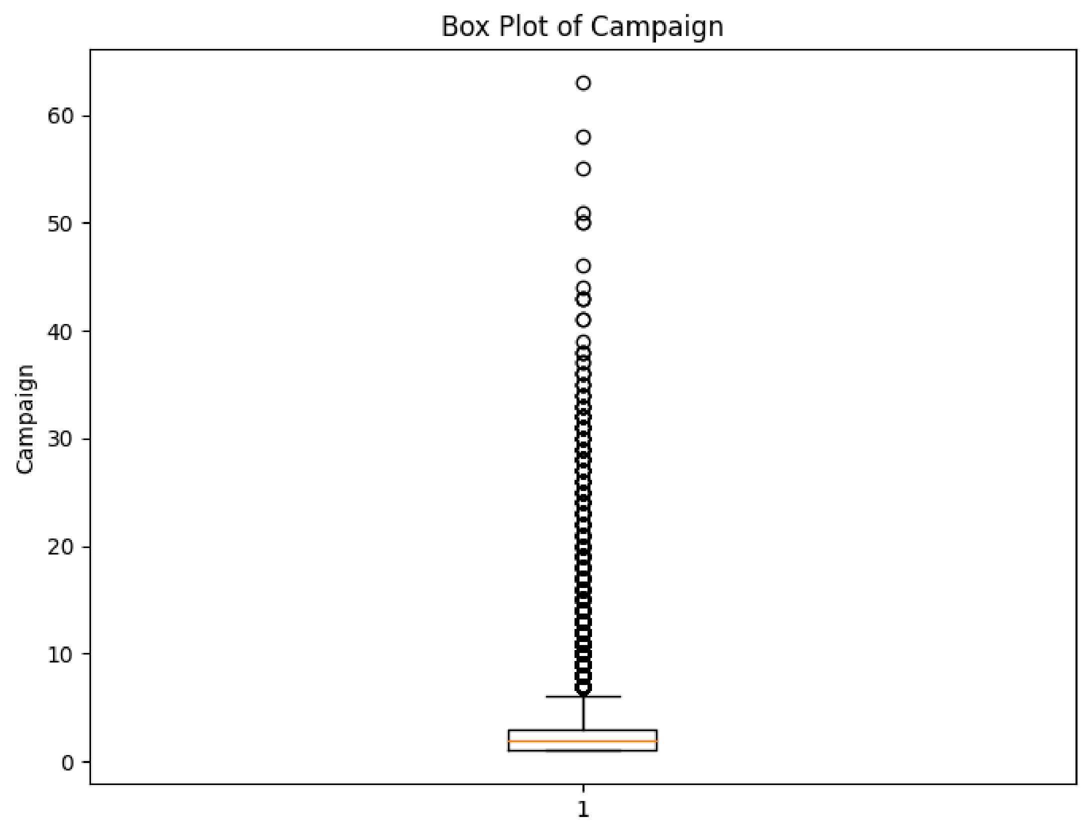
```
Outlier Statistics for Balance:
Q1: 72.0
Q3: 1428.0
IQR: 1356.0
Lower Bound: -1962.0
Upper Bound: 3462.0
Number of Outliers: 4729
```

```
# Box plot for 'duration'
plt.figure(figsize=(8, 6))
plt.boxplot(df['duration'])
plt.title('Box Plot of Duration')
plt.ylabel('Duration')
plt.show()

q1_campaign = np.percentile(df['duration'], 25)
q3_campaign = np.percentile(df['duration'], 75)
iqr_campaign = q3_campaign - q1_campaign
lower_bound_campaign = q1_campaign - 1.5 * iqr_campaign
upper_bound_campaign = q3_campaign + 1.5 * iqr_campaign
outliers_campaign = [x for x in df['duration'] if x < lower_bound_campaign or x > upper_bound_campaign]

print("Outlier Statistics for duration:")
print("Q1:", q1_campaign)
print("Q3:", q3_campaign)
print("IQR:", iqr_campaign)
print("Lower Bound:", lower_bound_campaign)
print("Upper Bound:", upper_bound_campaign)
print("Number of Outliers:", len(outliers_campaign))

plt.show()

```



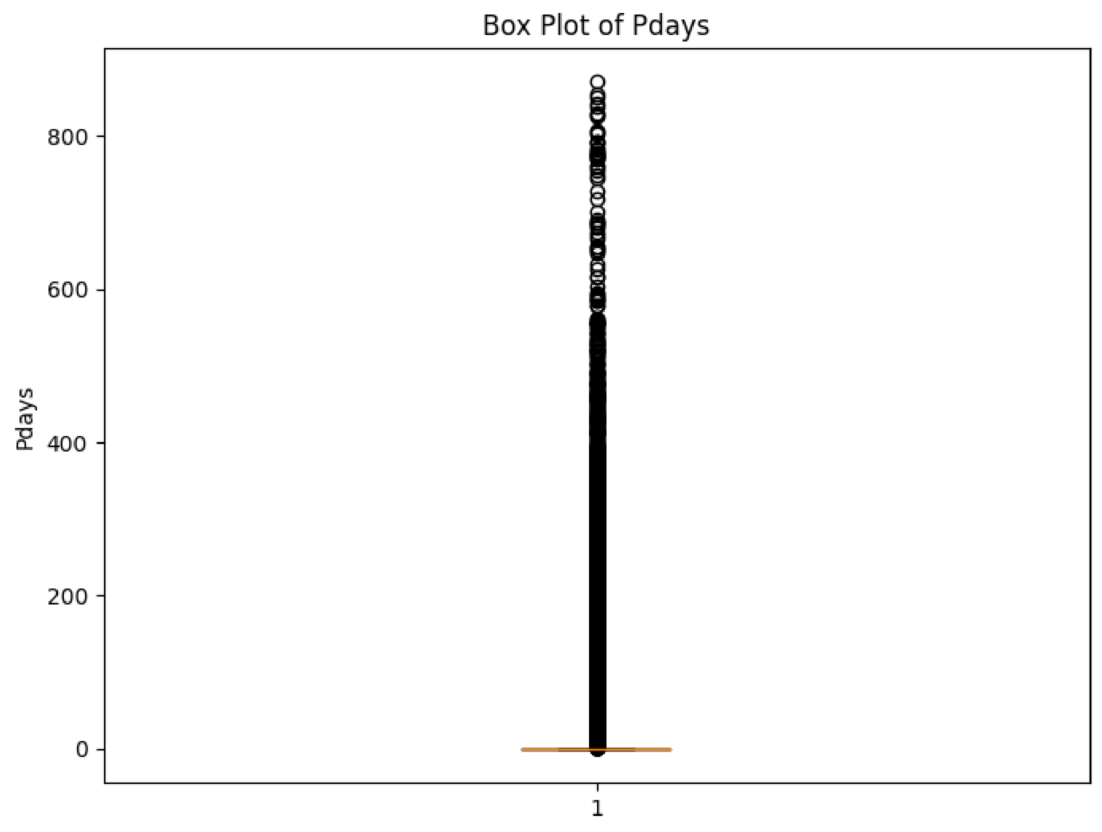
```
Outlier Statistics for duration:
Q1: 103.0
Q3: 319.0
IQR: 216.0
Lower Bound: -221.0
Upper Bound: 643.0
Number of Outliers: 3235
```

```

plt.figure(figsize=(8, 6))
plt.boxplot(df['campaign'])
plt.title('Box Plot of Campaign')
plt.ylabel('Campaign')
plt.show()

q1_campaign = np.percentile(df['campaign'], 25)
q3_campaign = np.percentile(df['campaign'], 75)
iqr_campaign = q3_campaign - q1_campaign
lower_bound_campaign = q1_campaign - 1.5 * iqr_campaign
upper_bound_campaign = q3_campaign + 1.5 * iqr_campaign
outliers_campaign = [x for x in df['campaign'] if x < lower_bound_campaign or x > upper_bound_campaign]

print("Outlier Statistics for campaign:")
print("Q1:", q1_campaign)
print("Q3:", q3_campaign)
print("IQR:", iqr_campaign)
print("Lower Bound:", lower_bound_campaign)
print("Upper Bound:", upper_bound_campaign)
print("Number of Outliers:", len(outliers_campaign))

plt.show()

```



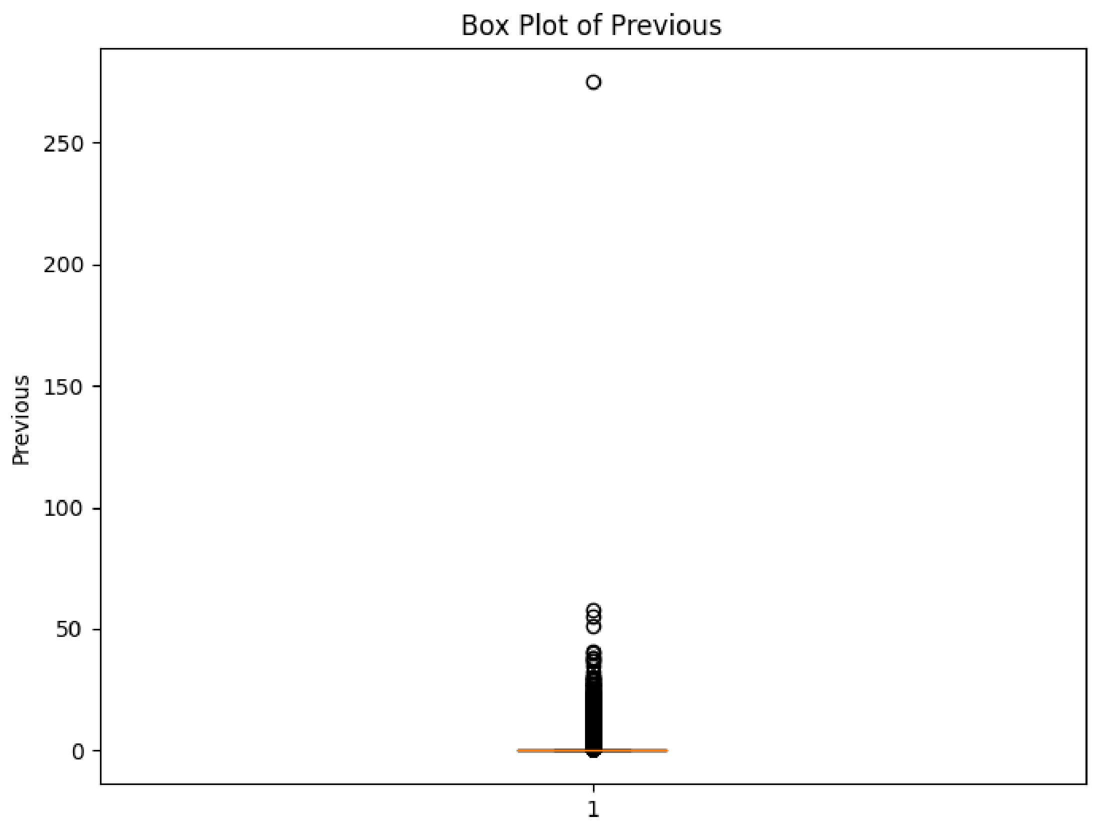
```
Outlier Statistics for campaign:
Q1: 1.0
Q3: 3.0
IQR: 2.0
Lower Bound: -2.0
Upper Bound: 6.0
Number of Outliers: 3064
```

```

plt.figure(figsize=(8, 6))
plt.boxplot(df['pdays'])
plt.title('Box Plot of Pdays')
plt.ylabel('Pdays')
plt.show()

q1_campaign = np.percentile(df['pdays'], 25)
q3_campaign = np.percentile(df['pdays'], 75)
iqr_campaign = q3_campaign - q1_campaign
lower_bound_campaign = q1_campaign - 1.5 * iqr_campaign
upper_bound_campaign = q3_campaign + 1.5 * iqr_campaign
outliers_campaign = [x for x in df['pdays'] if x < lower_bound_campaign or x > upper_bound_campaign]

print("Outlier Statistics for pdays:")
print("Q1:", q1_campaign)
print("Q3:", q3_campaign)
print("IQR:", iqr_campaign)
print("Lower Bound:", lower_bound_campaign)
print("Upper Bound:", upper_bound_campaign)
print("Number of Outliers:", len(outliers_campaign))

plt.show()

```



```
Outlier Statistics for pdays:
Q1: -1.0
Q3: -1.0
IQR: 0.0
Lower Bound: -1.0
Upper Bound: -1.0
Number of Outliers: 8257
```

```

plt.figure(figsize=(8, 6))
plt.boxplot(df['previous'])
plt.title('Box Plot of Previous')
plt.ylabel('Previous')
plt.show()


q1_campaign = np.percentile(df['previous'], 25)
q3_campaign = np.percentile(df['previous'], 75)
iqr_campaign = q3_campaign - q1_campaign
lower_bound_campaign = q1_campaign - 1.5 * iqr_campaign
upper_bound_campaign = q3_campaign + 1.5 * iqr_campaign
outliers_campaign = [x for x in df['previous'] if x < lower_bound_campaign or x > upper_bound_campaign]

print("Outlier Statistics for previous:")
print("Q1:", q1_campaign)
print("Q3:", q3_campaign)
print("IQR:", iqr_campaign)
print("Lower Bound:", lower_bound_campaign)
print("Upper Bound:", upper_bound_campaign)
print("Number of Outliers:", len(outliers_campaign))

plt.show()
```



```
Outlier Statistics for previous:
Q1: 0.0
Q3: 0.0
IQR: 0.0
Lower Bound: 0.0
Upper Bound: 0.0
Number of Outliers: 8257
```

OUTLIER TREATMENT USING MIN-MAX SCALING

Min-Max scaling transforms numerical features to a uniform range between 0 and 1, ensuring consistent scaling across variables. This prevents dominance by features with larger ranges and facilitates fair comparison. It preserves relative differences, crucial for maintaining dataset integrity. Additionally, it can enhance model performance, especially for algorithms sensitive to feature scale, by mitigating bias towards larger-magnitude features.

```
from sklearn.preprocessing import MinMaxScaler # For Rescaling Data
# Normalization : Min-Max Scaling
mms = MinMaxScaler()
df_noncat[noncategorical_columns]  = mms.fit_transform(df_noncat[noncategorical_columns] )

df_noncat
```

	srno	age	balance	duration	campaign	pdays	previous
0	0	0.519481	0.092259	0.053070	0.000000	0.000000	0.000000
1	1	0.337662	0.073067	0.030704	0.000000	0.000000	0.000000
2	2	0.194805	0.072822	0.015453	0.000000	0.000000	0.000000
3	3	0.376623	0.086476	0.018707	0.000000	0.000000	0.000000
4	4	0.194805	0.072812	0.040260	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...
45206	45206	0.428571	0.080293	0.198658	0.032258	0.000000	0.000000
45207	45207	0.688312	0.088501	0.092721	0.016129	0.000000	0.000000
45208	45208	0.701299	0.124689	0.229158	0.064516	0.212156	0.010909
45209	45209	0.506494	0.078868	0.103294	0.048387	0.000000	0.000000
45210	45210	0.246753	0.099777	0.073404	0.016129	0.216743	0.040000

45211 rows × 7 columns

Merging of both the dataset to give one final pre-processed data.

```
# df_ppd = df_cat.join(df_noncat); df_ppd # Pre-Processed Dataset
df_ppd = pd.merge(df_cat, df_noncat, on='srno')
df_ppd
```

	srno	job	marital	education	default	housing	loan	contact	day	month	poutcome	y	cluster	age	balan
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	0.0922
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	0.0730
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	0.0728
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	0.0864
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	0.0728
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	0.0802
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	0.0885
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	0.1246
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	0.0788
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	0.0997

45211 rows × 19 columns

Choosing the relevant input and output features.

```
# 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

```
# 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']
```

3.1.4 DATA BIFURCATION: TRAINING AND TESTING SETS

```
# 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

```

OBSERVATION:

train_bank_inputs - contains all the input variables used for training.(80%)
test_bank_inputs - contains all the input variables used for testing.(20%)
train_bank_output - contains the output variable used for training.(80%)
test_bak_oupput - contains the output variables used for testing.(20%)

DECISION TREE

```
# Decision Tree : Model (Training Subset)
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: 0.04252886772155762 seconds
Memory used: 283.171875 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
```

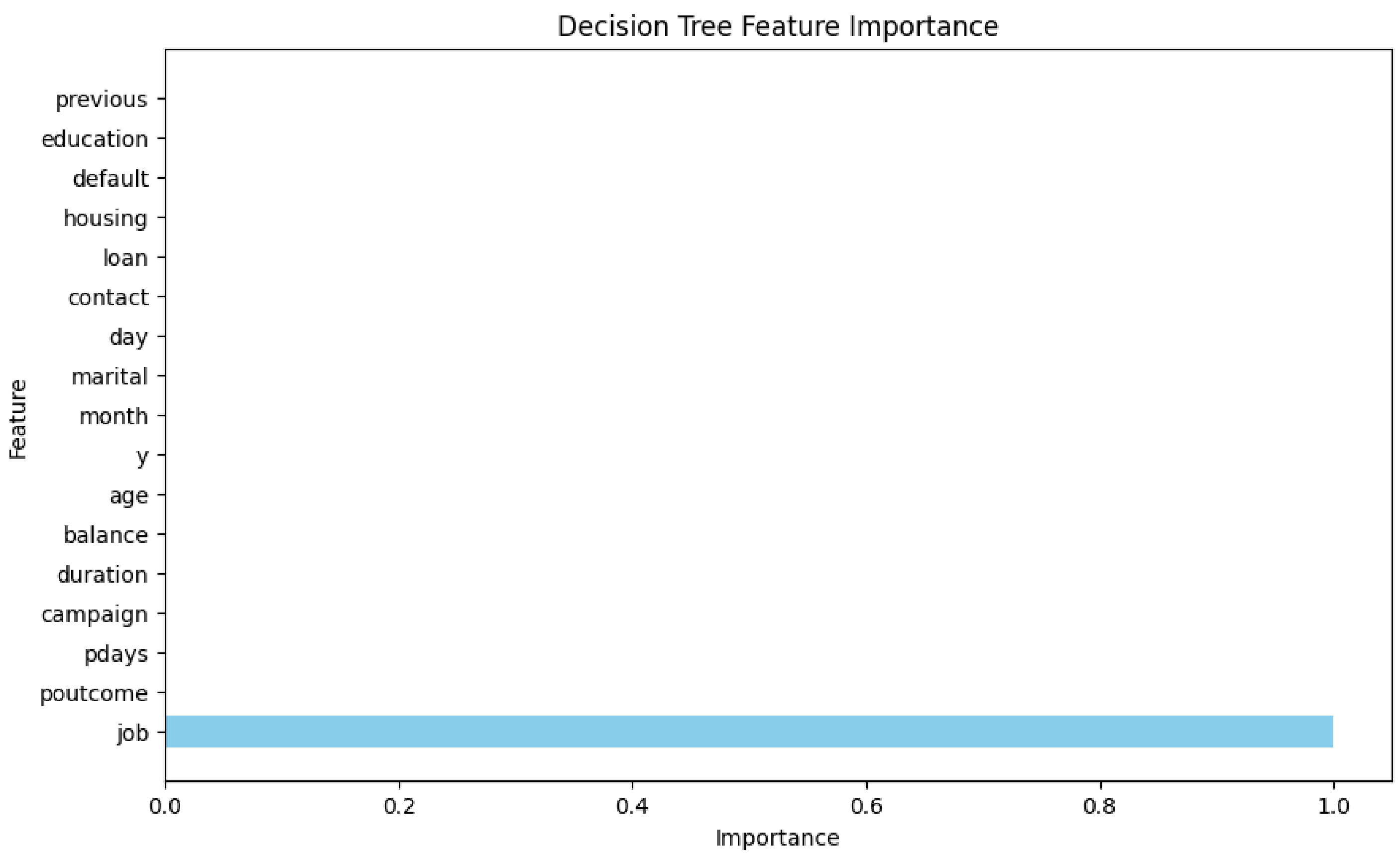
According to the above tree, only 'job' variable is used to determine the cluster to which a datapoint belongs.

If job is less than 2.5, it belongs to cluster, if job is between 2.5 and 6.5 it belongs to cluster 1 else(job > 6.5) it belongs to cluster 2.

```
# 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

# Assuming dtc_imp_features is your DataFrame containing feature importance
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()

```



As only 'job' is coming out to be the important feature, we will prune the algorithm so as to remove 'job' and see what are the other contributing features.

```
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: 0.06053805351257324 seconds
Memory used: 302.28125 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
```

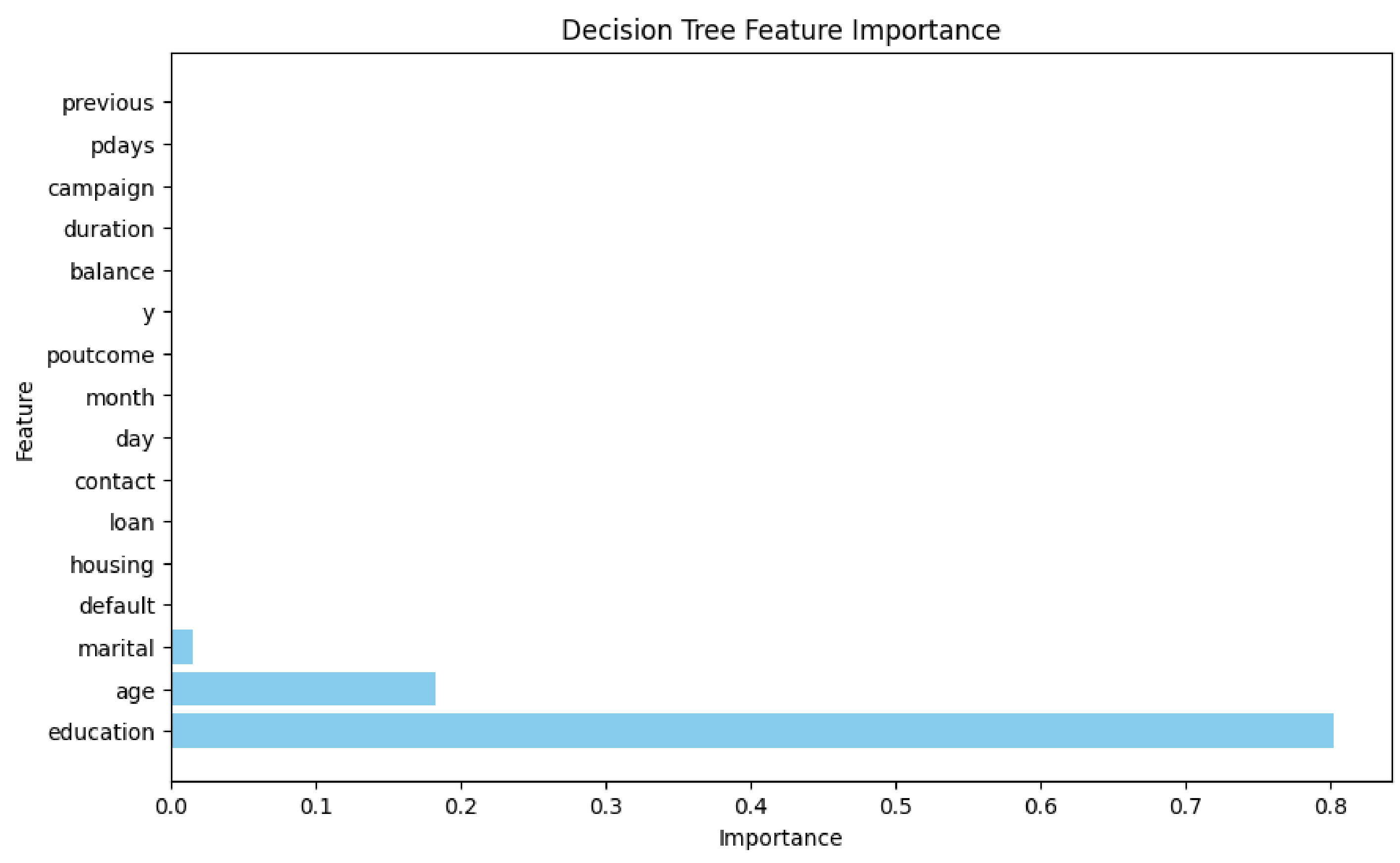
OBSERVATION:

- Root Node: Education
- Splitting Criteria:
  - Education Split:
    - The initial split is based on the education feature.
    - If education is less than or equal to 1.50, it follows the left branch of the tree. Otherwise, it follows the right branch.
  - Age Split:
    - Within the left branch (when education is less than or equal to 1.50), there is a further split based on the age feature.
    - If age is less than or equal to 0.50, it follows the left branch again. Otherwise, it follows the right branch.
  - Education Split (Again):
    - Within the left branch (when age is less than or equal to 0.50), there is another split based on the education feature.
    - This split further refines the decision based on different levels of education.
  - Marital Status and Age Splits:
    - Within the right branch (when education is greater than 1.50), there are splits based on marital and age.
    - These splits further segment the data based on marital status and age to make more specific predictions.
- Interpretation:
  - The decision tree segments the data based on the values of education, age, and possibly marital.
  - It makes predictions based on these splits, assigning each observation to one of the classes.

```
# 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

# Assuming dtc_imp_features is your DataFrame containing feature importance
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 : Model Prediction (Training Subset)
dtc_model_predict = dtc_model.predict(train_bank_inputs); dtc_model_predict
```

```
array([0., 2., 1., ..., 0., 0., 1.])
```

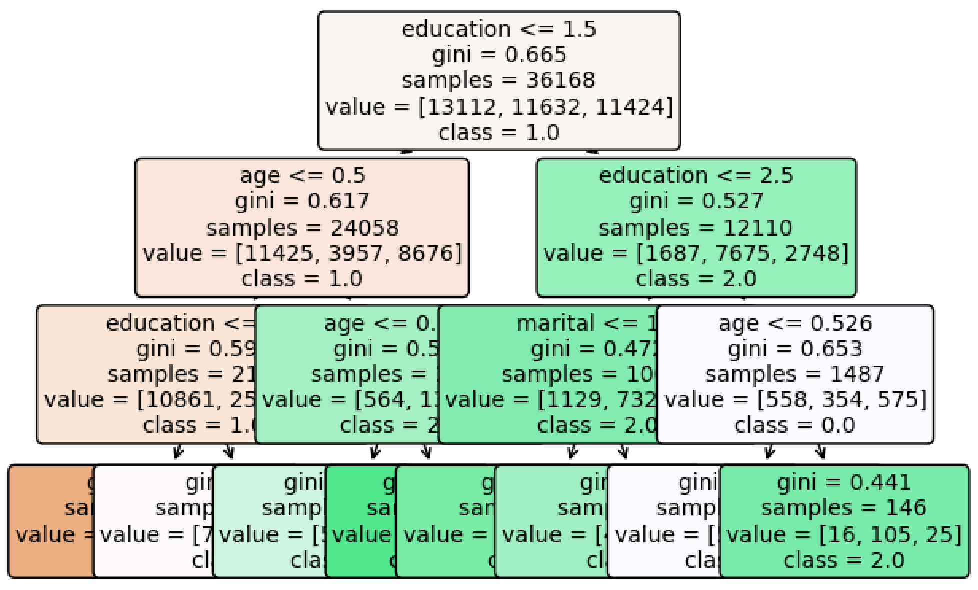
```
# Decision Tree : Prediction (Testing Subset)
dtc_predict = dtc_model.predict(test_bank_inputs); dtc_predict

```

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

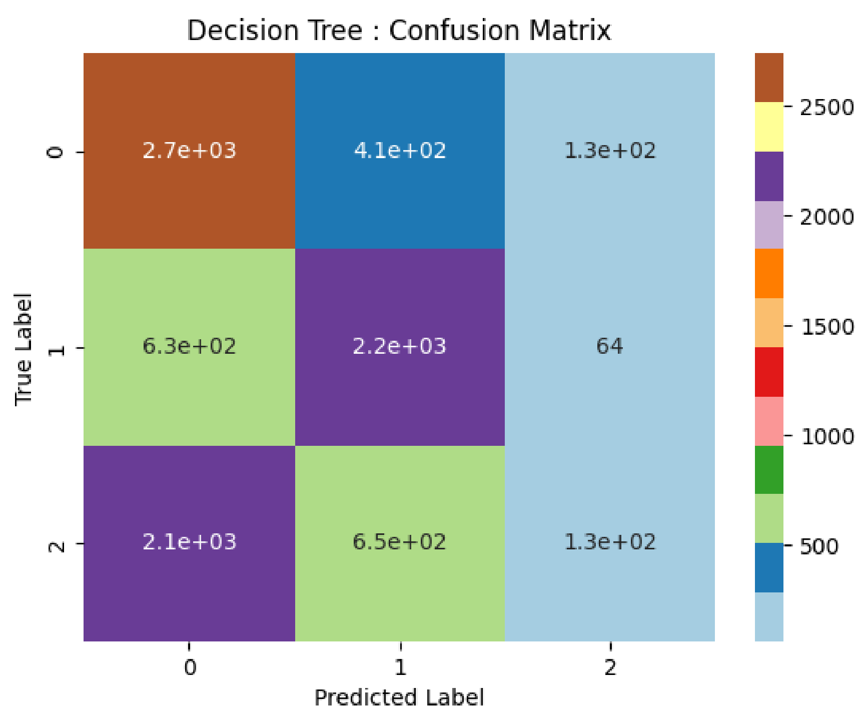
```
# Decision Tree : Prediction Evaluation (Testing Subset)
dtc_predict_conf_mat = pd.DataFrame(confusion_matrix(test_bank_output, dtc_predict)); print(dtc_predict_conf_mat)
dtc_predict_perf = classification_report(test_bank_output, dtc_predict); print(dtc_predict_perf)

```

```
      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()

```



LOGISTIC REGRESSION

```
# Subset bank datset based on Inputs as {'age', 'job', 'duration','balance','campaign','previous','education','marital'} & Output as {y}
bank_inputs = df_ppd[['marital', 'education','default','housing','loan','contact','day','month','poutcome','y',
                      'age','balance','duration','campaign','pdays','previous']]; bank_inputs

#bank_inputs = df_ppd[['age', 'job', 'duration','balance','campaign','previous','education','marital']]; bank_inputs

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

```
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(['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']
```

```
# 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

```

```
from sklearn.linear_model import LogisticRegression

start_time = time.time()
lr = LogisticRegression(random_state=45030)
lr_model = lr.fit(train_bank_inputs, train_bank_output)
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:",lr_model)
```

```
Time taken: 1.0916695594787598 seconds
Memory used: 309.9921875 MB
Model: LogisticRegression(random_state=45030)
```

```
# Extract coefficients and intercept
coefficients = lr_model.coef_
intercept = lr_model.intercept_

# Display coefficients
print("Coefficients:", coefficients)
print("Intercept:", intercept)

```

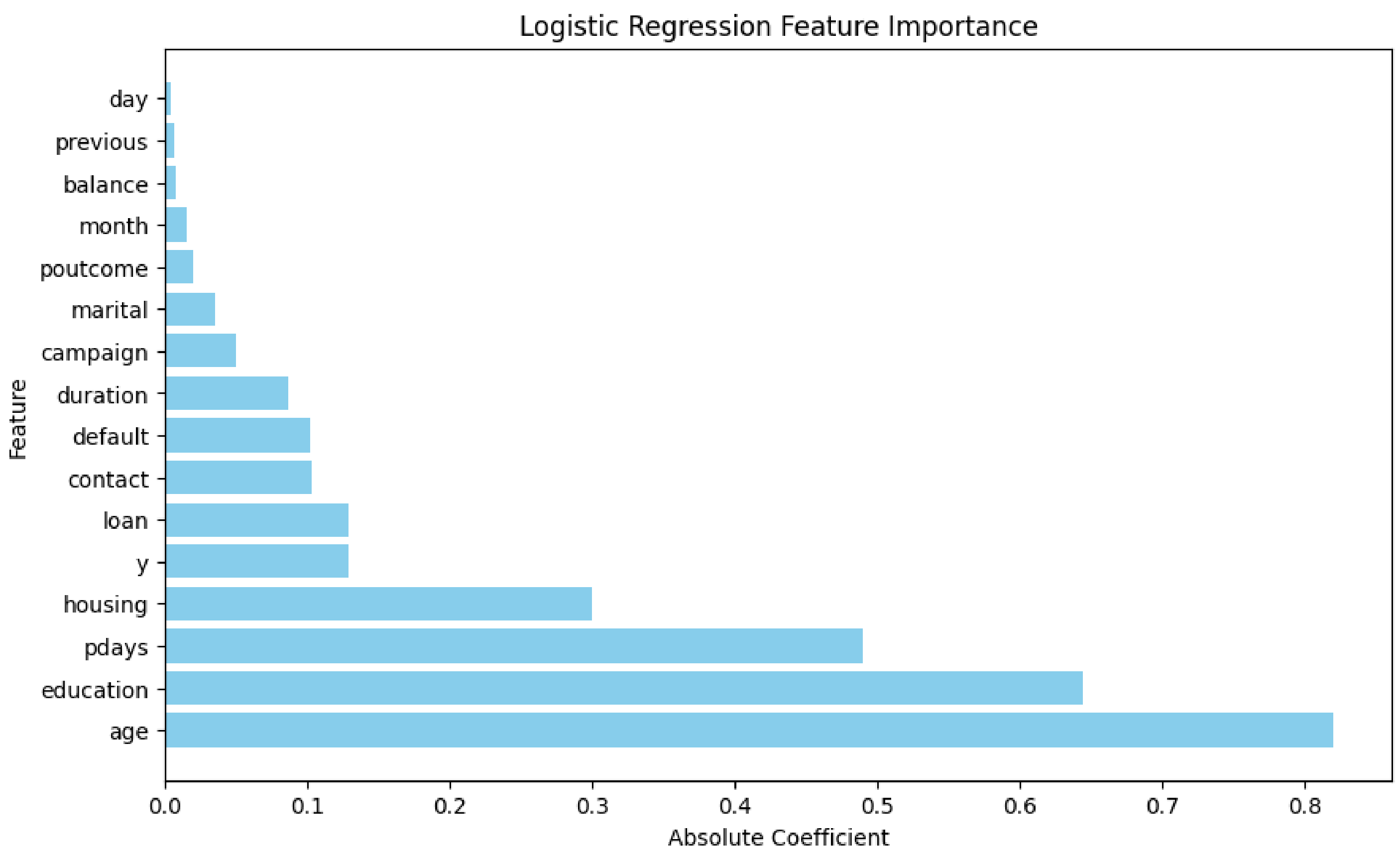
```
Coefficients: [[-3.51872122e-02 -6.43992010e-01  1.02407164e-01  3.00373032e-01
   1.29059790e-01  1.03359071e-01 -4.23885962e-03  1.57397858e-02
  -2.00665330e-02 -1.29066654e-01 -8.20251122e-01 -7.26023396e-03
   8.61047526e-02  4.94994862e-02  4.90539785e-01  6.64027074e-03]
 [-6.18511671e-02  6.65081930e-01 -2.26978525e-02 -2.54482552e-01
  -1.72591126e-01 -1.23410021e-01  4.88290867e-04 -3.87014807e-03
  -5.28492537e-02  5.27415937e-02  2.43391091e+00  1.00394272e-01
  -5.88151572e-02 -3.67087881e-02 -5.17063674e-01 -1.22062826e-03]
 [ 9.70383793e-02 -2.10899196e-02 -7.97093116e-02 -4.58904805e-02
   4.35313356e-02  2.00509500e-02  3.75056875e-03 -1.18696377e-02
   7.29157867e-02  7.63250605e-02 -1.61365979e+00 -9.31340382e-02
  -2.72895954e-02 -1.27906980e-02  2.65238890e-02 -5.41964247e-03]]
Intercept: [ 0.90337427 -1.14299614  0.23962187]
```

These coefficients represent the weights assigned to each feature in the logistic regression model. Positive coefficients indicate a positive association with the target class, while negative coefficients indicate a negative association. The magnitude of the coefficient reflects the strength of the association.

```
# Extract coefficients and feature names
coefficients = lr_model.coef_[0]
feature_names = bank_inputs.columns

# Create a DataFrame to store coefficients and feature names
lr_imp_features = pd.DataFrame({'feature': feature_names, 'coefficient': coefficients})

# Sort features by absolute coefficient value
lr_imp_features['abs_coefficient'] = np.abs(lr_imp_features['coefficient'])
lr_imp_features.sort_values('abs_coefficient', ascending=False, inplace=True)

# Display the DataFrame
print(lr_imp_features)

```

```
      feature  coefficient  abs_coefficient
10        age    -0.820251         0.820251
1   education    -0.643992         0.643992
14      pdays     0.490540         0.490540
3     housing     0.300373         0.300373
9           y    -0.129067         0.129067
4        loan     0.129060         0.129060
5     contact     0.103359         0.103359
2     default     0.102407         0.102407
12   duration     0.086105         0.086105
13   campaign     0.049499         0.049499
0     marital    -0.035187         0.035187
8    poutcome    -0.020067         0.020067
7       month     0.015740         0.015740
11    balance    -0.007260         0.007260
15   previous     0.006640         0.006640
6         day    -0.004239         0.004239
```

```
import matplotlib.pyplot as plt

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(lr_imp_features['feature'], lr_imp_features['abs_coefficient'], color='skyblue')
plt.xlabel('Absolute Coefficient')
plt.ylabel('Feature')
plt.title('Logistic Regression Feature Importance')
plt.show()

```



In logistic regression, the absolute magnitude of the coefficients indicates the importance of each feature. Features with larger absolute coefficients have a greater influence on the predicted outcome. Positive coefficients indicate a positive association with the target class, while negative coefficients indicate a negative association.

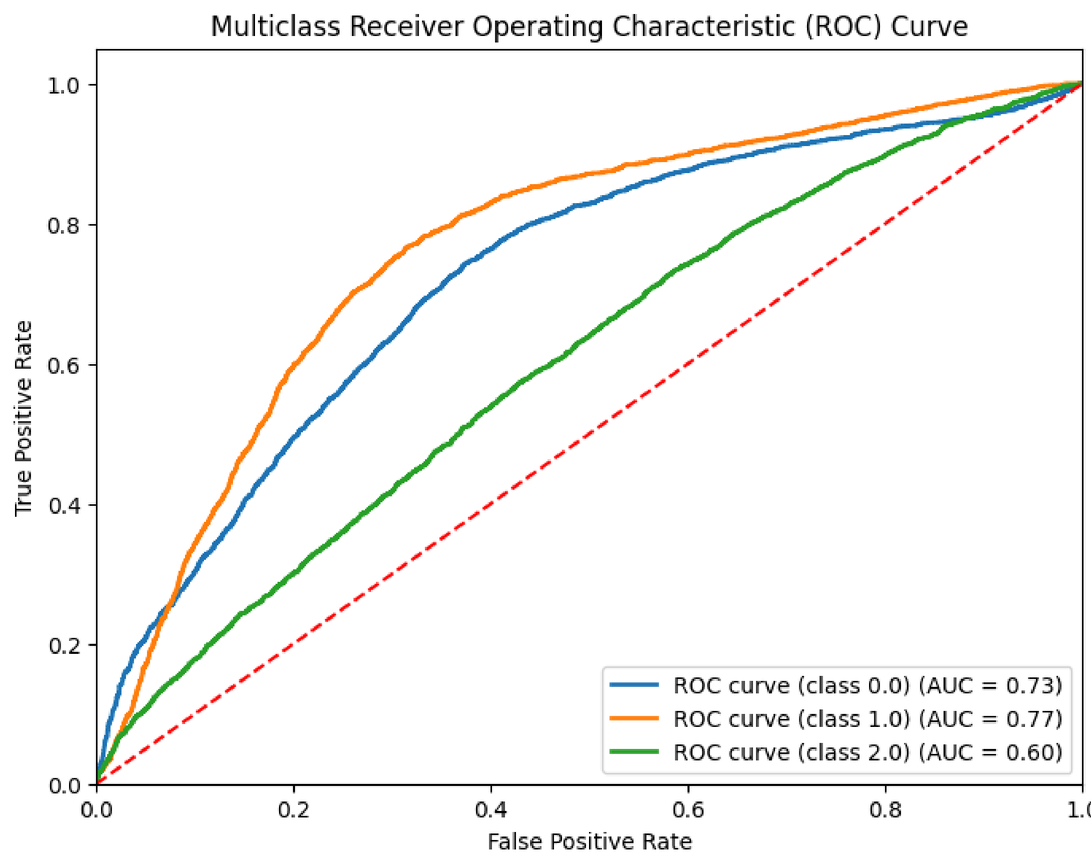
```
# Logistic Regression: Prediction (Testing Subset)
lr_predict = lr_model.predict(test_bank_inputs)
print(lr_predict)
```

```
[1. 0. 0. ... 0. 1. 0.]
```

```
# Logistic Regression: Prediction Evaluation (Testing Subset)
lr_predict_conf_mat = pd.DataFrame(confusion_matrix(test_bank_output, lr_predict))
print(lr_predict_conf_mat)

lr_predict_perf = classification_report(test_bank_output, lr_predict)
print(lr_predict_perf)

```

```
      0     1    2
0  2298   574  406
1   615  1726  568
2  1363   767  726
              precision    recall  f1-score   support

         0.0       0.54      0.70      0.61      3278
         1.0       0.56      0.59      0.58      2909
         2.0       0.43      0.25      0.32      2856

    accuracy                           0.53      9043
   macro avg       0.51      0.52      0.50      9043
weighted avg       0.51      0.53      0.51      9043
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

# Get the probabilities for each class
lr_probabilities = lr_model.predict_proba(test_bank_inputs)

# Initialize an empty array to store ROC-AUC scores for each class
roc_auc_scores = []

# Plot the ROC-AUC curve
plt.figure(figsize=(8, 6))
for i in range(len(lr_model.classes_)):
    fpr, tpr, _ = roc_curve(test_bank_output, lr_probabilities[:, i], pos_label=lr_model.classes_[i])
    roc_auc = auc(fpr, tpr)  # Compute AUC score for the current class
    roc_auc_scores.append(roc_auc)  # Append AUC score to the array
    plt.plot(fpr, tpr, lw=2, label='ROC curve (class {}) (AUC = {:.2f})'.format(lr_model.classes_[i], roc_auc))

plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print("ROC-AUC scores for each class:", roc_auc_scores)

```

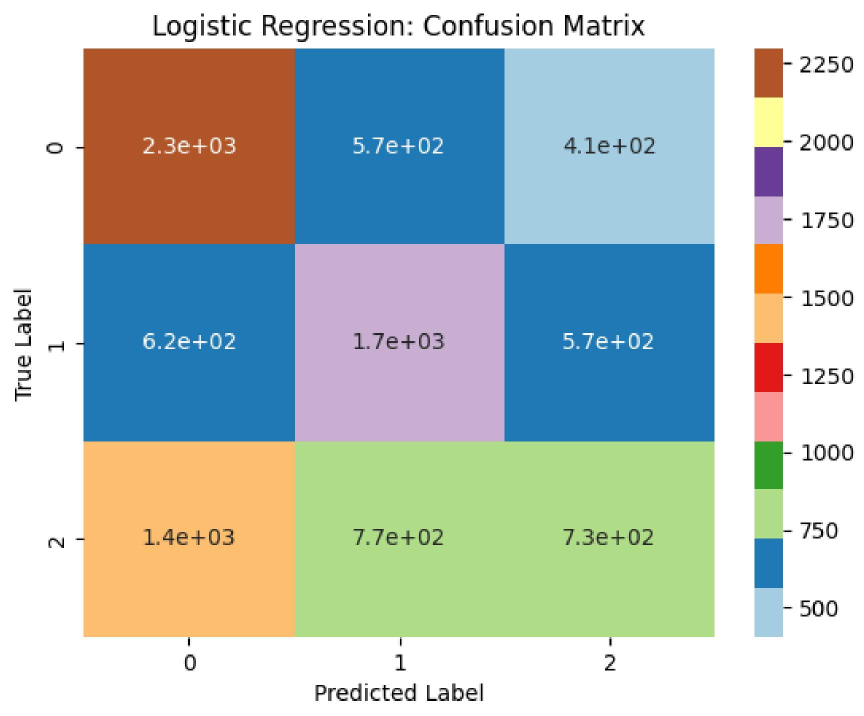


```
ROC-AUC scores for each class: [0.7257926506283578, 0.7659449447051823, 0.602533707842277]
```

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. TPR represents the proportion of actual positive cases correctly classified as positive (sensitivity), while FPR represents the proportion of actual negative cases incorrectly classified as positive.

The logistic regression model achieved ROC-AUC scores of 0.726, 0.766, and 0.603 for classes 0, 1, and 2, respectively indicating better performance in distinguishing between the classes.

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

```



K-NEAREST NEIGHBOR

```
from sklearn.neighbors import KNeighborsClassifier

start_time = time.time()
knn = KNeighborsClassifier()
knn_model = knn.fit(train_bank_inputs, train_bank_output)
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:", knn_model)

```

```
Time taken: 0.0221707820892334 seconds
Memory used: 307.90234375 MB
Model: KNeighborsClassifier()
```

```
print("Number of neighbors:", knn_model.n_neighbors)
print("Algorithm:", knn_model.algorithm)
print("Leaf size:", knn_model.leaf_size)

```

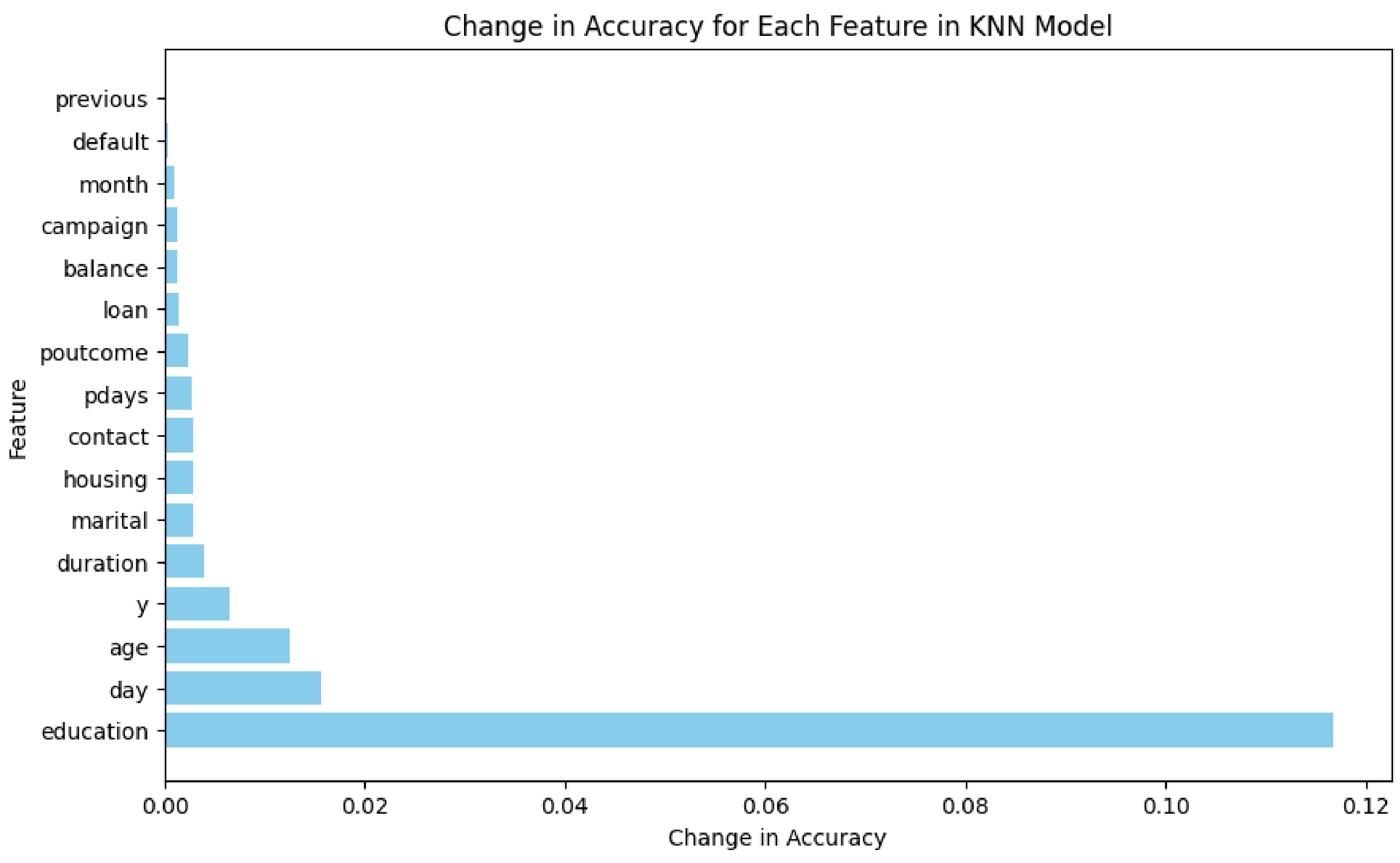
```
Number of neighbors: 5
Algorithm: auto
Leaf size: 30
```

Since K-nearest neighbors (KNN) doesn't inherently provide feature importances like decision trees, we can't directly use the feature_importances_ attribute. However, we can still gain insights into feature importance by observing the effect of each feature on the model's performance.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import pandas as pd

# Train the KNN model
k = 5  # Specify the number of neighbors (k)
knn_model = KNeighborsClassifier(n_neighbors=k)
knn_model.fit(train_bank_inputs, train_bank_output)

# Calculate accuracy with all features
accuracy_all_features = accuracy_score(test_bank_output, knn_model.predict(test_bank_inputs))
print("Accuracy with all features:", accuracy_all_features)

# Initialize an empty DataFrame to store feature importance
knn_imp_features = pd.DataFrame(columns=['feature', 'accuracy_change'])

# Iterate over each feature
for feature in train_bank_inputs.columns:
    # Train KNN model without the current feature
    knn_model_without_feature = KNeighborsClassifier(n_neighbors=k)
    knn_model_without_feature.fit(train_bank_inputs.drop(feature, axis=1), train_bank_output)

    # Calculate accuracy without the current feature
    accuracy_without_feature = accuracy_score(test_bank_output, knn_model_without_feature.predict(test_bank_inputs.drop(feature, axis=1)))

    # Calculate change in accuracy
    accuracy_change = accuracy_all_features - accuracy_without_feature

    # Append feature importance to DataFrame
    #knn_imp_features = knn_imp_features.append({'feature': feature, 'accuracy_change': accuracy_change}, ignore_index=True)
    knn_imp_features = pd.concat([knn_imp_features, pd.DataFrame({'feature': [feature], 'accuracy_change': [accuracy_change]})], ignore_index=True)

# Sort features by absolute change in accuracy
knn_imp_features['abs_accuracy_change'] = knn_imp_features['accuracy_change'].abs()
knn_imp_features.sort_values('abs_accuracy_change', ascending=False, inplace=True)

# Display the DataFrame
print(knn_imp_features)

```

```
Accuracy with all features: 0.5323454605772421
      feature  accuracy_change  abs_accuracy_change
1   education         0.116775             0.116775
6         day        -0.015592             0.015592
10        age         0.012496             0.012496
9           y        -0.006524             0.006524
12   duration        -0.003981             0.003981
0     marital         0.002875             0.002875
3     housing        -0.002875             0.002875
5     contact        -0.002875             0.002875
14      pdays         0.002654             0.002654
8    poutcome        -0.002322             0.002322
4        loan        -0.001438             0.001438
11    balance        -0.001327             0.001327
13   campaign         0.001327             0.001327
7       month        -0.000885             0.000885
2     default         0.000332             0.000332
15   previous         0.000221             0.000221
```

```
import matplotlib.pyplot as plt

# Plot the change in accuracy for each feature
plt.figure(figsize=(10, 6))
plt.barh(knn_imp_features['feature'], knn_imp_features['abs_accuracy_change'], color='skyblue')
plt.xlabel('Change in Accuracy')
plt.ylabel('Feature')
plt.title('Change in Accuracy for Each Feature in KNN Model')
plt.show()

```



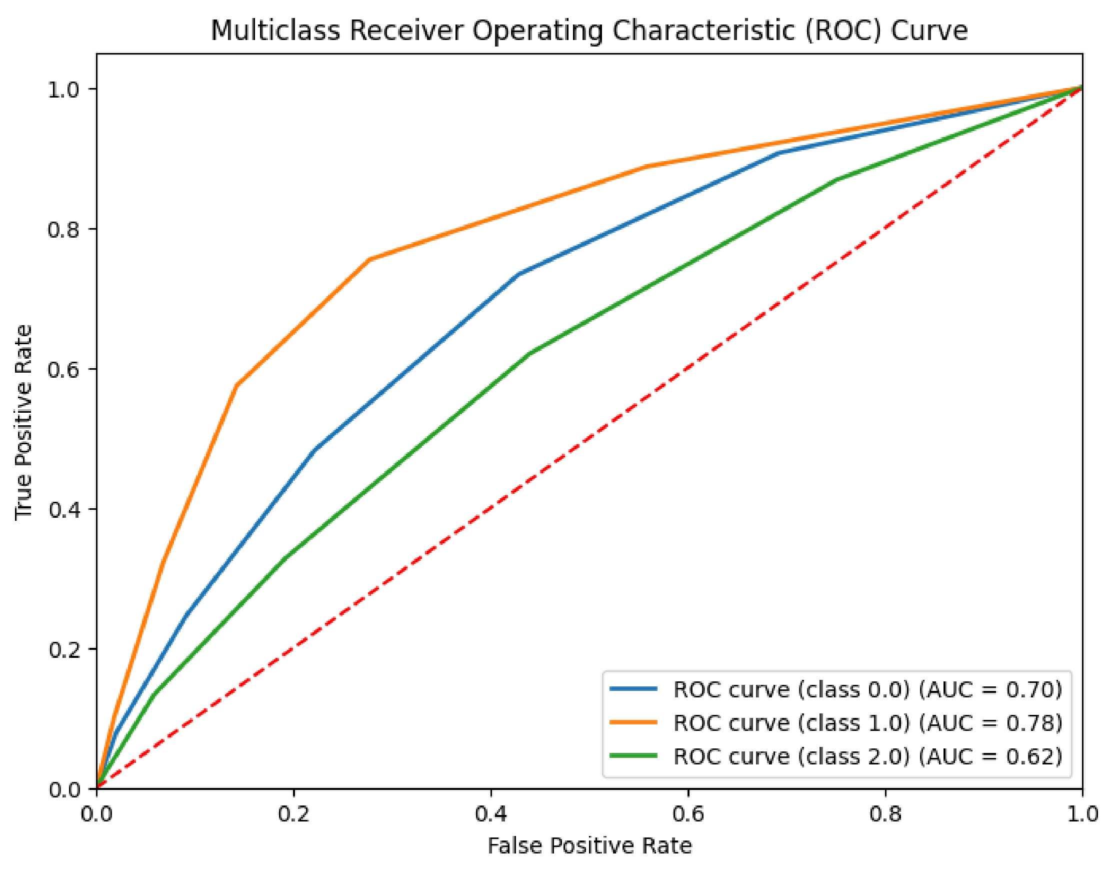
```
# KNN: Prediction (Testing Subset)
knn_predict = knn_model.predict(test_bank_inputs)
print(knn_predict)

```

```
[1. 2. 0. ... 0. 0. 0.]
```

```
# KNN: Prediction Evaluation (Testing Subset)
knn_predict_conf_mat = pd.DataFrame(confusion_matrix(test_bank_output, knn_predict))
print(knn_predict_conf_mat)

knn_predict_perf = classification_report(test_bank_output, knn_predict)
print(knn_predict_perf)

```

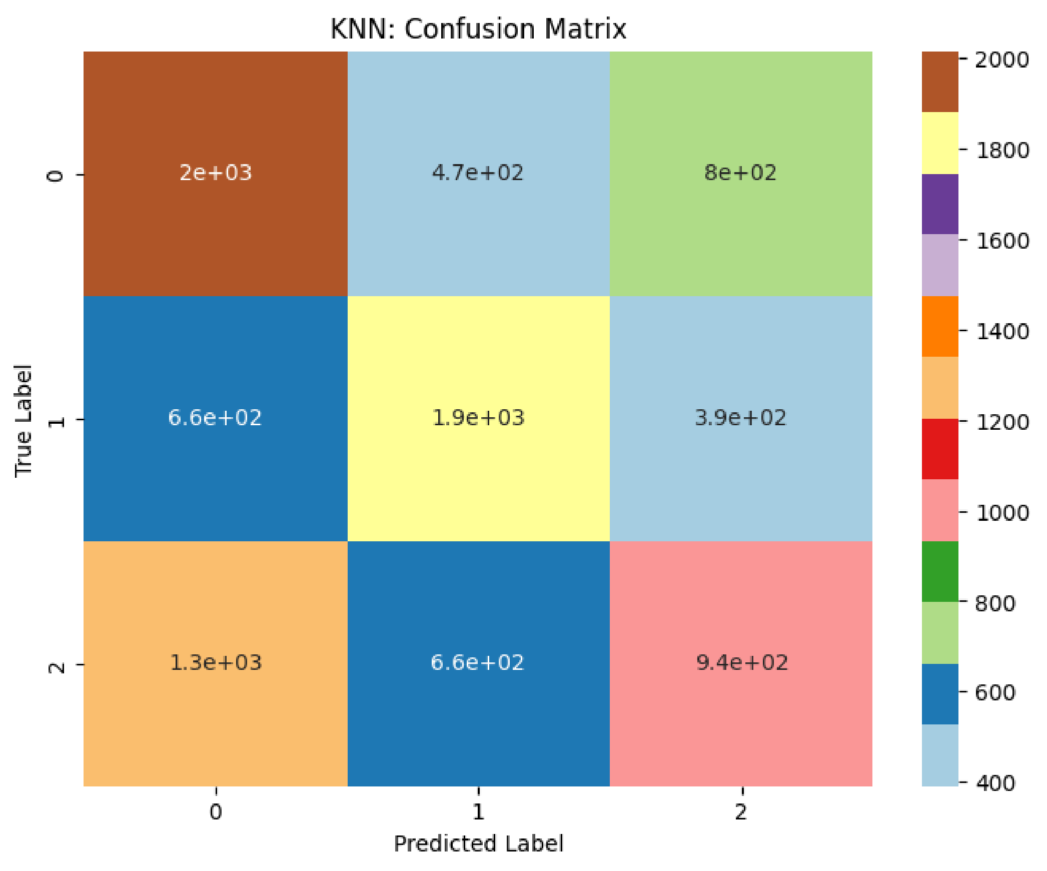
```
      0     1    2
0  2016   466  796
1   657  1861  391
2  1264   655  937
              precision    recall  f1-score   support

         0.0       0.51      0.62      0.56      3278
         1.0       0.62      0.64      0.63      2909
         2.0       0.44      0.33      0.38      2856

    accuracy                           0.53      9043
   macro avg       0.53      0.53      0.52      9043
weighted avg       0.53      0.53      0.52      9043
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

# Get the probabilities for each class using predict_proba
knn_probabilities = knn_model.predict_proba(test_bank_inputs)

# Initialize an empty array to store ROC-AUC scores for each class
roc_auc_scores = []

# Plot the ROC-AUC curve
plt.figure(figsize=(8, 6))
for i in range(len(knn_model.classes_)):
    fpr, tpr, _ = roc_curve(test_bank_output, knn_probabilities[:, i], pos_label=knn_model.classes_[i])
    roc_auc = auc(fpr, tpr)  # Compute AUC score for the current class
    roc_auc_scores.append(roc_auc)  # Append AUC score to the array
    plt.plot(fpr, tpr, lw=2, label='ROC curve (class {}) (AUC = {:.2f})'.format(knn_model.classes_[i], roc_auc))

plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print("ROC-AUC scores for each class:", roc_auc_scores)

```



```
ROC-AUC scores for each class: [0.6952427997737287, 0.7819260644281831, 0.6164670126980807]
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute confusion matrix
knn_conf_mat = confusion_matrix(test_bank_output, knn_predict)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(knn_conf_mat, annot=True, cmap='Paired')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('KNN: Confusion Matrix')
plt.show()

```



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

k_values = [7, 9, 11, 13]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_bank_inputs, train_bank_output)
    knn_pred = knn.predict(test_bank_inputs)
    accuracy = accuracy_score(test_bank_output, knn_pred)
    print(f'Accuracy for k={k}: {accuracy}')

```

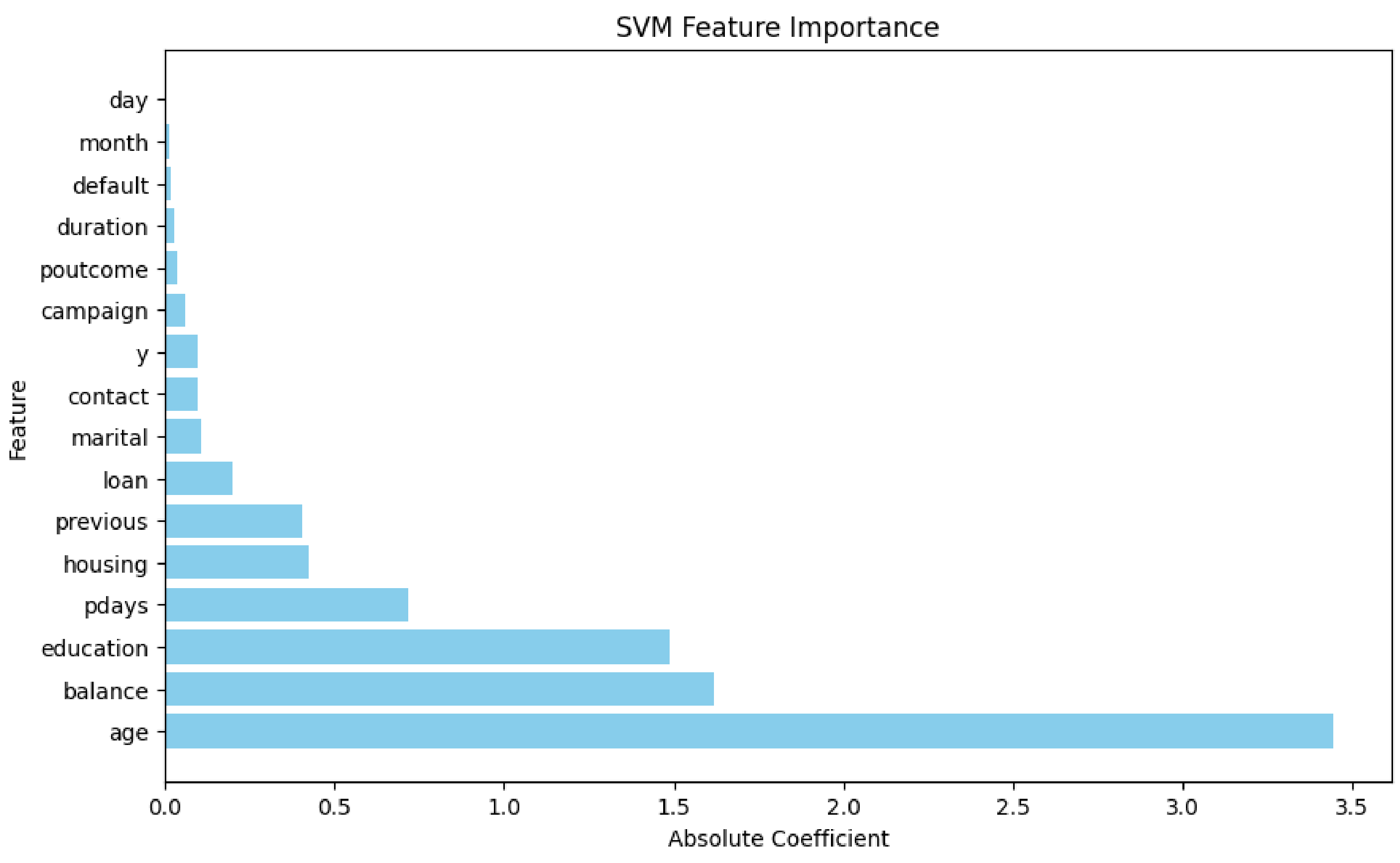
```
Accuracy for k=7: 0.5415238305871946
Accuracy for k=9: 0.5473847174610196
Accuracy for k=11: 0.5486011279442663
Accuracy for k=13: 0.5525821077076192
```

SUPPORT VECTOR MACHINE(SVM)

```
from sklearn.svm import SVC

start_time = time.time()
# Define the SVM model with a linear kernel and regularization parameter C=45030
svm_model = SVC(kernel='linear', C=1)

# Train the SVM model on the training dataset
svm_model.fit(train_bank_inputs, train_bank_output)
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")
```

```
Time taken: 215.94265294075012 seconds
Memory used: 540.15234375 MB
```

```
def plot_feature_importance_svm(model, feature_names):
    # Get coefficients from the model
    coefficients = model.coef_[0]

    # Create DataFrame to store feature importance
    svm_imp_features = pd.DataFrame({'feature': feature_names, 'coefficient': coefficients})

    # Sort features by absolute coefficient values
    svm_imp_features['abs_coefficient'] = np.abs(svm_imp_features['coefficient'])
    svm_imp_features.sort_values('abs_coefficient', ascending=False, inplace=True)

    # Plot feature importance
    plt.figure(figsize=(10, 6))
    plt.barh(svm_imp_features['feature'], svm_imp_features['abs_coefficient'], color='skyblue')
    plt.xlabel('Absolute Coefficient')
    plt.ylabel('Feature')
    plt.title('SVM Feature Importance')
    plt.show()

    # Return the DataFrame containing feature importance values
    return svm_imp_features

# Call the function and store the returned DataFrame
svm_feature_importance_df = plot_feature_importance_svm(svm_model, bank_inputs_names)
print(svm_feature_importance_df)
```



```
      feature  coefficient  abs_coefficient
10        age    -3.443606         3.443606
11    balance    -1.616925         1.616925
1   education    -1.489817         1.489817
14      pdays     0.717645         0.717645
3     housing     0.425993         0.425993
15   previous    -0.405033         0.405033
4        loan     0.201516         0.201516
0     marital    -0.106145         0.106145
5     contact     0.099003         0.099003
9           y    -0.098059         0.098059
13   campaign     0.060813         0.060813
8    poutcome     0.039008         0.039008
12   duration    -0.028575         0.028575
2     default     0.016498         0.016498
7       month     0.016070         0.016070
6         day    -0.003112         0.003112
```

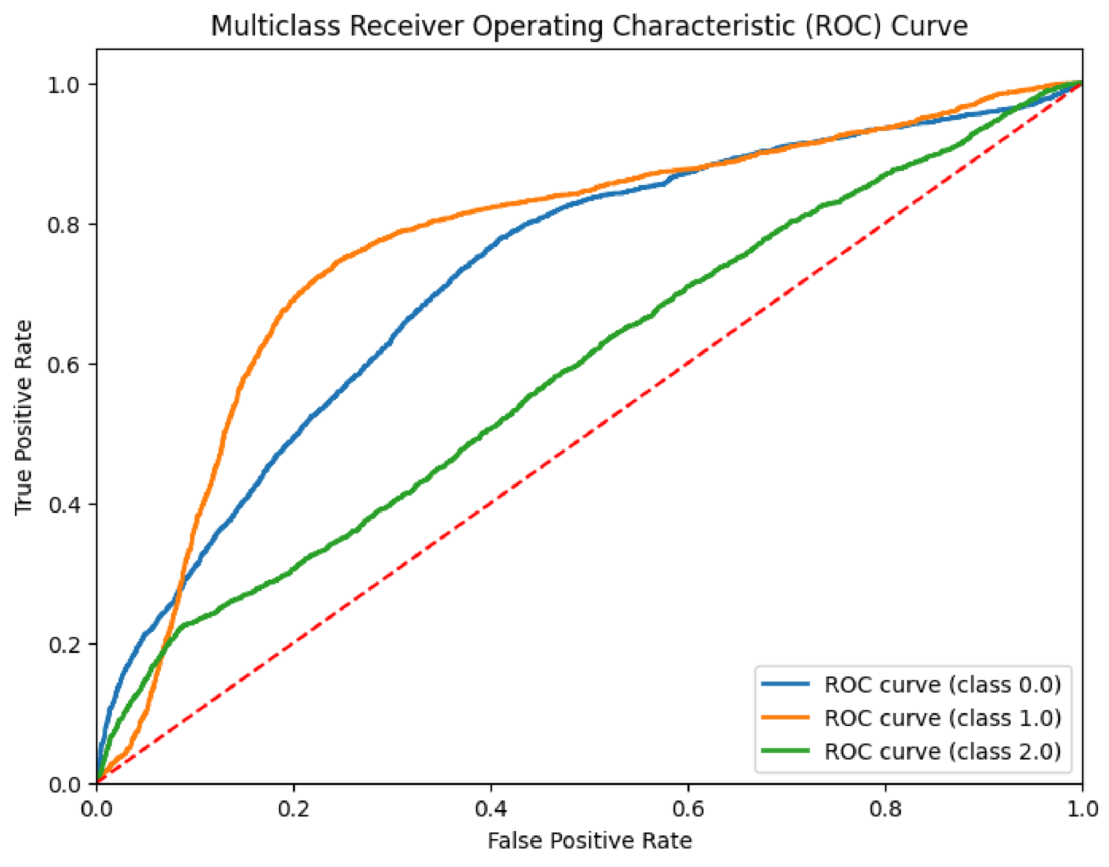
```
# SVM Prediction (Testing Subset)
svm_predict = svm_model.predict(test_bank_inputs)
print(svm_predict)
```

```
[1. 0. 0. ... 0. 1. 2.]
```

```python
# SVM Prediction Evaluation (Testing Subset)
svm_predict_conf_mat = pd.DataFrame(confusion_matrix(test_bank_output, svm_predict))
print(svm_predict_conf_mat)

svm_predict_perf = classification_report(test_bank_output, svm_predict)
print(svm_predict_perf)

```

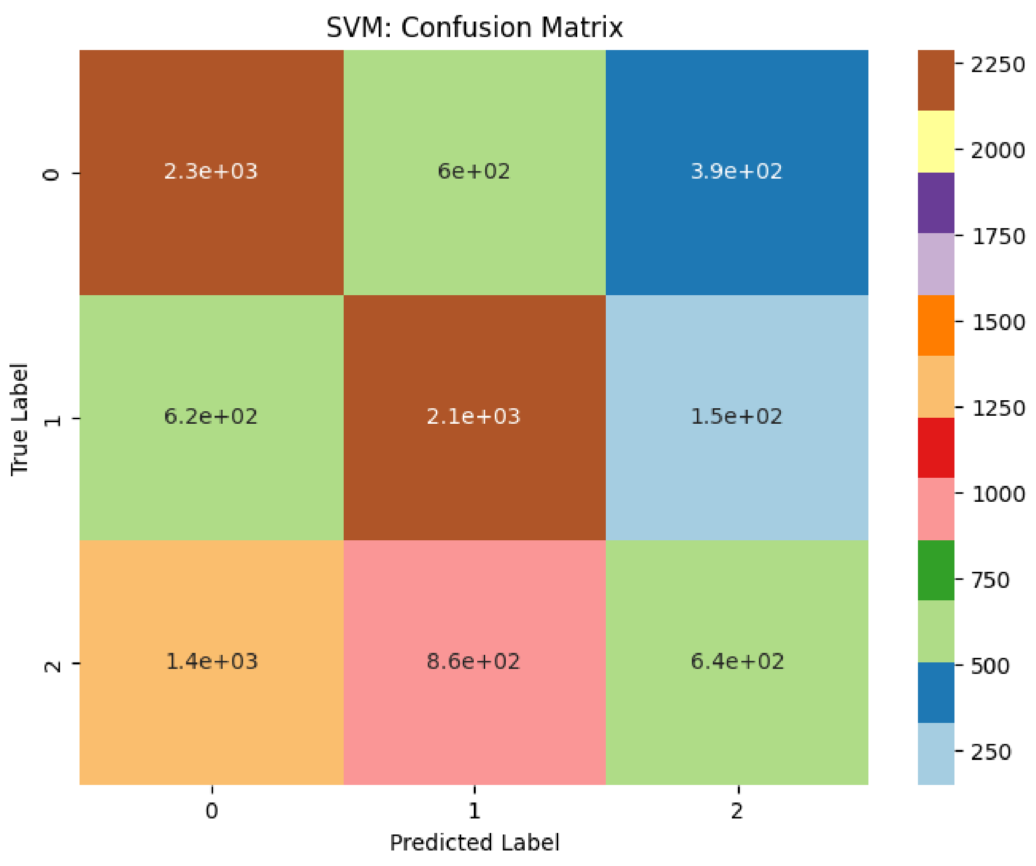
```
      0     1    2
0  2289   596  393
1   625  2134  150
2  1354   864  638
              precision    recall  f1-score   support

         0.0       0.54      0.70      0.61      3278
         1.0       0.59      0.73      0.66      2909
         2.0       0.54      0.22      0.32      2856

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

```python
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

# Get the decision function scores for each class
svm_scores = svm_model.decision_function(test_bank_inputs)

# Compute the ROC-AUC score for each class
roc_auc_scores = []

# For each class, compute ROC-AUC
for i in range(len(svm_model.classes_)):
    y_true = (test_bank_output == svm_model.classes_[i]).astype(int)
    roc_auc_scores.append(roc_auc_score(y_true, svm_scores[:, i]))

# Plot the ROC-AUC curve
plt.figure(figsize=(8, 6))
for i in range(len(svm_model.classes_)):
    fpr, tpr, _ = roc_curve((test_bank_output == svm_model.classes_[i]).astype(int), svm_scores[:, i])
    plt.plot(fpr, tpr, lw=2, label='ROC curve (class {})'.format(svm_model.classes_[i]))

plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Print the ROC-AUC score for each class
print("ROC-AUC scores for each class:", roc_auc_scores)

```



```
ROC-AUC scores for each class: [0.7258362009708075, 0.7693952175897899, 0.5888670402701245]
```

```
# Compute confusion matrix for SVM
svm_conf_mat = confusion_matrix(test_bank_output, svm_predict)

# Plot confusion matrix for SVM
plt.figure(figsize=(8, 6))
sns.heatmap(svm_conf_mat, annot=True, cmap='Paired')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('SVM: Confusion Matrix')
plt.show()

```



```
Start coding or generate with AI.
```