Final Report Bank Marketing Campaign:
Opening a Term Deposit

By

Md Nazmun Hasan Nafees

Abstract

This project investigates the factors influencing client behavior in subscribing to term deposit accounts using the **Bank Marketing dataset**, which comprises 45,211 records across 17 features. The dataset includes client demographics (e.g., age, job, marital status), financial attributes (e.g., balance, loan status), and campaign-related data (e.g., contact duration and previous outcomes). The target variable, "deposit," represents whether a client subscribed to a term deposit.

The primary goal of this analysis is to provide actionable insights to financial institutions, enabling the design of more effective marketing strategies. Rigorous data preprocessing was performed, including handling missing values, encoding categorical features, and scaling numerical variables. Post-preprocessing, machine learning models—**Decision Tree Classifier** and **Logistic Regression**—were implemented to predict client behavior and identify key factors affecting subscriptions. These models were chosen for their interpretability and predictive capability.

The analysis revealed that **call duration** was the most significant factor influencing client decisions, while features such as **balance, loan status, and age** also played vital roles. Logistic Regression outperformed the Decision Tree in terms of accuracy (78.51% vs. 74.57%) and provided more consistent cross-validation results, indicating its robustness. Recommendations include focusing on clients with higher balances and designing engaging questionnaires to extend call durations, thus improving the likelihood of subscriptions.

The findings of this project can aid banks in optimizing resource allocation for future campaigns and tailoring their strategies to target high-potential clients effectively. By leveraging machine learning and data-driven insights, this study highlights practical solutions to enhance marketing outcomes in the financial sector.

Introduction

Access to financial services is a crucial driver of economic development, and effective marketing strategies play a pivotal role in expanding the reach of these services. In the context of banking, understanding client behavior is essential for designing campaigns that resonate with potential customers. This project utilizes the **Bank Marketing dataset**, sourced from the UCI Machine Learning Repository, to analyze factors influencing a client's decision to subscribe to a term deposit.

The dataset consists of 45,211 records and 17 features, capturing client demographics (e.g., age, job, marital status), financial attributes (e.g., balance, loan status), and campaign-specific information (e.g., call duration and contact type). The target variable, **"deposit,"** indicates whether a client subscribed to a term deposit during the marketing campaign.

Key Objectives of the Study:

- **Identify Key Factors Influencing Client Decisions:**
  By examining various demographic, financial, and campaign-related features, the study seeks to pinpoint the most influential factors affecting subscription rates.
- **Support Data-Driven Marketing Strategies:**
  The analysis aims to provide actionable insights for banks, helping optimize future marketing campaigns to target high-potential clients more effectively.
- **Bridge Data Science and Marketing Strategy:**
  This project demonstrates the application of machine learning and statistical techniques in deriving insights that can transform marketing strategies in the financial sector.

Through a combination of exploratory data analysis and predictive modeling, this project offers practical solutions to enhance the efficiency and effectiveness of marketing campaigns in banking.

Data Preparation and Methodology

1.Low variance Column

Step: Removing Low Variance Columns

- **Why?** Columns with a single unique value, such as a constant or low variance, provide no meaningful variability to the dataset. These features do not contribute to distinguishing patterns and often add unnecessary complexity to the model.
- **Impact:**
    - Reduces **dimensionality**: Makes the dataset simpler and easier to handle.
    - Improves **model efficiency**: The model processes fewer features, reducing computational cost.
    - Avoids **overfitting**: Irrelevant features may increase noise, harming model performance.

2.Handling Duplicate Data

Step: Initial Check for Duplicates and Missing Data

- Used df.isnull().sum() to identify missing values. While no obvious duplicates were found, this step showed some **disguised missing values**.
- **Observation:** A category labeled "unknown" was identified. Such placeholder values often represent missing information but are not easily detectable.

Step: Replacing "Unknown" with NaN

- **Why?** Treating "unknown" as missing values (NaN) allows the use of robust imputation techniques. Without this step, the model might misinterpret "unknown" as meaningful data.
- **Outcome:** Highlighted gaps in critical features, prompting the need for **imputation** to fill these missing values.

```
# Calculate the number of null values in each column

## but lets look at if there is any unknown value ?
unknown =  (df == 'unknown').sum()
unknown
df.replace("unknown", np.nan, inplace=True)
# replace unknown by NaN , pandas can recognize it



null_counts = df.isnull().sum()

# # Iterate over the counts and check if they are zero or not
# for col, count in null_counts.items():
#     if count != 0:
#         print(f"{col} --> {count} --> Not ok")
null_counts
```

3.Imputation: For handling missing values in categorical features, we chose **conditional imputation** over simpler methods like mode imputation. *Mode imputation* could have introduced bias by filling missing values with the most frequent class, potentially leading to incorrect or unrepresentative values. Instead, conditional imputation identifies specific conditions or patterns in the data that influence the values of the missing entries.

In the conditional imputation process, we first analyzed the dataset to understand the factors that significantly determine the missing values, such as relationships between other features. These factors were then used to generate synthetic values for the missing data, ensuring the imputation was contextually accurate and aligned with the underlying patterns in the dataset.

By using conditional imputation, we avoided bias and ensured that the filled values were realistic, preserving the integrity of the dataset and improving the overall quality of the model. This approach made the imputation process more robust and reflective of the real-world relationships present in the data.

```
## we see that there are lots of unknown value, but now we have to handle them.

df['job'].unique()


## using conditional imputation
df ['job'] = np.where(
    (df['job'].isna())& (df['age']>50) & (df['housing'] == 'no') & (df['loan'] == 'no'), "retired",
    np.where(
        (df['job'].isna())& (df['age']<=50) & (df['balance'] <=1000), 'unemployed',
        np.where(
            (df['job'].isna())& (df['balance']>=1000),'self-employed',
            np.where(
                (df['job'].isna())& (df['age']<35) & (df['marital']== 'single'), 'student',
                df['job']
            )
        )
    )
)
print(df.isnull().sum())
## the job missing values are imputated
## lets analyze the job NaN value, and see how can we deal with it.
## why do you think they reported job as missing  lets see
```

4.Outlier Detection and Treatment

Outlier analysis ensures that extreme values do not unduly influence the model's predictions.

- **Why Focus Only on Numerical Features?**
    - Outliers in numerical features can skew the model, leading to inaccuracies in predictions. Categorical features, on the other hand, inherently lack a numerical range for outlier detection.
- **IQR Method for Outlier Detection:**
    - **Process:**
        - Calculated the Interquartile Range (IQR) as the difference between the 75th and 25th percentiles.
        - Identified outliers as values falling below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR.
    - **Treatment:**
        - Outliers were either capped at the boundary values or removed, depending on the feature's importance and the extent of the outliers' influence.

```
## None of them are normally distributed. So we are going to remove outlier by IQR.

# Define columns to remove outliers using IQR
columns = ['age','campaign']

for col in columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    up_bound = Q3 + 1.5 * IQR
    low_bound = Q1 - 1.5 * IQR

    # Filter the DataFrame for the current column
    df = df[(df[col] >= low_bound) & (df[col] <= up_bound)]
```

5.Verifying Data Types: Inconsistent Data type

- Analyzed each feature to ensure its data type accurately reflected its nature. For instance:
  - Ensured numerical features (e.g., age, balance) were not mistakenly treated as categorical.
  - Corrected any mismatches to avoid computational errors during modeling.

6.Normalization of Features

Normalization standardizes feature scales, ensuring fair contribution to the model's learning process.

- **Why Normalize Features?**
  - Features with larger scales (e.g., balance in dollars vs. age in years) can disproportionately impact model weights, leading to biased predictions.
- **Techniques Used:**
  - **Standardization:** Scaled numerical features to have a mean of 0 and standard deviation of 1, maintaining original feature relationships. Mainly used when the data shape was close to normal distribution.
  - **Robust Scaling:** Applied to features with extreme outliers, as it uses the median and IQR for scaling.
  - **Log Transformation:** Applied to features with highly skewed distributions to stabilize variance and normalize skewness.

```
standard = StandardScaler()
min_max = MinMaxScaler()
robust = RobustScaler()


# z normalization, as the feature is close to normally distributed
df.loc[:, 'age'] = standard.fit_transform(df[['age']])

# robust scaling
df.loc[:, 'balance'] = robust.fit_transform(df[['balance']])
df.loc[:, 'campaign'] = robust.fit_transform(df[['campaign']])

# For transformations
df.loc[:, 'duration'] = np.log1p(df['duration'])
df.loc[:, 'pdays'] = np.log1p(df['previous'])
```

7.Feature Engineering

Feature engineering enhances the dataset by refining existing features and introducing new ones.

- **Feature Selection (Feature Deletion):**
    - Removed redundant or irrelevant features that did not contribute significantly to the target variable's prediction.
- **Feature Creation:**
    - **Why Create Features?**
        - Adding new features can capture additional relationships and improve model performance.
    - **Examples:**
        - Created three new features based on domain knowledge, such as interaction terms between education and job or aggregations like average monthly balance.

```
# Create the campaign and contacts features
df['multiple_contacts'] = (df['campaign'] > 1).astype(int)
df['previous_contact'] = (df['previous'] > 0).astype(int)

# Create the job stability feature
stable_jobs = ['management', 'technician', 'admin.']
df['employment_stability'] = df['job'].apply(lambda x: 1 if x in stable_jobs else 0)

# Create the loan affinity feature
df['loan_affinity'] = (df['housing'] == 'yes').astype(int) + (df['loan'] == 'yes').astype(int)

# Display the updated DataFrame
df.info()
```

In this step, several new features were created to enhance the dataset's predictive power. The **multiple_contacts** feature indicates whether a client was contacted multiple times in the current campaign, with a binary value where 1 means multiple contacts and 0 means only one contact. Similarly, the **previous_contact** feature captures whether the client had prior contact in previous campaigns, set to 1 if they were contacted before and 0 otherwise. The **employment_stability** feature was derived by classifying clients into stable and unstable job categories, based on a predefined list of job types such as management, technician, and admin. A value of 1 denotes a stable job, while 0 denotes instability. Lastly, the **loan_affinity** feature measures a client's exposure to loans, taking values from 0 to 2 based on whether they have a housing or personal loan.

These new features provide deeper insights into client behavior, engagement, and financial stability. For instance, **multiple_contacts** and **previous_contact** help assess how client engagement over time influences their likelihood to subscribe to a term deposit. Meanwhile, **employment_stability** reflects the financial security of the client's employment status, while **loan_affinity** captures the client's financial obligations, which could affect their ability to invest in financial products. These additional features aim to improve the model's ability to predict client behavior by offering more granular data on client characteristics.

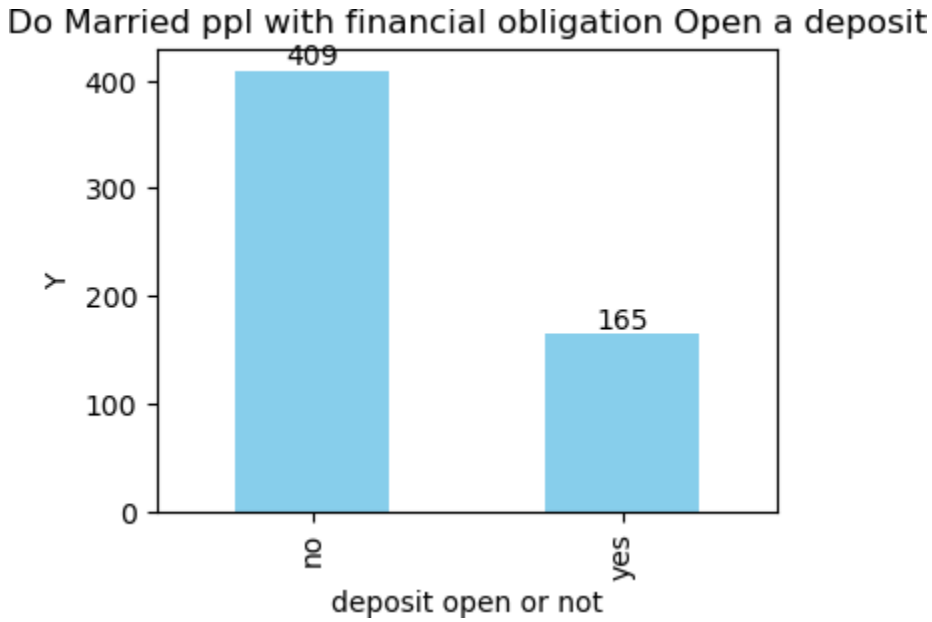
8.Encoding:

The encoding process converts categorical features into numerical values for machine learning models. **Binary Encoding** was applied to features like job, marital, and month, which have many categories, to reduce dimensionality by representing each category as a binary number across multiple columns. **Label Encoding** was used for the **ordinal** feature education, where categories like 'primary', 'secondary', and 'tertiary' were mapped to integers (0, 1, 2). For features with low cardinality like default, housing, loan, and contact, **binary encoding** or **label encoding** was used, mapping categories like 'yes' and 'no' to 1 and 0. The .info() method was used to verify the changes and ensure the dataset is ready for modeling.

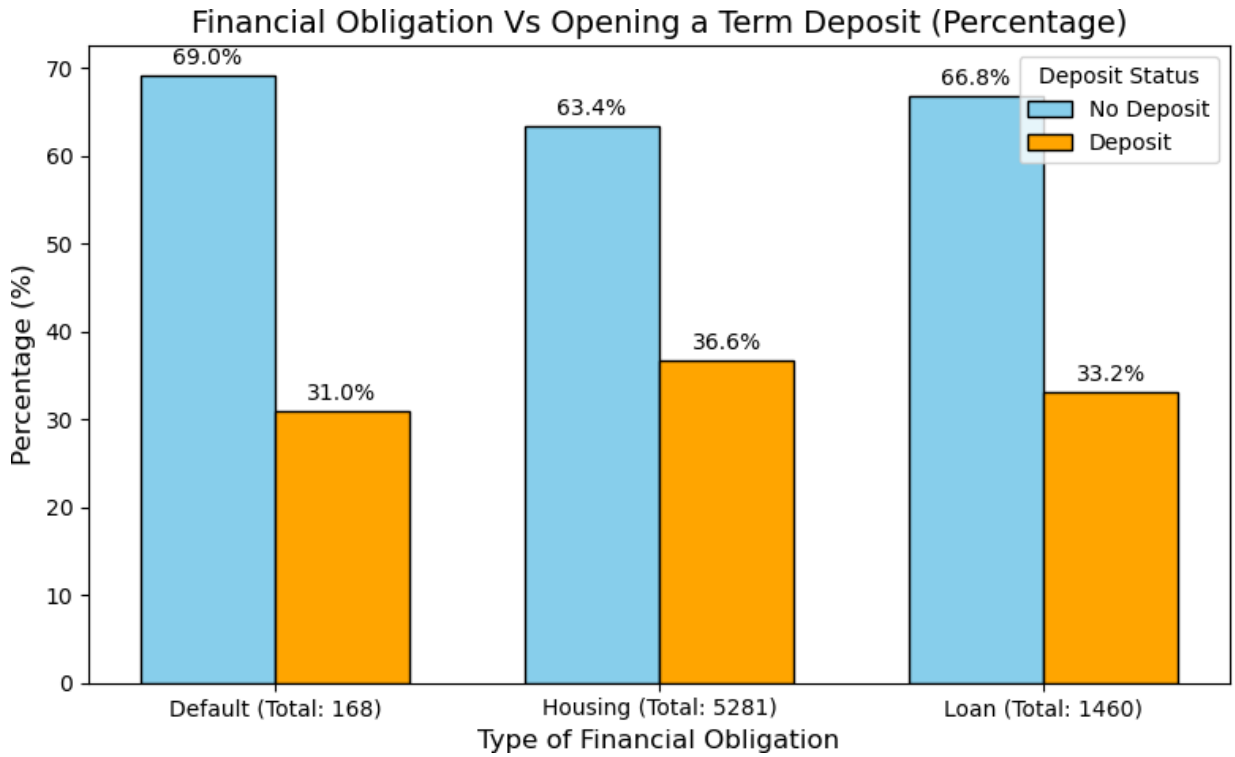
```
# Initialize the Binary Encoder
binary_encoder = ce.BinaryEncoder(cols=['job','marital','month'])


# - moderate, ordinal, --> binary encoding
# Apply Binary Encoding to the 'job' feature in df
df = binary_encoder.fit_transform(df)


## remember that we have to do all the binary in one go, if we are using the


# - low, nominal -- binary encoding

# -low, ordinal
## Appy label encoder
df['education'] = df['education'].map({'primary': 0, 'secondary': 1, 'tertiary': 2})

## 2d binary encoding (low cardinality)
#default
df['default'] = df['default'].map({'no':0, 'yes':1})
#housing
df['housing'] = df['housing'].map({'no':0, 'yes':1})
#loan
df['loan'] = df['loan'].map({'no':0, 'yes':1})
#contact
df['contact'] = df['contact'].map({'telephone':0, 'cellular':1})
```

Findings and Insights

Married individuals with financial obligations, such as housing loans or personal loans, are significantly less likely to open a term deposit, highlighting the impact of financial constraints on savings behavior.



We can see that that 165 ppl opened the bank deposit and 409 did not.This is likely due to several factors, and these financial conditions may indicate certain behavior patterns or priorities that affect their decision-making regarding savings or investments. For instance, people with multiple loans (e.g., housing, personal) or those who have defaulted on credit may be under significant financial pressure. They might prioritize immediate liquidity or paying off debts over locking money in a term deposit.



Although financial constraints vary, the analysis reveals that the disparity in deposit opening behavior remains consistent across different types of financial obligations, with housing loans being the most prevalent, affecting 50% of the surveyed individuals.

In the previous graph, we observed a general disparity between those who opened a deposit (71%) and those who did not (29%). However, in this updated analysis, we see that for each financial obligation category (Default, Housing, Loan), the disparity between those who opened a deposit and those who did not is consistently around a 2:1 ratio. This suggests that the financial obligations, regardless of their type, affect a similar proportion of people in terms of deposit opening behavior.

Notably, the most influential category appears to be the Housing loan, as it accounts for 50% of the total dataset. This indicates that nearly half of the individuals surveyed are dealing with housing debt, which may have a significant impact on their ability or willingness to open a deposit account.

This analysis reinforces the idea that while different financial obligations affect people's behavior similarly, housing loans are the most prevalent in the dataset, with a notable proportion of people in financial constraint due to housing debt.

Machine Learning Methodology:

```
# Split the features and target variable
X = df.drop(columns=['deposit'])  # Features
y = df['deposit']  # Target variable

# Step 1: Split into train and test sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 2: Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)
y_pred_tree_test = decision_tree.predict(X_test)
```

(parameter) random_state: Int | Ra

```
# Step 3: Logistic Regression
log_reg = LogisticRegression(max_iter=10000, random_state=42)
log_reg.fit(X_train, y_train)
y_pred_log_reg_test = log_reg.predict(X_test)
```

In this code, the dataset is first split into features (X) and the target variable (y). Then, the train_test_split function is used to partition the data into training and testing sets, with 80% of the data used for training and 20% reserved for testing. This setup allows for training the models on one set of data and evaluating their performance on another set to assess how well they generalize.

The **Decision Tree Classifier** is then trained on the training data and used to predict the target variable (y) on the test set. The model's performance is evaluated using the accuracy_score and classification_report, which provides metrics like precision, recall, and F1-score. Additionally, 10-fold cross-validation is performed on the training data to estimate the model's performance more robustly by splitting the training data into 10 subsets, training the model on 9, and testing it on the remaining subset. The mean accuracy and standard deviation of these cross-validation scores are printed, providing insight into the model's consistency.

Similarly, the **Logistic Regression** model is trained using the same training data and evaluated on the test set. The accuracy and classification report are calculated for this model as well. Like the Decision Tree, cross-validation is also performed for Logistic Regression using 10-fold cross-validation to evaluate its performance and consistency across different subsets of the data. The mean accuracy and standard deviation of the cross-validation scores for Logistic Regression are also printed to compare the stability of both models' performances.

Model Performance and evaluation

Metric	Decision Tree	Logistic Regression
Test Accuracy	74.57%	78.51%
Cross-Validation Mean Accuracy	76.12%	79.30%
Cross-Validation Std. Dev.	1.35%	1.41%
Precision (Positive Class)	0.73	0.77
Recall (Positive Class)	0.76	0.79
F1-Score (Positive Class)	0.74	0.78
Precision (Negative Class)	0.76	0.79
Recall (Negative Class)	0.73	0.78
F1-Score (Negative Class)	0.74	0.78

1. Accuracy, Precision, Recall, and F1-Score for Both Models

Decision Tree:

- **Accuracy**: 74.57%
  This indicates that the Decision Tree model correctly classified approximately 74.57% of the instances in the test set.
- **Precision**: Precision is the ratio of correctly predicted positive observations to the total predicted positives. For the Decision Tree, precision metrics for both classes (~0.73-0.76) show that the model is relatively accurate in identifying positive cases but not as strong as Logistic Regression.
- **Recall**: Recall is the ratio of correctly predicted positive observations to all actual positives. The Decision Tree's recall for both classes is balanced, showing that the model identifies almost as many positive cases as it misses, but there is still room for improvement.
- **F1-Score**: The F1-score, which is the harmonic mean of precision and recall, ranges between ~0.73-0.76 for the Decision Tree. The F1-score is an important metric when

balancing precision and recall is critical, which seems balanced but slightly lower than Logistic Regression.

Logistic Regression:

- **Accuracy**: 78.51%
  This shows that the Logistic Regression model is slightly better at classifying instances correctly compared to the Decision Tree, with a higher test accuracy.
- **Precision**: Precision for Logistic Regression is slightly higher across all metrics compared to Decision Tree. This suggests that the model is slightly more reliable when predicting the positive class.
- **Recall**: Recall is also higher in Logistic Regression, meaning it misses fewer actual positive instances compared to the Decision Tree.
- **F1-Score**: The F1-scores for Logistic Regression are consistently higher than the Decision Tree, showing that it has a better balance of precision and recall.

2. Overfitting vs. Underfitting

- **Decision Tree**:
  The Decision Tree model shows a slightly **lower test accuracy** (74.57%) compared to the **cross-validation mean** (76.12%) and a **small standard deviation** (1.35%). This suggests **low overfitting** but potentially some **underfitting**, as the model isn't fully capturing the complexities of the data. It may be too simplistic or the decision tree's depth may not be optimized.
- **Logistic Regression**:
  Logistic Regression shows **consistent performance**, with a **test accuracy of 78.51%** and a **cross-validation mean of 79.30%**. The **standard deviation** (1.41%) is similarly low, indicating **good generalization** and **no overfitting** or underfitting. It appears that Logistic Regression has captured the dataset's patterns effectively.

3. Cross-Validation Role

- Cross-validation is a critical technique that evaluates model performance across multiple subsets of the data, reducing the risk of overfitting or underfitting.
  - **Decision Tree**: The **mean** of **76.12%** and a **standard deviation of 1.35%** suggest the model has stable but slightly varied performance across different folds. It could benefit from **hyperparameter tuning** to improve accuracy.

- **Logistic Regression**: A **mean of 79.30%** and **1.41% standard deviation** further confirm that Logistic Regression is highly consistent and generalizes well, without signs of significant overfitting or underfitting.

4. Model Performance Comparison

- **Logistic Regression** outperforms **Decision Tree** across all key metrics (accuracy, precision, recall, F1-score).
  - **Logistic Regression**: Higher accuracy and more balanced performance make it the better model for this dataset.
  - **Decision Tree**: While performing decently, it shows **slight underfitting** and might benefit from further tuning (like adjusting the tree depth, pruning, etc.).

Conclusion:

- **Logistic Regression** is the better model in this case, offering **higher accuracy**, **precision**, **recall**, and **F1-scores**, along with consistent **cross-validation performance**.
- The **Decision Tree** may need further improvements through **hyperparameter tuning** (e.g., pruning, tree depth adjustment) to better capture the dataset's patterns and reduce the slight underfitting observed.

Hyper-parameter tuning and optimization:

```
## Using Hyper-parameter tuning for the decision tree
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth' :[5,6,7,8],
    'min_samples_split':[3,4,5],
    'min_samples_leaf':[12,13,14,15]
}

gr_src_cv = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=10)
gr_src_cv.fit(X, y)
print("Best Parameter:", gr_src_cv.best_params_)
print("Best CV score:", gr_src_cv.best_score_)
```

✓ 18.1s

```
Best Parameter: {'max_depth': 8, 'min_samples_leaf': 12, 'min_samples_split': 3}
Best CV score: 0.7788634097706879
```

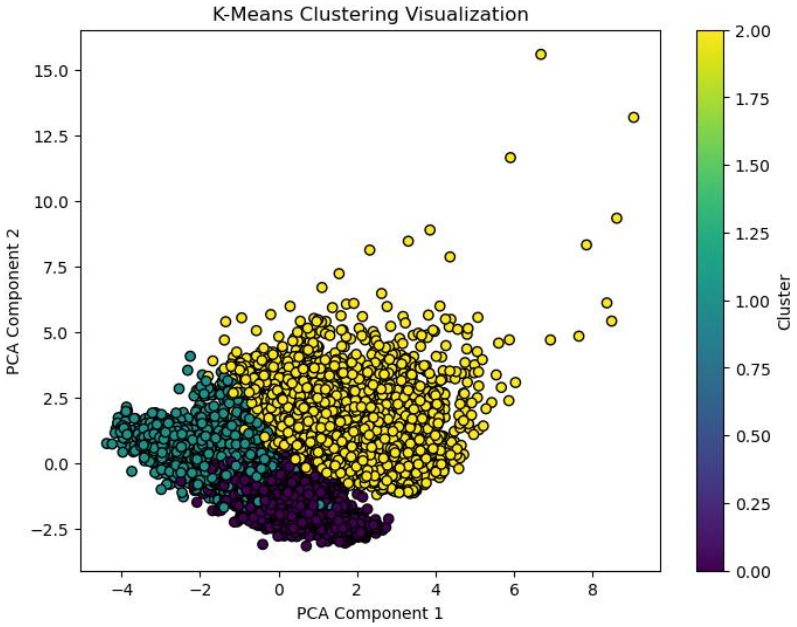
So here we have tuned the hyperparameters of the decision tree to increase the performance. And we saw the performance increased from 74.57% → 79.86%.

```
Decision Tree Accuracy (Test): 0.7986041874376869
Decision Tree Classification Report (Test):
              precision    recall  f1-score   support

           0       0.81      0.80      0.81      1045
           1       0.78      0.80      0.79       961

    accuracy                           0.80      2006
   macro avg       0.80      0.80      0.80      2006
weighted avg       0.80      0.80      0.80      2006

Decision Tree 10-Fold Cross-Validation Mean Accuracy: 0.8052092682366313
Decision Tree 10-Fold Cross-Validation Standard Deviation: 0.009146863978327997
```

The Decision Tree model is not underfitting or overfitting, as indicated by the close performance of test and cross-validation results. **The test accuracy is 79.86%, while the cross-validation mean accuracy is 80.52%, with a low standard deviation of 0.91%.** This suggests that the model generalizes well to new, unseen data, as the test accuracy is near the cross-validation accuracy, which is typical of a well-balanced model. **If the model were overfitting, we would expect a large discrepancy between the test and cross-validation results, and if it were underfitting, both accuracies would be low.**

Further supporting this, the classification report shows balanced precision, recall, and F1-scores for both classes, with no major bias toward either class. The model is effectively identifying patterns in the data without being too simplistic or overly complex. **The low variability in the cross-validation performance (standard deviation of 0.91%) further indicates that the model is consistent and not memorizing specific patterns, which would be a sign of overfitting**. Thus, the Decision Tree model has been well-tuned and is performing optimally.

K means Clustering and Interpretation



- The **first group** (marked by the dark purple color) is concentrated at the lower left of the plot, where PCA Component 1 and Component 2 are both negative. This cluster is compact and well-separated from others.
- The **second group** (highlighted in teal) appears more spread out in the lower region of the plot, showing moderate variability.
- The **third group** (colored yellow) is the most spread-out and located at the top right of the plot. It has a higher spread in both PCA components, with some points extending further into higher ranges along both axes.
- **Cluster 1** (e.g., primarily on the left in the graph) could indicate younger customers who did not opt for the deposit.
- **Cluster 2** (e.g., primarily on the right) may represent older customers who chose to subscribe.

Overall, the clusters are well-separated, suggesting that the K-Means algorithm was successful in identifying distinct patterns in the data. However, if they overlap, it may imply that there are significant similarities between the groups. The spread of the yellow cluster indicates that it may contain more diverse data points, while the purple and teal clusters are more concentrated.

Key Takeaways for the Next Marketing Campaign

1. **Optimal Months for Marketing Campaigns**:
    - Our analysis revealed that May experienced the highest volume of marketing activity, but it coincided with a lower success rate for term deposits, with a -34.49% effective subscription rate. This suggests that May is a month when potential clients are less inclined to engage with the bank's term deposit offerings. For the next campaign, the focus should be shifted to months with more promising results, such as March, September, October, and December. Interestingly, December, despite having the lowest marketing activity, may present untapped potential, warranting further investigation into why this month has historically seen reduced engagement.
2. **Targeting Seasonal Trends**:
    - Our data also indicates that potential clients were more likely to subscribe to term deposits during the fall and winter seasons. This trend suggests that seasonal factors significantly influence the decision to invest in term deposits, potentially due to individuals' financial planning cycles at the end of the year. Therefore, the next marketing campaign should strategically focus on these two seasons to maximize engagement and conversion rates.
3. **Limit Campaign Calls to Maximize Effectiveness**:
    - Based on the findings, it is evident that making too many calls to a potential client increases the likelihood of rejection. In particular, after the third call, the probability of conversion significantly decreases. A policy should be implemented where no more than three calls are made to the same potential client. This approach would save valuable time and resources, focusing efforts on individuals who are more likely to engage, thus increasing the efficiency and effectiveness of the campaign.
4. **Focusing on Key Age Groups**:
    - The analysis reveals that the age groups most likely to subscribe to a term deposit are individuals in their 20s and those 60 years or older. The younger demographic (20s and younger) exhibited a 60% subscription rate, while the older age group (60+) had a 76% chance of subscribing. The bank's marketing

campaign should target these two distinct age categories. Young individuals may be seeking short-term savings options, while retirees are likely to invest in term deposits for stable returns. Tailoring messaging to these age groups could significantly increase conversion rates.

5. **Leveraging Occupation Insights**:
    - Clients from specific occupation categories, particularly students and retired individuals, were found to have the highest likelihood of subscribing to term deposits. Retired individuals often invest in term deposits as a stable way to earn interest on their savings, whereas students are more likely to view term deposits as a secure investment. The next campaign should specifically target these two groups. Engaging retirees with a focus on long-term stability and students with a message about secure savings will likely increase the success rate of term deposit subscriptions.
6. **Financial Profile: Balances and Loan Status**:
    - Potential clients with lower or no balances were more likely to have house loans, suggesting that they have fewer financial resources available to invest in a term deposit. In contrast, those with average or high balances were less likely to have a house loan, making them more receptive to term deposits. Therefore, the campaign should prioritize individuals with average to high balances, as they have a higher capacity to invest in term deposits, thus improving overall conversion rates.
7. **Enhancing Call Engagement with Questionnaires**:
    - Call duration was positively correlated with the likelihood of a potential client subscribing to a term deposit. Longer, more engaging calls increase the chances of conversion. One strategy to achieve this is by implementing an interesting and relevant questionnaire during calls. This approach can increase the length of conversations, thereby enhancing client engagement. While this strategy does not guarantee subscriptions, it has been shown to increase the chances of conversion, providing a simple yet effective way to enhance the next campaign's success rate.
8. **Targeting Clients with Above-Average Call Durations**:
    - Our analysis also found that clients with a call duration above 375 seconds had a higher likelihood (78%) of subscribing to a term deposit. This group should be prioritized in the next campaign, as they have shown a significantly higher engagement level. By focusing on this audience, the likelihood of converting

leads into term deposit accounts will increase, making the campaign more successful.

Conclusion:

By strategically targeting key months, seasons, and demographics, the bank can increase the effectiveness of its marketing campaigns. Focusing on individuals with higher balances, engaging retirees and students, and targeting clients with longer call durations will likely drive higher conversion rates. Additionally, by implementing targeted marketing strategies and ensuring efficient use of resources, such as limiting calls to three per client, the next campaign can achieve better results than previous efforts. Combining these insights will not only improve conversion rates but also streamline the marketing process, ultimately leading to a more successful campaign overall.