Analyzing Bank Marketing Data: Identifying Patterns to Improve Future Campaign Strategies.

Prepared by : Sghiouar Imane
BEN SBEH Chaimae
Chakkour Karima
Under the supervision of : Pr. Belcaid Anass

January 2025

Contents

1 Problem Description :

1.1 Introduction :

In today's highly competitive banking and finance industry, customer acquisition and retention are essential for sustained growth. As financial institutions aim to design effective marketing strategies, understanding customer behavior, preferences, and reactivity becomes a critical asset. This study seeks to enhance marketing efforts by identifying and addressing inefficiencies in promoting term deposit subscriptions, a key product for many banks.

1.2 Context and Significance :

1.2.1 Domain :

Our project is based in the domain of banking marketing, specifically focusing on customer relationship management (CRM)[1]. It revolves around promoting products like term deposits. These deposits are important for banks as they provide stable funding and help build long-term relationships with customers. However, marketing campaigns for these products can be expensive and often don't have a high success rate. This highlights the need for more focused and efficient marketing strategies.

1.2.2 Problem Statement :

The data aims to solve the problem of predicting whether a client will subscribe to a term deposit using demographic, financial, and historical interaction data. Banks often spend a lot on large marketing campaigns, but they usually target clients who have a low chance of converting, leading to wasted resources and lower profits. This analysis focuses on using predictive modeling to help banks target the right clients, reduce campaign costs, and improve efficiency.

1.2.3 Importance of Addressing the Problem :

- *Marketing Efficiency:* Identifying clients with a high likelihood of subscribing allows banks to design targeted campaigns, reducing wasted resources and time.

- *Cost Reduction:* Focusing on interested customer groups helps optimize marketing budgets and avoid wasting money on unlikely subscribers.

- *Actionable Insights:* Predictive analytics can reveal important patterns in customer behavior, helping make better decisions about outreach, products, and engagement strategies.

[1]Customer Relationship Management (CRM) is a strategy used by businesses, including banks, to manage and improve interactions with customers, aiming to enhance satisfaction, loyalty, and long-term relationships.

1.3 Objectives :

Following the problem description and its significance, the primary goal of this project is to develop a robust predictive model that classifies whether a client will subscribe to a term deposit (yes) or not (no) based on the available data.
To achieve this, the following specific objectives will guide the work:

Feature Identification : Identify which factors—such as age, job type, previous contact outcomes, or campaign duration—most strongly influence client decisions.

Marketing Strategy Refinement : Leverage the insights from the predictive model to design data-driven marketing strategies aimed at improving client engagement and increasing the success rates of future campaigns.

Performance Evaluation and Recommendations : Assess the model's performance using a range of evaluation metrics, and provide actionable recommendations to improve model accuracy, including strategies for handling data limitations or imbalances.

2 Data Understanding and Description :

2.1 Data Collection :

2.1.1 Sources of the data :

The dataset used in this project, ***bank.csv***, was obtained from Kaggle, where it is linked to the study by Moro et al. (2014). This study, titled **"A Data-Driven Approach to Predict the Success of Bank Telemarketing"**, was published in DecisionSupportSystems[2] (Elsevier[3], Volume 62, Pages 22-31, June 2014). The dataset contains detailed information on the bank's telemarketing campaigns, including customer demographics, socio-economic attributes, and data from previous campaigns.

For more information, the study can be accessed directly on ScienceDirect.

2.1.2 Accessing and Preparing the Dataset :

The dataset was accessed through Kaggle, where it is linked to the study by Moro et al. (2014). After downloading the dataset in CSV format, it was imported into Python using the pandas library and the `pandas.read_csv()` function.
The process of loading the dataset into Python is illustrated in the following code :

```
import pandas as pd

# Load the dataset :
data = pd.read_csv('bank.csv')

# Display the first Five rows of the data :
data.head()
```

After executing the code, the dataset was successfully loaded, and the following table presents the first five rows of the data for initial inspection :

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

[2]Decision Support Systems is an academic journal that publishes research on systems designed to assist in decision-making through data analysis and modeling.
[3]Elsevier is a global academic publisher specializing in scientific, technical, and medical research.

2.2 Data Description :

2.2.1 Overview of the dataset structure :

The dataset, consisting of 11,162 rows and 17 columns, is stored in a compact CSV file with a size of 0.88 MB. It contains a mix of numerical and categorical variables, which offer a strong basis for meaningful analysis.

To programmatically confirm the number of rows and columns, the following code snippet can be used :

```
# Display the shape of the dataset (rows,columns) :

data.shape
```

In this case, the `raw_data.shape` method returns a tuple where the first value represents the number of rows and the second value indicates the number of columns in the dataset.

2.2.2 Key features/variables and their types :

This section will present a detailed examination of the variables, highlighting their characteristics , type and the value they contribute to the analysis.

Numerical Variables :

- *Ratio-scaled :*
  - *age :* Represents the client's age in years.
  - *balance :* Represent the client's account balance expressed in monetary units.
  - *duration :* Represent the duration (measured in seconds) of the last contact made with a client .
  - *campaign :* Represent the number of contacts made with a client during the current marketing campaign. This refers to how many times the bank attempted to reach out to a client (e.g., via phone calls, emails, etc.) as part of the ongoing campaign.
  - *previous :* Represents the number of contacts made with a client before the current campaign.
- *Interval-scaled :*

- *day :* Represents the day of the month (from 1 to 31) when a client was last contacted from previous campaign.

- *pdays :* Represent the number of days since the customer was last contacted from previous campaigns, where -1 indicates no previous contact.

Categorical Variables :

- *job :* (Nominal) Represent the type of job held by the client, such as administrative, technician, or management.

- *marital :* (Nominal) Represent the marital status of the customer, which could be single, married, or divorced.

- *education :* (Ordinal) Represent the customer's highest educational level.

- *contact :* (Nominal) Represent the communication channel used for client contact, such as cellular, telephone, or unknown.

- *month :* (Nominal) Represents the month of the last contact with a client.

- *poutcome :* (Nominal) Represent the outcome of the previous marketing campaign (e.g., success, failure, unknown).

Binary Variables:

- *default :* Indicates whether the customer has credit card debt (1 = Yes, 0 = No).

- *housing :* Reflects whether the customer has a housing loan (1 = Yes, 0 = No).

- *loan :* Shows if the customer has a personal loan (1 = Yes, 0 = No).

- *deposit :* The **target variable**, indicates whether the customer subscribed to a term deposit (1 = Yes, 0 = No).

To gain further insights into the dataset, the `raw_data.info()` method can be executed. This method provides a detailed summary of the dataset, including the total number of entries, the names and data types of each column, and the count of non-null values for each column.

2.2.3 Description of the target variable :

The dataset includes a target variable **Deposit**, which is binary in nature and represents the outcome of the marketing campaign. It indicates whether a customer subscribed to a term deposit, with two possible values: 'yes' and 'no'.

We can determine the distribution of the target variable by executing the code :

```
# Display value counts for the 'Deposit' column:

data['Deposit'].value_counts()
```

The method `value_counts()` counts the occurrences of each category in the Deposit column, where it shows that out of 11,162 entries, 5,289 customers successfully subscribed (yes), while 5,873 customers did not (no).

This variable is critical to the analysis, serving as the key outcome to evaluate and predict the effectiveness of the marketing strategies.

3 Exploratory Data Analysis (EDA) :

3.1 Initial Data Exploration :

3.1.1 Statistics summary for numerical variables :

In this section , we will use the `describe()` method from Python's Pandas library to get a quick summary of the data. It provides a statistical overview, including metrics like the **mean**, **standard deviation (std)** , **minimum**, and **maximum** values for each numeric column. This makes it easier to understand the dataset and spot any weird patterns or trends.

```
# Get the statistics summary of numerical variables :
data.describe()

# Save the summary as CSV file :
data.describe().to_csv('summary_statistics.csv')
```

	age	balance	day	duration	campaign	pdays	previous
count	11162.0	11162.0	11162.0	11162.0	11162.0	11162.0	11162.0
mean	41.231947679627304	1528.5385235620856	15.658036194230425	371.99381831213043	2.508421429851281	51.33040673714388	0.8325568894463358
std	11.913369192215445	3225.4133259461923	8.420739541006434	347.1283857163069	2.722077181661534	108.75828197196235	2.2920072186703067
min	18.0	-6847.0	1.0	2.0	1.0	-1.0	0.0
25%	32.0	122.0	8.0	138.0	1.0	-1.0	0.0
50%	39.0	550.0	15.0	255.0	2.0	-1.0	0.0
75%	49.0	1708.0	22.0	496.0	3.0	20.75	1.0
max	95.0	81204.0	31.0	3881.0	63.0	854.0	58.0

3.1.2 Interpretation of the statistical summary :

In this section, we will interpret the statistical summary of each variable in the dataset. By examining key metrics such as the mean, median, and range, we aim to understand the distribution of the data and identify any notable patterns.

The *age* variable :

– **Count:** 11,162 entries, no missing values for age.

– **Mean:** The average customer age is approximately 41.23 years, indicating the data leans towards middle-aged individuals.

– **Std (Standard Deviation):** Approximately 11.91 years, suggesting moderate variation in ages.

– **Min:** The youngest customer age is 18 years.

– **Max:** The oldest customer age is 95 years.

– **25%, 50%, 75%:** Most customers are aged between 32 years (25th percentile) and 49 years (75th percentile), with a median of 39 years.

The *balance* variable :

Represent the Customer's account balance.

- **Count:** 11,162 entries, no missing values for balance.
- **Mean:** The average balance is 1528.54 units.
- **Std (Standard Deviation):** Approximately 3225.41, indicating high variability in balances.
- **Min:** The smallest balance is -6847, representing debt or overdraft cases.
- **Max:** The largest balance is 81,204, an extreme outlier as it far exceeds the 75th percentile (1708).
- **25%, 50%, 75%:** Most of the individuals have balances ranging from 122 to 1708, with a median of 550.

The *day* variable :

Represents the day of the month when the last contact with a customer was made.

- **Count:** 11,162 entries, no missing values for day.
- **Mean:** The average day of interaction is aproximately 15.66, possibly corresponding to days of the month.
- **Std (Standard Deviation):** Approximately 8.42 days, indicating most interactions are clustered within certain days.
- **Min:** The earliest day recorded is the 1st.
- **Max:** The latest day recorded is the 31st.
- **25%, 50%, 75%:** Most interactions occurred between the 8th and the 22nd (75th percentile), with a median of the 15th.

The *duration* variable :

Represent the duration of the last contact with the customer, measured in seconds.

- **Count:** 11,162 entries, no missing values for duration.
- **Mean:** The average call duration is approximately 372 seconds (about 6.2 minutes).
- **Std (Standard Deviation):** Approximately 347 seconds, indicating significant variability in call lengths.
- **Min:** The shortest call lasted for 2 seconds.
- **Max:** The longest call lasted for 3881 seconds (about 65 minutes), which is unusually long.
- **25%, 50%, 75%:** Half of the calls lasted between 138 and 496 seconds, with a median of 255 seconds (about 4.25 minutes).

The *campaign* variable :

Represent the number of contacts made with a customer during the current marketing campaign.

– **Count:** 11,162 entries, no missing values for campaign.
– **Mean:** The average number of contacts per campaign is approximately 2.51.
– **Std (Standard Deviation):** Approximately 2.72, indicating variability in the number of contacts per individual.
– **Min:** The minimum value is 1 (all individuals were contacted at least once).
– **Max:** The maximum value is 63 contacts for a single campaign.
– **25%, 50%, 75%:** Half of the individuals were contacted between 1 and 3 times, with a median of 2.

The *pdays* variable :

Represent The number of days since the customer was last contacted.

– **Count:** 11,162 entries, no missing values for campaign.
– **Mean:** The average pdays value is approximately 51.33 days.
– **Std (Standard Deviation):** Approximately 108.76, showing high variability in the number of days since the last contact.
– **Min:** The minimum value is -1, indicate individuals who were never contacted before.
– **Max:** The maximum value is 854 days, possibly an outlier or long-term customers.
– **25%, 50%, 75%:** Half of the individuals fall within -1 (never contacted) and 20.75 days, with many having no previous contacts (median = -1) .

The *previous* variable :

Represents the number of contacts made with a client before the current campaign.

– **Count:** 11,162 entries, no missing values for campaign.
– **Mean:** The average number of previous contacts is approximately 0.83.
– **Std (Standard Deviation):** Approximately 2.29, indicating variability, with some individuals contacted far more than others.
– **Min:** The minimum value is 0, meaning some individuals had no prior contacts.
– **Max:** The maximum value is 58, an unusually high number, likely an outlier.
– **25%, 50%, 75%:** Half of the individuals were contacted no more than 1 time, with a median of 0.

3.1.3 Visualization of Numerical variables Distributions:

In this section, we will explore the distribution of numerical variables in the dataset through visualizations (histogram). By examining how these features are distributed, we can get a better idea of their overall pattern, where most of the values tend to be, and if there are any unusual data points (outliers). This will help us make decisions for the next steps in the analysis and modeling process.

To avoid repetition, we will select the numeric columns from the dataset and use a loop to generate histograms for each of them, without having to write individual code for each numerical column.

We will save each diagram as a PNG file using the `plt.savefig` method, making it easy to organize and integrate into the report.

```
# Select Numeric columns :
numeric_cols = data.select_dtypes(include=['int64', 'float64']).columns

# Plot histograms for numeric columns :
for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    plt.hist(data[col], bins=20, color='blue', alpha=0.7, edgecolor='black')
    plt.title(f'Distribution_of_{col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Save each figure as a PNG file :
    plt.savefig(f'{col}_distribution.png', format='png')

    # Show the plot
    plt.show()
```

Distribution of *age* :

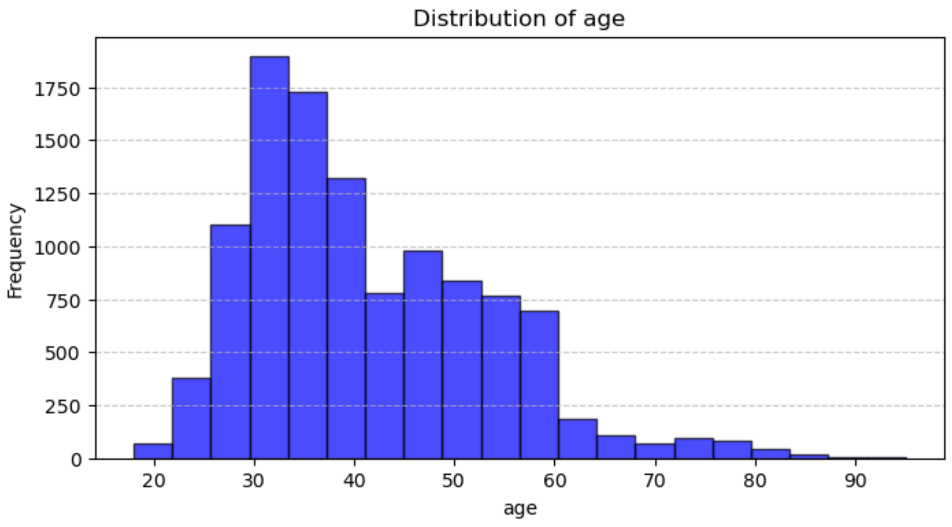

Figure 1: Distribution of Customers Age.

The histogram represents the distribution of the age variable, which is right-skewed, meaning most people in the dataset are younger or middle-aged. The most common age group is between 30 and 35, with fewer individuals as the age increases. The ages range from 18 to around 90, but the majority fall between 30 and 50. There are only a small number of older individuals, especially above 70.

The age distribution suggests that the previous marketing campaign primarily reached middle-aged individuals.

Distribution of ***balance*** :

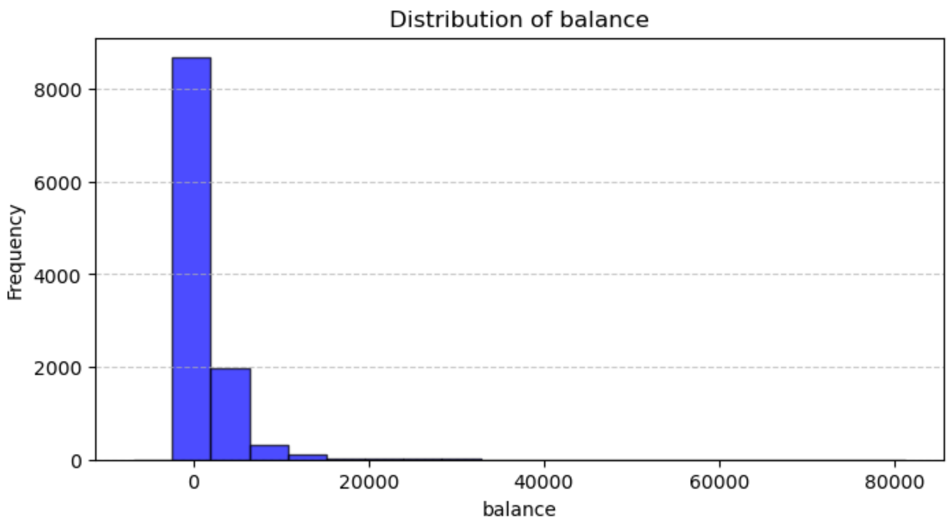

Figure 2: Distribution of Account Balances.

The histogram represents the distribution of the balance variable. It shows that most people have balances close to zero, with the number of accounts decreasing as the balances get higher.

Distribution of ***day*** :

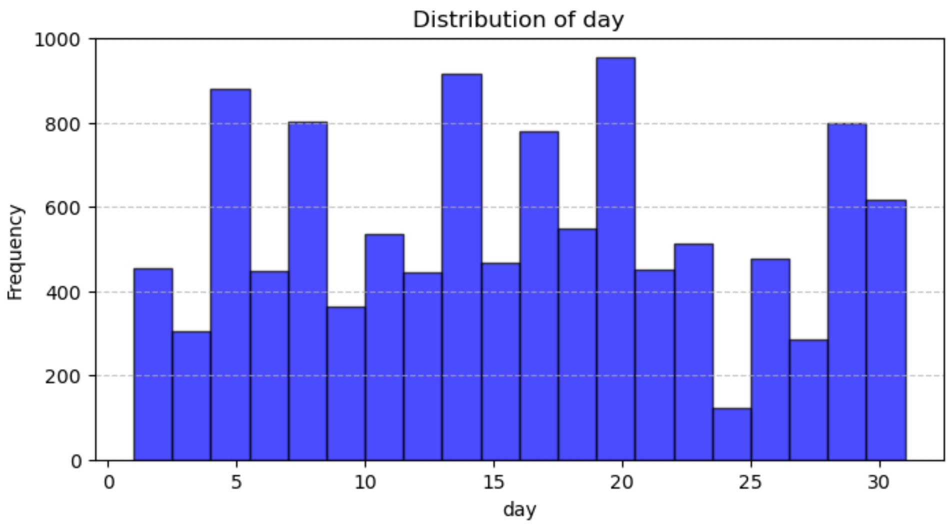

Figure 3: Daily Distribution of Client Contacts During the Marketing Campaign.

The histogram shows how often clients were contacted on each day of the month (from 1 to 31) during a marketing campaign. Some days, like around the 5th, 14th, and 20th, had more activity, while others, like the 10th and 25th, had fewer. This means the campaign had busy and quiet days.

Distribution of ***duration*** :

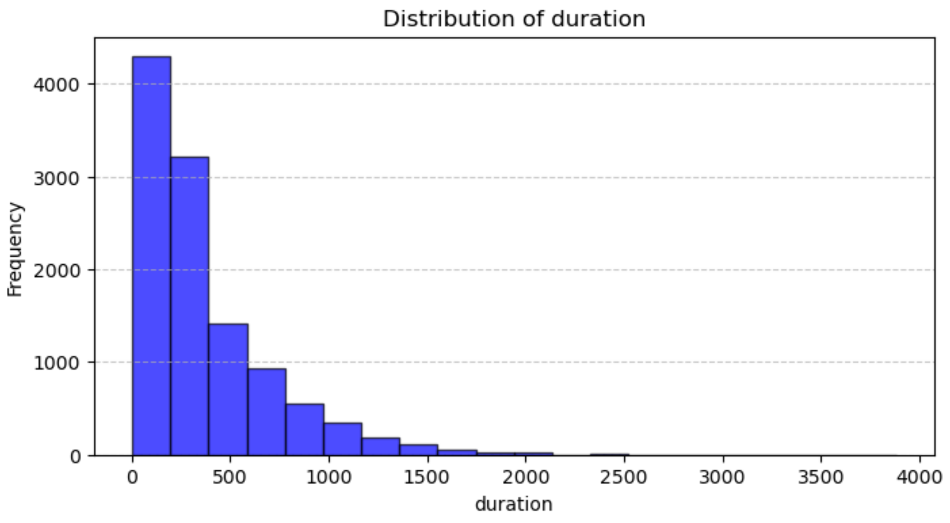

Figure 4: Duration in second of the last contact made with client

The histogram shows the distribution of the duration feature, which represents the length of the last contact with a customer in seconds. The data is heavily skewed to the right, with most calls lasting under 500 seconds (equivalent to 8.33 minutes), and only a few exceeding this range. This indicates that the majority of customer interactions during the campaign were short, while longer calls were rare.

Distribution of *campaign* :

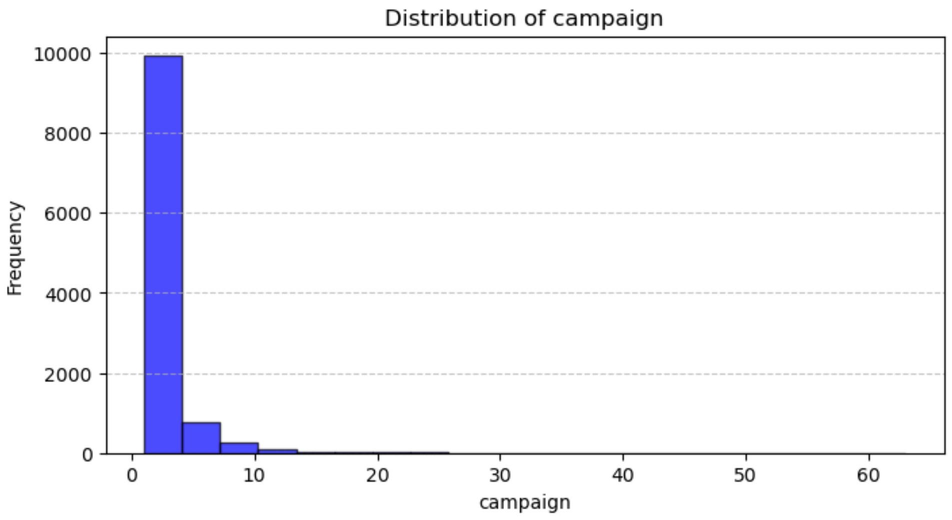

Figure 5: Number of Contacts made with a customer during a Campaign

The histogram shows the distribution of the campaign feature, which represents the number of times a client was contacted during the current marketing campaign. The data is highly skewed, with most clients being contacted only once or twice, while very few were contacted more than 10 times.

The distribution indicates that the campaign primarily focused on minimal client interactions, possibly to avoid overwhelming customers.

Distribution of *pdays*:

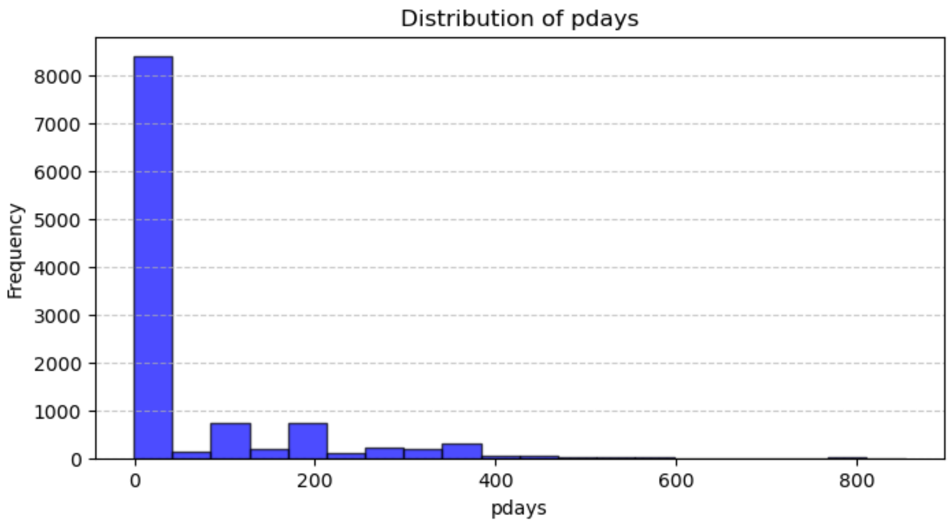

Figure 6: Distribution of Days Since Last Contact in Marketing Campaigns

The histogram illustrates the distribution of the pdays feature, which represents the number of days since the customer was last contacted in previous campaigns. The highest frequency of occurrences is at 0 days, which suggest that most clients have recently been contacted. As the number of days since the last contact increases, the frequency of occurrences decreases, with a noticeable drop after around 200 days.

Most customers have been contacted recently, while fewer have been contacted after a longer time. This suggests that the campaigns focus more on reaching customers quickly and less on re-engaging those who haven't been contacted in a while.

Distribution of ***previous***:

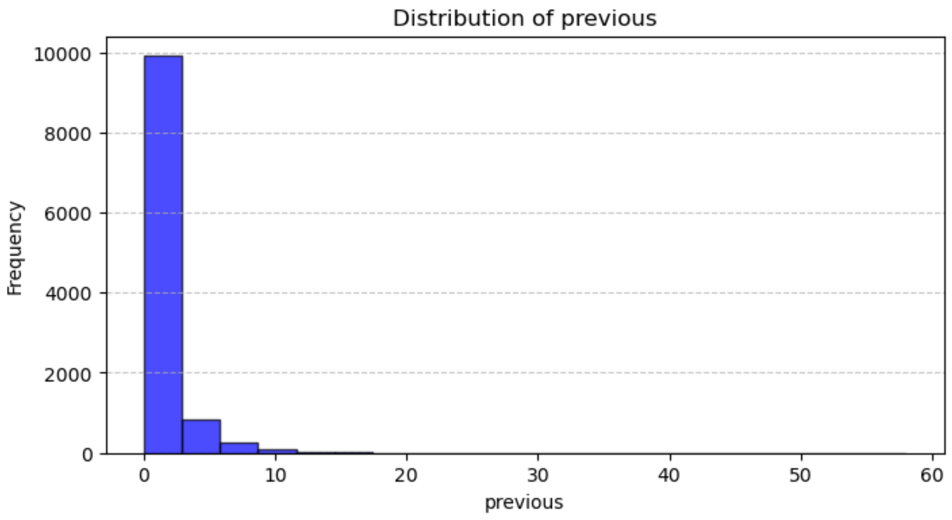

Figure 7: Number of Contacts made with a client before the current campaign.

The histogram illustrates the distribution of the pdays feature, which represent the number of contacts made with a client before the current campaign. Most clients have not been contacted previously. The frequency drops quickly as the number of previous contacts goes up, with most clients having 1 to 5 prior contacts. After 10 contacts, the frequency decreases even more. This shows that many clients are being contacted for the first time, while fewer have been contacted several times before.

This suggests that the campaign's main goal is to reach new clients, with less focus on re-engaging those who have been contacted in past campaigns.

3.1.4 Analysis of Categorical variables :

In this section , we will explore the categorical variables in our dataset in order to understand their characteristics and the role they play in influencing the outcomes of our analysis.

To achieve this, we will begin by using the `describe` method to summarize the frequency and unique values for each categorical variable.

To avoid repetition , we will select the categorical columns from the dataset and use a loop to generate histograms for each of them, without having to write individual code for each column.

The resulting summary statistics are then converted into a DataFrame using the to_frame() method and saved as a CSV file (to make it easier to include and analyze in the report).

```
# Select Categorical columns :
categorical_columns = data.select_dtypes(include=['object']).columns

# Loop through categorical columns
for col in categorical_columns:

    # Get the description of the column :
    description_df = data[col].describe()

    # Convert the description to a DataFrame
    description_df = description_df.to_frame()

    # Save the description as a CSV file
    description_df.to_csv(f'{col}_description.csv', index=True)
```

For the visualization part, we will generate bar charts to display the distribution of categorical and binary variables. These charts will be customized for clarity and saved as PNG files, ensuring they are well-organized and easy to integrate into the report.

```
# Plot bar charts for each columns :

for col in categorical_columns:
    plt.figure(figsize=(8, 4))
    data[col].value_counts().plot(kind='bar', color='orange', alpha=0.7,
        ↪ edgecolor='black')
    plt.title(f'Distribution␣of␣{col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Save the figure as a PNG file :
    plt.savefig(f'{col}_distribution.png', format='png')

    # Show the plot
    plt.show()
```

Distribution of ***job*** :

The **job** feature represents the type of occupation held by the client, such as administrative, technician, or management. It provides valuable insights into how a client's professional background may influence their interaction with the marketing campaign.

To analyze the distribution of this variable, we will use the describe method provide the most common professions among clients, the total number of unique job types, and the frequency of each occupation.

	job
count	11162
unique	12
top	management
freq	2566

Table 1: Summary statistics for the 'job' column

Next, we can visualize the distribution of the job feature through a bar chart.

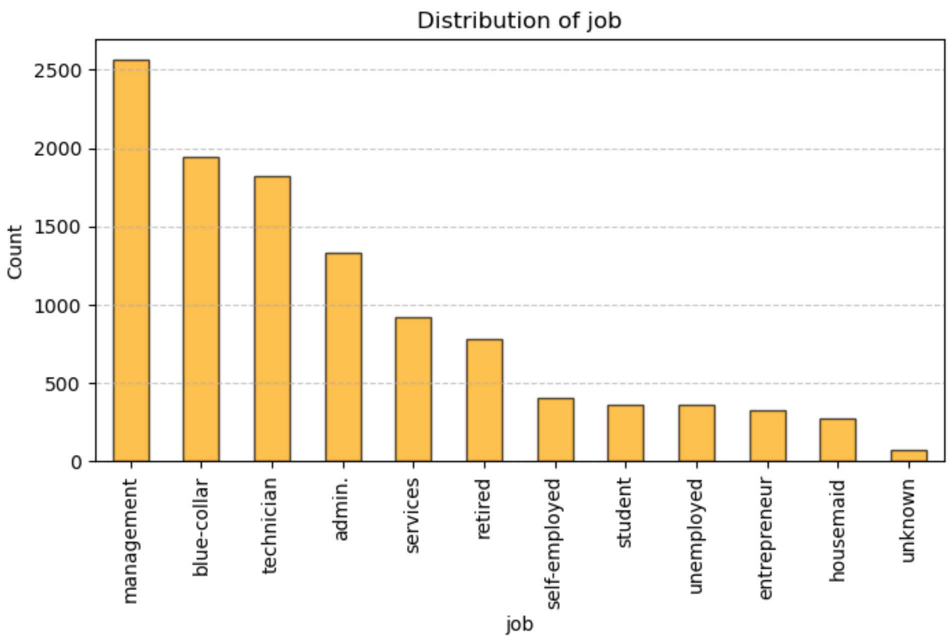

Figure 8: Distribution of jobs held by clients.

The summary of the 'job' column shows a total of 11,162 entries, with 12 distinct job types. This means that there are 12 different categories of occupations represented in the dataset. The most common job is "management," which appears 2,566 times.

Distribution of *marital* :

The **marital** feature represents the marital status of the customer, which could be single, married, or divorced. This information provides valuable context for understanding customer behavior and preferences, as marital status can influence decision-making and financial priorities.

To analyze this, we will use the `describe` method to identify the most common marital status and its frequency .

	marital
count	11162
unique	3
top	married
freq	6351

Table 2: Summary statistics for the 'marital' column

Next, we can visualize the distribution of the marital feature through a bar chart.

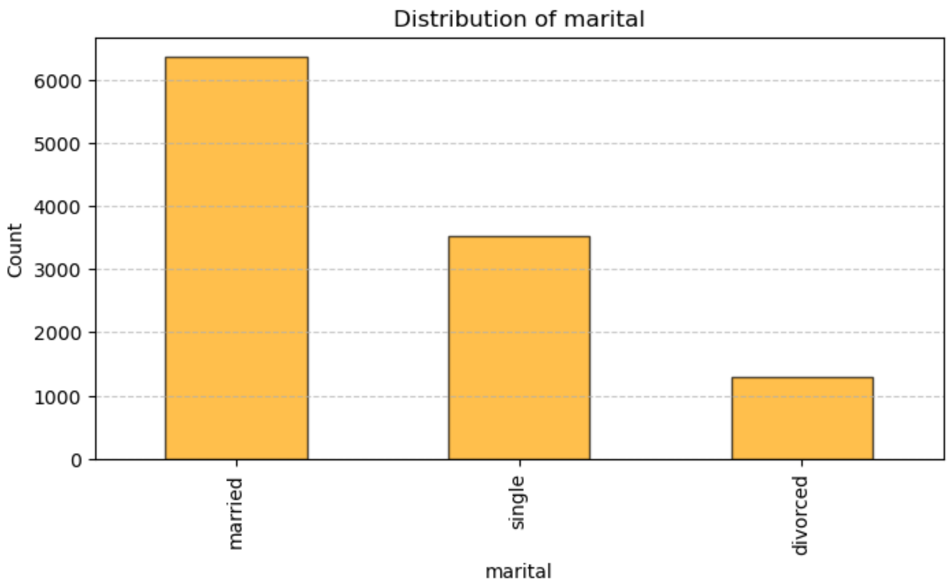

Figure 9: Distribution of clients marital status .

The marital status distribution reveals that the majority of individuals are married, with 6,351 out of 11,162 records representing married clients.

This suggests that marketing efforts focus more on targeting married individuals, promoting products related to long-term financial stability, such as term deposits, insurance, or savings plans.

Distribution of *education* :

The **education** variable represent the customer's highest educational level.

Education level can impact how customers respond to financial products. People with higher education may prefer complex investment options, while those with lower education might be more interested in simpler, accessible financial products.

To analyze this, we will use the `describe` method to identify the most frequent education level and its frequency.

	education
count	11162
unique	4
top	secondary
freq	5476

Table 3: Summary statistics for the 'education' column

Next, we can visualize the distribution of the education feature through a bar chart.

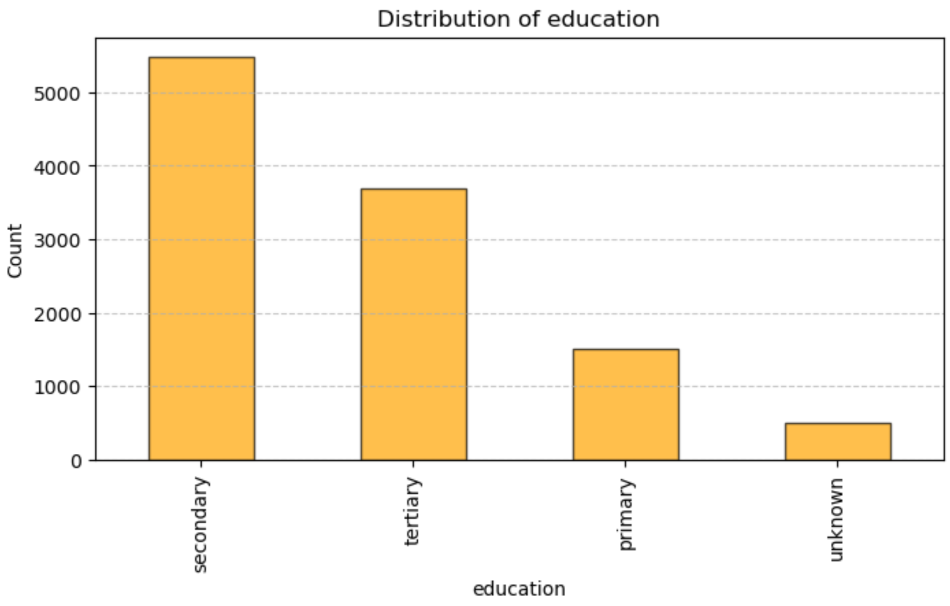

Figure 10: Distribution of clients education level.

The education level distribution reveals that most individuals have completed secondary education, with 5,476 out of 11,162 records falling into this category.

This suggests that the marketing campaign should have focused more on offering clear, easy-to-understand financial products, such as savings accounts or personal loans, to individuals with secondary education.

Distribution of *contact* :

The contact variable represents the communication channel used for client contact, such as cellular, telephone, or unknown.

To analyze this, we will use the **describe** method to identify the most frequent communication channel and its frequency.

	contact
count	11162
unique	3
top	cellular
freq	8042

Table 4: Summary statistics for the 'contact' column

Next, we can visualize the distribution of the contact feature through a bar chart.

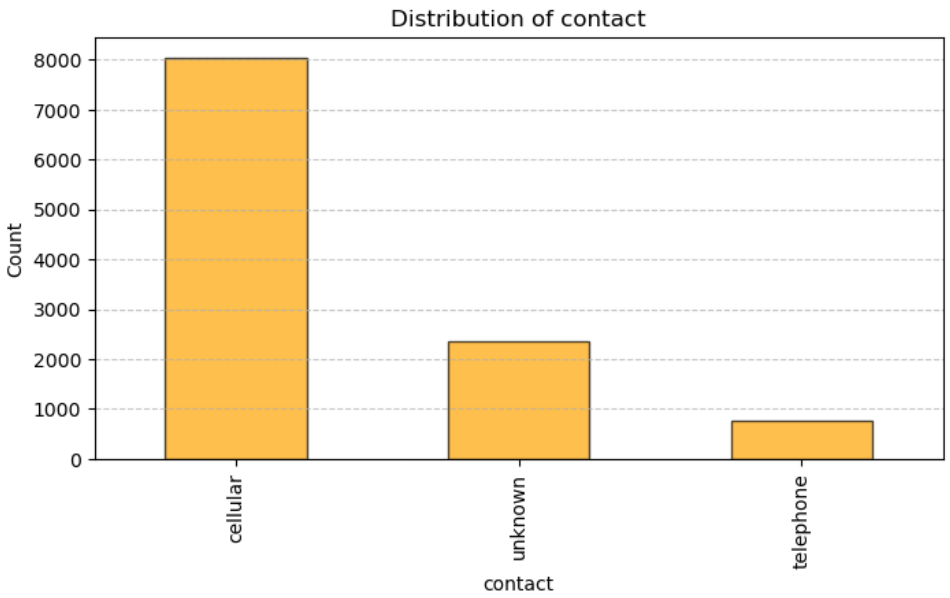

Figure 11: Distribution of contact.

The contact distribution reveals that most individuals prefer being contacted by cellular, with 8,042 out of 11,162 records indicating this preference. This suggests that the marketing campaign should have focused more on reaching customers through cellular contact, as this method is the most popular among the dataset.

Distribution of *month* :

The month variable represents the month of the last contact with a client.

To analyze this, we will use the **describe** method to identify the most frequent month and its frequency.

	month
count	11162
unique	12
top	may
freq	2824

Table 5: Summary statistics for the 'month' column

Next, we can visualize the distribution of the month feature through a bar chart.

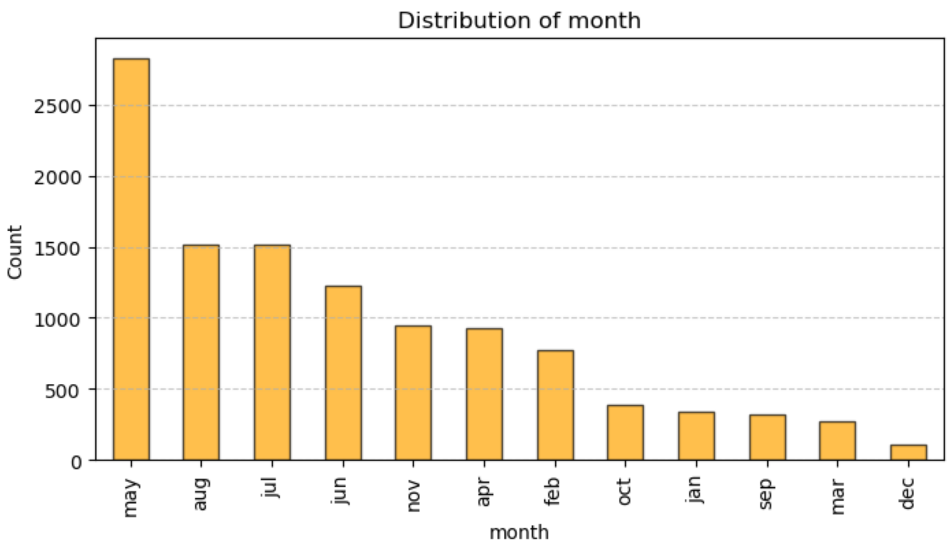

Figure 12: Distribution of month of the last contact with a client.

The distribution of the month reveals that May is the most common month for the last contact with a client, with 2,824 out of 11,162 contacts occurring in this month. This suggests that the marketing campaign had the highest level of customer engagement in May, which could indicate a peak period for outreach or promotional activities.

Distribution of ***poutcome*** :

The poutcoume represents the outcome of the previous marketing campaigns .

To analyze this, we will use the **describe** method to identify the most frequent outcomes of the marketing campaigns.

	poutcome
count	11162
unique	4
top	unknown
freq	8326

Table 6: Summary statistics for the 'poutcome' column

Next, we can visualize the distribution of the poutcome feature through a bar chart.

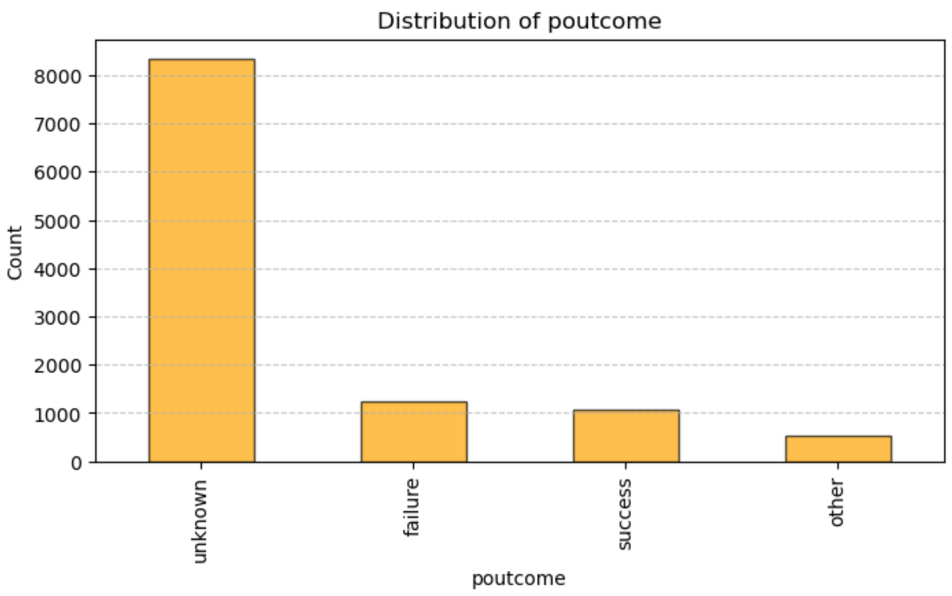

Figure 13: Distribution of the outcomes of the previous marketing campaigns.

The distribution shows that the most common outcome of the marketing campaigns is "unknown" with 8,326 out of 11,162 records categorized as unknown. This suggests that a significant portion of the campaign results were either not recorded or not determined, highlighting a potential need to improve the tracking and documentation of campaign outcomes.

3.1.5 Analysis of binary variables :

In this section , we will explore the binary variables in our dataset in order to understand their characteristics and the role they play in influencing the outcomes of our analysis.

To analyze this, we will use the describe method to identify the most frequent value (either "yes" or "no") and its frequency.

Distribution of *default* :

The **default** variable indicates whether the customer has credit card debt or not.

	default
count	11162
unique	2
top	no
freq	10994

Table 7: Summary statistics for the 'default' column

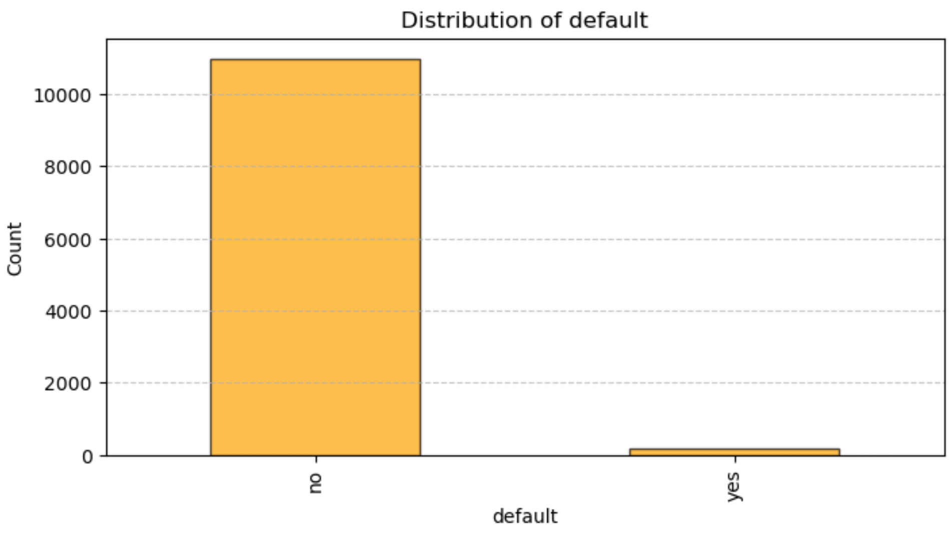

Figure 14: Distribution of default feature.

The distribution of the default variable reveals that the majority of customers, 10,994 out of 11,162, do not have credit card debt, as indicated by the "no" value. Only a small portion of the customers, 168, have credit card debt, as indicated by the "yes" value.

This suggests that marketing efforts could be more effectively directed toward customers without credit card debt, perhaps offering products that promote savings or debt management.

Distribution of ***housing*** :

The **housing** variable reflects whether the customer has a housing loan.

	housing
count	11162
unique	2
top	no
freq	5881

Table 8: Summary statistics for the 'housing' column

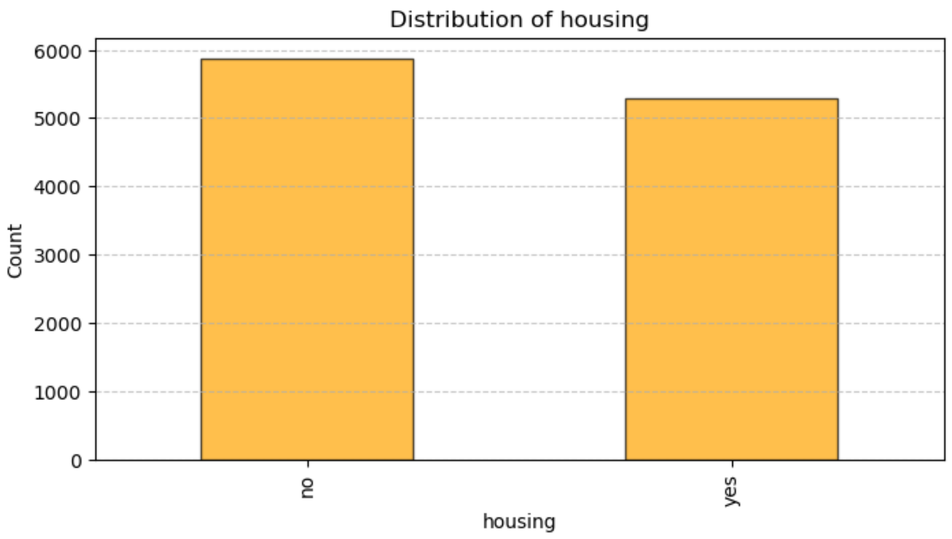

Figure 15: Distribution of the housing feature.

The distribution shows that out of 11,162 records the most common value is "no," which appears 5,881 times, indicating that the majority of customers do not have a housing loan. This suggests that marketing efforts could be focused on promoting housing-related products, such as mortgage loans or home equity offerings, to the segment without housing loans.

Distribution of ***loan*** :

The **loan** variable indicates if the customer has a personal loan or not.

	loan
count	11162
unique	2
top	no
freq	9702

Table 9: Summary statistics for the 'loan' column

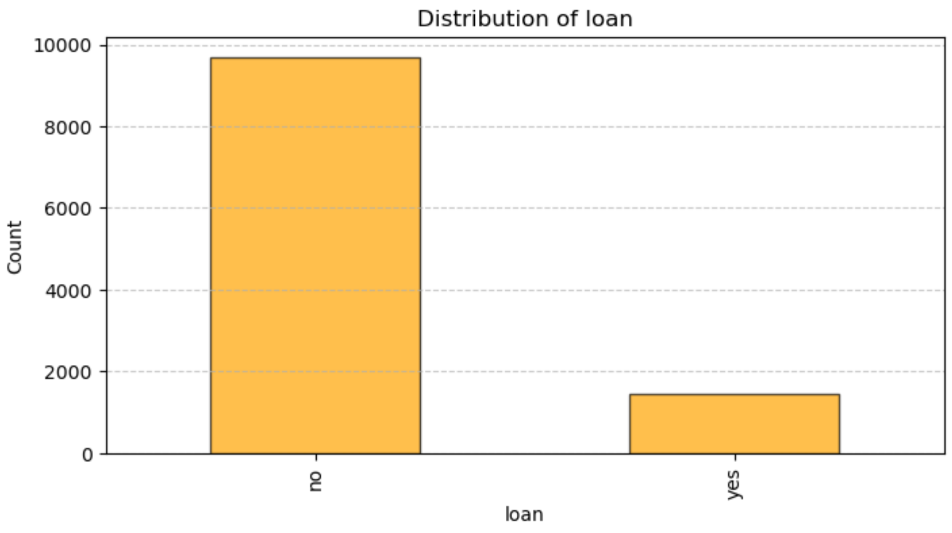

Figure 16: Distribution of the loan feature.

The distribution shows that out of 11,162 records the most common value is "no," which appears 9,702 times, indicating that most customers do not have a personal loan. This suggests that marketing efforts could focus on promoting personal loans to the larger group of customers who currently do not have one.

Distribution of *deposit* the target variable :

The target variable , indicates whether the customer subscribed to a term deposit or not.

For visualizing the target variable, we will use a circular chart (pie chart) to display the distribution of each class, helping to clearly show the proportion of each category.

```
# Target variable distribution
plt.figure(figsize=(6, 4))
data['deposit'].value_counts().plot(kind='pie', autopct='%1.1f%%', colors=['
    ↪ skyblue', 'lightgreen'], startangle=90)
plt.title('Distribution␣of␣Target␣Variable␣(deposit)')
plt.ylabel('')

# Save the chart as a PNG file
plt.savefig('deposit_distribution.png', format='png')

# Display the chart
plt.show()
```

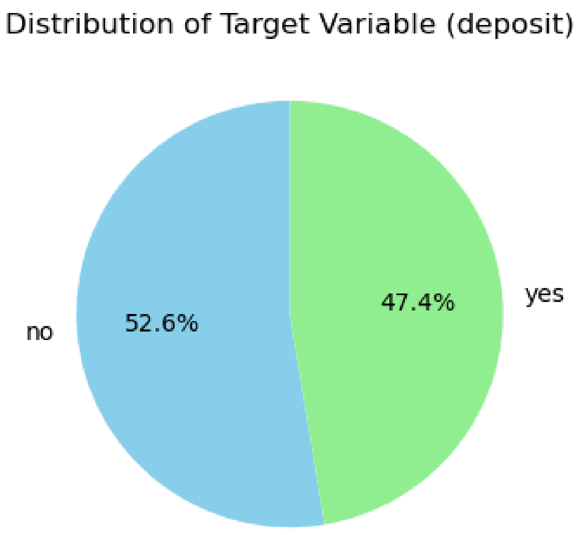

Figure 17: Distribution of the target feature.

The distribution of the target variable shows that out of 11,162 records, 5,873 (52.7%) customers did not subscribe to the marketing offer (”no”), while 5,289 (47.3%) customers did subscribe (”yes”).

This indicates that slightly less than half of the customers responded positively to the offer, and there is potential to increase engagement in future campaigns by targeting the group that did not subscribe.

3.2 Correlation Analysis :

Having examined the individual distributions of each feature, the next step is to explore potential relationships between them. Correlation analysis allows us to identify how features are related to one another, highlighting any strong linear dependencies or associations. By analyzing the correlation between variables, we can uncover patterns that might inform feature selection and guide further modeling decisions.

3.2.1 Correlation matrix :

We will begin by selecting the numerical features in the dataset and then compute the correlation matrix to assess the relationships between them. This will allow us to identify any strong linear dependencies. To visualize these relationships more clearly, we will generate a heatmap of the correlation matrix, which will highlight the most significant correlations between the features.

```python
# Select only numeric columns for correlation
data_num = data.select_dtypes(include=['number'])
data_num.head()


# Compute the correlation matrix
correlation_matrix = data_num.corr()

# Save the correlation matrix as a CSV file
correlation_matrix.to_csv('correlation_matrix.csv', index=True)

# Display the correlation matrix
print("Correlation␣Matrix:")
print(correlation_matrix)
```

	age	balance	day	duration	campaign	pdays	previous
age	1.0	0.11229988859873198	-0.0007624209205460853	0.0001892280737160303	-0.005277936156040662	0.0027738343117695813	0.02016856121844694
balance	0.11229988859873198	1.0	0.010467439549070373	0.022436131268962916	-0.013893822542985709	0.01741114863267417	0.030805246871564575
day	-0.0007624209205460853	0.010467439549070373	1.0	-0.018511399167089253	0.13700683429735644	-0.07723161298140314	-0.05898068354621472
duration	0.0001892280737160303	0.022436131268962916	-0.018511399167089253	1.0	-0.041557458759623876	-0.02739155324503983	-0.026716171271670003
campaign	-0.005277936156040662	-0.013893822542985709	0.13700683429735644	-0.041557458759623876	1.0	-0.10272604750934175	-0.04969949797455876
pdays	0.0027738343117695813	0.01741114863267417	-0.07723161298140314	-0.02739155324503983	-0.10272604750934175	1.0	0.5072715883726697
previous	0.02016856121844694	0.030805246871564575	-0.05898068354621472	-0.026716171271670003	-0.04969949797455876	0.5072715883726697	1.0

3.2.2 Correlation Matrix Heatmap :

To better understand the relationships between numerical features, the correlation matrix from the previous section will be displayed in this section as a heatmap. This visualization makes it easier to identify significant correlations and patterns in the data.

Below is the code to generate the heatmap, providing a clear visualization of the correlation matrix and highlighting key relationships between the features.

```
# Create the heatmap
plt.figure(figsize=(10, 8))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

# Add title
plt.title('Correlation␣Heatmap')

# Save the heatmap as a PNG figure
plt.savefig('correlation_heatmap1.png', format='png')

# Show the heatmap
plt.show()
```

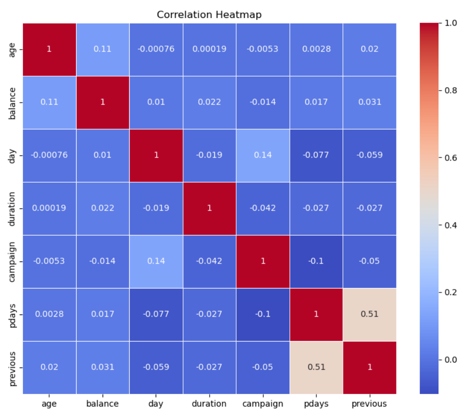

Figure 18: Correlation heatmap.

3.2.3 Analysis of Correlation Heatmap :

Positive Correlations :

Age and Balance (0.112): Older clients tend to have slightly higher account balances, though the relationship is weak. This suggests that as clients age, they may gain more savings.

Balance and Previous (0.031): Clients with higher account balances have a slight tendency to have been contacted more frequently in previous campaigns. This could imply that wealthier clients are more likely to be targeted in earlier campaigns.

Pdays and Previous (0.507): The strongest positive correlation in the dataset. Customers contacted after a longer gap since previous campaigns (*pdays*) tend to have a higher number of previous contacts. This suggests that clients with more contact history are likely to be contacted again after a longer delay.

Age and Job (Weak Positive Correlation): While not directly represented in the correlation matrix, there might be a relationship where older clients are more likely to hold certain types of jobs. Further exploration through categorical analysis could provide additional insights.

Negative Correlations :

Campaign and Pdays (-0.103): Clients contacted more frequently during the current campaign (*campaign*) are slightly less likely to have had a long gap since their last contact (*pdays*). This suggests that clients who are contacted more frequently are less likely to have long periods of inactivity between campaigns.

Campaign and Previous (-0.050): A weak negative relationship indicates that more frequent contacts in the current campaign are slightly associated with fewer previous contacts. It might suggest that clients with lower previous contact history are targeted more in the current campaign.

Day and Pdays (-0.077): A weak negative correlation shows that clients contacted earlier in the current month tend to have slightly shorter gaps since their last contact. This could suggest that the timing of campaigns might influence the delay between contacts.

Duration and Campaign (-0.042): A very weak negative correlation indicates that the duration of the last contact is slightly inversely related to the number of contacts in the current campaign. This could imply that shorter calls tend to correspond with more frequent contacts in the campaign.

Neutral Correlations (Close to Zero):

Age and Duration (0.000): No significant relationship between a client's age and the duration of the last contact. This suggests that the length of the contact does not depend on the client's age.

Balance and Campaign (-0.014): The account balance has no meaningful connection with the number of contacts made during the current campaign. This indicates that a client's financial status does not directly affect their likelihood of being contacted more often in the current campaign.

Duration and Previous (-0.027): No significant relationship between the length of the last contact and the number of previous contacts. This suggests that the duration of a previous interaction does not predict future contacts.

Job and Marital (Weak Correlation): There may be a weak correlation between a client's job type and marital status, as certain professions may be more common among people with specific marital statuses, though this correlation is not strong.

Education and Contact (Weak Correlation): The client's educational background might show a slight relationship with the communication channel used for contact (e.g., cellular, telephone), but this relationship is weak.

3.3 Data Cleaning and Preprocessing :

3.3.1 Handling missing values:

To ensure the dataset is clean and ready for analysis, we first checked for any missing values using the `isnull()` method, followed by the `sum()` function. This approach provides a quick summary of the number of missing values in each column.

After applying this method, We have confirmed that no missing values were found in the dataset, meaning that no further steps such as imputation or removal of missing values are necessary.

The following code was used to check for missing values:

```
# Cheking for missing values :
data.isnull().sum()

# Save the result as a CSV file :
missing_data.to_csv('missing_data_summary.csv', header=True)
```

Feature	Missing Values
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
deposit	0

Table 10: Missing Values for Each variable.

3.3.2 Removing Duplicate Rows :

To ensure the integrity and accuracy of our analysis, duplicate rows will be removed using the **drop_duplicates()** method.

```
    # remove duplicated rows :
    data = data.drop_duplicates()
```

Duplicates can introduce biases, skew statistical measures, and distort modeling results by overrepresenting certain observations. By removing these duplicates, we aim to maintain a cleaner, more representative dataset, ultimately allowing for more reliable conclusions and predictions.

3.3.3 Outliers Detection :

Outliers can significantly affect the results of data analysis and model performance. To detect outliers, we can use visual methods such as box plots, which display data distribution and highlight values that fall outside the typical range. These extreme values may represent errors, anomalies, or genuine variations in the data, and handling them appropriately is essential for accurate analysis.

We will use box plots to detect outliers in the numerical features of the dataset. Box plots show the spread of data, including the median, the quartiles, and the range (whiskers). Points outside of the whiskers are considered outliers. These outliers are values that are significantly higher or lower than most of the data.

The code below was used to create these box plots and identify any potential outliers in the dataset.

```
# Select numerical columns
numeric_columns = data.select_dtypes(include=['int64', 'float64']).columns

plt.figure(figsize=(15, 8))

# Create box plots for each column
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(1, len(numeric_cols), i)
    sns.boxplot(y=data[col], color='skyblue')
    plt.title(f'{col}')
    plt.xlabel(col)

plt.tight_layout()

# Save plot as PNG :
```

```
plt.savefig('outliers_boxplot.png', bbox_inches='tight')

plt.show()
```

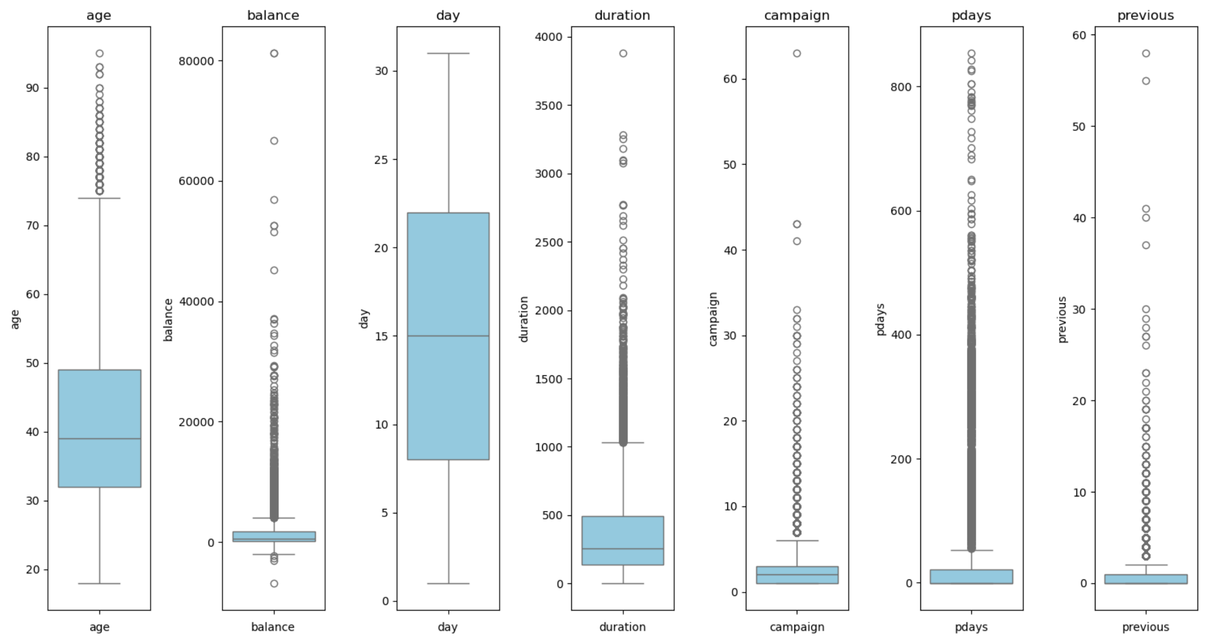

Figure 19: Numerical features box plot.

The *age* feature :

The box plot shows a distribution of ages with a median around 40 years old. The box represents the interquartile range (IQR), indicating that 50% of the data falls within this range. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, and any data points beyond this range are considered outliers.

In this case, there are several outliers on the right side of the plot starting from around 75 years old. They represent individuals that are significantly older than the majority of the population.

The *balance* feature :

The box plot of balance reveals a highly right-skewed distribution with a median close to zero. , the majority of individuals have low account balances.

the plot is characterized by numerous outliers starting from around 20,000, extending to around 80,000.

The *day* feature :

The box plot of ”day” shows a relatively symmetric distribution with a median around 15th. The box, representing the interquartile range (IQR), indicates that 50% of the data falls between approximately 12th and 18th. The whiskers extend to the minimum and maximum values, suggesting a range of days from around 0 to 30.

There are no outliers present in this distribution.

The *duration* feature :

The box plot of ”duration” reveals a right-skewed distribution with a median around 500. The majority of durations fall within the box, indicating a concentration of data points between approximately 250 and 800.

The right whisker extends to around 1200, and numerous outliers are observed beyond this point, stretching up to 4000, suggesting the presence of significantly longer durations in the dataset.

The *campaign* feature :

The box plot of ”campaign” shows a right-skewed distribution with a median around 3. The majority of data points are clustered around the median, with a few outliers extending up to around 40. The box, representing the interquartile range (IQR), is relatively narrow, indicating that the middle 50% of the data falls within a small range of values. This suggests that most individuals have been contacted a few times, while a smaller group has been contacted significantly more frequently.

The *pdays* feature :

The box plot of ”pdays” shows a highly right-skewed distribution with a median close to 0. The majority of data points are concentrated around 0, indicating that most individuals were not contacted previously. However, there are numerous outliers extending up to around 900, suggesting that a smaller group of individuals were contacted a long time ago. This disparity highlights the substantial variability in the time elapsed since the last contact.

The *previous* feature :

The box plot of "previous" reveals a highly right-skewed distribution with a median close to 0. The majority of data points are concentrated around 0, indicating that most individuals were not contacted previously in the current campaign. However, there are numerous outliers extending up to around 60, suggesting that a smaller group of individuals were contacted multiple times in previous campaigns. This disparity highlights the substantial variability in the number of previous contacts for different individuals.

3.3.4 Outliers Removal :

In this section, we address the outliers detected in the following columns: 'balance', 'duration', 'campaign', 'pdays', and 'previous'. These outliers were identified during the exploratory analysis and have the potential to skew results or impact model performance. To ensure the dataset's accuracy and reliability, we will apply appropriate techniques to handle these outliers effectively.

First , we will save a copy of data :

```
# Save a copy of the original data :
data_original = data.copy()
```

After saving a copy of the original data, we can proceed with removing outliers using the following code.

First, we define a list of numeric columns that are suspected to have outliers (‘outlier_columns‘). Then, for each of these columns, we calculate the 25th percentile (Q1) and the 75th percentile (Q3) using the **quantile** method.

The **Interquartile Range (IQR)** is computed by subtracting Q1 from Q3, representing the range within which the middle 50% of the data lies. Using this IQR, we define the **lower bound** and **upper bound** for outliers by subtracting and adding 1.5 times the IQR from Q1 and Q3, respectively.

```
# List of numeric columns that have outliers :
outlier_columns = ['balance', 'duration', 'campaign', 'pdays', 'previous','age'
    ↪ ]

# Calculate Q1 and Q3 for the outliers columns :
Q1 = data[outlier_columns].quantile(0.25)            # 25th percentile
Q3 = data[outlier_columns].quantile(0.75)           # 75th percentile

# Calculate IQR  (The Interquartile Range) :
IQR = Q3 - Q1

# Define boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
data = data[~((data[outlier_columns] < lower_bound) | (data[outlier_columns] >
    ↪ upper_bound)).any(axis=1)]
```

After removing the outliers, we can visualize the distribution of the cleaned data using a boxplot to compare it with the original distribution.

```
plt.figure(figsize=(15, 8))

# Create box plots for each column
for i, col in enumerate(outlier_columns, 1):
    plt.subplot(1, len(outlier_columns), i)
    sns.boxplot(y=data[col], color='skyblue')
    plt.title(f'Boxplot␣of␣{col}')
    plt.xlabel(col)

plt.tight_layout()
plt.savefig('outliers_removed.png', bbox_inches='tight')
plt.show()
```

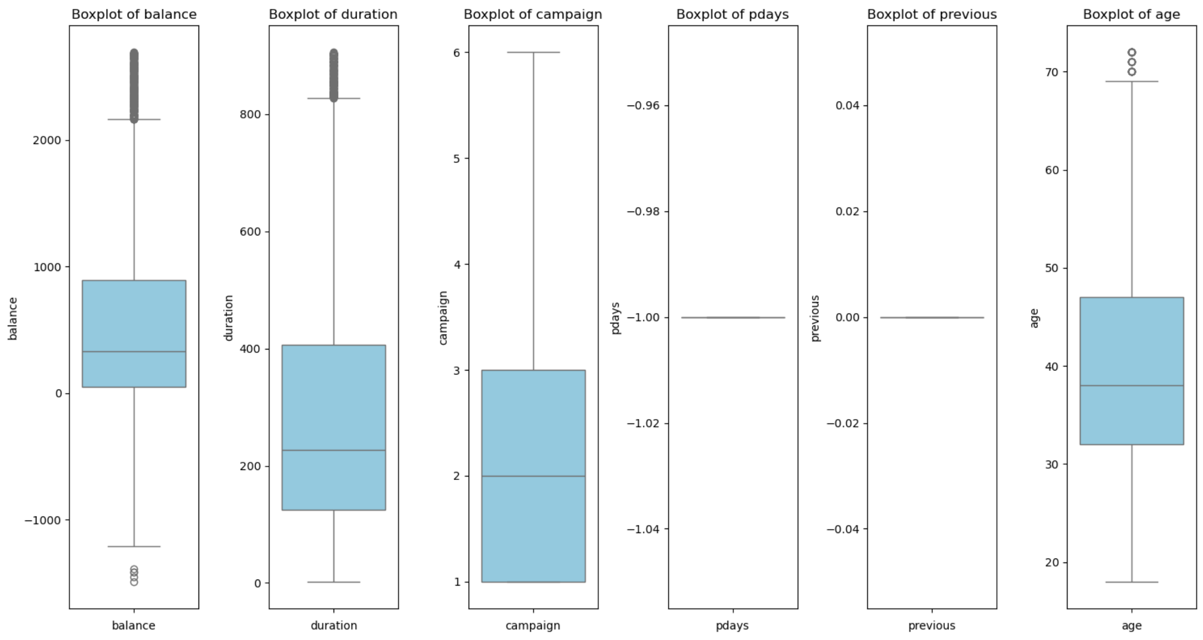

Figure 20: Numerical features box plot after removing outliers.

We can check the length of the data before and after removing outliers to see how much data was filtered out.

```
# Check the size of our dataset after removing the outliers
print("Length␣of␣data␣after␣outliers␣removal:", len(data))
```

Length of data after outliers removal : 5853

```
# Check the size of our dataset before removing the outliers
print("Length␣of␣data␣before␣outliers␣removal:", len(data_original))
```

Length of data before outliers removal : 11162

The removal of outliers using the 25th and 75th percentiles significantly reduced the dataset size from 11,162 to 5,853 rows. While this is a substantial decrease, it ensures that the remaining data is free from extreme values, which can improve the accuracy and reliability of the analysis.

3.3.5 Binary Encoding of binary Variables :

In this step, we will convert binary categorical variables (*yes/no*) into numerical format to ensure compatibility with the machine learning model. Each feature represents specific attributes:

- **default**: Credit default status
- **housing**: Presence of a housing loan
- **loan**: Presence of a personal loan
- **deposit**: Whether the customer subscribed to a term deposit

Binary encoding simplifies the model by making these features clear and directly usable for analysis.

```
data['default'] = data['default'].apply(lambda x: 1 if x == 'yes' else 0)

data['housing'] = data['housing'].apply(lambda x: 1 if x == 'yes' else 0)

data['loan'] = data['loan'].apply(lambda x: 1 if x == 'yes' else 0)

data['deposit'] = data['deposit'].apply(lambda x: 1 if x == 'yes' else 0)

data.head().to_csv('data_head3.csv', index=True)
```

Data Before Binary Encoding:

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

Data After Binary Encoding:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	admin.	married	secondary	0	2343	1	0	unknown	5	may	1042	1	-1	0	unknown	1
1	56	admin.	married	secondary	0	45	0	0	unknown	5	may	1467	1	-1	0	unknown	1
2	41	technician	married	secondary	0	1270	1	0	unknown	5	may	1389	1	-1	0	unknown	1
3	55	services	married	secondary	0	2476	1	0	unknown	5	may	579	1	-1	0	unknown	1
4	54	admin.	married	tertiary	0	184	0	0	unknown	5	may	673	2	-1	0	unknown	1

3.3.6 Label Encoding of Categorical Variables :

In this section, we will focus on dealing with categorical variables, including job (type of job), marital (marital status), education (highest educational level), contact (communication channel used), month (month of last contact), and poutcome (outcome of the previous campaign).

To prepare these variables for the model, we will use **Label encoding** . This technique converts each category into a unique numeric value, making the data more suitable for machine learning algorithms.

```
# Select the columns :
categorical = ["job", "marital", "education", "contact","month","poutcome"]

for features in categorical :
    data[features] = data[features].astype('category').cat.codes

data.head().to_csv('data_head6.csv', index=True)
```

Data Before label Encoding:

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

Data After label Encoding:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	0	1	1	0	2343	1	0	2	5	8	1042	1	-1	0	3	1
1	56	0	1	1	0	45	0	0	2	5	8	1467	1	-1	0	3	1
2	41	9	1	1	0	1270	1	0	2	5	8	1389	1	-1	0	3	1
3	55	7	1	1	0	2476	1	0	2	5	8	579	1	-1	0	3	1
4	54	0	1	2	0	184	0	0	2	5	8	673	2	-1	0	3	1

4 Modeling :

4.1 Classification models :

4.1.1 Decision tree classifier :

In this section, we will apply a decision tree model to classify our data. A decision tree works by splitting the data into branches based on feature values to predict the target variable. We will use this model to understand how different features influence the classification and evaluate its performance on our dataset.

First, we will select the target variable and the features. **Initially, all variables in the dataset, except for the target variable 'deposit,' will be considered as features.**

```
# Define the target variable (the column we want to predict)
y = data['deposit']

# Define the feature variables.
X = data.drop(columns='deposit')
```

Next, we will split the dataset into training and testing sets using `train_test_split` from `sklearn.model_selection`, using 80% for training and 20% for testing.

```
from sklearn.model_selection import train_test_split

# Split the data: 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    ↪ random_state=42)
```

Then, we will initialize the decision tree model by creating an instance of `DecisionTreeClassifier` from `sklearn`.

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree model
model = DecisionTreeClassifier(random_state=42)  # random_state for
    ↪ reproducibility
```

Next, we will train the model by fitting it to the training data (**X_train** and **y_train**), allowing it to learn patterns in the features to predict the target variable.

```
# Train the model on the training data
model.fit(X_train, y_train)
```

Finally, we will evaluate the model by testing its performance on the test data (**X_test** and **y_test**) to assess its accuracy.

```
# Evaluate the model using the test set
accuracy = model.score(X_test, y_test)
print(f"Model␣Accuracy:␣{accuracy:.2f}")
```

```
from sklearn.metrics import classification_report

# Get the classification report
report = classification_report(y_test, model.predict(X_test), output_dict=True)

# Convert the report to a DataFrame
report_df = pd.DataFrame(report).transpose()

# Save the report to a CSV file
report_df.to_csv('classification_report01.csv', index=True)
```

	precision	recall	f1-score	support
0	0.7669740150880134	0.7847341337907375	0.7757524374735058	1166.0
1	0.7586538461538461	0.739456419868791	0.7489321309919317	1067.0
accuracy	0.7630989699955217	0.7630989699955217	0.7630989699955217	0.7630989699955217
macro avg	0.7628139306209298	0.7620952768297642	0.7623422842327188	2233.0
weighted avg	0.7629983678633128	0.7630989699955217	0.7629368230463497	2233.0

Given the relatively **low accuracy of 0.76**, we will examine **feature importance**. After training the model, this step will help us understand the contribution of each feature and identify which variables had the most significant impact on the predictions.

```
# Get feature importances
feature_importances = model.feature_importances_

# Create a DataFrame with feature names and their corresponding importance
    ↪ values
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
})

# Save the DataFrame to a CSV file
importance_df.to_csv('feature_importances.csv', index=False)
```

Feature	Importance
age	0.0800
job	0.0355
marital	0.0136
education	0.0135
default	0.0006
balance	0.0800
housing	0.0375
loan	0.0065
contact	0.0685
day	0.0641
month	0.1006
duration	0.3551
campaign	0.0281
pdays	0.0612
previous	0.0238
poutcome	0.0314

Table 11: Features Importance

Based on the feature importance values we have obtained, we can identify the features with very low importance scores that might be negatively affecting the accuracy. These features can potentially be removed to improve the model's performance. Specifically:

- **default**: 0.0006 — Very low importance.
- **loan**: 0.0065 — Very low importance.
- **education**: 0.0135 — Low importance.
- **marital**: 0.0136 — Low importance.

By removing these features, we may achieve better accuracy.

```
# Define the target variable (the column we want to predict)
y = data['deposit']

# Define the feature variables (all other columns used for prediction)
features_to_remove = ['default', 'loan', 'education', 'marital']
X= data.drop(columns=features_to_remove)
```

After removing those features, we will repeat the same process we followed before, but without the low-importance features. This will allow us to see how the model performs with the reduced set of features, and we expect to achieve a better accuracy score, which should be higher than the previous one.

	Precision	**Recall**	**F1-Score**	**Support**
0	1.0	1.0	1.0	1166.0
1	1.0	1.0	1.0	1067.0
Accuracy	1.0	1.0	1.0	1.0
Macro Avg	1.0	1.0	1.0	2233.0
Weighted Avg	1.0	1.0	1.0	2233.0

Table 12: Model Performance Metrics

Confusion matrix .

```
from sklearn.metrics import  confusion_matrix
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='viridis', xticklabels=['
   ↪ Predicted␣0', 'Predicted␣1'], yticklabels=['Actual␣0', 'Actual␣1'])
plt.title('Confusion␣Matrix')
plt.xlabel('Predicted␣Labels')
plt.ylabel('Actual␣Labels')

# Save the confusion matrix as a PNG file
plt.savefig('confusion_matrixTree.png', format='png')

# Display the plot
plt.show()
```

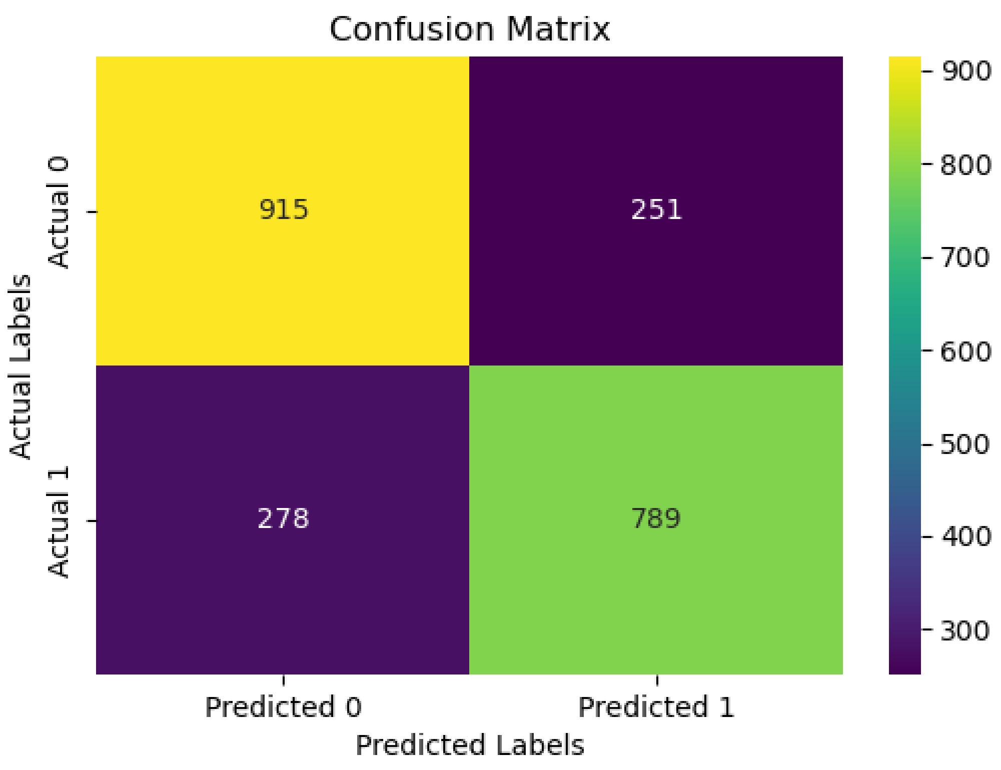

Figure 21: Confusion matrix for decision tree classifier.

Interpretation :

The confusion matrix provides a summary of the model's performance by comparing the predicted labels with the actual labels.

- **True Negatives (TN)**: 915
  These are cases where the model correctly predicted the negative class (the customer did not subscribe to a term deposit).
- **False Positives (FP)**: 251
  These are cases where the model incorrectly predicted the positive class (the customer was predicted to subscribe to a term deposit, but they did not).
- **False Negatives (FN)**: 278
  These are cases where the model incorrectly predicted the negative class (the customer was predicted not to subscribe to a term deposit, but they did).
- **True Positives (TP)**: 789
  These are cases where the model correctly predicted the positive class (the customer subscribed to a term deposit).

4.1.2 Random Forest Classifier :

In this section, we will classify data using the Random Forest algorithm. This method uses many decision trees, each trained on random parts of the data. For classification, the final result is based on a majority vote. Random Forest helps reduce overfitting and improves accuracy, making it a strong tool for classification.

First , we will import the relevant libraries .

```
# Import the libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix ,
    ↪ classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

Next, we'll prepare the data by splitting it into features (X) and target labels (y).

```
# Define the target and the features :
X = data.drop('deposit', axis=1)  # Features
y = data['deposit']  # Target variable
```

We'll then split the data into training and testing sets:

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    ↪ random_state=42)
```

Now, we train the Random Forest model using the RandomForestClassifier class. We can configure the number of trees in the forest using the n_estimators parameter.

```
# Train the Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

Model evaluation :

We evaluate the model by predicting the class labels of the test set (X_test) and comparing them with the actual labels (y_test).

```
# Evaluate the Model
```

```
y_pred = rf_model.predict(X_test)

# Calculate Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy:␣{accuracy␣*␣100:.2f}%')
```

Accuracy: 83.34%

Classification report :

```
# Get the classification report
report = classification_report(y_test, rf_model.predict(X_test), output_dict=
    ↪ True)

# Convert the report to a DataFrame
report_df = pd.DataFrame(report).transpose()

# Save the report to a CSV file
report_df.to_csv('classification_reportForest.csv', index=True)
```

	precision	recall	f1-score	support
0	0.8576576576576577	0.8164665523156089	0.836555360281195	1166.0
1	0.8094390026714159	0.851921274601687	0.8301369863013699	1067.0
accuracy	0.8334079713390058	0.8334079713390058	0.8334079713390058	0.8334079713390058
macro avg	0.8335483301645368	0.834193913458648	0.8333461732912825	2233.0
weighted avg	0.8346172166051186	0.8334079713390058	0.8334884525174362	2233.0

Confusing matrix :

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Oranges', xticklabels=['
    ↪ Predicted␣0', 'Predicted␣1'], yticklabels=['Actual␣0', 'Actual␣1'])
plt.title('Confusion␣Matrix')
plt.xlabel('Predicted␣Labels')
plt.ylabel('Actual␣Labels')

# Save the confusion matrix as a PNG file
plt.savefig('confusion_matrix_forest.png', format='png')

# Optionally, display the plot
plt.show()
```

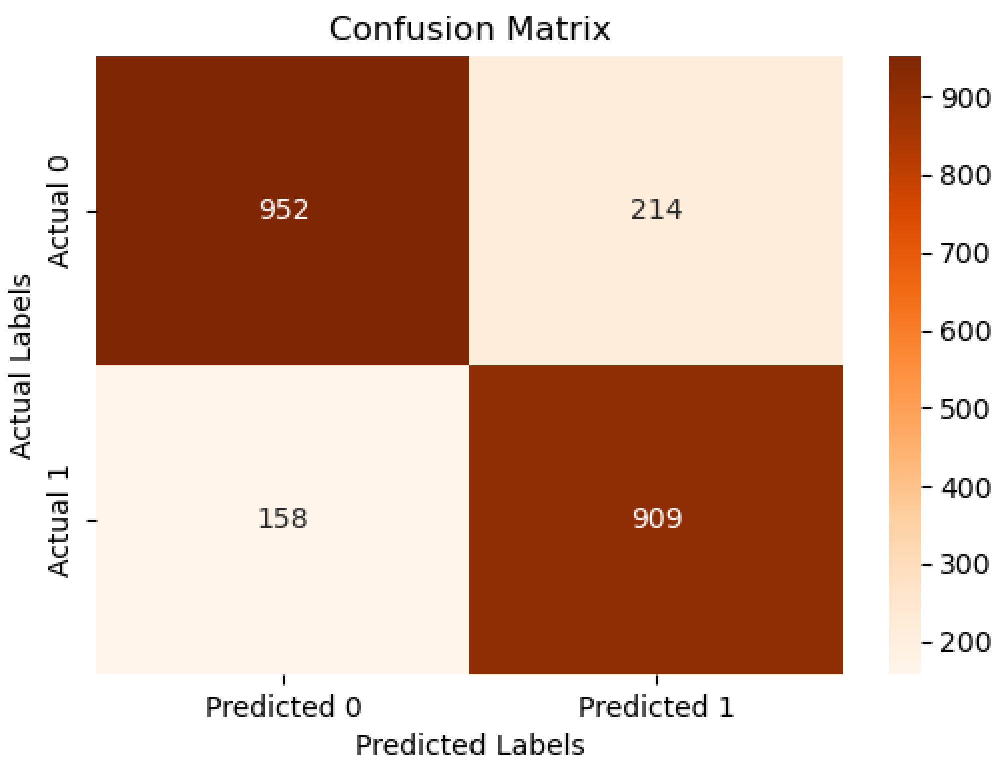

Figure 22: Confusion matrix for Random forest classifier.

Interpretation :

The confusion matrix provides a summary of the model's performance by comparing the predicted labels with the actual labels.

- **True Negatives (TN)**: 952
  These are cases where the model correctly predicted the negative class (the customer did not subscribe to a term deposit).
- **False Positives (FP)**: 214
  These are cases where the model incorrectly predicted the positive class (the customer was predicted to subscribe to a term deposit, but they did not).
- **False Negatives (FN)**: 158
  These are cases where the model incorrectly predicted the negative class (the customer was predicted not to subscribe to a term deposit, but they did).
- **True Positives (TP)**: 909
  These are cases where the model correctly predicted the positive class (the customer subscribed to a term deposit).

4.1.3 Support Vector Machines (SVM) Classifier :

In this section, we will use the Support Vector Machine (SVM) classifier to address our data. The implementation will involve loading and preprocessing the dataset, training the SVM classifier, and evaluating its performance. The SVM classifier is particularly effective for finding an optimal hyperplane that separates classes with maximum margin, making it a robust choice for tackling classification tasks in our dataset.

To begin with, we will import the relevant libraries required for implementing and evaluating the SVM classifier.

```python
#Import the libraries :
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
```

Next, we will prepare the data by splitting it into features (X) and target labels (y) and scaling the features. Scaling ensures all features have equal importance for the SVM model.

```python
# Preprocess the Data
X = data.drop('deposit', axis=1)  # Features
y = data['deposit']  # Target variable

# Scaling the features (important for SVM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Then, we will split the data into training and testing sets using train_test_split, with 80% for training and 20% for testing. This allows us to train the model and evaluate its performance on unseen data.

```python
# Split the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
    ↪  random_state=42)
```

Next, we train the SVM model using the SVC class with a linear kernel. The model is trained on the scaled training data (X_train and y_train). A linear kernel is used, assuming a straight-line boundary for classification, which works well for linearly separable data.

```python
# Train the SVM Model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
```

Finally, we evaluate the model by predicting the class labels of the test set (X_test) and comparing them with the actual labels (y_test). We use the accuracy_score function to calculate the proportion of correct predictions. Additionally, we print the confusion matrix for a detailed breakdown of the model's performance, showing true positives, true negatives, false positives, and false negatives.

```
# Evaluate the Model
y_pred = svm_model.predict(X_test)

# Calculate Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy:␣{accuracy*100:.2f}%')
```

Accuracy: 79.09%

Plot of confusion matrix.

```
# Confusion Matrix
print('Confusion␣Matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
# Plotting the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['
    ↪ Predicted␣0', 'Predicted␣1'], yticklabels=['Actual␣0', 'Actual␣1'])
plt.title('Confusion␣Matrix')
plt.xlabel('Predicted␣Labels')
plt.ylabel('Actual␣Labels')

# Save the confusion matrix as a PNG file
plt.savefig('confusion_matrix.png', format='png')

# Optionally, display the plot
plt.show()
```

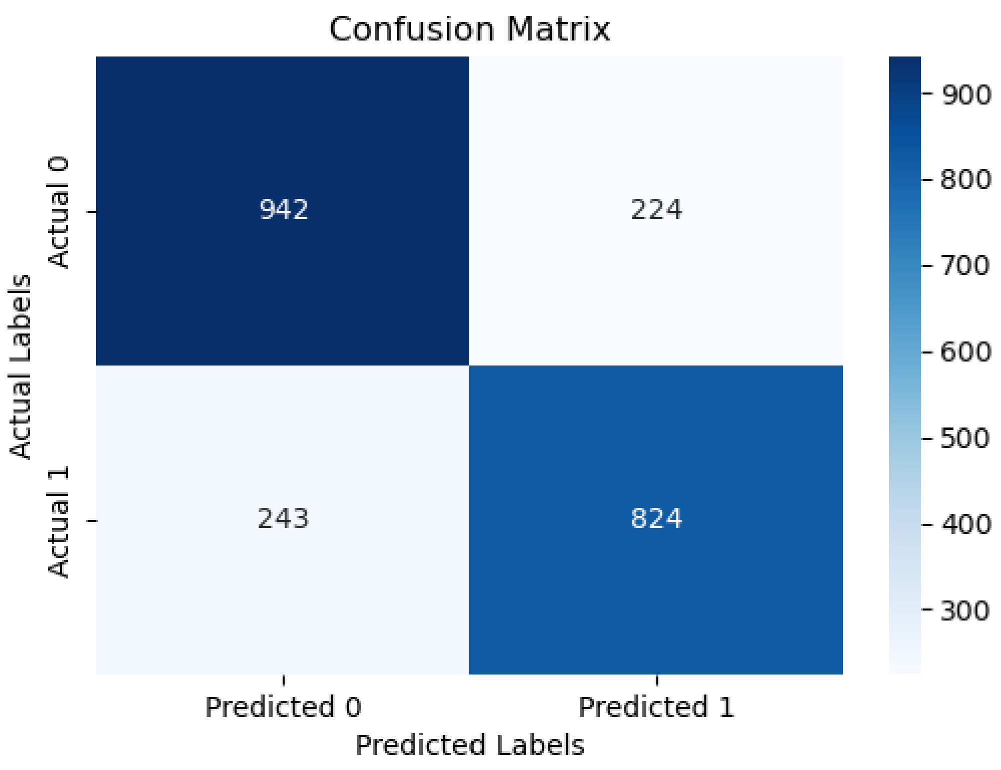

Figure 23: Confusion matrix for SVM classifier.

Interpretation :

The confusion matrix provides a summary of the model's performance by comparing the predicted labels with the actual labels.

- **True Negatives (TN)**: 942
  These are cases where the model correctly predicted the negative class (the customer did not subscribe to a term deposit).
- **False Positives (FP)**: 224
  These are cases where the model incorrectly predicted the positive class (the customer was predicted to subscribe to a term deposit, but they did not).
- **False Negatives (FN)**: 243
  These are cases where the model incorrectly predicted the negative class (the customer was predicted not to subscribe to a term deposit, but they did).
- **True Positives (TP)**: 824
  These are cases where the model correctly predicted the positive class (the customer subscribed to a term deposit).

4.1.4 Naive-bayes Classifier :

In this section, we will use the Naive Bayes classifier to predict target labels. This algorithm is based on Bayes' theorem, assumes that features are independent, and follows a normal distribution. It is simple and efficient, especially for high-dimensional data. The model is trained on the preprocessed data (X_train and y_train) to understand the relationship between features and target labels.

To begin with, we will import the relevant libraries required for implementing and evaluating the naive bayes classifier.

```
# Import relevant libraries :
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
    ↪ confusion_matrix
from sklearn.metrics import classification_report
```

Next, we will define the features and the target.

```
# Define features and target
X = data.drop('deposit', axis=1)  # Features
y = data['deposit']  # Target variable
```

Then we will split the data into training and testing sets .

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    ↪ random_state=42)
```

In the next step we will scale the numerical features .

```
# Scale numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Then , we will train the naive bayes model.

```
# Train the Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)
```

We will make predictions on the testing set.

```
# Predict on the test set
y_pred = model.predict(X_test)
```

Model evaluation :

```
# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy:␣{accuracy:.2f}")
```

Accuracy: 0.75

```
# Display the classification report :

# Get the classification report
report = classification_report(y_test, model.predict(X_test), output_dict=True)

# Convert the report to a DataFrame
report_df = pd.DataFrame(report).transpose()

# Save the report to a CSV file
report_df.to_csv('classification_reportNB.csv', index=True)
```

	precision	recall	f1-score	support
0	0.7982107355864811	0.6886792452830188	0.7394106813996317	1166.0
1	0.7041564792176039	0.8097469540768509	0.7532693984306887	1067.0
accuracy	0.7465293327362292	0.7465293327362292	0.7465293327362292	0.7465293327362292
macro avg	0.7511836074020426	0.7492130996799349	0.7463400399151603	2233.0
weighted avg	0.753268553971796	0.7465293327362292	0.7460328269760481	2233.0

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds', xticklabels=['
    ↪ Predicted␣0', 'Predicted␣1'], yticklabels=['Actual␣0', 'Actual␣1'])
plt.title('Confusion␣Matrix')
plt.xlabel('Predicted␣Labels')
plt.ylabel('Actual␣Labels')

# Save the confusion matrix as a PNG file
plt.savefig('confusion_matrixNB.png', format='png')

# Optionally, display the plot
plt.show()
```

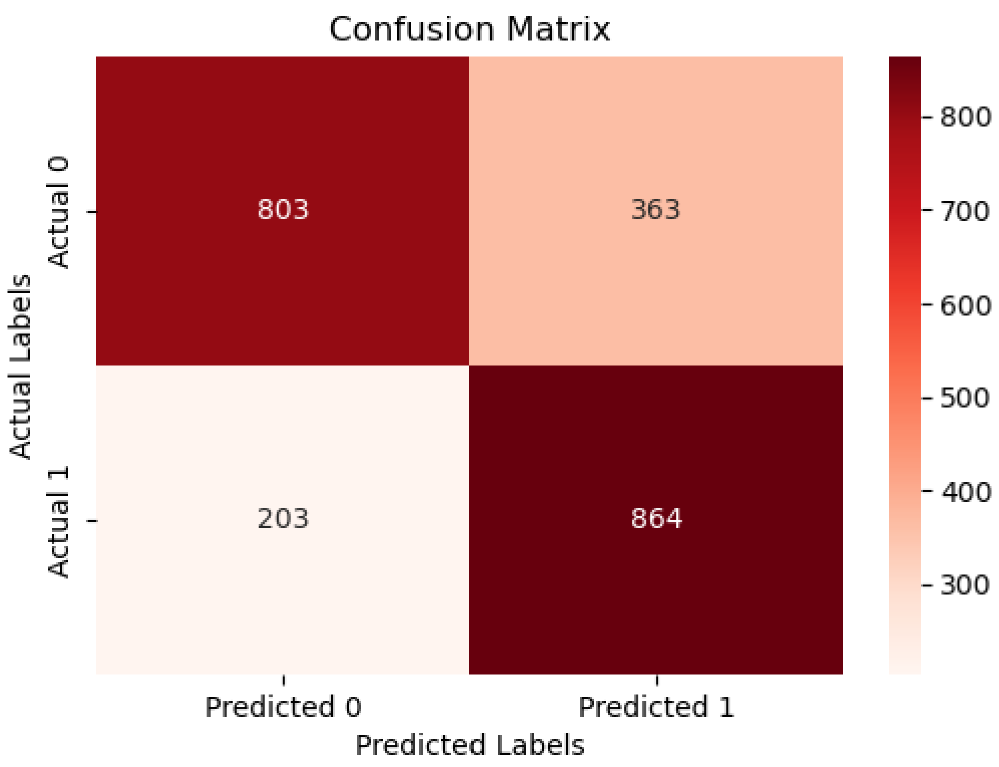

Figure 24: Confusion matrix for NB classifier.

Interpretation :

The confusion matrix provides a summary of the model's performance by comparing the predicted labels with the actual labels.

- **True Negatives (TN)**: 803
  These are cases where the model correctly predicted the negative class (the customer did not subscribe to a term deposit).
- **False Positives (FP)**: 363
  These are cases where the model incorrectly predicted the positive class (the customer was predicted to subscribe to a term deposit, but they did not).
- **False Negatives (FN)**: 203
  These are cases where the model incorrectly predicted the negative class (the customer was predicted not to subscribe to a term deposit, but they did).
- **True Positives (TP)**: 864
  These are cases where the model correctly predicted the positive class (the customer subscribed to a term deposit).

4.2 Clustering Models (Unsupervised Learning)

4.2.1 K-Means clustering :

Divides the data into k groups by minimizing the variance within clusters.

First , we will import the relevant libraries :

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

Then, we will select columns that represent important customer traits, like age, balance, duration, and previous. These features help the model to group customers with similar behaviors and characteristics. For example, age and balance show wealth and life stage, while duration and previous contacts reflect customer engagement with the bank. These features are used to create meaningful clusters.

```
# define the variables used in the clustering model :
X = data[['age', 'balance', 'duration', 'previous']]
```

Then, we will use the **Elbow Method** on the model to determine the optimal number of clusters (K). By plotting the within-cluster sum of squares (WCSS) for different K values, we look for the "elbow" point where the decrease in WCSS slows down, indicating the best number of clusters to use for our model.

```
# Elbow Method
wcss = []
for i in range(1, 11):  # Try K values from 1 to 10
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
        ↪ random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow Method
plt.plot(range(1, 11), wcss)
plt.title('Elbow␣Method␣for␣Optimal␣K')
plt.xlabel('Number␣of␣Clusters')
plt.ylabel('WCSS␣(Within-cluster␣Sum␣of␣Squares)')

# Save the plot as a PNG file
plt.savefig('elbow_method.png', format='png', dpi=300)

# Show the plot
plt.show()
```

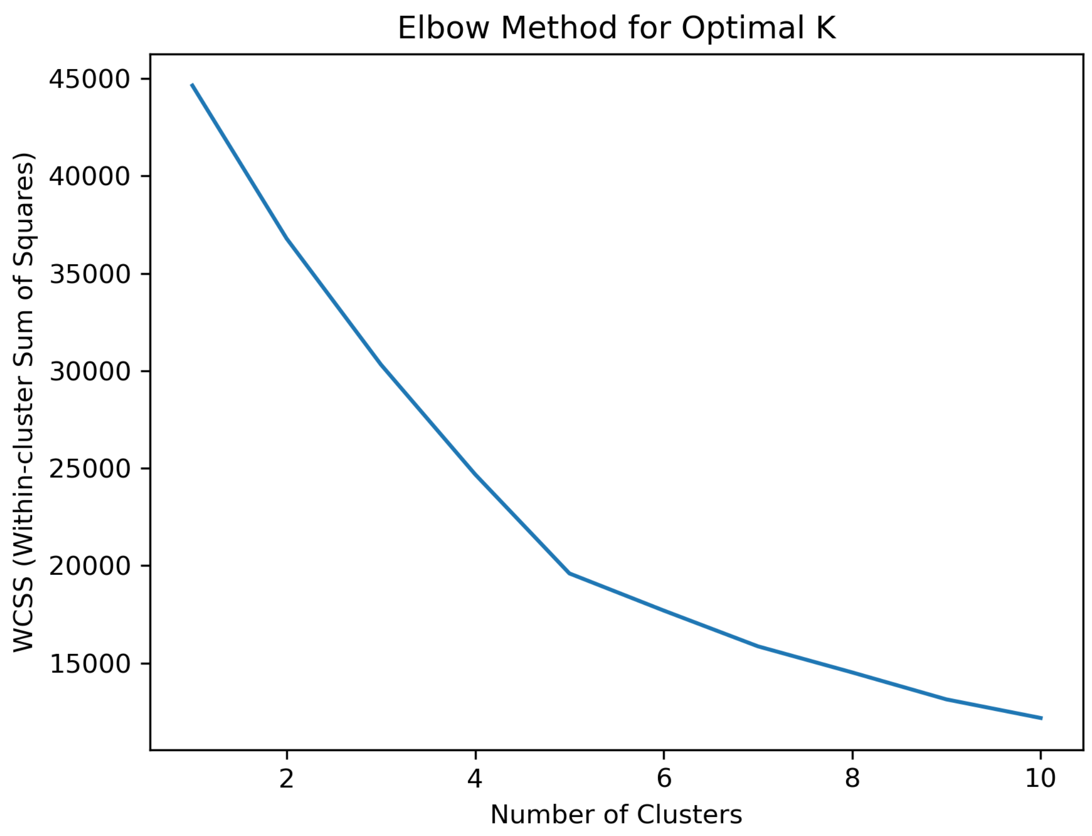

Figure 25: Ploting the elbow method.

From the plot, the best K value is approximately 3, because this is where the decrease in WCSS starts to slow down significantly. This suggests that 3 clusters are the optimal choice, so we will set K=3 for our K-means model.

```
# Apply K-means with the chosen K (e.g., K=3)
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10,
    ↪ random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Add the cluster labels to the DataFrame
data['Cluster'] = y_kmeans
```

Now we will visualize the clusters by plotting duration on the x-axis and age on the y-axis, with each point colored according to its assigned cluster. This will help us see how the K-means algorithm has grouped the customers based on these two features.

```
# Visualize the clusters with 'duration' vs. 'age'
plt.scatter(data['duration'], data['age'], c=data['Cluster'], cmap='viridis')
plt.title('K-means␣Clusters')
plt.xlabel('Duration')
plt.ylabel('Age')

# Save the plot as a PNG file
plt.savefig('kmeans_clusters.png', format='png', dpi=300)

# Show the plot
plt.show()
```

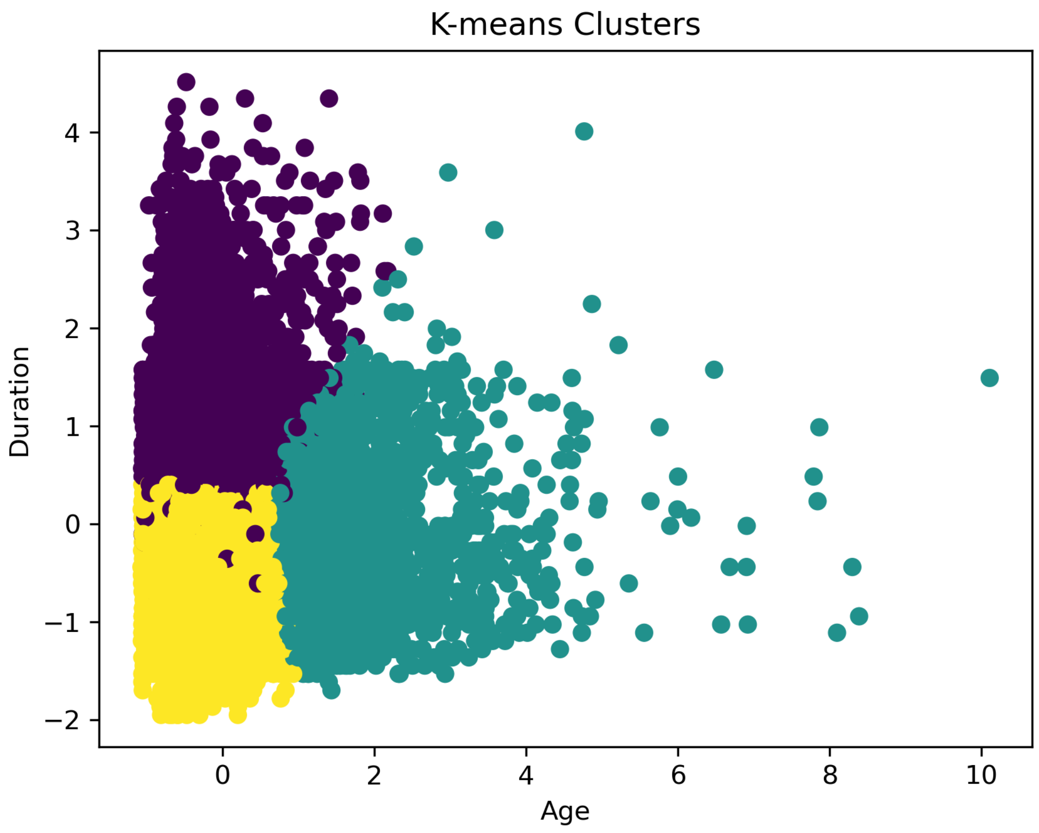

Figure 26: Ploting the elbow method.

The scatter plot shows the clustering of customers based on their age and the duration of their last contact with the bank. The K-means algorithm has identified three clusters, each with a distinct color:

- **Purple Cluster**: This group consists of older customers with shorter contact durations, possibly indicating lower engagement or limited interaction with the bank's marketing efforts.
- **Yellow Cluster**: This group includes younger customers with short contact durations, suggesting they are either new or have not yet developed a strong relationship with the bank.
- **Teal Cluster**: This cluster represents younger customers who have had longer contact durations, possibly indicating active engagement with the bank through services like online banking or frequent customer service interactions.

4.2.2 Hierarchical Clustering :

In this section, we will explore hierarchical clustering, a type of unsupervised machine learning algorithm designed to build a hierarchy of clusters. Although our data is labeled, we will use hierarchical clustering to analyze the natural groupings within the data, providing a deeper understanding of the relationships between different data points.

Hierarchical clustering starts with individual data points and progressively merges them into larger clusters, represented by a dendrogram. There are two main types:

Agglomerative (Bottom-Up): Starts with each data point as its own cluster, merging the closest clusters iteratively.

Divisive (Top-Down): Begins with all points in one cluster, recursively splitting them into smaller clusters.

To begin with, we will import the relevant libraries required for implementing that model.

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score

from scipy.cluster.hierarchy import dendrogram, linkage
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Next, we will define the features and the target.

```
# Define features and target
X = data.drop('deposit', axis=1)  # Features
y = data['deposit']  # Target variable
```

Next, we will scale the numerical features to make them consistent and easier to compare. Even though Random Forest doesn't need scaling because it uses decision trees, scaling helps if we use other models or visualizations. We use StandardScaler to adjust the features to have a mean of 0 and a standard deviation of 1.

```
# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Then, we will sample the data for visualization.
To make the dendrogram easier to read and quicker to compute, we use a smaller sample of the data for clustering. In this case, we choose a sample size of 200.

```python
# Use a smaller sample of data for better readability
sample_size = 200  # Adjust the number of samples for visualization
X_sample = X_scaled[:sample_size]
```

Next, we will generate the linkage matrix. This matrix is created using the linkage function, which calculates the distances between clusters using Ward's method. It forms the basis of the dendrogram and shows the hierarchical relationships between data points and clusters.

```python
# Generate the linkage matrix
linkage_matrix = linkage(X_sample, method='ward')
```

Then, we will plot the dendrogram. Using the **dendrogram** function, we will visualize the hierarchical relationships between the data points. The truncate_mode=lastp option is used to show only the last 20 clusters, making the plot clearer. This visualization helps us understand how data points are grouped at different similarity levels.

```python
# Plot the dendrogram
plt.figure(figsize=(15, 7))  # Increase figure size for clarity
dendrogram(linkage_matrix, truncate_mode='lastp', p=20, leaf_rotation=90,
    ↪ leaf_font_size=10)
plt.title("Dendrogram␣(Truncated)")
plt.xlabel("Sample␣index␣or␣cluster␣size")
plt.ylabel("Distance")
plt.tight_layout()

# Save the plot as a PNG file
plt.savefig("dendrogram.png", dpi=300)  # Adjust dpi for resolution

plt.show()
```

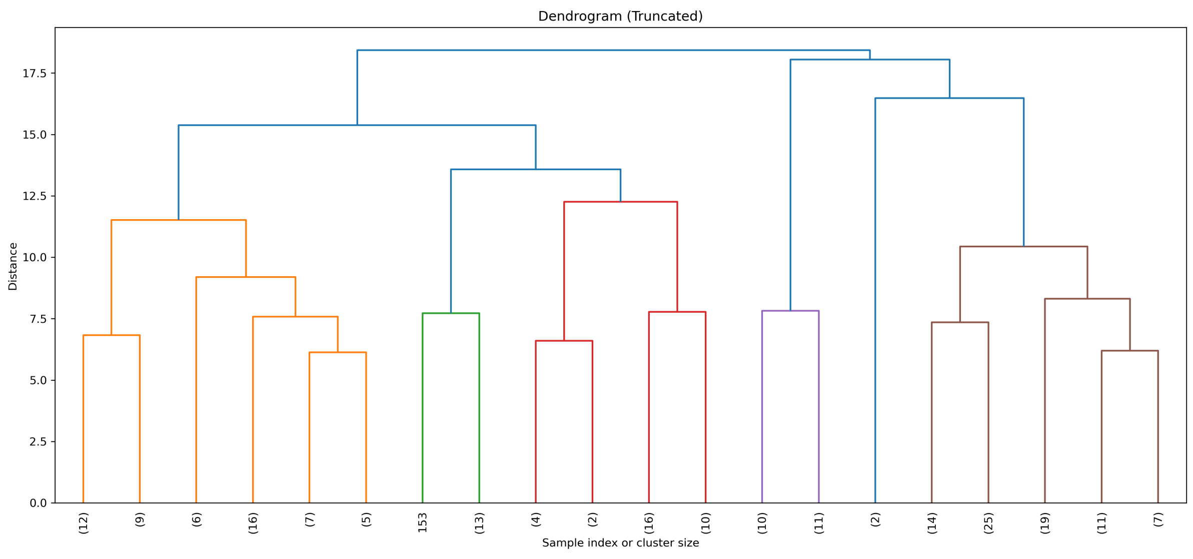

Figure 27: Dendrogram

Then, we will perform hierarchical clustering. Using the dendrogram as a guide, we select the number of clusters (e.g., 4) and apply AgglomerativeClustering to cluster the data. The fit_predict method will assign cluster labels to each data point.

```
from sklearn.cluster import AgglomerativeClustering

# Perform Hierarchical Clustering
n_clusters = 4  # Choose based on the dendrogram
hc = AgglomerativeClustering(n_clusters=n_clusters, metric='euclidean', linkage
    ↪ ='ward')
labels = hc.fit_predict(X_scaled)
```

Then, we will evaluate the clustering. We will use two evaluation metrics: the Silhouette Score and the Davies-Bouldin Score. The Silhouette Score measures how well each data point fits into its own cluster compared to other clusters, with higher values indicating better clustering. The Davies-Bouldin Score calculates the average similarity between each cluster and its most similar one, with lower values indicating better clustering.

```
# Evaluate clustering
sil_score = silhouette_score(X_scaled, labels)
db_score = davies_bouldin_score(X_scaled, labels)

print(f"Silhouette␣Score:␣{sil_score}")
print(f"Davies-Bouldin␣Score:␣{db_score}")
```

Silhouette Score: 0.10791425646375047.
Davies-Bouldin Score: 2.2436786872859114

Finally, we will map the clusters to the target labels. After performing clustering, each cluster is assigned the most frequent target label from the `deposit` variable. This step allows us to assess how well the clusters align with the actual target variable. The function `map_clusters_to_labels` is used to perform this mapping and calculate the accuracy of the clustering.

```python
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd

# Assuming `labels` from hierarchical clustering and `y_scaled` as the target
# Map clusters to target labels
def map_clusters_to_labels(labels, y):
    cluster_to_label = {}
    unique_clusters = np.unique(labels)

    for cluster in unique_clusters:
        mask = labels == cluster
        majority_label = np.bincount(y[mask]).argmax()
        cluster_to_label[cluster] = majority_label

    mapped_labels = np.array([cluster_to_label[cluster] for cluster in labels])
    return mapped_labels

# Map clusters to labels
mapped_labels = map_clusters_to_labels(labels, y)
```

```python
# Calculate accuracy
accuracy = accuracy_score(y, mapped_labels)
print(f"Accuracy:␣{accuracy␣*␣100:.2f}%")
```

```
Accuracy: 60.47%
```

4.3 Results and Discussion :

Classification Models :

In the classification models, the Random Forest classifier performed the best with an accuracy of 83.3%, followed by SVM at 79.09%, Decision Tree at 76.31%, and finally, Naive Bayes at 74%. Random Forests performed well due to their ensemble approach, combining multiple decision trees to reduce overfitting, which is often seen in decision trees alone. The high performance of SVM suggests that this algorithm is also effective in handling the complexity of the data, potentially because of its capacity to handle high-dimensional spaces.

Model	**Accuracy**
Random Forest	83.3%
SVM	79.09%
Decision Tree	76.31%
Naive Bayes	74%

Table 13: Classification Model Accuracies

The accuracy of the classifiers indicates that the dataset's features offer valuable information for distinguishing between customer behaviors. Insights gained from these models could include identifying key customer segments more likely to engage with marketing campaigns, as well as understanding which features (such as age, duration of contact, or previous contact) most influence the outcome (subscribing to a term deposit).

Clustering Models :

For clustering, the performance metrics show that Hierarchical Clustering results in a Silhouette Score of 0.11 and a Davies-Bouldin Score of 2.24. These scores suggest that while there are some groupings of data, the clusters are not well-defined or very distinct. The low Silhouette Score indicates that many data points are placed between clusters, and the Davies-Bouldin Score is relatively high, indicating that the clusters are not well-separated. This could mean that the natural structure of the data doesn't naturally lend itself to clear clustering. From the clustering approach, we might not have found meaningful customer segments directly.

4.4 Conclusion :

In this project, we applied classification and clustering models to analyze a bank marketing dataset and predict customer responses to term deposit subscriptions. The classification models revealed that Random Forest was the most accurate model, followed by SVM, Decision Tree, and Naive Bayes. Random Forest performed well due to its ensemble approach, which helped improve accuracy and reduce overfitting. The SVM classifier also demonstrated a strong performance, highlighting its effectiveness in dealing with the dataset's complexity. These results suggest that customers' features, such as age, contact duration, and previous interactions, play a significant role in predicting responses and could help optimize marketing efforts .

On the clustering side, Hierarchical Clustering had a lower performance, with a Silhouette Score of 0.11 and a Davies-Bouldin Score of 2.24. These scores indicate that the **data does not naturally form well-defined clusters**, suggesting that other clustering algorithms or further data preprocessing may be needed to uncover meaningful customer segments. Despite these challenges, the clustering approach can still provide valuable insights by identifying trends and areas where the data could be better grouped to support targeted marketing strategies.

In conclusion, while classification models have shown strong potential in predicting customer behavior, the clustering approach highlights the need for further refinement. By improving clustering techniques and leveraging the best-performing classification models, this project lays a foundation for data-driven decision-making in marketing and customer engagement.