Problem statement:Build a decision tree classifier to predict whether a customer will purchase a product or service based on their demographic and behavioral data. Dataset:Bank Marketing About the dataset: The dataset contains demographic and behavioral information of individuals contacted during a marketing campaign. It includes features such as age, job type, marital status, education level, credit default status, housing loan status, and contact details. The target variable 'y' indicates whether each individual subscribed to a product or service ('yes' or 'no').

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier

# Load the data
data_path = 'c:\\Users\\Admin\\Downloads\\bank.csv'
df = pd.read_csv(data_path, sep=';')

# Display the first few rows of the dataframe
print(df.head())
```

```
   age        job  marital    education  default housing loan    contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone   
1   57   services  married  high.school  unknown      no   no  telephone   
2   37   services  married  high.school       no     yes   no  telephone   
3   40     admin.  married     basic.6y       no      no   no  telephone   
4   56   services  married  high.school       no      no  yes  telephone   

  month day_of_week  ...  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  ...         1    999         0  nonexistent          1.1   
1   may         mon  ...         1    999         0  nonexistent          1.1   
2   may         mon  ...         1    999         0  nonexistent          1.1   
3   may         mon  ...         1    999         0  nonexistent          1.1   
4   may         mon  ...         1    999         0  nonexistent          1.1   

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y  
0          93.994          -36.4      4.857       5191.0  no  
1          93.994          -36.4      4.857       5191.0  no  
2          93.994          -36.4      4.857       5191.0  no  
3          93.994          -36.4      4.857       5191.0  no  
4          93.994          -36.4      4.857       5191.0  no  

[5 rows x 21 columns]
```

In [ ]:
```
# Check the structure of the dataset
print(df.info())

# Summary statistics of the dataset
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
None
               age      duration      campaign         pdays      previous  \
count  41188.00000  41188.000000  41188.000000  41188.000000  41188.000000   
mean      40.02406    258.285010      2.567593    962.475454      0.172963   
std       10.42125    259.279249      2.770014    186.910907      0.494901   
min       17.00000      0.000000      1.000000      0.000000      0.000000   
25%       32.00000    102.000000      1.000000    999.000000      0.000000   
50%       38.00000    180.000000      2.000000    999.000000      0.000000   
75%       47.00000    319.000000      3.000000    999.000000      0.000000   
max       98.00000   4918.000000     56.000000    999.000000      7.000000   

       emp.var.rate  cons.price.idx  cons.conf.idx     euribor3m   nr.employed  
count  41188.000000    41188.000000   41188.000000  41188.000000  41188.000000  
mean       0.081886       93.575664     -40.502600      3.621291   5167.035911  
std        1.570960        0.578840       4.628198      1.734447     72.251528  
min       -3.400000       92.201000     -50.800000      0.634000   4963.600000  
25%       -1.800000       93.075000     -42.700000      1.344000   5099.100000  
50%        1.100000       93.749000     -41.800000      4.857000   5191.000000  
75%        1.400000       93.994000     -36.400000      4.961000   5228.100000  
max        1.400000       94.767000     -26.900000      5.045000   5228.100000  
```

In [ ]:

```python
# Check for missing values
missing_values = df.isnull().sum()
print(missing_values)
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

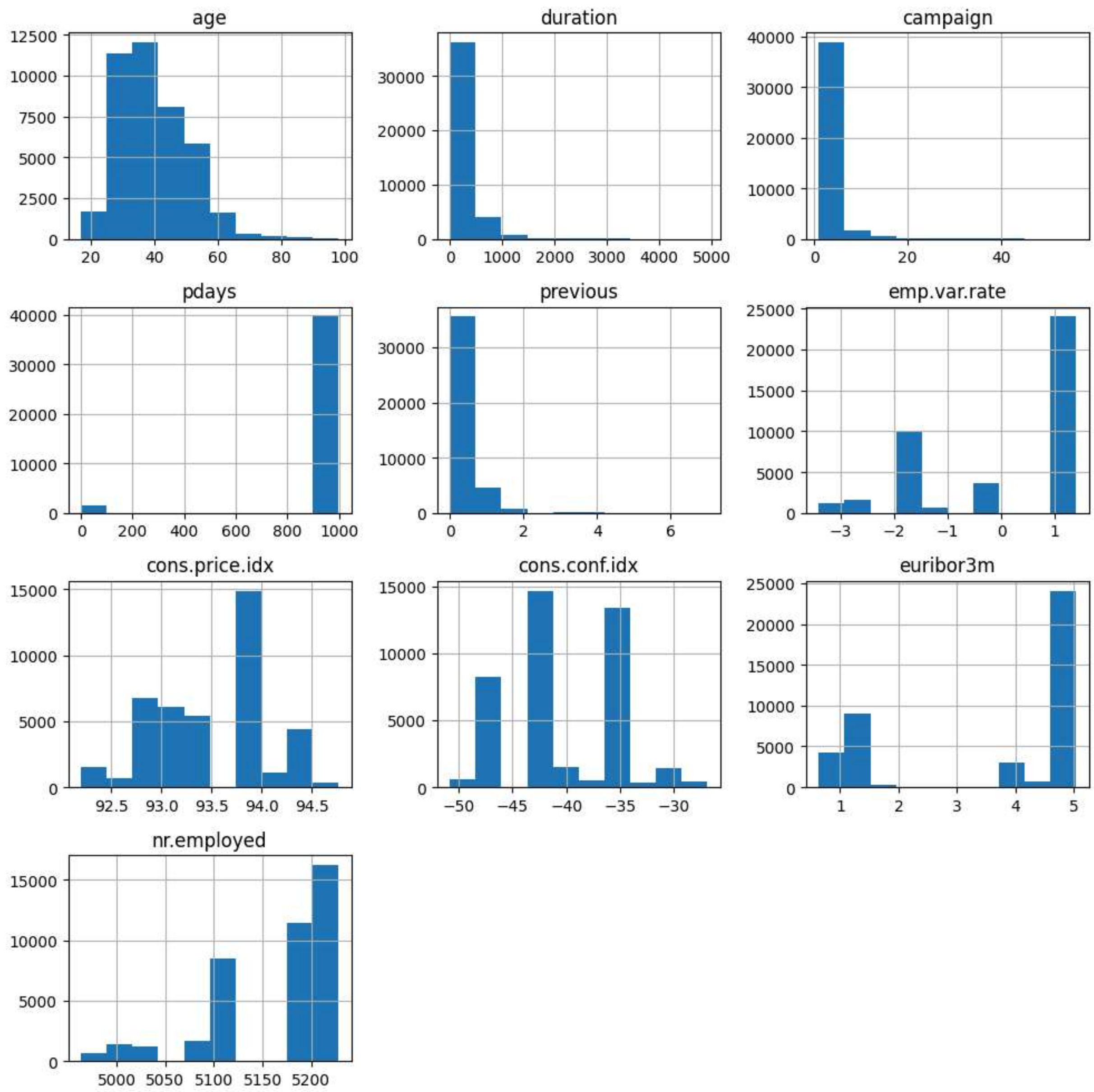
In [ ]:

```python
# Analyze categorical variables
categorical_columns = df.select_dtypes(include=['object']).columns

# Display unique values and their counts for each categorical column
for col in categorical_columns:
    print(f'Column: {col}')
    print(df[col].value_counts())
    print('\n')
```

```
Column: job
job
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: count, dtype: int64


Column: marital
marital
married     24928
single      11568
divorced     4612
unknown        80
Name: count, dtype: int64


Column: education
education
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: count, dtype: int64


Column: default
default
no         32588
unknown     8597
yes            3
Name: count, dtype: int64


Column: housing
housing
yes        21576
no         18622
unknown      990
Name: count, dtype: int64


Column: loan
```

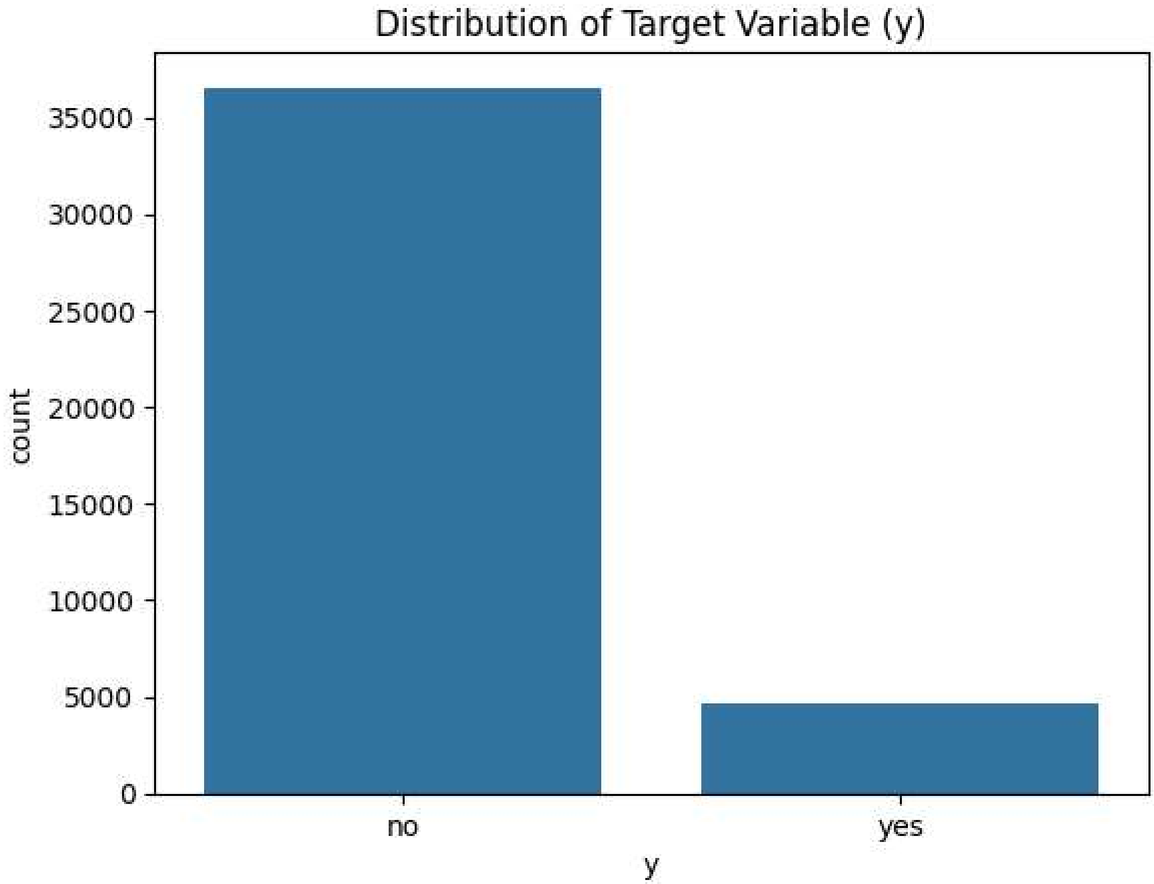
```
loan
no         33950
yes         6248
unknown      990
Name: count, dtype: int64


Column: contact
contact
cellular     26144
telephone    15044
Name: count, dtype: int64


Column: month
month
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182
Name: count, dtype: int64


Column: day_of_week
day_of_week
thu    8623
mon    8514
wed    8134
tue    8090
fri    7827
Name: count, dtype: int64


Column: poutcome
poutcome
nonexistent    35563
failure         4252
success         1373
Name: count, dtype: int64


Column: y
y
no     36548
yes     4640
Name: count, dtype: int64
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Histograms for numerical variables
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns

df[numerical_columns].hist(figsize=(10, 10))
plt.tight_layout()
plt.show()
```

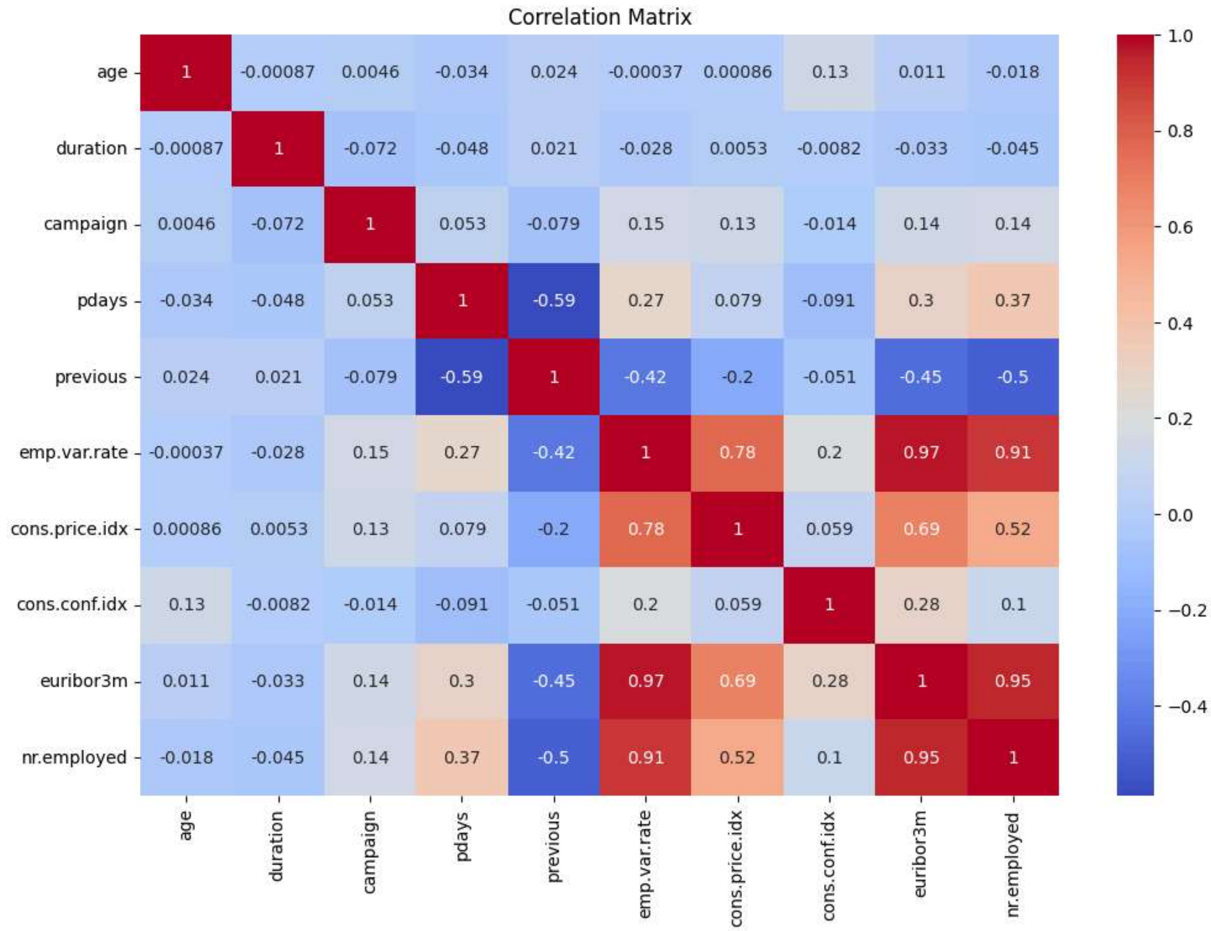


In [ ]:
```
# Analyze the target variable 'y'
print(df['y'].value_counts())

# Plot the distribution of the target variable
sns.countplot(x='y', data=df)
plt.title('Distribution of Target Variable (y)')
plt.show()
```

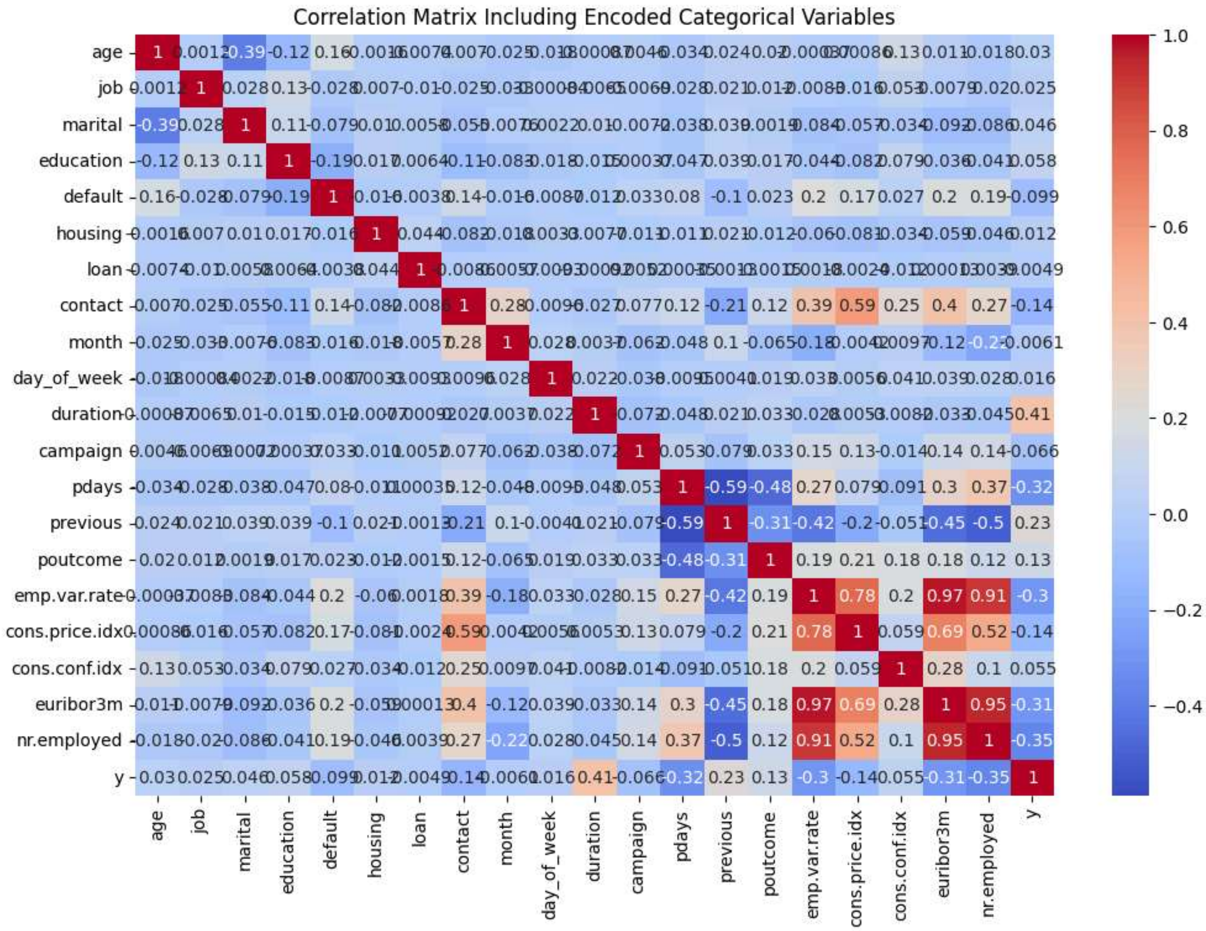
```
y
no     36548
yes     4640
Name: count, dtype: int64
```



In [ ]:
```
# Select only numerical columns for correlation matrix
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
corr_matrix = df[numerical_columns].corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




```
In [ ]: # Encode categorical variables using LabelEncoder
from sklearn.preprocessing import LabelEncoder

label_encoders = {}
categorical_columns = df.select_dtypes(include=['object']).columns

for col in categorical_columns:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])

# Compute the correlation matrix including encoded categorical variables
corr_matrix = df.corr()

# heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Including Encoded Categorical Variables')
plt.show()
```



```
In [ ]: # Assign X (features) and y (target)
        X = df.drop('y_yes', axis=1)
        y = df['y_yes']

        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
In [ ]: # Initialize the classifier
        dt_classifier = DecisionTreeClassifier(random_state=42)
        # Train the classifier on the training data
        dt_classifier.fit(X_train, y_train)
```

```
Out[ ]: ▾ DecisionTreeClassifier
        DecisionTreeClassifier(random_state=42)
```

```
In [ ]: # Predict on the test data
        y_pred = dt_classifier.predict(X_test)

        # Evaluate the model
        from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

        print("Accuracy:", accuracy_score(y_test, y_pred))
        print("\nClassification Report:\n", classification_report(y_test, y_pred))
        print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8866229667394999

Classification Report:
               precision    recall  f1-score   support

       False       0.94      0.93      0.94      7303
        True       0.50      0.52      0.51       935

    accuracy                           0.89      8238
   macro avg       0.72      0.73      0.72      8238
weighted avg       0.89      0.89      0.89      8238


Confusion Matrix:
 [[6816  487]
 [ 447  488]]
```

The performance metrics of a decision tree classifier trained on demographic and behavioral data to predict whether customers will purchase a product or service. The classifier achieved an accuracy of approximately 88.7%, indicating that it correctly predicted the outcome for nearly 89% of the instances in the test set. The classification report reveals that the model performs well in identifying non-purchasing customers (precision of 94% and recall of 93%), but less effectively for purchasing customers (precision of 50% and recall of 52%). The confusion matrix further illustrates these results, showing 6816 true negatives, 487 false negatives, 447 false positives, and 488 true positives