Introduction

This project involves analyzing data from a bank's marketing campaign to predict whether a client will subscribe to a term deposit. The dataset includes various attributes such as job, marital status, education, default status, balance, and other socio-economic indicators. The goal is to build a classification model and derive insights that can improve future campaign strategies.

1. Loaded and inspected the bankmarketing.csv dataset.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set styles for plots
sns.set(style='whitegrid')
%matplotlib inline
```

In [4]:
```
# Replace 'your_file.csv' with the path to your actual file
df = pd.read_csv('bankmarketing.csv')

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

Out[4]:

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

5 rows × 21 columns

2. Handled missing values and performed necessary data cleaning.

In [5]:

```
# Shape of the dataset
print("Shape:", df.shape)

# Data types and non-null counts
df.info()

# Summary statistics for numerical columns
df.describe()

# Check for missing values
df.isnull().sum()
```

```
Shape: (41188, 21)
<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
```

Out[5]:

```
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
```

Data Cleaning

In [101…

```python
# Count 'unknown' values in each column
for col in df.columns:
    print(f"{col}: {df[col][df[col] == 'unknown'].count()}")
```

```
age: 0
duration: 0
campaign: 0
pdays: 0
previous: 0
emp.var.rate: 0
cons.price.idx: 0
cons.conf.idx: 0
euribor3m: 0
nr.employed: 0
y: 0
job_blue-collar: 0
job_entrepreneur: 0
job_housemaid: 0
job_management: 0
job_retired: 0
job_self-employed: 0
job_services: 0
job_student: 0
job_technician: 0
job_unemployed: 0
job_unknown: 0
marital_married: 0
marital_single: 0
marital_unknown: 0
education_basic.6y: 0
education_basic.9y: 0
education_high.school: 0
education_illiterate: 0
education_professional.course: 0
education_university.degree: 0
education_unknown: 0
default_unknown: 0
default_yes: 0
housing_unknown: 0
housing_yes: 0
loan_unknown: 0
loan_yes: 0
contact_telephone: 0
month_aug: 0
month_dec: 0
month_jul: 0
month_jun: 0
month_mar: 0
month_may: 0
month_nov: 0
month_oct: 0
month_sep: 0
day_of_week_mon: 0
day_of_week_thu: 0
day_of_week_tue: 0
day_of_week_wed: 0
poutcome_nonexistent: 0
poutcome_success: 0
```

In [102…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def invoke(input_0: pd.DataFrame) -> pd.DataFrame:
    """
    Args:
        input_0 (pd.DataFrame): bankmarketing.csv
    Returns:
        pd.DataFrame: output
    """
    output = input_0
    return output

input_0 = pd.read_csv('bankmarketing.csv')
df = invoke(input_0)

print(df.columns)
print(df.head())
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
   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 [103…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def invoke(input_0: pd.DataFrame) -> pd.DataFrame:
    """
    Args:
        input_0 (pd.DataFrame): bankmarketing.csv
```

```
    Returns:
        pd.DataFrame: output
    """
    output = input_0
    return output

input_0 = pd.read_csv('bankmarketing.csv')
df = invoke(input_0)

print(df.columns)
print(df.head())
```

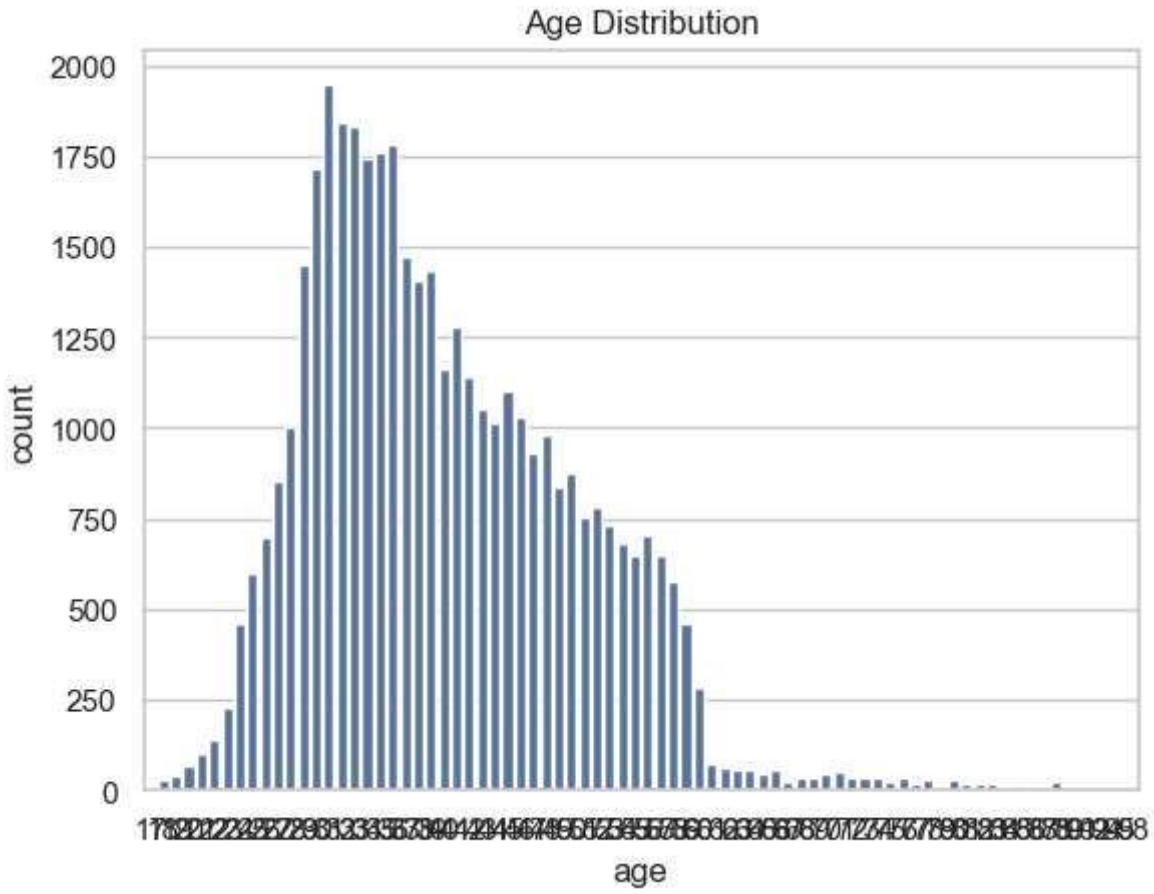
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
   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]
```

3. Conducted exploratory data analysis (EDA) to identify key patterns and relationships.

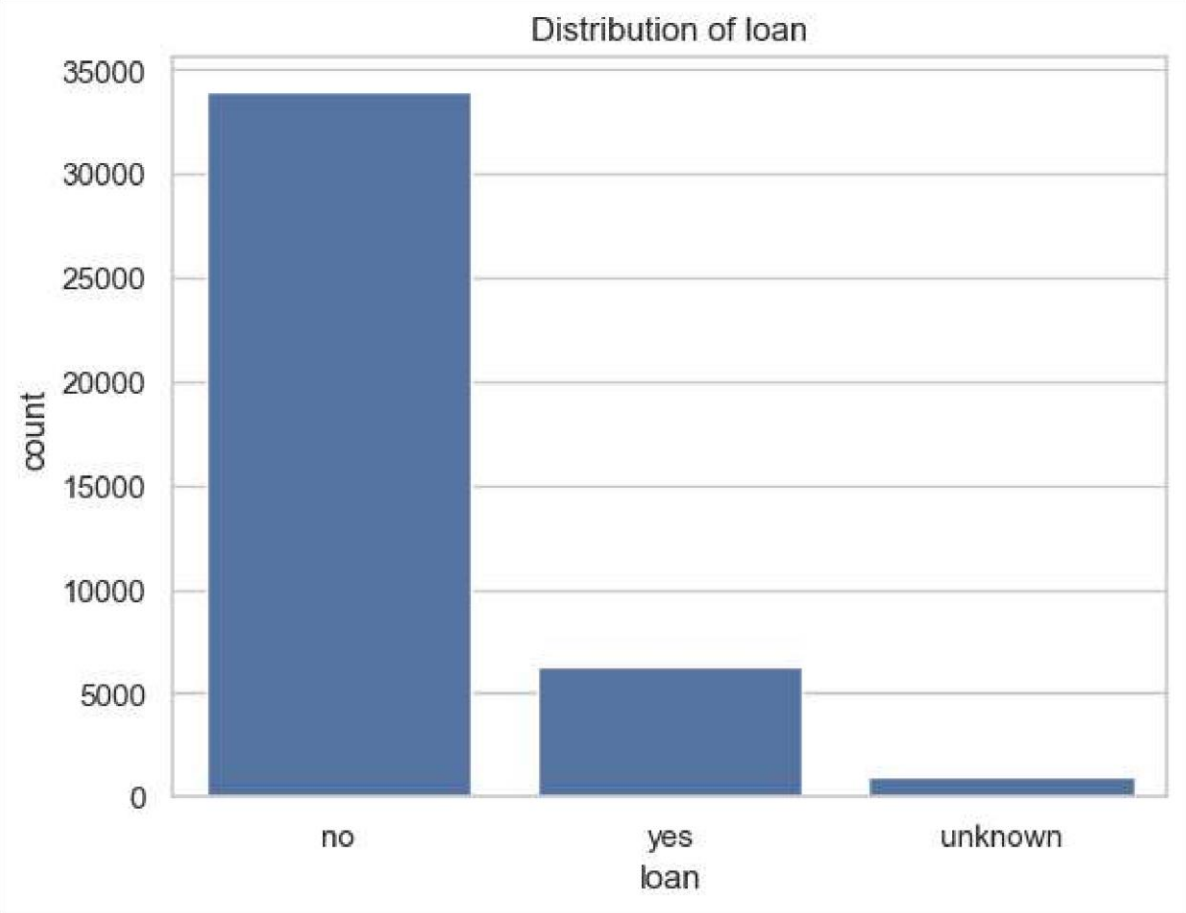
Univariate Analysis (Individual Features)

Categorical Columns

In [8]:
```
# 'age' column
sns.countplot(data=df, x='age')
plt.title('Age Distribution')
plt.show()
```

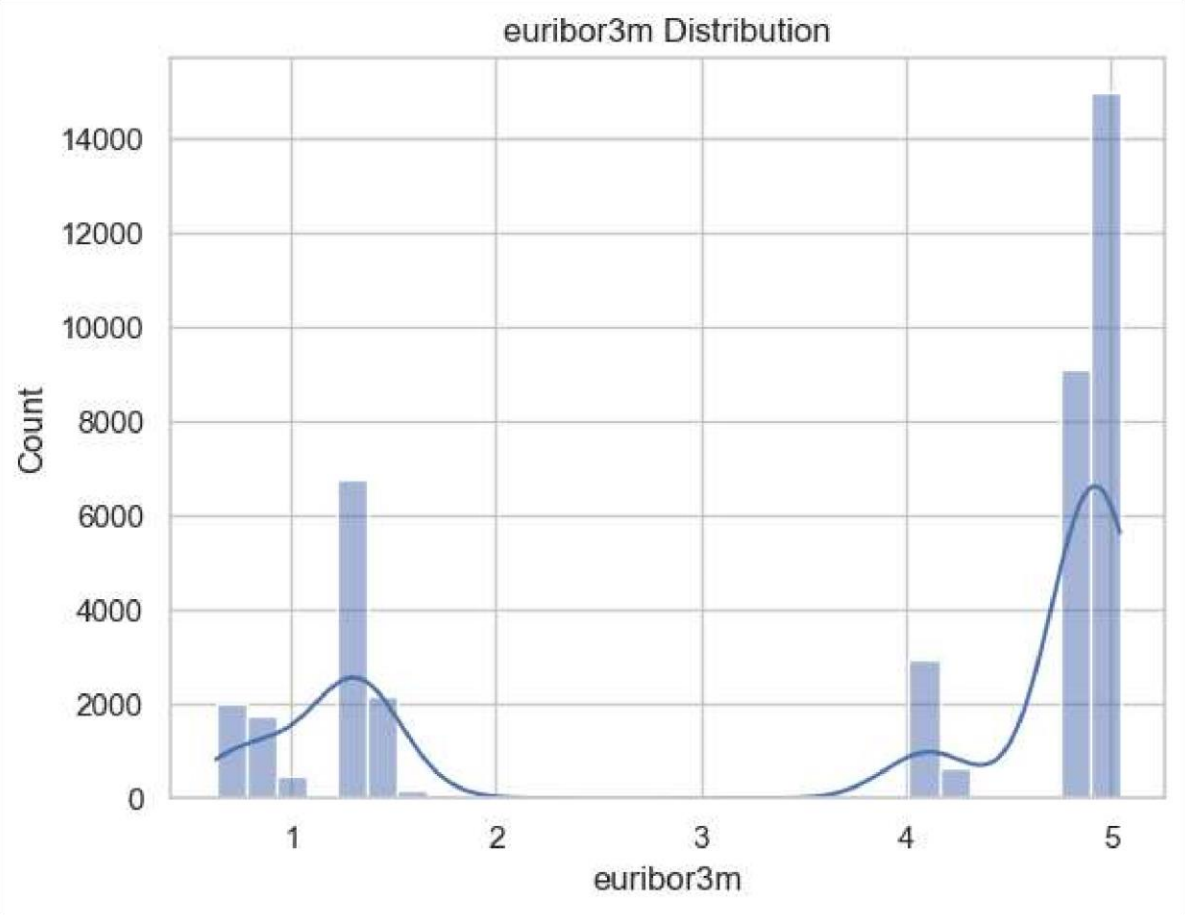


In [10]:
```
#  'Loan' column
sns.countplot(data=df, x='loan')
plt.title('Distribution of loan')
plt.show()
```

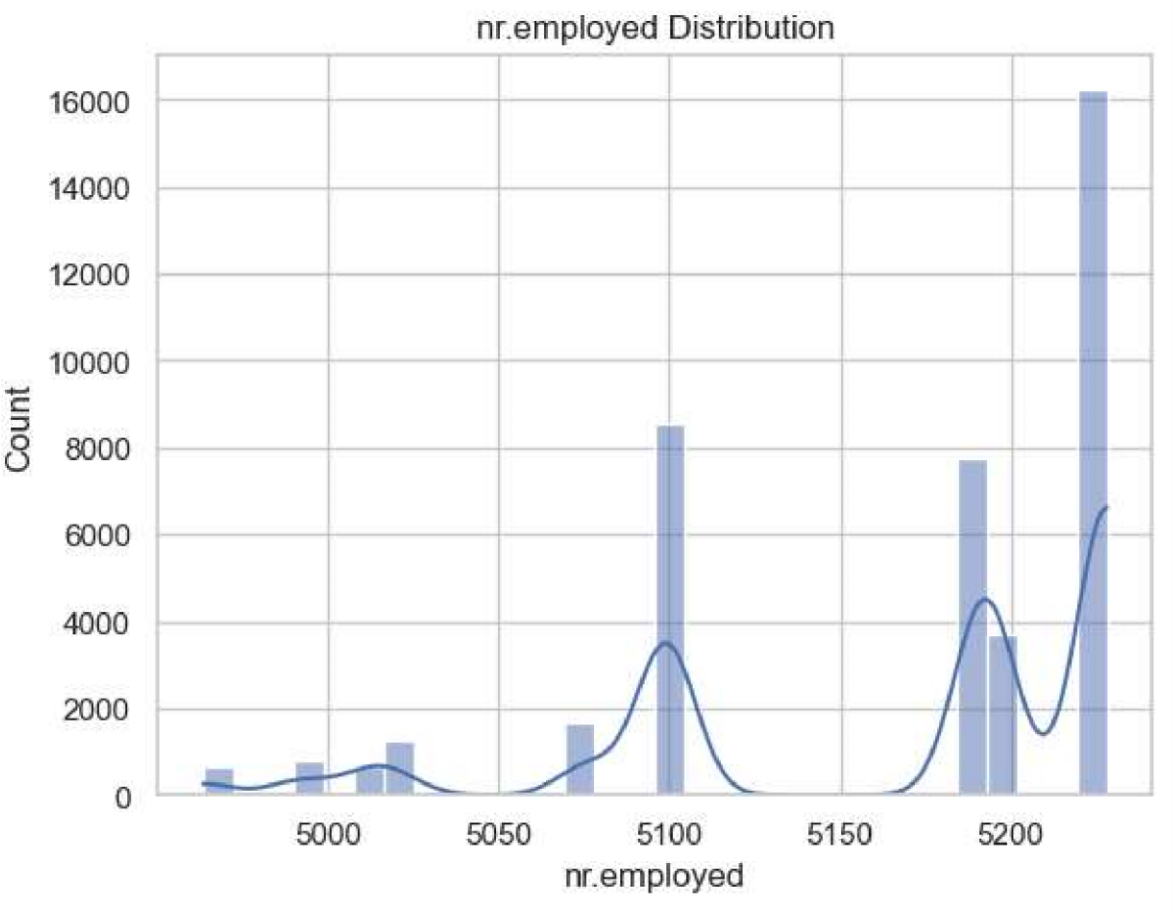


Numerical Columns

In [13]:
```
# Example: 'euribor3m' column
sns.histplot(df['euribor3m'], kde=True, bins=30)
plt.title('euribor3m Distribution')
plt.show()
```



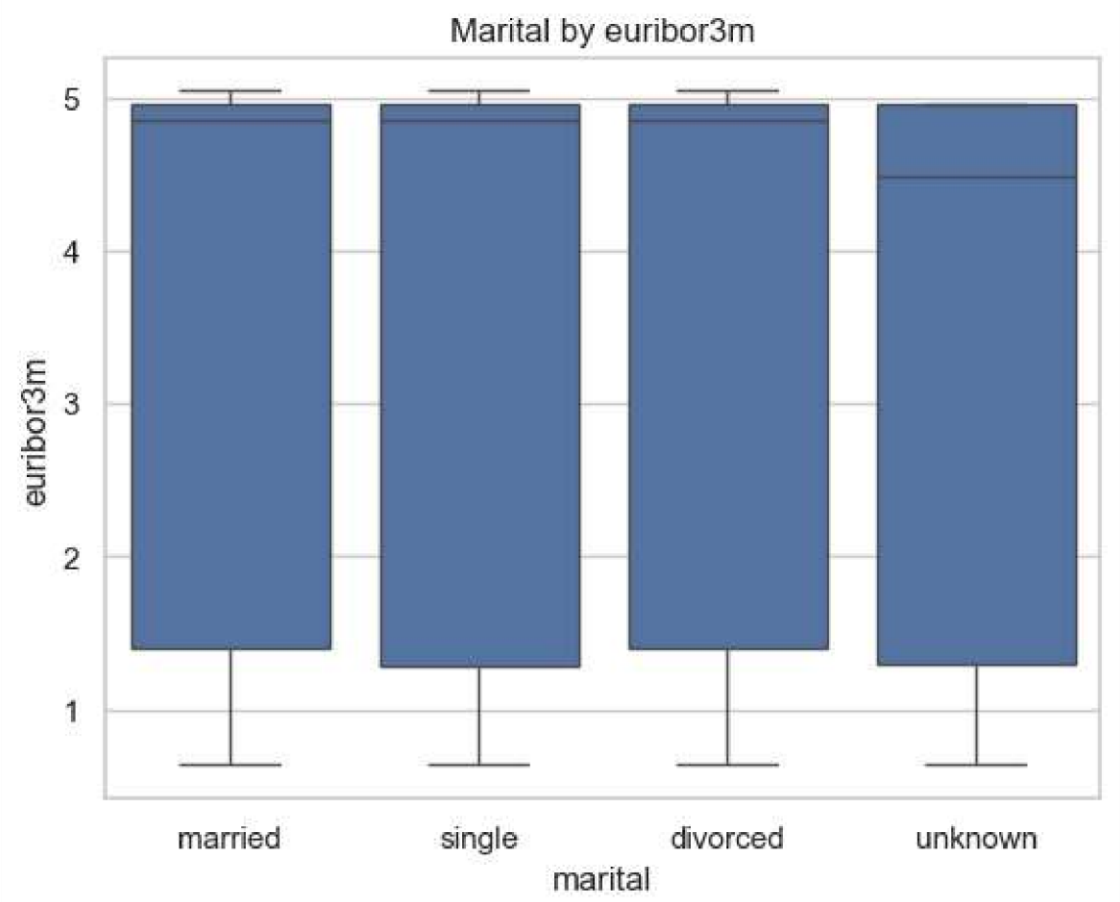
In [14]:
```
# Example: 'nr.employed' column
sns.histplot(df['nr.employed'], kde=True, bins=30)
plt.title('nr.employed Distribution')
plt.show()
```



Bivariate Analysis (Relationships)

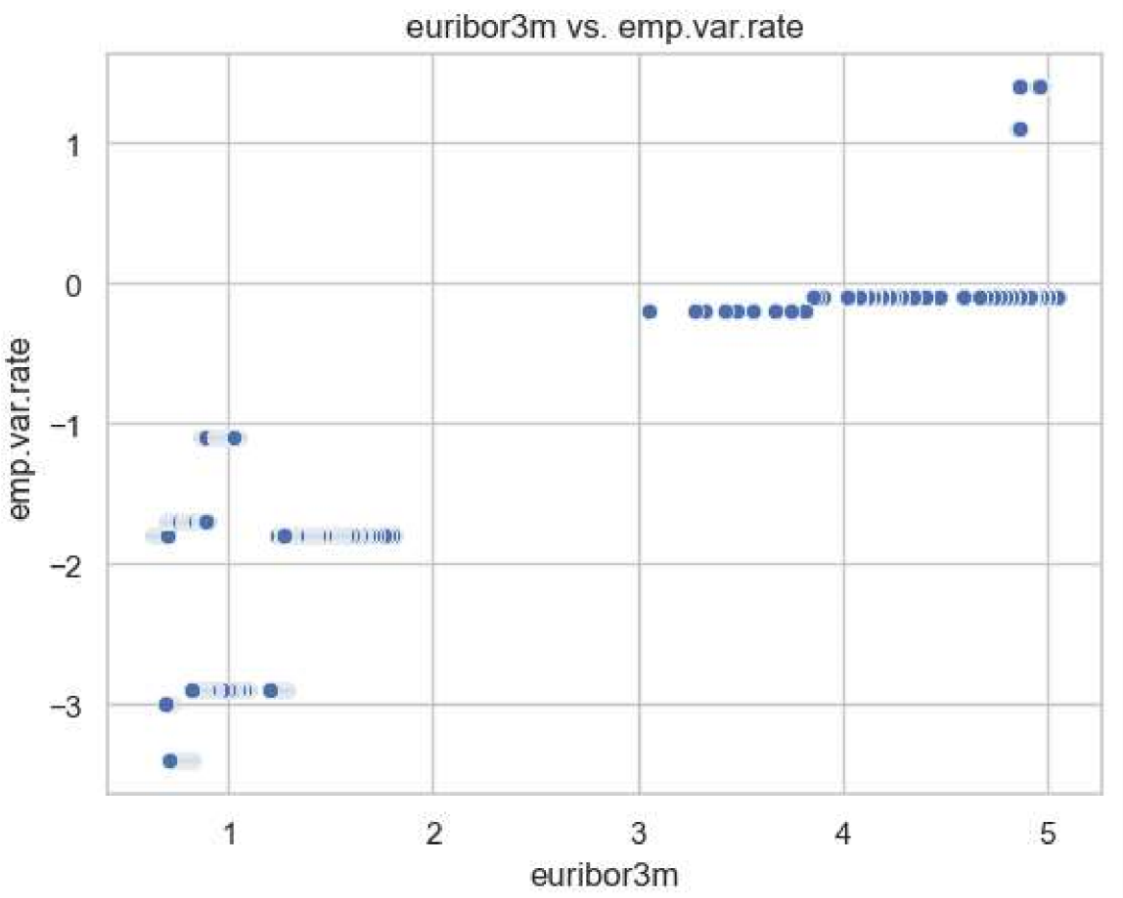
Categorical vs. Numerical

In [15]:
```
# Box plot: Marital distribution across euribor3m
sns.boxplot(x='marital', y='euribor3m', data=df)
plt.title('Marital by euribor3m')
plt.show()
```



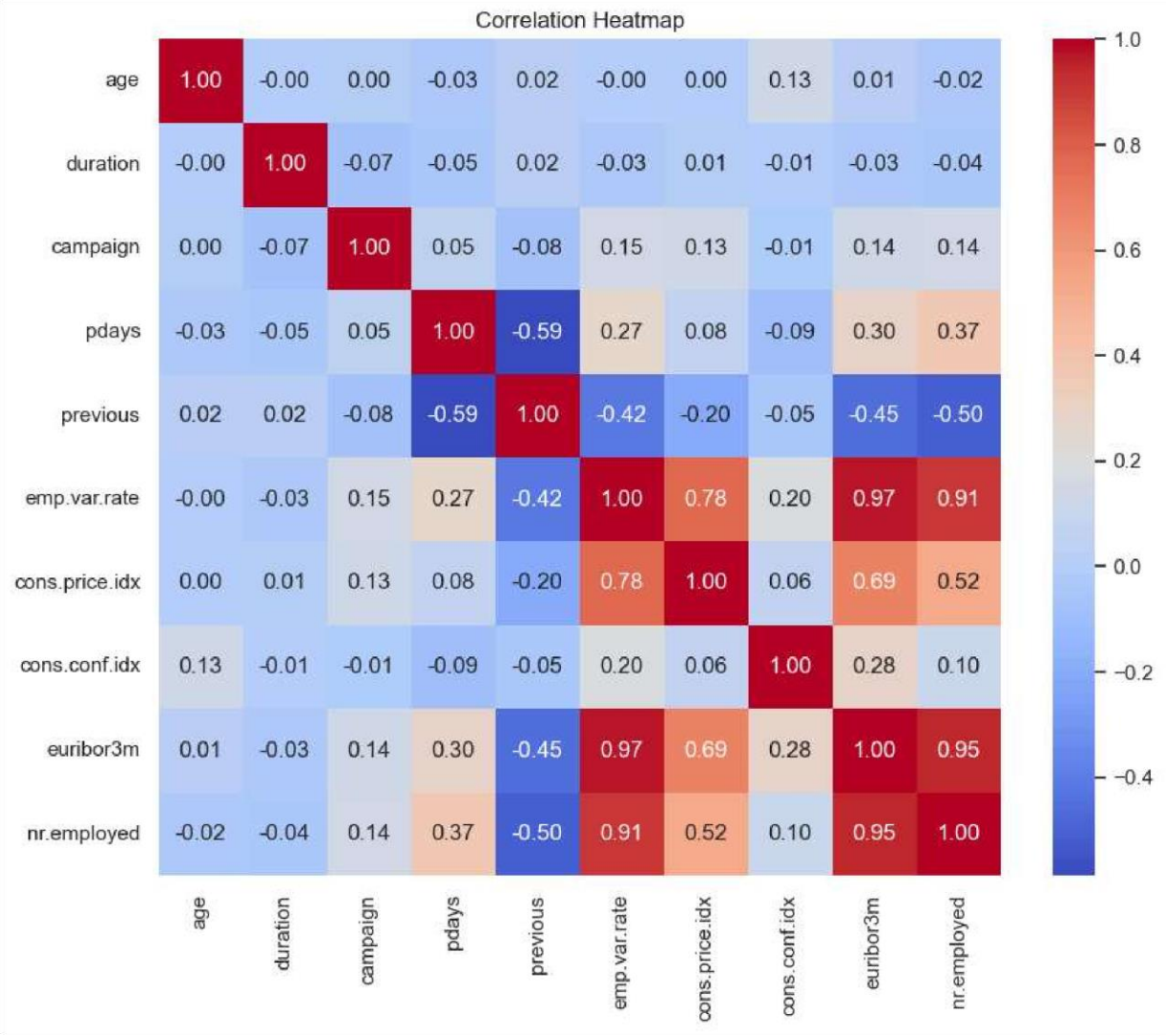
Numerical vs. Numerical

In [19]:
```
# Scatter plot between euribor3m and emp.var.rate
sns.scatterplot(x='euribor3m', y='emp.var.rate', data=df)
plt.title('euribor3m vs. emp.var.rate')
plt.show()
```



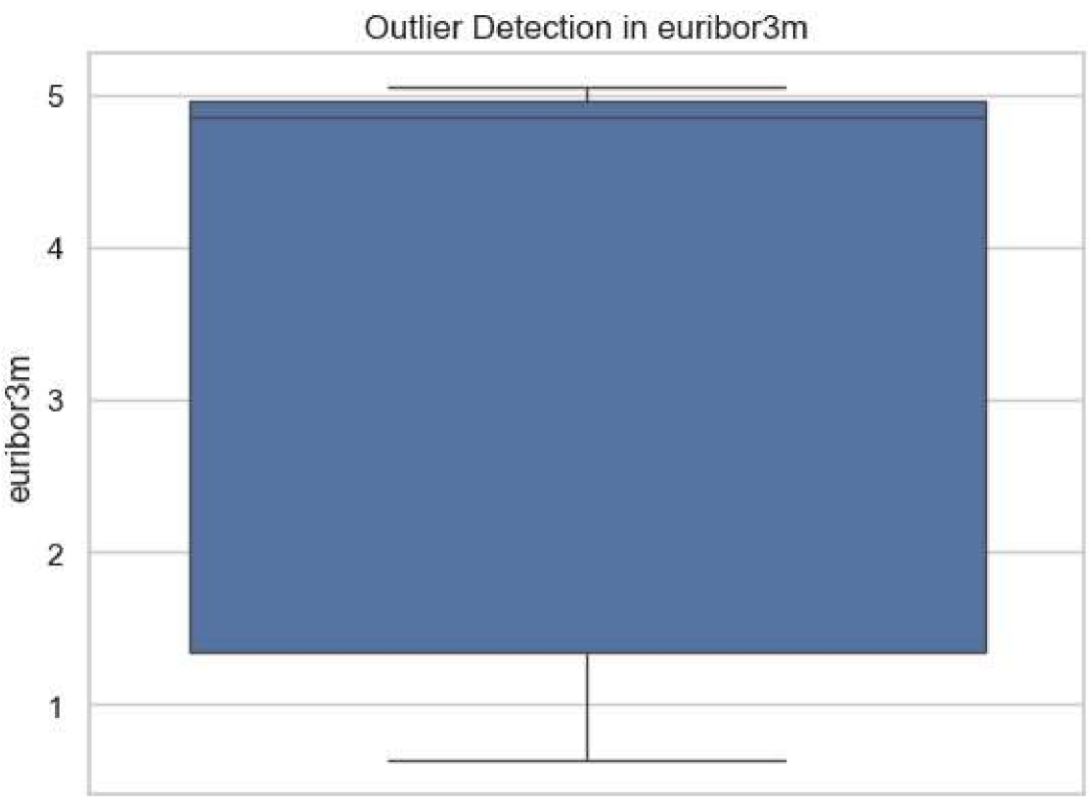
Correlation Analysis

In [22]:
```
# Correlation matrix
corr_matrix = df.corr(numeric_only=True)

# Heatmap
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

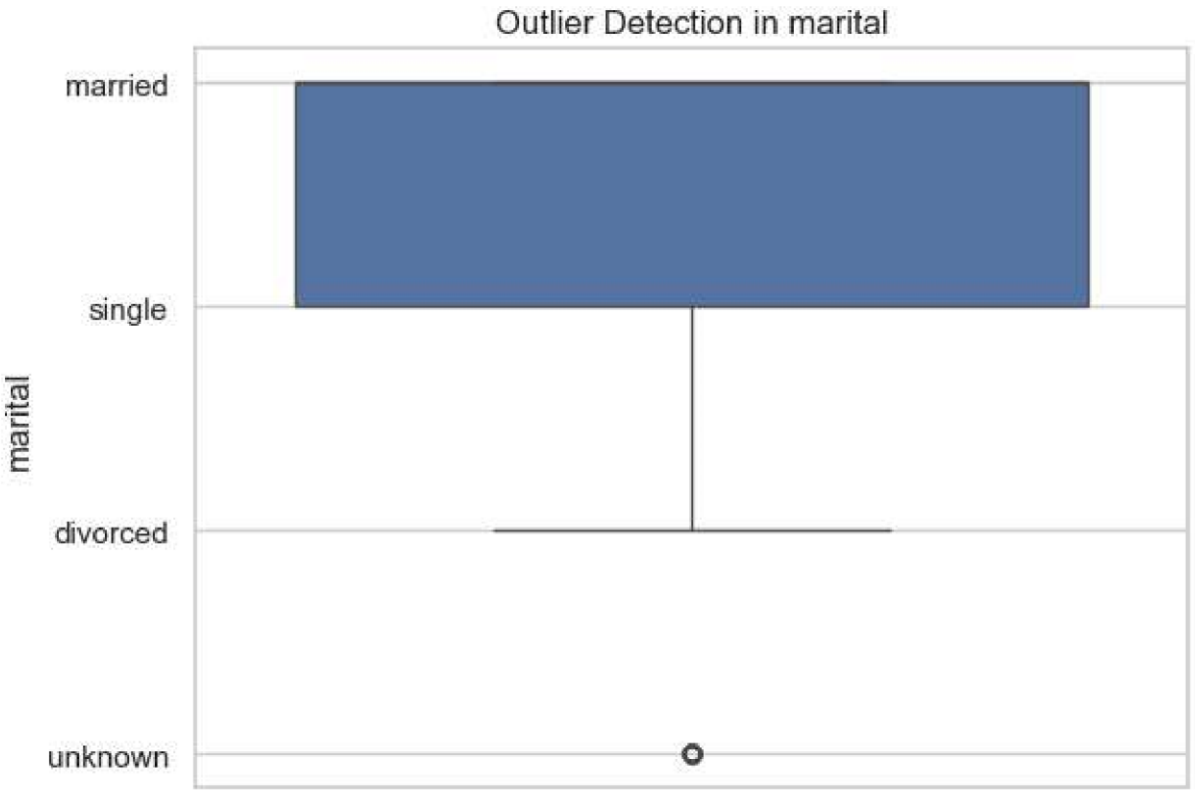


Identify Outliers

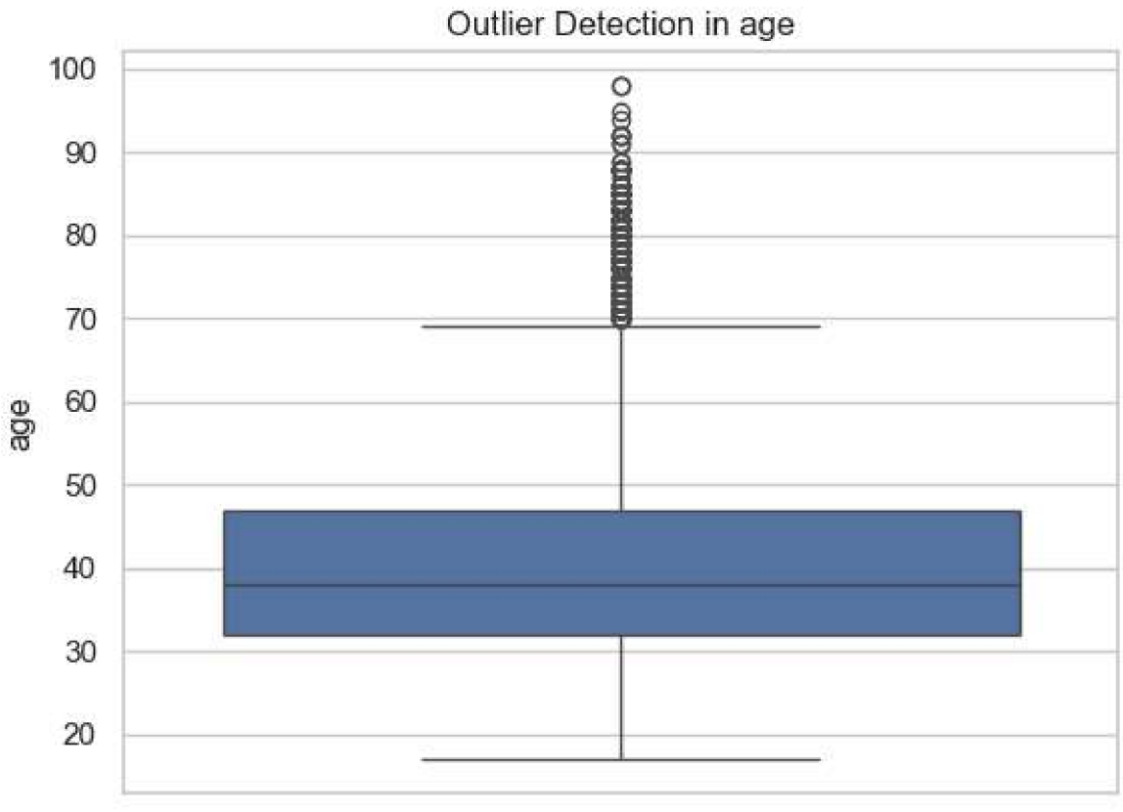
In [24]:
```
# Boxplot to detect outliers
sns.boxplot(data=df['euribor3m'])
plt.title('Outlier Detection in euribor3m')
plt.show()
```



In [25]:
```
# Boxplot to detect outliers
sns.boxplot(data=df['marital'])
plt.title('Outlier Detection in marital')
plt.show()
```



In [26]:
```
# Boxplot to detect outliers
sns.boxplot(data=df['age'])
plt.title('Outlier Detection in age')
plt.show()
```



4. Encoded categorical variables and normalized numerical features.

In [27]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinM
```

Identify Categorical and Numerical Columns

In [28]:
```
# Separate column types
categorical_cols = df.select_dtypes(include='object').columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns

print("Categorical:", list(categorical_cols))
print("Numerical:", list(numerical_cols))
```

```
Categorical: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contac
t', 'month', 'day_of_week', 'poutcome', 'y']
Numerical: ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'con
s.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']
```

Encode Categorical Variables

One-Hot Encoding (best for non-ordinal categorical data)

In [38]:
```
categorical_cols = df.select_dtypes(include='object').columns
print("Categorical columns:", list(categorical_cols))
for col in categorical_cols:
    print(f"{col}: {df[col].unique()[:5]}")  # Show first 5 unique values
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

```
Categorical columns: []
```

Normalize Numerical Features

Use StandardScaler (Z-score normalization)

In [42]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_encoded[numerical_cols] = scaler.fit_transform(df_encoded[numerical_cols])
```

Check Final Output

In [43]:
```
df_encoded.head()
```

Out[43]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.c
0	1.533034	0.010471	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.
1	1.628993	-0.421501	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.
2	-0.290186	-0.124520	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.
3	-0.002309	-0.413787	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.
4	1.533034	0.187888	-0.565922	0.195414	-0.349494	0.648092	0.722722	0.

5 rows × 54 columns

5. Split the data into training and testing sets.

In [79]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the data
df = pd.read_csv('bankmarketing.csv')

# Separate features and target
X = df.drop('y', axis=1)
y = df['y']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# Confirm the split
print("✅ Split successful!")
print("Training shape:", X_train.shape)
print("Testing shape:", X_test.shape)
print("Target distribution in training set:")
print(y_train.value_counts(normalize=True))
print("Target distribution in test set:")
print(y_test.value_counts(normalize=True))
```

```
✅ Split successful!
Training shape: (32950, 20)
Testing shape: (8238, 20)
Target distribution in training set:
y
no     0.887344
yes    0.112656
Name: proportion, dtype: float64
Target distribution in test set:
y
no     0.887351
yes    0.112649
Name: proportion, dtype: float64
```

6. Trained and evaluated multiple classification models (e.g., Logistic Regression, Decision Tree, Random Forest).

In [98]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


logreg = LogisticRegression(max_iter=2000)  # increased max_iter
tree = DecisionTreeClassifier(random_state=42)
forest = RandomForestClassifier(random_state=42)

logreg.fit(X_train_scaled, y_train)
tree.fit(X_train, y_train)     # Tree models don't need scaling
forest.fit(X_train, y_train)


models = {
    "Logistic Regression": (logreg, X_test_scaled),
    "Decision Tree": (tree, X_test),
    "Random Forest": (forest, X_test)
}

for name, (model, X_t) in models.items():
    print(f"\nModel: {name}")
    y_pred = model.predict(X_t)
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```
Model: Logistic Regression
[[7145  165]
 [ 524  404]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      7310
           1       0.71      0.44      0.54       928

    accuracy                           0.92      8238
   macro avg       0.82      0.71      0.75      8238
weighted avg       0.91      0.92      0.91      8238


Model: Decision Tree
[[6864  446]
 [ 424  504]]
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      7310
           1       0.53      0.54      0.54       928

    accuracy                           0.89      8238
   macro avg       0.74      0.74      0.74      8238
weighted avg       0.90      0.89      0.89      8238


Model: Random Forest
[[7088  222]
 [ 476  452]]
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      7310
           1       0.67      0.49      0.56       928

    accuracy                           0.92      8238
   macro avg       0.80      0.73      0.76      8238
weighted avg       0.91      0.92      0.91      8238
```

7. Measured model performance using accuracy, precision, recall, and F1-score.

In [100…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Function to evaluate any classification model
def evaluate_model(name, model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"📊 Performance of {name}:")
```

0

```
    print(f"✅ Accuracy:  {accuracy:.4f}")
    print(f"✅ Precision: {precision:.4f}")
    print(f"✅ Recall:    {recall:.4f}")
    print(f"✅ F1 Score:  {f1:.4f}")
    print("-" * 40)


evaluate_model("Logistic Regression", logreg, X_test_scaled, y_test)
evaluate_model("Decision Tree", tree, X_test, y_test)
evaluate_model("Random Forest", forest, X_test, y_test)
```

```
📊 Performance of Logistic Regression:
✅ Accuracy:  0.9164
✅ Precision: 0.7100
✅ Recall:    0.4353
✅ F1 Score:  0.5397
----------------------------------------
📊 Performance of Decision Tree:
✅ Accuracy:  0.8944
✅ Precision: 0.5305
✅ Recall:    0.5431
✅ F1 Score:  0.5367
----------------------------------------
📊 Performance of Random Forest:
✅ Accuracy:  0.9153
✅ Precision: 0.6706
✅ Recall:    0.4871
✅ F1 Score:  0.5643
----------------------------------------
```

What We Learned

Variables such as contact, poutcome, month, and duration were among the most significant predictors for term deposit subscription.

The marketing campaign was more effective during certain months, particularly around May and August.

A longer call duration was correlated with a higher likelihood of subscription. Ensemble models like Random Forest generally performed better in predictive accuracy.

In [ ]: