# Full code-Capston Project - Marketing campaign

February 18, 2026

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report,␣
↪roc_auc_score, accuracy_score
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import shap
# Python
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

[2]:
```python
df= pd.read_csv("bank-full.csv", sep=";")
df.head()
```

[2]:
```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  duration  campaign  pdays  previous poutcome   y
0  unknown    5   may       261         1     -1         0  unknown  no
1  unknown    5   may       151         1     -1         0  unknown  no
```

```
2  unknown    5   may        76          1      -1           0  unknown  no
3  unknown    5   may        92          1      -1           0  unknown  no
4  unknown    5   may       198          1      -1           0  unknown  no
```

[3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

[ ]:
```python

```

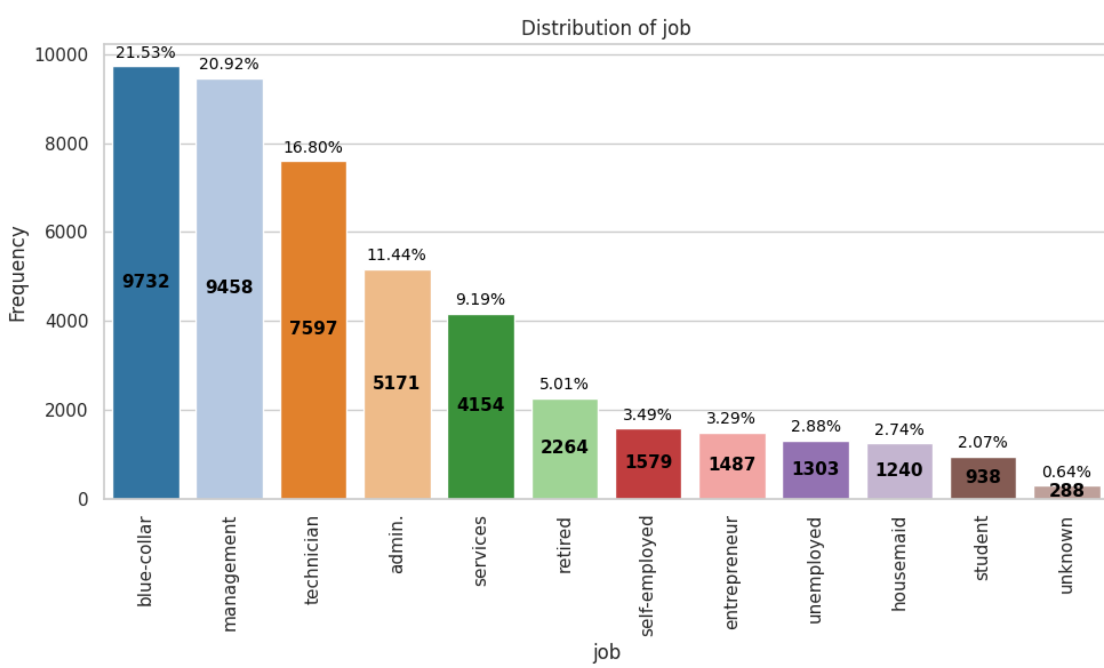
## 0.1 Univariate analysis for categorical and object variables

[4]:
```python
# Univariate analysis of categorical and object variables

def plot_object(dataframe, column_name):
  """
  Plots a bar chart showing category frequencies with both frequency (inside bar)
  and proportion (above bar) labels.
  Parameters:- dataframe: pandas DataFrame- column_name: str, name of the␣
  ↪categorical column to visualize
  """
  # Count frequencies and proportions
  value_counts = dataframe[column_name].value_counts()
  proportions = value_counts / len(dataframe)
```

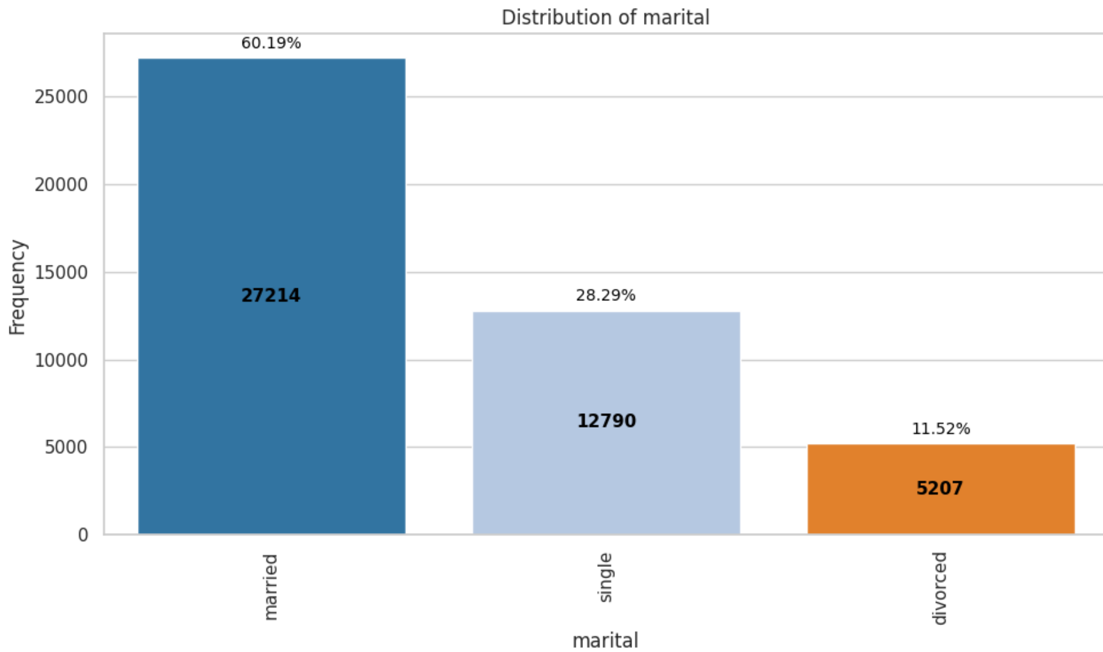
```
# Set plot style
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
# Bar plot
palette1=sns.color_palette(palette='tab20')
ax = sns.barplot(x=value_counts.index, y=value_counts.values, palette=palette1)
# Annotate bars
for i, (count, prop) in enumerate(zip(value_counts.values, proportions.
↪values)):
# Frequency inside bar
  ax.text(i, count * 0.5, f'{count}', ha='center', va='center',fontsize=11,␣
↪color='black', fontweight='bold')
# Proportion above bar
  ax.text(i, count + max(value_counts.values) * 0.02, f'{prop:.
↪2%}',ha='center', fontsize=10, color='black')
plt.title(f'Distribution of {column_name}')
plt.xlabel(column_name)
plt.xticks(rotation=90)
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

[ ]:

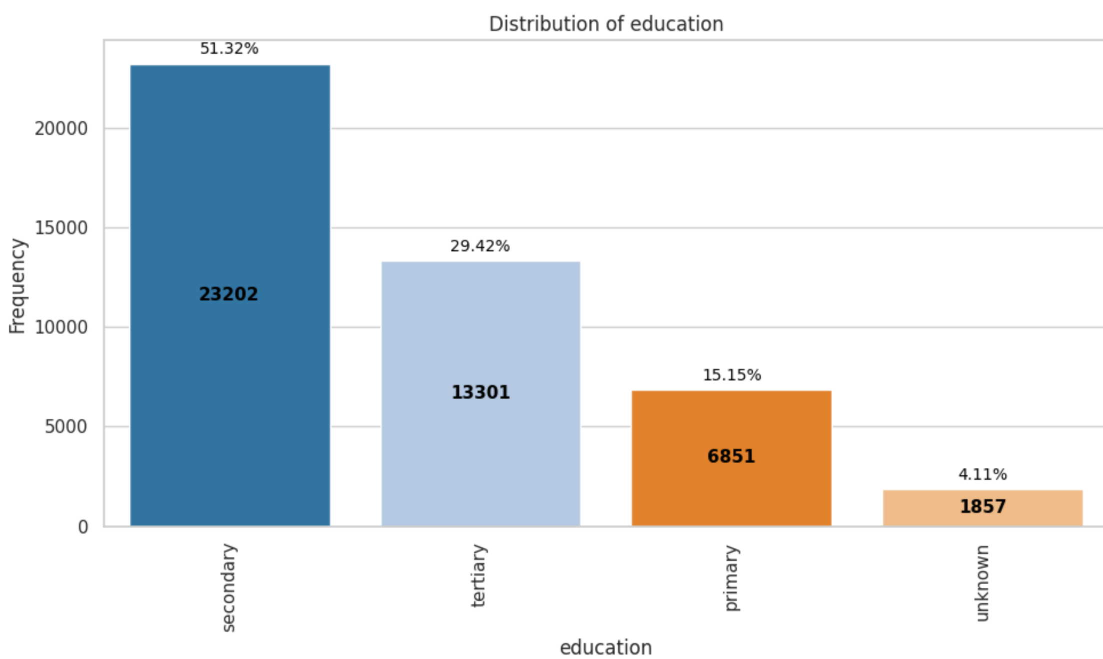
[5]:
```
plot_object(df, "job")
```



```
[6]: plot_object(df, 'marital')
```

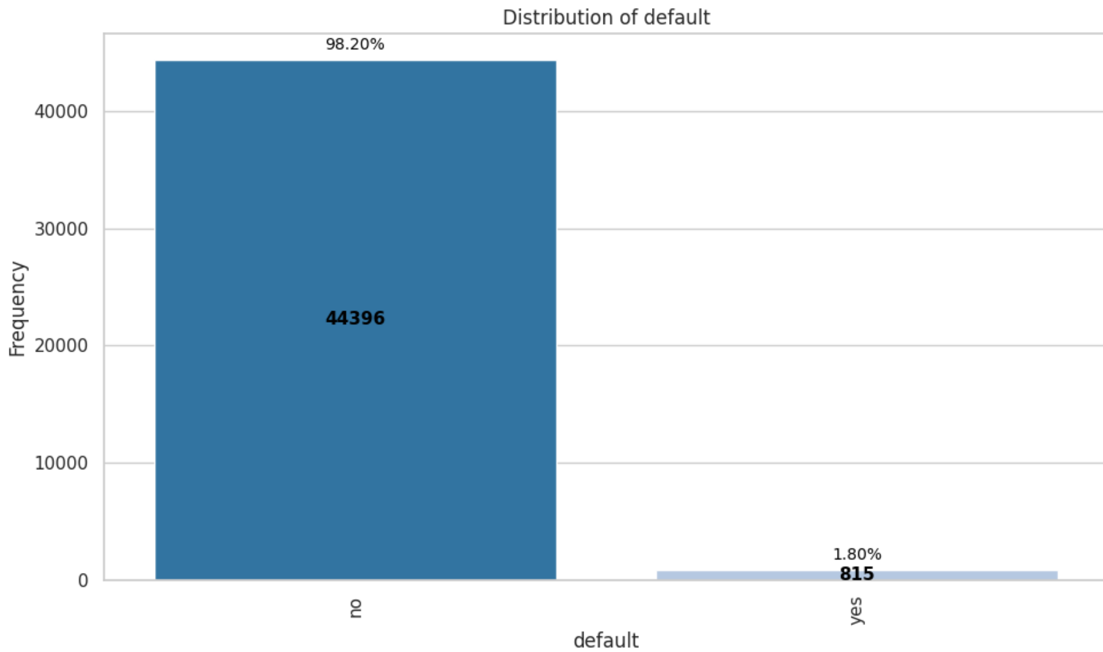



```
[7]: plot_object(df, 'education')
```



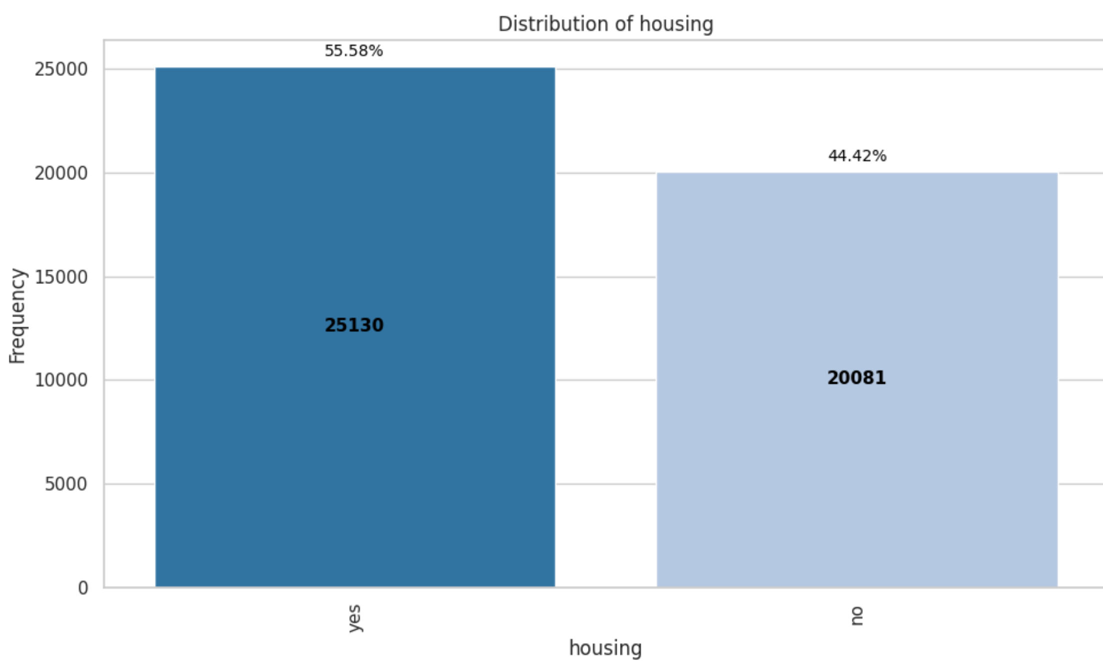
[8]:

```
plot_object(df, 'default')
```

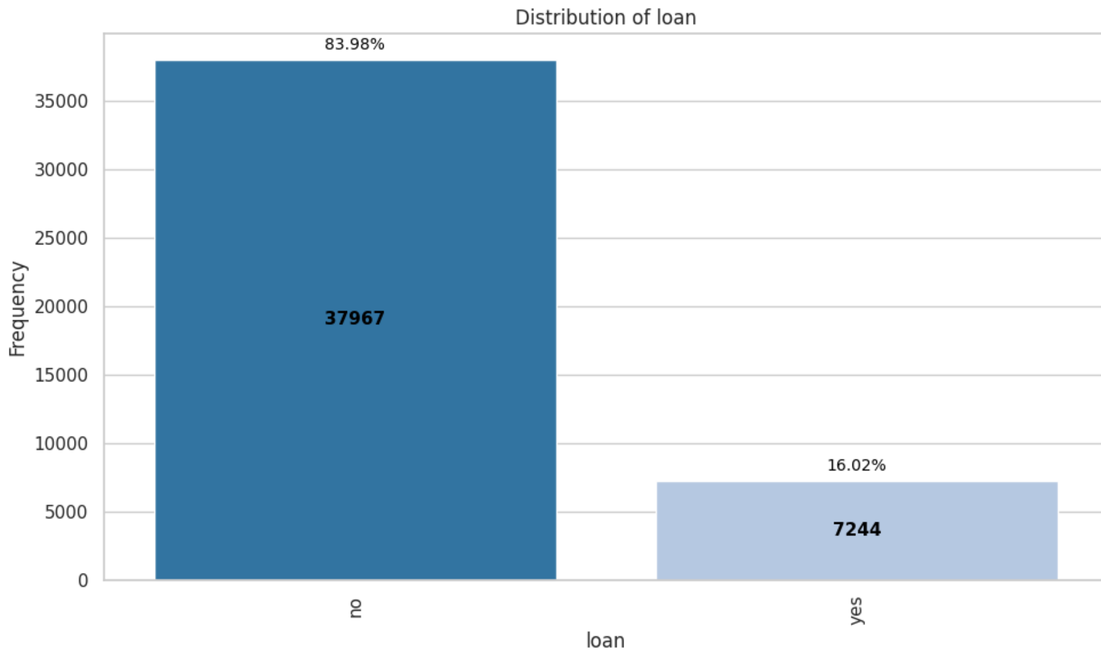



[9]:

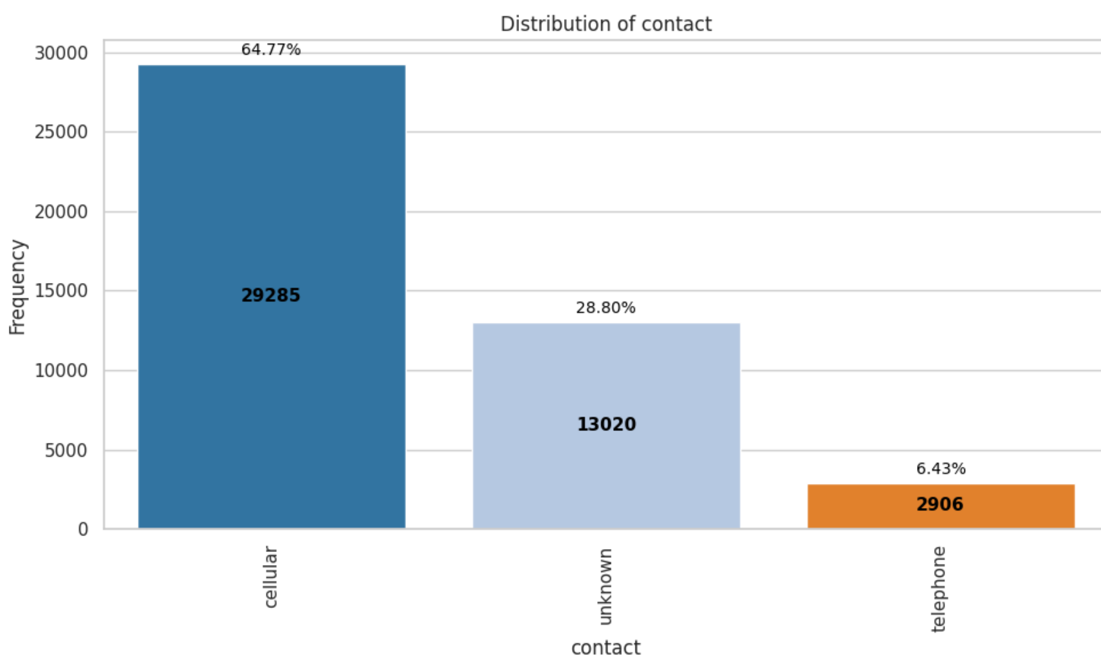
```
plot_object(df, 'housing')
```



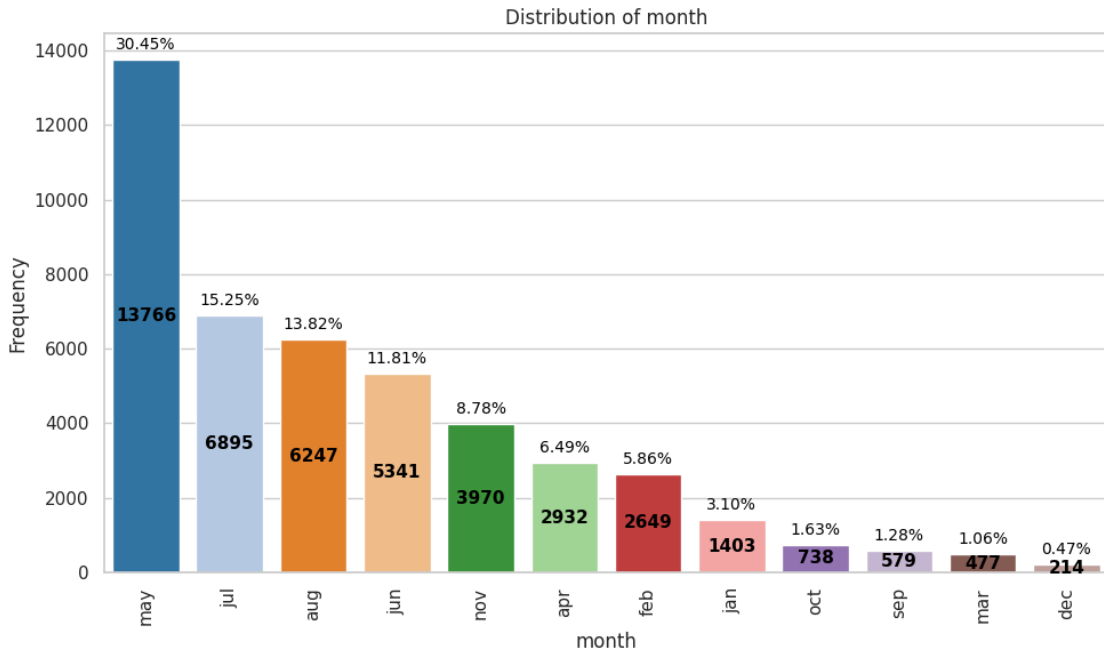
[10]:
```
plot_object(df, 'loan')
```



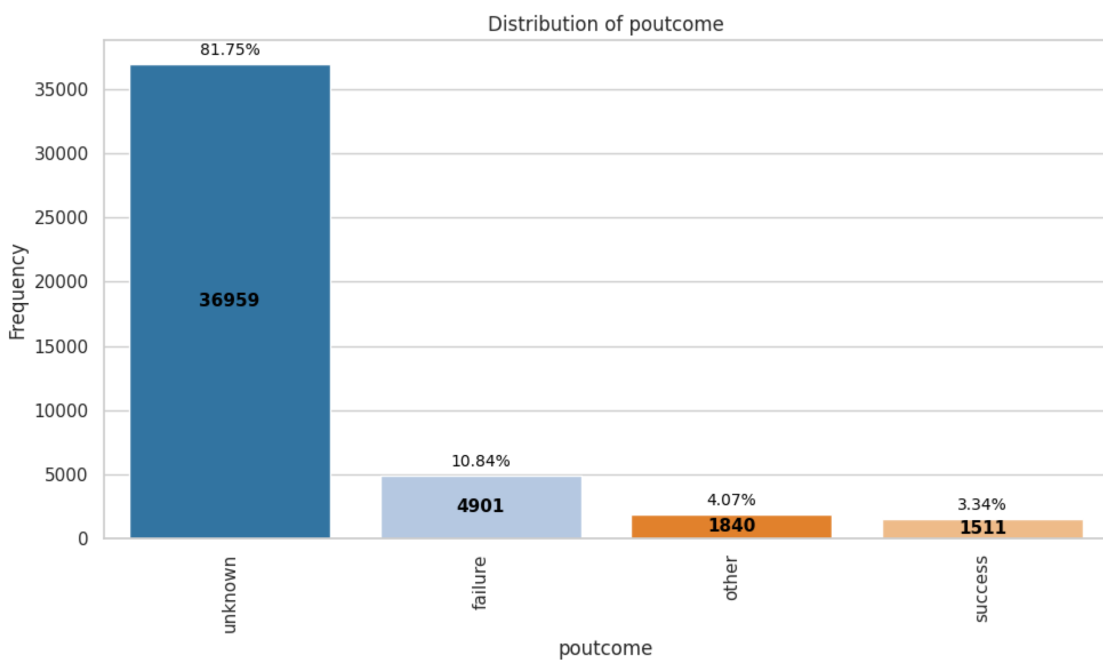

[11]:
```
plot_object(df, 'contact')
```



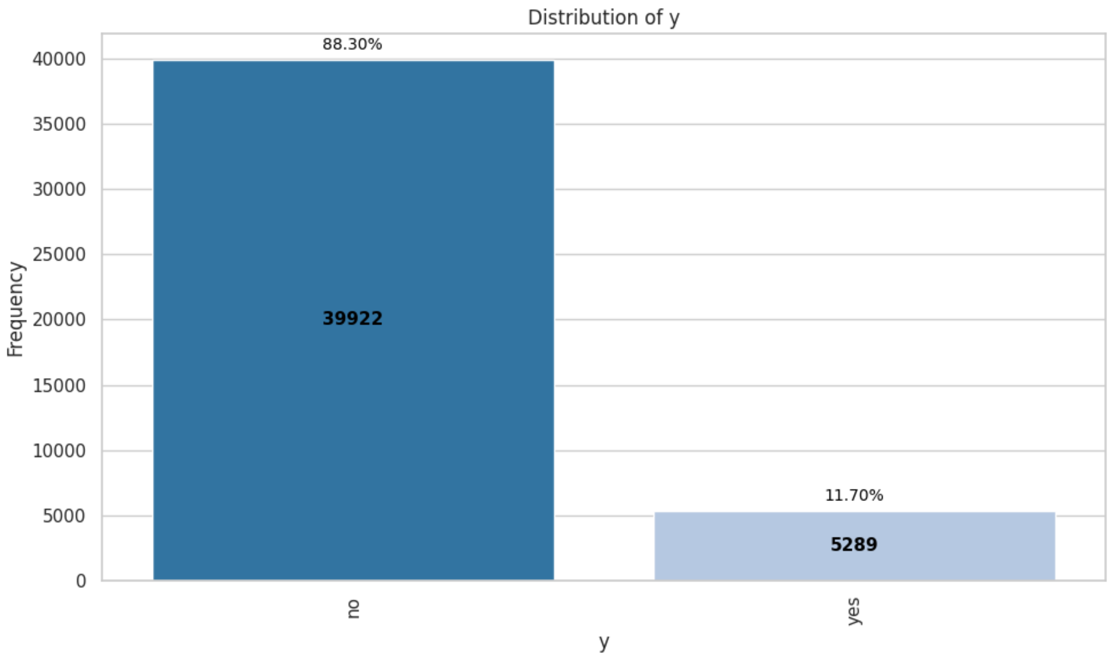
[12]:
```
plot_object(df, 'month')
```




[13]:
```
plot_object(df, 'poutcome')
```



[14]:
```
plot_object(df, 'y')
```



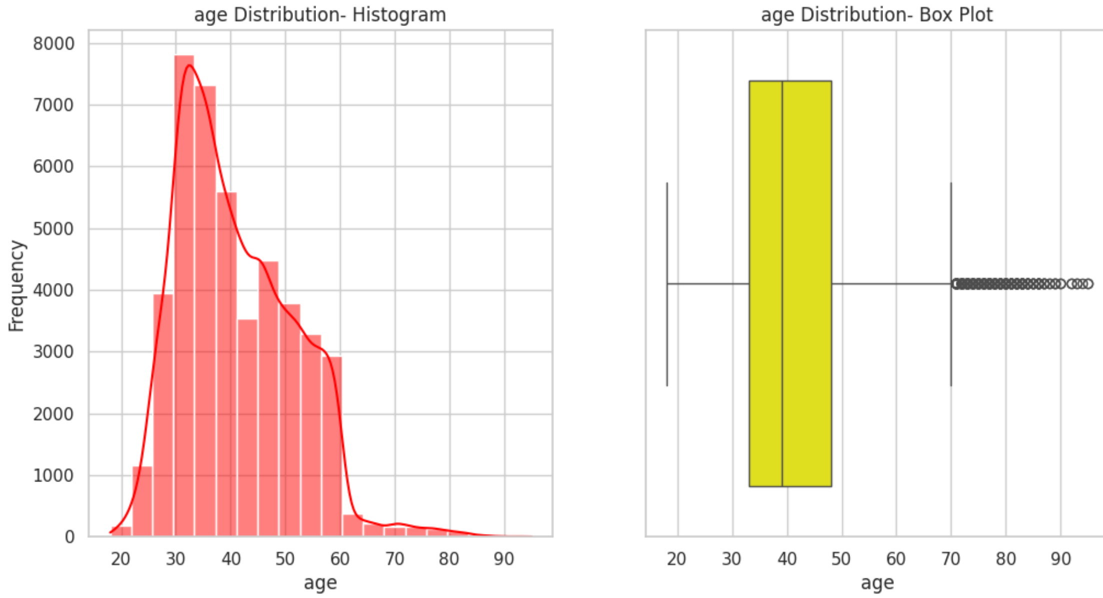

## 0.2 Univariate Analysis for numerical variables

[15]:
```
# univariate analysis of continuous variables
def cont_plot(df, var):
 #var="Age"
 # Set plot style
 sns.set(style="whitegrid")
 # Create a figure with two subplots: histogram and box plot
 plt.figure(figsize=(12, 6))
 # Histogram
 # Box plot
 plt.subplot(1, 2, 1)
 sns.histplot(df[var], bins=20, kde=True, color='red')
 plt.title(var+' Distribution- Histogram')
 plt.xlabel(var)
 plt.ylabel('Frequency')
 plt.subplot(1, 2, 2)
 sns.boxplot(x=df[var], color='yellow')
 plt.title(var+' Distribution- Box Plot')
```

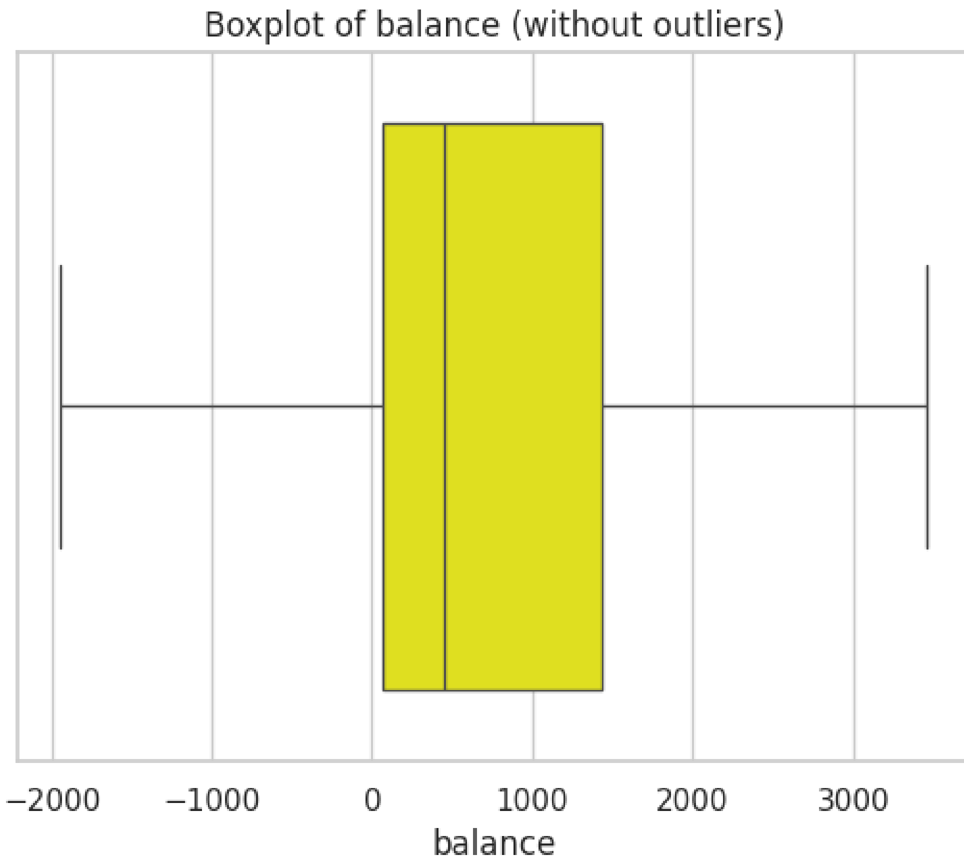
[16]:
```
def NoOutlier(df,var):
    sns.boxplot(x=df[var], showfliers=False,color="yellow")
    plt.title("Boxplot of "+ var +" (without outliers)")
```

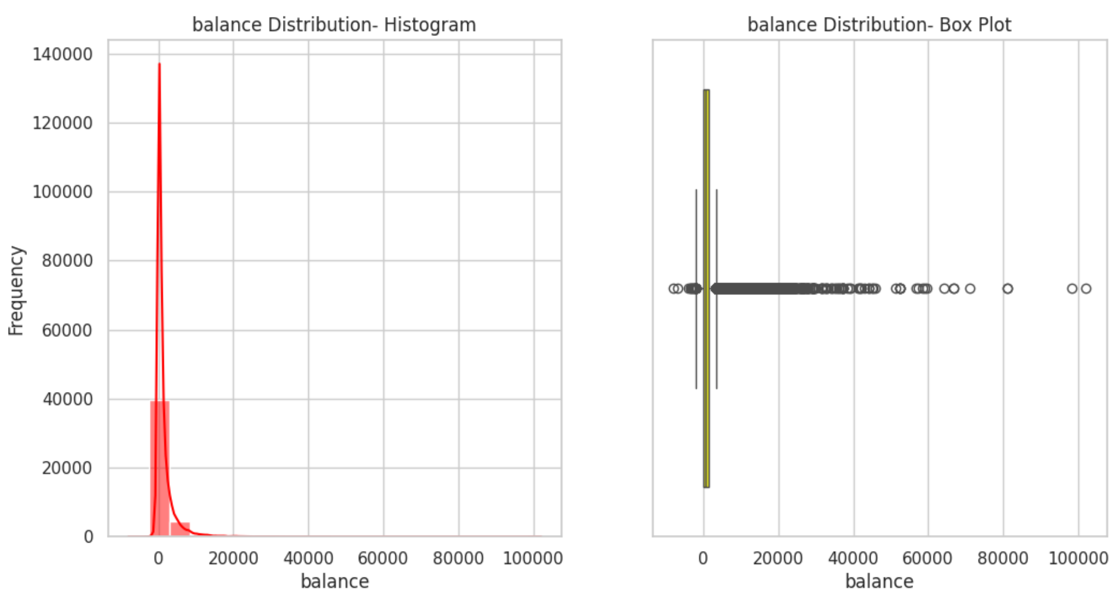
```
    plt.show()
```

[17]:
```
cont_plot(df,'age' )
```

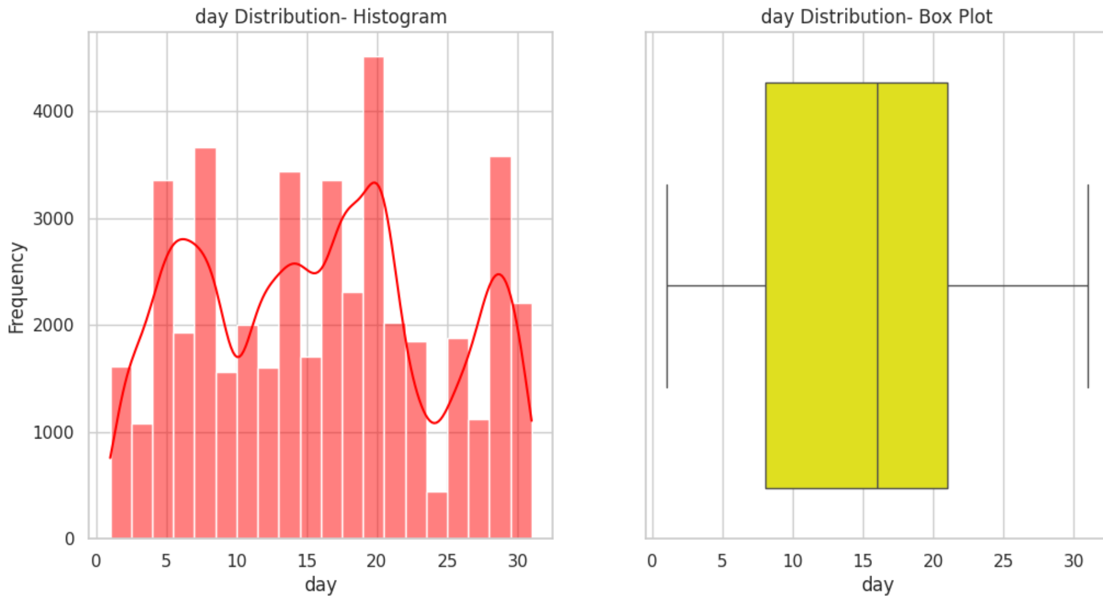



[18]:
```
# Boxplot without outliers
NoOutlier(df,"balance")
cont_plot(df,'balance' )
```

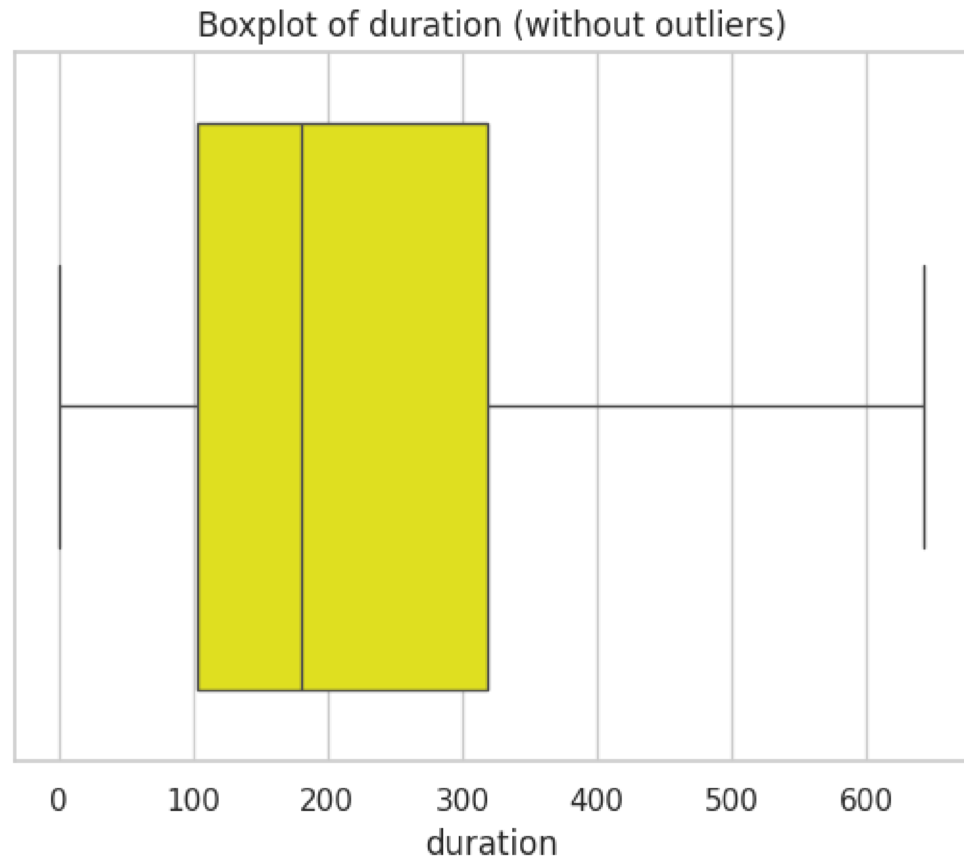

Boxplot of balance (without outliers)
−2000
−1000
0
1000
2000
3000
balance


balance Distribution- Histogram
Frequency
140000
120000
100000
80000
60000
40000
20000
0
0
20000
40000
60000
80000
100000
balance
balance Distribution- Box Plot
0
20000
40000
60000
80000
100000
balance

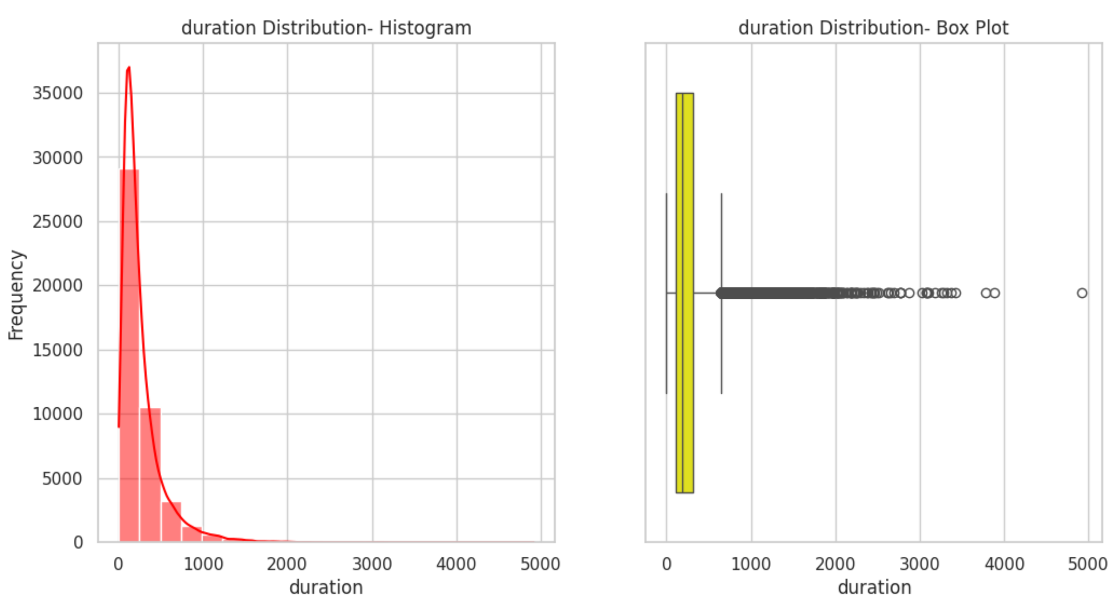
[19]:
```
cont_plot(df,'day' )
```




[20]:
```
NoOutlier(df,"duration")
cont_plot(df,'duration' )
```

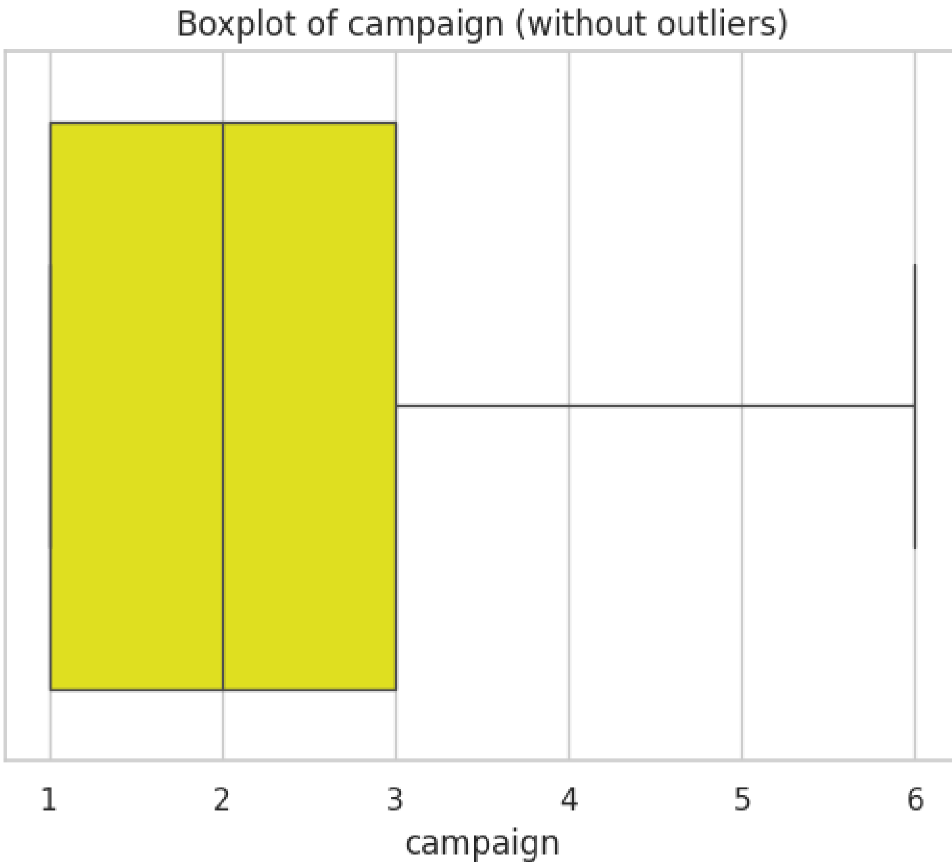


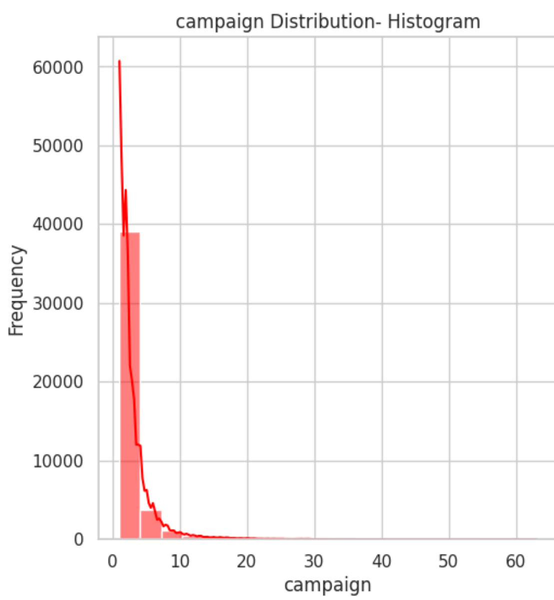
Boxplot of duration (without outliers)
0
100
200
300
400
500
600
duration

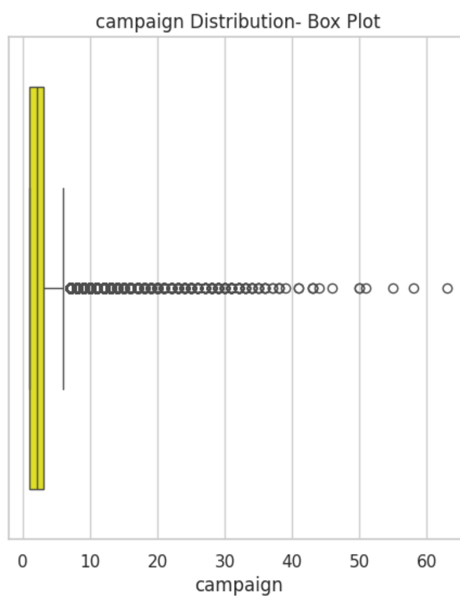



duration Distribution- Histogram
Frequency
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000
duration
duration Distribution- Box Plot
0
1000
2000
3000
4000
5000
duration

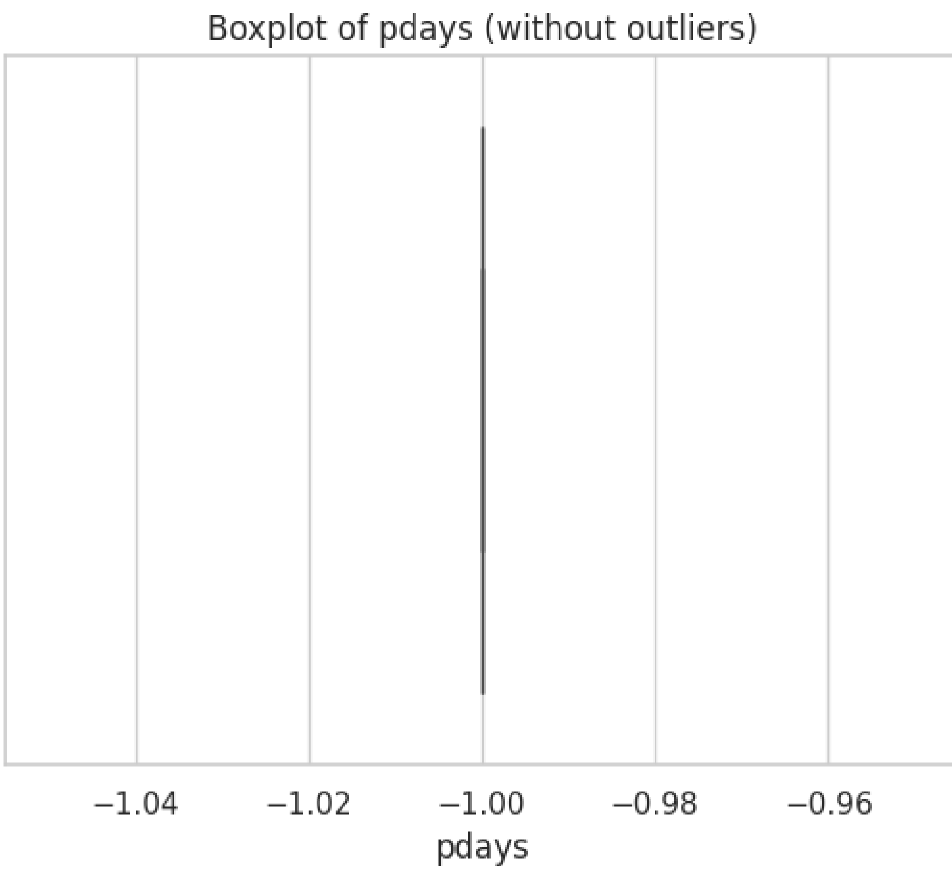
[21]:
```
NoOutlier(df,"campaign")
cont_plot(df,'campaign' )
```

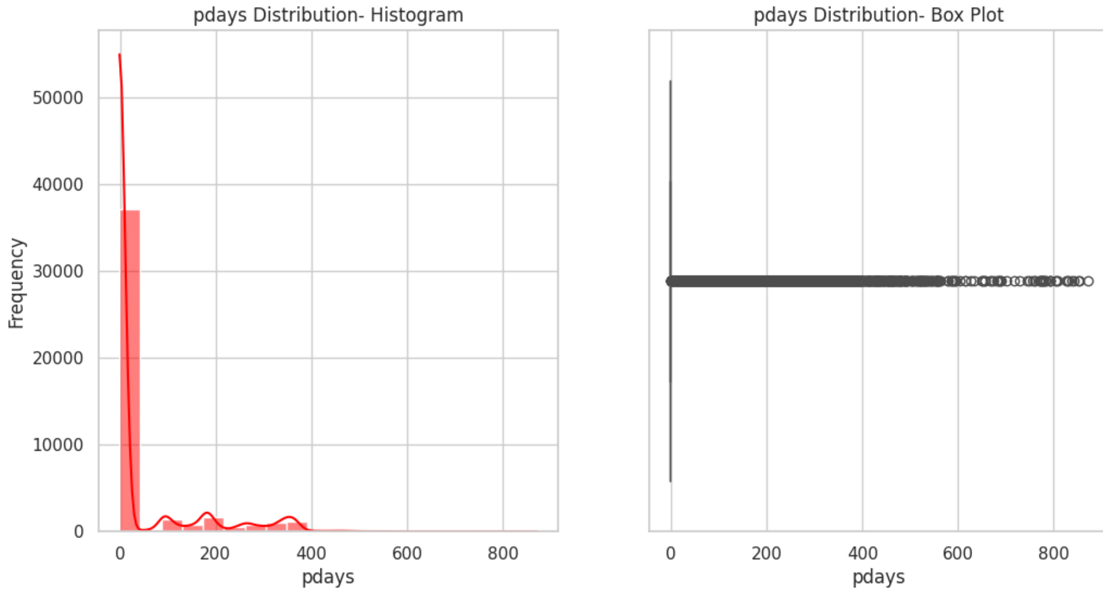


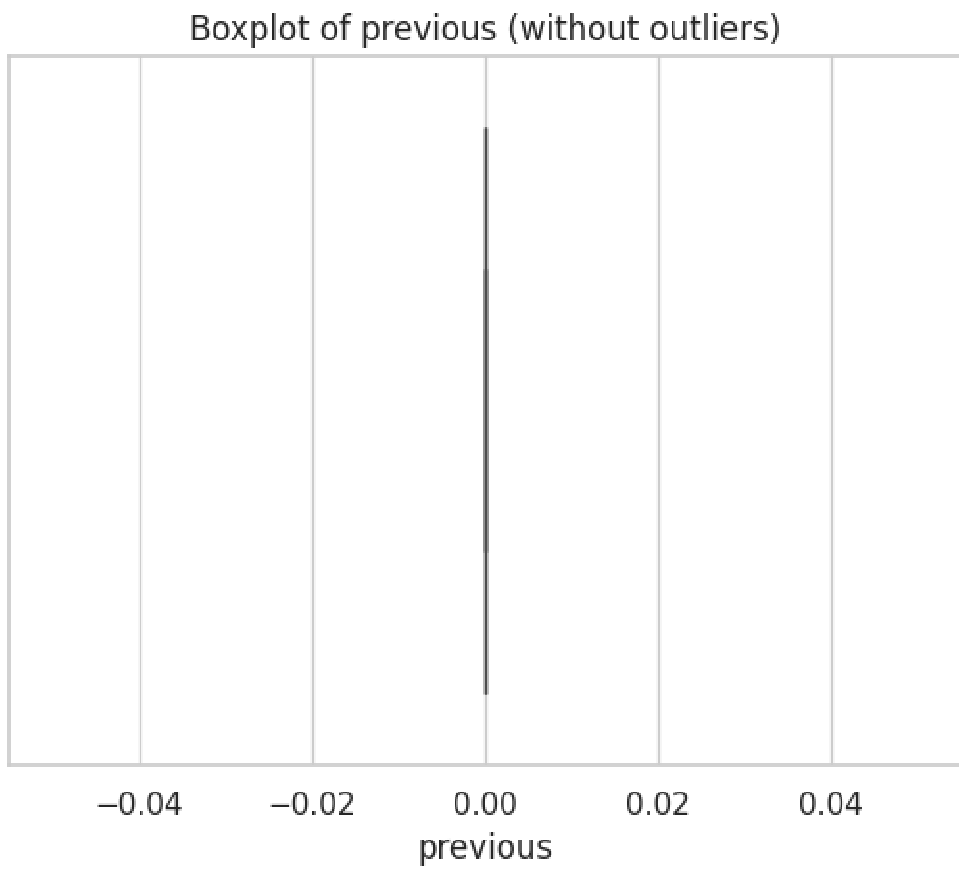



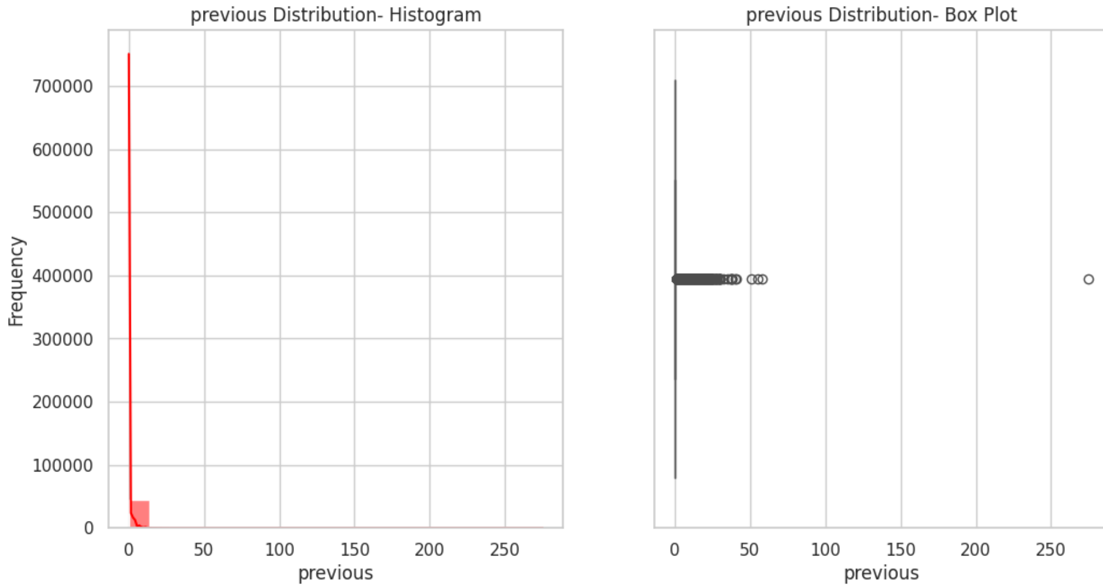
[22]:
```
NoOutlier(df,"pdays")
cont_plot(df,'pdays' )
```



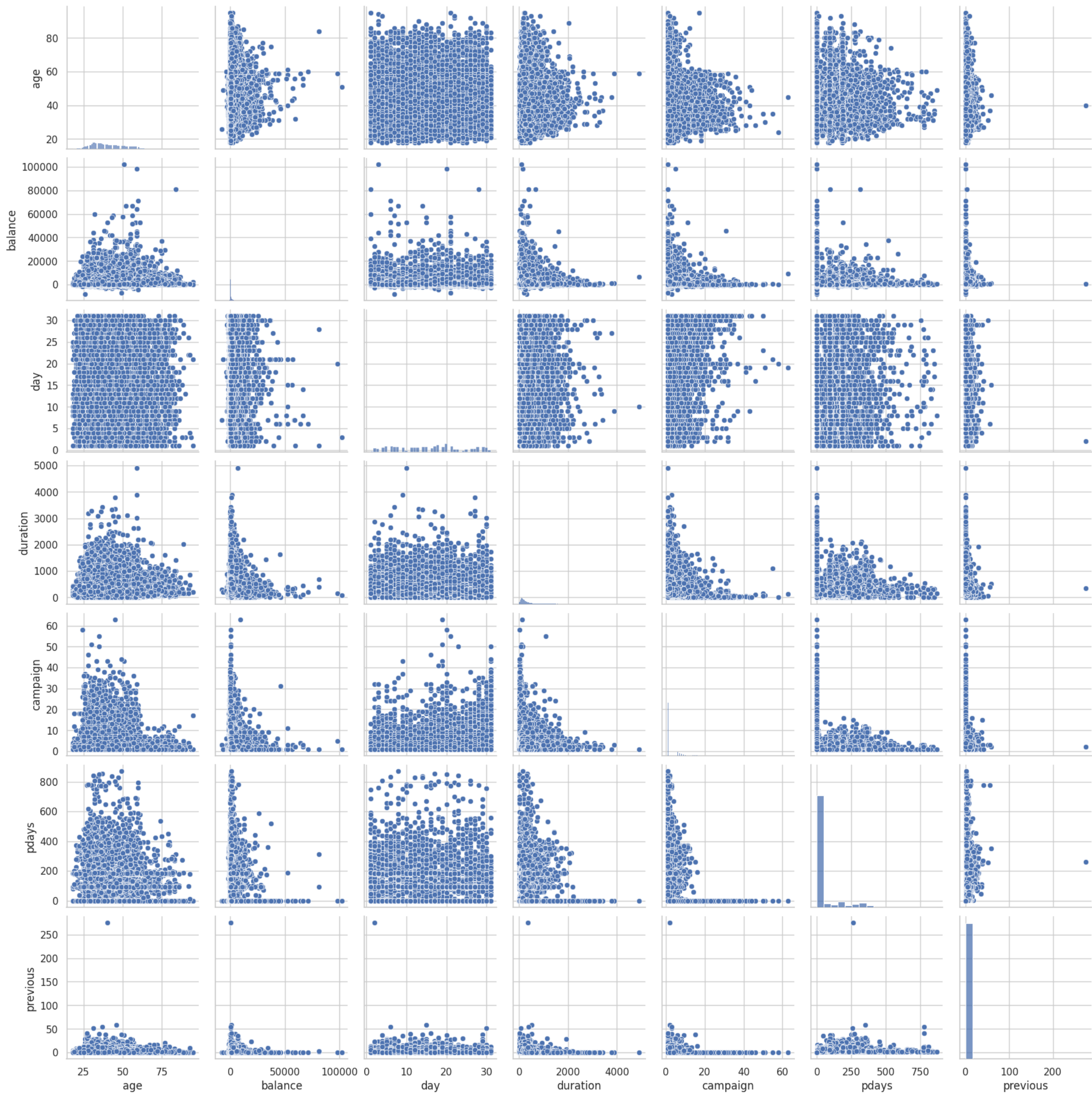



[23]:
```
NoOutlier(df,"previous")
cont_plot(df,'previous' )
```

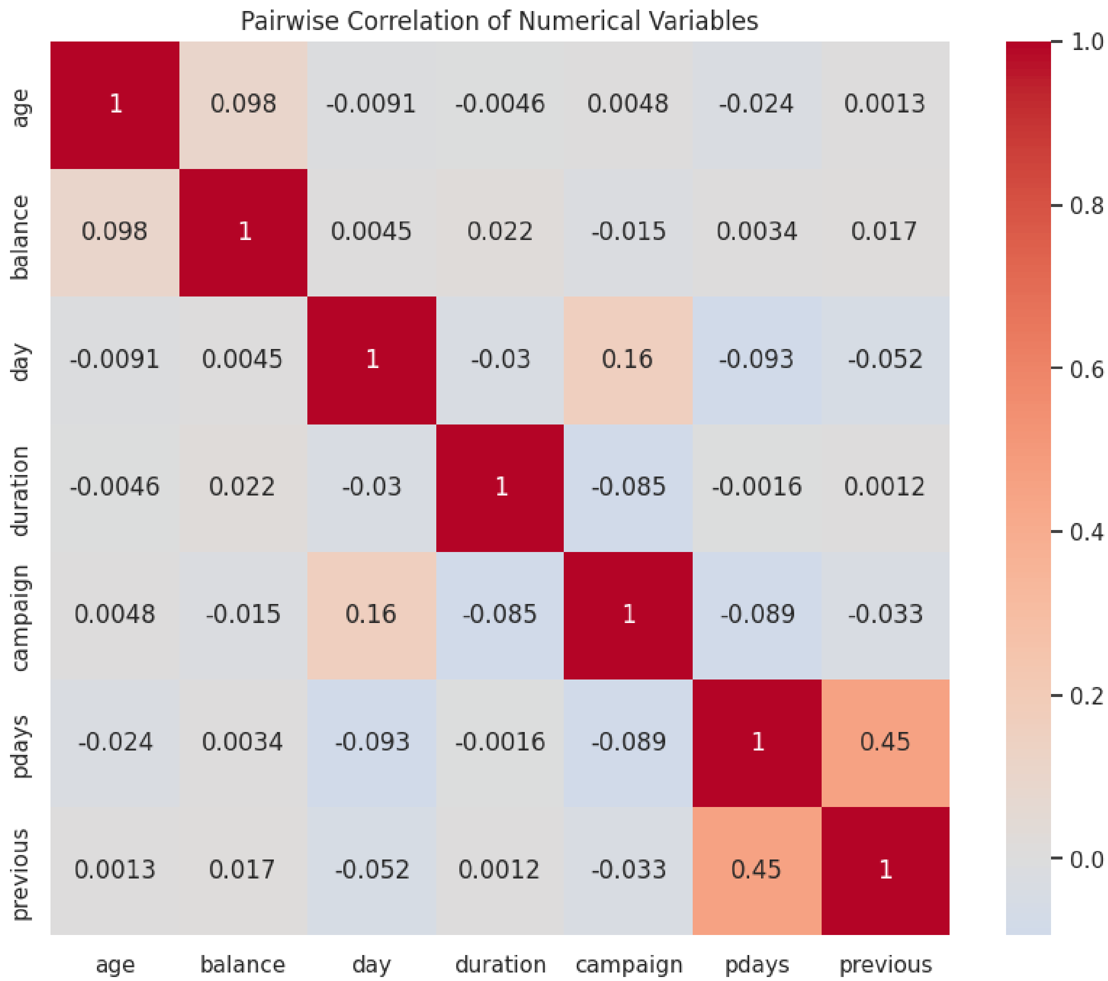





[24]:
```
# Pairwise scatter plots for numerical variables
sns.pairplot(df.select_dtypes(include=['number']))
plt.show()
```




[25]:
```python
# Compute correlation matrix
corr_matrix = df.select_dtypes(include=['number']).corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title("Pairwise Correlation of Numerical Variables")
plt.show()
```




[26]:
```
int_columns = df.select_dtypes('int64').columns.tolist()

print(int_columns)
```

```
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
```

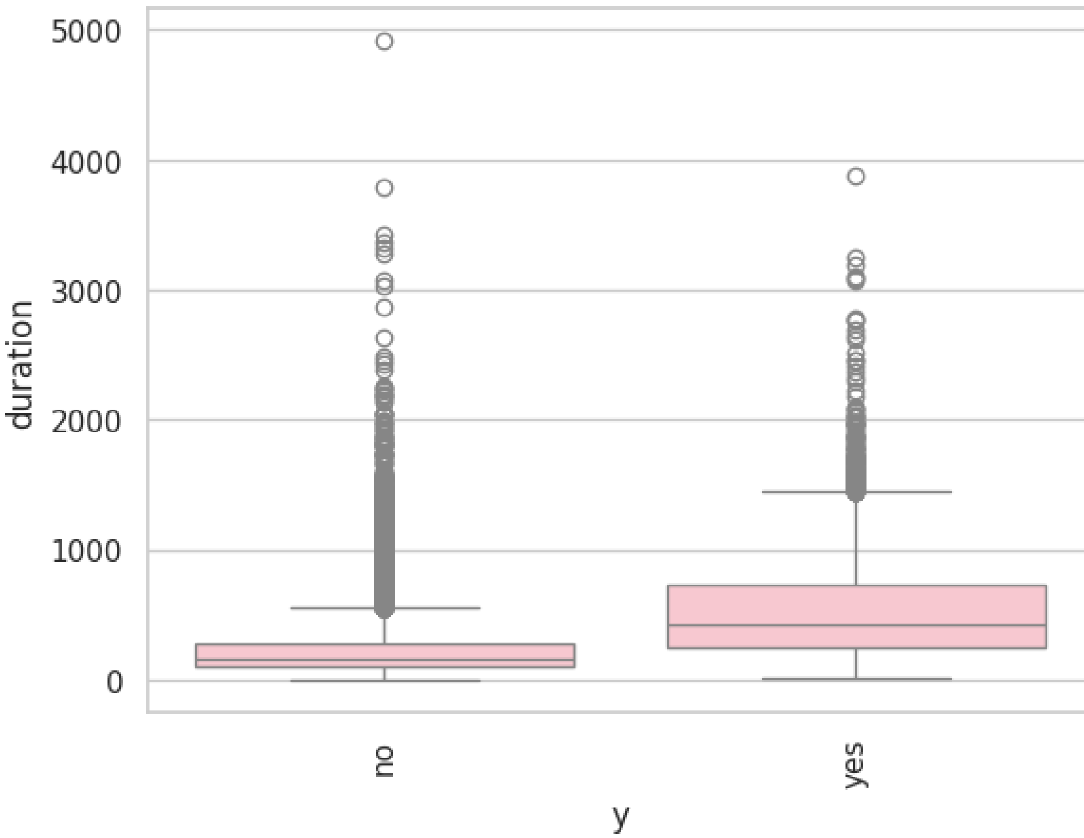
[ ]:
```

```

[27]:
```
object_columns = df.select_dtypes(include=['object']).columns.tolist()

print(object_columns)
```

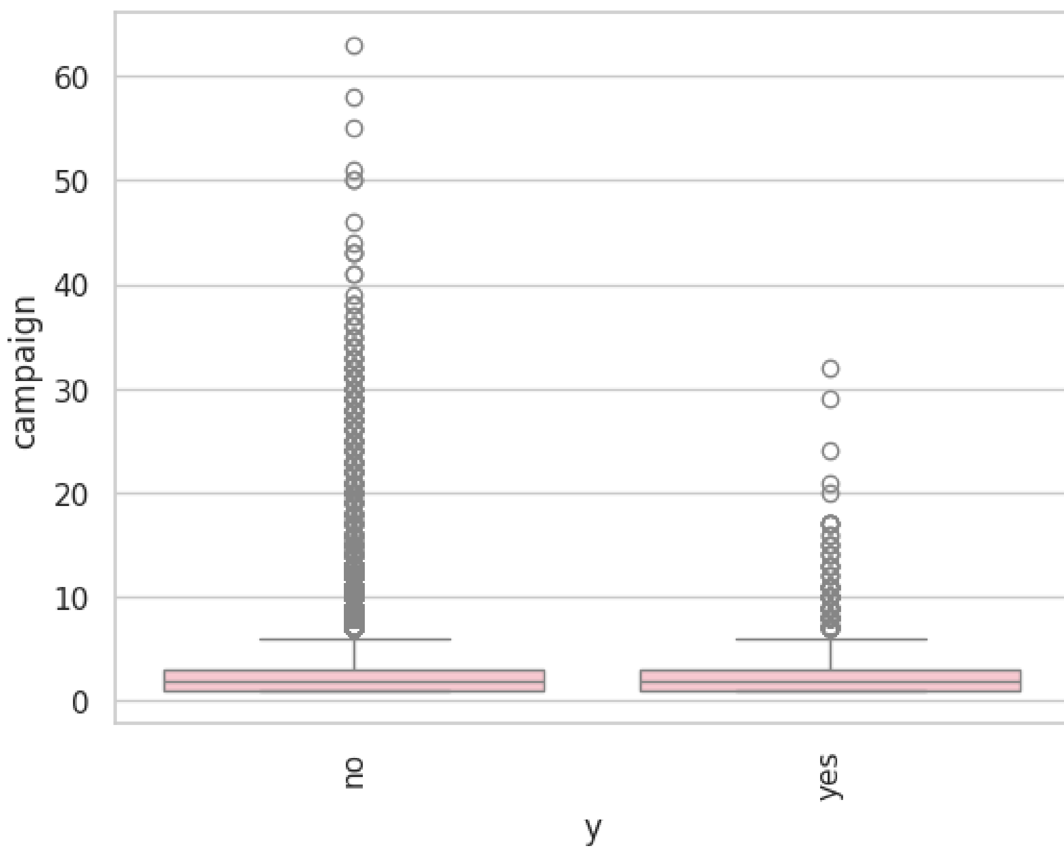
```
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
'month', 'poutcome', 'y']
```

[28]:
```
def cat_cont_plot(df, xvar, yvar):
    """
    Creates histograms of a continuous variable across categories.

    Parameters:
    - df: pandas DataFrame
    - xvar: str, categorical column name (e.g. 'Target')
    - yvar: str, continuous column name (e.g. 'Age')
    """
    sns.boxplot(x=xvar, y=yvar, data=df, color="pink")
    plt.xticks(rotation=90)


    # Create a larger facet grid of histograms by category
    #g = sns.FacetGrid(df, col=xvar, col_wrap=4, height=6, aspect=0.5)  #␣
 ↪Increased size and layout
    #g.map(sns.histplot, yvar, bins=20, color="green")

    # Add titles and labels
    #g.set_axis_labels(yvar, "Frequency")
    #g.set_titles(col_template="{col_name}")
   # plt.tight_layout()
    plt.show()
```

[29]:
```
cat_cont_plot(df,"y", "duration")
```



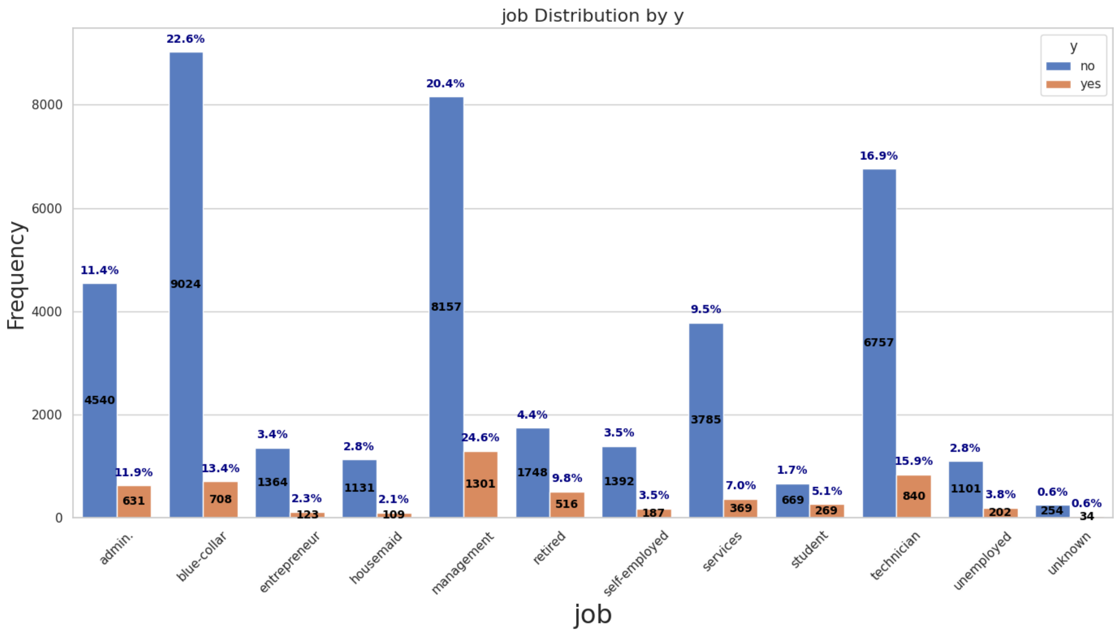

[30]:
```
cat_cont_plot(df,"y", "campaign")
```



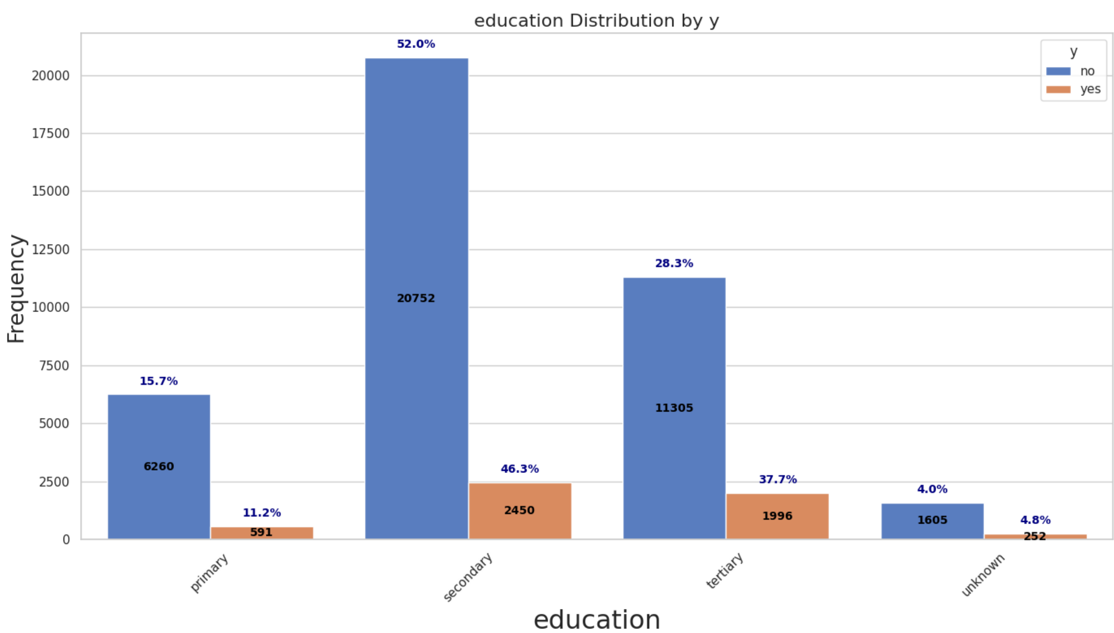

[31]:
```
def cat_cat_plot(dataframe, column_name, hue_column):
    """
    Plots a grouped bar chart showing category frequencies split by hue,
    with both frequency (inside bar) and proportion (above bar) labels.

    Parameters:
    - dataframe: pandas DataFrame
    - column_name: str, name of the categorical column to visualize (x-axis)
    - hue_column: str, name of the second categorical variable to group by (hue)
    """
    # Count combinations of column and hue
    counts_df = dataframe.groupby([column_name, hue_column]).size().
 ↪reset_index(name='count')
    total_counts = dataframe[column_name].value_counts()

    # Set plot style
    sns.set(style="whitegrid")
    plt.figure(figsize=(14, 8))  # Larger frame

    # Create bar plot
```

```
    ax = sns.barplot(x=column_name, y='count', hue=hue_column, data=counts_df,␣
↪palette="muted")
    # Annotate bars
    for container in ax.containers:
        for bar in container:
            height = bar.get_height()
            x = bar.get_x() + bar.get_width() / 2
            category = bar.get_label()
            base_x = int(round(x))  # used for proportion lookup
            if height > 0:
                ax.text(x, height * 0.5, f'{int(height)}', ha='center',␣
↪va='center',
                        fontsize=10, color='black', fontweight='bold')
                ax.text(x, height + max(counts_df['count']) * 0.02, f'{height /␣
↪sum(container.datavalues):.1%}',
                        ha='center', fontsize=10, color='navy',␣
↪fontweight='bold')


    # Beautify plot
    plt.title(f'{column_name} Distribution by {hue_column}', fontsize=16)
    plt.xlabel(column_name, fontsize=23)
    plt.ylabel('Frequency', fontsize=19)
    plt.xticks(rotation=45)
    plt.legend(title=hue_column)
    plt.tight_layout()
    plt.show()
```

[32]:
```
cat_cat_plot(df,'job',"y")
```




[33]:
```
cat_cat_plot(df,'education',"y")
```




[ ]:

[34]:
```
#import numpy as np

# Indicator: whether client was contacted before
df['previous_contact'] = (df['pdays'] != -1).astype(int)

# Replace -1 with NaN so pdays is only meaningful when contact exists
df['pdays'] = df['pdays'].replace(-1, np.nan)
```

# 1 Research Question 1: Which customer and campaign features best predict term deposit subscription?

[35]:
```
rq1_vars=["age", "job","education", "marital", "balance", "default",␣
↪"housing","loan", "contact","duration", "poutcome", "previous","campaign"␣
↪,'previous_contact','pdays',"y"]
```

[36]:
```
dfrq1=df[rq1_vars]
```

[37]:
```
dfrq1.head()
```

[37]:
```
   age           job  education  marital  balance default housing loan  \
0   58    management   tertiary  married     2143      no     yes   no
1   44    technician  secondary   single       29      no     yes   no
2   33  entrepreneur  secondary  married        2      no     yes  yes
3   47   blue-collar    unknown  married     1506      no     yes   no
4   33       unknown    unknown   single        1      no      no   no

   contact  duration poutcome  previous  campaign  previous_contact  pdays   y
0  unknown       261  unknown         0         1                 0    NaN  no
1  unknown       151  unknown         0         1                 0    NaN  no
2  unknown        76  unknown         0         1                 0    NaN  no
3  unknown        92  unknown         0         1                 0    NaN  no
4  unknown       198  unknown         0         1                 0    NaN  no
```

[ ]:

[ ]:

[ ]:

## 1.1 Preprocessing

[38]:
```
dfrq1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
```

```
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   age               45211 non-null  int64  
 1   job               45211 non-null  object 
 2   education         45211 non-null  object 
 3   marital           45211 non-null  object 
 4   balance           45211 non-null  int64  
 5   default           45211 non-null  object 
 6   housing           45211 non-null  object 
 7   loan              45211 non-null  object 
 8   contact           45211 non-null  object 
 9   duration          45211 non-null  int64  
 10  poutcome          45211 non-null  object 
 11  previous          45211 non-null  int64  
 12  campaign          45211 non-null  int64  
 13  previous_contact  45211 non-null  int64  
 14  pdays             8257 non-null   float64
 15  y                 45211 non-null  object 
dtypes: float64(1), int64(6), object(9)
memory usage: 5.5+ MB
```

[ ]:

[ ]:

[39]:
```
dfrq1.shape
```

[39]:
```
(45211, 16)
```

### 1.1.1 Ecoding all object variables into dummy variable.

[40]:
```
cat_cols = dfrq1.select_dtypes(include='object').columns

dfrq1_enc = pd.get_dummies(dfrq1, columns=cat_cols, drop_first=True)

print(dfrq1_enc.shape)
print(dfrq1_enc.head())
```

```
(45211, 32)
   age  balance  duration  previous  campaign  previous_contact  pdays  \
0   58     2143       261         0         1                 0    NaN   
1   44       29       151         0         1                 0    NaN   
2   33        2        76         0         1                 0    NaN   
3   47     1506        92         0         1                 0    NaN   
4   33        1       198         0         1                 0    NaN   
```

```
   job_blue-collar  job_entrepreneur  job_housemaid  …  marital_single  \
0                0                 0              0  …               0
1                0                 0              0  …               1
2                0                 1              0  …               0
3                1                 0              0  …               0
4                0                 0              0  …               1

   default_yes  housing_yes  loan_yes  contact_telephone  contact_unknown  \
0            0            1         0                  0                1
1            0            1         0                  0                1
2            0            1         1                  0                1
3            0            1         0                  0                1
4            0            0         0                  0                1

   poutcome_other  poutcome_success  poutcome_unknown  y_yes
0               0                 0                 1      0
1               0                 0                 1      0
2               0                 0                 1      0
3               0                 0                 1      0
4               0                 0                 1      0

[5 rows x 32 columns]
```

[41]:
```
X = dfrq1_enc.drop(columns=['y_yes'])
y = dfrq1_enc['y_yes']
```

[42]:
```
#Train-test split (stratified)
X_train, X_test, y_train, y_test = train_test_split( X, y,
    test_size=0.3,
    stratify=y,
    random_state=42
)
```

Given the presence of substantial outliers in several numerical variables, RobustScaler was used to scale the features, as it relies on the median and interquartile range and is less sensitive to extreme values.

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

Since pdays is undefined for clients never previously contacted, a binary indicator was introduced to represent prior contact status. Remaining missing values in pdays were imputed using the median to avoid introducing artificial extremes.

The variable pdays uses the value $-1$ to indicate clients who were never previously contacted. As this value is not a valid numerical quantity, it was replaced with missing values, and a binary indicator variable (previous_contact) was created to capture prior contact status. Since logistic regression does not handle missing values natively, a median imputation strategy was applied within a modeling pipeline prior to robust scaling and model fitting. RobustScaler was used due to the presence of substantial outliers in several numerical variables.

[43]:
```
log_reg_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),   # handles NaNs (pdays)
    ('scaler', RobustScaler()),                       # robust to outliers
    ('model', LogisticRegression(
        max_iter=1000,
        class_weight='balanced',
        random_state=42
    ))
])
log_reg_pipeline.fit(X_train, y_train)
y_pred = log_reg_pipeline.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix of the Logistic Regression \n",cm)
print("Classification report of the Logistic Regression␣
 ↪\n",classification_report(y_test, y_pred))
#ROC-AUC
y_prob = log_reg_pipeline.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_prob)

print("ROC-AUC of the Logistic Regression:", roc_auc)
```

```
Confusion matrix of the Logistic Regression
 [[10005  1972]
 [  343  1244]]
Classification report of the Logistic Regression
               precision    recall  f1-score   support

           0       0.97      0.84      0.90     11977
           1       0.39      0.78      0.52      1587

    accuracy                           0.83     13564
   macro avg       0.68      0.81      0.71     13564
weighted avg       0.90      0.83      0.85     13564

ROC-AUC of the Logistic Regression: 0.8917974426830168
```

**Interpret coefficients**

[44]:
```
import pandas as pd
import numpy as np

coef = log_reg_pipeline.named_steps['model'].coef_[0]
```

```python
coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coef,
    'Odds_Ratio': np.exp(coef)
}).sort_values(by='Odds_Ratio', ascending=False)

coef_df
```

[44]:

| | Feature | Coefficient | Odds_Ratio |
|---|---|---|---|
| 29 | poutcome_success | 2.387607 | 10.887405 |
| 2 | duration | 1.192009 | 3.293691 |
| 14 | job_student | 0.564808 | 1.759110 |
| 19 | education_tertiary | 0.513514 | 1.671154 |
| 11 | job_retired | 0.429690 | 1.536781 |
| 20 | education_unknown | 0.305059 | 1.356705 |
| 18 | education_secondary | 0.258492 | 1.294975 |
| 28 | poutcome_other | 0.251713 | 1.286227 |
| 22 | marital_single | 0.187751 | 1.206533 |
| 5 | previous_contact | 0.182628 | 1.200368 |
| 26 | contact_telephone | 0.053333 | 1.054781 |
| 1 | balance | 0.040106 | 1.040921 |
| 3 | previous | 0.030486 | 1.030956 |
| 0 | age | 0.007305 | 1.007332 |
| 6 | pdays | 0.000238 | 1.000239 |
| 30 | poutcome_unknown | -0.098979 | 0.905762 |
| 21 | marital_married | -0.180855 | 0.834557 |
| 10 | job_management | -0.204880 | 0.814745 |
| 4 | campaign | -0.235807 | 0.789933 |
| 23 | default_yes | -0.251401 | 0.777711 |
| 16 | job_unemployed | -0.282399 | 0.753973 |
| 15 | job_technician | -0.326132 | 0.721710 |
| 17 | job_unknown | -0.366712 | 0.693009 |
| 13 | job_services | -0.444485 | 0.641155 |
| 8 | job_entrepreneur | -0.536855 | 0.584584 |
| 9 | job_housemaid | -0.543343 | 0.580803 |
| 7 | job_blue-collar | -0.543642 | 0.580630 |
| 12 | job_self-employed | -0.545387 | 0.579617 |
| 25 | loan_yes | -0.685377 | 0.503900 |
| 24 | housing_yes | -0.849504 | 0.427627 |
| 27 | contact_unknown | -1.326366 | 0.265440 |

[ ]:

```python

```

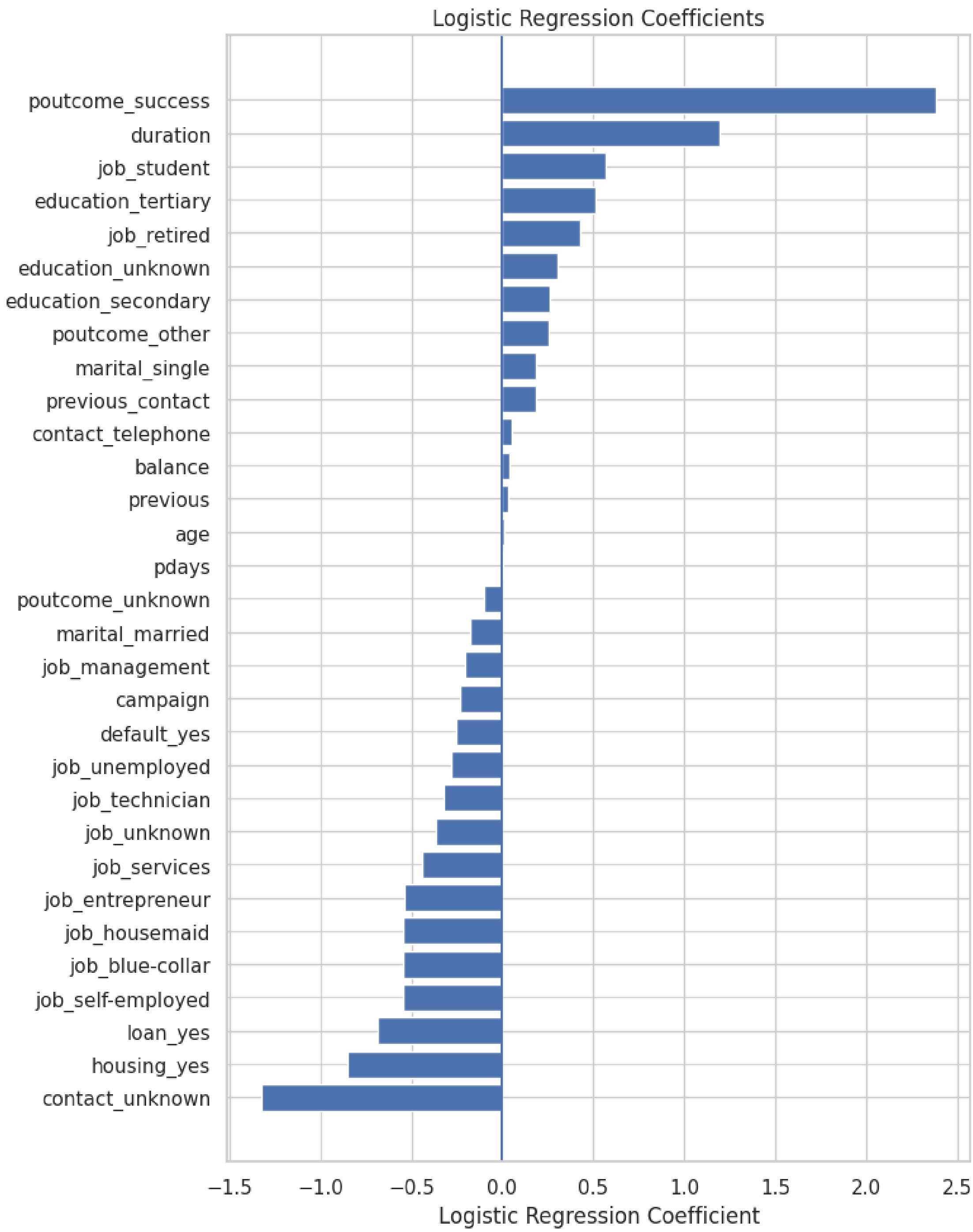
Interpretation rule:

Odds Ratio > 1 → increases probability of subscription

Odds Ratio < 1 → decreases probability

[45]:
```python
#Extract coefficients and odds ratios from the pipeline
coef = log_reg_pipeline.named_steps['model'].coef_[0]

coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coef,
    'Odds_Ratio': np.exp(coef)
})

coef_df = coef_df.sort_values(by='Odds_Ratio', ascending=False)
```

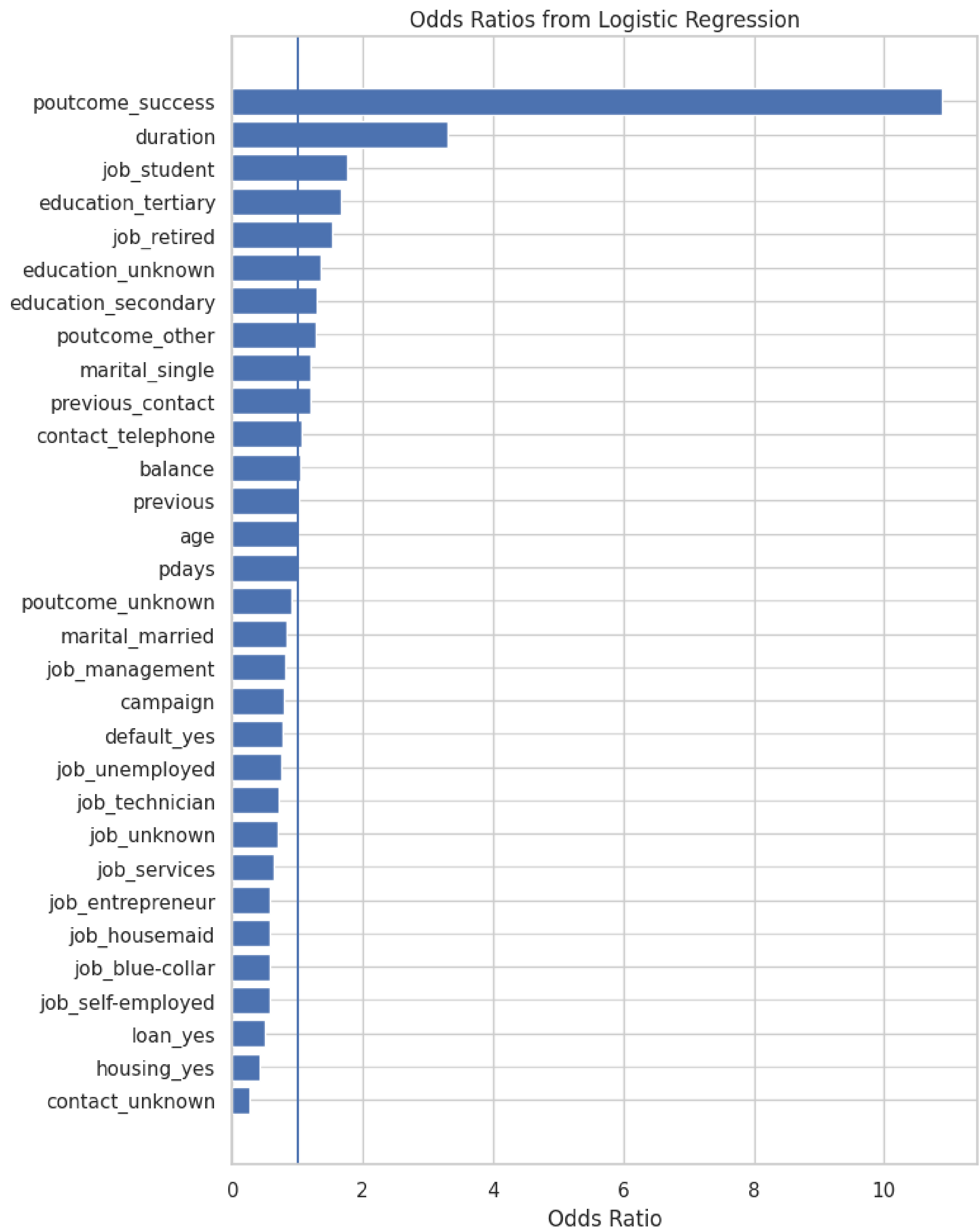
[46]:
```python
#Visualise coefficients (direction + strength)
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 10))
plt.barh(coef_df['Feature'], coef_df['Coefficient'])
plt.axvline(0)
plt.xlabel('Logistic Regression Coefficient')
plt.title('Logistic Regression Coefficients')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```




[47]:

```
#Visualise odds ratios
plt.figure(figsize=(8, 10))
plt.barh(coef_df['Feature'], coef_df['Odds_Ratio'])
plt.axvline(1)
plt.xlabel('Odds Ratio')
```

```python
plt.title('Odds Ratios from Logistic Regression')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



# Fit Random Forest (no scaling needed)

Random Forest does NOT need scaling and handles outliers naturally.

[48]:
```
rf = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    min_samples_leaf=5,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)
```

[48]:
```
RandomForestClassifier(class_weight='balanced', min_samples_leaf=5,
                       n_estimators=300, n_jobs=-1, random_state=42)
```

**Model evaluation**

[49]:
```
y_pred = rf.predict(X_test)
y_prob = rf.predict_proba(X_test)[:, 1]

print("The Confusion Matrix of Random Forest:\n", confusion_matrix(y_test,␣
 ↪y_pred))
print("\nClassification Report of Random Forest:\n",␣
 ↪classification_report(y_test, y_pred))
print("The ROC-AUC of Random Forest:", roc_auc_score(y_test, y_prob))
```

```
The Confusion Matrix of Random Forest:
 [[10483  1494]
 [  386  1201]]

Classification Report of Random Forest:
               precision    recall  f1-score   support

           0       0.96      0.88      0.92     11977
           1       0.45      0.76      0.56      1587

    accuracy                           0.86     13564
   macro avg       0.71      0.82      0.74     13564
weighted avg       0.90      0.86      0.88     13564

The ROC-AUC of Random Forest: 0.9023631672951818
```

**Feature importance**

[50]:
```
feature_importance = pd.DataFrame({
    'Feature': X.columns,
```

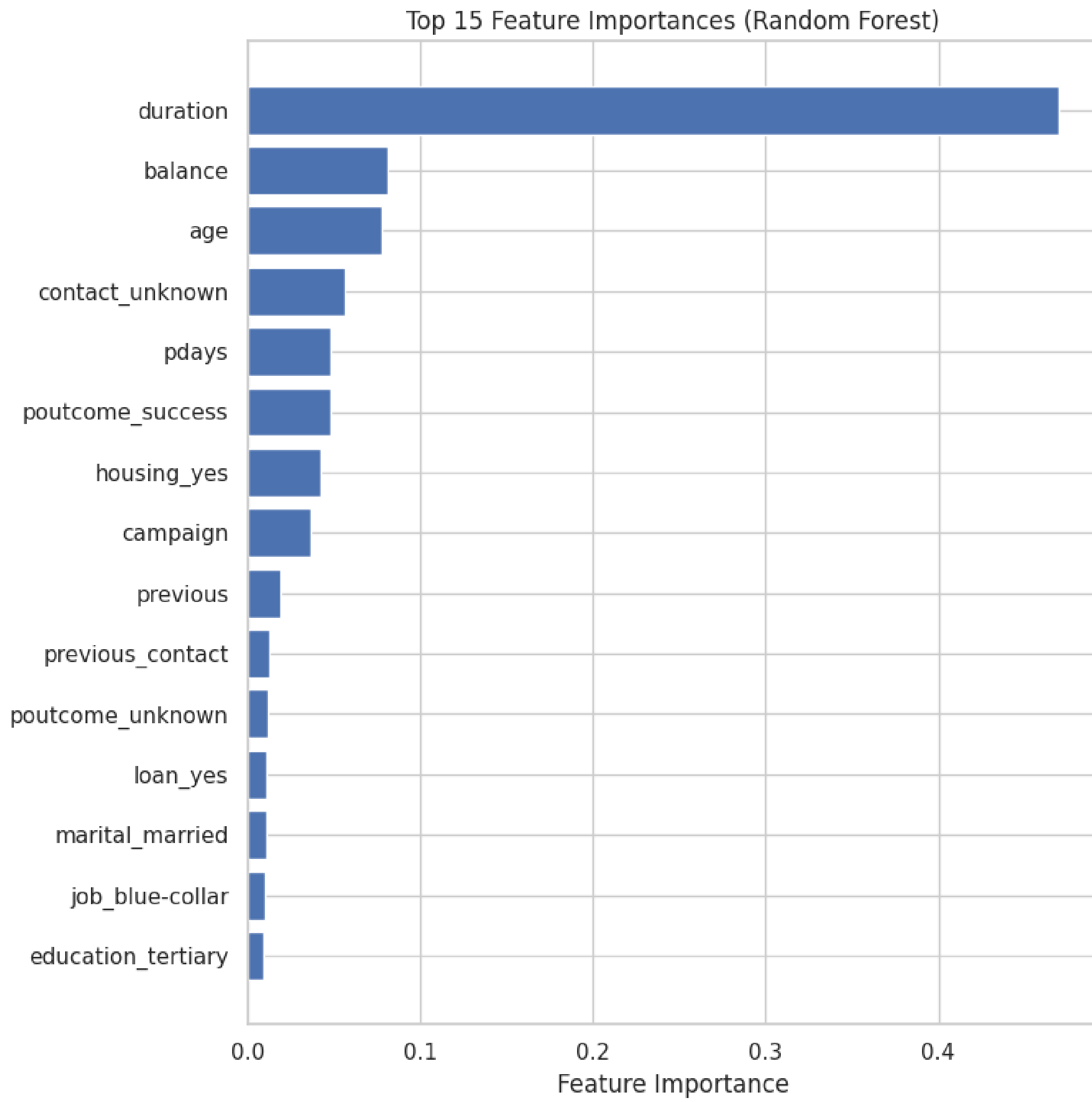
```
    'Importance': rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

feature_importance.head(15)
```

[50]:

| | Feature | Importance |
|---|---|---|
| 2 | duration | 0.469407 |
| 1 | balance | 0.080987 |
| 0 | age | 0.077718 |
| 27 | contact_unknown | 0.055870 |
| 6 | pdays | 0.047498 |
| 29 | poutcome_success | 0.047468 |
| 24 | housing_yes | 0.041986 |
| 4 | campaign | 0.035903 |
| 3 | previous | 0.019197 |
| 5 | previous_contact | 0.012006 |
| 30 | poutcome_unknown | 0.011736 |
| 25 | loan_yes | 0.011049 |
| 21 | marital_married | 0.010674 |
| 7 | job_blue-collar | 0.009519 |
| 19 | education_tertiary | 0.008935 |

[51]:

```
#Visualise top feature importances
top_features = feature_importance.head(15)

plt.figure(figsize=(8, 8))
plt.barh(top_features['Feature'], top_features['Importance'])
plt.xlabel('Feature Importance')
plt.title('Top 15 Feature Importances (Random Forest)')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```




A Random Forest classifier was trained to identify key customer and campaign features associated with term deposit subscription. As a tree-based model, Random Forest does not require feature scaling and is robust to outliers. Class imbalance was addressed using balanced class weights. Model performance was evaluated using recall, F1-score, and ROC–AUC. Feature importance scores were extracted to identify the most influential predictors.

## 1.2 SHAP

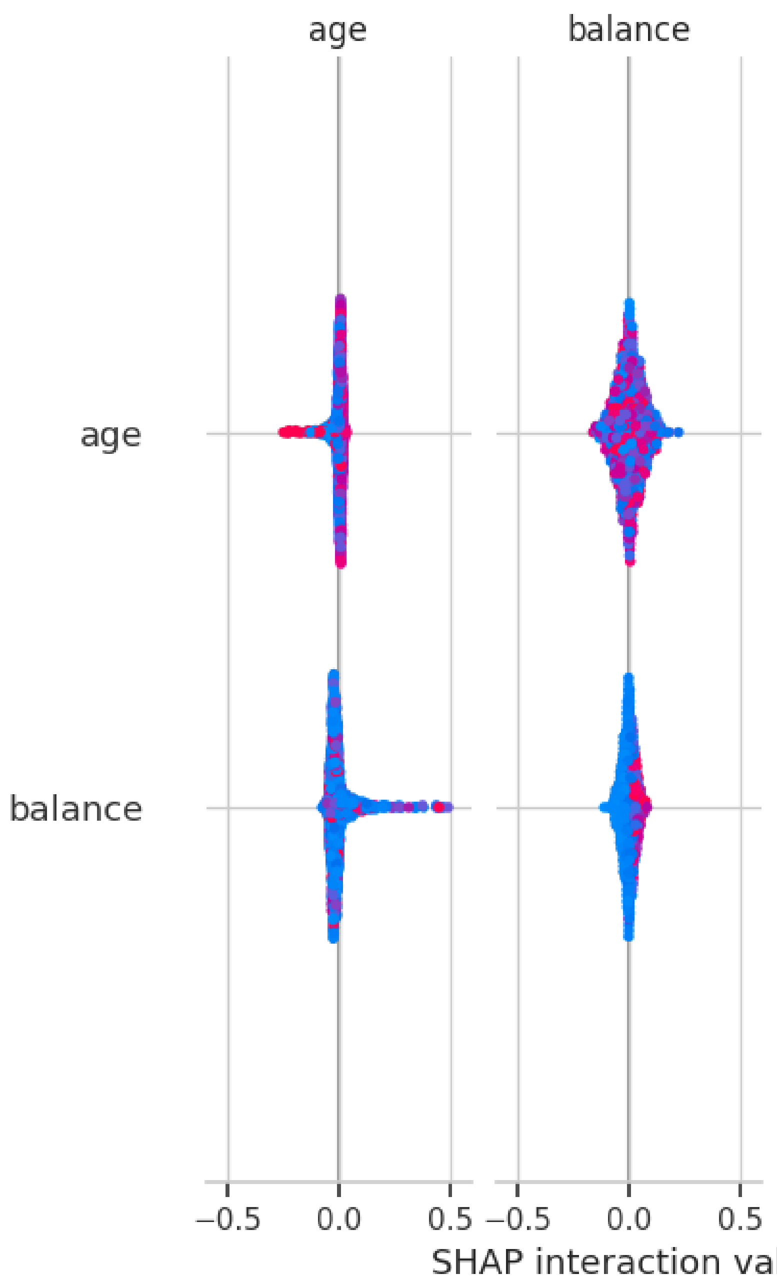
To enhance interpretability of the Random Forest model, SHAP (SHapley Additive exPlanations) values were computed. SHAP provides a model-agnostic explanation framework that quantifies both the magnitude and direction of each feature's contribution to the predicted outcome. The SHAP summary plot highlights the most influential features and reveals whether higher or lower values of each feature increase the likelihood of subscription.

[52]:
```python
# Take a random sample of the test set
X_test_shap = X_test.sample(n=1000, random_state=42)
```

[53]:
```python
#explainer = shap.TreeExplainer(rf)
#shap_values = explainer.shap_values(X_test_shap)
```

[54]:
```python
# Use the unified SHAP API
explainer = shap.Explainer(rf, X_train)

# Compute SHAP values on the sample
shap_values = explainer(X_test_shap)

# Summary plot (this WILL work)
shap.summary_plot(
    shap_values.values,
    X_test_shap,
    plot_type="dot",
    show=True
)
```

```
100%|===================| 1996/2000 [02:34<00:00]
```




[ ]:

[ ]:

[55]:
```
# ---------------------------------------------------
# Compute SHAP INTERACTION values (slow but needed once)
# ---------------------------------------------------
shap_interaction_values = explainer.shap_interaction_values(X_test_shap)
```

```python
# For binary classification: take positive class ("yes")
shap_inter_pos = shap_interaction_values[:, :, 1]

# --------------------------------------------------
# Quantify MAIN vs INTERACTION effects
# --------------------------------------------------
# ---- MAIN EFFECTS (positive class only) ----
shap_main = np.mean(
    np.abs(shap_values.values[:, :, 1]),  # class "yes"
    axis=0
)
# Mean absolute MAIN effects per feature
#shap_main = np.mean(np.abs(shap_values.values), axis=0)

# Mean absolute INTERACTION effects
shap_inter = np.mean(np.abs(shap_inter_pos), axis=0)

# Remove self-interactions (diagonal)
np.fill_diagonal(shap_inter, 0)

# Aggregate interaction strength per feature
interaction_strength = shap_inter.sum(axis=1)

# --------------------------------------------------
# Comparison table (proof of weak interactions)
# --------------------------------------------------
comparison = pd.DataFrame({
    "feature": X_test_shap.columns,
    "main_effect": shap_main,
    "interaction_effect": interaction_strength,
    "interaction_ratio": interaction_strength / shap_main
}).sort_values("interaction_ratio", ascending=False)

comparison
```

```
FEATURE_DEPENDENCE::independent does not support interactions!
```

[55]:

| | feature | main_effect | interaction_effect | interaction_ratio |
|---|---|---|---|---|
| 0 | age | 0.016881 | 0.0 | 0.0 |
| 16 | job_unemployed | 0.000485 | 0.0 | 0.0 |
| 29 | poutcome_success | 0.016465 | 0.0 | 0.0 |
| 28 | poutcome_other | 0.000807 | 0.0 | 0.0 |
| 27 | contact_unknown | 0.046133 | 0.0 | 0.0 |
| 26 | contact_telephone | 0.000857 | 0.0 | 0.0 |
| 25 | loan_yes | 0.009922 | 0.0 | 0.0 |
| 24 | housing_yes | 0.037571 | 0.0 | 0.0 |

| | | | | |
|---|---|---|---|---|
| 23 | default_yes | 0.000445 | 0.0 | 0.0 |
| 22 | marital_single | 0.007459 | 0.0 | 0.0 |
| 21 | marital_married | 0.010193 | 0.0 | 0.0 |
| 20 | education_unknown | 0.000677 | 0.0 | 0.0 |
| 19 | education_tertiary | 0.013163 | 0.0 | 0.0 |
| 18 | education_secondary | 0.002640 | 0.0 | 0.0 |
| 17 | job_unknown | 0.000004 | 0.0 | 0.0 |
| 15 | job_technician | 0.002213 | 0.0 | 0.0 |
| 1 | balance | 0.020867 | 0.0 | 0.0 |
| 14 | job_student | 0.001402 | 0.0 | 0.0 |
| 13 | job_services | 0.001417 | 0.0 | 0.0 |
| 12 | job_self-employed | 0.000555 | 0.0 | 0.0 |
| 11 | job_retired | 0.000969 | 0.0 | 0.0 |
| 10 | job_management | 0.003568 | 0.0 | 0.0 |
| 9 | job_housemaid | 0.000505 | 0.0 | 0.0 |
| 8 | job_entrepreneur | 0.000599 | 0.0 | 0.0 |
| 7 | job_blue-collar | 0.005405 | 0.0 | 0.0 |
| 6 | pdays | 0.012604 | 0.0 | 0.0 |
| 5 | previous_contact | 0.007716 | 0.0 | 0.0 |
| 4 | campaign | 0.017559 | 0.0 | 0.0 |
| 3 | previous | 0.009495 | 0.0 | 0.0 |
| 2 | duration | 0.134146 | 0.0 | 0.0 |
| 30 | poutcome_unknown | 0.007265 | 0.0 | 0.0 |

[ ]:

```
```

Although Random Forest models are capable of capturing nonlinear interactions, SHAP interaction analysis shows that interaction effects are negligible for all predictors. The model's predictions are therefore driven primarily by additive contributions of individual features rather than by complex feature interactions.

Methods / Explainability section

SHAP interaction values were computed on a representative subset of the test data to assess whether the Random Forest model relied on nonlinear feature interactions. Interaction effects were quantified and compared to main feature effects using the ratio of interaction strength to main effect magnitude.

Results / Interpretation section

Across all predictors, interaction effects were effectively zero relative to main effects. This indicates that the Random Forest model predominantly relies on additive feature contributions, with no evidence of strong pairwise interactions influencing subscription predictions.

This finding suggests that the predictive structure of the data is largely linear-additive, despite the use of a non-linear model.

# 2 RQ2 Can machine learning models outperform traditional statistical models in predicting subscription?

### 2.0.1 Logistic Regression

[56]:
```
log_reg_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),   # handles NaNs (pdays)
    ('scaler', RobustScaler()),                      # robust to outliers
    ('model', LogisticRegression(
        max_iter=1000,
        class_weight='balanced',
        random_state=42
    ))
])
log_reg_pipeline.fit(X_train, y_train)
```

[56]:
```
Pipeline(steps=[('imputer', SimpleImputer(strategy='median')),
                ('scaler', RobustScaler()),
                ('model',
                 LogisticRegression(class_weight='balanced', max_iter=1000,
                                    random_state=42))])
```

[57]:
```
y_pred = log_reg_pipeline.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
#ROC-AUC
y_prob = log_reg_pipeline.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_prob)

print("The ROC-AUC of Logistic Regression:", roc_auc)
print("\n the confusion matrix of Logistic regression:\n",cm)
print("\n the classification report of Logistic Regression␣
 ↪\n",classification_report(y_test, y_pred))
```

```
The ROC-AUC of Logistic Regression: 0.8917974426830168

 the confusion matrix of Logistic regression:
 [[10005  1972]
 [  343  1244]]

 the classification report of Logistic Regression
               precision    recall  f1-score   support

           0       0.97      0.84      0.90     11977
           1       0.39      0.78      0.52      1587

    accuracy                           0.83     13564
   macro avg       0.68      0.81      0.71     13564
```

```
weighted avg       0.90      0.83      0.85     13564
```

[58]:

```python
import pandas as pd
import numpy as np

coef = log_reg_pipeline.named_steps['model'].coef_[0]

coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coef,
    'Odds_Ratio': np.exp(coef)
}).sort_values(by='Odds_Ratio', ascending=False)

coef_df
```

[58]:

| | Feature | Coefficient | Odds_Ratio |
|---|---|---|---|
| 29 | poutcome_success | 2.387607 | 10.887405 |
| 2 | duration | 1.192009 | 3.293691 |
| 14 | job_student | 0.564808 | 1.759110 |
| 19 | education_tertiary | 0.513514 | 1.671154 |
| 11 | job_retired | 0.429690 | 1.536781 |
| 20 | education_unknown | 0.305059 | 1.356705 |
| 18 | education_secondary | 0.258492 | 1.294975 |
| 28 | poutcome_other | 0.251713 | 1.286227 |
| 22 | marital_single | 0.187751 | 1.206533 |
| 5 | previous_contact | 0.182628 | 1.200368 |
| 26 | contact_telephone | 0.053333 | 1.054781 |
| 1 | balance | 0.040106 | 1.040921 |
| 3 | previous | 0.030486 | 1.030956 |
| 0 | age | 0.007305 | 1.007332 |
| 6 | pdays | 0.000238 | 1.000239 |
| 30 | poutcome_unknown | -0.098979 | 0.905762 |
| 21 | marital_married | -0.180855 | 0.834557 |
| 10 | job_management | -0.204880 | 0.814745 |
| 4 | campaign | -0.235807 | 0.789933 |
| 23 | default_yes | -0.251401 | 0.777711 |
| 16 | job_unemployed | -0.282399 | 0.753973 |
| 15 | job_technician | -0.326132 | 0.721710 |
| 17 | job_unknown | -0.366712 | 0.693009 |
| 13 | job_services | -0.444485 | 0.641155 |
| 8 | job_entrepreneur | -0.536855 | 0.584584 |
| 9 | job_housemaid | -0.543343 | 0.580803 |
| 7 | job_blue-collar | -0.543642 | 0.580630 |
| 12 | job_self-employed | -0.545387 | 0.579617 |
| 25 | loan_yes | -0.685377 | 0.503900 |
| 24 | housing_yes | -0.849504 | 0.427627 |

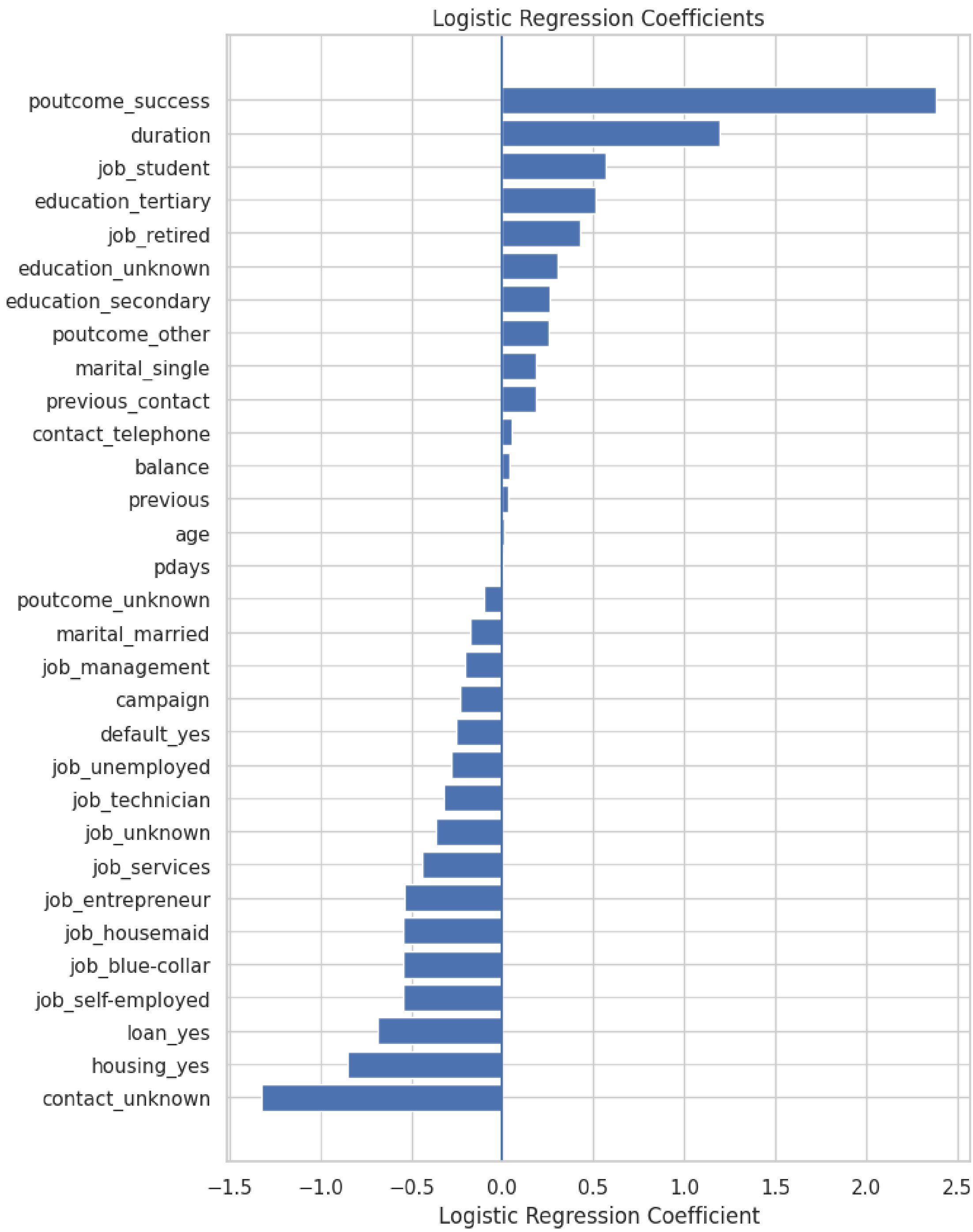
```
27      contact_unknown    -1.326366    0.265440
```

[59]:

```
#Extract coefficients and odds ratios from the pipeline
coef = log_reg_pipeline.named_steps['model'].coef_[0]

coef_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coef,
    'Odds_Ratio': np.exp(coef)
})

coef_df = coef_df.sort_values(by='Odds_Ratio', ascending=False)
```

[60]:

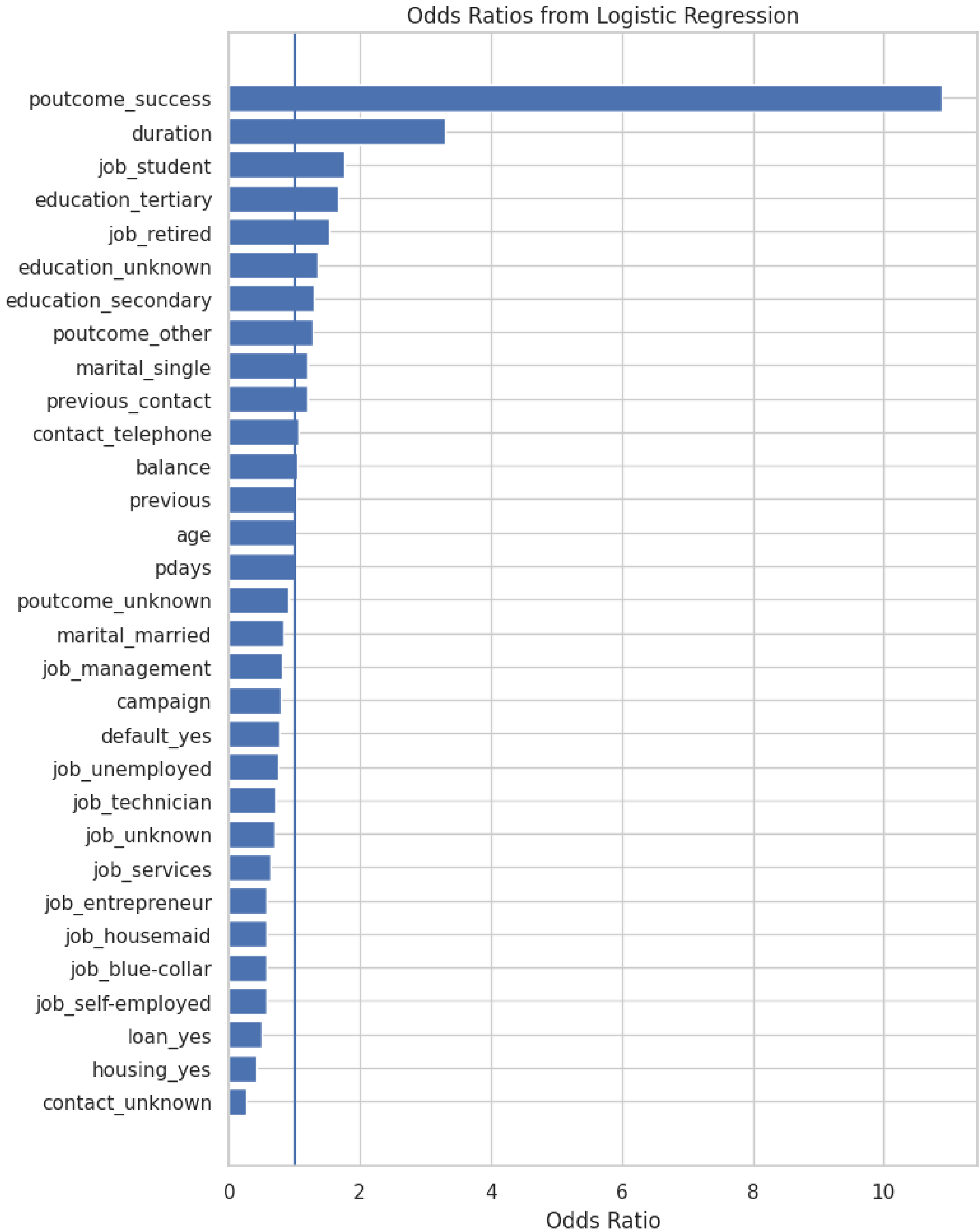
```
#Visualise coefficients (direction + strength)
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 10))
plt.barh(coef_df['Feature'], coef_df['Coefficient'])
plt.axvline(0)
plt.xlabel('Logistic Regression Coefficient')
plt.title('Logistic Regression Coefficients')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```




[61]:

```
#Visualise odds ratios
plt.figure(figsize=(8, 10))
plt.barh(coef_df['Feature'], coef_df['Odds_Ratio'])
plt.axvline(1)
plt.xlabel('Odds Ratio')
```

```python
plt.title('Odds Ratios from Logistic Regression')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



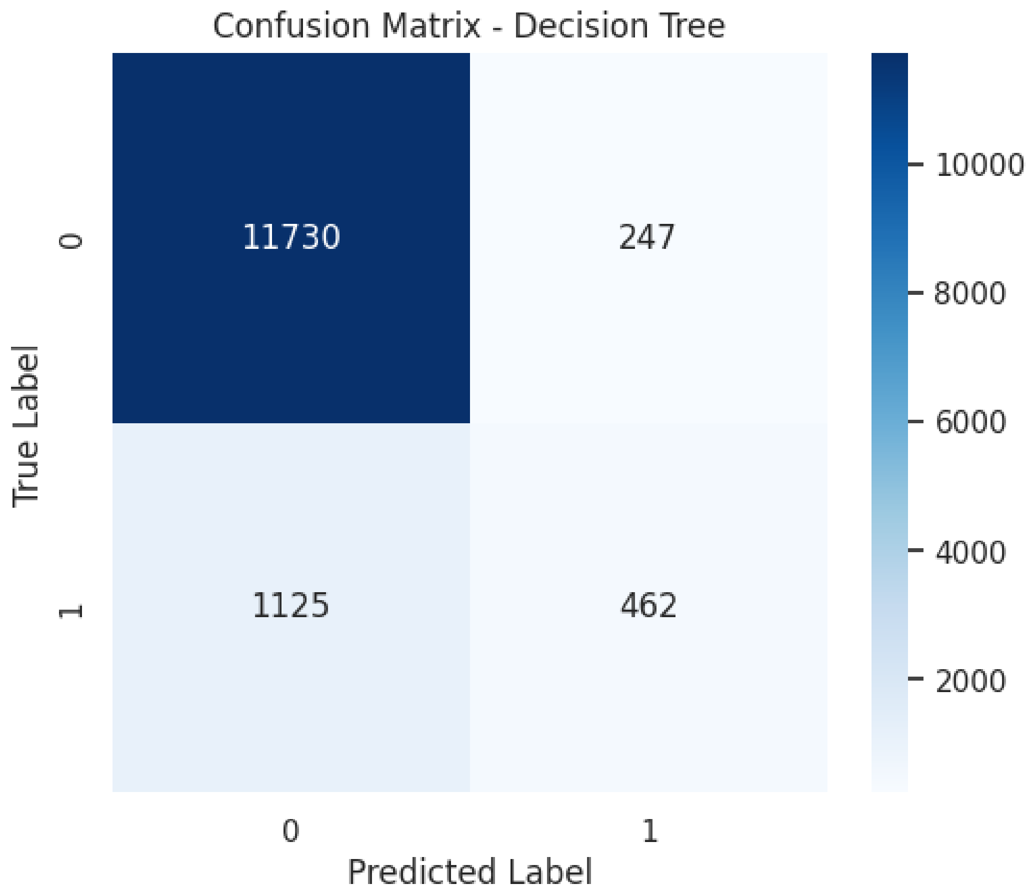
### 2.0.2 Decision Tree

[62]:
```
#model_dt = DecisionTreeClassifier(random_state=42)
model_dt = DecisionTreeClassifier(
    max_depth=4,      # limit depth
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42
)
model_dt.fit(X_train, y_train)
y_pred_dt = model_dt.predict(X_test)
# Get predicted probabilities for the positive class
y_prob_dt = model_dt.predict_proba(X_test)[:, 1]

# Compute AUC
auc_dt = roc_auc_score(y_test, y_prob_dt)

print("The AUC of Decision Tree:", auc_dt)
print("Accuracy of Decision Tree:", accuracy_score(y_test, y_pred_dt))
print("\nClassification Report of Decision Tree:\n",␣
↪classification_report(y_test, y_pred_dt))
cm_dt = confusion_matrix(y_test, y_pred_dt)
print("\nConfusion Matrix of Decision Tree:\n", cm_dt)
```

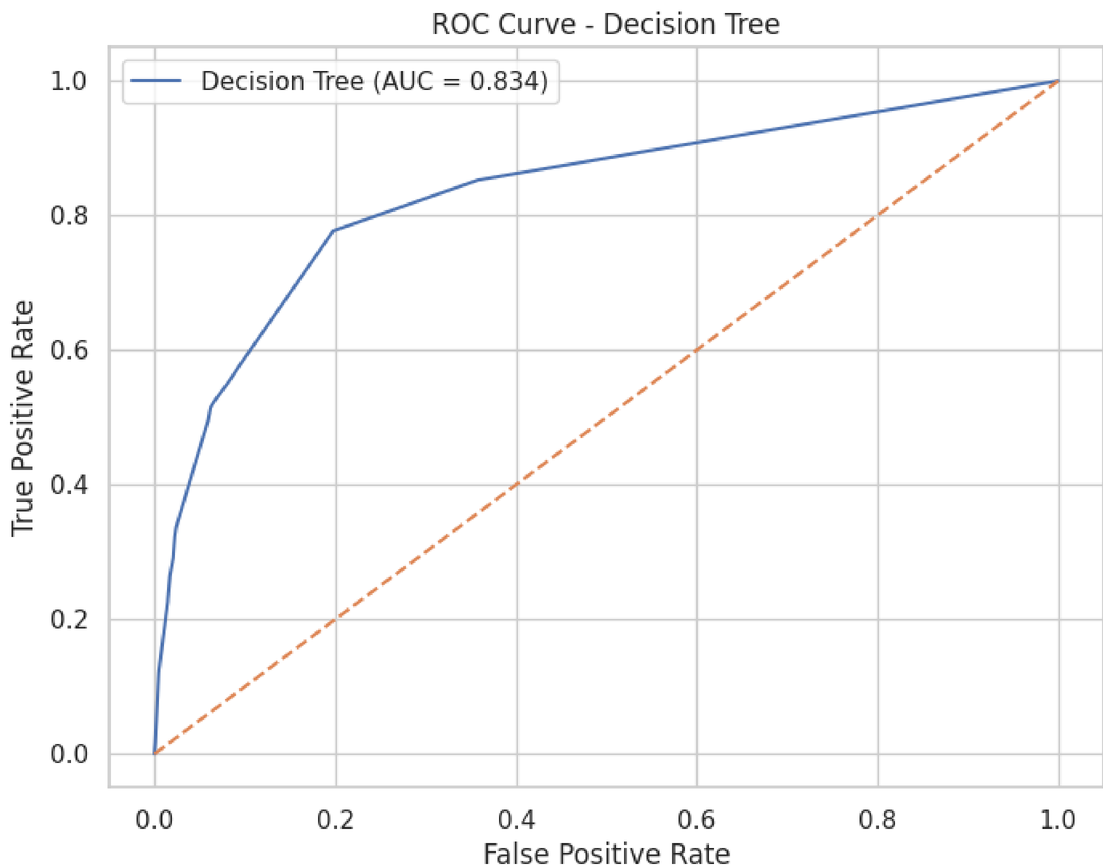
```
The AUC of Decision Tree: 0.8336726204746873
Accuracy of Decision Tree: 0.898849896785609

Classification Report of Decision Tree:
               precision    recall  f1-score   support

           0       0.91      0.98      0.94     11977
           1       0.65      0.29      0.40      1587

    accuracy                           0.90     13564
   macro avg       0.78      0.64      0.67     13564
weighted avg       0.88      0.90      0.88     13564


Confusion Matrix of Decision Tree:
 [[11730   247]
 [ 1125   462]]
```

[63]:
```
plt.figure(figsize=(6,5))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues',
            xticklabels=model_dt.classes_,
            yticklabels=model_dt.classes_)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

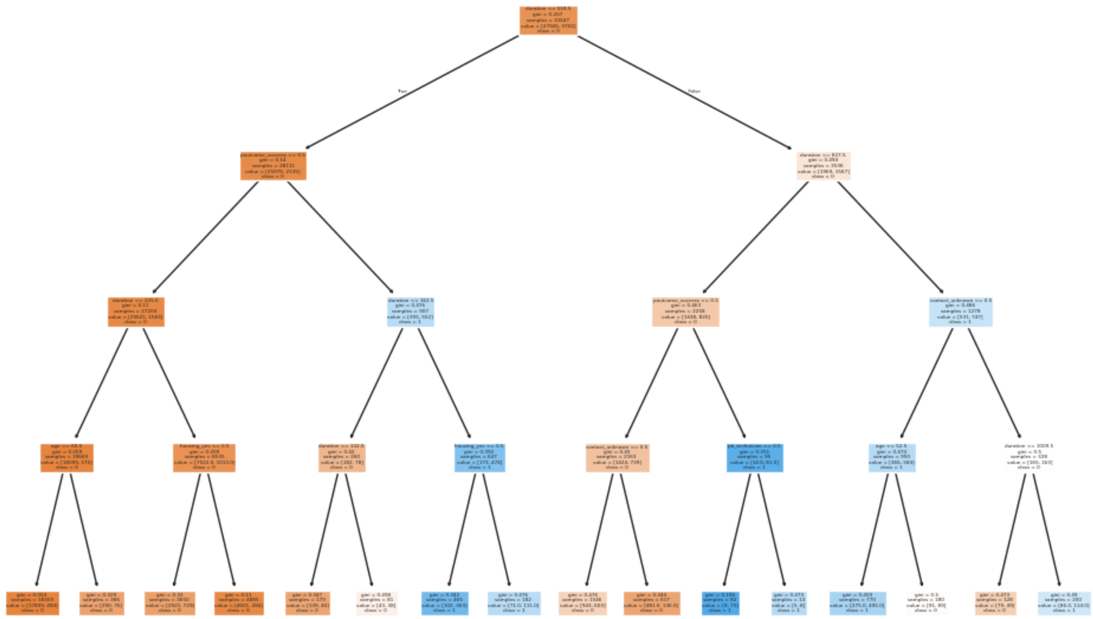
```python
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```




[64]:
```python
fpr, tpr, thresholds = roc_curve(y_test, y_prob_dt)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"Decision Tree (AUC = {auc_dt:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--")  # random model line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree")
plt.legend()
plt.show()
```




[65]:
```
plt.figure(figsize=(12,8))

tree.plot_tree(
    model_dt,
    feature_names=X_train.columns,              # <-- use dataframe columns
    class_names=model_dt.classes_.astype(str),  # <-- get class labels
    filled=True
)

plt.show()
```



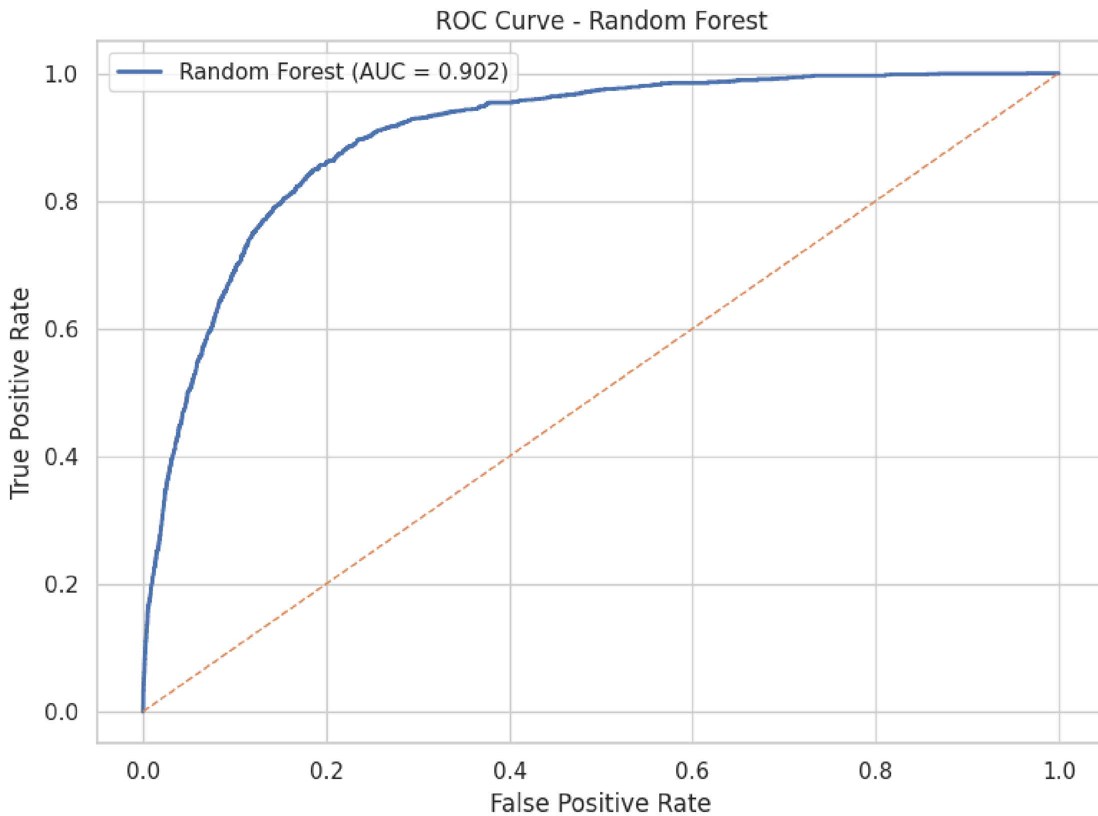
### 2.0.3 Random Forest

[66]:
```
rf2 = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    min_samples_leaf=5,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

rf2.fit(X_train, y_train)
y_pred_rf = rf2.predict(X_test)
y_prob_rf = rf2.predict_proba(X_test)[:, 1]

print("The confusion matrix of Random Forest:\n", confusion_matrix(y_test,␣
 ↪y_pred_rf))
print("\nThe classification Report of Random Forest:\n",␣
 ↪classification_report(y_test, y_pred_rf))
print("The ROC-AUC of Random Forest:", roc_auc_score(y_test, y_prob_rf))
```

```
The confusion matrix of Random Forest:
 [[10483  1494]
```

```
 [  386  1201]]

The classification Report of Random Forest:
               precision    recall  f1-score   support

           0       0.96      0.88      0.92     11977
           1       0.45      0.76      0.56      1587

    accuracy                           0.86     13564
   macro avg       0.71      0.82      0.74     13564
weighted avg       0.90      0.86      0.88     13564

The ROC-AUC of Random Forest: 0.9023631672951818
```

[67]:
```
# Compute ROC curve
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_prob_rf)

# Compute AUC (again, for label in plot)
auc_rf = roc_auc_score(y_test, y_prob_rf)

# Plot
plt.figure(figsize=(8,6))
plt.plot(fpr_rf, tpr_rf, linewidth=2,
         label=f"Random Forest (AUC = {auc_rf:.3f})")

# Diagonal baseline
plt.plot([0,1], [0,1], linestyle='--', linewidth=1)

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Random Forest")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



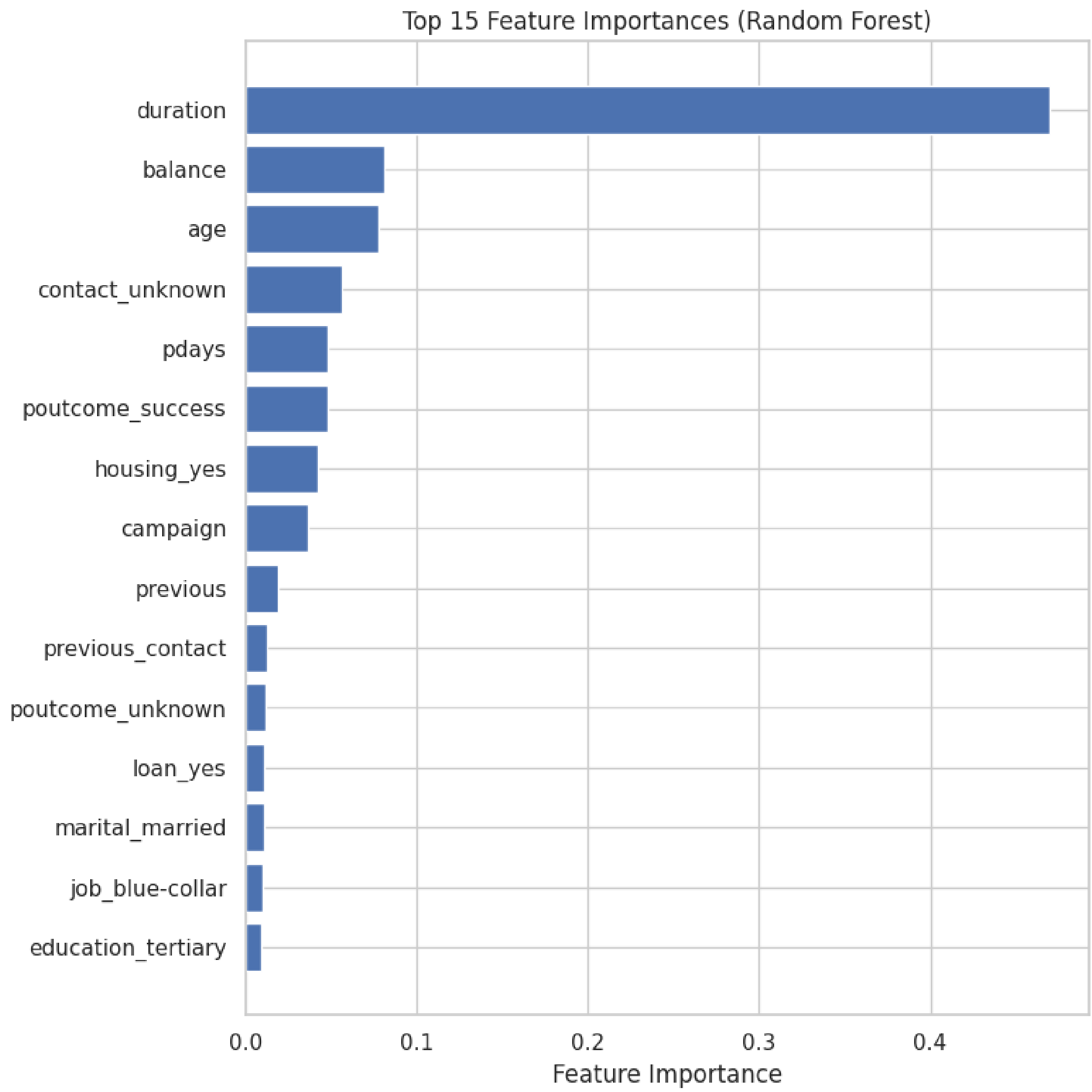

[68]:
```
feature_importance_rf = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf2.feature_importances_
}).sort_values(by='Importance', ascending=False)

feature_importance_rf.head(15)
```

[68]:

| | Feature | Importance |
|---|---|---|
| 2 | duration | 0.469407 |
| 1 | balance | 0.080987 |
| 0 | age | 0.077718 |
| 27 | contact_unknown | 0.055870 |
| 6 | pdays | 0.047498 |
| 29 | poutcome_success | 0.047468 |
| 24 | housing_yes | 0.041986 |
| 4 | campaign | 0.035903 |
| 3 | previous | 0.019197 |
| 5 | previous_contact | 0.012006 |
| 30 | poutcome_unknown | 0.011736 |
| 25 | loan_yes | 0.011049 |
| 21 | marital_married | 0.010674 |

```
7      job_blue-collar    0.009519
19  education_tertiary    0.008935
```

[69]:
```python
#Visualise top feature importances
top_features_rf = feature_importance_rf.head(15)

plt.figure(figsize=(8, 8))
plt.barh(top_features_rf['Feature'], top_features_rf['Importance'])
plt.xlabel('Feature Importance')
plt.title('Top 15 Feature Importances (Random Forest)')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



### 2.0.4 Extreme Gradient Boosting

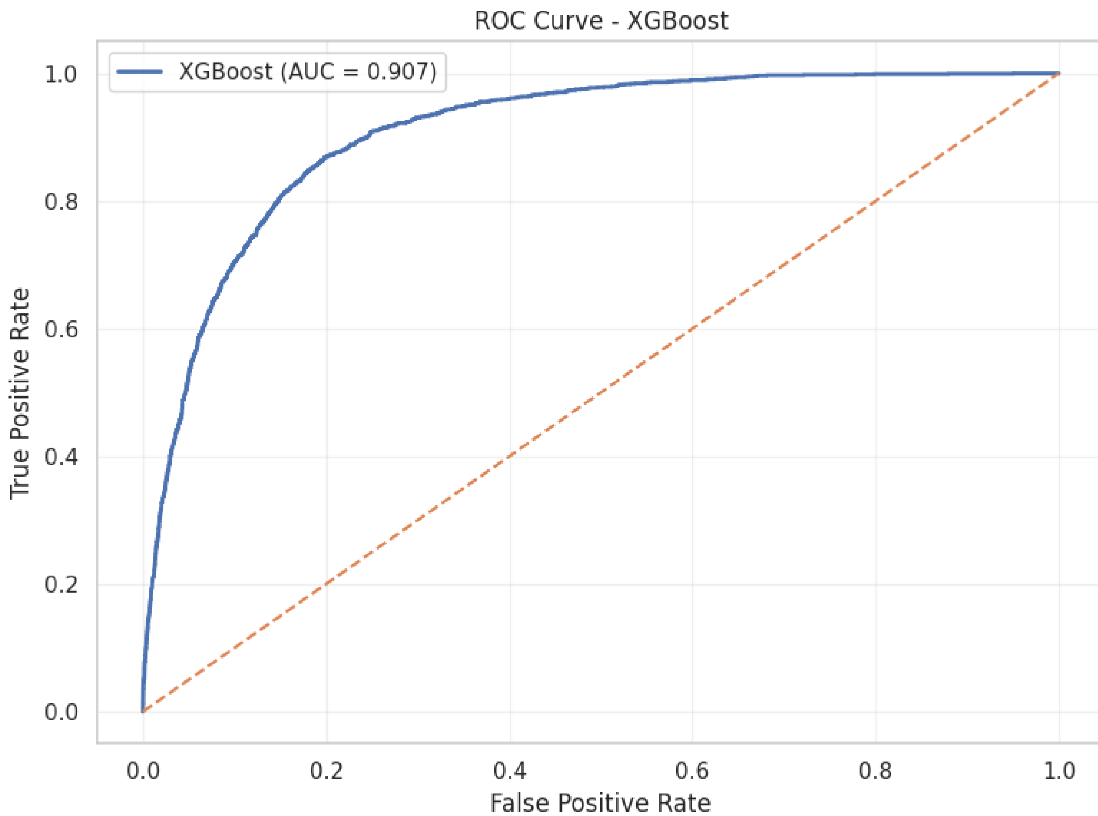
[70]:
```
model_xgb = XGBClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=4,
    subsample=0.8,
    colsample_bytree=0.8,
    tree_method="hist",
    random_state=42,
    eval_metric="logloss"
)

#  Fit with NumPy
model_xgb.fit(
    X_train.to_numpy(dtype=np.float32),
    y_train.to_numpy(dtype=np.float32)
)

#  Predict with NumPy
y_pred_xgb = model_xgb.predict(
    X_test.to_numpy(dtype=np.float32)
)

print("Accuracy of xgboost:", accuracy_score(y_test, y_pred_xgb))
print("\nClassification Report of xgboost:\n", classification_report(y_test,␣
↪y_pred_xgb))
print("\nConfusion Matrix  of xgboost:\n", confusion_matrix(y_test, y_pred_xgb))
```

```
Accuracy of xgboost: 0.9030521969920378

Classification Report of xgboost:
               precision    recall  f1-score   support

           0       0.92      0.97      0.95     11977
           1       0.64      0.39      0.48      1587

    accuracy                           0.90     13564
   macro avg       0.78      0.68      0.71     13564
weighted avg       0.89      0.90      0.89     13564


Confusion Matrix  of xgboost:
 [[11634   343]
 [  972   615]]
```

[71]:
```
# Get predicted probabilities (positive class)
y_prob_xgb = model_xgb.predict_proba(
    X_test.to_numpy(dtype=np.float32)
)[:, 1]

# Compute AUC
auc_xgb = roc_auc_score(y_test, y_prob_xgb)
print("ROC-AUC (XGBoost):", auc_xgb)
```

```
ROC-AUC (XGBoost): 0.907066784535935
```

[72]:
```
# Compute ROC curve
fpr_xgb, tpr_xgb, thresholds = roc_curve(y_test, y_prob_xgb)

plt.figure(figsize=(8,6))
plt.plot(fpr_xgb, tpr_xgb, linewidth=2,
         label=f"XGBoost (AUC = {auc_xgb:.3f})")

# Random baseline
plt.plot([0,1], [0,1], linestyle='--')

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - XGBoost")
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```




[ ]:

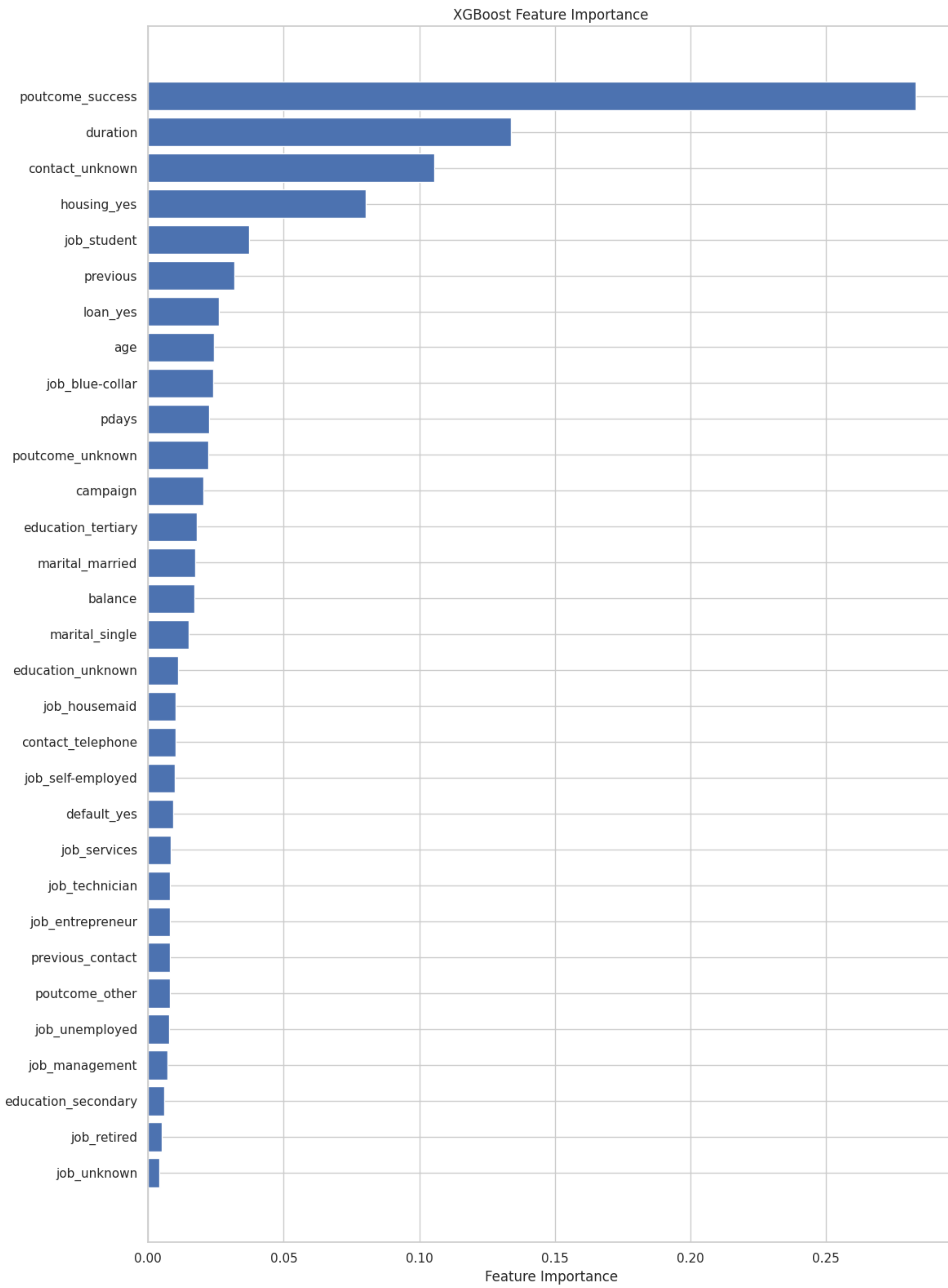
[73]:

```
# Save feature names BEFORE conversion
feature_names = X_train.columns.tolist()

# Convert to numpy
X_train_np = X_train.to_numpy(dtype=np.float32)
X_test_np  = X_test.to_numpy(dtype=np.float32)
y_train_np = y_train.to_numpy(dtype=np.float32)

# Train
model_xgb.fit(X_train_np, y_train_np)

# After training, manually set feature names
model_xgb.get_booster().feature_names = feature_names
```

[74]:

```
# Get importance values
importances = model_xgb.feature_importances_

# Create DataFrame
importance_df = pd.DataFrame({
```

```python
    "Feature": feature_names,
    "Importance": importances
})

# Sort by importance
importance_df = importance_df.sort_values(by="Importance", ascending=True)

# Plot
plt.figure(figsize=(12, 16))   # Bigger figure
plt.barh(importance_df["Feature"], importance_df["Importance"])
plt.xlabel("Feature Importance")
plt.title("XGBoost Feature Importance")
plt.tight_layout()
plt.show()
```


XGBoost Feature Importance
poutcome_success
duration
contact_unknown
housing_yes
job_student
previous
loan_yes
age
job_blue-collar
pdays
poutcome_unknown
campaign
education_tertiary
marital_married
balance
marital_single
education_unknown
job_housemaid
contact_telephone
job_self-employed
default_yes
job_services
job_technician
job_entrepreneur
previous_contact
poutcome_other
job_unemployed
job_management
education_secondary
job_retired
job_unknown
0.00
0.05
0.10
0.15
0.20
0.25
Feature Importance

# 3 Research Question 3: How early in a campaign can reliable predictions be made?

#### 3.0.1 Scenario 1: the early-stage, pre-campaign or start of the campaign

[ ]:

```

```

[75]:

```
rq3_vars_sc1=["age", "job","education", "marital", "balance", "default",␣
↪"housing","loan","y"]
dfrq3_sc1=df[rq3_vars_sc1]
#Ecoding all object variables into dummy variable.
cat_cols_sc1 = dfrq3_sc1.select_dtypes(include='object').columns

dfrq3_sc1_enc = pd.get_dummies(dfrq3_sc1, columns=cat_cols_sc1, drop_first=True)

print(dfrq3_sc1_enc.shape)
#print(dfrq3_sc1_enc.head())

#Independent and depend variables split
X_sc1 = dfrq3_sc1_enc.drop(columns=['y_yes'])
y_sc1 = dfrq3_sc1_enc['y_yes']
#Train-test split (stratified)
#from sklearn.model_selection import train_test_split

X_train_sc1, X_test_sc1, y_train_sc1, y_test_sc1 = train_test_split( X_sc1,␣
↪y_sc1,
    test_size=0.3,
    stratify=y_sc1,
    random_state=42
)

log_reg_pipeline_sc1 = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),   # handles NaNs (pdays)
    ('scaler', RobustScaler()),                      # robust to outliers
    ('model', LogisticRegression(
        max_iter=1000,
        class_weight='balanced',
        random_state=42
    ))
])
log_reg_pipeline_sc1.fit(X_train_sc1, y_train_sc1)

#model evaluation
y_pred_sc1 = log_reg_pipeline_sc1.predict(X_test_sc1)

cm_sc1 = confusion_matrix(y_test_sc1, y_pred_sc1)
print("The confusion matrix of Scenario 1 is : \n",cm_sc1)
```

```
print("The classification report of Scenario 1 is :
 ↪\n",classification_report(y_test_sc1, y_pred_sc1))

#ROC-AUC
#from sklearn.metrics import roc_auc_score

y_prob_sc1 = log_reg_pipeline_sc1.predict_proba(X_test_sc1)[:, 1]
roc_auc_sc1 = roc_auc_score(y_test_sc1, y_prob_sc1)

print("The ROC-AUC of Scenario 1 is:", roc_auc_sc1)
```

```
(45211, 22)
The confusion matrix of Scenario 1 is :
 [[7543 4434]
 [ 597  990]]
The classification report of Scenario 1 is :
               precision    recall  f1-score   support

           0       0.93      0.63      0.75     11977
           1       0.18      0.62      0.28      1587

    accuracy                           0.63     13564
   macro avg       0.55      0.63      0.52     13564
weighted avg       0.84      0.63      0.70     13564

The ROC-AUC of Scenario 1 is: 0.6689852515578194
```

#### 3.0.2 Scenario 2 : mid-stage or during the campaign stage

[ ]:

```
```

[77]:

```
rq3_sc2_vars=["age", "job","education", "marital", "balance", "default",
 ↪"housing","loan",'previous_contact','pdays',"poutcome","y"]
dfrq3_sc2=df[rq3_sc2_vars]
#Ecoding all object variables into dummy variable.
cat_cols_sc2= dfrq3_sc2.select_dtypes(include='object').columns

dfrq3_sc2_enc = pd.get_dummies(dfrq3_sc2, columns=cat_cols_sc2, drop_first=True)

print(dfrq3_sc2_enc.shape)
#print(dfrq3_sc2_enc.head())

#Independent and depend variables split
X_sc2 = dfrq3_sc2_enc.drop(columns=['y_yes'])
y_sc2 = dfrq3_sc2_enc['y_yes']
#Train-test split (stratified)
```

```python
X_train_sc2, X_test_sc2, y_train_sc2, y_test_sc2 = train_test_split( X_sc2,␣
 ↪y_sc2,
    test_size=0.3,
    stratify=y_sc2,
    random_state=42
)

log_reg_pipeline_sc2 = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),   # handles NaNs (pdays)
    ('scaler', RobustScaler()),                      # robust to outliers
    ('model', LogisticRegression(
        max_iter=1000,
        class_weight='balanced',
        random_state=42
    ))
])
log_reg_pipeline_sc2.fit(X_train_sc2, y_train_sc2)

#model evaluation
y_pred_sc2 = log_reg_pipeline_sc2.predict(X_test_sc2)

cm_sc2 = confusion_matrix(y_test_sc2, y_pred_sc2)
print("The confusion matrix of Scenario 2 is: \n",cm_sc2)
print("The classification report of Scenario 2 is:␣
 ↪\n",classification_report(y_test_sc2, y_pred_sc2))

#ROC-AUC
#from sklearn.metrics import roc_auc_score

y_prob_sc2 = log_reg_pipeline_sc2.predict_proba(X_test_sc2)[:, 1]
roc_auc_sc2 = roc_auc_score(y_test_sc2, y_prob_sc2)

print("The ROC-AUC of Scenario 2 is:", roc_auc_sc2)
```

```
(45211, 27)
The confusion matrix of Scenario 2 is:
 [[8707 3270]
 [ 643  944]]
The classification report of Scenario 2 is:
               precision    recall  f1-score   support

           0       0.93      0.73      0.82     11977
           1       0.22      0.59      0.33      1587

    accuracy                           0.71     13564
   macro avg       0.58      0.66      0.57     13564
```

```
weighted avg       0.85      0.71      0.76     13564

The ROC-AUC of Scenario 2 is: 0.7218435997287176
```

### 3.0.3 Scenario 3 the late stage or active engagement stage

[ ]:
```
#rq1_vars=["age", "job","education", "marital", "balance", "default",␣
↪"housing","loan", "contact","duration", "poutcome", "previous","campaign"␣
↪,'previous_contact','pdays',"y"]
```

[78]:
```
rq3_sc3_vars=["age", "job","education", "marital", "balance", "default",␣
↪"housing","loan",'previous_contact','pdays',"poutcome","duration","campaign",␣
↪"contact","y"]
dfrq3_sc3=df[rq3_sc3_vars]
#Ecoding all object variables into dummy variable.
cat_cols_sc3= dfrq3_sc3.select_dtypes(include='object').columns

dfrq3_sc3_enc = pd.get_dummies(dfrq3_sc3, columns=cat_cols_sc3, drop_first=True)

print(dfrq3_sc3_enc.shape)
#print(dfrq3_sc2_enc.head())

#Independent and depend variables split
X_sc3 = dfrq3_sc3_enc.drop(columns=['y_yes'])
y_sc3 = dfrq3_sc3_enc['y_yes']
#Train-test split (stratified)


X_train_sc3, X_test_sc3, y_train_sc3, y_test_sc3 = train_test_split( X_sc3,␣
↪y_sc3,
    test_size=0.3,
    stratify=y_sc3,
    random_state=42
)

log_reg_pipeline_sc3 = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),   # handles NaNs (pdays)
    ('scaler', RobustScaler()),                      # robust to outliers
    ('model', LogisticRegression(
        max_iter=1000,
        class_weight='balanced',
        random_state=42
    ))
])
log_reg_pipeline_sc3.fit(X_train_sc3, y_train_sc3)

#model evaluation
```

```
y_pred_sc3 = log_reg_pipeline_sc3.predict(X_test_sc3)

cm_sc3 = confusion_matrix(y_test_sc3, y_pred_sc3)
print("The confusion matrix of Scenario 3 is: \n",cm_sc3)
print("The classification report of Scenario 3 is:
↪\n",classification_report(y_test_sc3, y_pred_sc3))

#ROC-AUC
#from sklearn.metrics import roc_auc_score

y_prob_sc3 = log_reg_pipeline_sc3.predict_proba(X_test_sc3)[:, 1]
roc_auc_sc3 = roc_auc_score(y_test_sc3, y_prob_sc3)

print("The ROC-AUC of Scenario 3 is:", roc_auc_sc3)
```

```
(45211, 31)
The confusion matrix of Scenario 3 is:
 [[10005  1972]
 [  340  1247]]
The classification report of Scenario 3 is:
               precision    recall  f1-score   support

           0       0.97      0.84      0.90     11977
           1       0.39      0.79      0.52      1587

    accuracy                           0.83     13564
   macro avg       0.68      0.81      0.71     13564
weighted avg       0.90      0.83      0.85     13564

The ROC-AUC of Scenario 3 is: 0.8918576294545643
```

[ ]:

```

```

# 4 Research Question 4: Are prediction errors biased across demographic groups?

[80]:

```
#Define Variables
rq4_vars = [
    "age", "job","education", "marital", "balance", "default",
    "housing","loan", "contact","duration", "poutcome",
    "previous","campaign",'previous_contact','pdays'
]

X_rq4 = df[rq4_vars]  #X
y_rq4 = df["y"].map({"yes":1, "no":0})
#Train/Test Split
```

```python
#from sklearn.model_selection import train_test_split

X_train_rq4, X_test_rq4, y_train_rq4, y_test_rq4 = train_test_split(
    X_rq4, y_rq4,
    test_size=0.3,
    stratify=y_rq4,
    random_state=42
)
#One-Hot Encode Categorical Variables
X_train_enc_rq4 = pd.get_dummies(X_train_rq4, drop_first=True)
X_test_enc_rq4 = pd.get_dummies(X_test_rq4, drop_first=True)

# Align columns to avoid mismatch
X_train_enc_rq4, X_test_enc_rq4 = X_train_enc_rq4.align(
    X_test_enc_rq4,
    join='left',
    axis=1,
    fill_value=0
)
```

[81]:

```python
#Logistic Regression Pipeline
#from sklearn.pipeline import Pipeline
#from sklearn.impute import SimpleImputer
#from sklearn.preprocessing import RobustScaler
#from sklearn.linear_model import LogisticRegression

log_reg_pipeline_rq4 = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', RobustScaler()),
    ('model', LogisticRegression(
        max_iter=1000,
        class_weight='balanced',
        random_state=42
    ))
])

log_reg_pipeline_rq4.fit(X_train_enc_rq4, y_train_rq4)

y_pred_rq4 = log_reg_pipeline_rq4.predict(X_test_enc_rq4)

#Create Results DataFrame:
#For bias analysis, we need demographic variables from TEST SET.
results_rq4  = X_test_rq4 .copy()
results_rq4 ["y_true"] = y_test_rq4
results_rq4 ["y_pred"] = y_pred_rq4

#Function to Compute FPR and FNR
```

```python
#from sklearn.metrics import confusion_matrix
#import pandas as pd

def error_rates_by_group(data, group_var):

    output = []

    for group in data[group_var].unique():

        subset = data[data[group_var] == group]

        if len(subset) > 0:
            tn, fp, fn, tp = confusion_matrix(
                subset["y_true"],
                subset["y_pred"],
                labels=[0,1]
            ).ravel()

            fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
            fnr = fn / (fn + tp) if (fn + tp) > 0 else 0

            output.append([group, fpr, fnr])

    return pd.DataFrame(output, columns=[group_var, "False Positive Rate",␣
↪"False Negative Rate"])
```

### 4.0.1 Bias Analysis

[82]:
```python
#Marital Status
print("\n Bias analysis results for Marital Status␣
↪\n",error_rates_by_group(results_rq4, "marital"))
#Education
print(" \n Bias analysis results for Education ␣
↪\n",error_rates_by_group(results_rq4, "education"))
#Job
print("\n Bias analysis results for Job  \n",error_rates_by_group(results_rq4,␣
↪"job"))

#Age Groups
results_rq4["age_group"] = pd.cut(
    results_rq4["age"],
    bins=[0,30,40,50,60,100],
    labels=["<30","30-40","40-50","50-60","60+"]
)
```

```python
print("\n Bias analysis results for Age groups  \n",
  ↪error_rates_by_group(results_rq4, "age_group"))
```

```
 Bias analysis results for Marital Status
     marital  False Positive Rate  False Negative Rate
0  divorced             0.149286             0.222857
1    single             0.234251             0.171480
2   married             0.136445             0.243590

 Bias analysis results for Education
    education  False Positive Rate  False Negative Rate
0  secondary             0.144928             0.256233
1   tertiary             0.230635             0.166392
2    primary             0.096143             0.225989
3    unknown             0.205628             0.209877

 Bias analysis results for Job
              job  False Positive Rate  False Negative Rate
0     blue-collar             0.091355             0.304147
1      unemployed             0.211180             0.222222
2   self-employed             0.131336             0.173913
3        services             0.106074             0.320755
4         retired             0.385069             0.090909
5      technician             0.151755             0.253731
6          admin.             0.174436             0.245810
7      management             0.210051             0.170604
8       housemaid             0.102894             0.277778
9    entrepreneur             0.116505             0.322581
10        student             0.591837             0.030769
11        unknown             0.200000             0.428571

 Bias analysis results for Age groups
  age_group  False Positive Rate  False Negative Rate
0     30-40             0.158148             0.218182
1     40-50             0.130719             0.252560
2       60+             0.661376             0.163399
3       <30             0.218150             0.204473
4     50-60             0.141834             0.215827
```

[ ]:

[ ]: