Capston Project - Marketing campaign

February 4, 2026

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

[ ]:

0.1 Univariate analysis for categorical and object variables

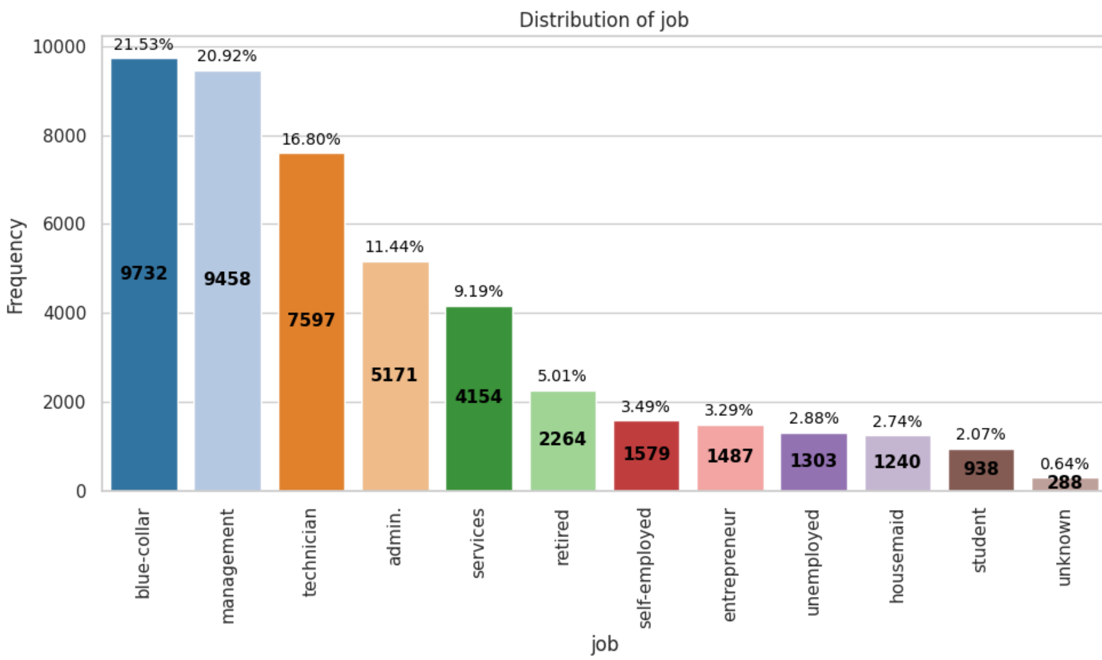
[4]:

```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')
```

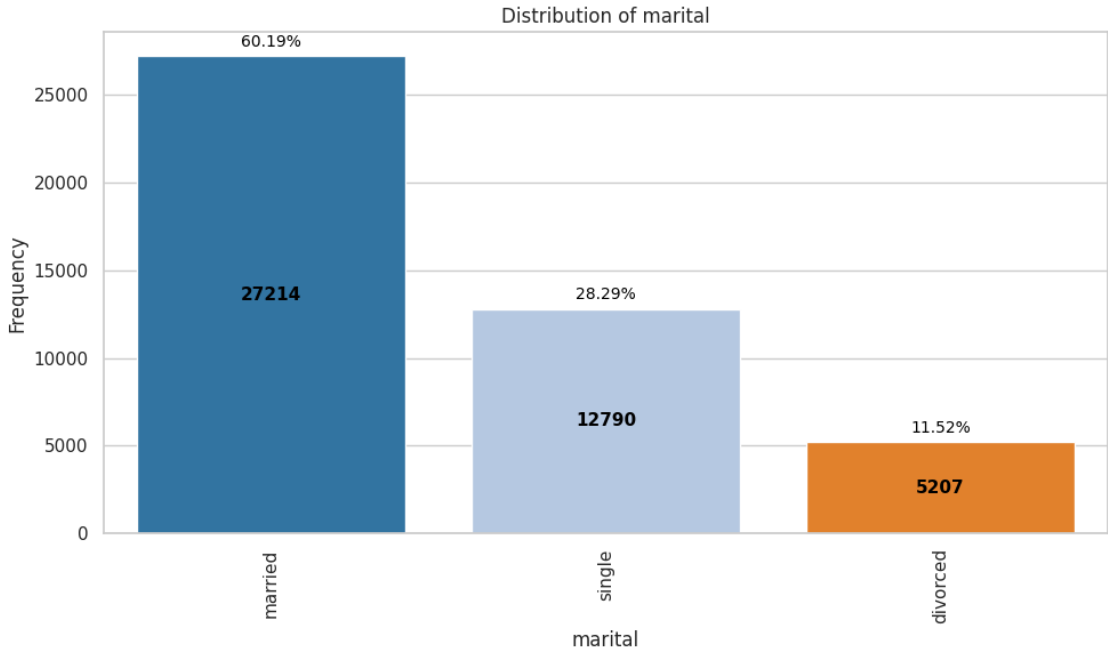
```
plt.title(f'Distribution of {column_name}')
plt.xlabel(column_name)
plt.xticks(rotation=90)
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

[ ]:

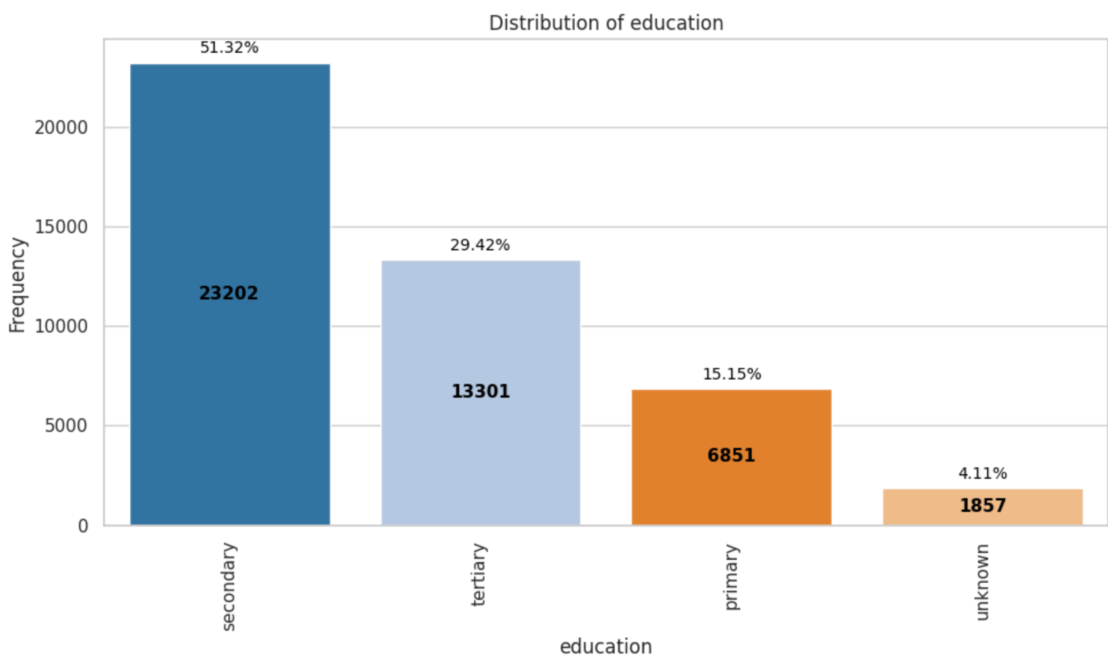
[5]:
```
plot_object(df, "job")
```



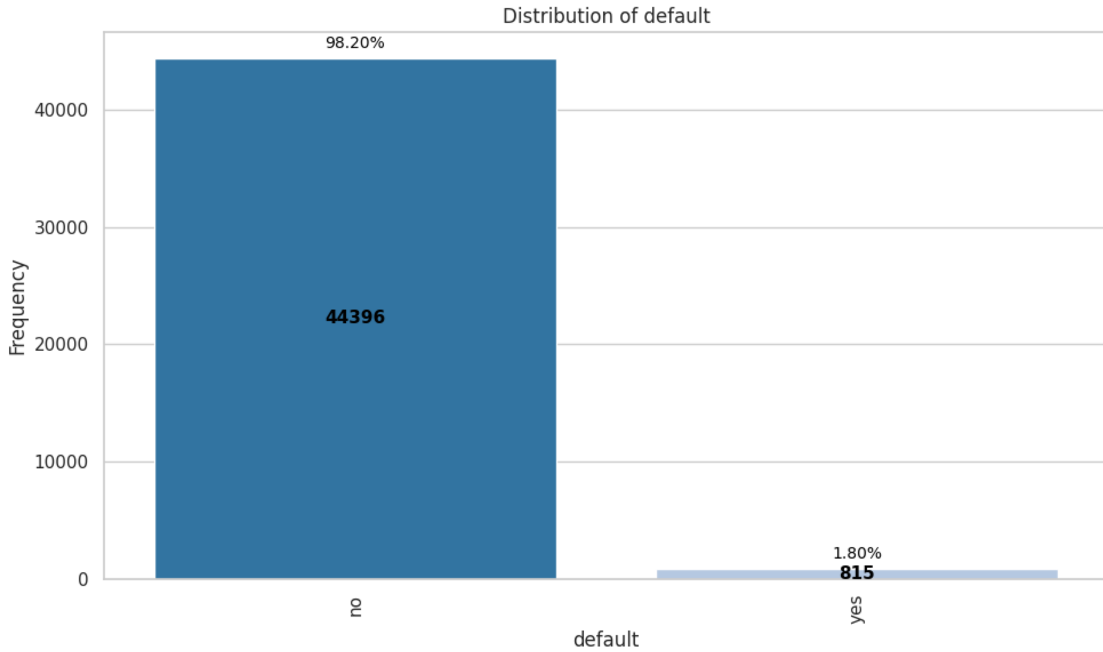
[6]:
```
plot_object(df, 'marital')
```



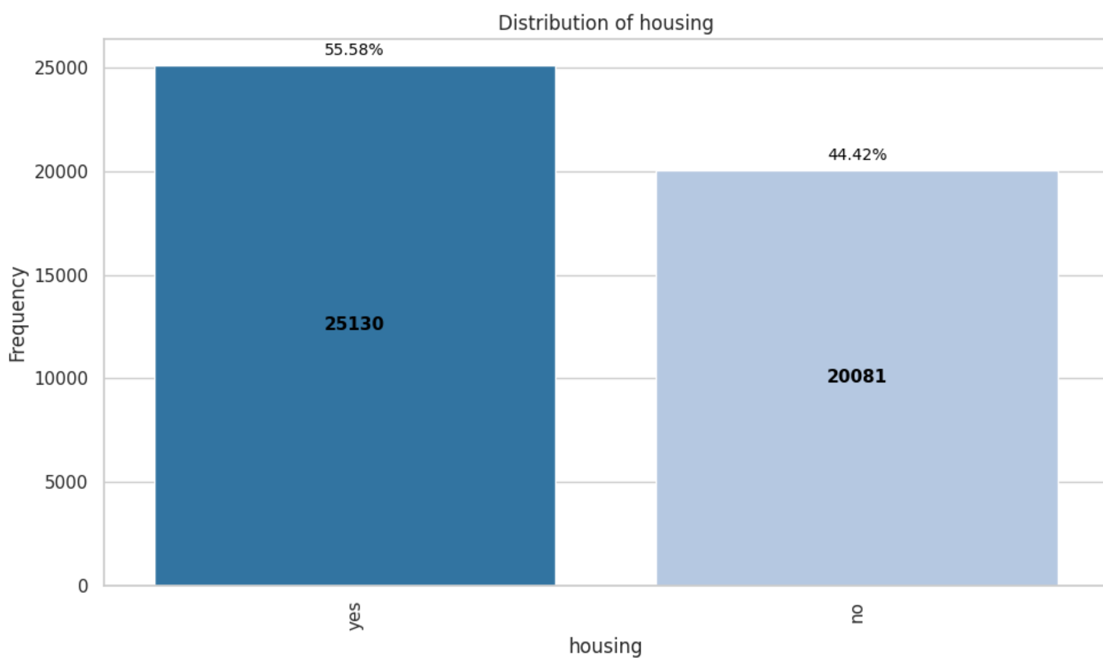
[7]:
```
plot_object(df, 'education')
```



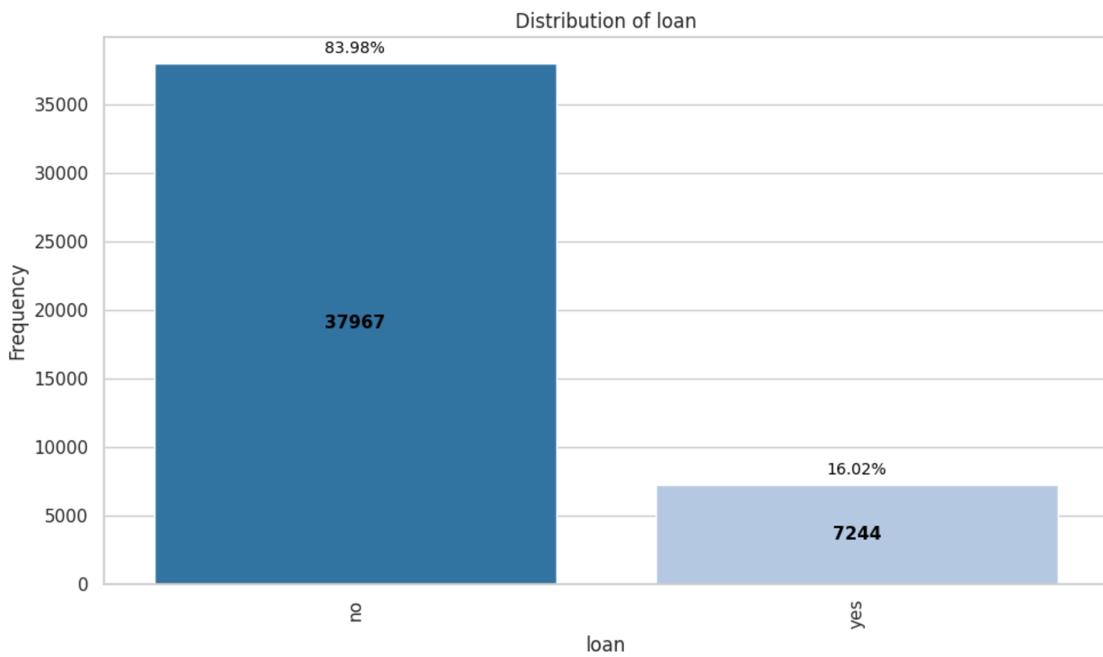
[8]:
```
plot_object(df, 'default')
```



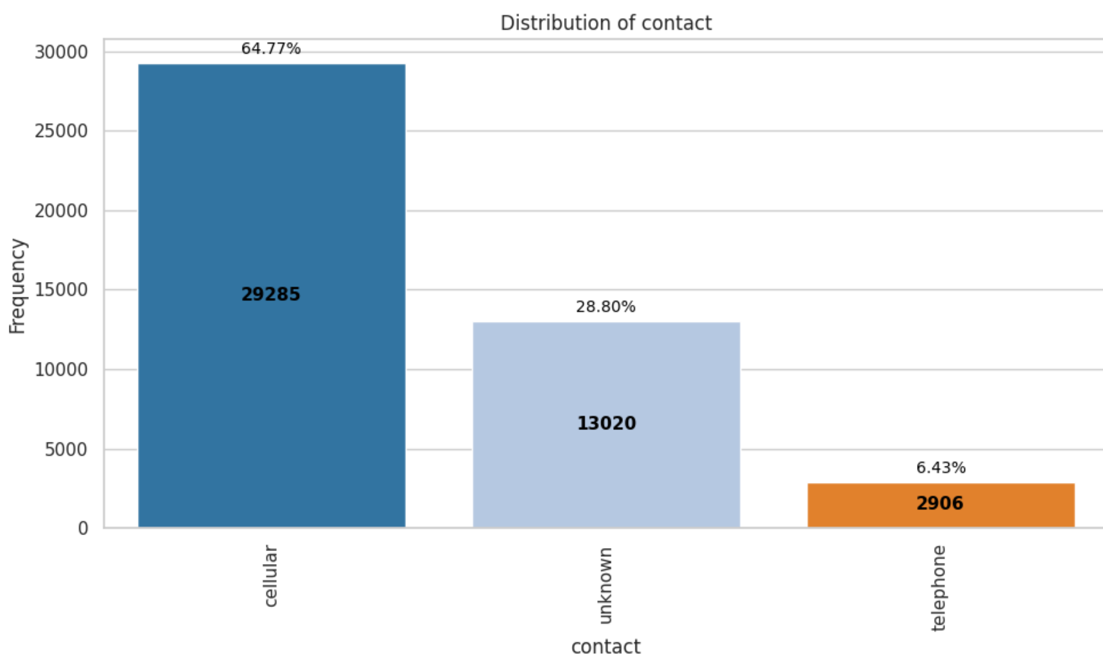
[9]:
```
plot_object(df, 'housing')
```



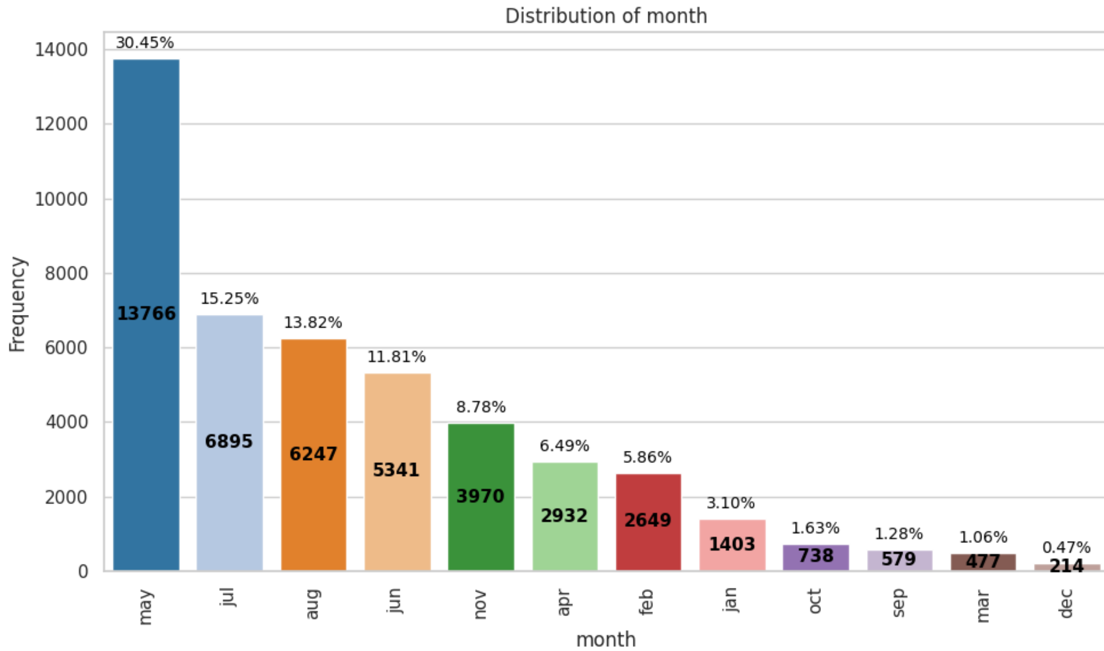
[10]:
```
plot_object(df, 'loan')
```



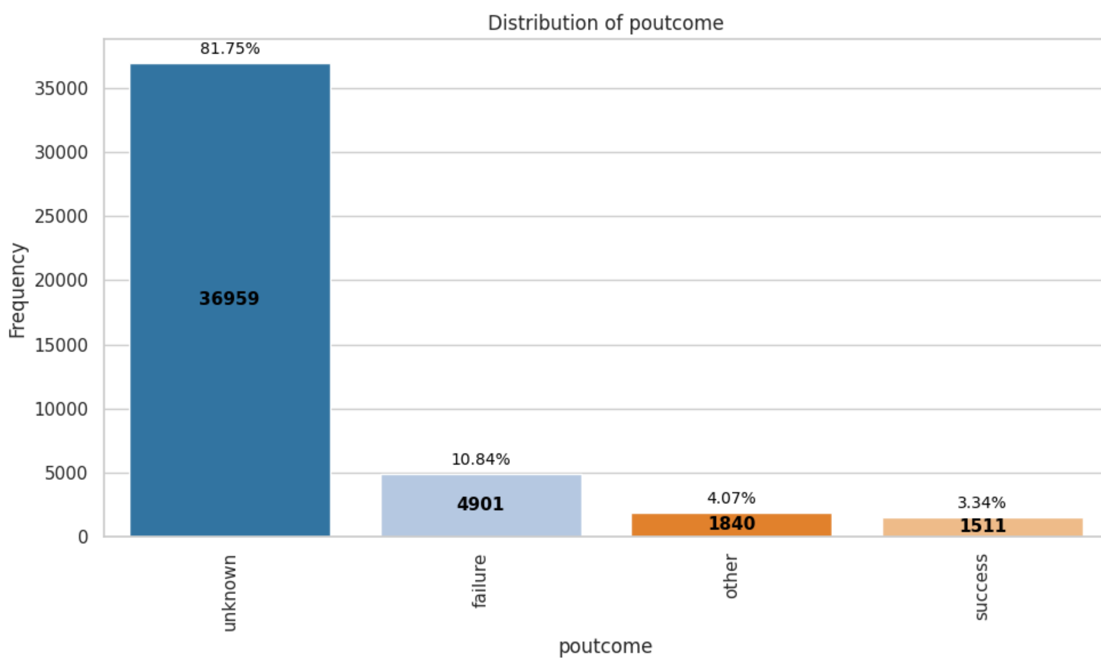
[11]:
```
plot_object(df, 'contact')
```



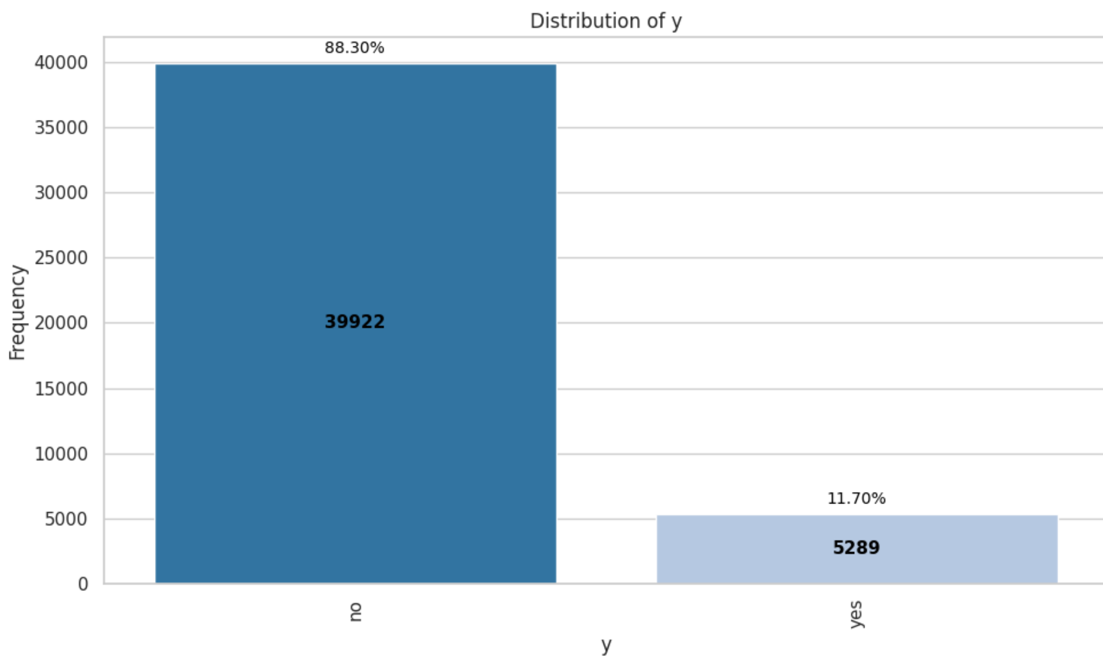
[12]:
```
plot_object(df, 'month')
```



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



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



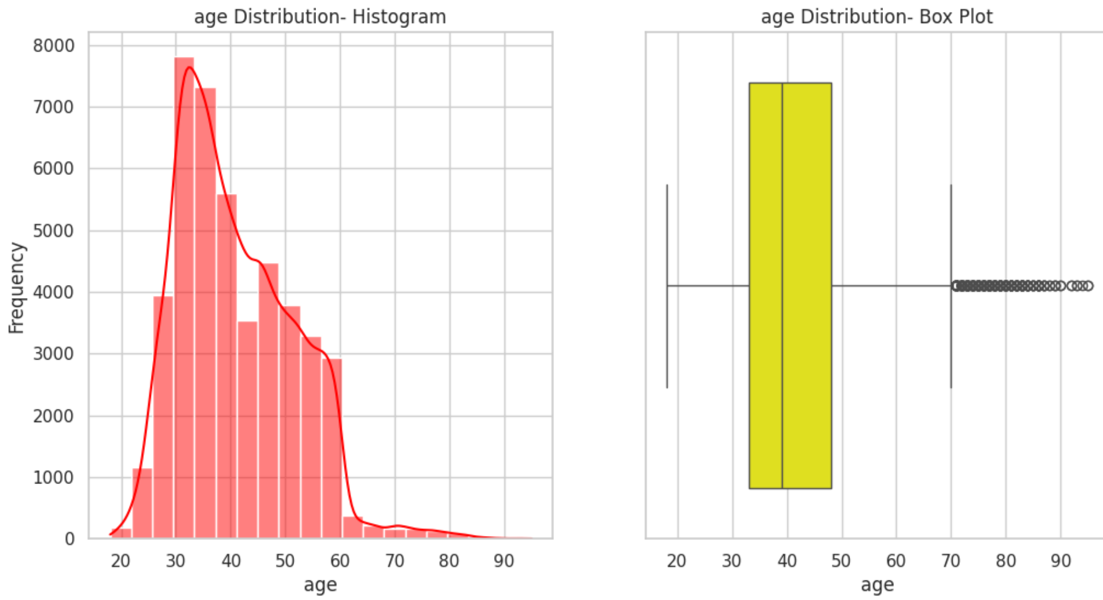
0.2 Univariate Analysis for numerical variables

[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()
```

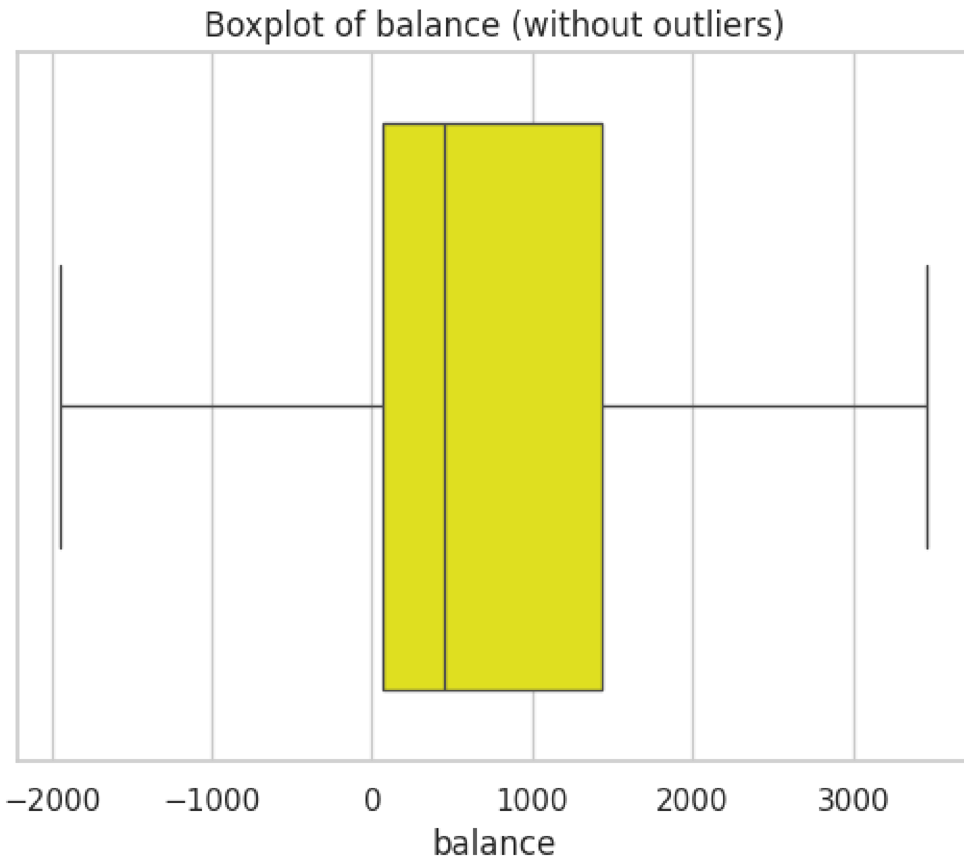
[17]:

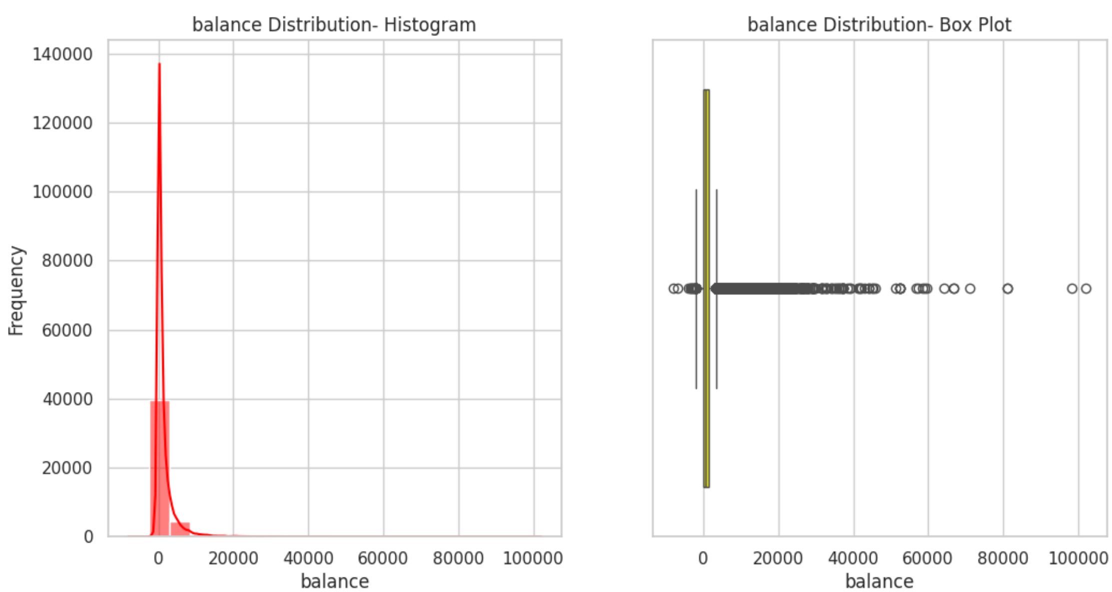
```
cont_plot(df,'age' )
```



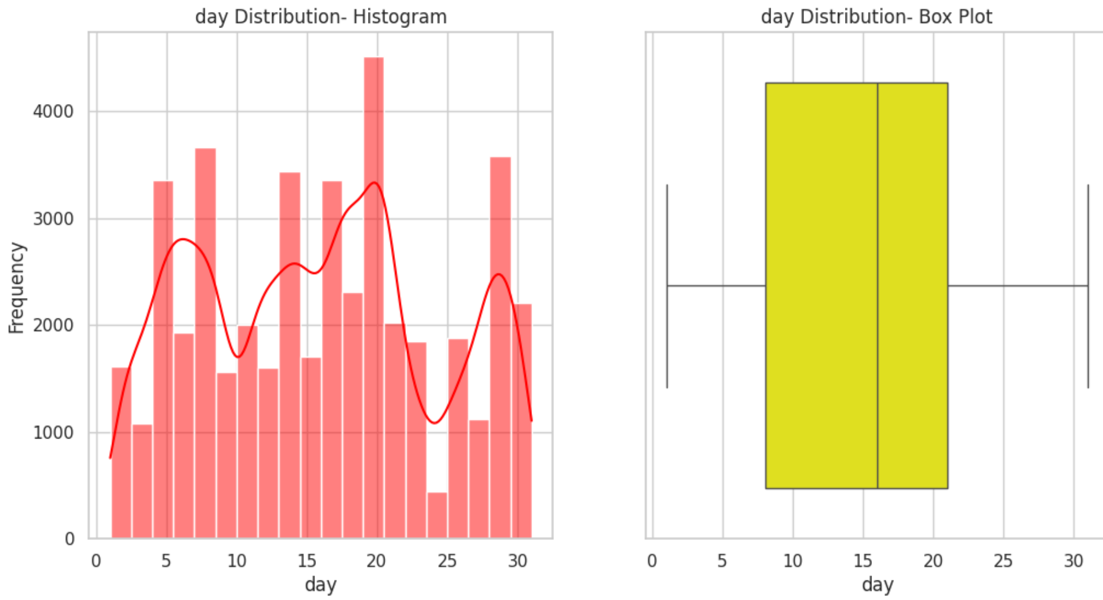
[18]:

```
# Boxplot without outliers
NoOutlier(df,"balance")
cont_plot(df,'balance' )
```

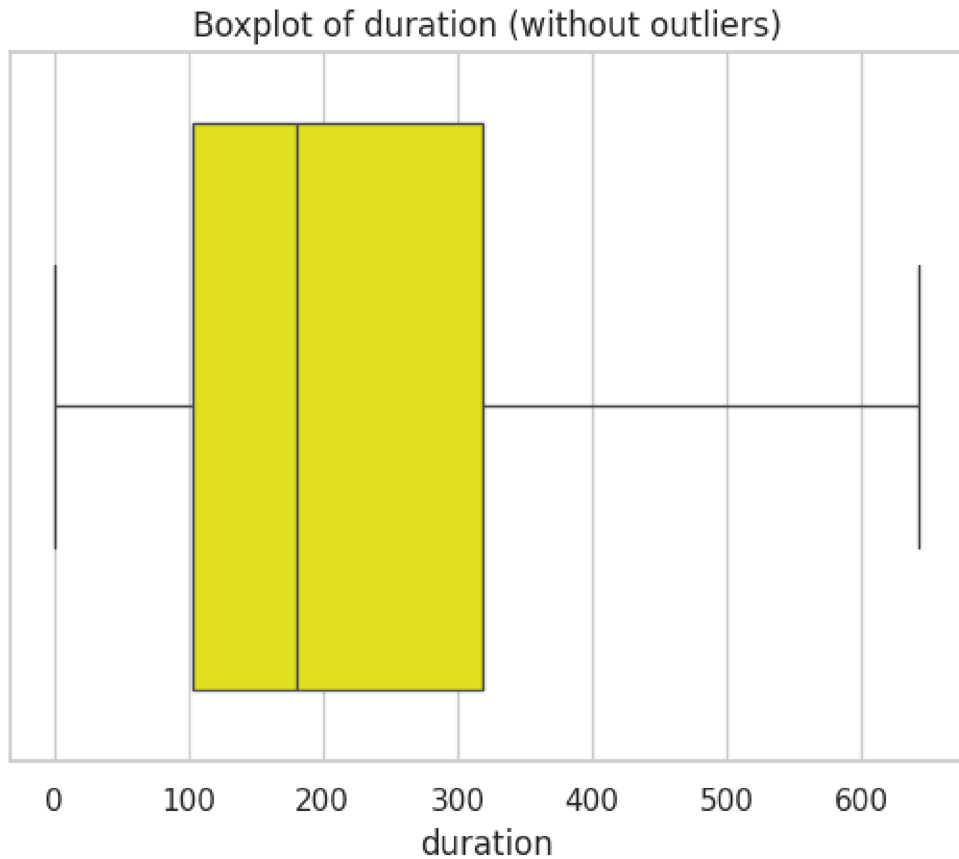

Boxplot of balance (without outliers)
−2000
−1000
0
1000
2000
3000
balance

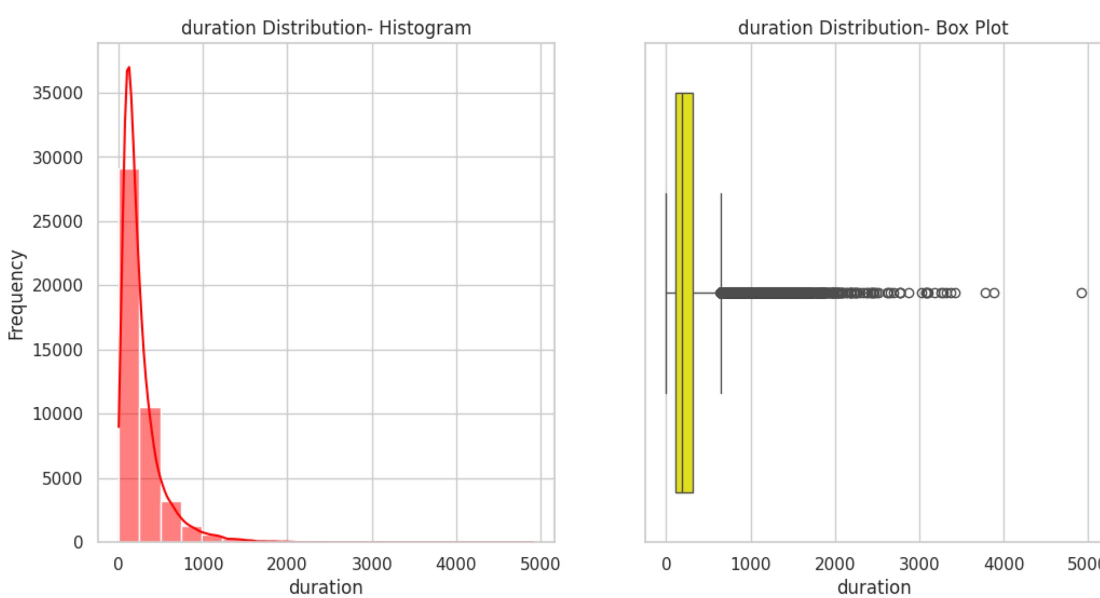

balance Distribution- Histogram
140000
120000
100000
80000
60000
40000
20000
0
Frequency
0
20000
40000
60000
80000
100000
balance
balance Distribution- Box Plot
0
20000
40000
60000
80000
100000
balance

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



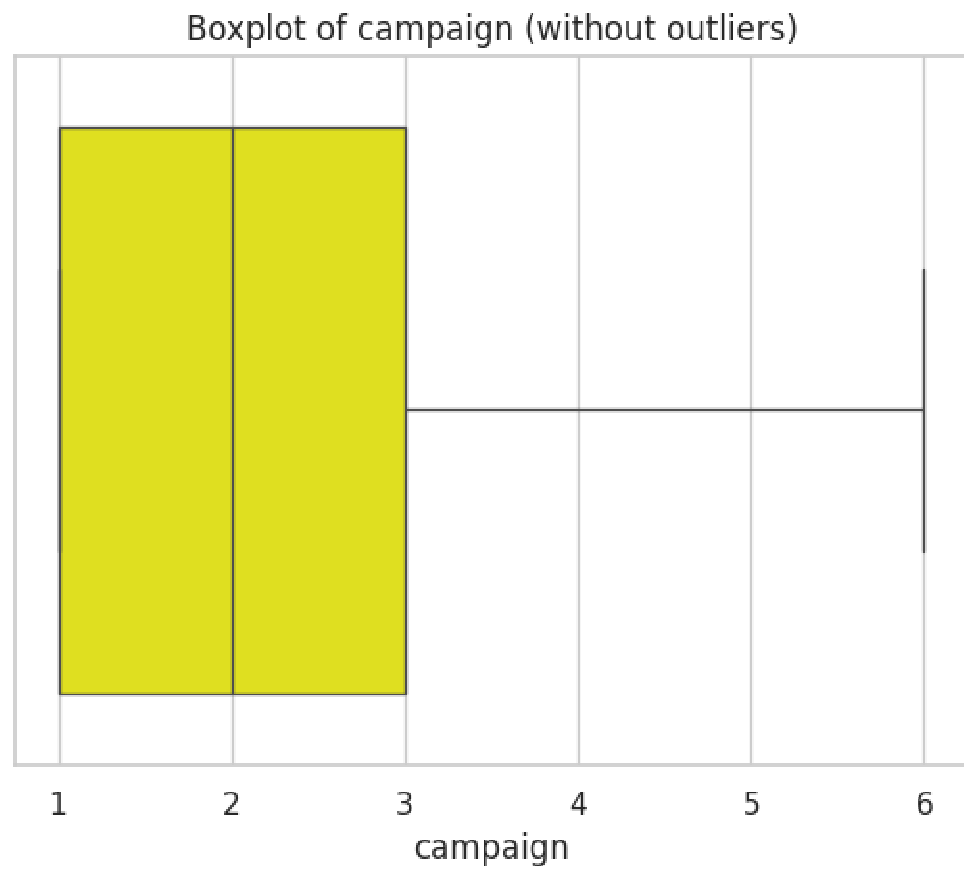
[20]:
```
NoOutlier(df,"duration")
cont_plot(df,'duration' )
```

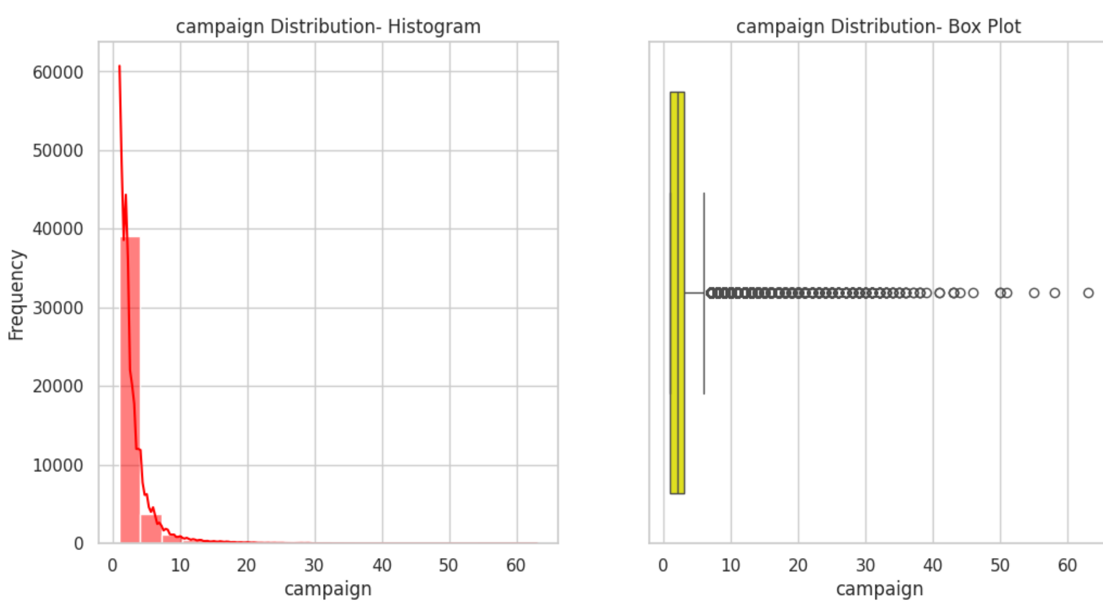

Boxplot of duration (without outliers)
0
100
200
300
400
500
600
duration


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]:

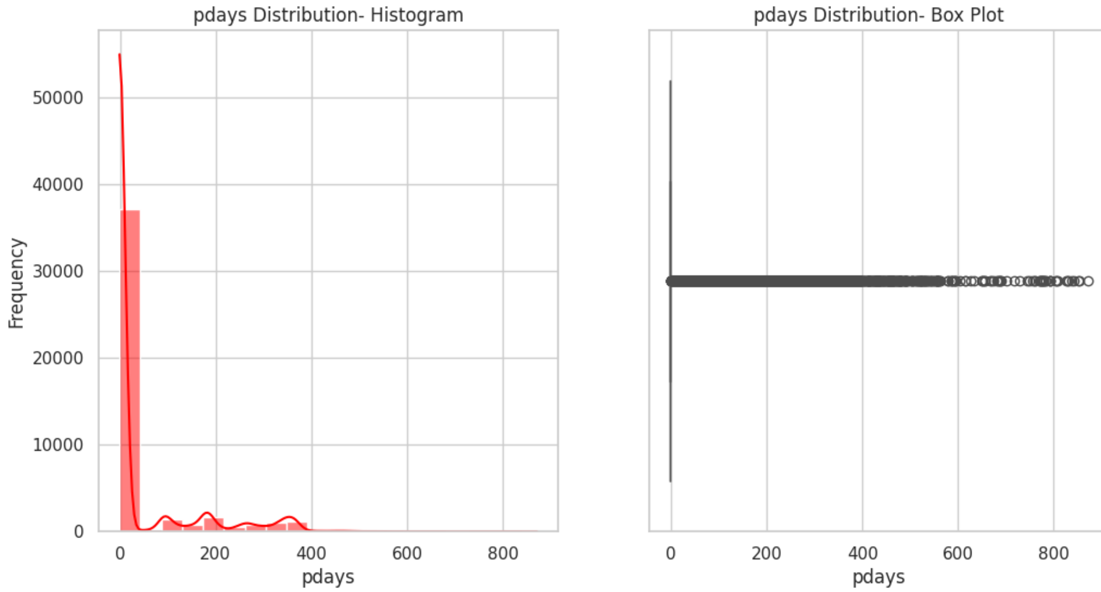
```
NoOutlier(df,"campaign")
cont_plot(df,'campaign' )
```



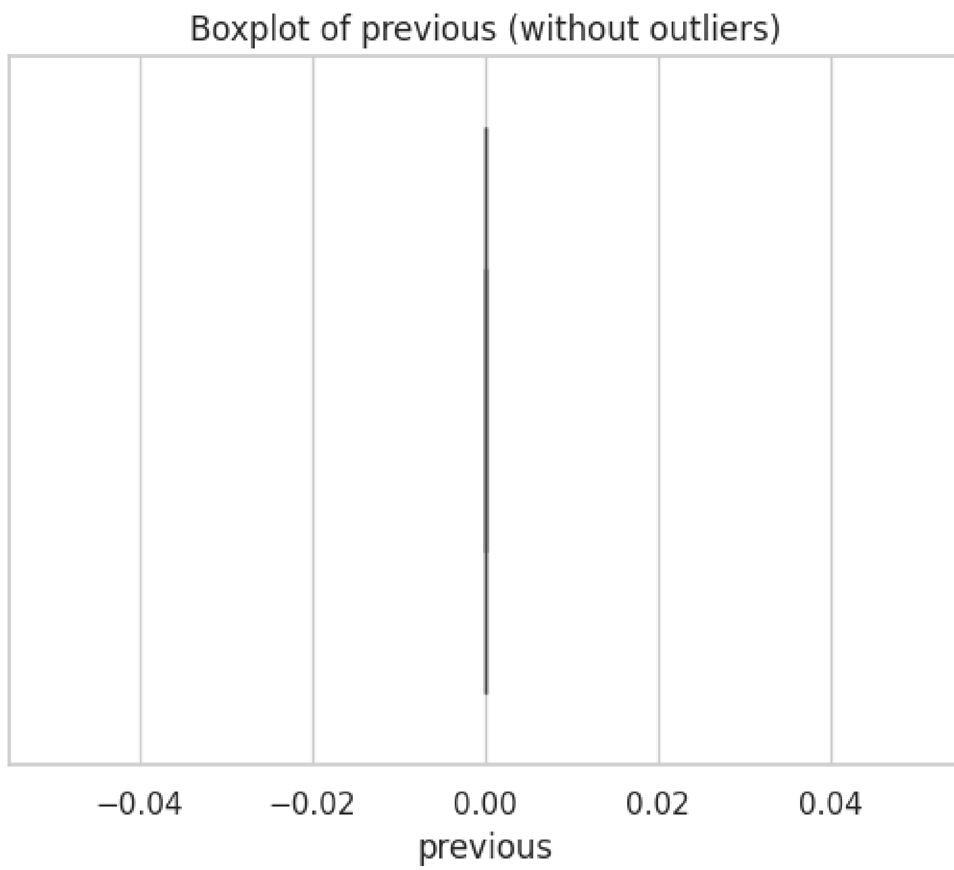


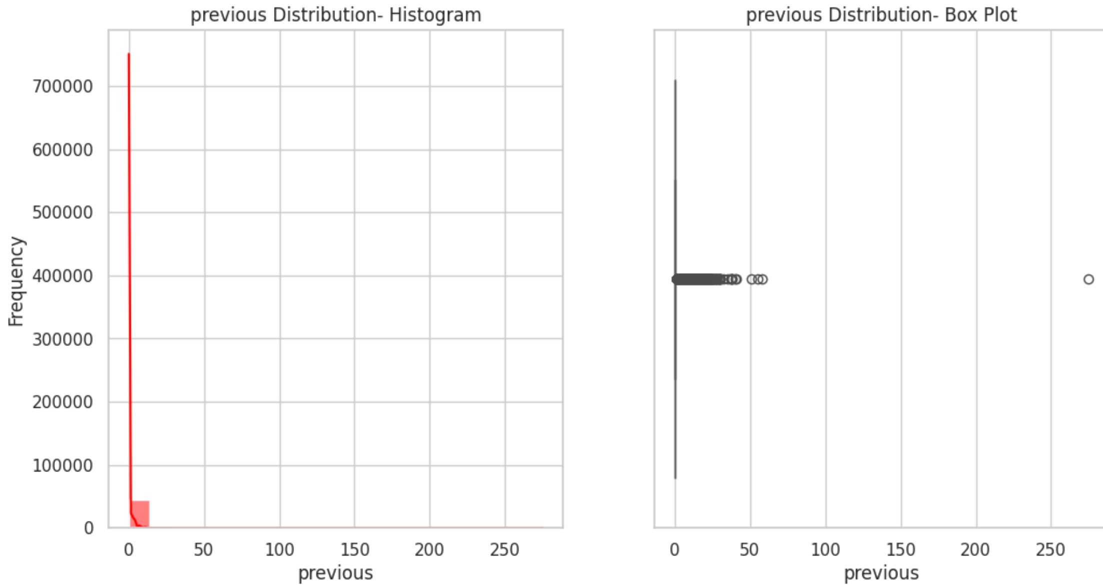
[22]:

```
NoOutlier(df,"pdays")
cont_plot(df,'pdays' )
```

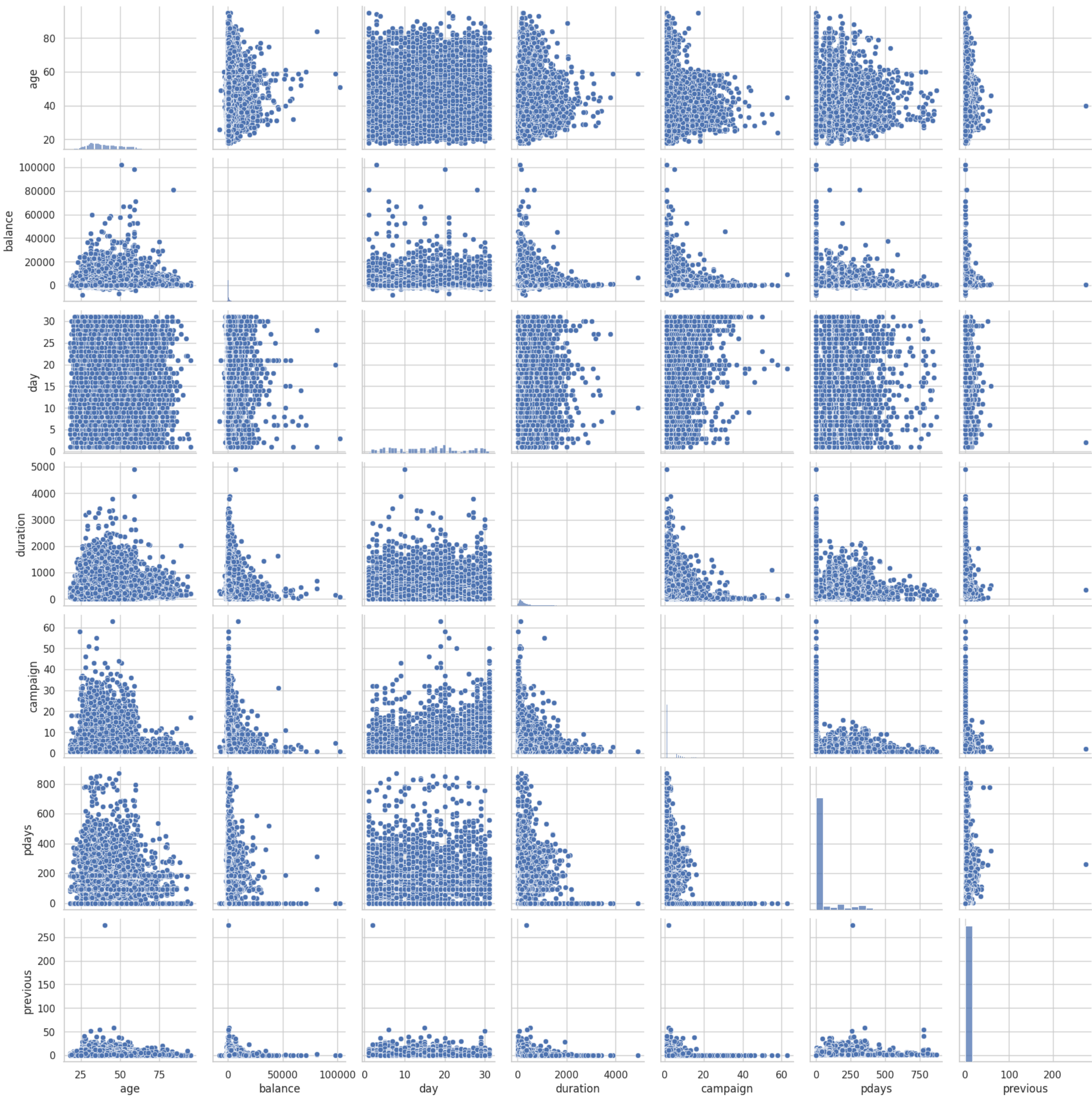


[23]:
```
NoOutlier(df,"previous")
cont_plot(df,'previous' )
```

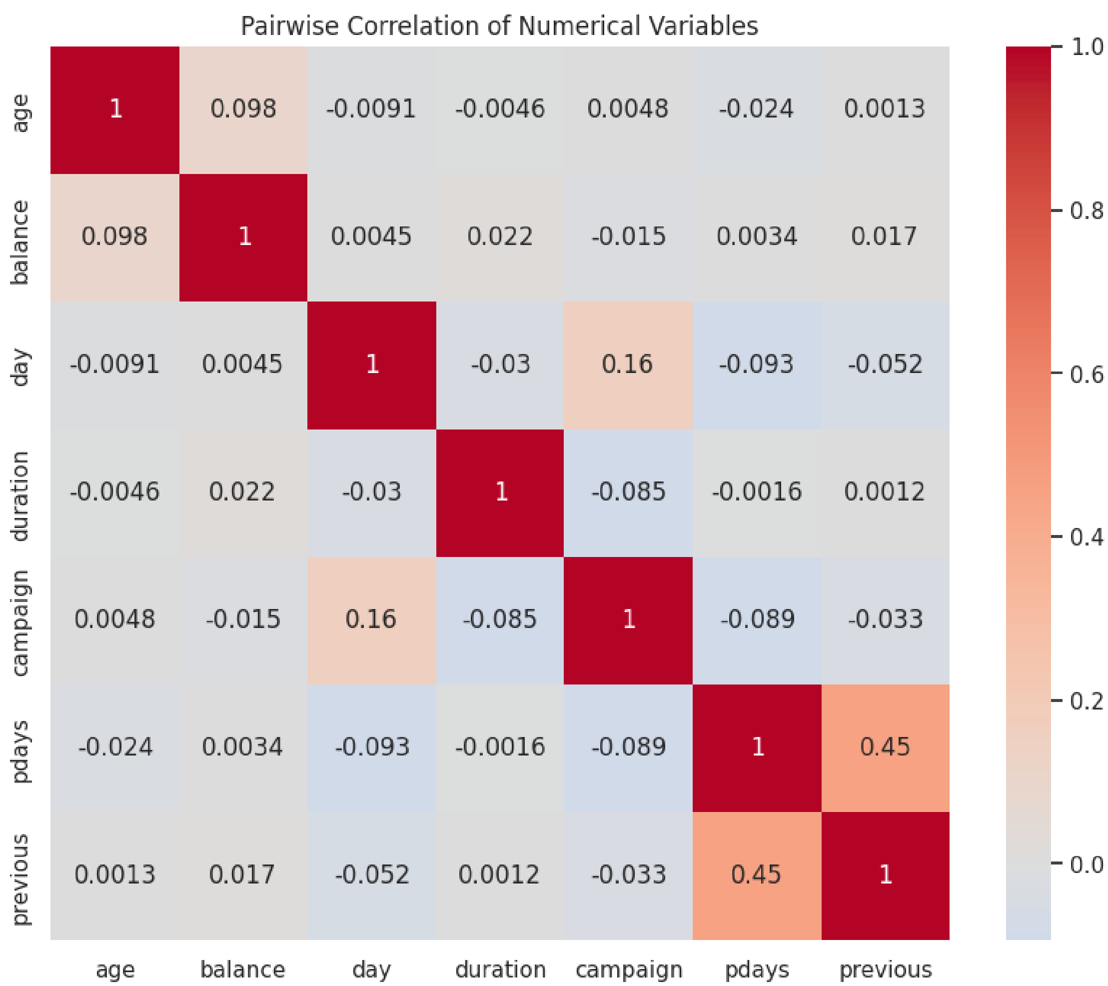




[24]:
```
import seaborn as sns
import matplotlib.pyplot as plt

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



[25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# 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']
```

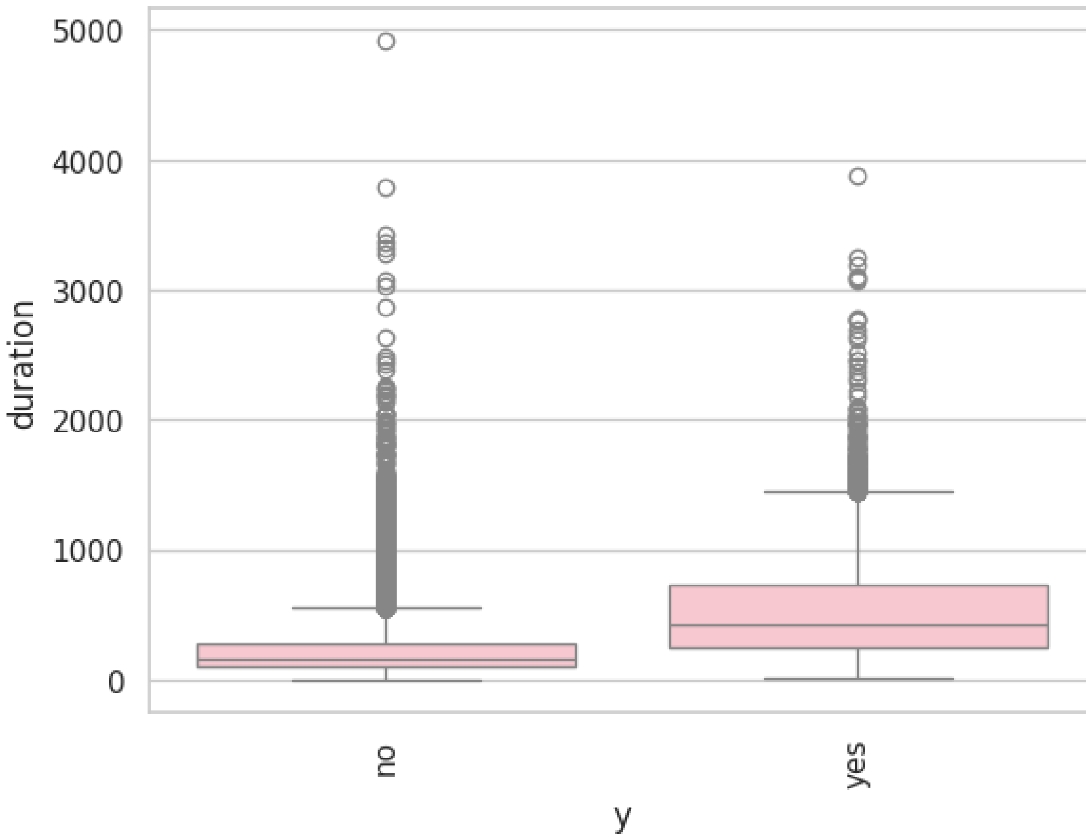
[ ]:
```

```

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

print(object_columns)
```

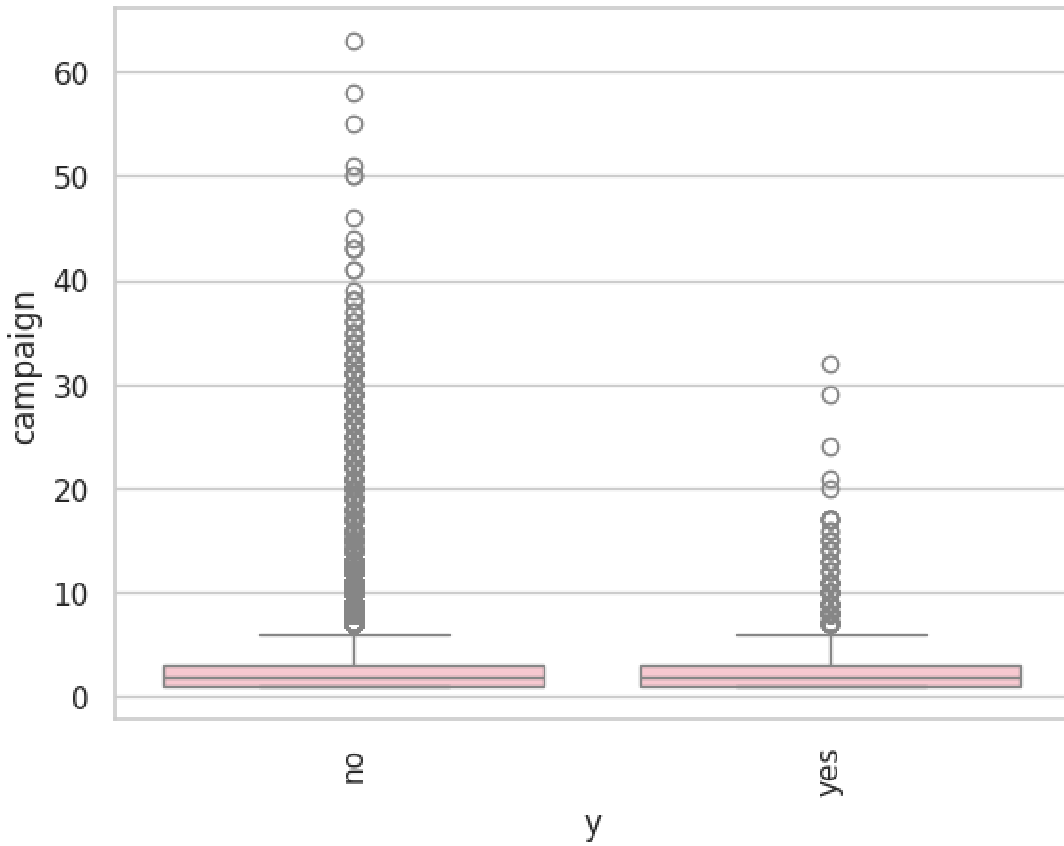
```
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
'month', 'poutcome', 'y']
```

[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")
```



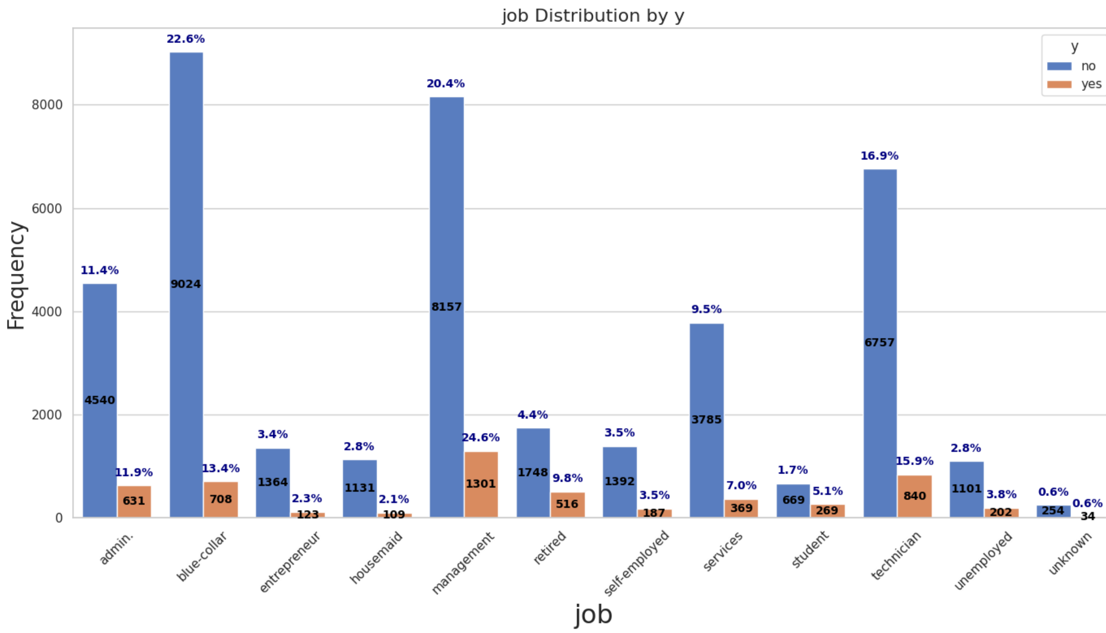
[30]:
```
cat_cont_plot(df,"y", "campaign")
```



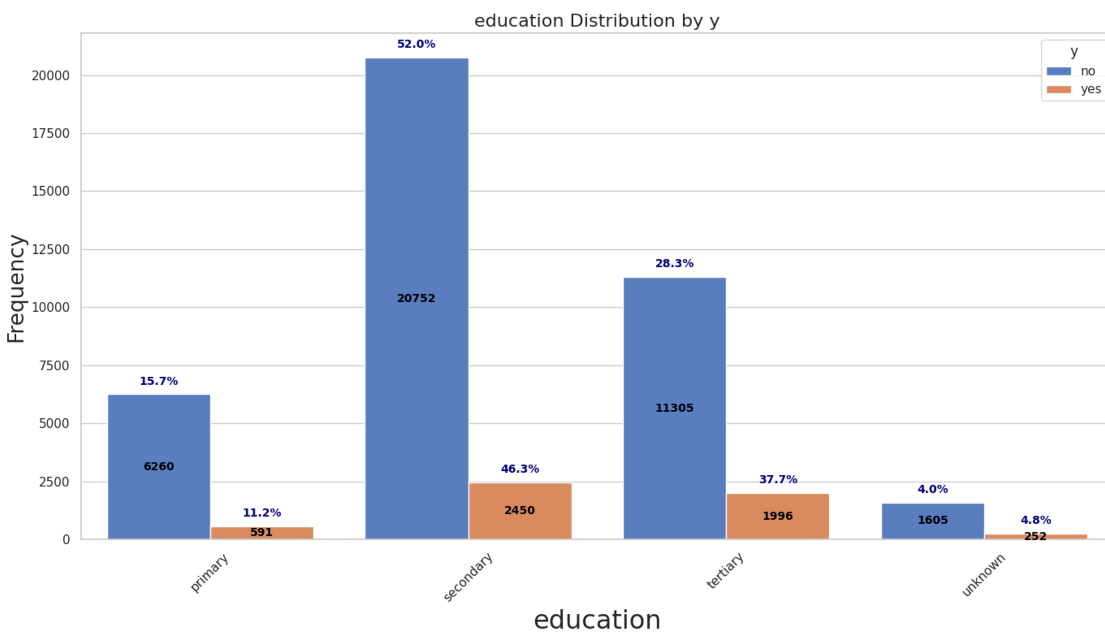
[31]:
```
def cat_cat_plot(dataframe, column_name, hue_column):
    """
    Plots a grouped bar chart showing category frequencies split by hue,
    with both frequency (inside bar) and proportion (above bar) labels.

    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" ,␣
↪"contact",'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  contact  previous_contact  \
0  unknown       261  unknown         0         1  unknown                 0
1  unknown       151  unknown         0         1  unknown                 0
2  unknown        76  unknown         0         1  unknown                 0
3  unknown        92  unknown         0         1  unknown                 0
4  unknown       198  unknown         0         1  unknown                 0

   pdays   y
0    NaN  no
1    NaN  no
2    NaN  no
3    NaN  no
4    NaN  no
```

[ ]:

[ ]:

[ ]:

1.1 Preprocessing

[38]:
```
dfrq1.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   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  contact           45211 non-null  object
 14  previous_contact  45211 non-null  int64
 15  pdays             8257 non-null   float64
 16  y                 45211 non-null  object
dtypes: float64(1), int64(6), object(10)
memory usage: 5.9+ MB
```

[ ]:
```

```

[ ]:
```

```

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

[39]:
```
(45211, 17)
```

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, 38)
   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  …  contact_telephone  \
0                0                 0              0  …                  0
1                0                 0              0  …                  0
2                0                 1              0  …                  0
3                1                 0              0  …                  0
4                0                 0              0  …                  0

   contact_unknown  poutcome_other  poutcome_success  poutcome_unknown  \
0                1               0                 0                 1
1                1               0                 0                 1
2                1               0                 0                 1
3                1               0                 0                 1
4                1               0                 0                 1

   contact_telephone  contact_unknown  contact_telephone  contact_unknown  \
0                  0                1                  0                1
1                  0                1                  0                1
2                  0                1                  0                1
3                  0                1                  0                1
4                  0                1                  0                1

   y_yes
0      0
1      0
2      0
3      0
4      0

[5 rows x 38 columns]
```

[41]:

```
from sklearn.model_selection import train_test_split

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

[42]:

```
#Train-test split (stratified)
from sklearn.model_selection import train_test_split

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]:

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.linear_model import LogisticRegression

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
    ))
])
```

[44]:

```
log_reg_pipeline.fit(X_train, y_train)
```

[44]:

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

Model evaluation

[45]:
```
from sklearn.metrics import confusion_matrix, classification_report

y_pred = log_reg_pipeline.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[10002  1975]
 [  341  1246]]
```

[46]:
```
print(classification_report(y_test, y_pred))
```

```
              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
```

[47]:
```
#ROC-AUC
from sklearn.metrics import roc_auc_score

y_prob = log_reg_pipeline.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_prob)

print("ROC-AUC:", roc_auc)
```

```
ROC-AUC: 0.89182053883049
```

Interpret coefficients

[48]:
```
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
```

[48]:

	Feature	Coefficient	Odds_Ratio
31	poutcome_success	2.385626	10.865859
2	duration	1.193241	3.297752
14	job_student	0.568677	1.765929
19	education_tertiary	0.518236	1.679063
11	job_retired	0.424058	1.528151
20	education_unknown	0.307635	1.360205
30	poutcome_other	0.260160	1.297138
18	education_secondary	0.258276	1.294695
22	marital_single	0.189902	1.209131
5	previous_contact	0.177928	1.194740
1	balance	0.040230	1.041050
3	previous	0.030356	1.030821
35	contact_telephone	0.012216	1.012291
33	contact_telephone	0.012216	1.012291
28	contact_telephone	0.012216	1.012291
26	contact_telephone	0.012216	1.012291
0	age	0.009416	1.009460
6	pdays	0.000241	1.000241
32	poutcome_unknown	-0.101671	0.903327
21	marital_married	-0.178142	0.836823
10	job_management	-0.203363	0.815982
4	campaign	-0.236401	0.789464
23	default_yes	-0.252914	0.776535
16	job_unemployed	-0.272010	0.761847
15	job_technician	-0.322706	0.724186
36	contact_unknown	-0.332182	0.717357
29	contact_unknown	-0.332182	0.717357
34	contact_unknown	-0.332182	0.717357
27	contact_unknown	-0.332182	0.717357
17	job_unknown	-0.345373	0.707957
13	job_services	-0.448088	0.638848
8	job_entrepreneur	-0.529650	0.588811
9	job_housemaid	-0.533286	0.586674
7	job_blue-collar	-0.536965	0.584520
12	job_self-employed	-0.538566	0.583585
25	loan_yes	-0.687911	0.502625
24	housing_yes	-0.849760	0.427518

[ ]:

```

```

Interpretation rule:

Odds Ratio > 1 → increases probability of subscription

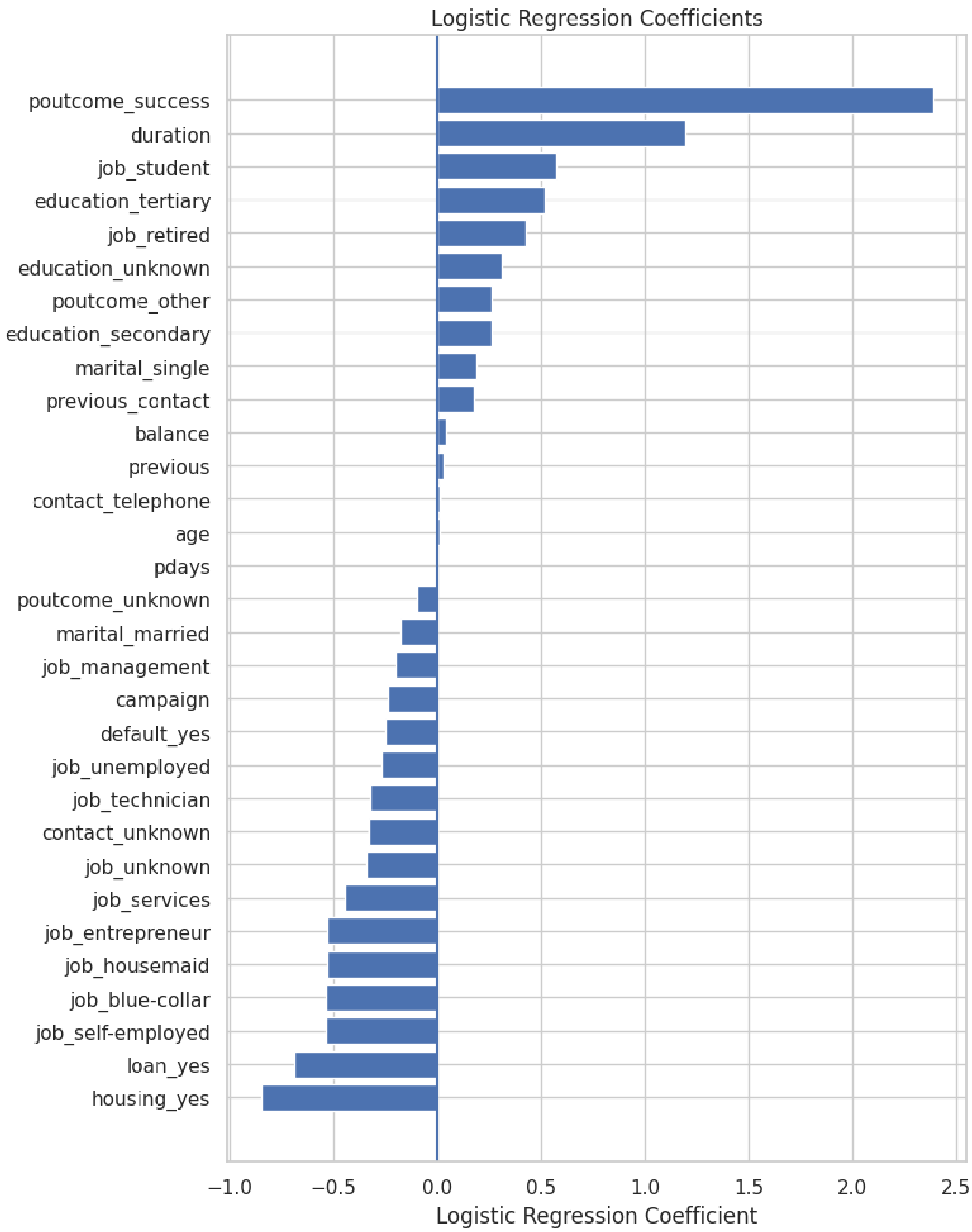
Odds Ratio < 1 → decreases probability

[49]:

```
#Extract coefficients and odds ratios from the pipeline
import pandas as pd
import numpy as np

# Extract coefficients
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)
```

[50]:

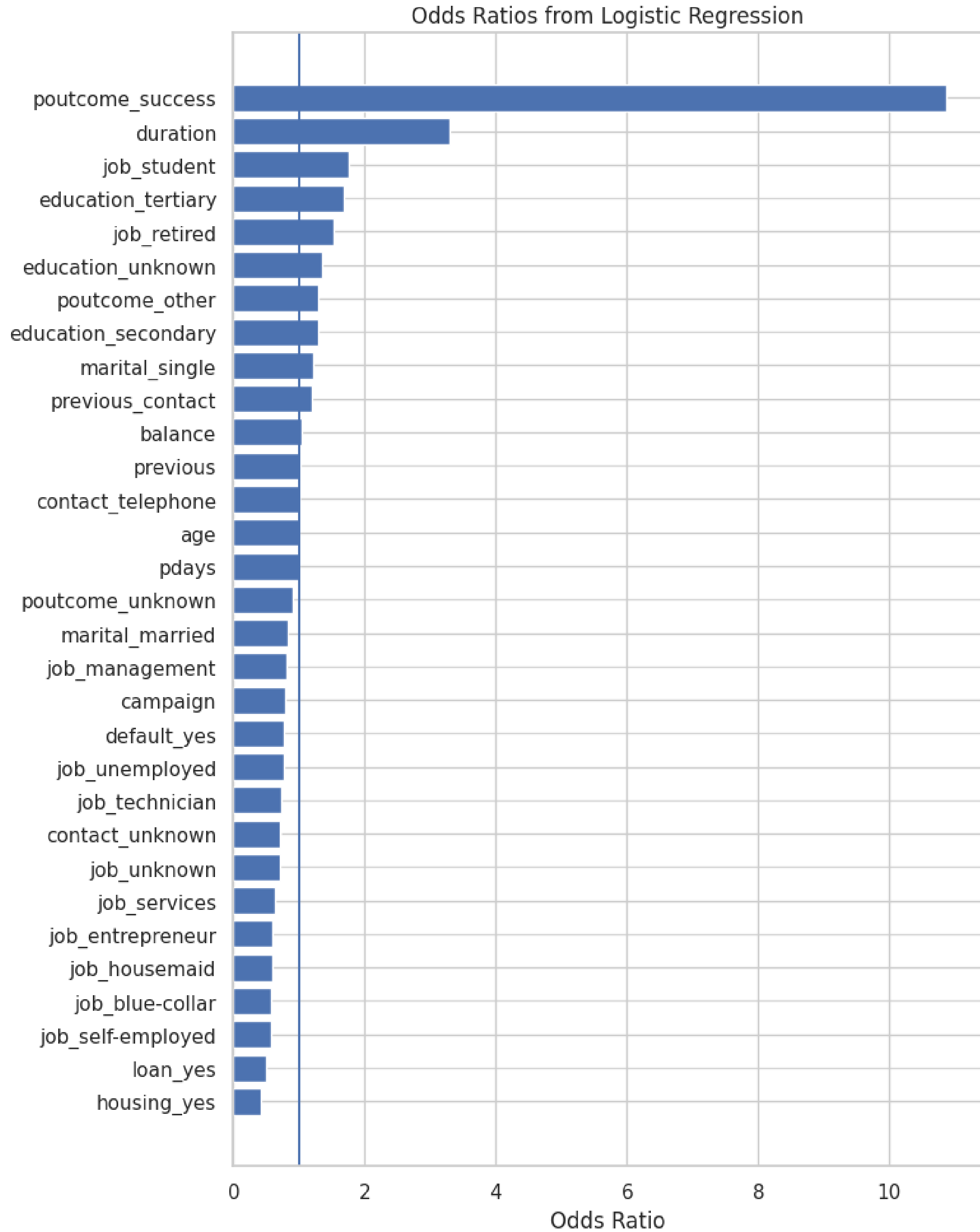
```
#Visualise coefficients (direction + strength)
import matplotlib.pyplot as plt

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()
```



[51]:

```
#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 Random forest

[52]:
```
X = dfrq1_enc.drop(columns=['y_yes'])
y = dfrq1_enc['y_yes']
```

[53]:
```
from sklearn.model_selection import train_test_split

X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(
    X, y,
    test_size=0.3,
    stratify=y,
    random_state=42
)
```

Fit Random Forest (no scaling needed)

Random Forest does NOT need scaling and handles outliers naturally.

[54]:
```
from sklearn.ensemble import RandomForestClassifier

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_rf, y_train_rf)
```

[54]:
```
RandomForestClassifier(class_weight='balanced', min_samples_leaf=5,
                       n_estimators=300, n_jobs=-1, random_state=42)
```

Model evaluation

[55]:
```
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪roc_auc_score

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

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))
```

```
Confusion Matrix:
 [[10494  1483]
 [  384  1203]]
```

```
Classification Report:
              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

ROC-AUC: 0.9014899329996021
```

Feature importance

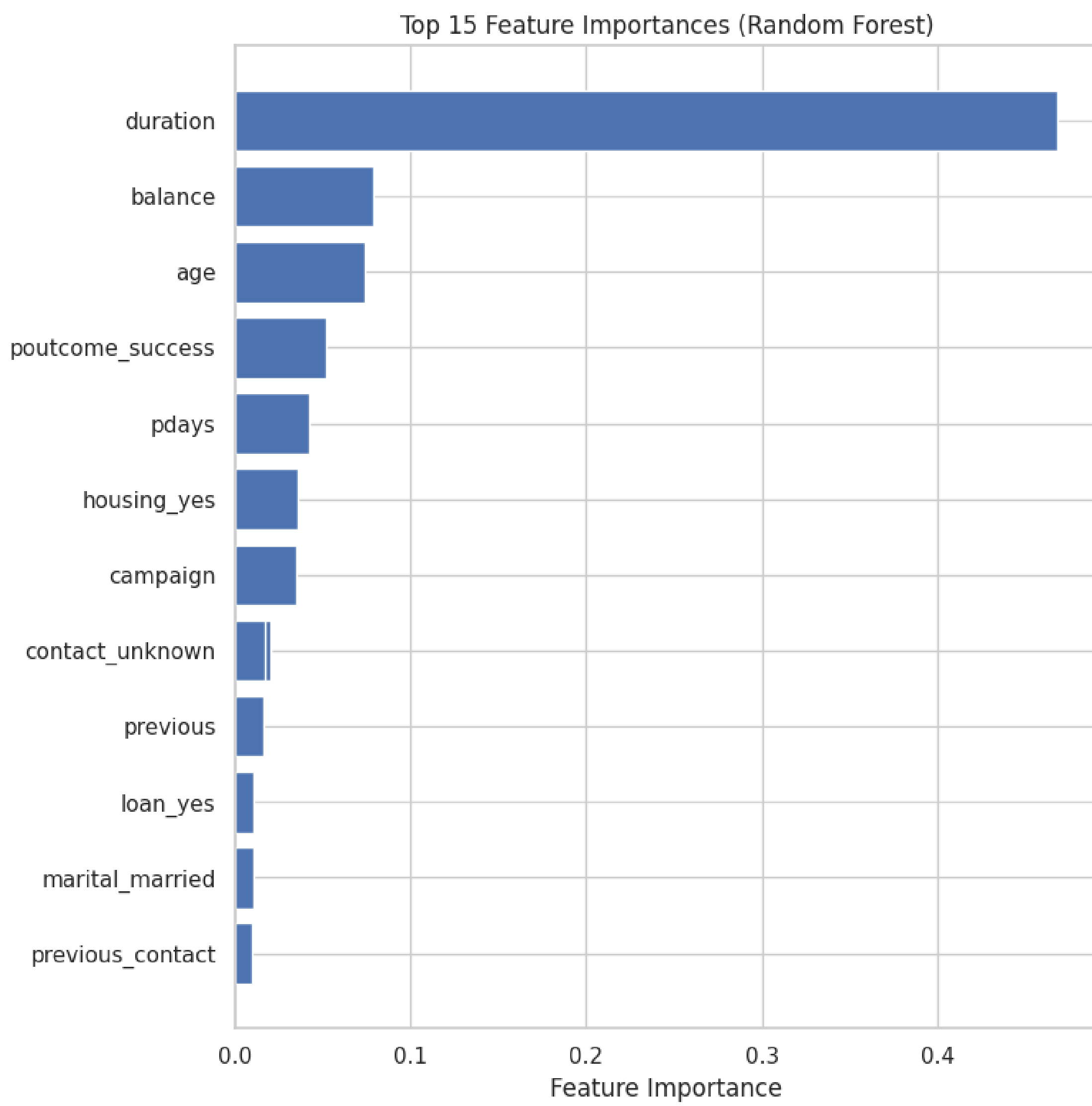
[56]:
```
import pandas as pd

feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

feature_importance.head(15)
```

[56]:

	Feature	Importance
2	duration	0.468353
1	balance	0.078820
0	age	0.073920
31	poutcome_success	0.051599
6	pdays	0.042453
24	housing_yes	0.035750
4	campaign	0.035113
27	contact_unknown	0.020728
34	contact_unknown	0.020588
29	contact_unknown	0.020351
36	contact_unknown	0.017481
3	previous	0.015889
25	loan_yes	0.010693
21	marital_married	0.010467
5	previous_contact	0.009960

[57]:
```
#Visualise top feature importances
import matplotlib.pyplot as plt

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.

2.1 SHAP

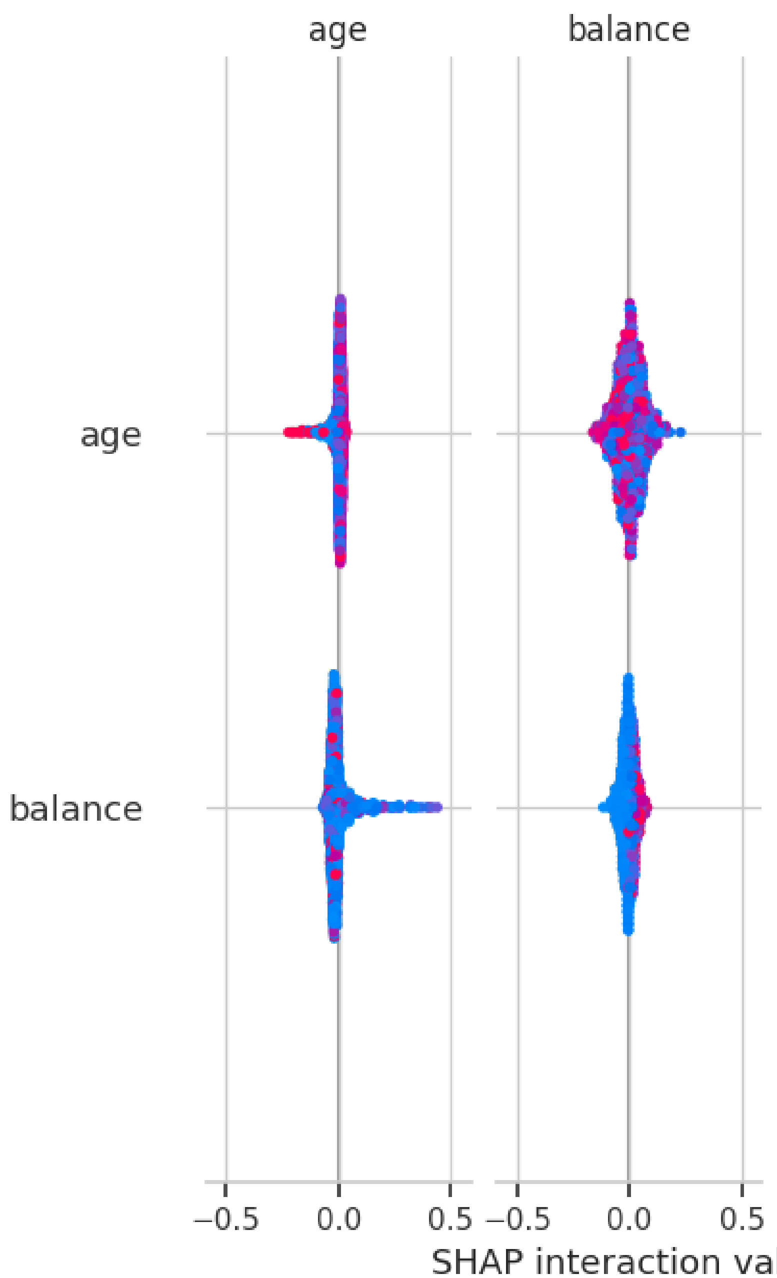
To enhance interpretability of the Random Forest model, SHAP (SHapley Additive exPlanations) values were computed. SHAP provides a model-agnostic explanation framework that quantifies both the magnitude and direction of each feature's contribution to the predicted outcome. The SHAP summary plot highlights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

[58]:
```python
# Take a random sample of the test set
X_test_shap = X_test.sample(n=1000, random_state=42)
```

[59]:
```python
import shap

explainer = shap.TreeExplainer(rf)

shap_values = explainer.shap_values(X_test_shap)
```

[64]:
```python
import shap

# 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%|===================| 1998/2000 [02:38<00:00]
```



[ ]:

```python

```

[69]:

```python

```

[68]:

```python
# ---------------------------------------------------
# 5. 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]

# --------------------------------------------------
# 6. 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)

# --------------------------------------------------
# 7. 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!
```

[68]:

	feature	main_effect	interaction_effect	interaction_ratio
0	age	0.016193	0.0	0.0
19	education_tertiary	0.011398	0.0	0.0
21	marital_married	0.010235	0.0	0.0
22	marital_single	0.006799	0.0	0.0
23	default_yes	0.000343	0.0	0.0
24	housing_yes	0.034003	0.0	0.0
25	loan_yes	0.010002	0.0	0.0
26	contact_telephone	0.000414	0.0	0.0

27	contact_unknown	0.014227	0.0	0.0
28	contact_telephone	0.000433	0.0	0.0
29	contact_unknown	0.013769	0.0	0.0
30	poutcome_other	0.001024	0.0	0.0
31	poutcome_success	0.017082	0.0	0.0
32	poutcome_unknown	0.007596	0.0	0.0
33	contact_telephone	0.000420	0.0	0.0
34	contact_unknown	0.013588	0.0	0.0
35	contact_telephone	0.000501	0.0	0.0
20	education_unknown	0.000665	0.0	0.0
18	education_secondary	0.002449	0.0	0.0
1	balance	0.020787	0.0	0.0
17	job_unknown	0.000015	0.0	0.0
2	duration	0.134506	0.0	0.0
3	previous	0.008536	0.0	0.0
4	campaign	0.017766	0.0	0.0
5	previous_contact	0.008049	0.0	0.0
6	pdays	0.013353	0.0	0.0
7	job_blue-collar	0.005185	0.0	0.0
8	job_entrepreneur	0.000691	0.0	0.0
9	job_housemaid	0.000449	0.0	0.0
10	job_management	0.003651	0.0	0.0
11	job_retired	0.001034	0.0	0.0
12	job_self-employed	0.000531	0.0	0.0
13	job_services	0.001159	0.0	0.0
14	job_student	0.001493	0.0	0.0
15	job_technician	0.002351	0.0	0.0
16	job_unemployed	0.000379	0.0	0.0
36	contact_unknown	0.012287	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.

```
[ ]:
```