In [125…

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(color_codes=True)
```

In [126…

```python
df=pd.read_csv('Assignment-2_Data.csv')
df.head()
```

Out[126]:

	Id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	1001	999.0	management	married	tertiary	no	2143.0	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	1002	44.0	technician	single	secondary	no	29.0	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	1003	33.0	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	1004	47.0	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	1005	33.0	unknown	single	unknown	no	1.0	no	no	unknown	5	may	198	1	-1	0	unknown	no

In [127…

```python
df.tail()
```

Out[127]:

	Id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
45206	46207	51.0	technician	married	tertiary	no	825.0	no	no	cellular	17	nov	977	3	-1	0	unknown	yes
45207	46208	71.0	retired	divorced	primary	no	1729.0	no	no	cellular	17	nov	456	2	-1	0	unknown	yes
45208	46209	72.0	retired	married	secondary	no	5715.0	no	no	cellular	17	nov	1127	5	184	3	success	yes
45209	46210	57.0	blue-collar	married	secondary	no	668.0	no	no	telephone	17	nov	508	4	-1	0	unknown	no
45210	46211	37.0	entrepreneur	married	secondary	no	2971.0	no	no	cellular	17	nov	361	2	188	11	other	no

In [128…

```python
df.shape
```

Out[128]:

```
(45211, 18)
```

In [129…

```python
df.columns
```

Out[129]:

```
Index(['Id', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome', 'y'],
      dtype='object')
```

In [130…

```python
df.describe()
```

Out[130]:

	Id	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45202.000000	45208.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	23606.000000	40.954714	1362.346620	15.806419	258.163080	2.763841	40.197828	0.580323
std	13051.435847	11.539144	3044.852387	8.322476	257.527812	3.098021	100.128746	2.303441
min	1001.000000	-1.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	12303.500000	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	23606.000000	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	34908.500000	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	46211.000000	999.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

In [131…

```
df.describe(include='O')
```

Out[131]:

	job	marital	education	default	housing	loan	contact	month	poutcome	y
count	45211	45211	45211	45211	45211	45211	45211	45211	45211	45211
unique	12	3	4	2	2	2	3	12	4	2
top	blue-collar	married	secondary	no	yes	no	cellular	may	unknown	no
freq	9732	27214	23202	44396	25130	37967	29285	13766	36959	39922

In [ ]:

```

```

Data Preprocessing Part 1:

In [132…

```
# drop identifier data:
df.drop(columns = 'Id', inplace=True)
df.head()
```

Out[132]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	999.0	management	married	tertiary	no	2143.0	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44.0	technician	single	secondary	no	29.0	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33.0	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47.0	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33.0	unknown	single	unknown	no	1.0	no	no	unknown	5	may	198	1	-1	0	unknown	no

In [133…

```
df.select_dtypes(include='object').nunique()
```

Out[133]:

```
job          12
marital       3
education     4
default       2
housing       2
loan          2
contact       3
month        12
poutcome      4
y             2
dtype: int64
```

In [134…

```
df.select_dtypes(include=['int', 'float']).nunique()
```

Out[134]:

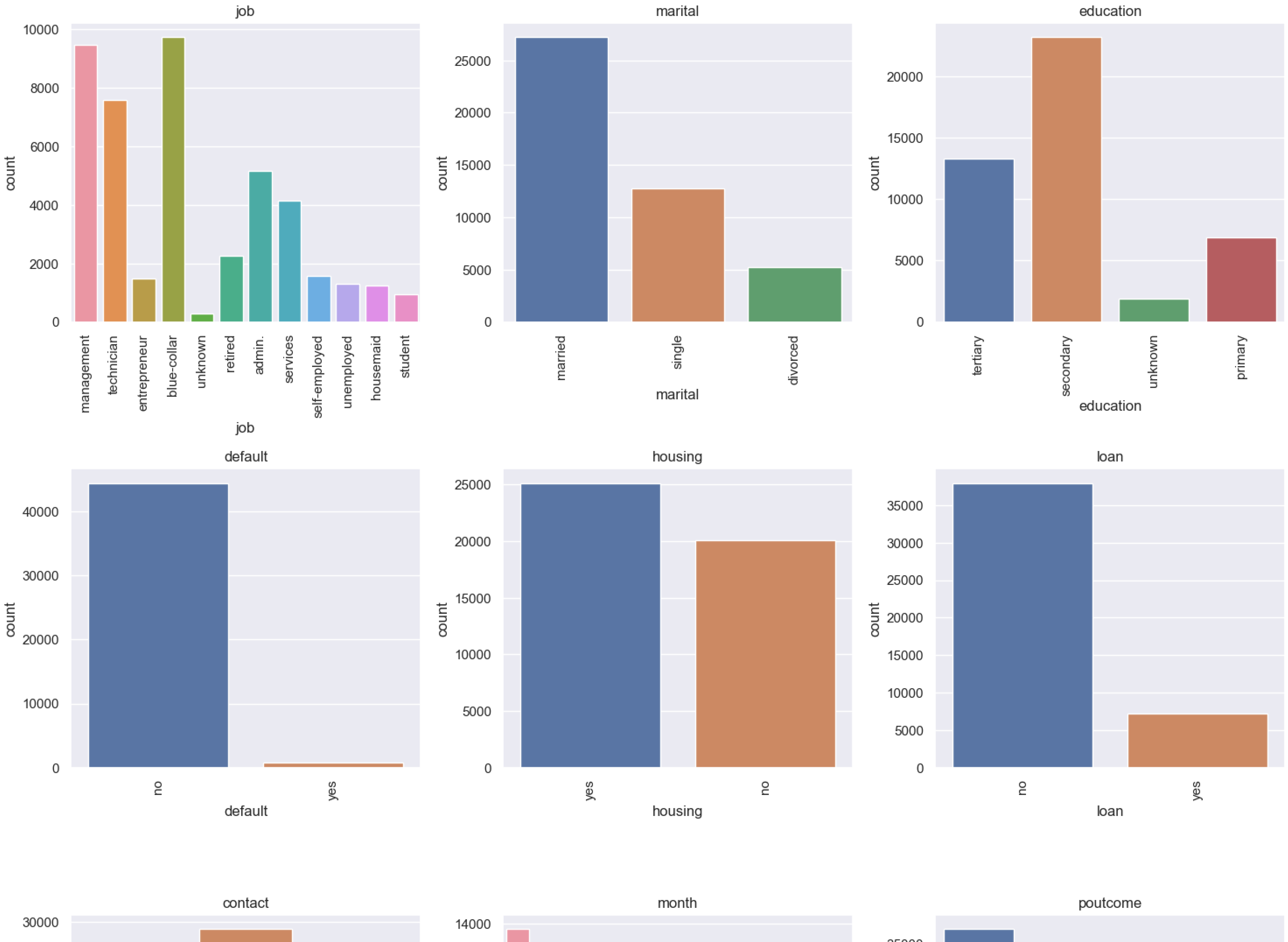
```
age           79
balance     7168
day           31
duration    1573
campaign      48
pdays        559
previous      41
dtype: int64
```

Exploratory Data Analysis (EDA):

In [135…

```
# get the names of all categorical('object' dtybe):
cols_names =df.select_dtypes('object').columns.tolist()
```

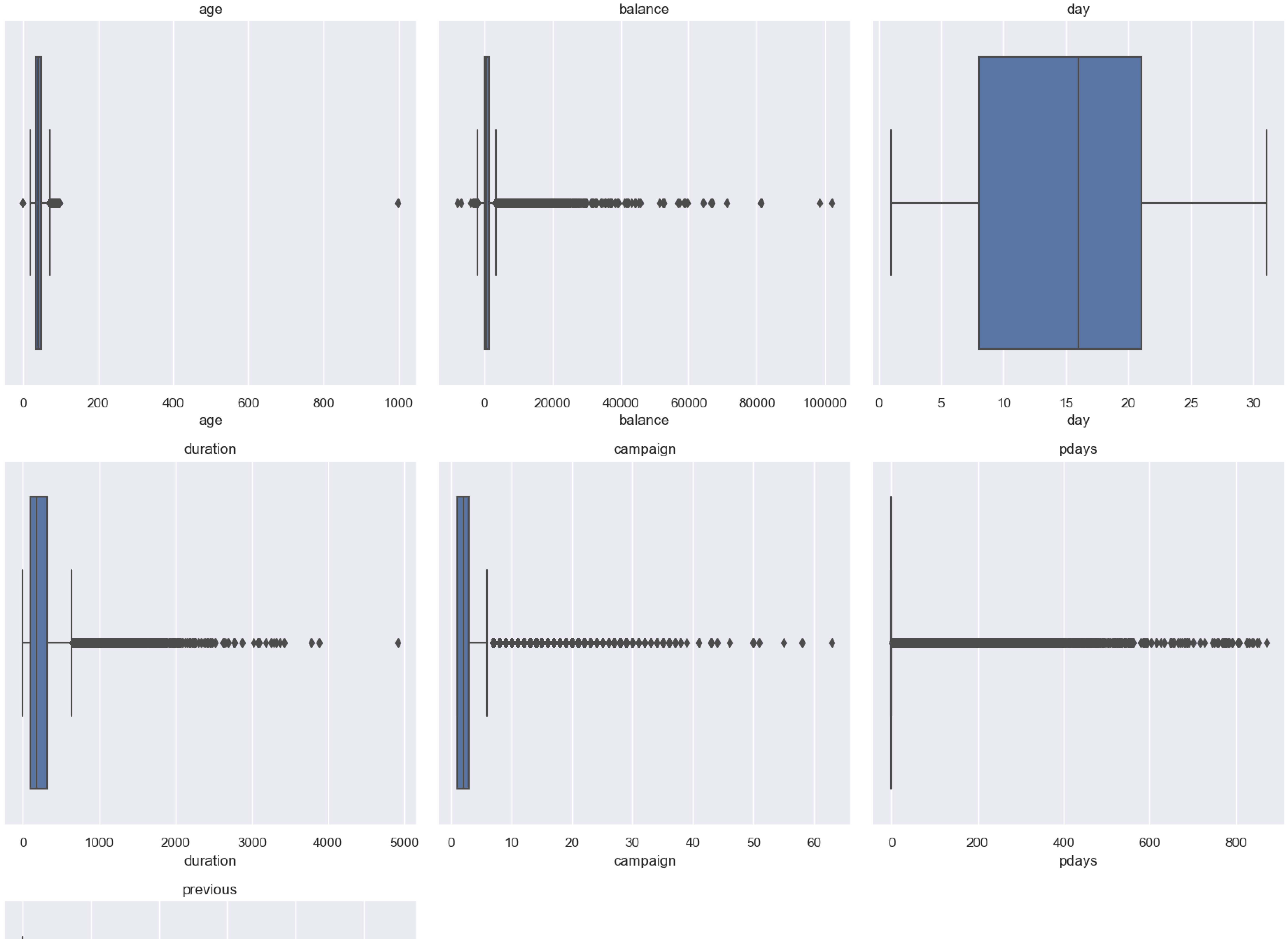
```python
# create a subplot figure:
num_cols = len(cols_names)
num_rows = num_cols+2
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a countplot for the top 5 values of each categorical variable using Seaborn
for i , names in enumerate (cols_names):
    top_values=df[names].value_counts().nlargest(12).index
    filtered_data=df[df[names].isin(top_values)]
    sns.countplot(x=names,data=filtered_data,ax=axs[i])
    axs[i].set_title(names)
    axs[i].tick_params(axis='x', rotation=90)

# remove all extra empty subplot:
if len(axs)> num_cols:
    for i in range(num_cols,len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

#show plot:
plt.show()
```


job
count
10000
8000
6000
4000
2000
0
management
technician
entrepreneur
blue-collar
unknown
retired
admin.
services
self-employed
unemployed
housemaid
student
job
marital
count
25000
20000
15000
10000
5000
0
married
single
divorced
marital
education
count
20000
15000
10000
5000
0
tertiary
secondary
unknown
primary
education
default
count
40000
30000
20000
10000
0
no
yes
default
housing
count
25000
20000
15000
10000
5000
0
yes
no
housing
loan
count
35000
30000
25000
20000
15000
10000
5000
0
no
yes
loan
contact
30000
month
14000
poutcome

In [136…

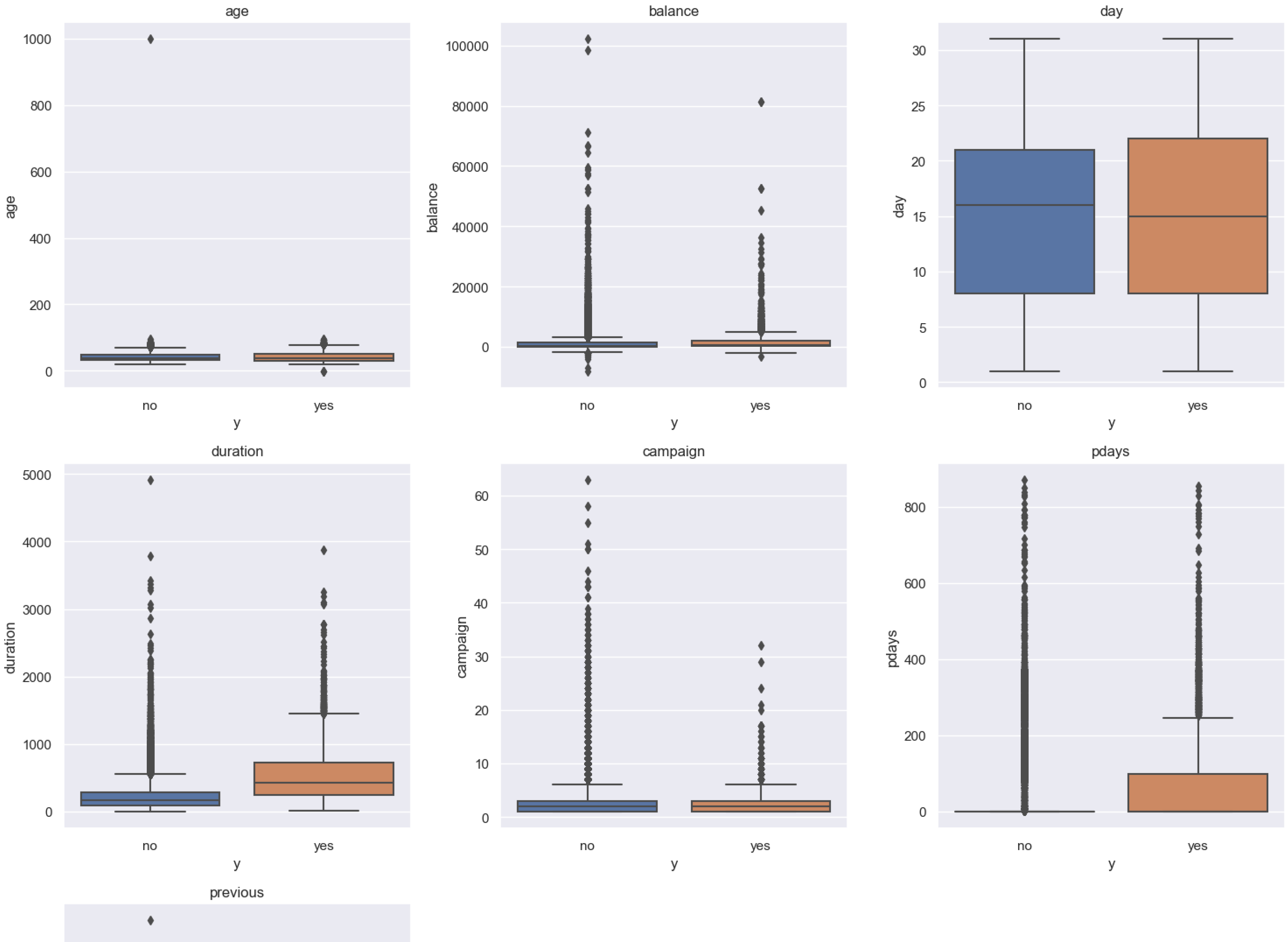
```
# Get the names of all columns with data type 'int' or 'float'
cols_names = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(cols_names)
num_rows = (num_cols + 2)
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each numerical variable using Seaborn
```

```python
for i,names in enumerate(cols_names):
    sns.boxplot(x=df[names], ax=axs[i])
    axs[i].set_title(names)

# Remove any extra empty subplots if needed
if  num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


age
0
200
400
600
800
1000
age
balance
0
20000
40000
60000
80000
100000
balance
day
0
5
10
15
20
25
30
day
duration
0
1000
2000
3000
4000
5000
duration
campaign
0
10
20
30
40
50
60
campaign
pdays
0
200
400
600
800
pdays
previous

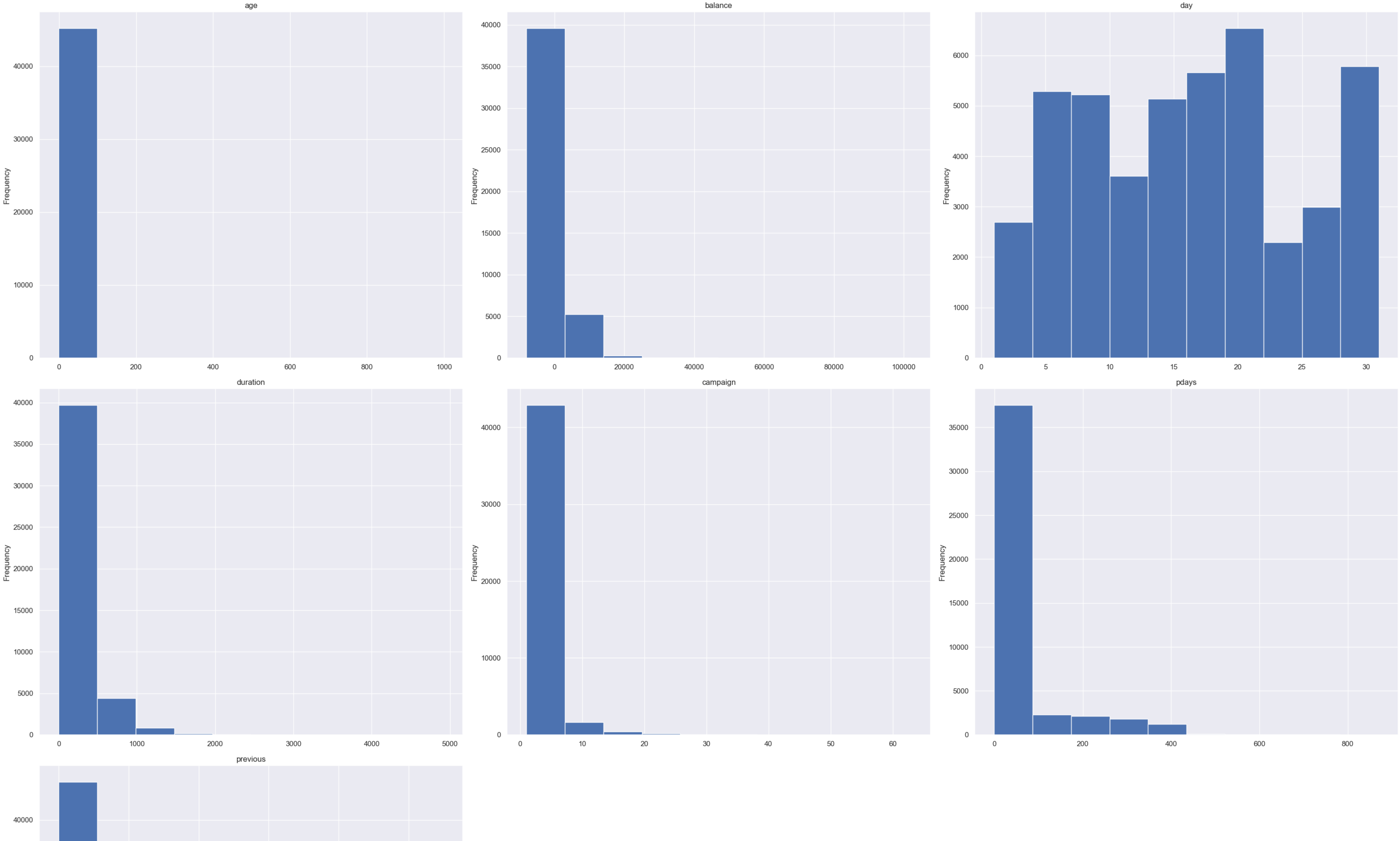
In [137…

```python
# create a box plot will all columns with data type(int):

# get the names of (int,float)data type columns:
cols_names = df.select_dtypes(include=['int','float']).columns.tolist()

# create a figure with subplots:
num_cols = len(cols_names)
num_raws = num_cols + 2      # to make sure there is enough rows for plot
fig , axs = plt.subplots(ncols= 3, nrows= num_rows,figsize=(15,5*num_rows))
axs= axs.flatten()

#create a boxplot using seaborn to show y values by columns names:
for i ,names in enumerate (cols_names):
    sns.boxplot(x='y',y=names,data = df , ax =axs[i])
    axs[i].set_title(names)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots:
fig.tight_layout()

# Show plot
plt.show()
```


age
1000
800
600
400
200
0
age
no
yes
y
balance
100000
80000
60000
40000
20000
0
balance
no
yes
y
day
30
25
20
15
10
5
0
day
no
yes
y
duration
5000
4000
3000
2000
1000
0
duration
no
yes
y
campaign
60
50
40
30
20
10
0
campaign
no
yes
y
pdays
800
600
400
200
0
pdays
no
yes
y
previous

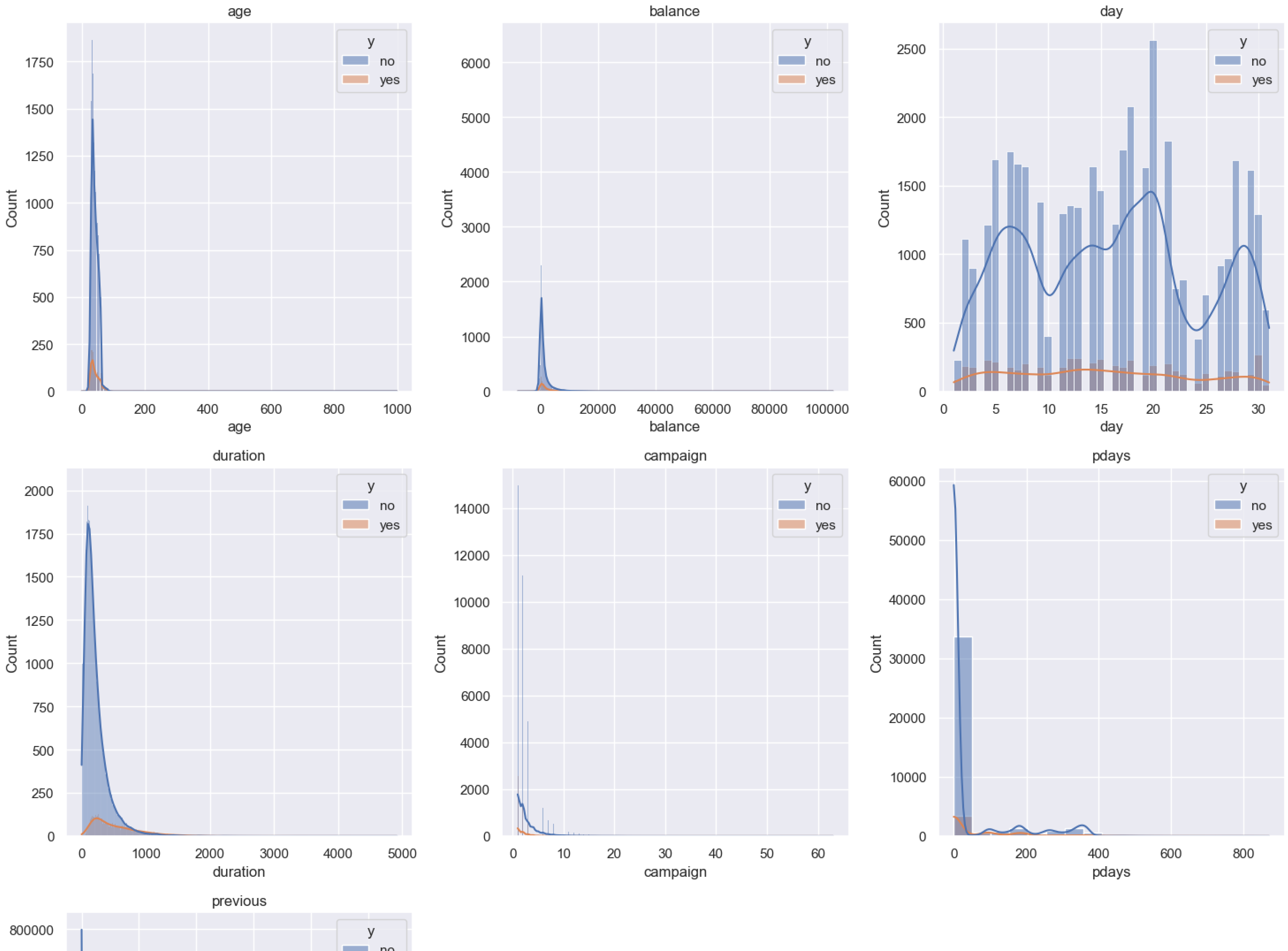
In [138…

```
# create a hist plot will all columns with data type(int):

# get the names of (int,float)data type columns:
cols_names = df.select_dtypes(include=['int','float']).columns.tolist()

# create a figure with subplots:
num_cols = len(cols_names)
num_raws = (num_cols + 2) //3   # to make sure there is enough rows for plot
fig , axs = plt.subplots(ncols= 3, nrows= num_rows,figsize=(30,8*num_rows))
axs= axs.flatten()

#create a histgram using seaborn to show y values by columns names:
for i ,names in enumerate (cols_names):
    df[names].plot.hist( ax =axs[i])
    axs[i].set_title(names)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots:
fig.tight_layout()

# Show plot
plt.show()
```


age
Frequency
0
10000
20000
30000
40000
0
200
400
600
800
1000
balance
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
0
20000
40000
60000
80000
100000
day
Frequency
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25
30
duration
Frequency
0
5000
10000
15000
20000
25000
30000
35000
40000
0
1000
2000
3000
4000
5000
campaign
Frequency
0
10000
20000
30000
40000
0
10
20
30
40
50
60
pdays
Frequency
0
5000
10000
15000
20000
25000
30000
35000
0
200
400
600
800
previous
40000

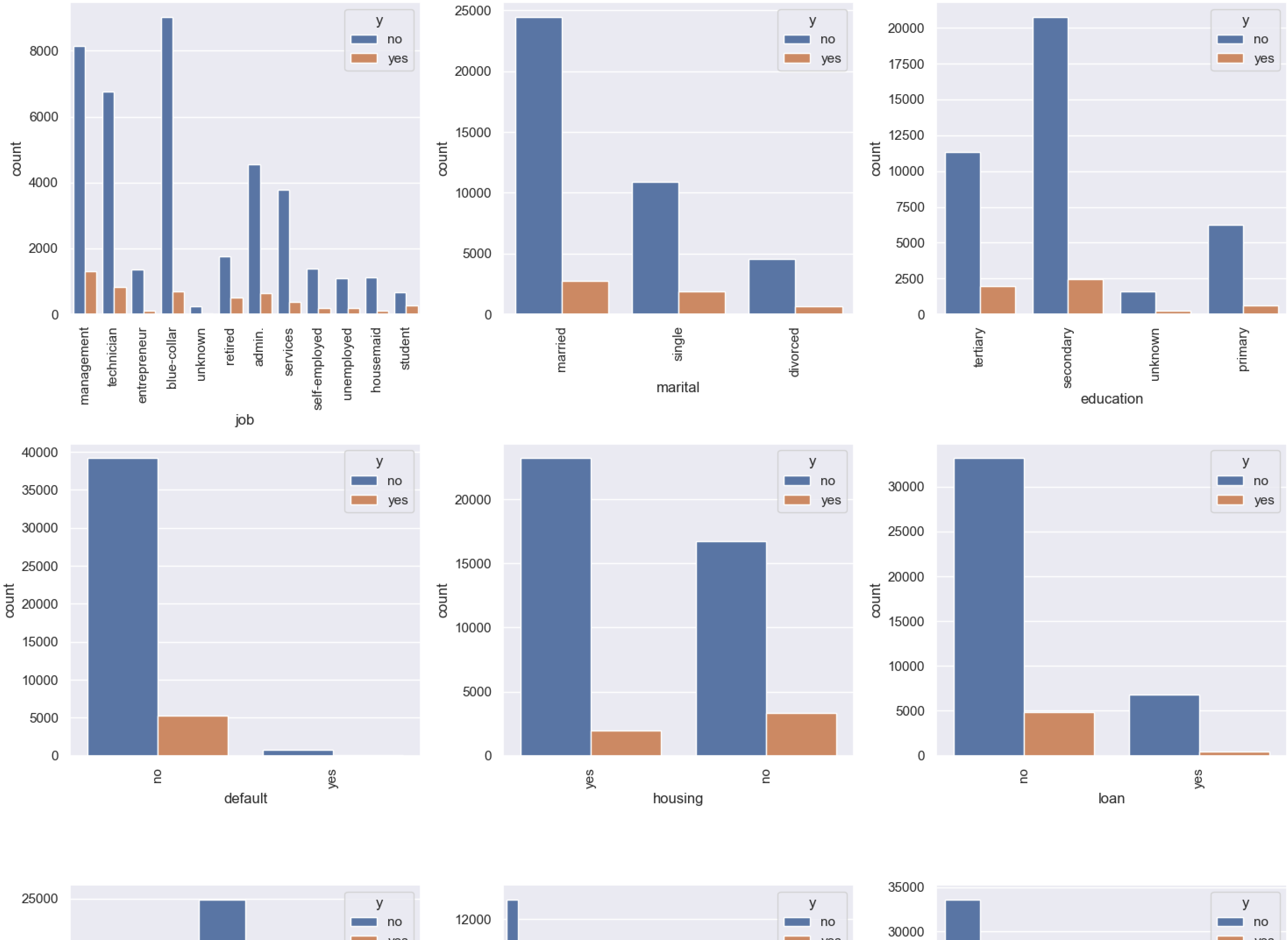
In [139…

```
# create a hist plot will all columns with data type(int):

# get the names of (int,float)data type columns:
cols_names = df.select_dtypes(include=['int','float']).columns.tolist()

# create a figure with subplots:
num_cols = len(cols_names)
num_raws = (num_cols + 2 )//3  # to make sure there is enough rows for plot
fig , axs = plt.subplots(ncols=3, nrows= num_rows,figsize=(15 ,5*num_rows))
axs= axs.flatten()

#create a histplot using seaborn to show y values by columns names:

for i ,names in enumerate (cols_names):
    sns.histplot(x=names ,data = df, hue='y',kde = True, ax=axs[i])
    axs[i].set_title(names)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots:
fig.tight_layout()

# Show plot
plt.show()
```


age
Count
0
250
500
750
1000
1250
1500
1750
0
200
400
600
800
1000
age
y
no
yes
balance
Count
0
1000
2000
3000
4000
5000
6000
0
20000
40000
60000
80000
100000
balance
y
no
yes
day
Count
0
500
1000
1500
2000
2500
0
5
10
15
20
25
30
day
y
no
yes
duration
Count
0
250
500
750
1000
1250
1500
1750
2000
0
1000
2000
3000
4000
5000
duration
y
no
yes
campaign
Count
0
2000
4000
6000
8000
10000
12000
14000
0
10
20
30
40
50
60
campaign
y
no
yes
pdays
Count
0
10000
20000
30000
40000
50000
60000
0
200
400
600
800
pdays
y
no
yes
previous
800000
y
no

In [140…

```
# list of categorical variables to plot
cat_vars = ['job', 'marital', 'education', 'default',
            'housing', 'loan', 'contact', 'month', 'poutcome']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='y', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```


y
no
yes
count
0
2000
4000
6000
8000
management
technician
entrepreneur
blue-collar
unknown
retired
admin.
services
self-employed
unemployed
housemaid
student
job
0
5000
10000
15000
20000
25000
married
single
divorced
marital
0
2500
5000
7500
10000
12500
15000
17500
20000
tertiary
secondary
unknown
primary
education
0
5000
10000
15000
20000
25000
30000
35000
40000
no
yes
default
0
5000
10000
15000
20000
yes
no
housing
0
5000
10000
15000
20000
25000
30000
no
yes
loan
25000
12000
30000
35000

In [ ]:

```

```

data preprocessing part 2:

In [141…

```
#check messing value:
missing= df.isnull().sum() *100 /df.shape[0]
missing[missing >0]
```

Out[141]:

```
age        0.019907
balance    0.006636
dtype: float64
```

In [142…

```
# we will delete missing values because it is very small amount:
df.dropna(inplace =True)
```

In [143…

```
#check messing value after droping:
missing= df.isnull().sum() *100 /df.shape[0]
missing[missing >0]
```

Out[143]:

```
Series([], dtype: float64)
```

In [144…

```
df.shape
```

Out[144]:

```
(45200, 17)
```

Label Encoding for Object Datatypes:

In [145…

```
# loab every categoricql column in dataframe:
for cols in df.select_dtypes(include='O').columns:
    # print the column name & unique values:
    print(f'{cols}: {df[cols].unique()}')
```

```
job: ['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown' 'admin.'
 'services' 'retired' 'self-employed' 'unemployed' 'housemaid' 'student']
```

```
marital: ['married' 'single' 'divorced']
education: ['tertiary' 'secondary' 'unknown' 'primary']
default: ['no' 'yes']
housing: ['yes' 'no']
loan: ['no' 'yes']
contact: ['unknown' 'cellular' 'telephone']
month: ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']
poutcome: ['unknown' 'failure' 'other' 'success']
y: ['no' 'yes']
```

In [146…

```python
# use label incoder to turn categoric dats into numeric data :
from sklearn import preprocessing
# Loop over each column in the DataFrame where dtype is 'object'
for cols in df.select_dtypes(include='object').columns:
    encoder= preprocessing.LabelEncoder()
    encoder.fit(df[cols].unique())
    df[cols]=encoder.transform(df[cols])
    # Print the column name and the unique encoded values
    print(f"{cols}: {df[cols].unique()}")
```

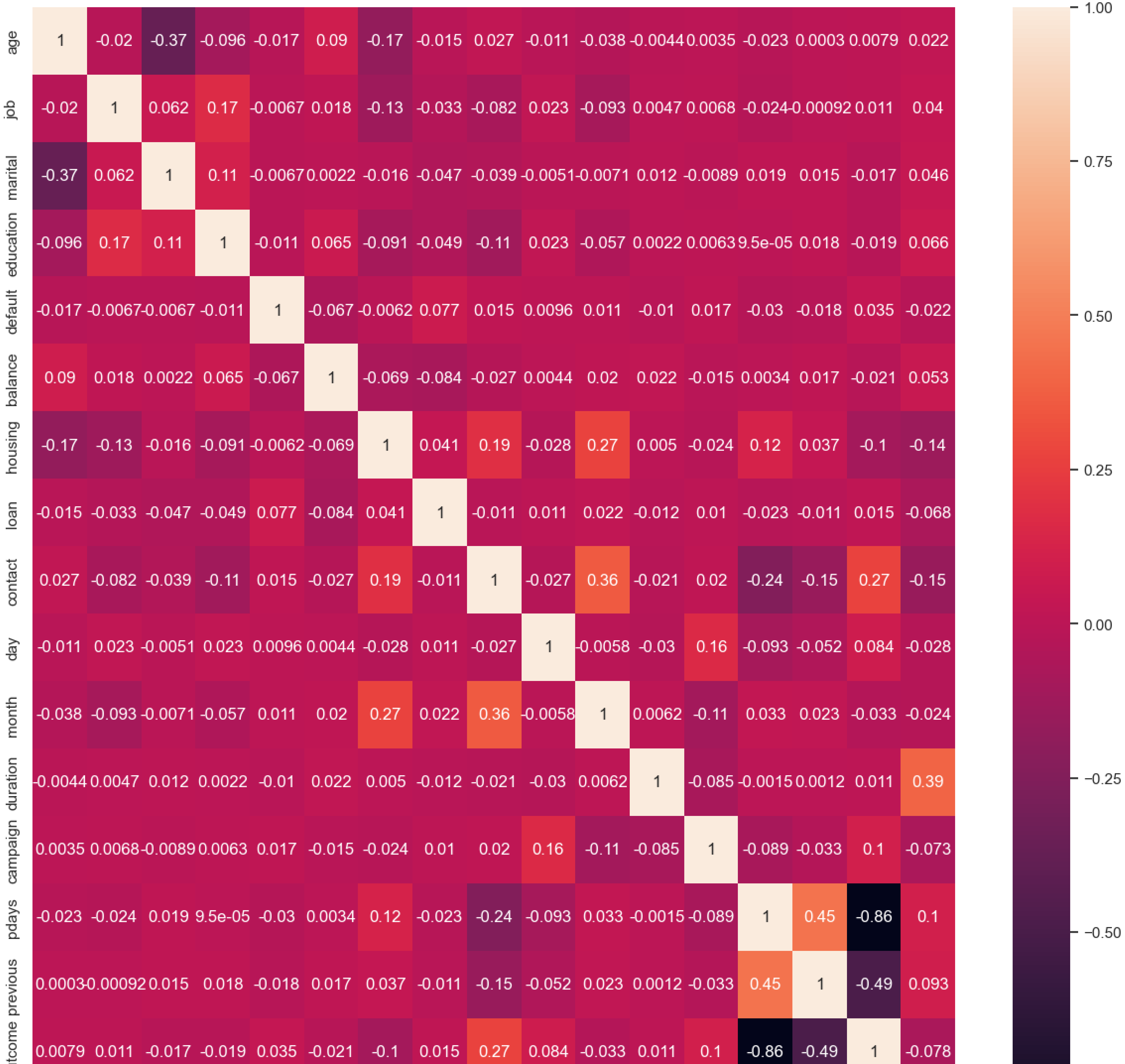
```
job: [ 4  9  2  1 11  0  7  5  6 10  3  8]
marital: [1 2 0]
education: [2 1 3 0]
default: [0 1]
housing: [1 0]
loan: [0 1]
contact: [2 0 1]
month: [ 8  6  5  1 10  9  2  4  3  7  0 11]
poutcome: [3 0 1 2]
y: [0 1]
```

In [147…

```python
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(),annot=True)
```

Out[147]:

```
<AxesSubplot:>
```



age	1	-0.02	-0.37	-0.096	-0.017	0.09	-0.17	-0.015	0.027	-0.011	-0.038	-0.0044	0.0035	-0.023	0.0003	0.0079	0.022
job	-0.02	1	0.062	0.17	-0.0067	0.018	-0.13	-0.033	-0.082	0.023	-0.093	0.0047	0.0068	-0.024	-0.00092	0.011	0.04
marital	-0.37	0.062	1	0.11	-0.0067	0.0022	-0.016	-0.047	-0.039	-0.0051	-0.0071	0.012	-0.0089	0.019	0.015	-0.017	0.046
education	-0.096	0.17	0.11	1	-0.011	0.065	-0.091	-0.049	-0.11	0.023	-0.057	0.0022	0.0063	9.5e-05	0.018	-0.019	0.066
default	-0.017	-0.0067	-0.0067	-0.011	1	-0.067	-0.0062	0.077	0.015	0.0096	0.011	-0.01	0.017	-0.03	-0.018	0.035	-0.022
balance	0.09	0.018	0.0022	0.065	-0.067	1	-0.069	-0.084	-0.027	0.0044	0.02	0.022	-0.015	0.0034	0.017	-0.021	0.053
housing	-0.17	-0.13	-0.016	-0.091	-0.0062	-0.069	1	0.041	0.19	-0.028	0.27	0.005	-0.024	0.12	0.037	-0.1	-0.14
loan	-0.015	-0.033	-0.047	-0.049	0.077	-0.084	0.041	1	-0.011	0.011	0.022	-0.012	0.01	-0.023	-0.011	0.015	-0.068
contact	0.027	-0.082	-0.039	-0.11	0.015	-0.027	0.19	-0.011	1	-0.027	0.36	-0.021	0.02	-0.24	-0.15	0.27	-0.15
day	-0.011	0.023	-0.0051	0.023	0.0096	0.0044	-0.028	0.011	-0.027	1	-0.0058	-0.03	0.16	-0.093	-0.052	0.084	-0.028
month	-0.038	-0.093	-0.0071	-0.057	0.011	0.02	0.27	0.022	0.36	-0.0058	1	0.0062	-0.11	0.033	0.023	-0.033	-0.024
duration	-0.0044	0.0047	0.012	0.0022	-0.01	0.022	0.005	-0.012	-0.021	-0.03	0.0062	1	-0.085	-0.0015	0.0012	0.011	0.39
campaign	0.0035	0.0068	-0.0089	0.0063	0.017	-0.015	-0.024	0.01	0.02	0.16	-0.11	-0.085	1	-0.089	-0.033	0.1	-0.073
pdays	-0.023	-0.024	0.019	9.5e-05	-0.03	0.0034	0.12	-0.023	-0.24	-0.093	0.033	-0.0015	-0.089	1	0.45	-0.86	0.1
previous	0.0003	-0.00092	0.015	0.018	-0.018	0.017	0.037	-0.011	-0.15	-0.052	0.023	0.0012	-0.033	0.45	1	-0.49	0.093
tcome	0.0079	0.011	-0.017	-0.019	0.035	-0.021	-0.1	0.015	0.27	0.084	-0.033	0.011	0.1	-0.86	-0.49	1	-0.078

train_test_split:

In [148…

```
X = df.drop('y', axis=1)
y = df['y']
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

In [149…

```
print('x_train',x_train)
print('*'*50)
print('y_train',y_train)
```

```
x_train         age          job  marital  education default  balance housing loan  \
9894   37.0      unknown  married    unknown      no   1699.0      no   no   
9217   35.0       admin.  married  secondary      no    214.0     yes  yes   
4124   38.0     services   single   tertiary      no    323.0     yes   no   
30085  30.0   management   single   tertiary      no     57.0     yes   no   
5386   41.0   technician  married   tertiary      no   -762.0     yes  yes   
...     ...          ...      ...        ...     ...      ...     ...  ...   
20757  34.0   technician   single  secondary      no   4367.0      no   no   
41993  22.0      student   single  secondary      no     23.0      no   no   
30403  35.0   management   single   tertiary      no    995.0      no   no   
42613  35.0   management  married   tertiary      no    323.0      no   no   
2732   46.0  blue-collar   single  secondary      no     57.0     yes   no   

        contact  day month  duration  campaign  pdays  previous poutcome  
9894    unknown    9   jun        63         1     -1         0  unknown  
9217    unknown    5   jun       247         1     -1         0  unknown  
4124    unknown   19   may       138         1     -1         0  unknown  
30085  cellular    4   feb       153         2     -1         0  unknown  
5386    unknown   23   may       145         1     -1         0  unknown  
...         ...  ...   ...       ...       ...    ...       ...      ...  
20757  cellular   13   aug       121         2     -1         0  unknown  
41993  cellular   27   oct       137         1     -1         0  unknown  
30403  cellular    5   feb        39         1     -1         0  unknown  
42613  cellular   11   jan       261         2     -1         0  unknown  
2732    unknown   14   may       194         4     -1         0  unknown  

[25316 rows x 16 columns]
**************************************************
y_train 9410     0
17272    0
30832    0
5397     0
34415    0
        ..
30414    1
21254    0
42624    1
43578    1
```

```
2743     0
Name: y, Length: 36160, dtype: int32
```

In [150…

```
print(x_train.shape)
```

```
(25316, 16)
```

In [151…

```
print(y_train.shape)
```

```
(36160,)
```

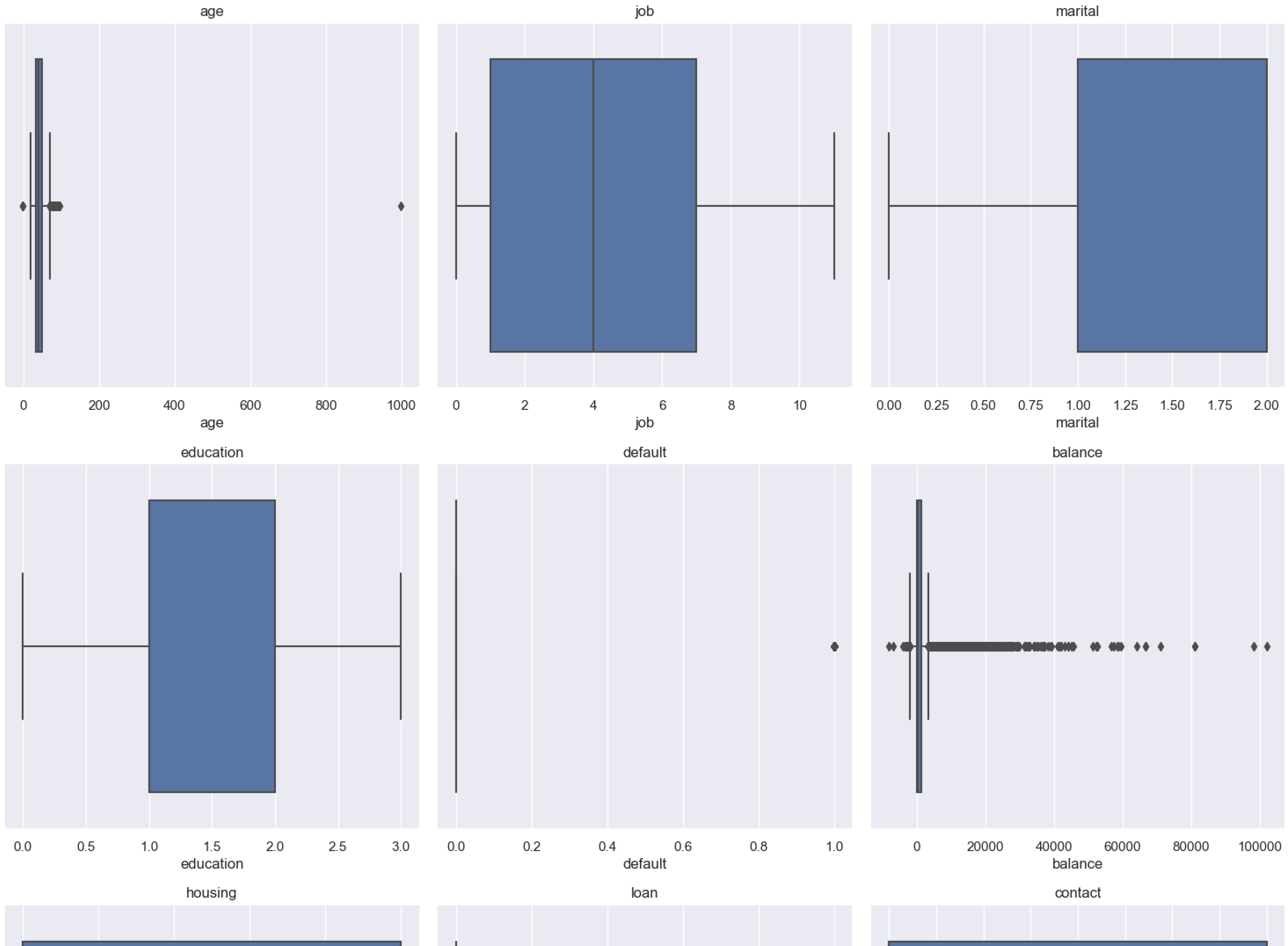
remove outer from train data using z_score:

In [166…

```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['age', 'balance', 'duration',
                    'campaign', 'pdays', 'previous']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

In [167…

```
# Get the names of all columns with data type 'int' or 'float'
cols_names = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(cols_names)
num_rows = (num_cols + 2)
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each numerical variable using Seaborn
for i,names in enumerate(cols_names):
    sns.boxplot(x=df[names], ax=axs[i])
    axs[i].set_title(names)

# Remove any extra empty subplots if needed
if  num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


age
0
200
400
600
800
1000
age
job
0
2
4
6
8
10
job
marital
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
marital
education
0.0
0.5
1.0
1.5
2.0
2.5
3.0
education
default
0.0
0.2
0.4
0.6
0.8
1.0
default
balance
0
20000
40000
60000
80000
100000
balance
housing
loan
contact

Decision Tree Classifier:

In [153…

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier(class_weight='balanced')
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train,y_train)
# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split': 2, 'random_state': 0}
```

In [154…

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=7, min_samples_leaf=2, min_samples_split=2, class_weight='balanced')
dtree.fit(X_train, y_train)
```

Out[154]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=7, min_samples_leaf=2,
                       random_state=0)
```

In [155…

```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 83.66 %
```

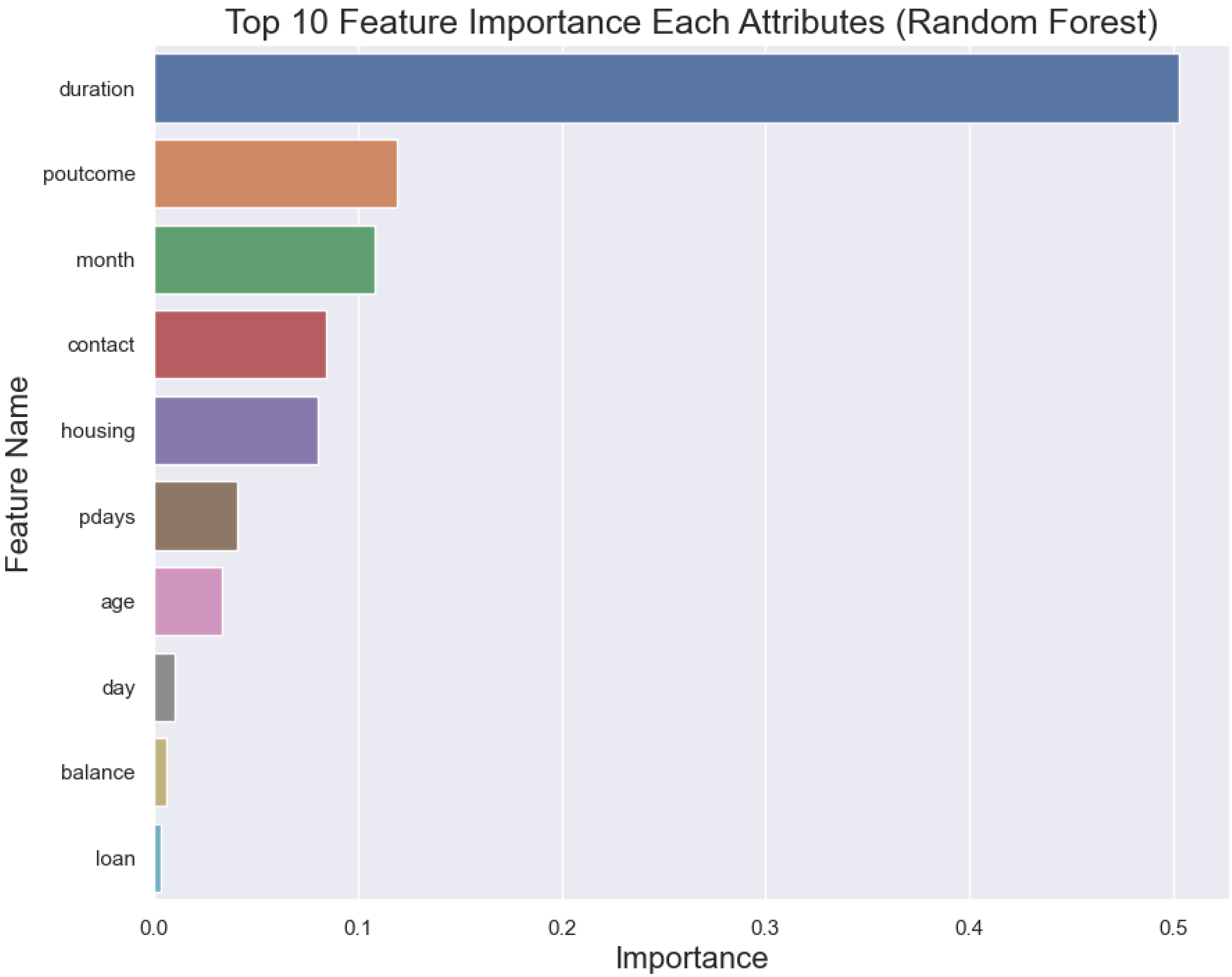
In [156…

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8366150442477877
Precision Score :  0.8366150442477877
Recall Score :  0.8366150442477877
Jaccard Score :  0.7191214224588761
Log Loss :  5.643226928370951
```

In [160…

```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```

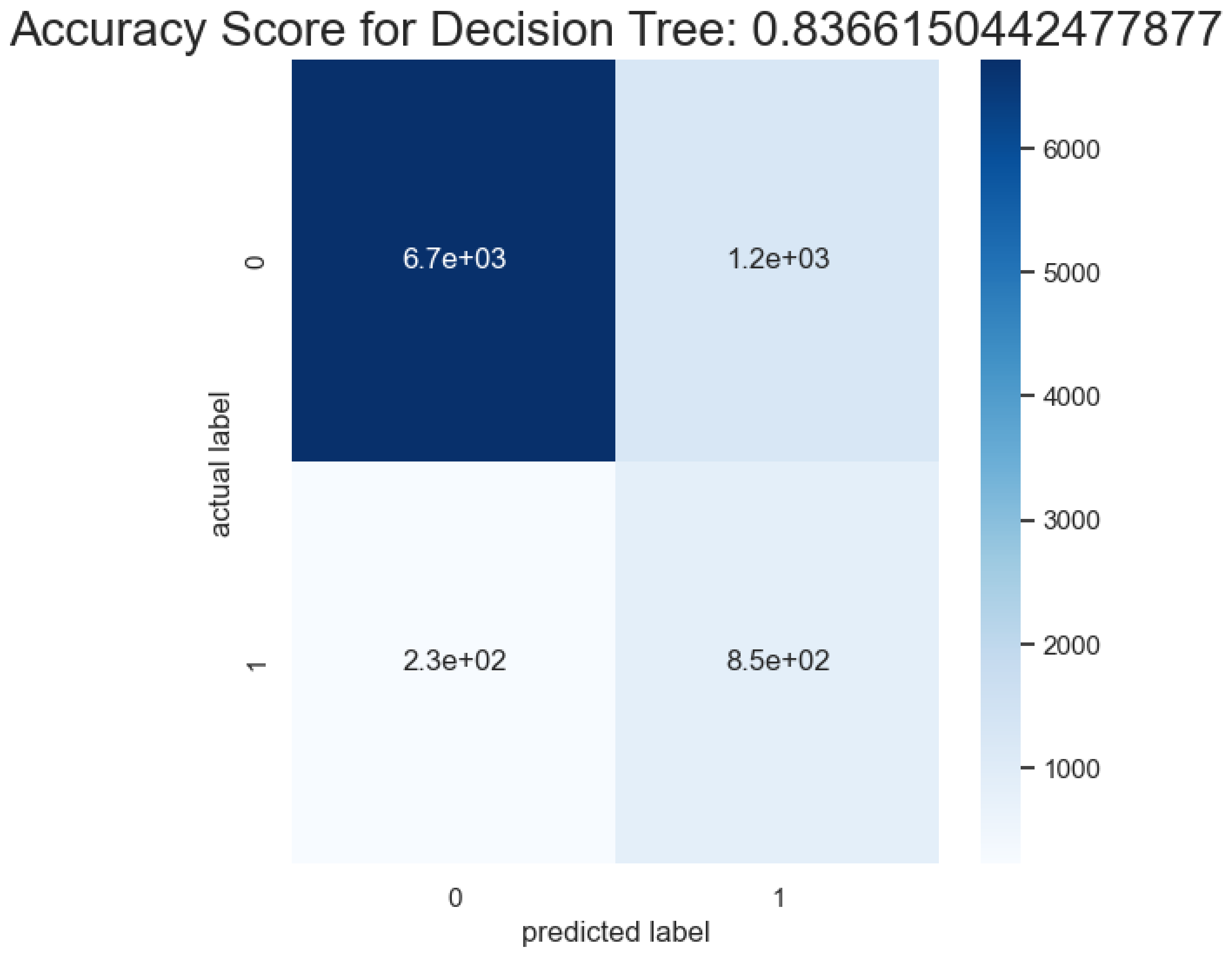


In [164…

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
plt.figure(figsize=(6,6))
sns.heatmap(data = cm ,annot=True,cmap='Blues')
plt.xlabel('predicted label')
plt.ylabel('actual label')
accuracy_score ='Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(accuracy_score,size=20)
```

Out[164]:

```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8366150442477877')
```

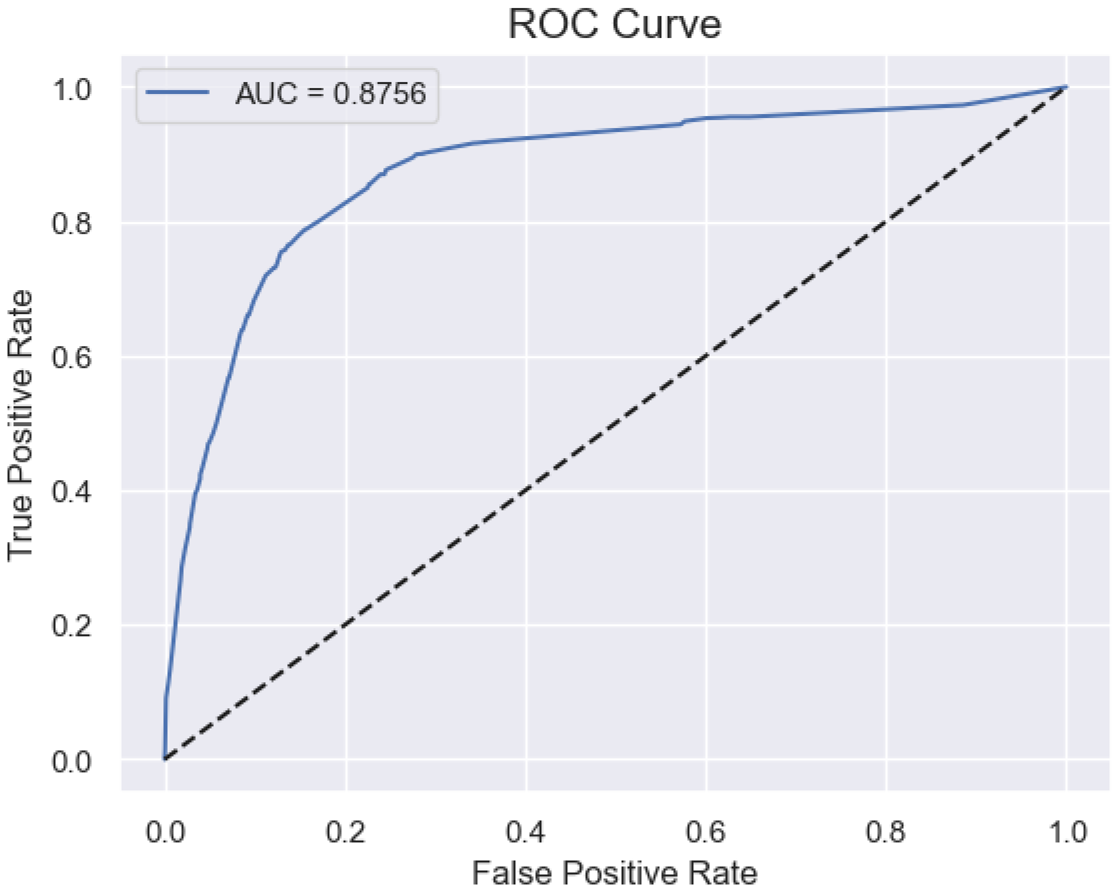


In [165…

```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba, columns=['y_pred_proba'])], axis=1)
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[165]:

```
<matplotlib.legend.Legend at 0x26c6bbac130>
```


ROC Curve
AUC = 0.8756
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0