import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.set()

file\_path = 'bank-additional-full.csv'  
df = pd.read\_csv(file\_path,sep=";")  
print(df.head())

age job marital education default housing loan contact \  
0 56 housemaid married basic.4y no no no telephone   
1 57 services married high.school unknown no no telephone   
2 37 services married high.school no yes no telephone   
3 40 admin. married basic.6y no no no telephone   
4 56 services married high.school no no yes telephone   
  
 month day\_of\_week ... campaign pdays previous poutcome emp.var.rate \  
0 may mon ... 1 999 0 nonexistent 1.1   
1 may mon ... 1 999 0 nonexistent 1.1   
2 may mon ... 1 999 0 nonexistent 1.1   
3 may mon ... 1 999 0 nonexistent 1.1   
4 may mon ... 1 999 0 nonexistent 1.1   
  
 cons.price.idx cons.conf.idx euribor3m nr.employed y   
0 93.994 -36.4 4.857 5191.0 no   
1 93.994 -36.4 4.857 5191.0 no   
2 93.994 -36.4 4.857 5191.0 no   
3 93.994 -36.4 4.857 5191.0 no   
4 93.994 -36.4 4.857 5191.0 no   
  
[5 rows x 21 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 41188 entries, 0 to 41187  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 41188 non-null int64   
 1 job 41188 non-null object   
 2 marital 41188 non-null object   
 3 education 41188 non-null object   
 4 default 41188 non-null object   
 5 housing 41188 non-null object   
 6 loan 41188 non-null object   
 7 contact 41188 non-null object   
 8 month 41188 non-null object   
 9 day\_of\_week 41188 non-null object   
 10 duration 41188 non-null int64   
 11 campaign 41188 non-null int64   
 12 pdays 41188 non-null int64   
 13 previous 41188 non-null int64   
 14 poutcome 41188 non-null object   
 15 emp.var.rate 41188 non-null float64  
 16 cons.price.idx 41188 non-null float64  
 17 cons.conf.idx 41188 non-null float64  
 18 euribor3m 41188 non-null float64  
 19 nr.employed 41188 non-null float64  
 20 y 41188 non-null object   
dtypes: float64(5), int64(5), object(11)  
memory usage: 6.6+ MB

print(df.describe())

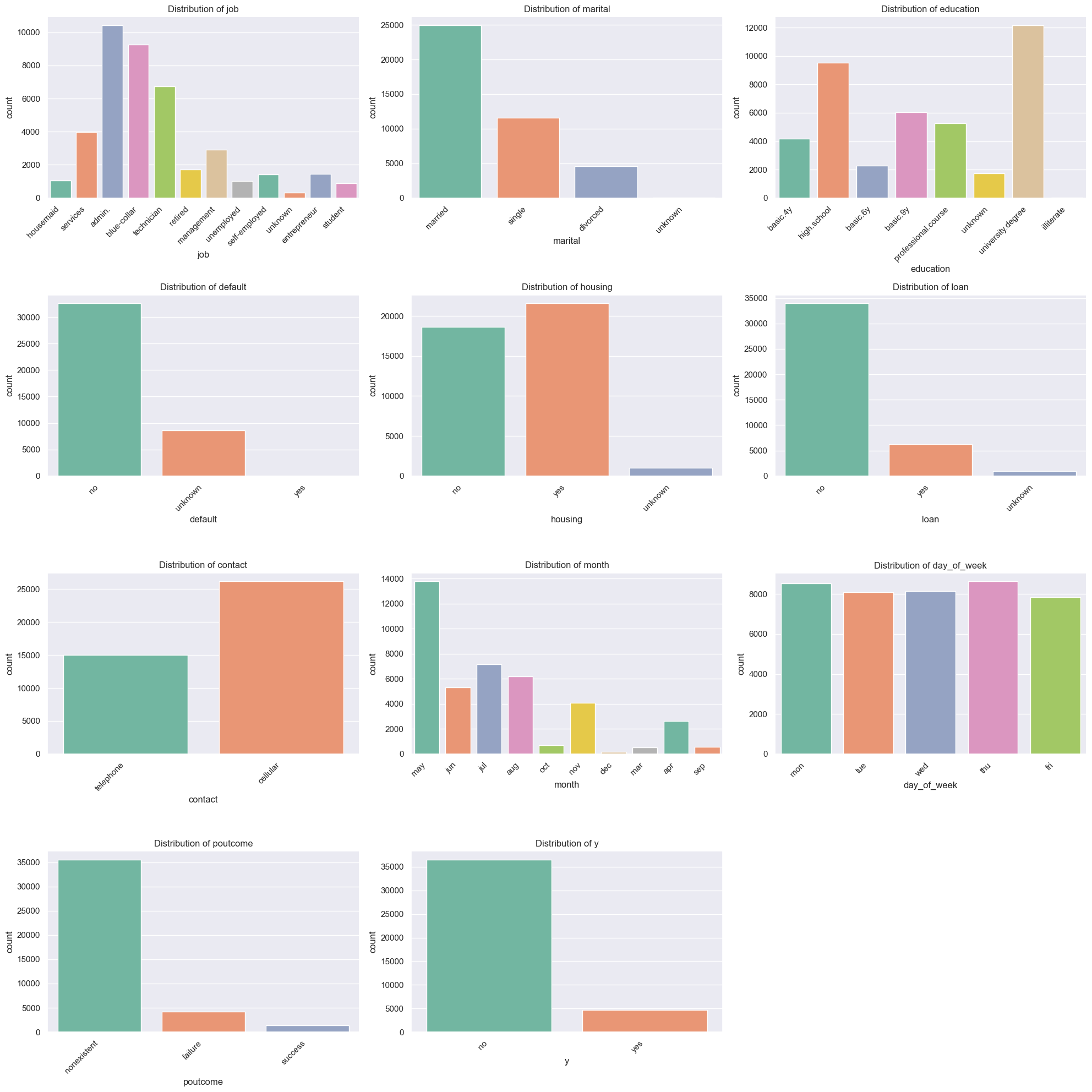
age duration campaign pdays previous \  
count 41188.00000 41188.000000 41188.000000 41188.000000 41188.000000   
mean 40.02406 258.285010 2.567593 962.475454 0.172963   
std 10.42125 259.279249 2.770014 186.910907 0.494901   
min 17.00000 0.000000 1.000000 0.000000 0.000000   
25% 32.00000 102.000000 1.000000 999.000000 0.000000   
50% 38.00000 180.000000 2.000000 999.000000 0.000000   
75% 47.00000 319.000000 3.000000 999.000000 0.000000   
max 98.00000 4918.000000 56.000000 999.000000 7.000000   
  
 emp.var.rate cons.price.idx cons.conf.idx euribor3m nr.employed   
count 41188.000000 41188.000000 41188.000000 41188.000000 41188.000000   
mean 0.081886 93.575664 -40.502600 3.621291 5167.035911   
std 1.570960 0.578840 4.628198 1.734447 72.251528   
min -3.400000 92.201000 -50.800000 0.634000 4963.600000   
25% -1.800000 93.075000 -42.700000 1.344000 5099.100000   
50% 1.100000 93.749000 -41.800000 4.857000 5191.000000   
75% 1.400000 93.994000 -36.400000 4.961000 5228.100000   
max 1.400000 94.767000 -26.900000 5.045000 5228.100000

## **1. Analyzing the data Set**

# cheak null value  
df.isnull().sum()

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

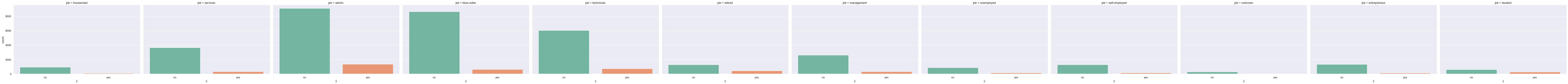
categorical\_columns = df.select\_dtypes(include="object").columns  
  
  
#categorical features visual  
n\_cols = 3   
n\_rows = (len(categorical\_columns) // n\_cols) + 1   
  
  
plt.figure(figsize=(20, 5 \* n\_rows))  
plt\_number = 1  
  
for column in categorical\_columns:  
   
 ax = plt.subplot(n\_rows, n\_cols, plt\_number)  
   
 sns.countplot(x=column, data=df, hue=column, palette='Set2', legend=False) # Assign column to hue  
   
  
 plt.xlabel(column)   
 plt.title(f'Distribution of {column}')  
 plt.xticks(rotation=45, ha='right')   
 plt\_number += 1  
  
plt.tight\_layout()  
  
plt.show()

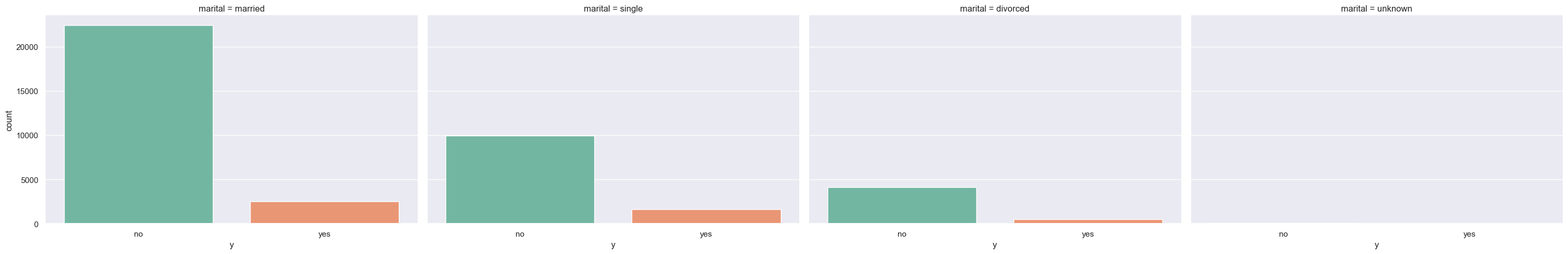


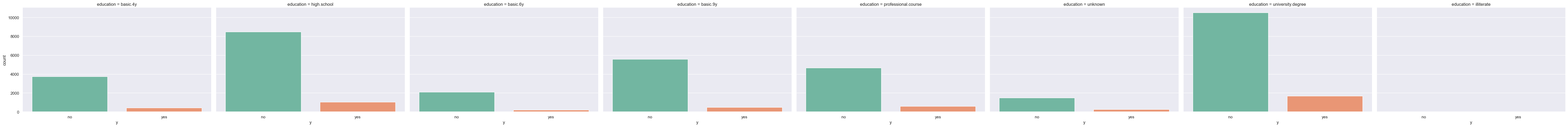
**Findings**

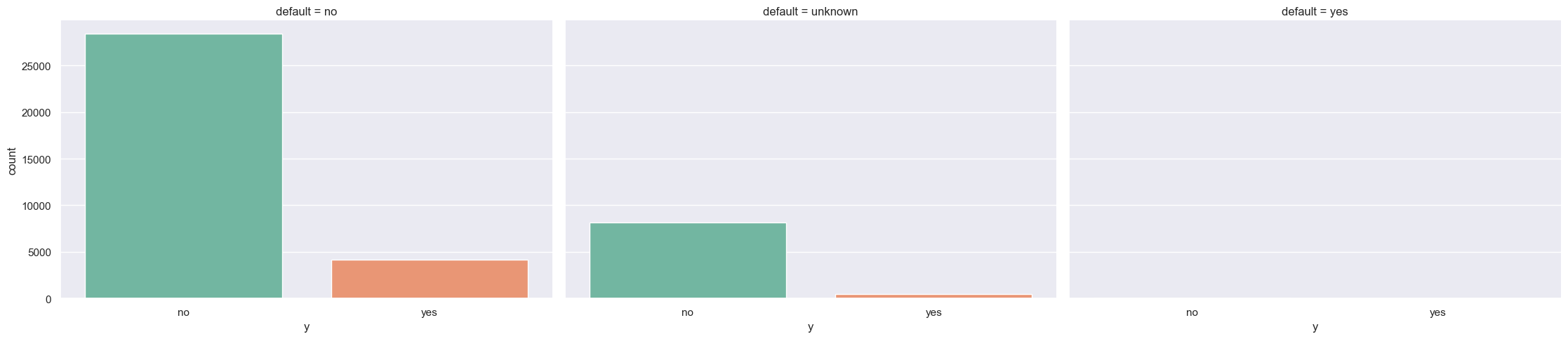
* we can find the categories of the categorical fetures and the count of it using this
* data we can understand the type of clients and the backround of them
* outcome of the previous marketing campaign
* number the client subscribed a term deposit

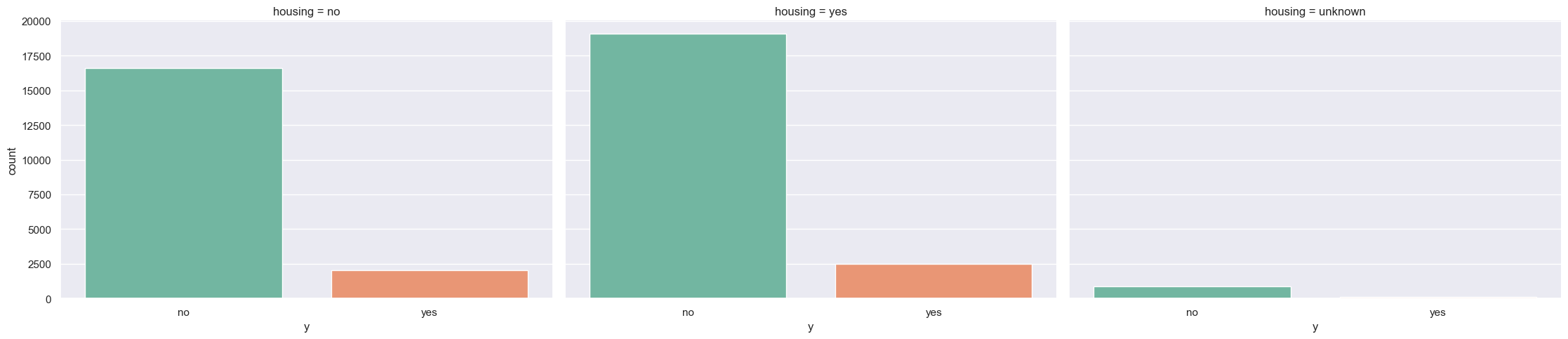
# categorical features and the target  
for col in categorical\_columns:  
 # Create a count plot for the target variable 'y' across each categorical column  
 sns.catplot(x="y", hue="y", col=col, kind="count", data=df, height=5, aspect=1.5, palette='Set2', legend=False)  
  
# Show the plots  
plt.show()

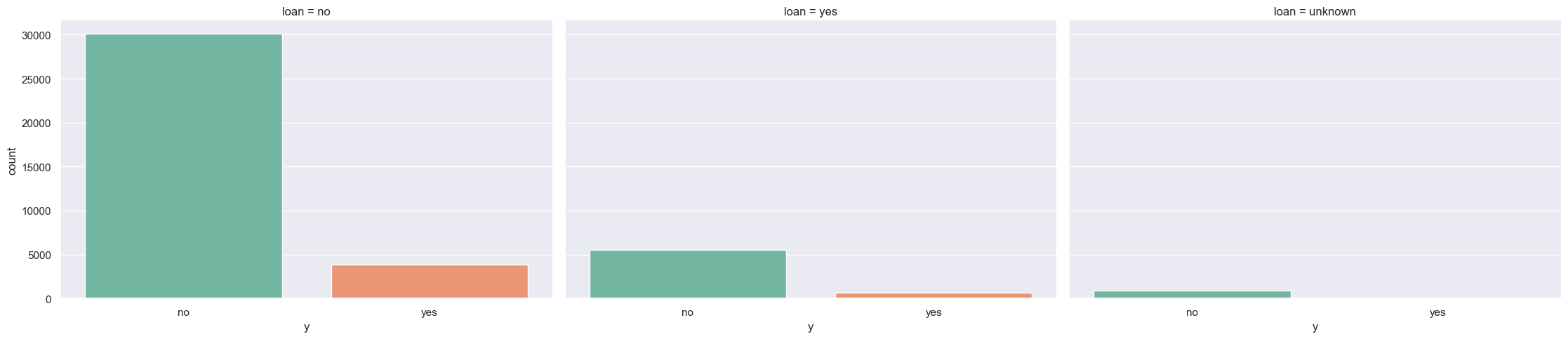


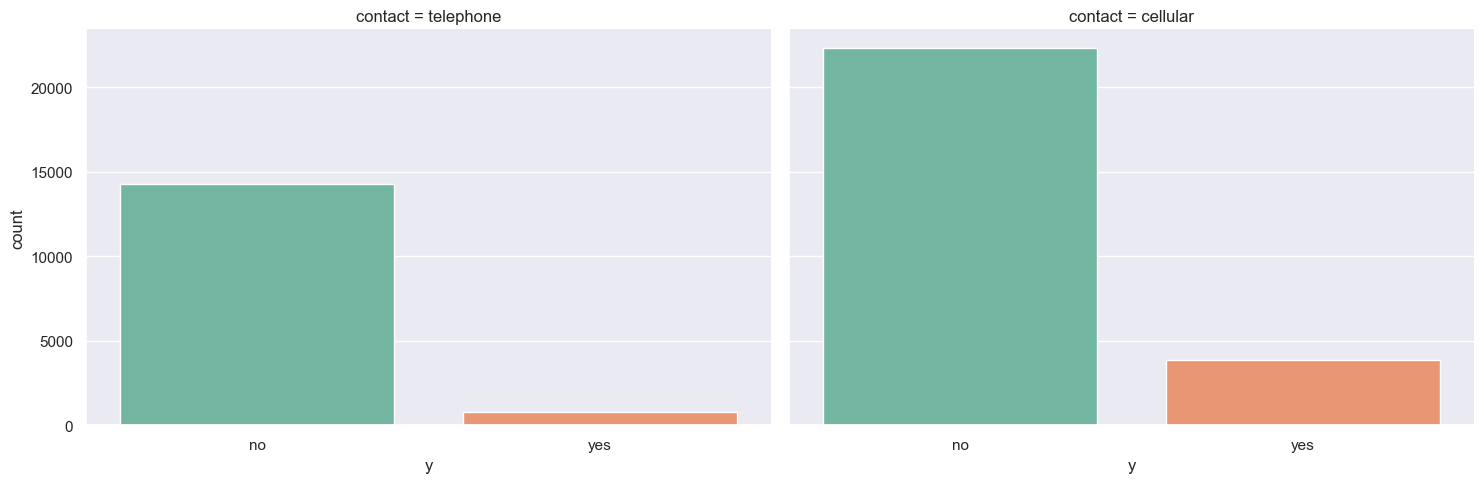


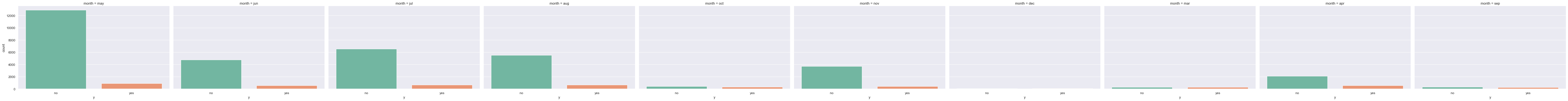


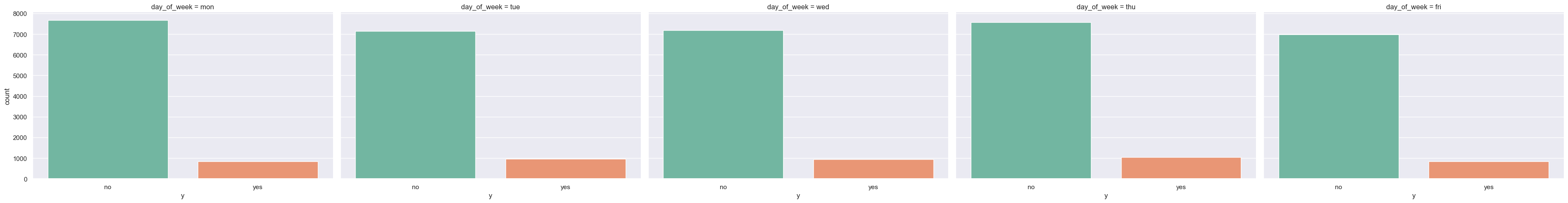


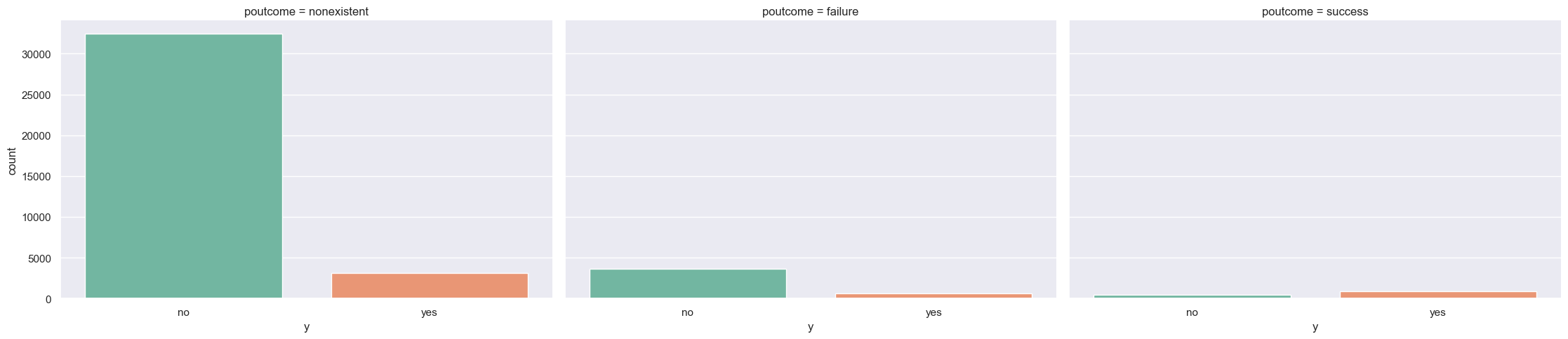


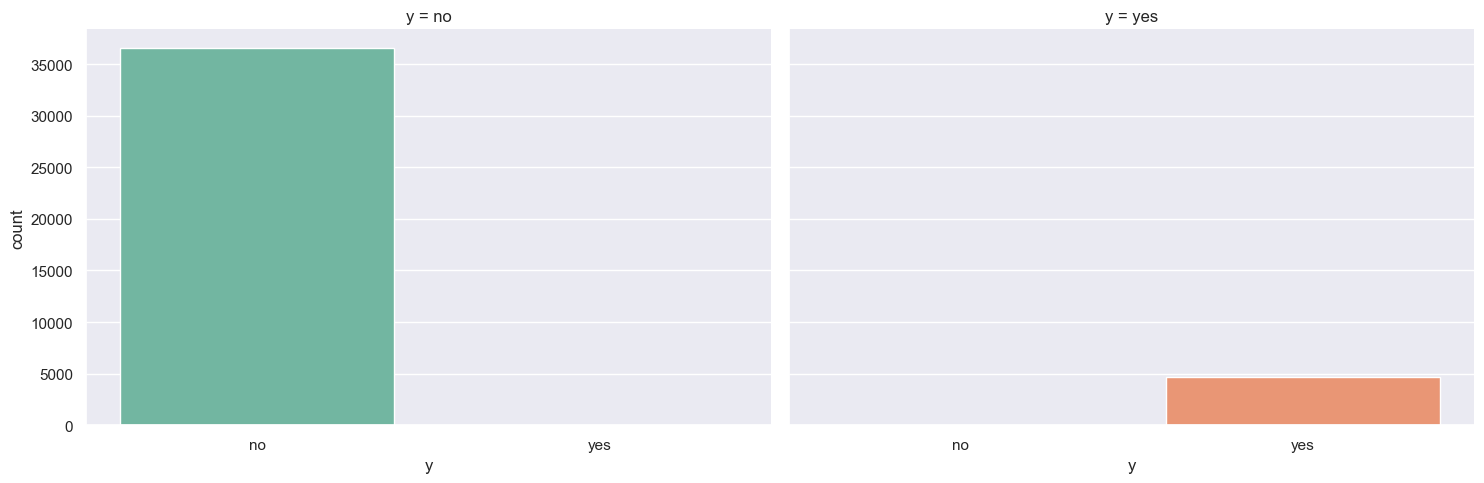












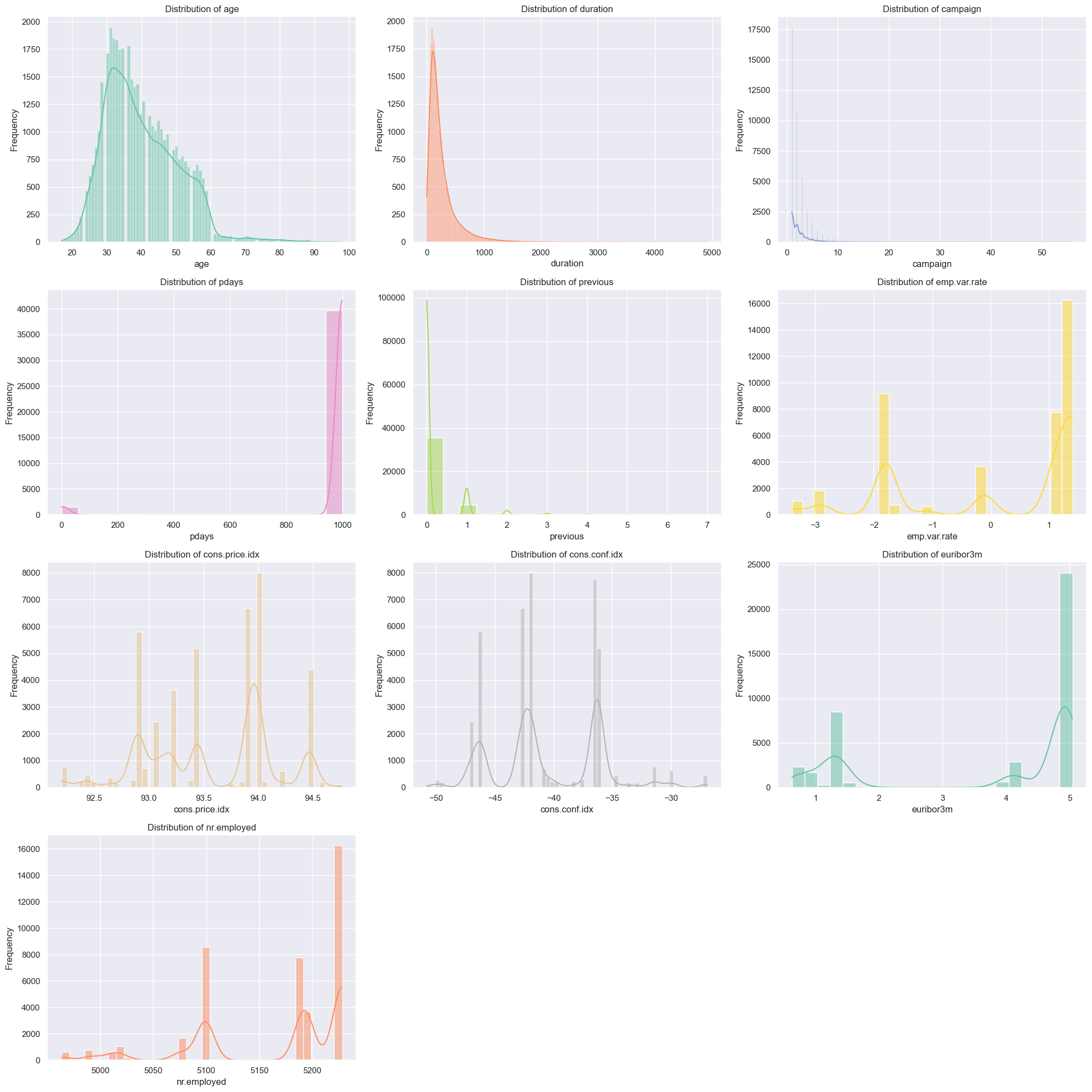
**Findings**

* according to the client type how many of them subscribe this
* outcome of the pre campaign
* the month clients show there highest interest in deposit

numerical\_columns = df.select\_dtypes(include=['number']).columns # Identify categorical columns  
for column in numerical\_columns:  
 print(f"{column}")

age  
duration  
campaign  
pdays  
previous  
emp.var.rate  
cons.price.idx  
cons.conf.idx  
euribor3m  
nr.employed

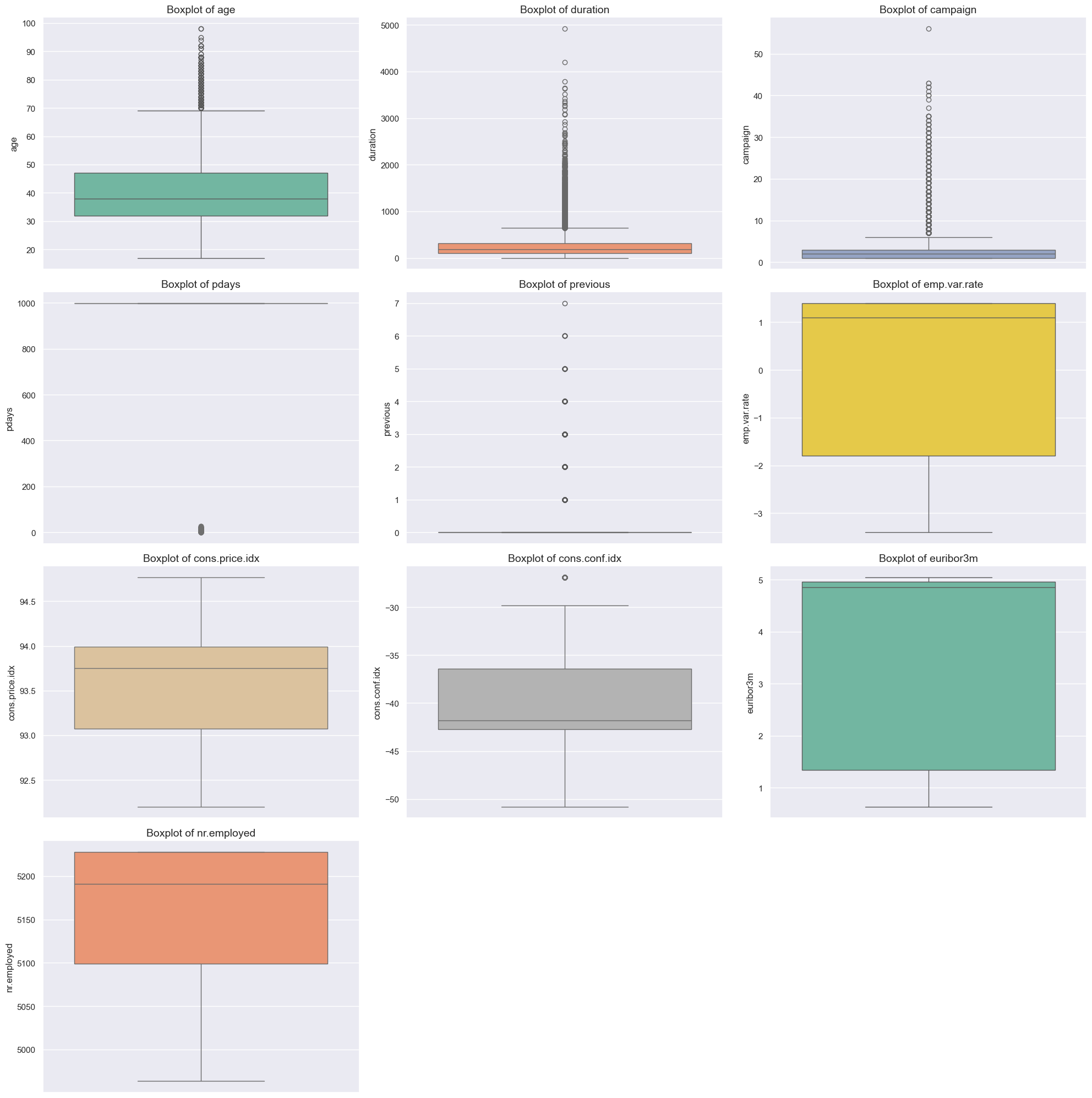
colors = sns.color\_palette("Set2", n\_colors=len(numerical\_columns))  
  
# Set up the figure with enough space for all plots  
n\_cols = 3 # You can change the number of columns as needed  
n\_rows = (len(numerical\_columns) // n\_cols) + 1 # Calculate number of rows  
  
plt.figure(figsize=(20, 5 \* n\_rows)) # Adjust the figure size based on number of rows  
plt\_number = 1  
  
# Loop through numerical columns and create histograms with KDE and different colors  
for idx, col in enumerate(numerical\_columns):  
 ax = plt.subplot(n\_rows, n\_cols, plt\_number)  
 sns.histplot(df[col], kde=True, color=colors[idx]) # Use a different color for each plot  
 plt.xlabel(col)   
 plt.ylabel('Frequency')   
 plt.title(f'Distribution of {col}')   
 plt\_number += 1  
  
plt.tight\_layout() # Adjust layout to avoid overlap  
plt.show()



**Findings**

* which range of the clients doing the deposits in there features like age, balance and etc

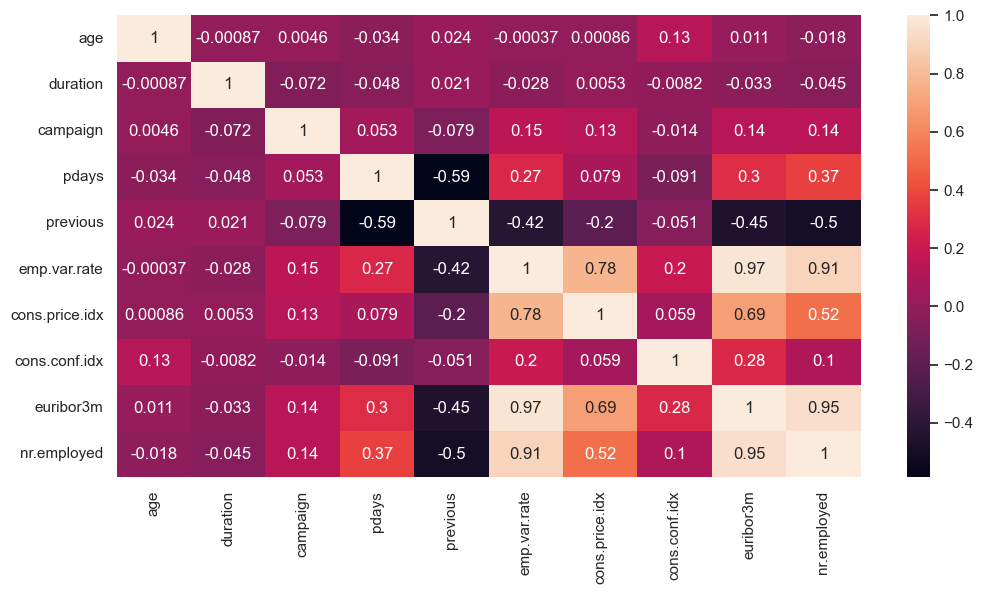
# Define a list of colors for different boxplots  
colors = sns.color\_palette("Set2", len(numerical\_columns))  
  
# Adjust figure size to accommodate many plots  
plt.figure(figsize=(20, 60))  
  
# Counter for subplot position  
plt\_number = 1  
  
# Loop through each numerical column to create box plots  
for idx, col in enumerate(numerical\_columns):  
 # Create subplot  
 ax = plt.subplot(12, 3, plt\_number)  
   
 # Create boxplot for the numerical feature with a unique color  
 sns.boxplot(y=df[col], color=colors[idx])  
   
 # Add labels and title  
 plt.ylabel(col, fontsize=12)  
 plt.title(f'Boxplot of {col}', fontsize=14)  
   
 # Increment subplot counter  
 plt\_number += 1  
  
# Adjust layout for better spacing  
plt.tight\_layout()  
  
# Display the plots  
plt.show()



**Findings**

* age , duration, campaign, pdays and previous has some outliers
* in the box plot of pdays most of the numvbers near to the 999 (witch mean NAN)
* in previous most of the numbers are 0 and the outliers are the only thing that have some other numbers

# Correlation between numerical fetures  
cor = df.select\_dtypes(include=['number']).corr()  
fig = plt.figure(figsize=(12,6))  
sns.heatmap(cor, annot= True)  
  
plt.show()



**Findings**

* euribor3m, employed, emp.var.rate are have correlation over 0.95

## **2. Feature Engneering**

df2 = df.copy()  
print(df2.head())

age job marital education default housing loan contact \  
0 56 housemaid married basic.4y no no no telephone   
1 57 services married high.school unknown no no telephone   
2 37 services married high.school no yes no telephone   
3 40 admin. married basic.6y no no no telephone   
4 56 services married high.school no no yes telephone   
  
 month day\_of\_week ... campaign pdays previous poutcome emp.var.rate \  
0 may mon ... 1 999 0 nonexistent 1.1   
1 may mon ... 1 999 0 nonexistent 1.1   
2 may mon ... 1 999 0 nonexistent 1.1   
3 may mon ... 1 999 0 nonexistent 1.1   
4 may mon ... 1 999 0 nonexistent 1.1   
  
 cons.price.idx cons.conf.idx euribor3m nr.employed y   
0 93.994 -36.4 4.857 5191.0 no   
1 93.994 -36.4 4.857 5191.0 no   
2 93.994 -36.4 4.857 5191.0 no   
3 93.994 -36.4 4.857 5191.0 no   
4 93.994 -36.4 4.857 5191.0 no   
  
[5 rows x 21 columns]

df2.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 41188 entries, 0 to 41187  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 41188 non-null int64   
 1 job 41188 non-null object   
 2 marital 41188 non-null object   
 3 education 41188 non-null object   
 4 default 41188 non-null object   
 5 housing 41188 non-null object   
 6 loan 41188 non-null object   
 7 contact 41188 non-null object   
 8 month 41188 non-null object   
 9 day\_of\_week 41188 non-null object   
 10 duration 41188 non-null int64   
 11 campaign 41188 non-null int64   
 12 pdays 41188 non-null int64   
 13 previous 41188 non-null int64   
 14 poutcome 41188 non-null object   
 15 emp.var.rate 41188 non-null float64  
 16 cons.price.idx 41188 non-null float64  
 17 cons.conf.idx 41188 non-null float64  
 18 euribor3m 41188 non-null float64  
 19 nr.employed 41188 non-null float64  
 20 y 41188 non-null object   
dtypes: float64(5), int64(5), object(11)  
memory usage: 6.6+ MB

### 2.1 Removing unwanted colums and outliers

df2.drop('duration', axis=1, inplace=True)

df2.groupby(['y','default']).size()

y default  
no no 28391  
 unknown 8154  
 yes 3  
yes no 4197  
 unknown 443  
dtype: int64

df2.drop('default', axis=1, inplace=True)

df2.groupby(['y','pdays']).size()

y pdays  
no 0 5  
 1 18  
 2 24  
 3 141  
 4 55  
 5 17  
 6 123  
 7 20  
 8 6  
 9 29  
 10 22  
 11 13  
 12 32  
 13 8  
 14 9  
 15 8  
 16 5  
 17 6  
 18 3  
 19 2  
 20 1  
 22 1  
 999 36000  
yes 0 10  
 1 8  
 2 37  
 3 298  
 4 63  
 5 29  
 6 289  
 7 40  
 8 12  
 9 35  
 10 30  
 11 15  
 12 26  
 13 28  
 14 11  
 15 16  
 16 6  
 17 2  
 18 4  
 19 1  
 21 2  
 22 2  
 25 1  
 26 1  
 27 1  
 999 3673  
dtype: int64

**Finding**

* in here 999 represent not contacted before and also previous they mentioned about how many time contacted so we need to drop

df2.drop('pdays', axis=1, inplace=True)

**Expalanation**

* last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

df2.groupby('age', sort= True)['age'].count()

age  
17 5  
18 28  
19 42  
20 65  
21 102  
 ...   
91 2  
92 4  
94 1  
95 1  
98 2  
Name: age, Length: 78, dtype: int64

**Finding**

* age is importand to predict the value
* 17 - 98 is more common age gap

df2.groupby(['y','campaign'], sort= True)['campaign'].count()

y campaign  
no 1 15342  
 2 9359  
 3 4767  
 4 2402  
 5 1479  
 6 904  
 7 591  
 8 383  
 9 266  
 10 213  
 11 165  
 12 122  
 13 88  
 14 68  
 15 49  
 16 51  
 17 54  
 18 33  
 19 26  
 20 30  
 21 24  
 22 17  
 23 15  
 24 15  
 25 8  
 26 8  
 27 11  
 28 8  
 29 10  
 30 7  
 31 7  
 32 4  
 33 4  
 34 3  
 35 5  
 37 1  
 39 1  
 40 2  
 41 1  
 42 2  
 43 2  
 56 1  
yes 1 2300  
 2 1211  
 3 574  
 4 249  
 5 120  
 6 75  
 7 38  
 8 17  
 9 17  
 10 12  
 11 12  
 12 3  
 13 4  
 14 1  
 15 2  
 17 4  
 23 1  
Name: campaign, dtype: int64

**Findings**

* in some distribtion are much skewed on left side so thy seems to have outliers

df2.groupby(['y','previous'], sort= True)['campaign'].count()

y previous  
no 0 32422  
 1 3594  
 2 404  
 3 88  
 4 32  
 5 5  
 6 2  
 7 1  
yes 0 3141  
 1 967  
 2 350  
 3 128  
 4 38  
 5 13  
 6 3  
Name: campaign, dtype: int64

### 2.2 Encoding dataset

df2['education'].value\_counts()

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

df\_encoded = pd.get\_dummies(df2, columns=['job', 'marital', 'contact', 'month', 'poutcome','day\_of\_week'])

education\_mapping = {  
 'illiterate': 1,  
 'unknown': 2,  
 'basic.4y': 3,  
 'basic.6y': 4,  
 'basic.9y': 5,  
 'high.school': 6,  
 'professional.course': 7,  
 'university.degree': 8  
}  
  
# Apply the mapping to the 'education' column  
df\_encoded['education'] = df2['education'].map(education\_mapping)

from sklearn.preprocessing import StandardScaler  
  
binary\_col = ['housing','loan','y']  
for col in binary\_col:  
 df\_encoded[col] = df\_encoded[col].apply(lambda x : True if x == 'yes' else False)

print(df\_encoded.head())

age education housing loan campaign previous emp.var.rate \  
0 56 3 False False 1 0 1.1   
1 57 6 False False 1 0 1.1   
2 37 6 True False 1 0 1.1   
3 40 4 False False 1 0 1.1   
4 56 6 False True 1 0 1.1   
  
 cons.price.idx cons.conf.idx euribor3m ... month\_oct month\_sep \  
0 93.994 -36.4 4.857 ... False False   
1 93.994 -36.4 4.857 ... False False   
2 93.994 -36.4 4.857 ... False False   
3 93.994 -36.4 4.857 ... False False   
4 93.994 -36.4 4.857 ... False False   
  
 poutcome\_failure poutcome\_nonexistent poutcome\_success day\_of\_week\_fri \  
0 False True False False   
1 False True False False   
2 False True False False   
3 False True False False   
4 False True False False   
  
 day\_of\_week\_mon day\_of\_week\_thu day\_of\_week\_tue day\_of\_week\_wed   
0 True False False False   
1 True False False False   
2 True False False False   
3 True False False False   
4 True False False False   
  
[5 rows x 48 columns]

print(df\_encoded[['emp.var.rate', 'euribor3m', 'nr.employed', 'y']].corr())

emp.var.rate euribor3m nr.employed y  
emp.var.rate 1.000000 0.972245 0.906970 -0.298334  
euribor3m 0.972245 1.000000 0.945154 -0.307771  
nr.employed 0.906970 0.945154 1.000000 -0.354678  
y -0.298334 -0.307771 -0.354678 1.000000

**Findings**

* emp.var.rate, euribor3m, nr.employed higher corrilation with each other so need to remove two other which have less corrilation with lable
* When we test the corrilation with y lable for the all variable have simmiler corr so instead of removing donig PCA will be much effective

# Doing PCA and make it as a feature  
from sklearn.decomposition import PCA  
from sklearn.preprocessing import StandardScaler  
  
  
features = df\_encoded[['emp.var.rate', 'euribor3m', 'nr.employed']]  
  
  
scaler = StandardScaler()  
scaled\_features = scaler.fit\_transform(features)  
  
  
pca = PCA(n\_components=1)  
pca\_features = pca.fit\_transform(scaled\_features)  
  
df\_encoded['pca\_feature'] = pca\_features  
  
df\_encoded.drop(['emp.var.rate', 'euribor3m', 'nr.employed'], axis=1, inplace=True)

print(df\_encoded.columns)

Index(['age', 'education', 'housing', 'loan', 'campaign', 'previous',  
 'cons.price.idx', 'cons.conf.idx', 'y', 'job\_admin.', 'job\_blue-collar',  
 'job\_entrepreneur', 'job\_housemaid', 'job\_management', 'job\_retired',  
 'job\_self-employed', 'job\_services', 'job\_student', 'job\_technician',  
 'job\_unemployed', 'job\_unknown', 'marital\_divorced', 'marital\_married',  
 'marital\_single', 'marital\_unknown', 'contact\_cellular',  
 'contact\_telephone', 'month\_apr', 'month\_aug', 'month\_dec', 'month\_jul',  
 'month\_jun', 'month\_mar', 'month\_may', 'month\_nov', 'month\_oct',  
 'month\_sep', 'poutcome\_failure', 'poutcome\_nonexistent',  
 'poutcome\_success', 'day\_of\_week\_fri', 'day\_of\_week\_mon',  
 'day\_of\_week\_thu', 'day\_of\_week\_tue', 'day\_of\_week\_wed', 'pca\_feature'],  
 dtype='object')

df\_encoded.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 41188 entries, 0 to 41187  
Data columns (total 46 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 41188 non-null int64   
 1 education 41188 non-null int64   
 2 housing 41188 non-null bool   
 3 loan 41188 non-null bool   
 4 campaign 41188 non-null int64   
 5 previous 41188 non-null int64   
 6 cons.price.idx 41188 non-null float64  
 7 cons.conf.idx 41188 non-null float64  
 8 y 41188 non-null bool   
 9 job\_admin. 41188 non-null bool   
 10 job\_blue-collar 41188 non-null bool   
 11 job\_entrepreneur 41188 non-null bool   
 12 job\_housemaid 41188 non-null bool   
 13 job\_management 41188 non-null bool   
 14 job\_retired 41188 non-null bool   
 15 job\_self-employed 41188 non-null bool   
 16 job\_services 41188 non-null bool   
 17 job\_student 41188 non-null bool   
 18 job\_technician 41188 non-null bool   
 19 job\_unemployed 41188 non-null bool   
 20 job\_unknown 41188 non-null bool   
 21 marital\_divorced 41188 non-null bool   
 22 marital\_married 41188 non-null bool   
 23 marital\_single 41188 non-null bool   
 24 marital\_unknown 41188 non-null bool   
 25 contact\_cellular 41188 non-null bool   
 26 contact\_telephone 41188 non-null bool   
 27 month\_apr 41188 non-null bool   
 28 month\_aug 41188 non-null bool   
 29 month\_dec 41188 non-null bool   
 30 month\_jul 41188 non-null bool   
 31 month\_jun 41188 non-null bool   
 32 month\_mar 41188 non-null bool   
 33 month\_may 41188 non-null bool   
 34 month\_nov 41188 non-null bool   
 35 month\_oct 41188 non-null bool   
 36 month\_sep 41188 non-null bool   
 37 poutcome\_failure 41188 non-null bool   
 38 poutcome\_nonexistent 41188 non-null bool   
 39 poutcome\_success 41188 non-null bool   
 40 day\_of\_week\_fri 41188 non-null bool   
 41 day\_of\_week\_mon 41188 non-null bool   
 42 day\_of\_week\_thu 41188 non-null bool   
 43 day\_of\_week\_tue 41188 non-null bool   
 44 day\_of\_week\_wed 41188 non-null bool   
 45 pca\_feature 41188 non-null float64  
dtypes: bool(39), float64(3), int64(4)  
memory usage: 3.7 MB

from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
# Separate features and target  
X = df\_encoded.drop(columns=["y"]) # Replace "target" with your actual target column name  
y = df\_encoded["y"]  
le = LabelEncoder()  
y = le.fit\_transform(y)  
  
# Split the dataset into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

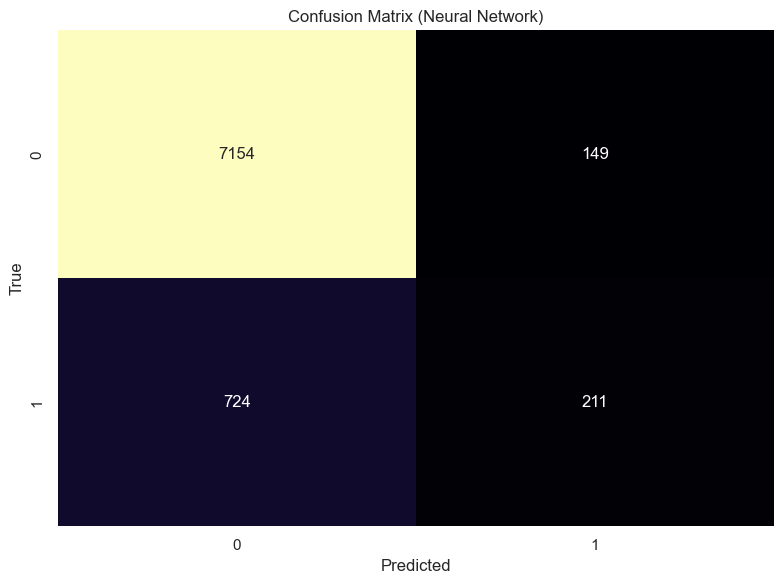
# Plotting the distribution of the target variable in both training and testing sets  
plt.figure(figsize=(12, 6))  
  
# Plot for the training set with a specific color palette  
plt.subplot(1, 2, 1)  
sns.countplot(x=y\_train, hue=y\_train, palette='Blues', legend=False)  
plt.title("Target Distribution in Training Set")  
  
# Plot for the testing set with a different color palette  
plt.subplot(1, 2, 2)  
sns.countplot(x=y\_test, hue=y\_test, palette='Oranges', legend=False)  
plt.title("Target Distribution in Test Set")  
  
# Display the plots  
plt.tight\_layout()  
plt.show()



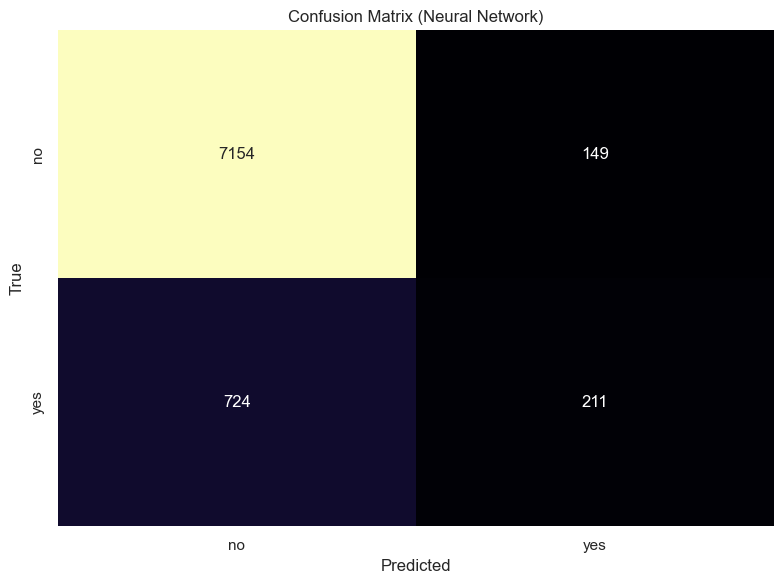
### Training Without Balanced Dataset

from sklearn.preprocessing import StandardScaler  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Input, Dense  
from tensorflow.keras.callbacks import EarlyStopping  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  
  
X\_train\_df = pd.DataFrame(X\_train)  
X\_test\_df = pd.DataFrame(X\_test)  
  
# Standardize the features  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train\_df) # Fit and transform on training data  
X\_test = scaler.transform(X\_test\_df) # Transform on test data  
  
  
# Define the model  
model\_no\_bal = Sequential([  
 Input(shape=(X\_train.shape[1],)), # Define input shape explicitly  
 Dense(64, activation='relu'), # Hidden layer 1  
 Dense(32, activation='relu'), # Hidden layer 2  
 Dense(1, activation='sigmoid') # Output layer for binary classification  
])  
  
# Compile the model  
model\_no\_bal.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
# Early stopping to prevent overfitting  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)  
  
# Train the model  
history = model\_no\_bal.fit(  
 X\_train, y\_train,  
 epochs=20,  
 batch\_size=32,  
 validation\_data=(X\_test, y\_test),  
 callbacks=[early\_stopping],  
)  
  
# Evaluate the Neural Network model  
results\_nn = model\_no\_bal.evaluate(X\_test, y\_test)  
print("\nNeural Network Test Loss:", results\_nn[0])  
print("Neural Network Test Accuracy:", results\_nn[1])  
  
# Make predictions using the Neural Network model  
y\_pred\_nn = (model\_no\_bal.predict(X\_test) > 0.5).astype("int32")  
  
# Evaluate the Neural Network model using metrics  
print("\nClassification Report (Neural Network):")  
print(classification\_report(y\_test, y\_pred\_nn))  
  
print("\nConfusion Matrix (Neural Network):")  
print(confusion\_matrix(y\_test, y\_pred\_nn))  
  
# Confusion matrix for Neural Network predictions  
cm\_nn = confusion\_matrix(y\_test, y\_pred\_nn)  
  
# Plot the confusion matrix with a different color palette  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm\_nn, annot=True, fmt='d', cmap='magma', cbar=False, xticklabels=np.unique(y\_test), yticklabels=np.unique(y\_test))  
  
plt.title('Confusion Matrix (Neural Network)')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.tight\_layout()  
plt.show()  
  
# Make predictions using the Neural Network model  
y\_pred\_nn = (model\_no\_bal.predict(X\_test) > 0.5).astype("int32")  
  
# Evaluate the Neural Network model using metrics  
print("\nClassification Report (Neural Network):")  
print(classification\_report(y\_test, y\_pred\_nn))  
  
print("\nConfusion Matrix (Neural Network):")  
print(confusion\_matrix(y\_test, y\_pred\_nn))  
  
# Confusion matrix for Neural Network predictions  
cm\_nn = confusion\_matrix(y\_test, y\_pred\_nn)  
  
# Plot the confusion matrix with a different color palette  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm\_nn, annot=True, fmt='d', cmap='magma', cbar=False, xticklabels=df["y"].unique(), yticklabels=df["y"].unique())  
  
plt.title('Confusion Matrix (Neural Network)')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.tight\_layout()  
plt.show()

Epoch 1/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8873 - loss: 0.3153 - val\_accuracy: 0.8971 - val\_loss: 0.2876  
Epoch 2/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 974us/step - accuracy: 0.9010 - loss: 0.2734 - val\_accuracy: 0.8966 - val\_loss: 0.2893  
Epoch 3/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 958us/step - accuracy: 0.9015 - loss: 0.2714 - val\_accuracy: 0.8940 - val\_loss: 0.2844  
Epoch 4/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 968us/step - accuracy: 0.9024 - loss: 0.2641 - val\_accuracy: 0.8916 - val\_loss: 0.2861  
Epoch 5/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 981us/step - accuracy: 0.9033 - loss: 0.2672 - val\_accuracy: 0.8937 - val\_loss: 0.2882  
Epoch 6/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 964us/step - accuracy: 0.9052 - loss: 0.2608 - val\_accuracy: 0.8946 - val\_loss: 0.2878  
Epoch 7/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 963us/step - accuracy: 0.9082 - loss: 0.2552 - val\_accuracy: 0.8904 - val\_loss: 0.2886  
Epoch 8/20  
1030/1030 ━━━━━━━━━━━━━━━━━━━━ 1s 974us/step - accuracy: 0.9081 - loss: 0.2577 - val\_accuracy: 0.8933 - val\_loss: 0.2878  
258/258 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - accuracy: 0.8942 - loss: 0.2874  
  
Neural Network Test Loss: 0.2844419777393341  
Neural Network Test Accuracy: 0.8940276503562927  
258/258 ━━━━━━━━━━━━━━━━━━━━ 0s 490us/step  
  
Classification Report (Neural Network):  
 precision recall f1-score support  
  
 0 0.91 0.98 0.94 7303  
 1 0.59 0.23 0.33 935  
  
 accuracy 0.89 8238  
 macro avg 0.75 0.60 0.63 8238  
weighted avg 0.87 0.89 0.87 8238  
  
  
Confusion Matrix (Neural Network):  
[[7154 149]  
 [ 724 211]]

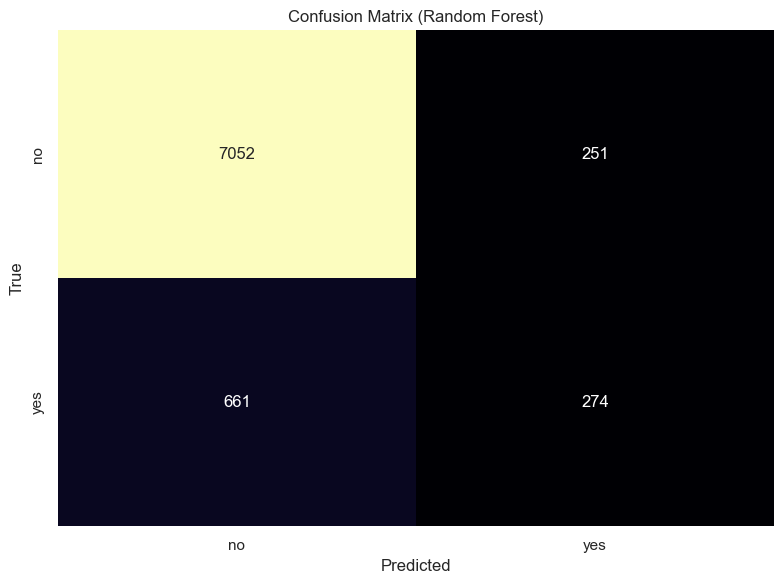


258/258 ━━━━━━━━━━━━━━━━━━━━ 0s 420us/step  
  
Classification Report (Neural Network):  
 precision recall f1-score support  
  
 0 0.91 0.98 0.94 7303  
 1 0.59 0.23 0.33 935  
  
 accuracy 0.89 8238  
 macro avg 0.75 0.60 0.63 8238  
weighted avg 0.87 0.89 0.87 8238  
  
  
Confusion Matrix (Neural Network):  
[[7154 149]  
 [ 724 211]]

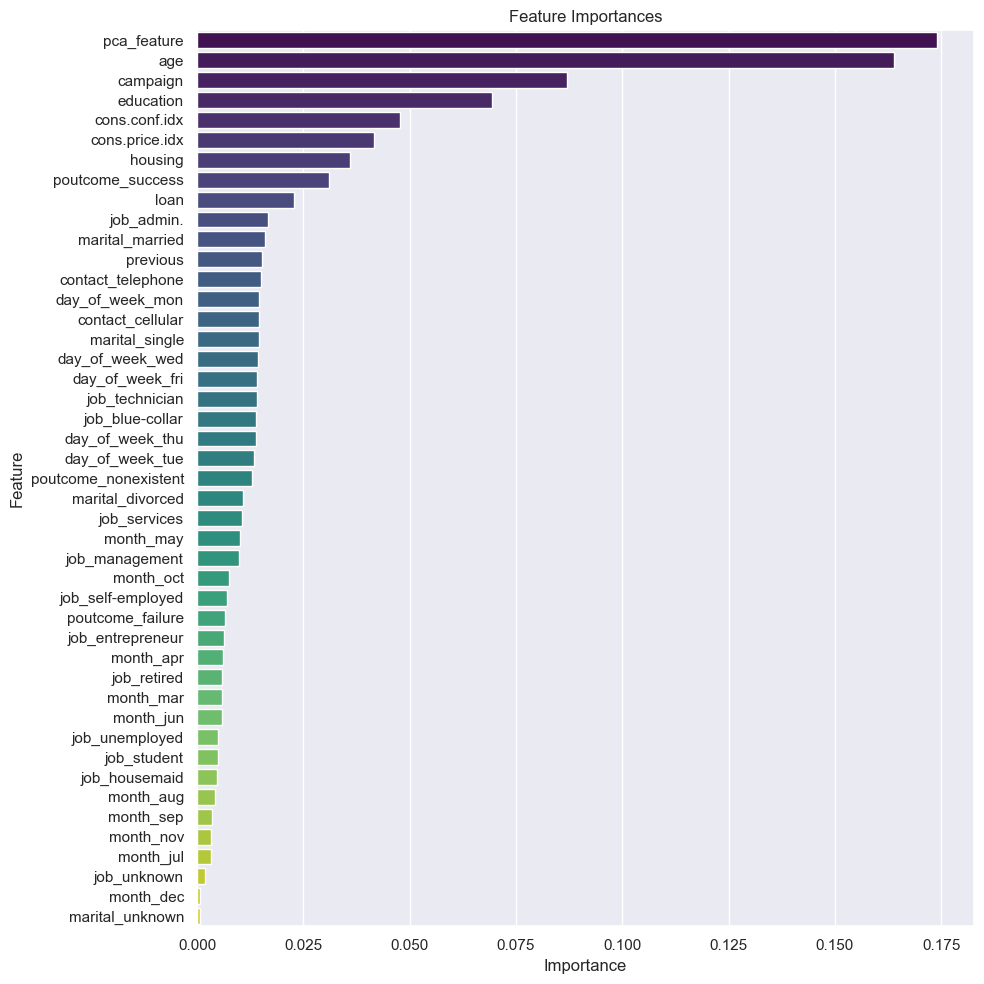


from sklearn.ensemble import RandomForestClassifier  
  
  
rf\_model\_no\_bal = RandomForestClassifier(  
 n\_estimators=100, # Number of trees  
 max\_depth=None, # Maximum depth of trees, None means no limit  
 random\_state=42, # Random seed for reproducibility  
 class\_weight='balanced' # Handle class imbalance  
)  
  
# Train the model  
rf\_model\_no\_bal.fit(X\_train, y\_train)  
  
# Make predictions  
y\_pred\_rf = rf\_model\_no\_bal.predict(X\_test)  
  
accuracy = accuracy\_score(y\_test, y\_pred\_rf)  
print("Accuracy:", accuracy)  
  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred\_rf))  
  
print("\nConfusion Matrix:")  
cm\_rf = confusion\_matrix(y\_test, y\_pred\_rf)  
print(cm\_rf)  
  
# Plot the confusion matrix with a color palette  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm\_rf, annot=True, fmt='d', cmap='magma', cbar=False, xticklabels=df["y"].unique(), yticklabels=df["y"].unique())  
  
plt.title('Confusion Matrix (Random Forest)')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.tight\_layout()  
plt.show()

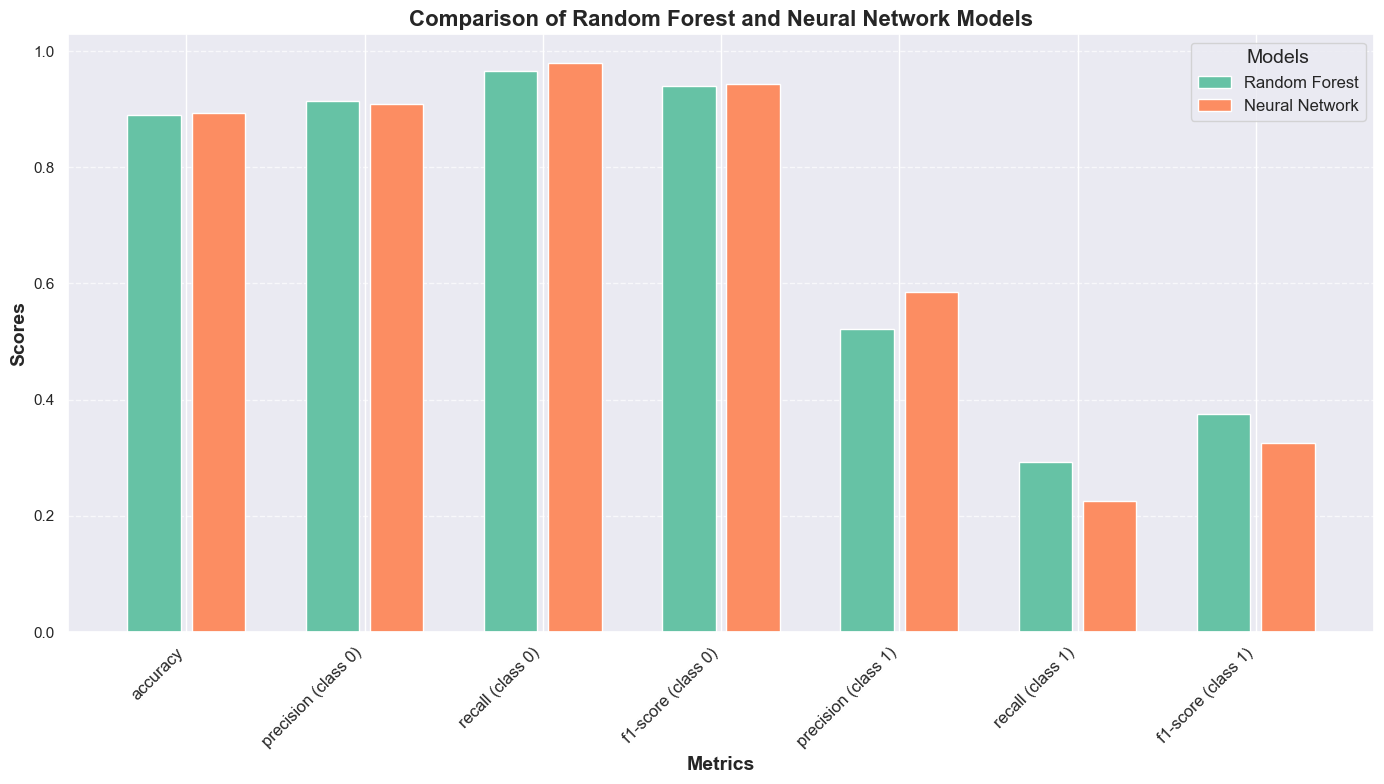
Accuracy: 0.8892935178441369  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.91 0.97 0.94 7303  
 1 0.52 0.29 0.38 935  
  
 accuracy 0.89 8238  
 macro avg 0.72 0.63 0.66 8238  
weighted avg 0.87 0.89 0.88 8238  
  
  
Confusion Matrix:  
[[7052 251]  
 [ 661 274]]



feature\_importances = pd.DataFrame({  
 'Feature': df\_encoded.drop(columns=["y"]).columns, # Replace 'y' with your target column name  
 'Importance': rf\_model\_no\_bal.feature\_importances\_  
}).sort\_values(by="Importance", ascending=False)  
  
# Plot the feature importances with a different color palette  
plt.figure(figsize=(10, 10))  
  
# Adding 'hue' to resolve the deprecation warning and control colors  
sns.barplot(x='Importance', y='Feature', data=feature\_importances, palette='viridis', hue='Feature', legend=False)  
  
plt.title('Feature Importances')  
plt.xlabel('Importance')  
plt.ylabel('Feature')  
plt.tight\_layout()  
plt.show()



import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
from sklearn.metrics import classification\_report, accuracy\_score  
  
# Assuming you have the predictions for both models: y\_pred\_rf and y\_pred\_nn  
# and their respective classification reports  
  
# Generate classification reports for both models  
report\_rf = classification\_report(y\_test, y\_pred\_rf, output\_dict=True)  
report\_nn = classification\_report(y\_test, y\_pred\_nn, output\_dict=True)  
  
# Extract metrics for both models  
metrics = ['accuracy', 'precision (class 0)', 'recall (class 0)', 'f1-score (class 0)',   
 'precision (class 1)', 'recall (class 1)', 'f1-score (class 1)']  
model\_names = ['Random Forest', 'Neural Network']  
  
# Random Forest metrics  
rf\_metrics = [  
 accuracy\_score(y\_test, y\_pred\_rf), # accuracy  
 report\_rf['0']['precision'], # precision for class 0  
 report\_rf['0']['recall'], # recall for class 0  
 report\_rf['0']['f1-score'], # f1-score for class 0  
 report\_rf['1']['precision'], # precision for class 1  
 report\_rf['1']['recall'], # recall for class 1  
 report\_rf['1']['f1-score'] # f1-score for class 1  
]  
  
# Neural Network metrics  
nn\_metrics = [  
 results\_nn[1], # accuracy from neural network evaluation  
 report\_nn['0']['precision'], # precision for class 0  
 report\_nn['0']['recall'], # recall for class 0  
 report\_nn['0']['f1-score'], # f1-score for class 0  
 report\_nn['1']['precision'], # precision for class 1  
 report\_nn['1']['recall'], # recall for class 1  
 report\_nn['1']['f1-score'] # f1-score for class 1  
]  
  
# Set up the bar chart data  
bar\_width = 0.3 # Slightly reduced bar width  
index = np.arange(len(metrics))  
  
# Set a Seaborn color palette  
sns.set\_palette("Set2") # You can experiment with different palettes, e.g., "deep", "muted", "pastel", "Set2"  
  
# Create the plot  
fig, ax = plt.subplots(figsize=(14, 8))  
  
# Plot the data with a smaller gap between the bars  
bar1 = ax.bar(index - bar\_width/2 - 0.03, rf\_metrics, bar\_width, label='Random Forest', color=sns.color\_palette()[0])  
bar2 = ax.bar(index + bar\_width/2 + 0.03, nn\_metrics, bar\_width, label='Neural Network', color=sns.color\_palette()[1])  
  
# Adding labels and title with a modern font  
ax.set\_xlabel('Metrics', fontsize=14, fontweight='bold')  
ax.set\_ylabel('Scores', fontsize=14, fontweight='bold')  
ax.set\_title('Comparison of Random Forest and Neural Network Models', fontsize=16, fontweight='bold')  
  
# Customize x-ticks for better readability  
ax.set\_xticks(index)  
ax.set\_xticklabels(metrics, rotation=45, ha='right', fontsize=12)  
  
# Add gridlines for better clarity  
ax.grid(axis='y', linestyle='--', alpha=0.7)  
  
# Adding a legend with a larger font size  
ax.legend(title='Models', fontsize=12, title\_fontsize=14)  
  
# Display the plot with a tight layout  
plt.tight\_layout()  
plt.show()



from sklearn.metrics import roc\_curve, auc  
import matplotlib.pyplot as plt  
  
# Get predicted probabilities for Random Forest (output class probabilities)  
y\_pred\_prob\_rf = rf\_model\_no\_bal.predict\_proba(X\_test)[:, 1] # Probabilities for class 1  
  
# Compute ROC curve for Random Forest  
fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test, y\_pred\_prob\_rf)  
roc\_auc\_rf = auc(fpr\_rf, tpr\_rf)  
  
# Get predicted probabilities for Neural Network (output class probabilities)  
y\_pred\_prob\_nn = model\_no\_bal.predict(X\_test)[:, 0] # Probabilities for class 1  
  
# Compute ROC curve for Neural Network  
fpr\_nn, tpr\_nn, \_ = roc\_curve(y\_test, y\_pred\_prob\_nn)  
roc\_auc\_nn = auc(fpr\_nn, tpr\_nn)  
  
# Plot ROC curves for both models  
plt.figure(figsize=(10, 6))  
plt.plot(fpr\_nn, tpr\_nn, color='red', lw=2, label=f'Neural Network (AUC = {roc\_auc\_nn:.2f})')  
plt.plot(fpr\_rf, tpr\_rf, color='blue', lw=2, label=f'Random Forest (AUC = {roc\_auc\_rf:.2f})')  
plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Random classifier line  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Curves for Neural Network and Random Forest')  
plt.legend(loc="lower right")  
plt.show()

258/258 ━━━━━━━━━━━━━━━━━━━━ 0s 469us/step

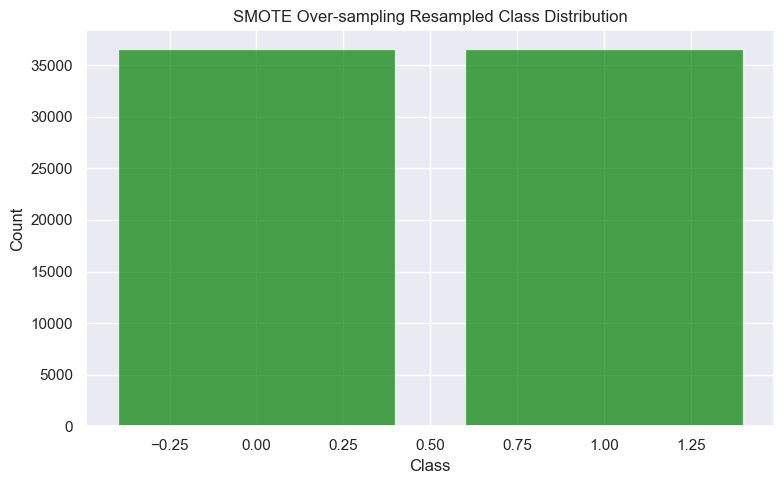


### Training using ballanced dataset (SWOT)

from imblearn.over\_sampling import SMOTE  
X = df\_encoded.drop(columns=["y"]) # Replace "target" with your actual target column name  
y = df\_encoded["y"]  
le = LabelEncoder()  
y = le.fit\_transform(y)  
  
# Step 1: Apply SMOTE (over-sampling)  
smote = SMOTE(random\_state=42)  
X\_resampled, y\_resampled = smote.fit\_resample(X, y)  
  
  
# Split the dataset into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.2, random\_state=42)

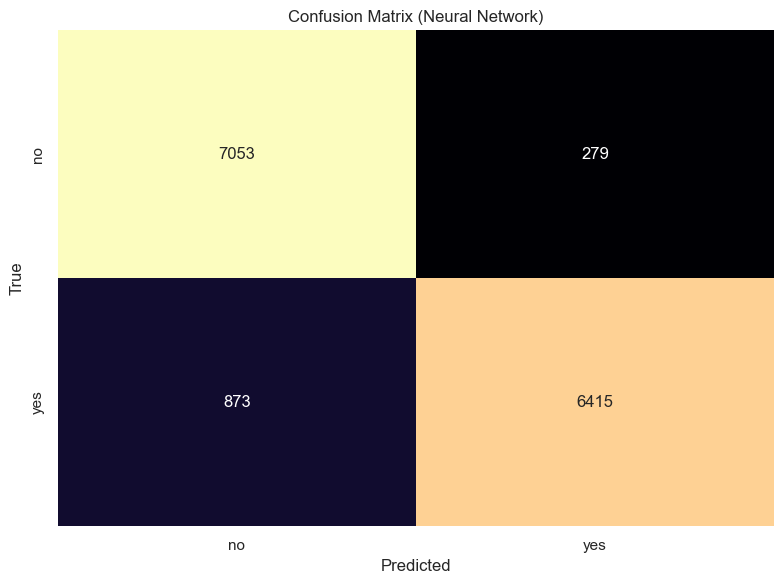
from collections import Counter  
  
# Resampled class distribution  
resampled\_distribution = Counter(y\_resampled)  
print(f"Class distribution after SMOTE over-sampling: {resampled\_distribution}")  
  
# Step 2: Plot the resampled class distribution  
fig, ax = plt.subplots(figsize=(8, 5))  
  
# Resampled distribution bar chart (after SMOTE over-sampling)  
ax.bar(resampled\_distribution.keys(), resampled\_distribution.values(), color='green', alpha=0.7)  
ax.set\_title("SMOTE Over-sampling Resampled Class Distribution")  
ax.set\_xlabel("Class")  
ax.set\_ylabel("Count")  
  
# Show the plot  
plt.tight\_layout()  
plt.show()

Class distribution after SMOTE over-sampling: Counter({0: 36548, 1: 36548})



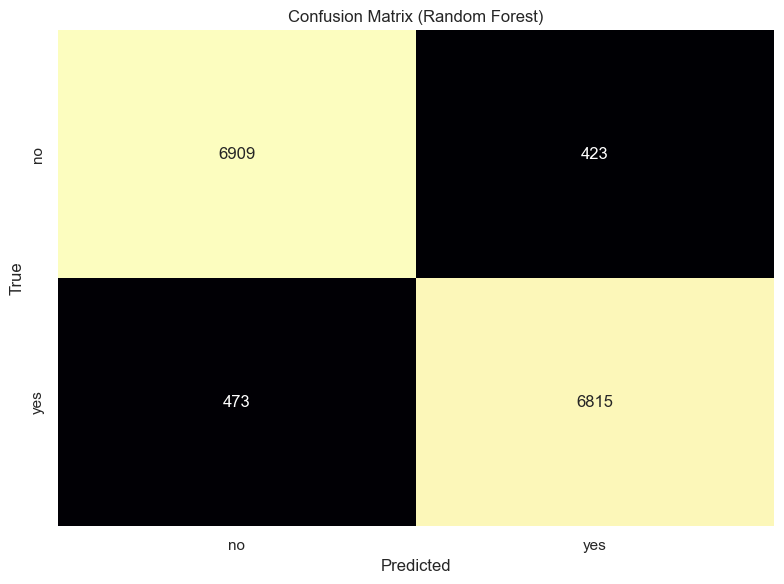
from sklearn.preprocessing import StandardScaler  
from sklearn.utils.class\_weight import compute\_class\_weight  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Input, Dense  
from tensorflow.keras.callbacks import EarlyStopping  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  
  
  
X\_train\_df = pd.DataFrame(X\_train)  
X\_test\_df = pd.DataFrame(X\_test)  
  
# Standardize the features  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train\_df) # Fit and transform on training data  
X\_test = scaler.transform(X\_test\_df) # Transform on test data  
  
  
# Define the model  
model\_bal = Sequential([  
 Input(shape=(X\_train.shape[1],)), # Define input shape explicitly  
 Dense(64, activation='relu'), # Hidden layer 1  
 Dense(32, activation='relu'), # Hidden layer 2  
 Dense(1, activation='sigmoid') # Output layer for binary classification  
])  
  
# Compile the model  
model\_bal.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
# Early stopping to prevent overfitting  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)  
  
# Train the model  
history = model\_bal.fit(  
 X\_train, y\_train,  
 epochs=20,  
 batch\_size=32,  
 validation\_data=(X\_test, y\_test),  
 callbacks=[early\_stopping]  
   
)  
  
# Evaluate the Neural Network model  
results\_nn = model\_bal.evaluate(X\_test, y\_test)  
print("\nNeural Network Test Loss:", results\_nn[0])  
print("Neural Network Test Accuracy:", results\_nn[1])  
  
# Make predictions using the Neural Network model  
y\_pred\_nn = (model\_bal.predict(X\_test) > 0.5).astype("int32")  
  
# Evaluate the Neural Network model using metrics  
print("\nClassification Report (Neural Network):")  
print(classification\_report(y\_test, y\_pred\_nn))  
  
print("\nConfusion Matrix (Neural Network):")  
print(confusion\_matrix(y\_test, y\_pred\_nn))  
  
# Confusion matrix for Neural Network predictions  
cm\_nn = confusion\_matrix(y\_test, y\_pred\_nn)  
  
# Plot the confusion matrix with a different color palette  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm\_nn, annot=True, fmt='d', cmap='magma', cbar=False, xticklabels=df["y"].unique(), yticklabels=df["y"].unique())  
  
plt.title('Confusion Matrix (Neural Network)')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.tight\_layout()  
plt.show()

Epoch 1/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8569 - loss: 0.3299 - val\_accuracy: 0.9184 - val\_loss: 0.2123  
Epoch 2/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 950us/step - accuracy: 0.9242 - loss: 0.1997 - val\_accuracy: 0.9202 - val\_loss: 0.2067  
Epoch 3/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 955us/step - accuracy: 0.9251 - loss: 0.1951 - val\_accuracy: 0.9210 - val\_loss: 0.2066  
Epoch 4/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9270 - loss: 0.1914 - val\_accuracy: 0.9184 - val\_loss: 0.2094  
Epoch 5/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9281 - loss: 0.1872 - val\_accuracy: 0.9210 - val\_loss: 0.2052  
Epoch 6/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9292 - loss: 0.1835 - val\_accuracy: 0.9220 - val\_loss: 0.2022  
Epoch 7/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9294 - loss: 0.1822 - val\_accuracy: 0.9225 - val\_loss: 0.2020  
Epoch 8/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 959us/step - accuracy: 0.9314 - loss: 0.1796 - val\_accuracy: 0.9202 - val\_loss: 0.2034  
Epoch 9/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 970us/step - accuracy: 0.9321 - loss: 0.1789 - val\_accuracy: 0.9204 - val\_loss: 0.2066  
Epoch 10/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 971us/step - accuracy: 0.9355 - loss: 0.1730 - val\_accuracy: 0.9199 - val\_loss: 0.2068  
Epoch 11/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 998us/step - accuracy: 0.9357 - loss: 0.1698 - val\_accuracy: 0.9222 - val\_loss: 0.2059  
Epoch 12/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9346 - loss: 0.1724 - val\_accuracy: 0.9212 - val\_loss: 0.2012  
Epoch 13/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9357 - loss: 0.1708 - val\_accuracy: 0.9215 - val\_loss: 0.2075  
Epoch 14/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9347 - loss: 0.1680 - val\_accuracy: 0.9221 - val\_loss: 0.2035  
Epoch 15/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9366 - loss: 0.1654 - val\_accuracy: 0.9223 - val\_loss: 0.2093  
Epoch 16/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9383 - loss: 0.1631 - val\_accuracy: 0.9224 - val\_loss: 0.2058  
Epoch 17/20  
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9388 - loss: 0.1627 - val\_accuracy: 0.9237 - val\_loss: 0.2052  
457/457 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - accuracy: 0.9196 - loss: 0.2002  
  
Neural Network Test Loss: 0.2011609822511673  
Neural Network Test Accuracy: 0.9212038516998291  
457/457 ━━━━━━━━━━━━━━━━━━━━ 0s 471us/step  
  
Classification Report (Neural Network):  
 precision recall f1-score support  
  
 0 0.89 0.96 0.92 7332  
 1 0.96 0.88 0.92 7288  
  
 accuracy 0.92 14620  
 macro avg 0.92 0.92 0.92 14620  
weighted avg 0.92 0.92 0.92 14620  
  
  
Confusion Matrix (Neural Network):  
[[7053 279]  
 [ 873 6415]]



from sklearn.ensemble import RandomForestClassifier  
  
  
rf\_model\_bal = RandomForestClassifier(  
 n\_estimators=100, # Number of trees  
 max\_depth=None, # Maximum depth of trees, None means no limit  
 random\_state=42, # Random seed for reproducibility  
 class\_weight='balanced' # Handle class imbalance  
)  
  
# Train the model  
rf\_model\_bal.fit(X\_train, y\_train)  
# Make predictions  
y\_pred\_rf = rf\_model\_bal.predict(X\_test)  
  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred\_rf))  
  
print("\nConfusion Matrix:")  
cm\_rf = confusion\_matrix(y\_test, y\_pred\_rf)  
print(cm\_rf)  
  
# Plot the confusion matrix with a color palette  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm\_rf, annot=True, fmt='d', cmap='magma', cbar=False, xticklabels=df["y"].unique(), yticklabels=df["y"].unique())  
  
plt.title('Confusion Matrix (Random Forest)')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.tight\_layout()  
plt.show()

Classification Report:  
 precision recall f1-score support  
  
 0 0.94 0.94 0.94 7332  
 1 0.94 0.94 0.94 7288  
  
 accuracy 0.94 14620  
 macro avg 0.94 0.94 0.94 14620  
weighted avg 0.94 0.94 0.94 14620  
  
  
Confusion Matrix:  
[[6909 423]  
 [ 473 6815]]



**Explanation**

* In the NN and RF calss(0) f1 score is much higer then Class(1) it wil possibly happens beacuse of the imbalanced dataset
* To reduce this we an use SWOT balance

y\_pred\_prob\_rf\_bal = rf\_model\_bal.predict\_proba(X\_test)[:, 1] # Probabilities for class 1  
  
# Compute ROC curve for Random Forest (with balancing)  
fpr\_rf\_bal, tpr\_rf\_bal, thresholds\_rf\_bal = roc\_curve(y\_test, y\_pred\_prob\_rf\_bal)  
roc\_auc\_rf\_bal = auc(fpr\_rf\_bal, tpr\_rf\_bal)  
  
# Predicted probabilities for Neural Network (with balancing)  
y\_pred\_prob\_nn\_bal = model\_bal.predict(X\_test)[:, 0] # With balancing  
fpr\_nn\_bal, tpr\_nn\_bal, thresholds\_nn\_bal = roc\_curve(y\_test, y\_pred\_prob\_nn\_bal)  
roc\_auc\_nn\_bal = auc(fpr\_nn\_bal, tpr\_nn\_bal)  
  
  
plt.figure(figsize=(12, 8))  
  
# With balancing  
plt.plot(fpr\_rf\_bal, tpr\_rf\_bal, color='blue', linestyle='-', lw=2, label=f'RF Bal (AUC = {roc\_auc\_rf\_bal:.2f})')  
plt.plot(fpr\_nn\_bal, tpr\_nn\_bal, color='red', linestyle='-', lw=2, label=f'NN Bal (AUC = {roc\_auc\_nn\_bal:.2f})')  
  
# Reference line  
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random Classifier')  
  
# Labels and legend  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Curves: With Balancing')  
plt.legend(loc="lower right")  
plt.show()

457/457 ━━━━━━━━━━━━━━━━━━━━ 0s 466us/step

