Statistical Testing - Assesment # 2

EDA - Assesment # 1 - Telecom

```
## First, let's import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import csv
import dataprep
from dataprep.eda import plot
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from scipy.stats import pointbiserialr
import scipy.stats as ss
from sklearn.preprocessing import LabelEncoder
```

Data Reading & General Overview - Variable Definition

```
# Load the dataset with semicolon as the delimiter
marketing = pd.read_csv('TeleCom_Data-1.csv', sep=';')

# Display the first few rows of the dataframe
marketing.head()
```

	age;"job";"marital";"education";"default";"housing";"loan";"contact";"month";"day_of_week";"duration";"campaign";"pdays'
0	40;"admin.";"married";"basic.6y";"no";"no";"no...
1	56;"services";"married";"high.school";"no";"no...
2	45;"services";"married";"basic.9y";"unknown";"...
3	59;"admin.";"married";"professional.course";"n...
4	41;"blue-collar";"married";"unknown";"unknown"...

```
# lets find a different way to process the data reading because pd.read_csv is not working with the semicolon separator,
cleaned_data = []

with open('TeleCom_Data-1.csv', mode='r', encoding='utf-8') as file:
    reader = csv.reader(file, delimiter=';')
    for row in reader:
        cleaned_data.append(row)

# Manually split each row by semicolon and handle the quotes
split_data = [line[0].split(';') for line in cleaned_data]

# Convert the split data to a DataFrame
marketing = pd.DataFrame(split_data[1:], columns=split_data[0])

# Clean the column names and data by removing extra quotation marks
marketing.columns = [col.strip().replace('"', '').strip() for col in marketing.columns]
marketing = marketing.applymap(lambda x: x.strip().replace('"', '').strip())


original_data = marketing.copy()

# Display the first few rows of the cleaned DataFrame
marketing.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_w
0	40	admin.	married	basic.6y	no	no	no	telephone	may	mon
1	56	services	married	high.school	no	no	yes	telephone	may	mon
2	45	services	married	basic.9y	unknown	no	no	telephone	may	mon
3	59	admin.	married	professional.course	no	no	no	telephone	may	mon
4	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon

5 rows × 21 columns

```
marketing.shape
```

```
(41180, 21)
```

Variable dictionary definitions

```
## I want to have here the dictionary of the columns and their data types
dict_market = pd.read_excel('Data_dictionary-1.xlsx')
dict_market
```

	Variable Name	Description
0	age	Age
1	job	Type of job
2	marital	Marital status
3	education	Level of education
4	default	Has credit in default
5	balance	Average yearly balance
6	housing	Has a housing loan
7	loan	Has a personal loan
8	contact	Contact communication type
9	day	Day of contact
10	month	Month of contact
11	duration	Last contact duration, in seconds (numeric). I...
12	campaign	Number of contacts performed during this campa...
13	pdays	Number of days that passed by after the client...
14	previous	Number of contacts performed before this campa...
15	poutcome	Outcome of the previous marketing campaign
16	emp.var.rate	employment variation rate - quarterly indicato...
17	cons.price.idx	consumer price index - monthly indicator (nume...
18	cons.conf.idx	consumer confidence index - monthly indicator ...

```
marketing.shape
```

```
(41180, 21)
```

Information of dataset variables

```
marketing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41180 entries, 0 to 41179
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             41180 non-null  object
 1   job             41180 non-null  object
 2   marital         41180 non-null  object
 3   education       41180 non-null  object
 4   default         41180 non-null  object
 5   housing         41180 non-null  object
 6   loan            41180 non-null  object
 7   contact         41180 non-null  object
 8   month           41180 non-null  object
 9   day_of_week     41180 non-null  object
 10  duration        41180 non-null  object
 11  campaign        41180 non-null  object
 12  pdays           41180 non-null  object
 13  previous        41180 non-null  object
 14  poutcome        41180 non-null  object
 15  emp.var.rate    41180 non-null  object
 16  cons.price.idx  41180 non-null  object
 17  cons.conf.idx   41180 non-null  object
 18  euribor3m       41180 non-null  object
 19  nr.employed     41180 non-null  object
 20  y               41180 non-null  object
dtypes: object(21)
memory usage: 6.6+ MB
```

```
duplicate_rows = marketing.duplicated().sum()

# Display number of duplicate rows
print("\nNumber of duplicate rows:", duplicate_rows)
```

```
Number of duplicate rows: 12
```

However, these duplicates can be different entries because there is nothing that separates one client from another client. They can coincide in the same features

```
## assigning correct type

numeric_columns = ['age', 'duration', 'campaign', 'pdays', 'previous',
                   'emp.var.rate', 'cons.price.idx', 'cons.conf.idx',
                   'euribor3m', 'nr.employed']

marketing[numeric_columns] = marketing[numeric_columns].apply(pd.to_numeric, errors='coerce')

# Verify the changes
marketing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41180 entries, 0 to 41179
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41180 non-null  int64
 1   job             41180 non-null  object
 2   marital         41180 non-null  object
 3   education       41180 non-null  object
 4   default         41180 non-null  object
 5   housing         41180 non-null  object
 6   loan            41180 non-null  object
 7   contact         41180 non-null  object
 8   month           41180 non-null  object
 9   day_of_week     41180 non-null  object
 10  duration        41180 non-null  int64
 11  campaign        41180 non-null  int64
 12  pdays           41180 non-null  int64
 13  previous        41180 non-null  int64
 14  poutcome        41180 non-null  object
 15  emp.var.rate    41180 non-null  float64
 16  cons.price.idx  41180 non-null  float64
 17  cons.conf.idx   41180 non-null  float64
 18  euribor3m       41180 non-null  float64
 19  nr.employed     41180 non-null  float64
 20  y               41180 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

Summary Statistics

```
## lets check the general description of the data using the describe methodology
marketing.describe()
```

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx
count	41180.000000	41180.000000	41180.000000	41180.000000	41180.000000	41180.000000	41180.000000
mean	40.021710	258.280427	2.567800	962.516707	0.172705	0.081901	93.575508
std	10.419593	259.299856	2.770225	186.809028	0.493719	1.571037	0.578762
min	17.000000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000
25%	32.000000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000
50%	38.000000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000
75%	47.000000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000
max	98.000000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000

```
## lets check the statistics for the categorical variables
marketing.describe(include='object')
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	
count	41180	41180	41180	41180	41180	41180	41180	41180	41180	41
unique	12	4	8	3	3	3	2	10	5	3
top	admin.	married	university.degree	no	yes	no	cellular	may	thu	nc
freq	10422	24921	12166	32581	21571	33943	26140	13765	8622	35

Data Cleaning & Processing

```
# Check for missing values
missing_values = marketing.isnull().sum()

# Display columns with missing values
missing_values[missing_values > 0]
```

```
Series([], dtype: int64)
```

```
# Function to display unique values for a specific column
def inspect_column(column_name):
    print(f"Unique values in column '{column_name}':")
    print(marketing[column_name].unique())
    print("\n")

# Call this function for each column you're interested in inspecting
inspect_column('age')
inspect_column('job')
inspect_column('marital')
inspect_column('education')
inspect_column('default')
inspect_column('housing')
inspect_column('loan')
```

```
Unique values in column 'age':
[40 56 45 59 41 24 25 29 57 35 54 46 39 30 55 37 49 34 52 58 32 38 44 42
 60 53 50 47 51 48 33 31 43 36 28 27 26 22 23 20 21 61 19 18 70 66 76 67
 73 88 95 77 68 75 63 80 62 65 72 82 64 71 69 78 85 79 83 81 74 17 87 91
 86 98 94 84 92 89]


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


Unique values in column 'marital':
['married' 'single' 'divorced' 'unknown']


Unique values in column 'education':
['basic.6y' 'high.school' 'basic.9y' 'professional.course' 'unknown'
 'basic.4y' 'university.degree' 'illiterate']


Unique values in column 'default':
['no' 'unknown' 'yes']


Unique values in column 'housing':
['no' 'yes' 'unknown']
```

```
inspect_column('contact')
inspect_column('month')
inspect_column('day_of_week')
inspect_column('duration')
inspect_column('campaign')
inspect_column('pdays')
```

```
Unique values in column 'contact':
['telephone' 'cellular']


Unique values in column 'month':
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']


Unique values in column 'day_of_week':
['mon' 'tue' 'wed' 'thu' 'fri']


Unique values in column 'duration':
[ 151  307  198 ... 1246 1556 1868]


Unique values in column 'campaign':
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 19 18 23 14 22 25 16 17 15 20 56
 39 35 42 28 26 27 32 21 24 29 31 30 41 37 40 33 34 43]


Unique values in column 'pdays':
[999   6   4   3   5   1   0  10   7   8   9  11   2  12  13  14  15  16
  21  17  18  22  25  26  19  27  20]
```

```
inspect_column('previous')
inspect_column('poutcome')
inspect_column('emp.var.rate')
inspect_column('cons.price.idx')
inspect_column('cons.conf.idx')
inspect_column('euribor3m')
inspect_column('nr.employed')
inspect_column('y')
```

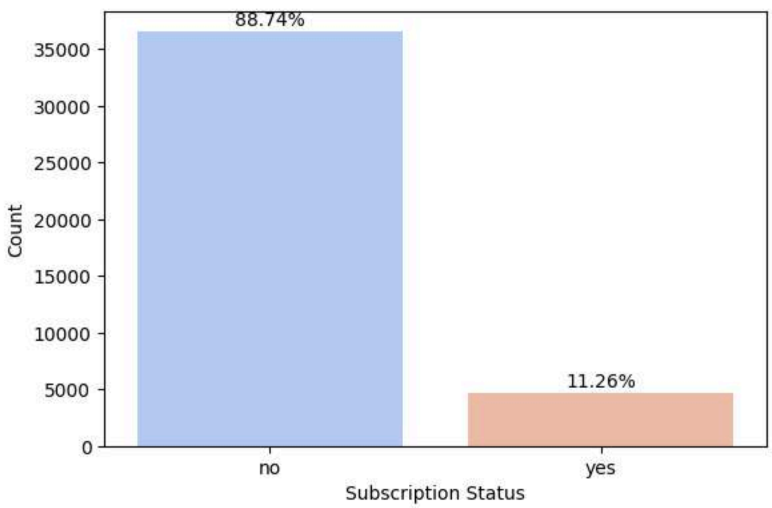
```
Unique values in column 'previous':
[0 1 2 3 4 5 6 7]


Unique values in column 'poutcome':
['nonexistent' 'failure' 'success']


Unique values in column 'emp.var.rate':
[ 1.1  1.4 -0.1 -0.2 -1.8 -2.9 -3.4 -3.  -1.7 -1.1]


Unique values in column 'cons.price.idx':
[93.994 94.465 93.918 93.444 93.798 93.2   92.756 92.843 93.075 92.893
 92.963 92.469 92.201 92.379 92.431 92.649 92.713 93.369 93.749 93.876
 94.055 94.215 94.027 94.199 94.601 94.767]


Unique values in column 'cons.conf.idx':
[-36.4 -41.8 -42.7 -36.1 -40.4 -42.  -45.9 -50.  -47.1 -46.2 -40.8 -33.6
 -31.4 -29.8 -26.9 -30.1 -33.  -34.8 -34.6 -40.  -39.8 -40.3 -38.3 -37.5
 -49.5 -50.8]


Unique values in column 'euribor3m':
[4.857 4.856 4.855 4.859 4.86  4.858 4.864 4.865 4.866 4.967 4.961 4.959
 4.958 4.96  4.962 4.955 4.947 4.956 4.966 4.963 4.957 4.968 4.97  4.965
 4.964 5.045 5.    4.936 4.921 4.918 4.912 4.827 4.794 4.76  4.733 4.7
 4.663 4.592 4.474 4.406 4.343 4.286 4.245 4.223 4.191 4.153 4.12  4.076
 4.021 3.901 3.879 3.853 3.816 3.743 3.669 3.563 3.488 3.428 3.329 3.282
```

Something went wrong while rendering the block. Please refresh the browser or **Send report**.

EDA Analysis

Target Variable

```
# Plot distribution of the target variable 'y' with percentages
plt.figure(figsize=(6, 4))
ax = sns.countplot(data=marketing, x='y', palette='coolwarm')

# Calculate percentages
total = len(marketing['y'])
for p in ax.patches:
    percentage = f'{100 * p.get_height() / total:.2f}%'
    ax.annotate(percentage, (p.get_x() + p.get_width() / 2, p.get_height()),
                ha='center', va='center', xytext=(0, 6), textcoords='offset points')  # Adjusted offset

plt.xlabel('Subscription Status')
plt.ylabel('Count')

# Show plot
plt.tight_layout()
plt.show()
```



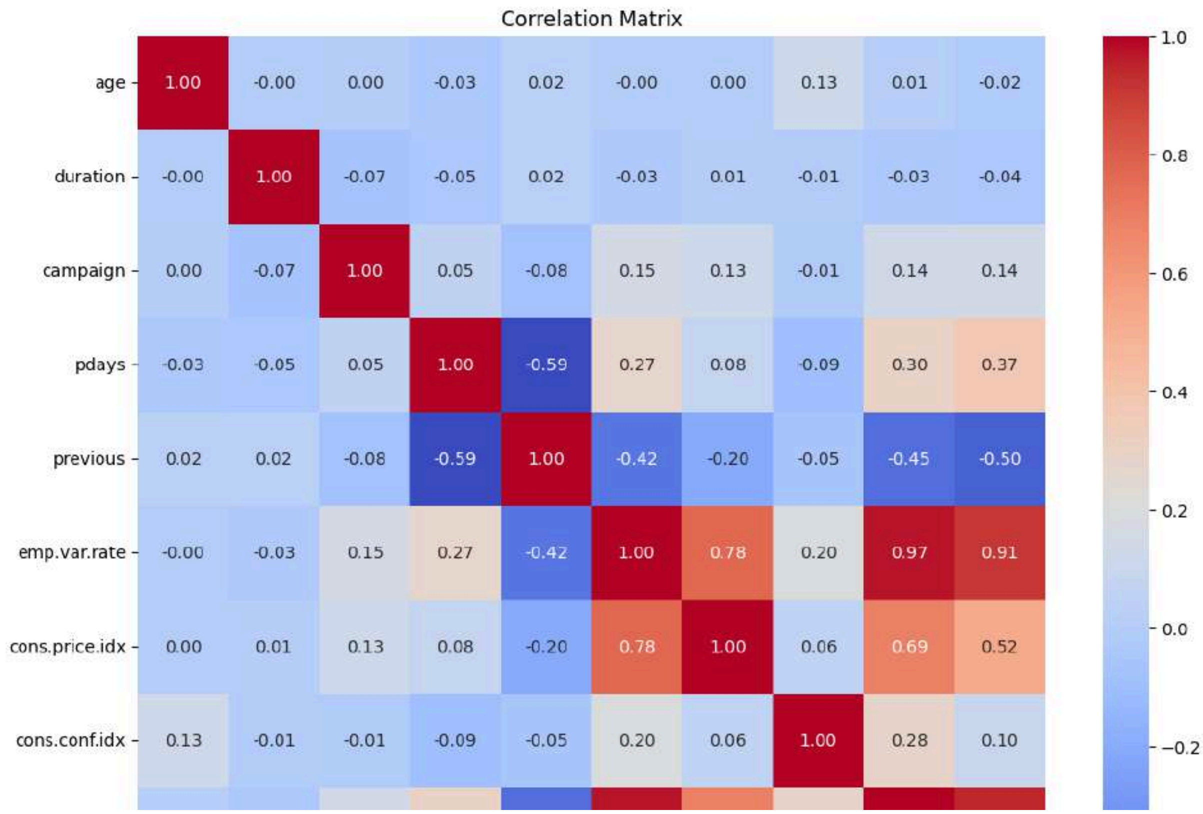
It is necessary to check variable by vvariable and analyze them individually and then what they relate with the y variable/

Heatmap for the numerical values

```
corr_matrix = marketing.corr()
```

```python
## generate a heatmap of the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
```

```
Text(0.5, 1.0, 'Correlation Matrix')
```

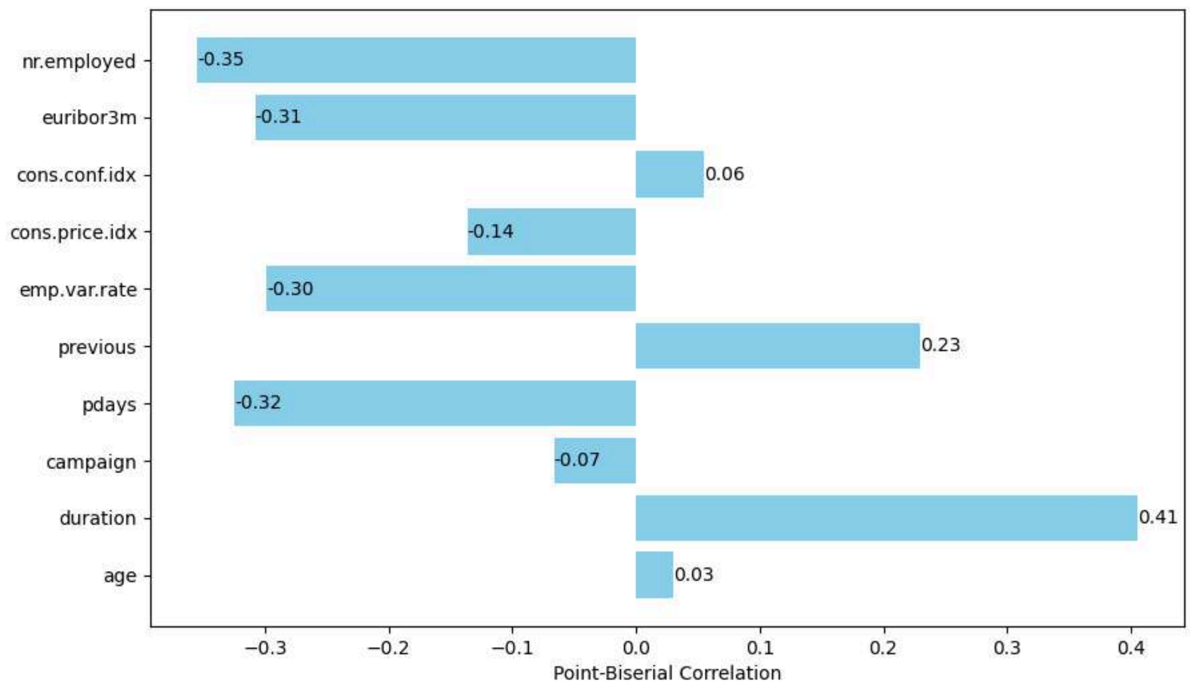


Biserial correlation (Binary vs numerical values)

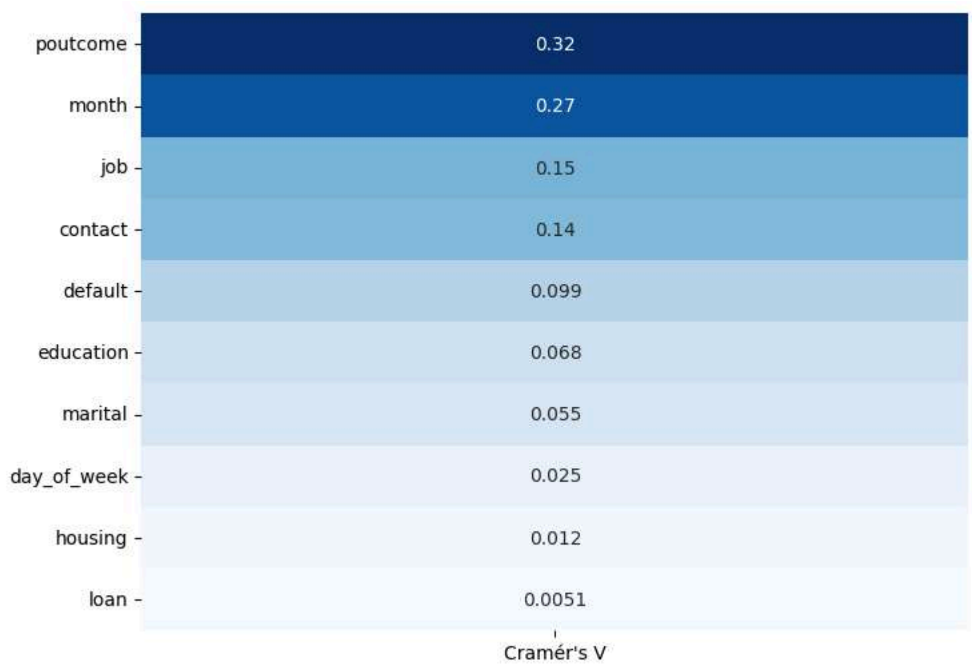
```python
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode 'y' (target variable)
marketing['y'] = label_encoder.fit_transform(marketing['y'])

# Check the encoded values
print(marketing['y'].unique())  # Should output: [0, 1]
```

```
[0 1]
```

```python
# Calculate Point-Biserial Correlation for each numerical column
correlations = {}
for col in numeric_columns:
    corr, _ = pointbiserialr(marketing[col], marketing['y'])
    correlations[col] = corr

# Create a DataFrame for easy plotting
correlation_df = pd.DataFrame(list(correlations.items()), columns=['Variable', 'Point-Biserial Correlation'])

# Plot the results with values on the bars
plt.figure(figsize=(10, 6))
ax = plt.barh(correlation_df['Variable'], correlation_df['Point-Biserial Correlation'], color='skyblue')

# Annotate the values on the bars
for index, value in enumerate(correlation_df['Point-Biserial Correlation']):
    plt.text(value, index, f'{value:.2f}', va='center', ha='left')

# Labels and title
plt.xlabel('Point-Biserial Correlation')
#plt.title('Point-Biserial Correlation Between Target Variable and Numerical Features')

# Show plot
plt.show()
```



```python
# Cramér's V calculation for categorical variables
def cramers_v(x, y):
    contingency_table = pd.crosstab(x, y)
    chi2 = ss.chi2_contingency(contingency_table)[0]
    n = contingency_table.sum().sum()
    r, k = contingency_table.shape
    return np.sqrt(chi2 / (n * (min(r - 1, k - 1))))

categorical_features = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', '

# Calculate Cramér's V for each categorical feature
cramers_v_scores = {feature: cramers_v(marketing[feature], marketing['y']) for feature in categorical_features}

# Convert results into DataFrame and plot heatmap
cramers_v_df = pd.DataFrame.from_dict(cramers_v_scores, orient='index', columns=['Cramér\'s V']).sort_values(by='Cramér\'

plt.figure(figsize=(8, 6))
sns.heatmap(cramers_v_df, annot=True, cmap='Blues', cbar=False)
#plt.title('Cramér\'s V Correlation between Categorical Features and Subscription Status')
plt.show()
```



Variable Analysis Detailed

Age

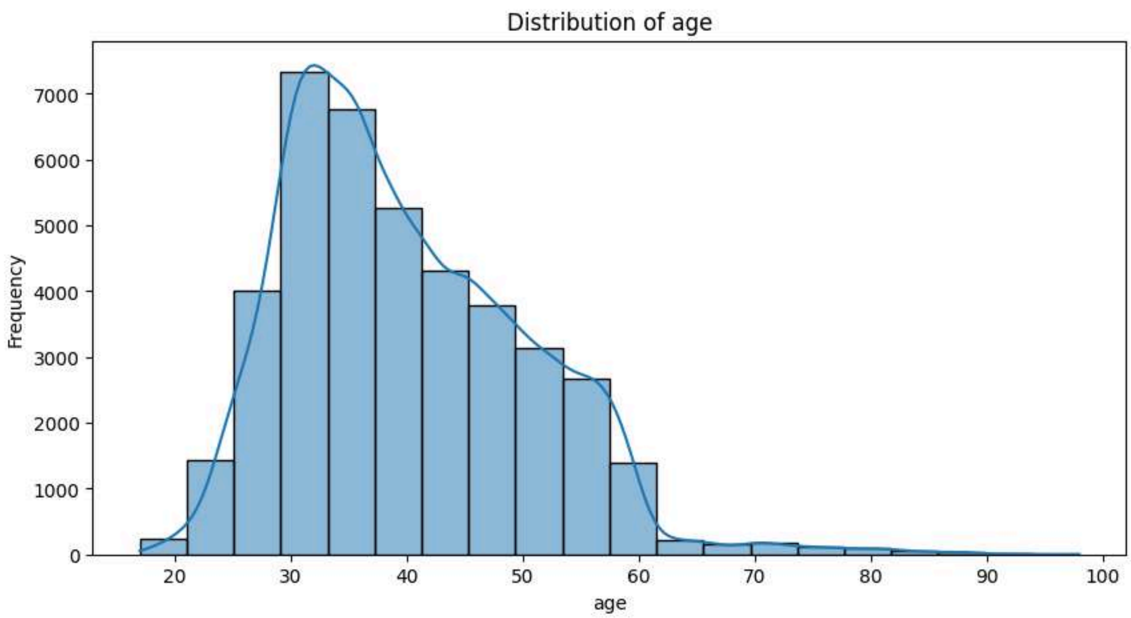
Univariate Analysis

Something went wrong while rendering the block. Please refresh the browser or Send report.

```python
def analyze_column(df, column_name, chart_type='bar'):
    """
    Analyze and visualize a specific column in the DataFrame.

    Parameters:
    df (pd.DataFrame): The DataFrame containing the data.
    column_name (str): The name of the column to analyze.
    chart_type (str): The type of chart to display ('bar' or 'pie'). Default is 'bar'.
    """
    # Summary of the column
    print(f"Summary of '{column_name}':\n{df[column_name].describe()}\n")

    # Unique values in the column
    print(f"Unique values in '{column_name}':\n{df[column_name].unique()}\n")

    # Visualization of the column distribution
    plt.figure(figsize=(10, 5))

    if df[column_name].dtype in ['int64', 'float64']:
        sns.histplot(df[column_name], kde=True, bins=20)
        plt.title(f'Distribution of {column_name}')
    else:
        if chart_type == 'bar':
            sns.countplot(y=df[column_name], order=df[column_name].value_counts().index)
            plt.title(f'Distribution of {column_name} - Bar Chart')
        elif chart_type == 'pie':
            df[column_name].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, figsize=(7, 7))
            plt.title(f'Distribution of {column_name} - Pie Chart')
            plt.ylabel('')  # Hide the y-label for pie charts

    plt.xlabel(column_name)
    plt.ylabel('Frequency')
    plt.show()
```

```python
# Example usage with the 'age' column
analyze_column(marketing, 'age')
```

```
Summary of 'age':
count    41180.000000
mean        40.021710
std         10.419593
min         17.000000
25%         32.000000
50%         38.000000
75%         47.000000
max         98.000000
Name: age, dtype: float64

Unique values in 'age':
[40 56 45 59 41 24 25 29 57 35 54 46 39 30 55 37 49 34 52 58 32 38 44 42
 60 53 50 47 51 48 33 31 43 36 28 27 26 22 23 20 21 61 19 18 70 66 76 67
 73 88 95 77 68 75 63 80 62 65 72 82 64 71 69 78 85 79 83 81 74 17 87 91
 86 98 94 84 92 89]
```



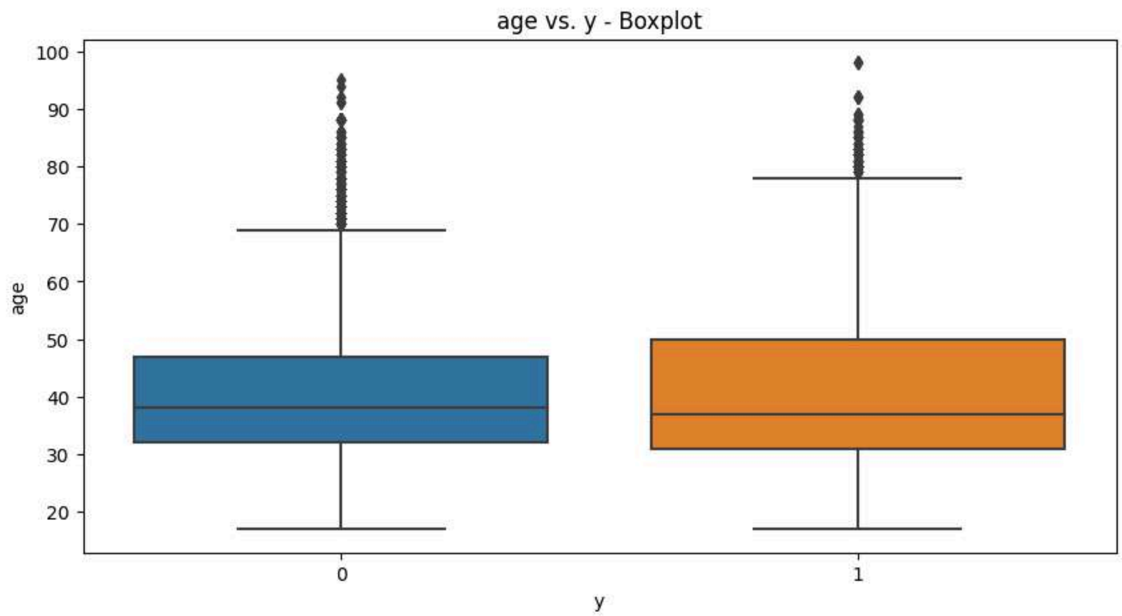
BIvariate Analysis

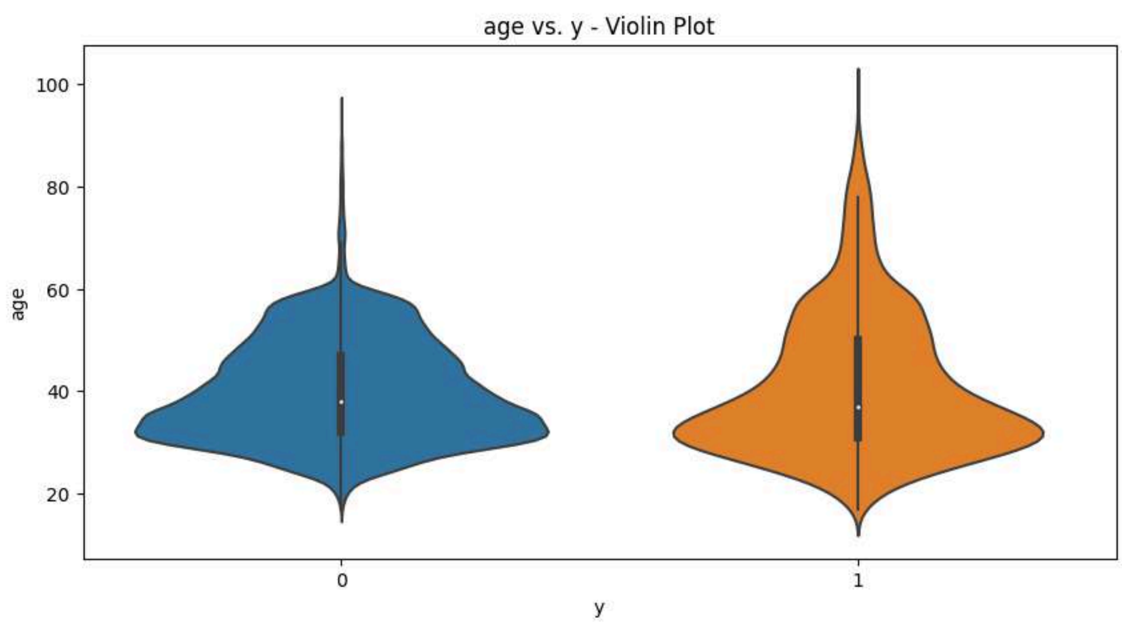
```python
def bivariate_analysis(df, independent_var, target_var='y', chart_type='stacked_bar', exclude_unknown=True):
    """
    Perform bivariate analysis between an independent variable and the target variable,
    including multiple plots to visualize the relationship.

    Parameters:
    df (pd.DataFrame): The DataFrame containing the data.
    independent_var (str): The name of the independent variable to analyze.
    target_var (str): The name of the target variable (default is 'y').
    chart_type (str): The type of chart to display for categorical variables
                      ('stacked_bar', 'bar', 'pie'). Default is 'stacked_bar'.
    exclude_unknown (bool): Whether to exclude 'unknown' categories. Default is True.
    """
    # Exclude 'unknown' values if required
    if exclude_unknown:
        df = df[df[independent_var] != 'unknown']

    print(f"Bivariate Analysis of '{independent_var}' and '{target_var}'\n")

    if df[independent_var].dtype in ['int64', 'float64']:
        # Numerical variable analysis

        # Boxplot
        plt.figure(figsize=(10, 5))
        sns.boxplot(x=target_var, y=independent_var, data=df)
        plt.title(f'{independent_var} vs. {target_var} - Boxplot')
        plt.xlabel(target_var)
        plt.ylabel(independent_var)
        plt.show()

        # Violin Plot
        plt.figure(figsize=(10, 5))
        sns.violinplot(x=target_var, y=independent_var, data=df)
        plt.title(f'{independent_var} vs. {target_var} - Violin Plot')
        plt.xlabel(target_var)
        plt.ylabel(independent_var)
        plt.show()

        # Histogram with KDE
        plt.figure(figsize=(10, 5))
        sns.histplot(df, x=independent_var, hue=target_var, kde=True, element='step')
        plt.title(f'{independent_var} vs. {target_var} - Histogram with KDE')
        plt.xlabel(independent_var)
        plt.ylabel('Density')
        plt.show()

        # Strip Plot
        plt.figure(figsize=(10, 5))
        sns.stripplot(x=target_var, y=independent_var, data=df, jitter=True)
        plt.title(f'{independent_var} vs. {target_var} - Strip Plot')
        plt.xlabel(target_var)
        plt.ylabel(independent_var)
        plt.show()

    else:
        # Categorical variable analysis

        if chart_type == 'stacked_bar':
            # Stacked Bar plot of proportions
            prop_df = (df.groupby([independent_var, target_var]).size() /
                       df.groupby([independent_var]).size()).unstack()
            ax = prop_df.plot(kind='bar', stacked=True, figsize=(10, 5))
            plt.title(f'{independent_var} vs. {target_var} - Stacked Proportion Bar Plot')
            plt.xlabel(independent_var)
            plt.ylabel('Proportion')

            # Add annotations with percentages
            for container in ax.containers:
                # Multiply the values by 100 to display percentages
                labels = [f'{v * 100:.0f}%' if v > 0 else '' for v in container.datavalues]
                ax.bar_label(container, labels=labels, label_type='center', color='white')

            plt.show()

        elif chart_type == 'bar':
            # Countplot
            plt.figure(figsize=(10, 5))
            sns.countplot(x=independent_var, hue=target_var, data=df)
            plt.title(f'{independent_var} vs. {target_var} - Count Plot')
```

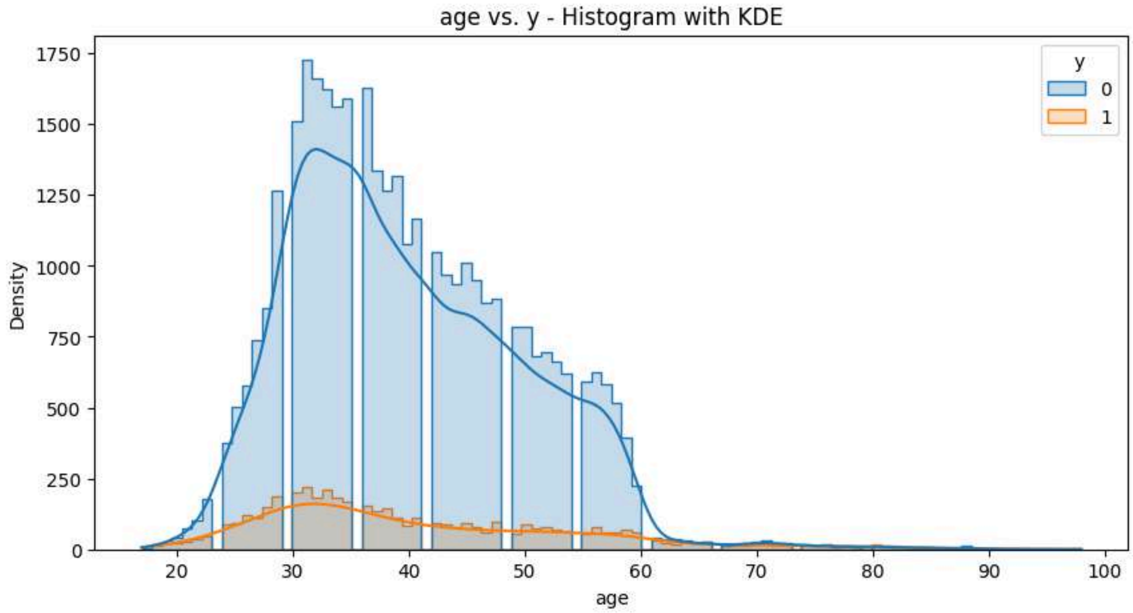
```
            plt.xlabel(independent_var)
            plt.ylabel('Count')
            plt.show()

        elif chart_type == 'pie':
            # Pie chart of proportions
            prop_df = df[target_var].groupby(df[independent_var]).value_counts(normalize=True).unstack()
            for idx, row in prop_df.iterrows():
                plt.figure(figsize=(7, 7))
                row.plot(kind='pie', autopct='%1.1f%%', startangle=90)
                plt.title(f'{independent_var}: {idx}')
                plt.ylabel('')
                plt.show()
```

```python
# Example usage with the 'age' column
bivariate_analysis(marketing, 'age')
```

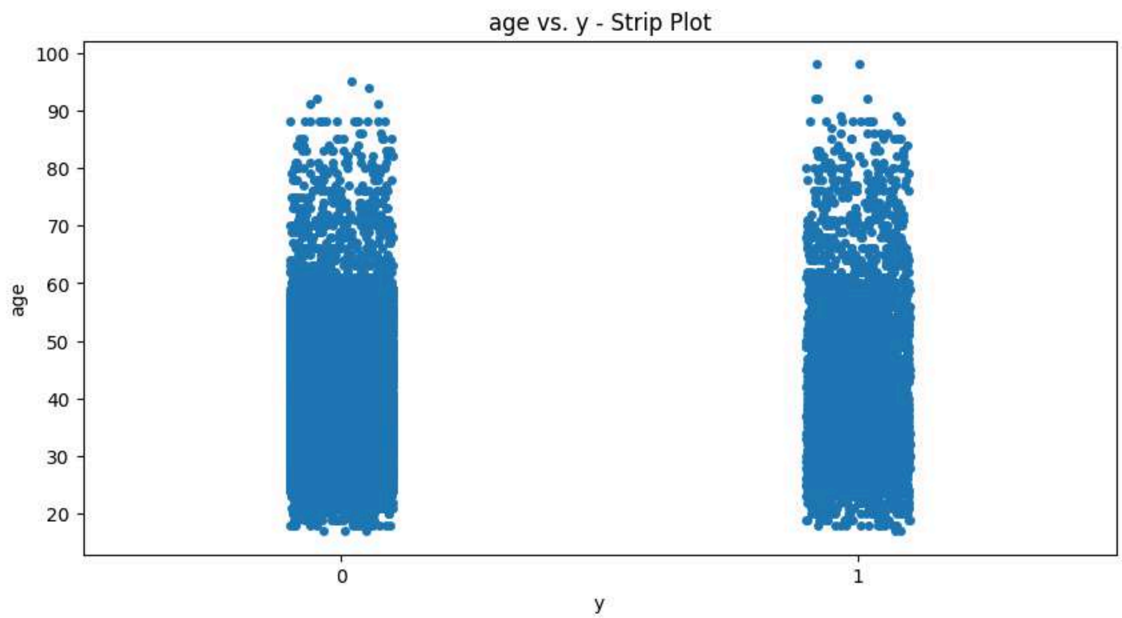
```
Bivariate Analysis of 'age' and 'y'
```



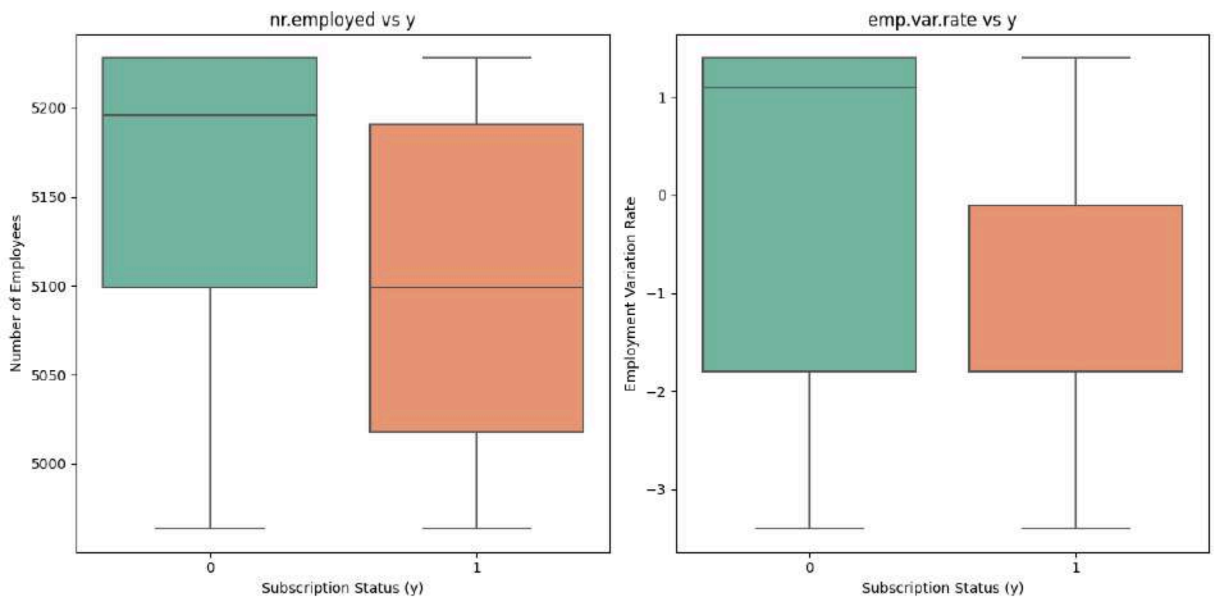





```
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
```

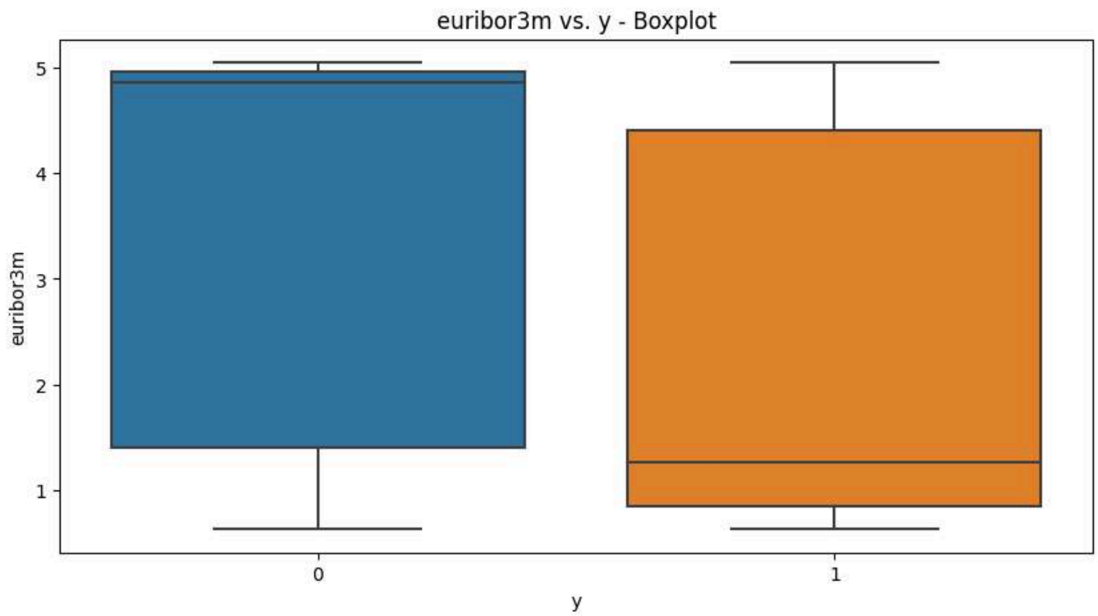


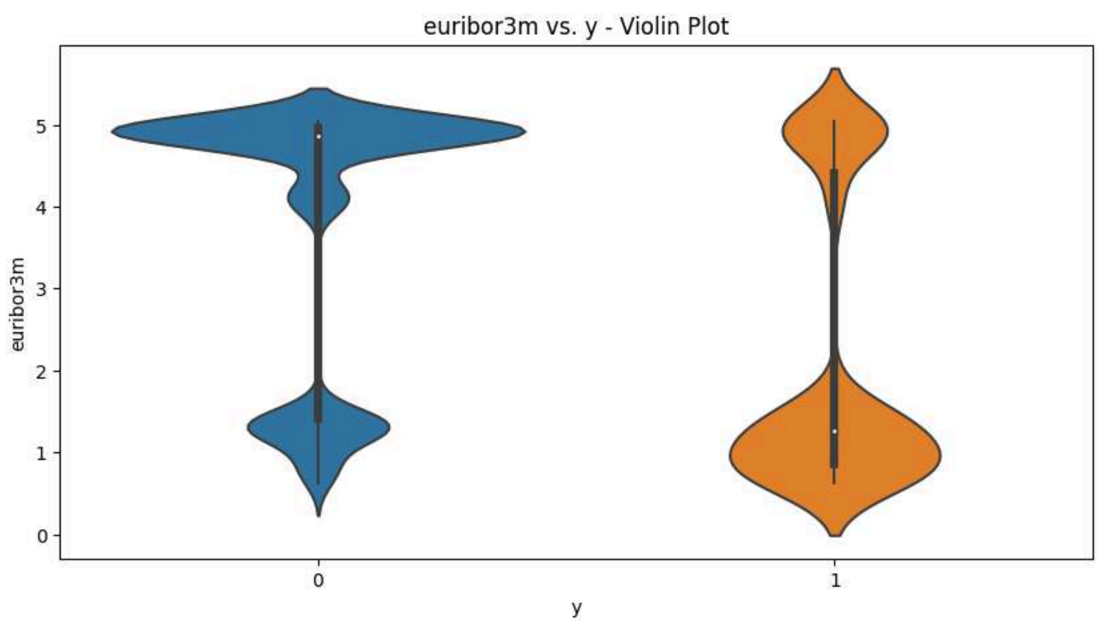
```python
plt.figure(figsize=(12, 6))

# Subplot 1: nr.employed vs y
plt.subplot(1, 2, 1)
sns.boxplot(x='y', y='nr.employed', data=marketing, palette="Set2")
plt.title('nr.employed vs y')
plt.xlabel('Subscription Status (y)')
plt.ylabel('Number of Employees')

# Subplot 2: emp.var.rate vs y
plt.subplot(1, 2, 2)
sns.boxplot(x='y', y='emp.var.rate', data=marketing, palette="Set2")
plt.title('emp.var.rate vs y')
plt.xlabel('Subscription Status (y)')
plt.ylabel('Employment Variation Rate')

# Show the combined plots
plt.tight_layout()
plt.show()
```

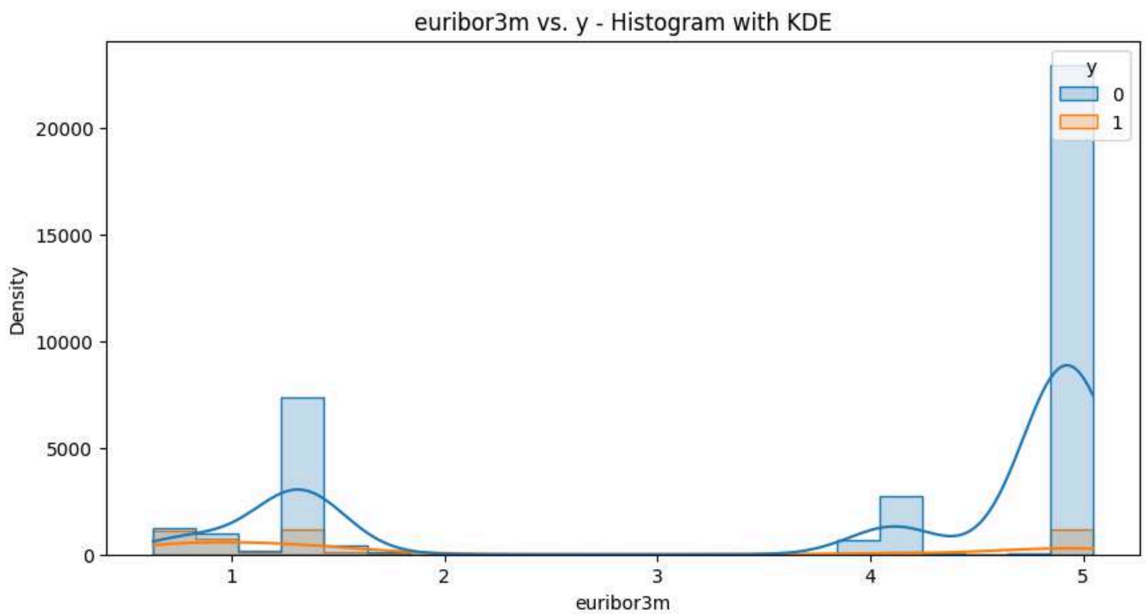


```
## lets plot euoribor3m vs
bivariate_analysis(marketing, 'euribor3m')
```

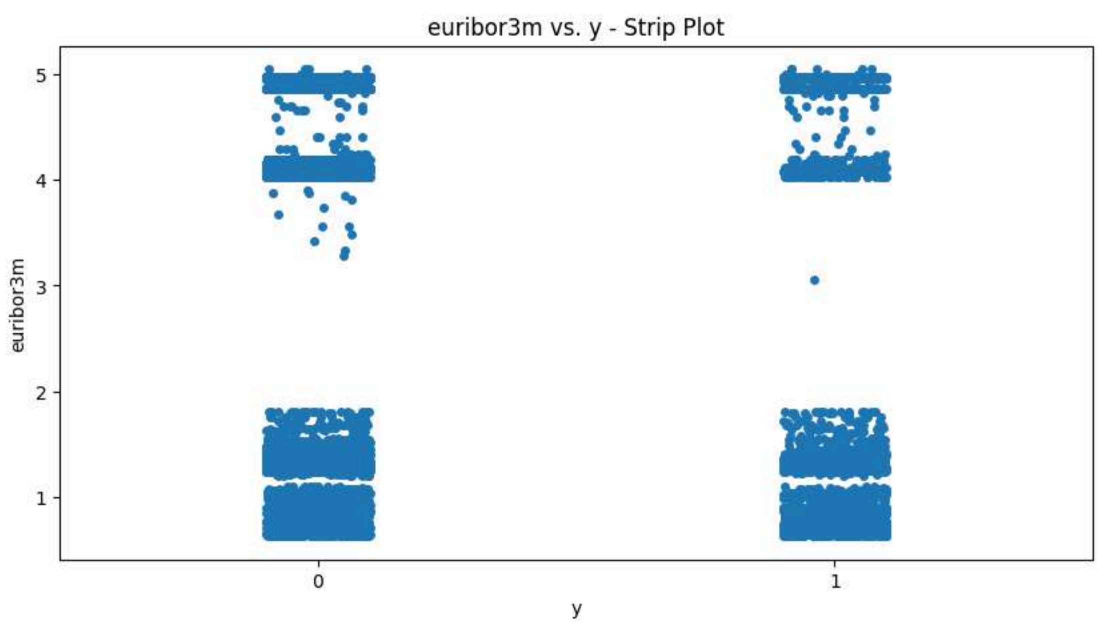
```
Bivariate Analysis of 'euribor3m' and 'y'
```





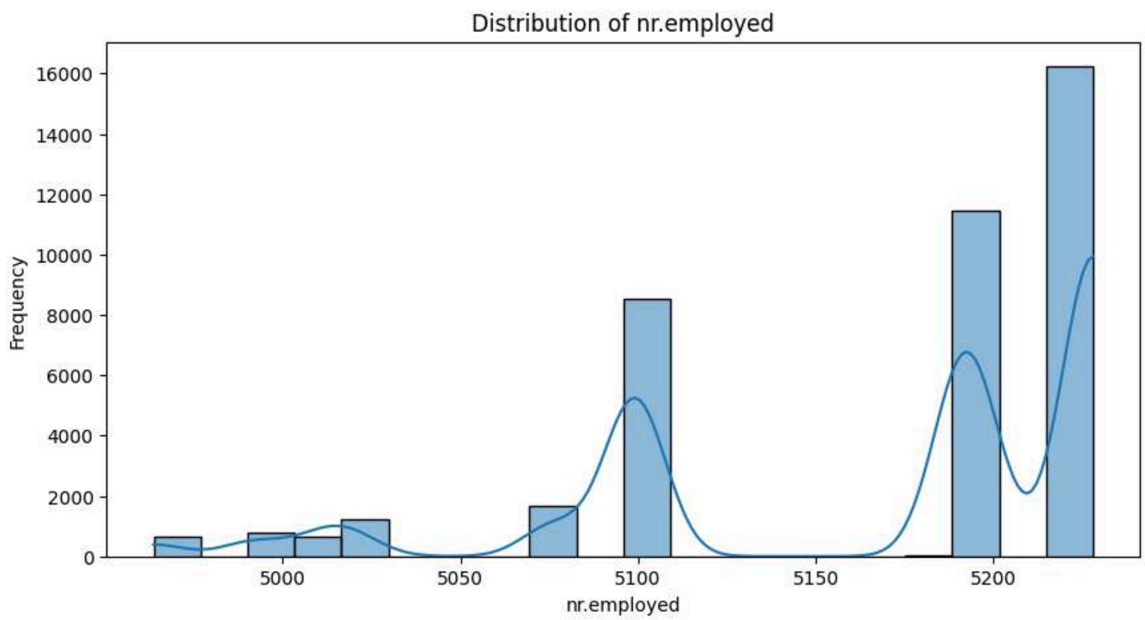



```
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
```



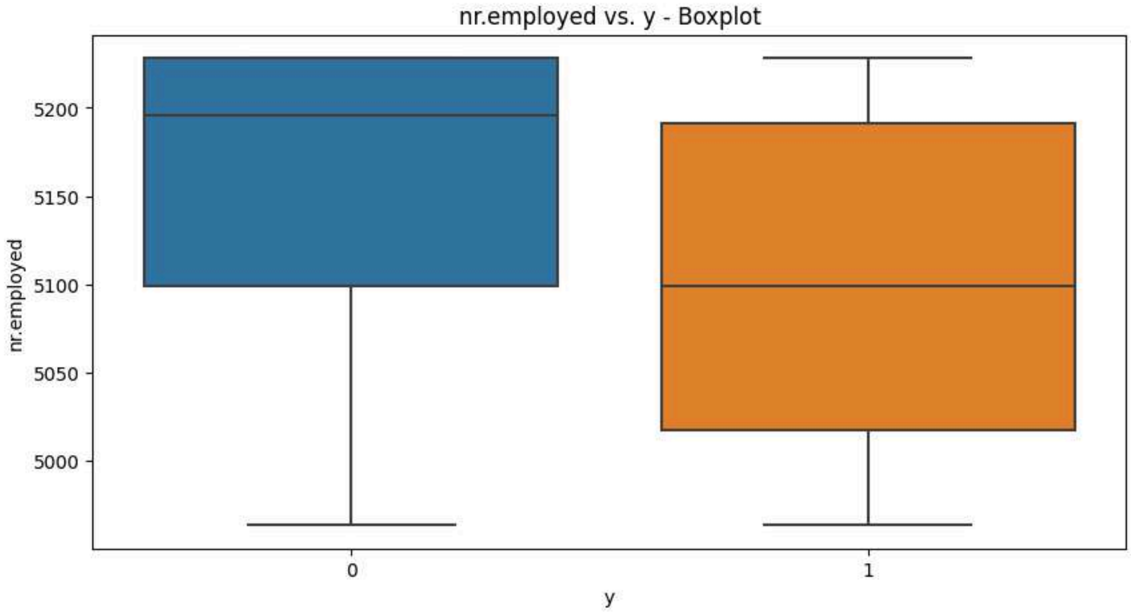
```
## lets analyze nr.employed
analyze_column(marketing, 'nr.employed')
```

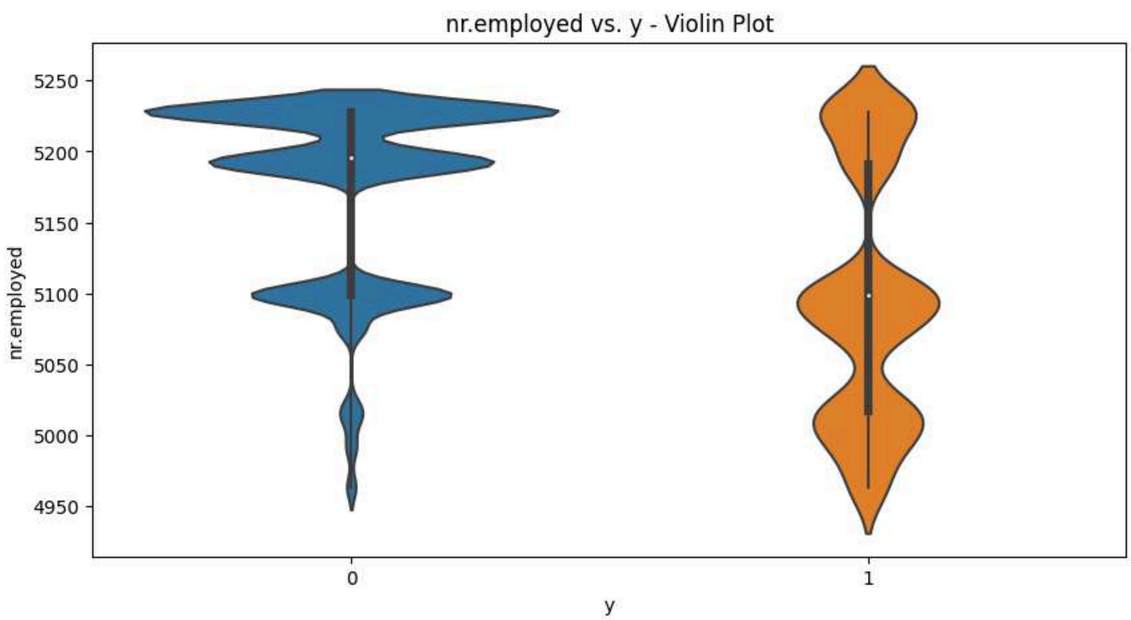
```
Summary of 'nr.employed':
count    41180.000000
mean      5167.053344
std         72.230334
min       4963.600000
25%       5099.100000
50%       5191.000000
75%       5228.100000
max       5228.100000
Name: nr.employed, dtype: float64

Unique values in 'nr.employed':
[5191.  5228.1 5195.8 5176.3 5099.1 5076.2 5017.5 5023.5 5008.7 4991.6
 4963.6]
```

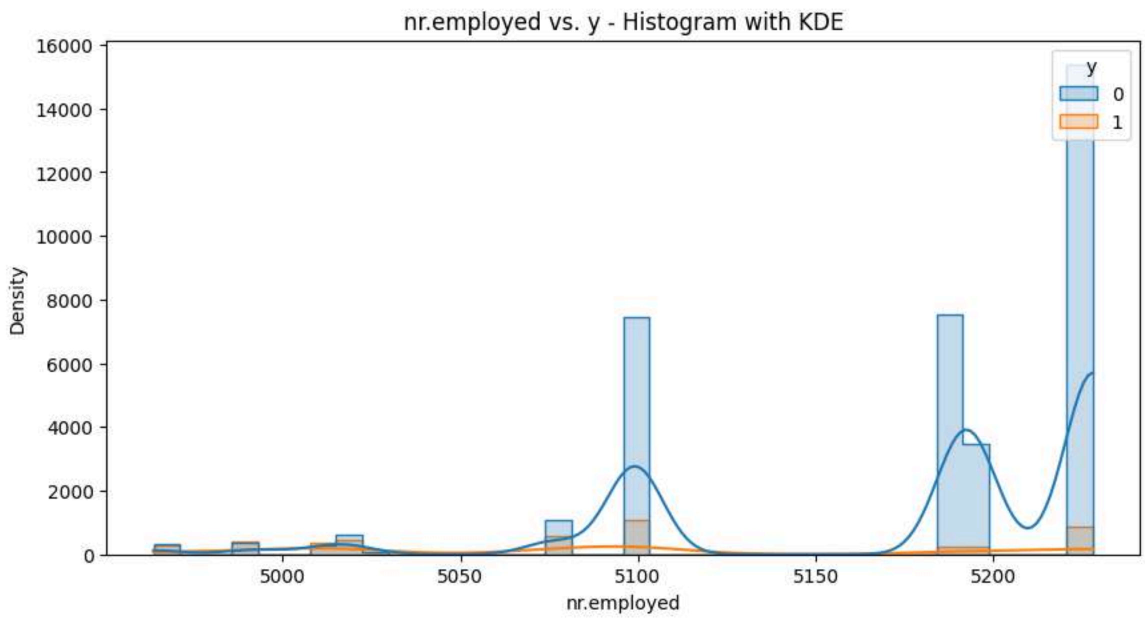


```
## bivariate for nr.employed
bivariate_analysis(marketing, 'nr.employed')
```

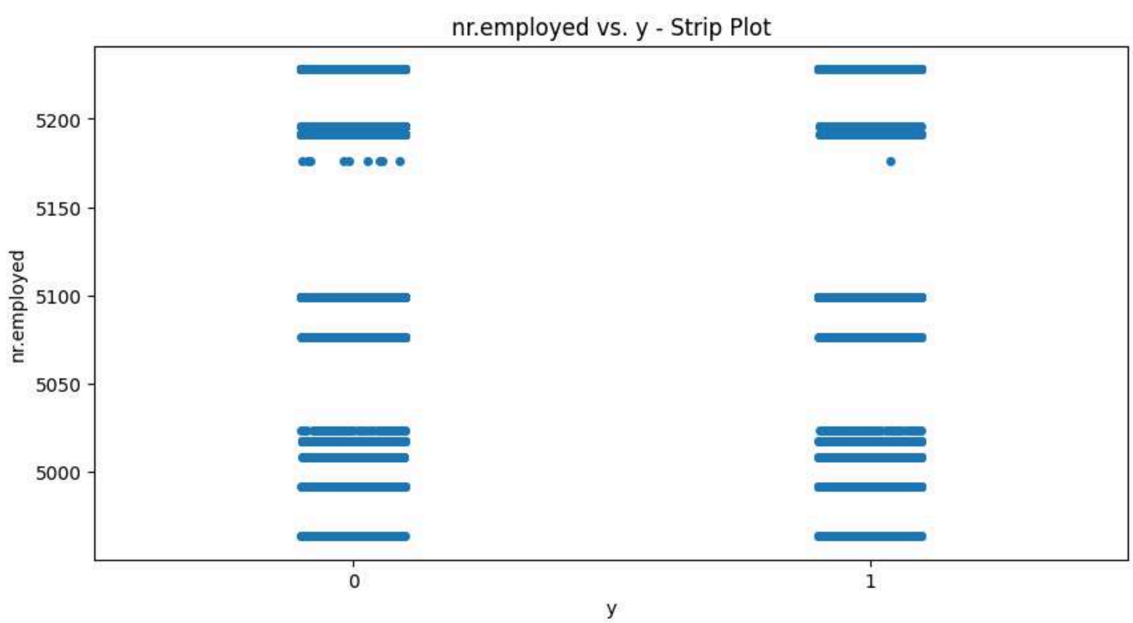
```
Bivariate Analysis of 'nr.employed' and 'y'
```








```
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
```



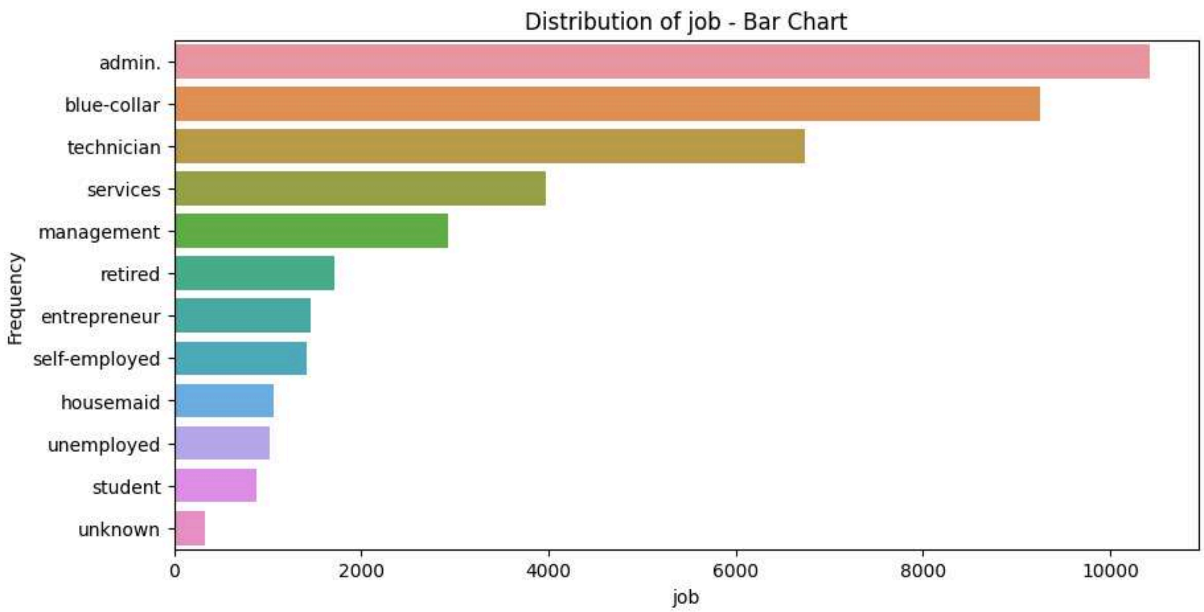
JOB

Univariate

Something went wrong while rendering the block. Please refresh the browser or **Send report**.

```
# Analyze the 'job' variable
analyze_column(marketing, 'job')
```

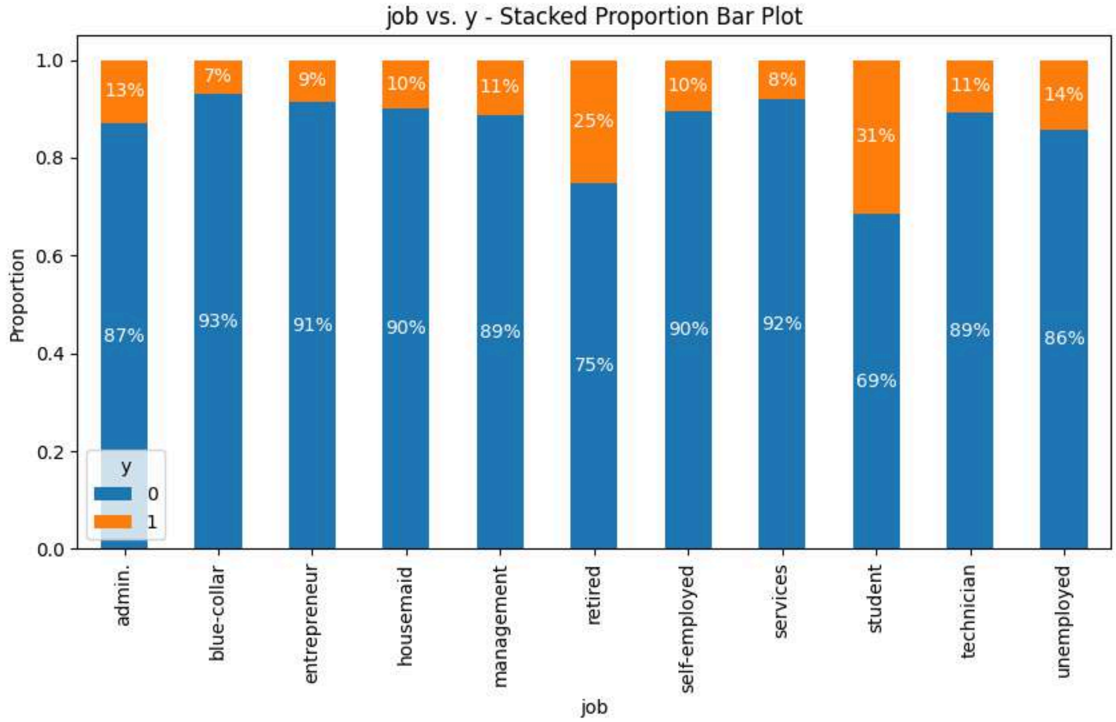
```
Summary of 'job':
count      41180
unique        12
top       admin.
freq       10422
Name: job, dtype: object

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



Bivariate

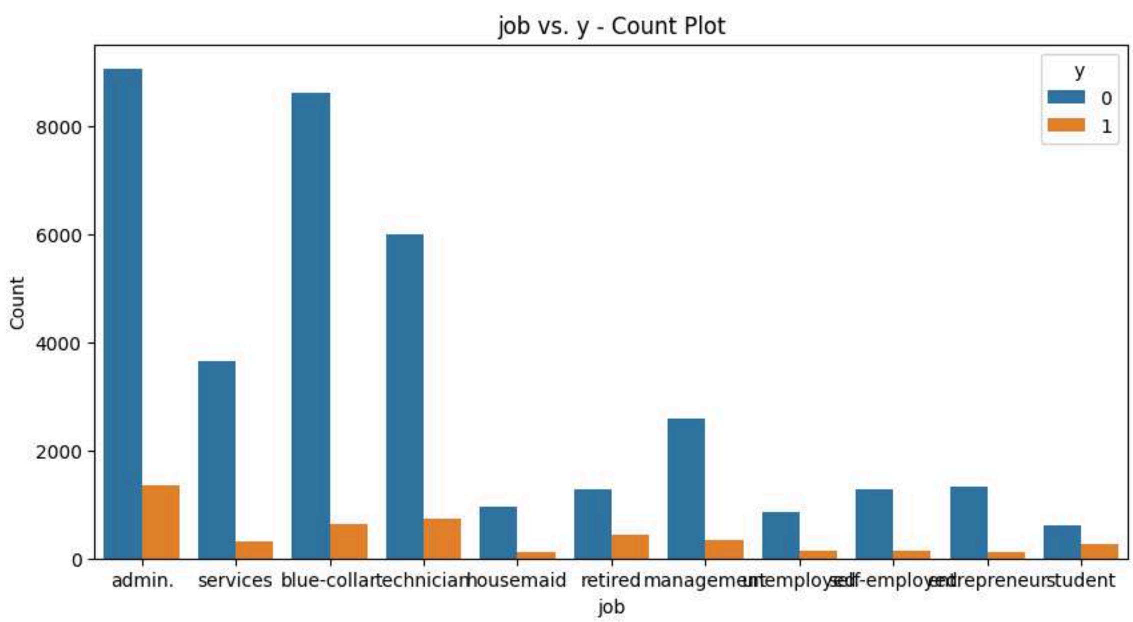
```
# Bivariate analysis of 'job' and 'y'
bivariate_analysis(marketing, 'job')
```

```
Bivariate Analysis of 'job' and 'y'
```




```
# Example usage with the 'job' column and a pie chart
bivariate_analysis(marketing, 'job', chart_type='bar')
```

```
Bivariate Analysis of 'job' and 'y'
```



```python
def generate_summary_table(df, independent_var, target_var='y'):
    """
    Generate a summary table with counts and percentages of the target variable for each category
    of the independent variable, sorted by the 'Yes_percent' in descending order.

    Parameters:
    df (pd.DataFrame): The DataFrame containing the data.
    independent_var (str): The name of the independent variable to analyze.
    target_var (str): The name of the target variable (default is 'y').

    Returns:
    pd.DataFrame: A DataFrame with counts and percentages for each category of the independent variable,
                  sorted by the 'Yes_percent'.
    """


    # Calculate counts
    count_df = df.groupby([independent_var, target_var]).size().unstack(fill_value=0)

    # Calculate percentages
    percentage_df = count_df.div(count_df.sum(axis=1), axis=0) * 100

    # Combine counts and percentages
    summary_df = count_df.join(percentage_df, lsuffix='_count', rsuffix='_percent')

    # Rename columns for clarity
    summary_df.columns = ['No_count', 'Yes_count', 'No_percent', 'Yes_percent']

    # Sort by 'Yes_percent' in descending order
    summary_df = summary_df.sort_values(by='Yes_percent', ascending=False)

    return summary_df



# Example usage with the 'job' column
job_summary = generate_summary_table(marketing, 'job')

# Display the summary table
job_summary
```

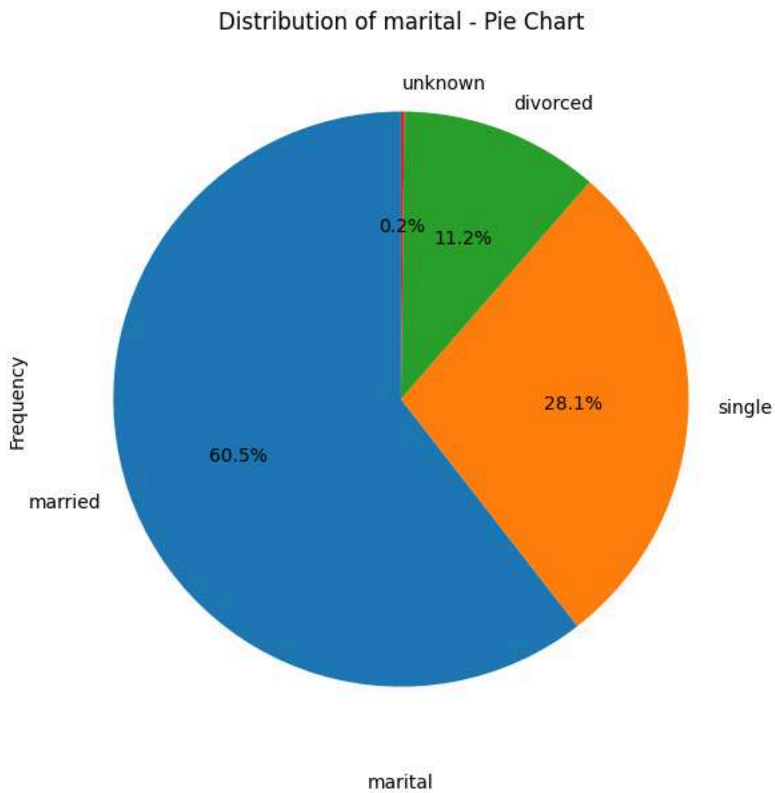
	No_count	Yes_count	No_percent	Yes_percent
job				
student	600	275	68.571429	31.428571
retired	1285	433	74.796275	25.203725
unemployed	870	144	85.798817	14.201183
admin.	9070	1352	87.027442	12.972558
management	2595	328	88.778652	11.221348
unknown	293	37	88.787879	11.212121
technician	6013	729	89.187185	10.812815
self-employed	1272	149	89.514426	10.485574
housemaid	953	106	89.990557	10.009443
entrepreneur	1332	124	91.483516	8.516484
services	3644	323	91.857827	8.142173
blue-collar	8615	638	93.104939	6.895061

Marital

Something went wrong while rendering the block. Please refresh the browser or **Send report**.

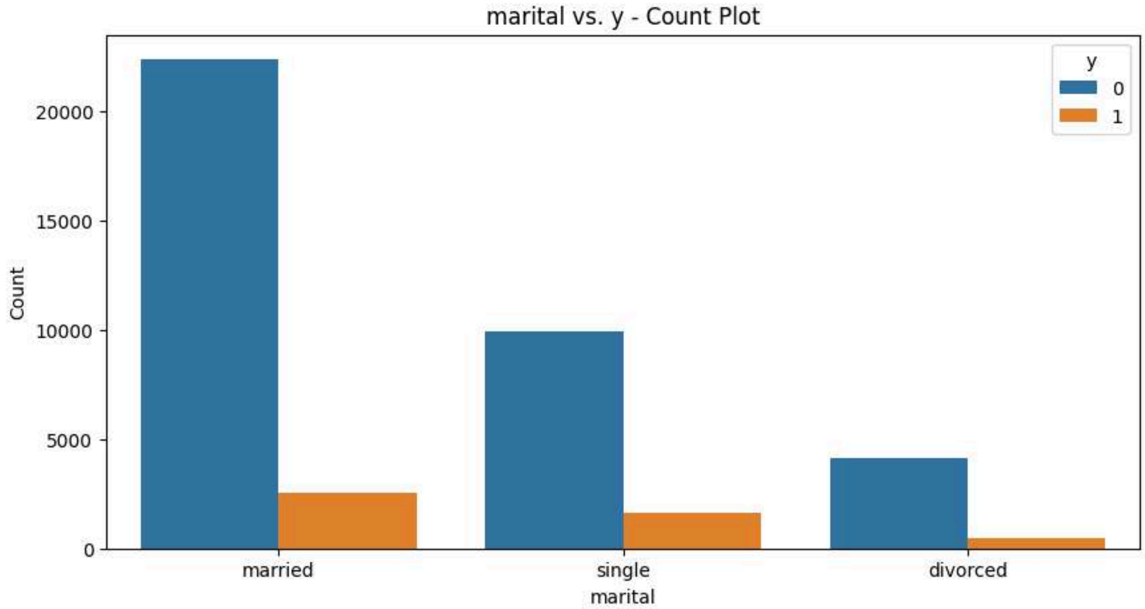
```python
# Example usage with the 'marital' column and a pie chart
analyze_column(marketing, 'marital', chart_type='pie')
```

```
Summary of 'marital':
count       41180
unique          4
top       married
freq        24921
Name: marital, dtype: object

Unique values in 'marital':
['married' 'single' 'divorced' 'unknown']
```



```
# Bivariate analysis of 'marital' and 'y'
bivariate_analysis(marketing, 'marital', chart_type='bar')
```

```
Bivariate Analysis of 'marital' and 'y'
```




```
# Generate summary table for 'marital' and 'y'
marital_summary = generate_summary_table(marketing, 'marital')

# Display the summary table
marital_summary
```

	No_count	Yes_count	No_percent	Yes_percent
marital				
unknown	68	12	85.000000	15.000000
single	9948	1620	85.995851	14.004149
divorced	4135	476	89.676860	10.323140
married	22391	2530	89.847919	10.152081

```
import matplotlib.pyplot as plt

def generate_pie_charts_per_category(df, column, target_var='y', exclude_unknown=True, colors=None):
    """
    Generate pie charts showing the relationship between each category of a given variable and the target variable.

    Parameters:
    df (pd.DataFrame): The DataFrame containing the data.
    column (str): The name of the column to generate pie charts for.
    target_var (str): The target variable, typically 'y'. Default is 'y'.
    exclude_unknown (bool): Whether to exclude 'unknown' categories from the charts. Default is True.
    colors (list of str): A list of colors to use for the pie chart. Default is None, which uses default colors.
    """
    # Filter out 'unknown' values if needed
    if exclude_unknown:
        df = df[df[column] != 'unknown']

    categories = df[column].unique()
    num_categories = len(categories)

    # Set default colors if none provided
    if colors is None:
        colors = ['#66b3ff', '#ff9999']  # Light blue for 0, light red for 1

    fig, axes = plt.subplots(1, num_categories, figsize=(5 * num_categories, 5), constrained_layout=True)
    fig.suptitle(f'{column.capitalize()} Relationship with {target_var.upper()}', fontsize=16)

    for i, category in enumerate(categories):
        category_data = df[df[column] == category][target_var]

        # Ensure that both 0 and 1 are present, even if one is missing from the data
        value_counts = category_data.value_counts()
        if 0 not in value_counts:
            value_counts[0] = 0
        if 1 not in value_counts:
            value_counts[1] = 0

        value_counts = value_counts.sort_index()  # Ensure consistent ordering

        value_counts.plot.pie(autopct='%1.1f%%', startangle=90, ax=axes[i], colors=colors)
        axes[i].set_title(f'{category.capitalize()}', fontsize=14)
        axes[i].set_ylabel('')  # Hide the y-label for pie charts

    plt.show()

# Now the colors for '0' and '1' will be consistent across all charts.
generate_pie_charts_per_category(marketing, 'marital', colors=['#66b3ff', '#ff9999'])
```

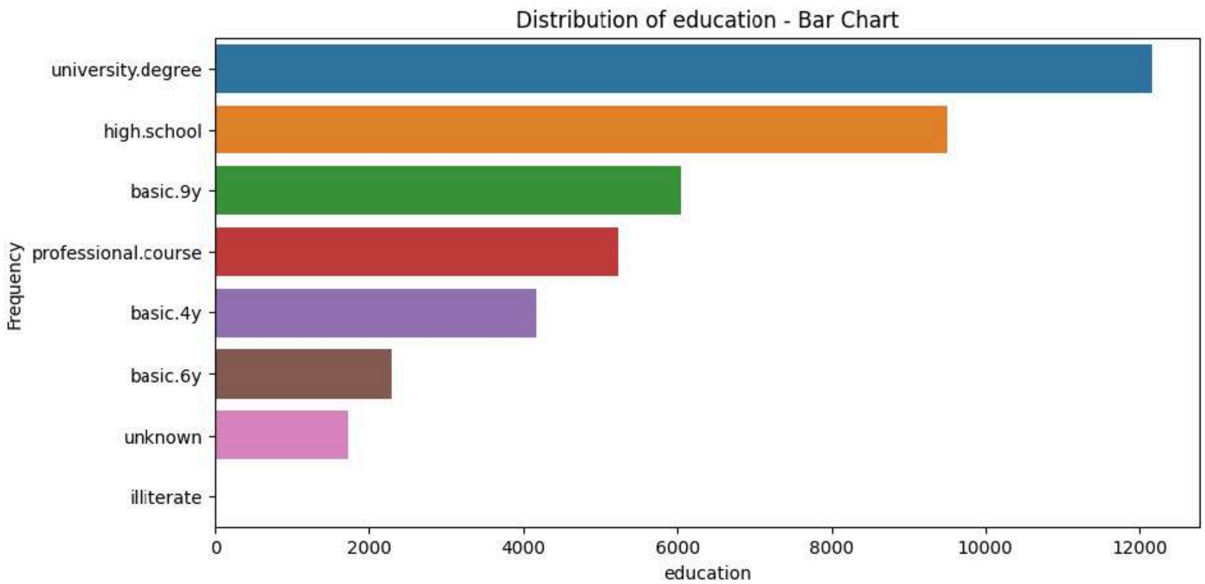
Not gigantic but single people are most common to say Yes.

Education

Something went wrong while rendering the block. Please refresh the browser or Send report.

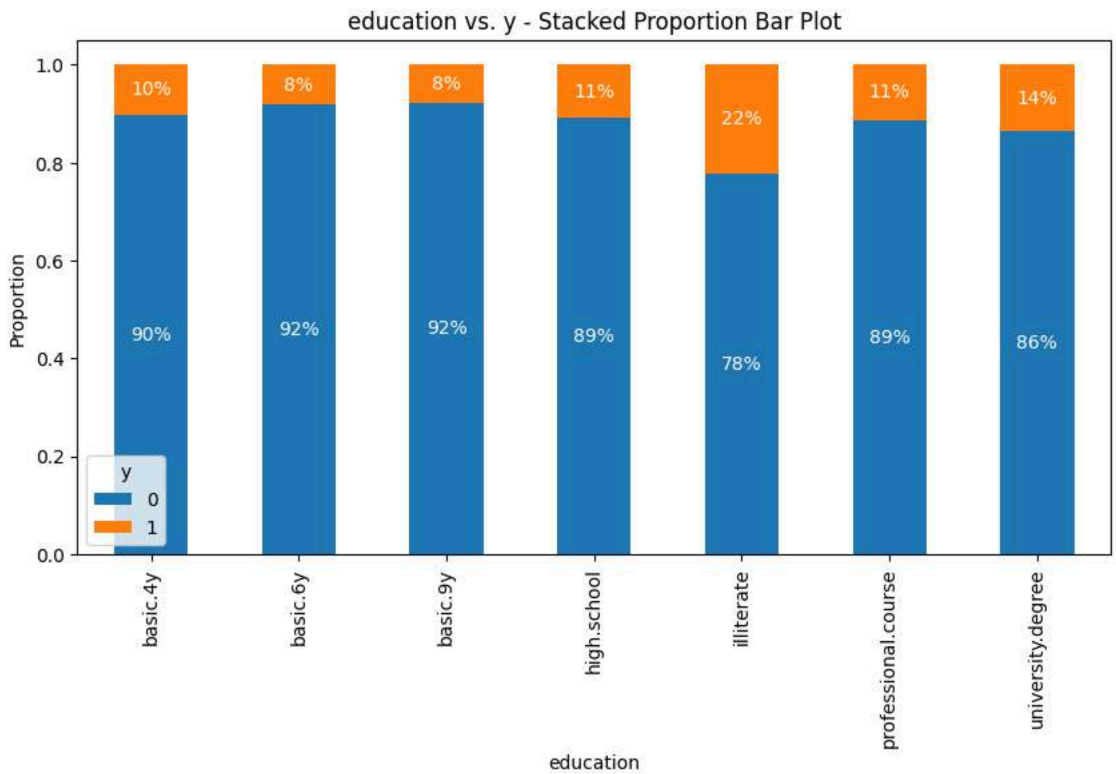
```
# Analyze the 'education' variable
analyze_column(marketing, 'education', chart_type='bar')
```

```
Summary of 'education':
count                 41180
unique                    8
top       university.degree
freq                  12166
Name: education, dtype: object

Unique values in 'education':
['basic.6y' 'high.school' 'basic.9y' 'professional.course' 'unknown'
 'basic.4y' 'university.degree' 'illiterate']
```



```
# Bivariate analysis of 'education' and 'y'
bivariate_analysis(marketing, 'education', chart_type='stacked_bar')
# Generate summary table for 'education' and 'y'
education_summary = generate_summary_table(marketing, 'education')

# Display the summary table
education_summary
```

```
Bivariate Analysis of 'education' and 'y'
```

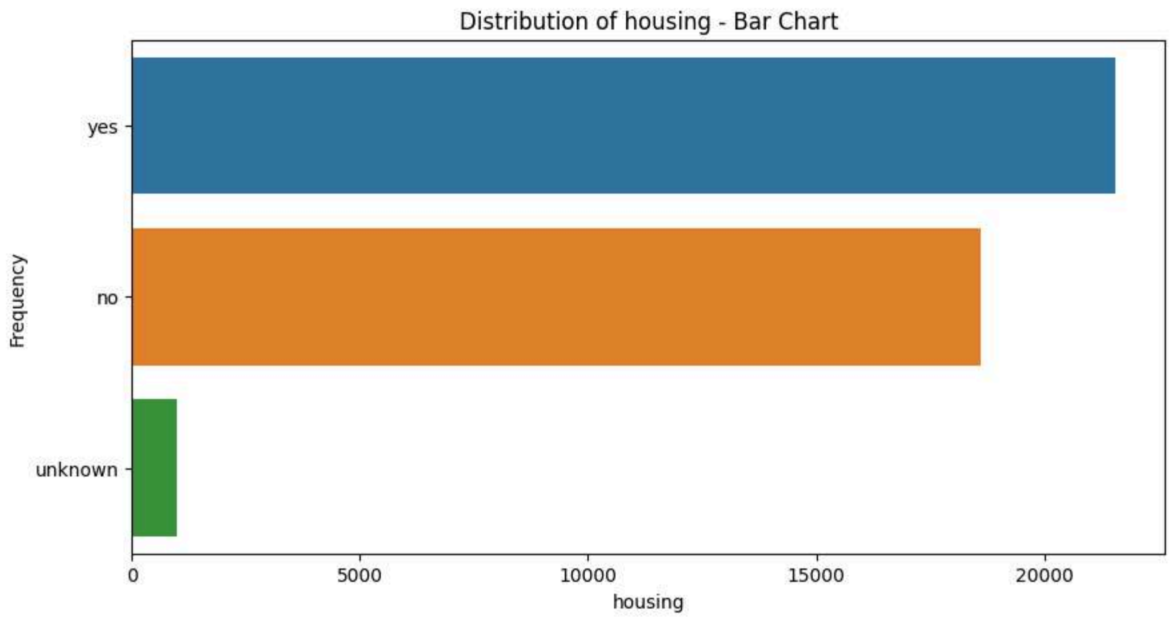


	No_count	Yes_count	No_percent	Yes_percent
education				
illiterate	14	4	77.777778	22.222222
unknown	1480	251	85.499711	14.500289
university.degree	10497	1669	86.281440	13.718560
professional.course	4647	594	88.666285	11.333715
high.school	8482	1031	89.162199	10.837801
basic.4y	3747	428	89.748503	10.251497
basic.6y	2104	188	91.797557	8.202443
basic.9y	5571	473	92.174057	7.825943

Loan

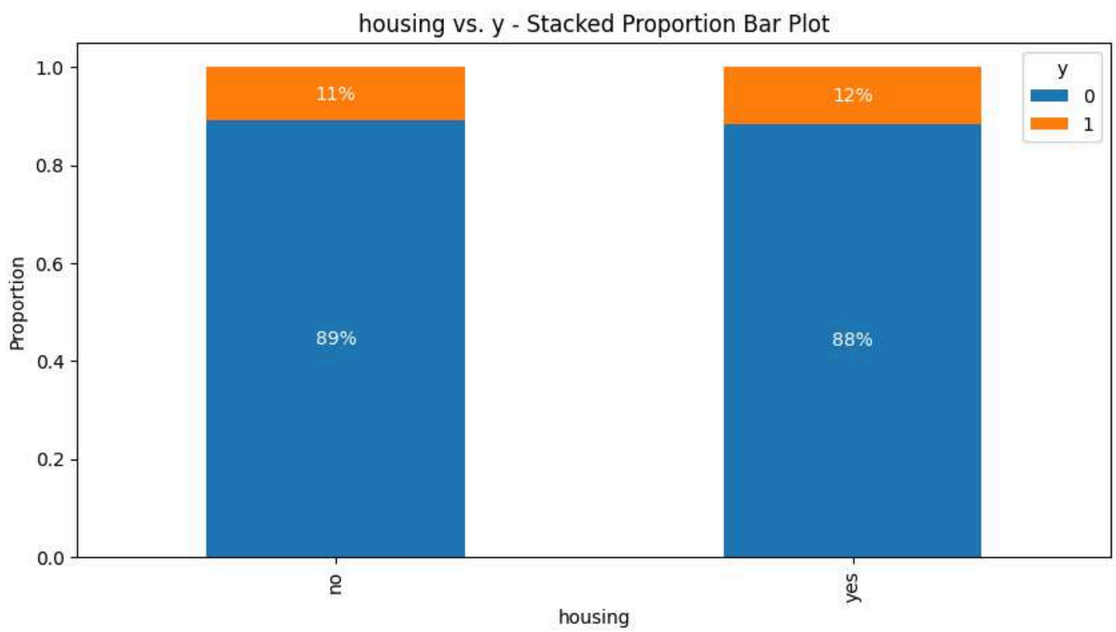
```
# Analyze the 'housing' variable
analyze_column(marketing, 'housing', chart_type='bar')
```

```
Summary of 'housing':
count     41180
unique        3
top         yes
freq      21571
Name: housing, dtype: object

Unique values in 'housing':
['no' 'yes' 'unknown']
```



```
# Bivariate analysis of 'housing' and 'y'
bivariate_analysis(marketing, 'housing', chart_type='stacked_bar')

# Generate summary table for 'housing' and 'y'
housing_summary = generate_summary_table(marketing, 'housing')

# Display the summary table
housing_summary
```

```
Bivariate Analysis of 'housing' and 'y'
```

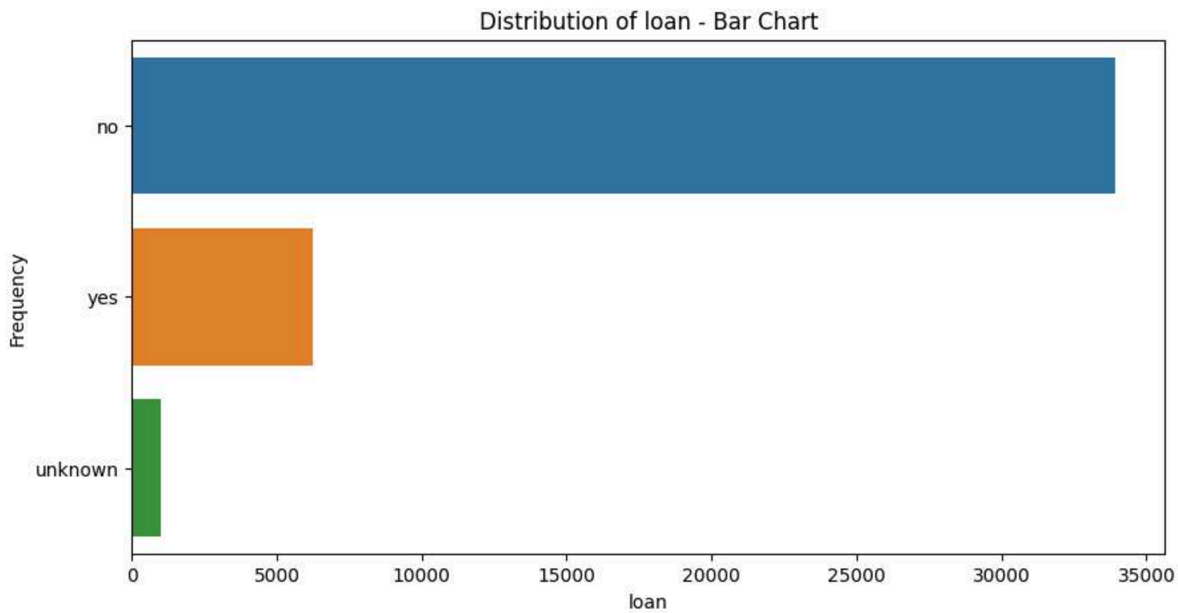


	No_count	Yes_count	No_percent	Yes_percent
housing				
yes	19065	2506	88.382551	11.617449
no	16594	2025	89.124013	10.875987
unknown	883	107	89.191919	10.808081

Loan

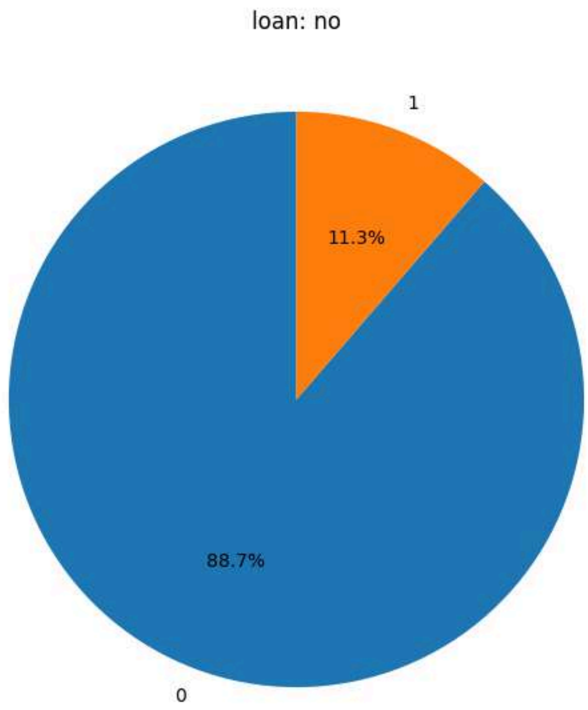
```
# Analyze the 'loan' variable
analyze_column(marketing, 'loan', chart_type='bar')
```

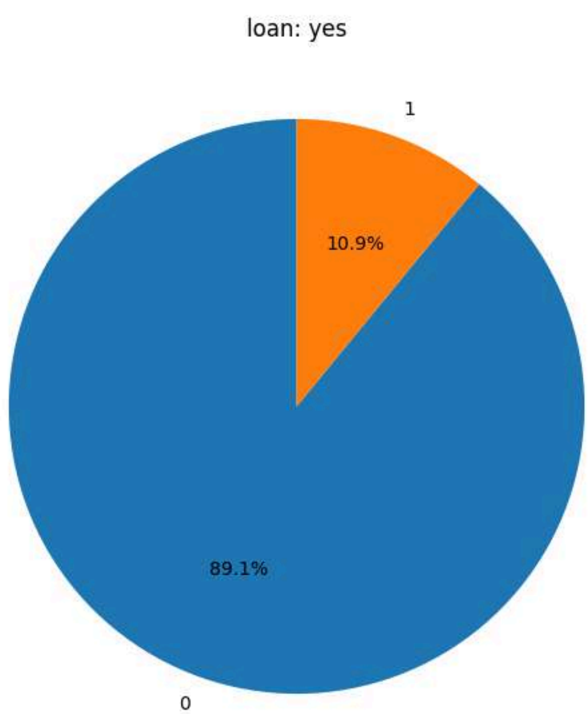
```
Summary of 'loan':
count     41180
unique        3
top          no
freq      33943
Name: loan, dtype: object

Unique values in 'loan':
['no' 'yes' 'unknown']
```



```
# Bivariate analysis of 'loan' and 'y'
bivariate_analysis(marketing, 'loan', chart_type='pie')
```

```
Bivariate Analysis of 'loan' and 'y'
```





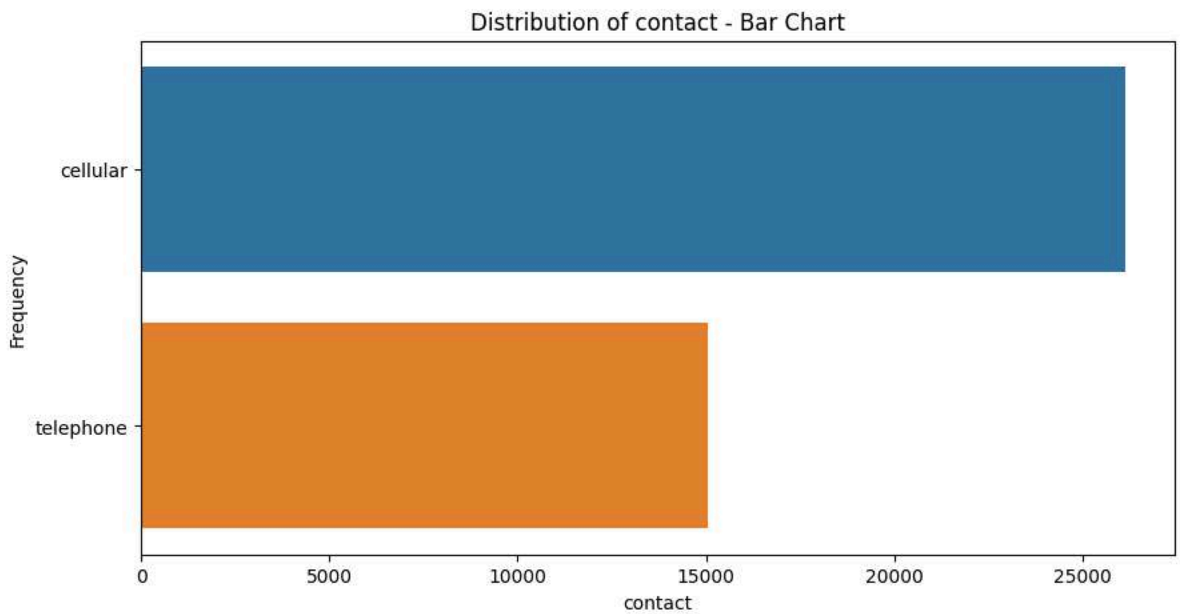
No influence in the main variable...

Contact

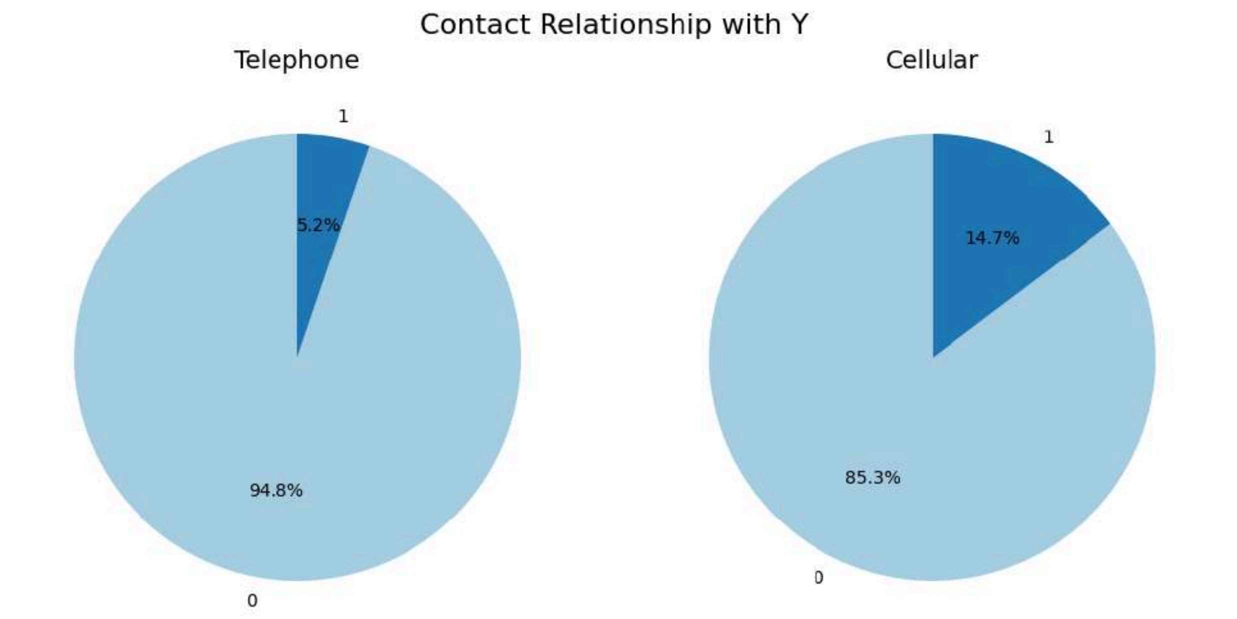
Something went wrong while rendering the block. Please refresh the browser or **Send report**.

```
# Analyze the 'contact' variable
analyze_column(marketing, 'contact', chart_type='bar')
```

```
Summary of 'contact':
count        41180
unique           2
top       cellular
freq         26140
Name: contact, dtype: object

Unique values in 'contact':
['telephone' 'cellular']
```

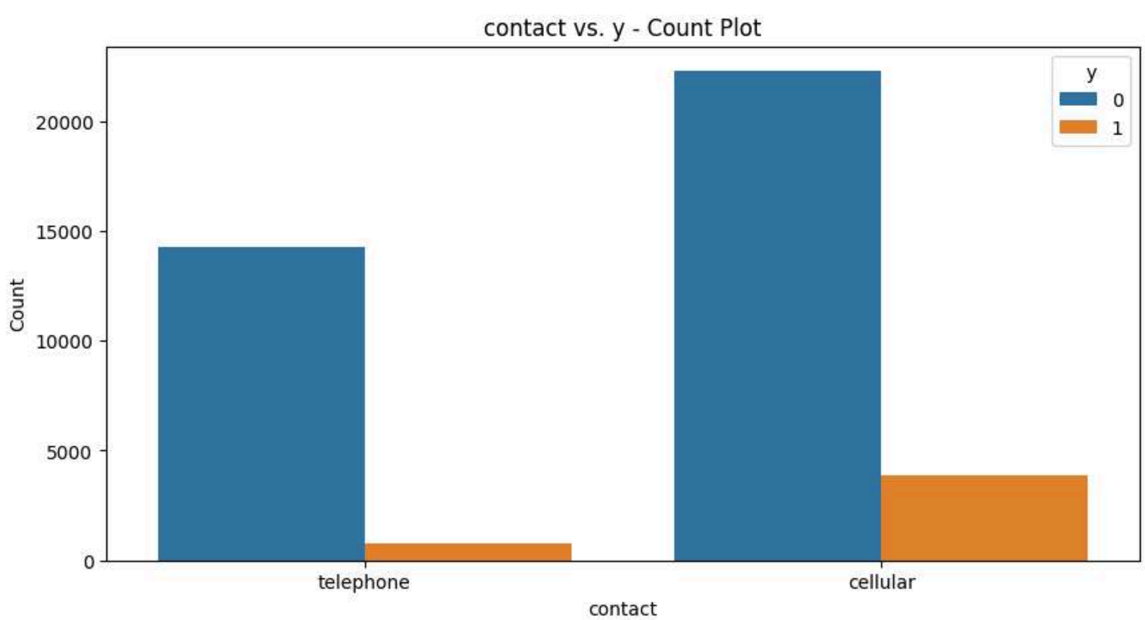


```
# Bivariate analysis of 'contact' and 'y'

generate_pie_charts_per_category(marketing, 'contact', colors=plt.cm.Paired.colors[:2])
```




```
bivariate_analysis(marketing, 'contact', chart_type='bar')
```

```
Bivariate Analysis of 'contact' and 'y'
```



```
summary_contact = generate_summary_table(marketing, 'contact')

summary_contact
```

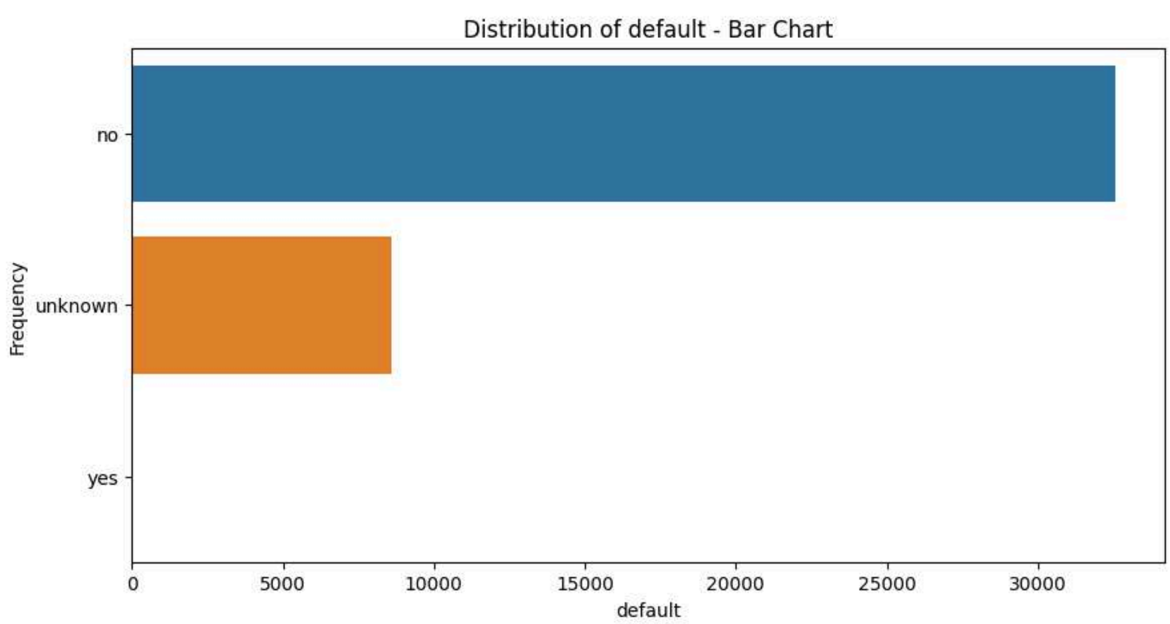
	No_count	Yes_count	No_percent	Yes_percent
contact				
cellular	22289	3851	85.267789	14.732211
telephone	14253	787	94.767287	5.232713

Better to contact them by cellular than telephone definitely.

Default

```
# Analyze the 'default' variable
analyze_column(marketing, 'default', chart_type='bar')
```

```
Summary of 'default':
count     41180
unique        3
top          no
freq      32581
Name: default, dtype: object

Unique values in 'default':
['no' 'unknown' 'yes']
```



```
bivariate_analysis(marketing, 'default', chart_type='stacked')

nonlocaldefault_summary = generate_summary_table(marketing, 'default')

# Display the summary table
print(default_summary)
```

```
Bivariate Analysis of 'default' and 'y'
```

Execution error

```
NameError: name 'default_summary' is not defined
```

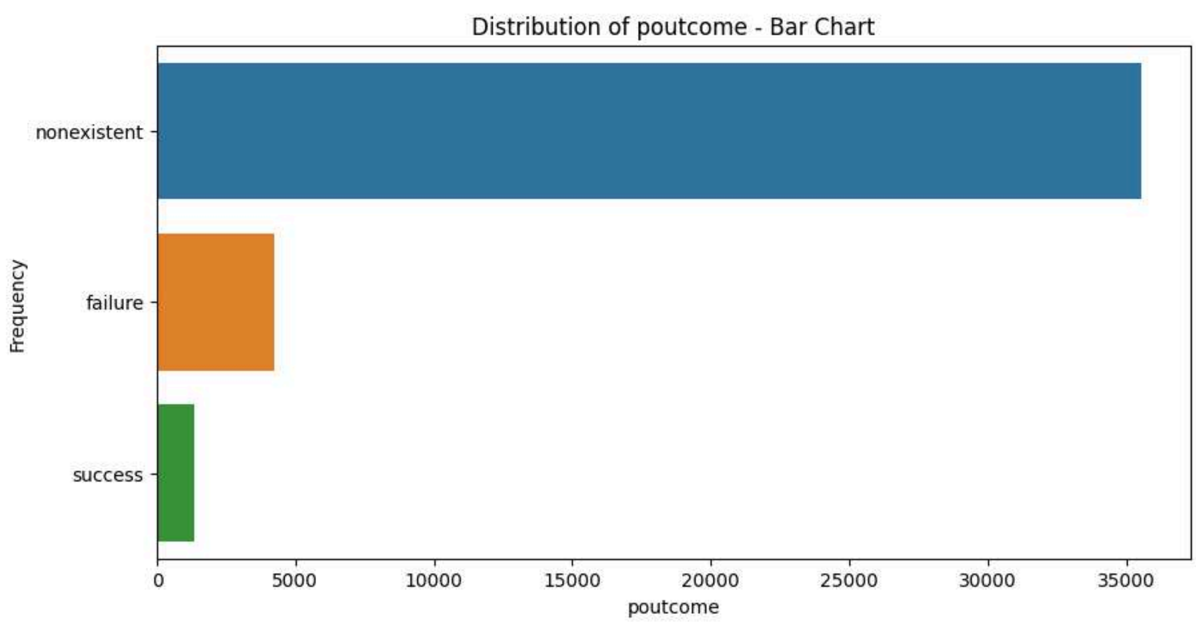
Show error details

Previous Outcome result

Something went wrong while rendering the block. Please refresh the browser or **Send report**.

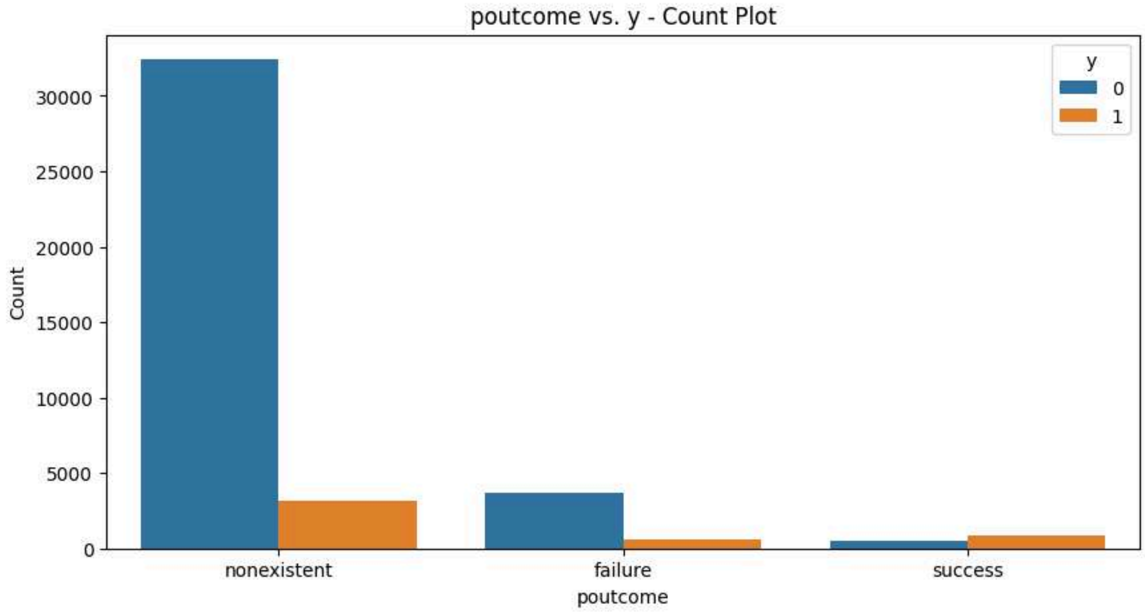
```
# Analyze the 'poutcome' variable
analyze_column(marketing, 'poutcome', chart_type='bar')
```

```
Summary of 'poutcome':
count           41180
unique              3
top       nonexistent
freq            35559
Name: poutcome, dtype: object

Unique values in 'poutcome':
['nonexistent' 'failure' 'success']
```

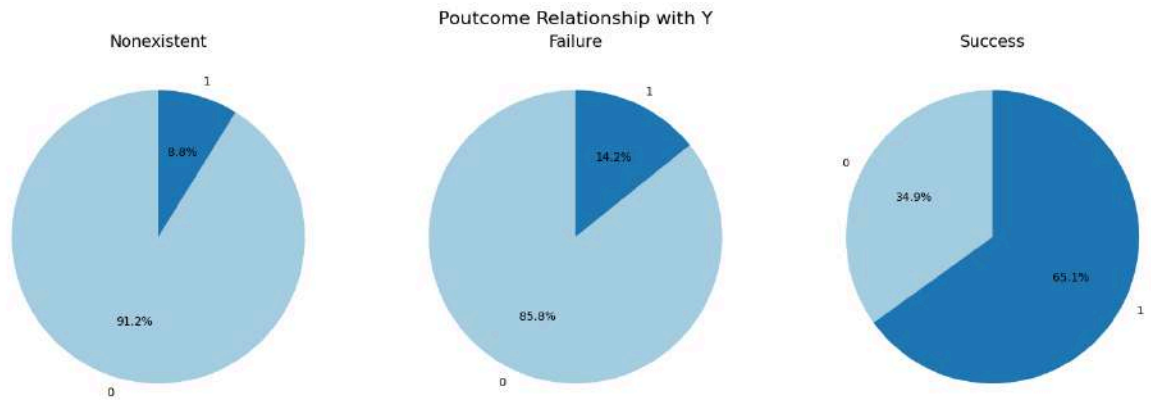


```python
# Bivariate analysis of 'poutcome' and 'y'
bivariate_analysis(marketing, 'poutcome', chart_type='bar')
```

```
Bivariate Analysis of 'poutcome' and 'y'
```




```python
# Bivariate analysis of 'poutcome' and 'y'
generate_pie_charts_per_category(marketing, 'poutcome', colors=plt.cm.Paired.colors[:2])
```



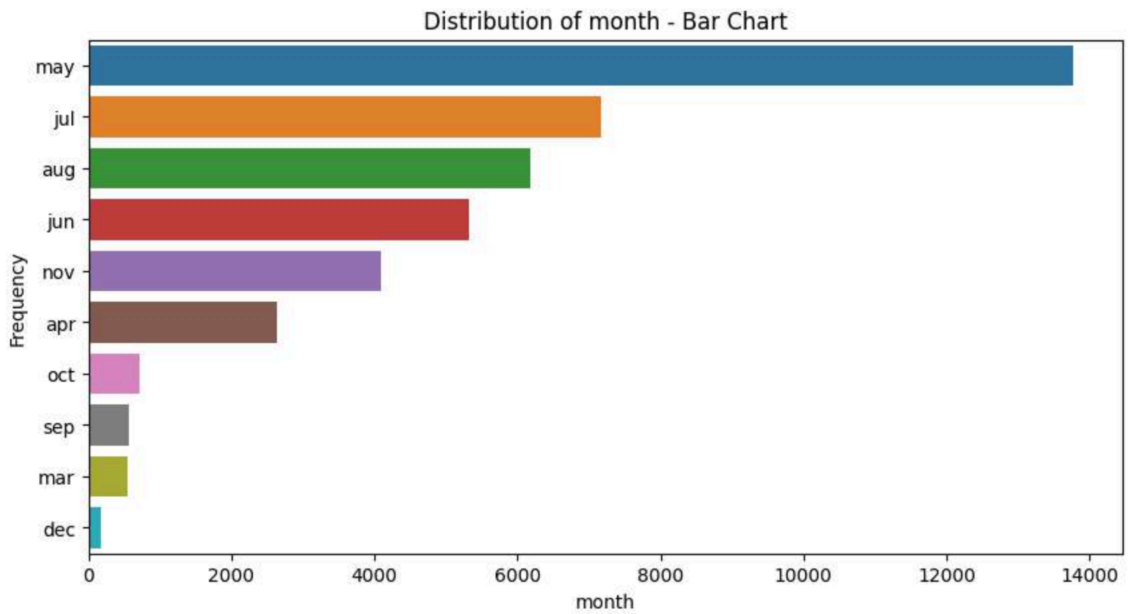

```python
# Generate summary table for 'poutcome' and 'y'
poutcome_summary = generate_summary_table(marketing, 'poutcome')

# Display the summary table
print(poutcome_summary)
```

```
             No_count  Yes_count  No_percent  Yes_percent
poutcome
success           479        892   34.938001    65.061999
failure          3645        605   85.764706    14.235294
nonexistent     32418       3141   91.166793     8.833207
```

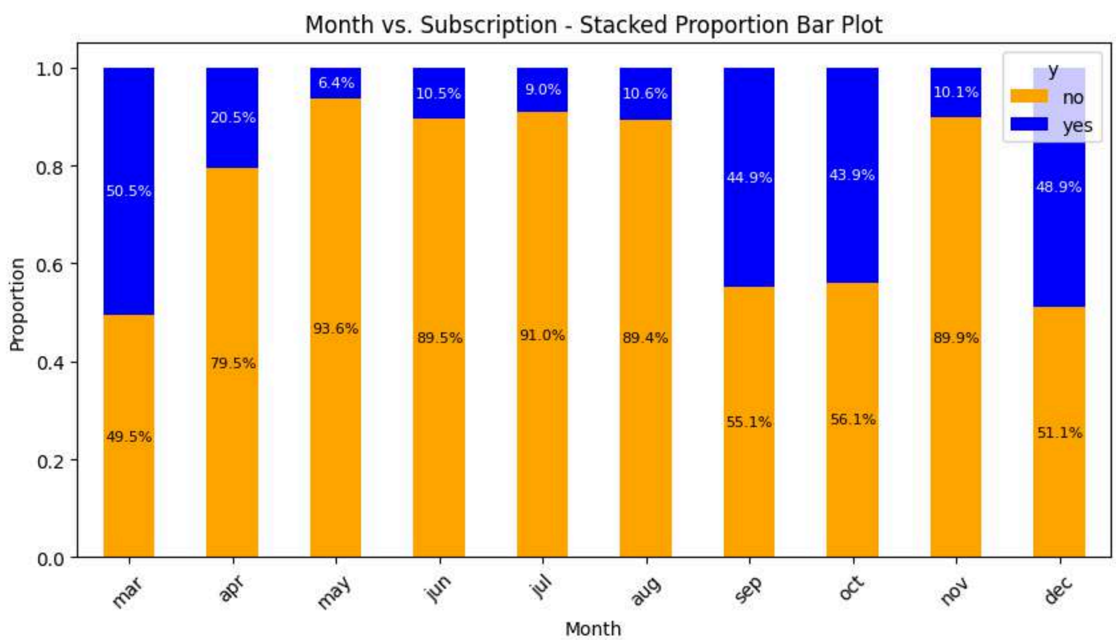
Month

```
# Analyze the 'month' variable
analyze_column(marketing, 'month', chart_type='bar')
```

```
Summary of 'month':
count     41180
unique       10
top         may
freq      13765
Name: month, dtype: object

Unique values in 'month':
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']
```



```python
import matplotlib.pyplot as plt

month_order = ['mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']

# Generate the proportion DataFrame for stacked bar plot
prop_df = (marketing.groupby(['month', 'y']).size() / marketing.groupby(['month']).size()).unstack()

# Order the DataFrame by the specified month order
prop_df = prop_df.loc[month_order]

# Plot the stacked bar plot
ax = prop_df.plot(kind='bar', stacked=True, figsize=(10, 5), color=['orange', 'blue'])
plt.title('Month vs. Subscription - Stacked Proportion Bar Plot')
plt.xlabel('Month')
plt.ylabel('Proportion')
plt.xticks(rotation=45)

# Annotate the bars with the percentages
for i in range(len(prop_df)):
    for j in range(len(prop_df.columns)):
        value = prop_df.iloc[i, j]
        # Set color based on the section of the bar: white for blue ('no') and black for orange ('yes')
        text_color = 'white' if j == 1 else 'black'
        # Display percentages with respective color
        ax.text(i, value / 2 if j == 0 else 1 - (value / 2),
                f'{value * 100:.1f}%', ha='center', va='center', color=text_color, fontsize=8)

plt.show()

# Generate and display the summary table
month_summary = generate_summary_table(marketing, 'month')
print(month_summary)
```

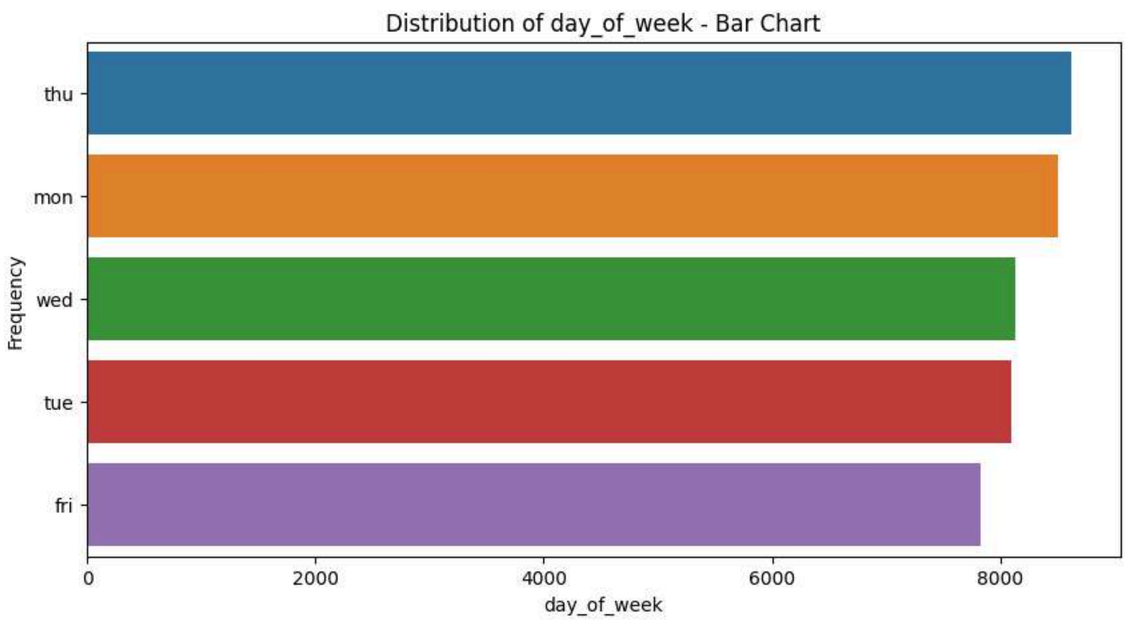



```
       No_count  Yes_count  No_percent  Yes_percent
month
mar         270        276   49.450549    50.549451
dec          93         89   51.098901    48.901099
sep         314        256   55.087719    44.912281
oct         403        315   56.128134    43.871866
apr        2093        539   79.521277    20.478723
aug        5523        655   89.397863    10.602137
jun        4759        559   89.488530    10.511470
nov        3683        414   89.895045    10.104955
jul        6525        649   90.953443     9.046557
may       12879        886   93.563385     6.436615
```

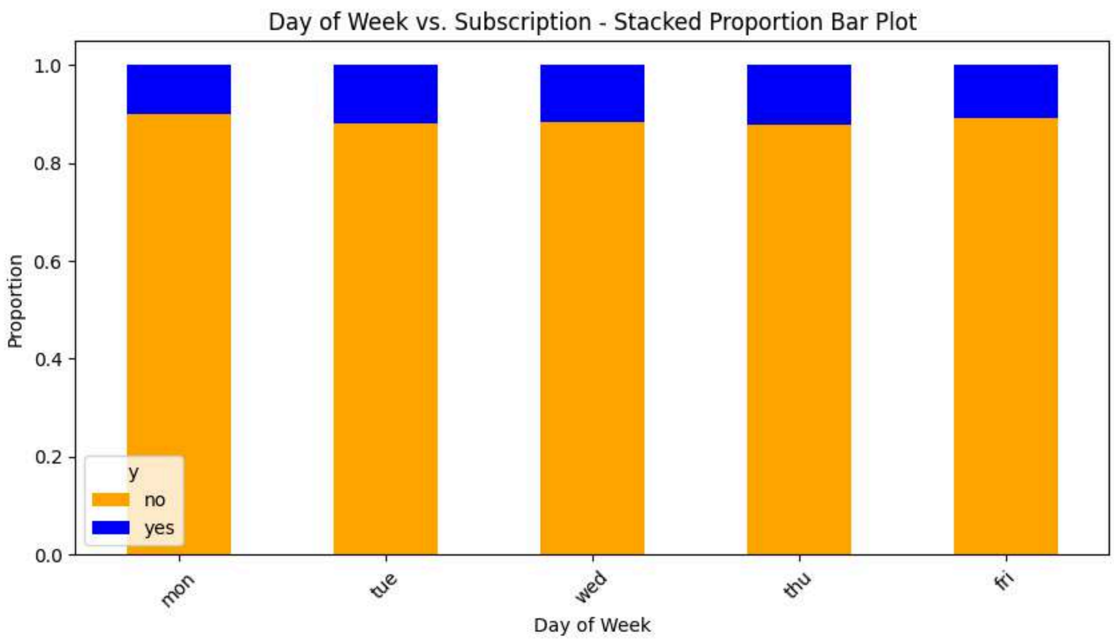
Day of Week

```
# Analyze the 'day_of_week' variable
analyze_column(marketing, 'day_of_week', chart_type='bar')
```

```
Summary of 'day_of_week':
count     41180
unique        5
top         thu
freq       8622
Name: day_of_week, dtype: object

Unique values in 'day_of_week':
['mon' 'tue' 'wed' 'thu' 'fri']
```



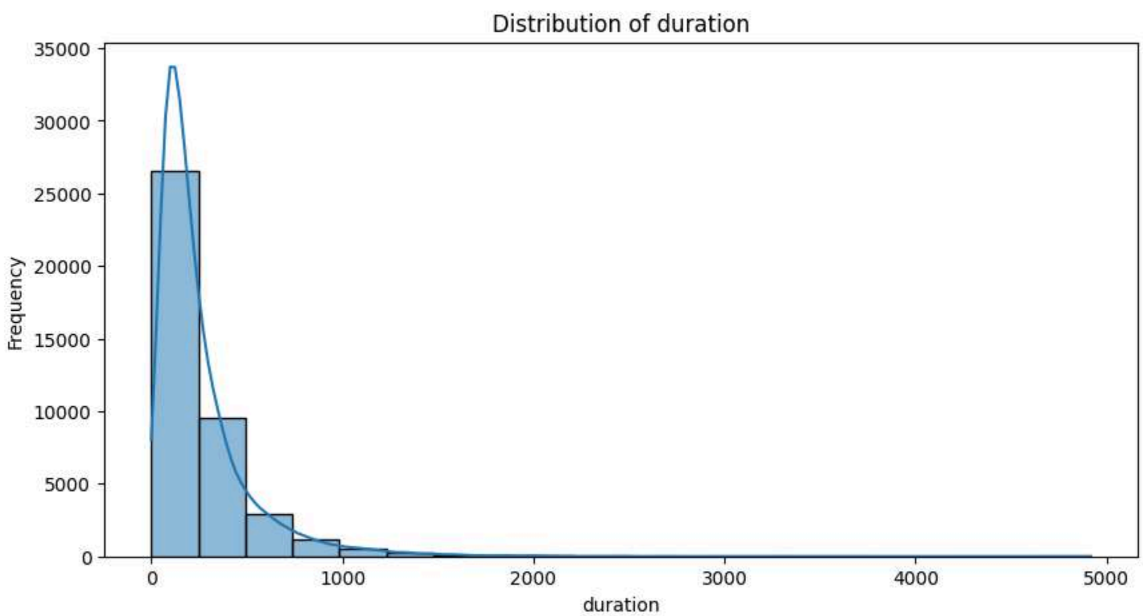
```python
# Define the correct order for the days of the week
day_order = ['mon', 'tue', 'wed', 'thu', 'fri']

# Generate the proportion DataFrame for stacked bar plot
prop_df = (marketing.groupby(['day_of_week', 'y']).size() / marketing.groupby(['day_of_week']).size()).unstack()

# Order the DataFrame by the specified day order
prop_df = prop_df.loc[day_order]

# Plot the stacked bar plot
prop_df.plot(kind='bar', stacked=True, figsize=(10, 5), color=['orange', 'blue'])
plt.title('Day of Week vs. Subscription - Stacked Proportion Bar Plot')
plt.xlabel('Day of Week')
plt.ylabel('Proportion')
plt.xticks(rotation=45)
plt.show()
```



Duration

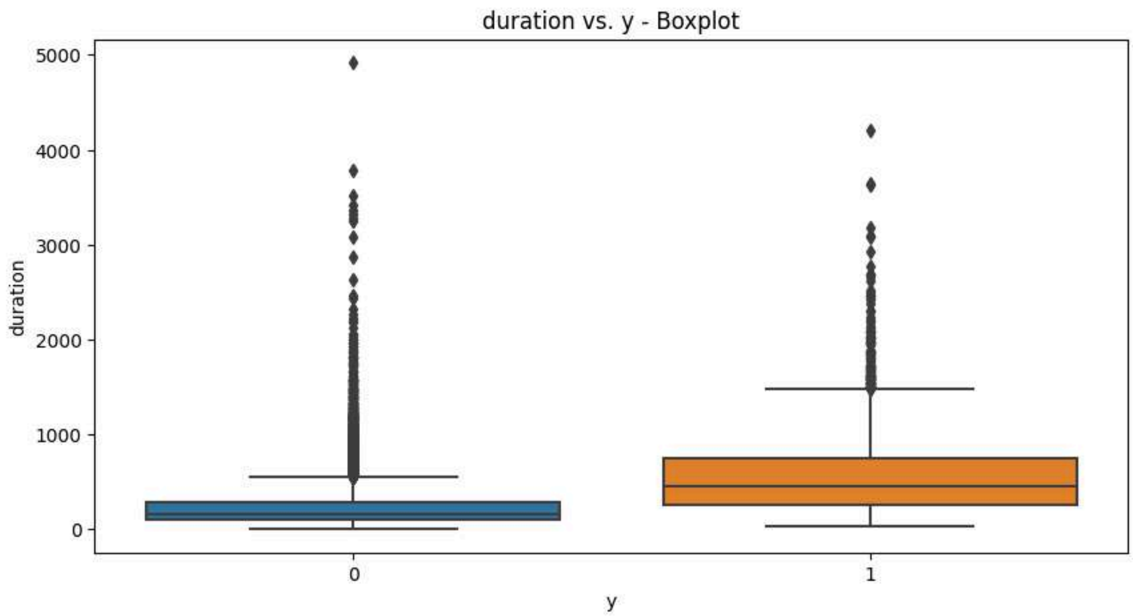
```
# Analyze the 'duration' variable
analyze_column(marketing, 'duration')
```

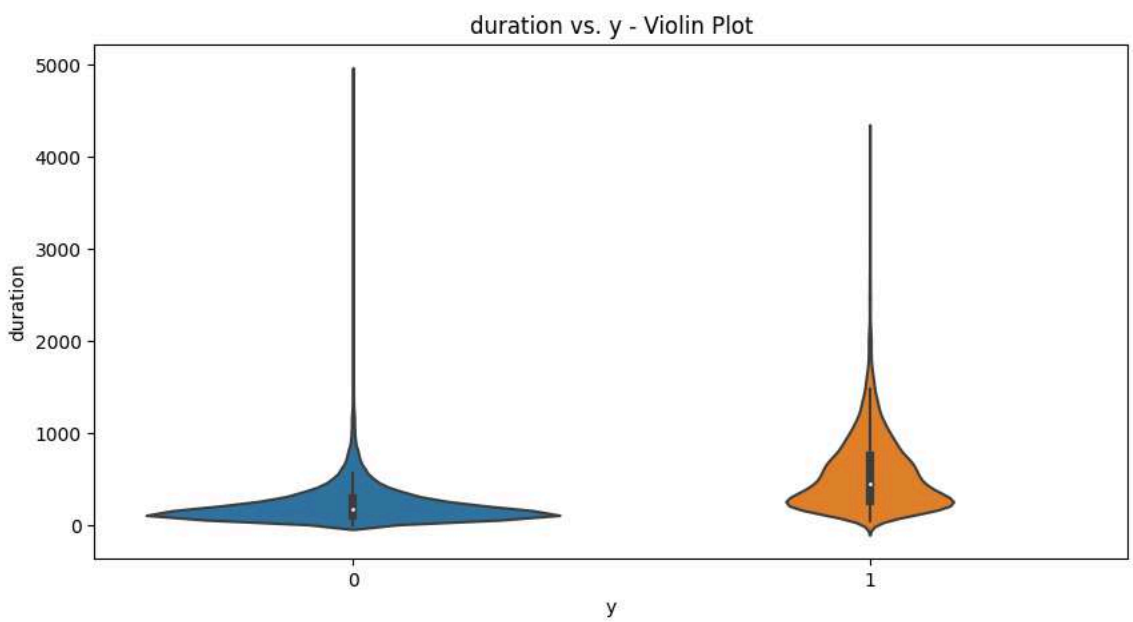
```
Summary of 'duration':
count    41180.000000
mean       258.280427
std        259.299856
min          0.000000
25%        102.000000
50%        180.000000
75%        319.000000
max       4918.000000
Name: duration, dtype: float64

Unique values in 'duration':
[ 151  307  198 ... 1246 1556 1868]
```

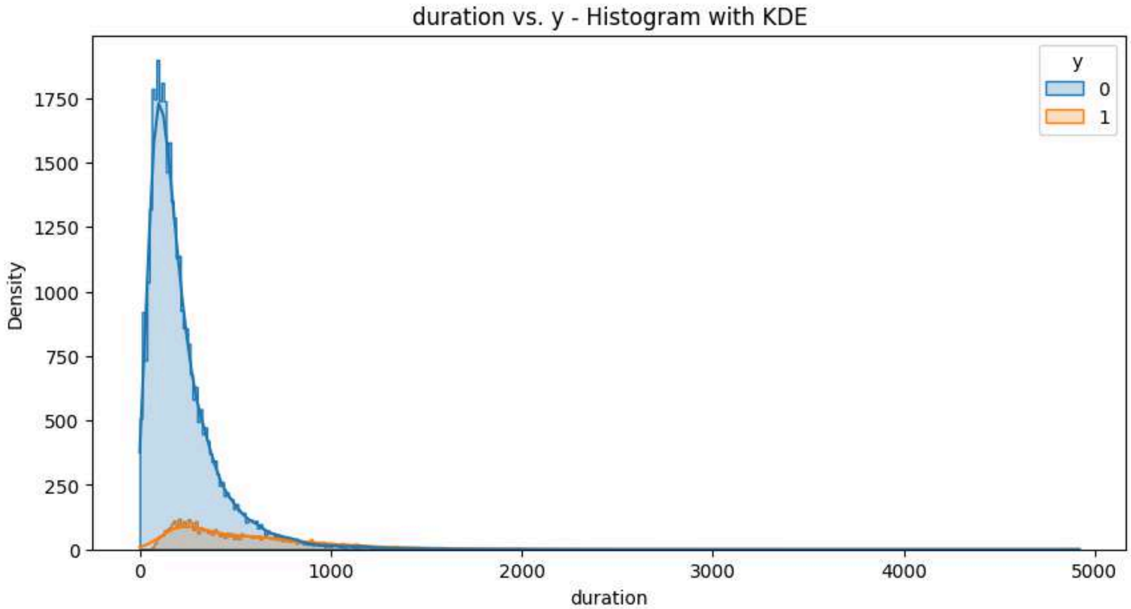


```
# Bivariate analysis of 'duration' and 'y'
bivariate_analysis(marketing, 'duration')
```

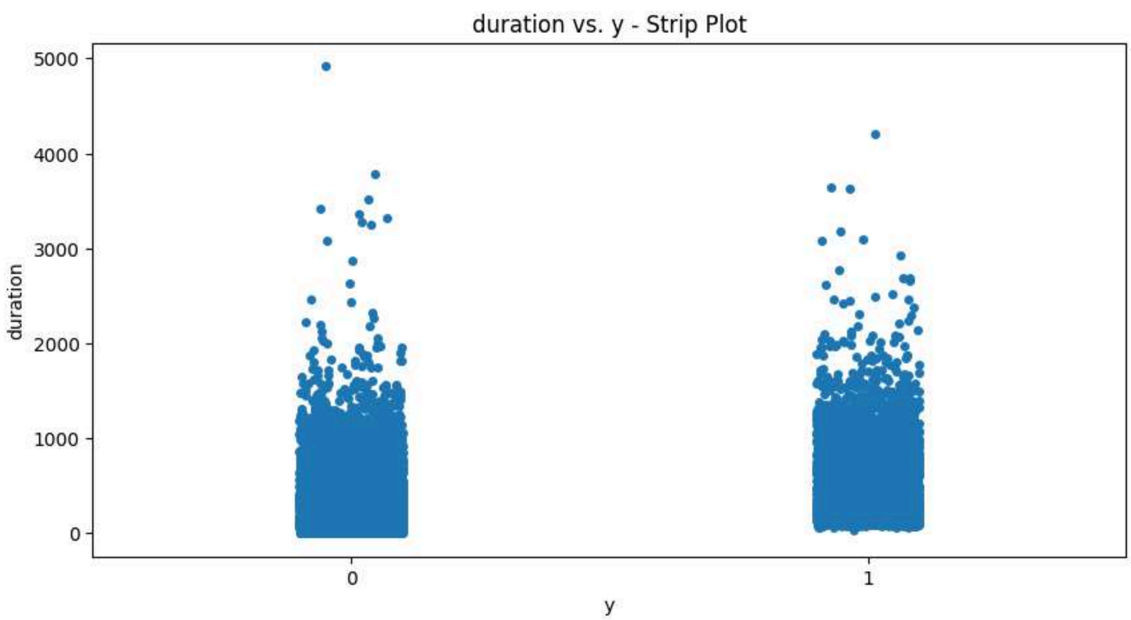
```
Bivariate Analysis of 'duration' and 'y'
```



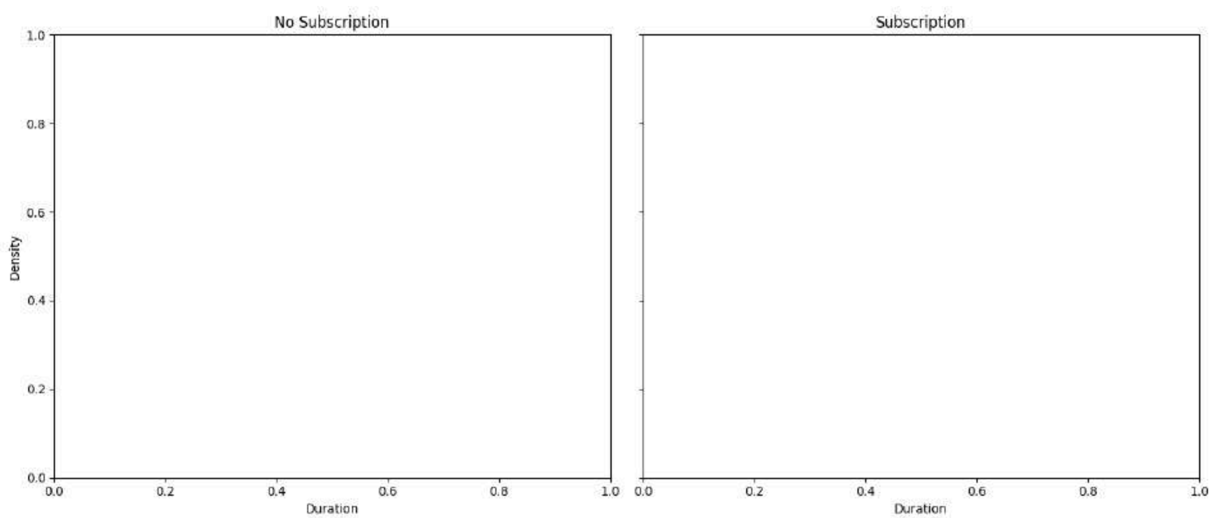


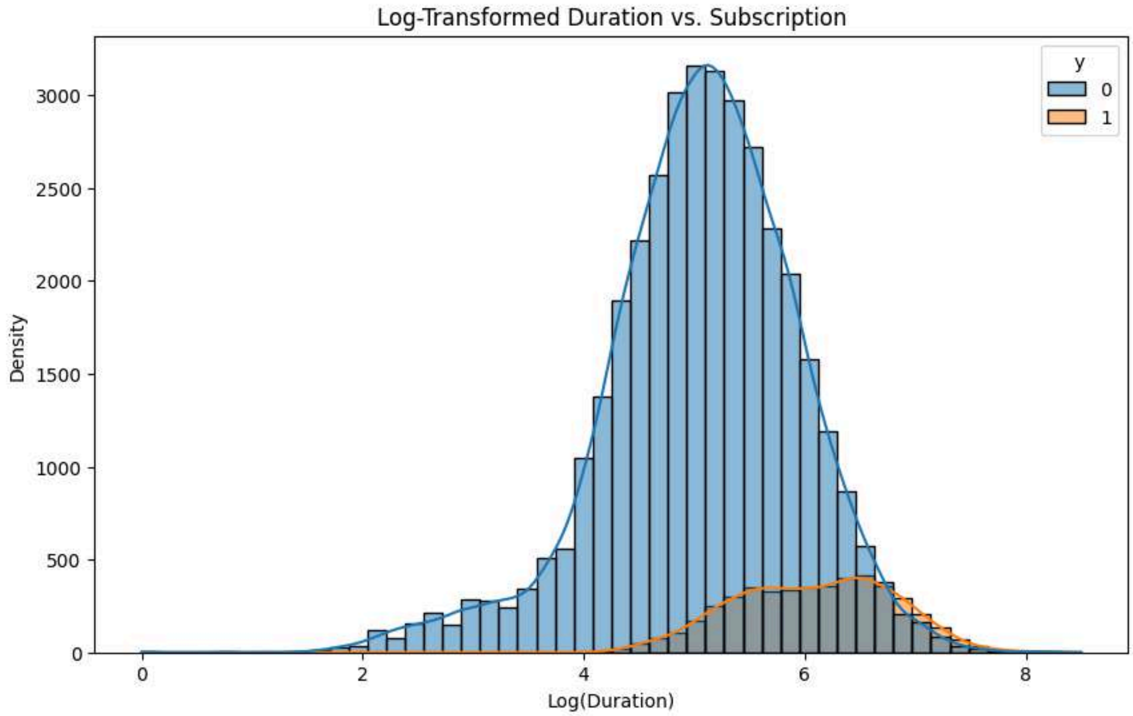


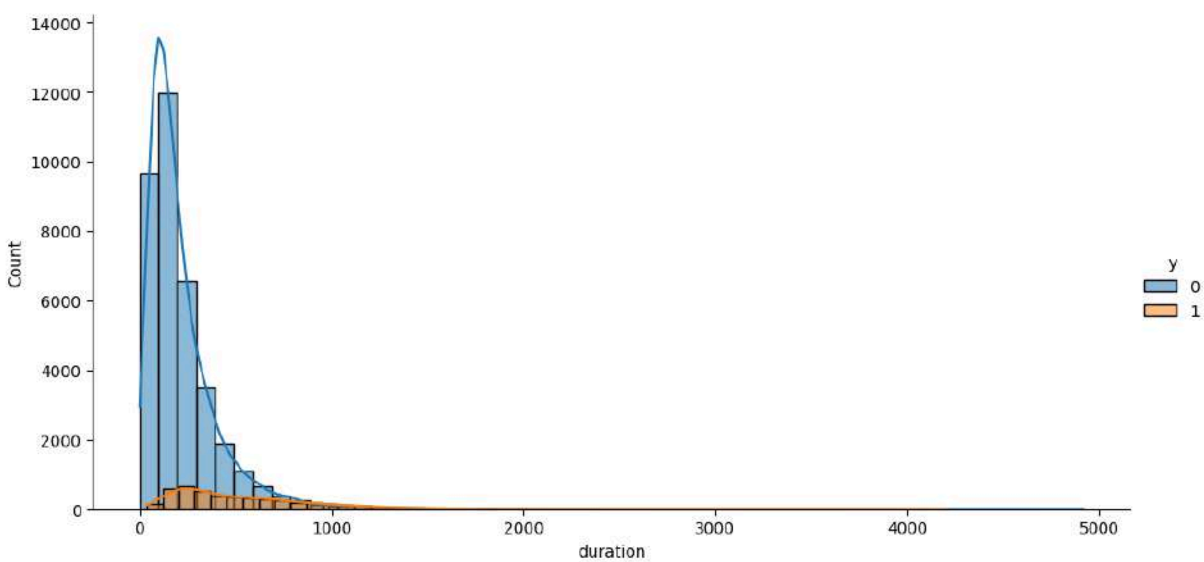

```
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
```

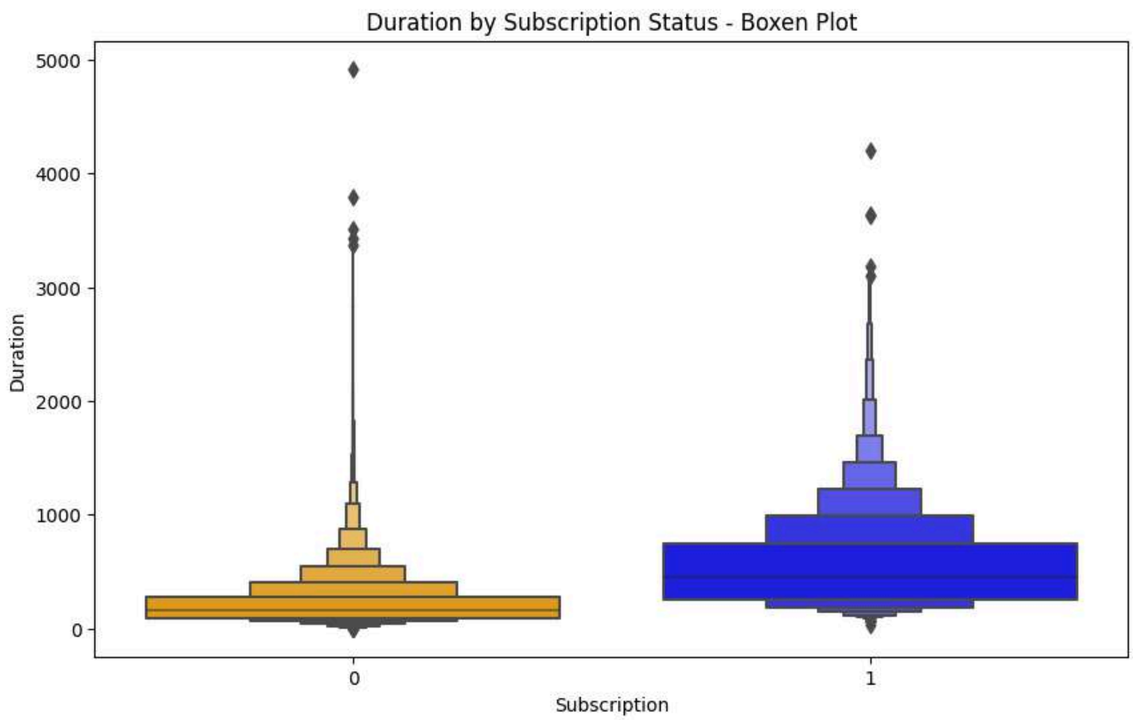


```python
# Option 1: Separate Histograms for Each Group
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)
sns.histplot(marketing[marketing['y'] == 'no']['duration'], kde=True, bins=50, ax=axes[0], color='blue')
axes[0].set_title('No Subscription')
axes[0].set_xlabel('Duration')
axes[0].set_ylabel('Density')
sns.histplot(marketing[marketing['y'] == 'yes']['duration'], kde=True, bins=50, ax=axes[1], color='orange')
axes[1].set_title('Subscription')
axes[1].set_xlabel('Duration')
plt.tight_layout()
plt.show()

# Option 2: Log Transformation
marketing['log_duration'] = np.log1p(marketing['duration'])  # Log transformation (log1p to handle zero values)
plt.figure(figsize=(10, 6))
sns.histplot(data=marketing, x='log_duration', hue='y', kde=True, bins=50)
plt.title('Log-Transformed Duration vs. Subscription')
plt.xlabel('Log(Duration)')
plt.ylabel('Density')
plt.show()

# Option 3: Faceted Histogram
g = sns.FacetGrid(marketing, hue='y', aspect=2, height=5)
g.map(sns.histplot, 'duration', bins=50, kde=True)
g.add_legend()
plt.show()

# Option 4: Boxen Plot
plt.figure(figsize=(10, 6))
sns.boxenplot(data=marketing, x='y', y='duration', palette=['orange', 'blue'])
plt.title('Duration by Subscription Status - Boxen Plot')
plt.xlabel('Subscription')
plt.ylabel('Duration')
plt.show()
```




Log-Transformed Duration vs. Subscription
Density
Log(Duration)
y
0
1
0
500
1000
1500
2000
2500
3000
0
2
4
6
8

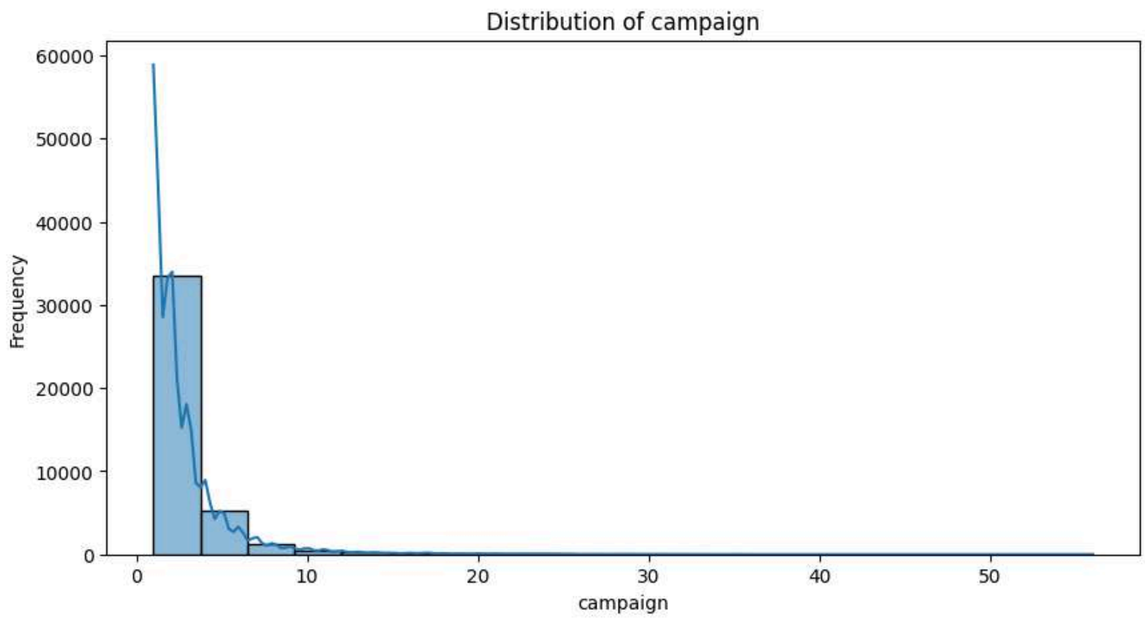

Count
duration
y
0
1
0
2000
4000
6000
8000
10000
12000
14000
0
1000
2000
3000
4000
5000


Duration by Subscription Status - Boxen Plot
5000
4000
3000
2000
1000
0
Duration
0
1
Subscription

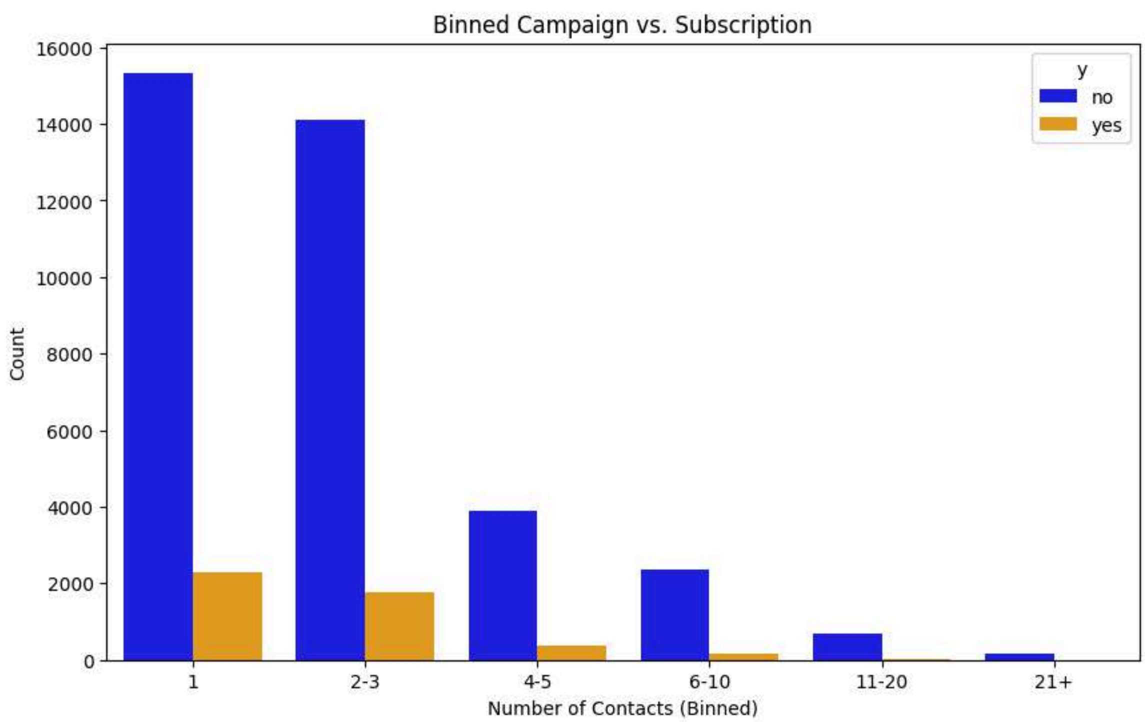
Campaign

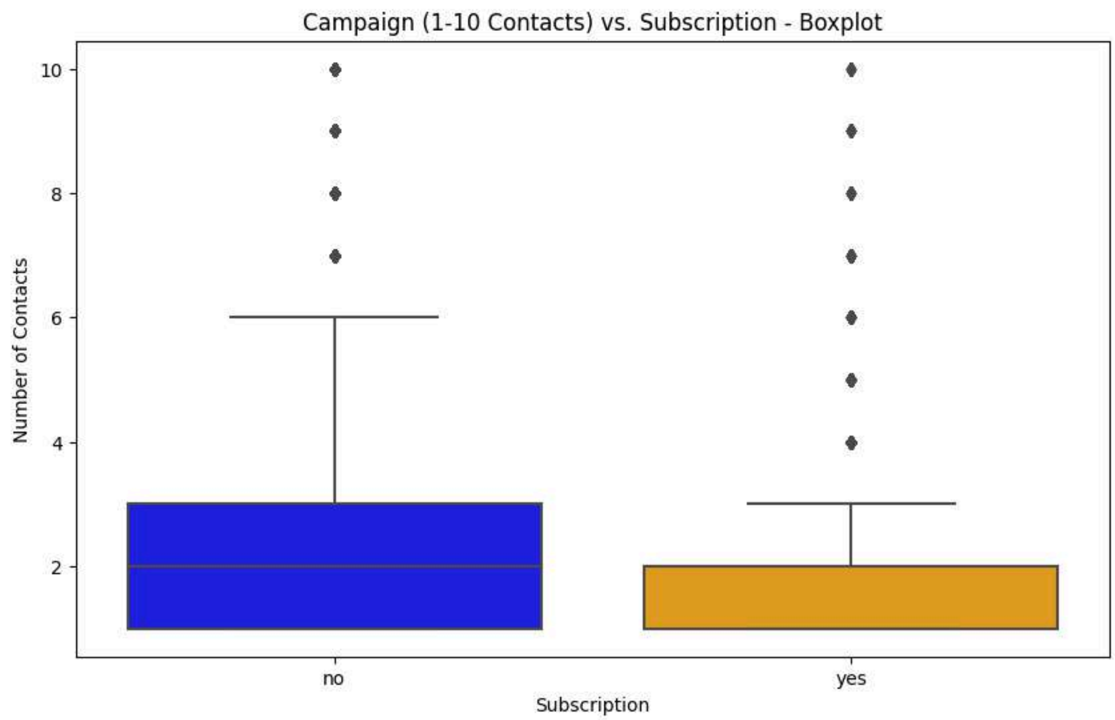
```
# Analyze the 'campaign' variable
analyze_column(marketing, 'campaign')
```

```
Summary of 'campaign':
count    41180.000000
mean         2.567800
std          2.770225
min          1.000000
25%          1.000000
50%          2.000000
75%          3.000000
max         56.000000
Name: campaign, dtype: float64

Unique values in 'campaign':
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 19 18 23 14 22 25 16 17 15 20 56
 39 35 42 28 26 27 32 21 24 29 31 30 41 37 40 33 34 43]
```



```
# Option 2: Binning
bins = [0, 1, 3, 5, 10, 20, np.inf]
labels = ['1', '2-3', '4-5', '6-10', '11-20', '21+']
marketing['campaign_binned'] = pd.cut(marketing['campaign'], bins=bins, labels=labels)
plt.figure(figsize=(10, 6))
sns.countplot(x='campaign_binned', hue='y', data=marketing, palette=['blue', 'orange'])
plt.title('Binned Campaign vs. Subscription')
plt.xlabel('Number of Contacts (Binned)')
plt.ylabel('Count')
plt.show()



# Limit the range of 'campaign' to focus on the more typical values (e.g., 1-10 contacts)
plt.figure(figsize=(10, 6))
sns.boxplot(data=marketing[marketing['campaign'] <= 10], x='y', y='campaign', palette=['blue', 'orange'])
plt.title('Campaign (1-10 Contacts) vs. Subscription - Boxplot')
plt.xlabel('Subscription')
plt.ylabel('Number of Contacts')
plt.show()
```






```python
# Generate summary statistics for the 'campaign' variable for each subscription category
campaign_summary = marketing.groupby('y')['campaign'].describe()

# Display the summary statistics
print(campaign_summary)
```

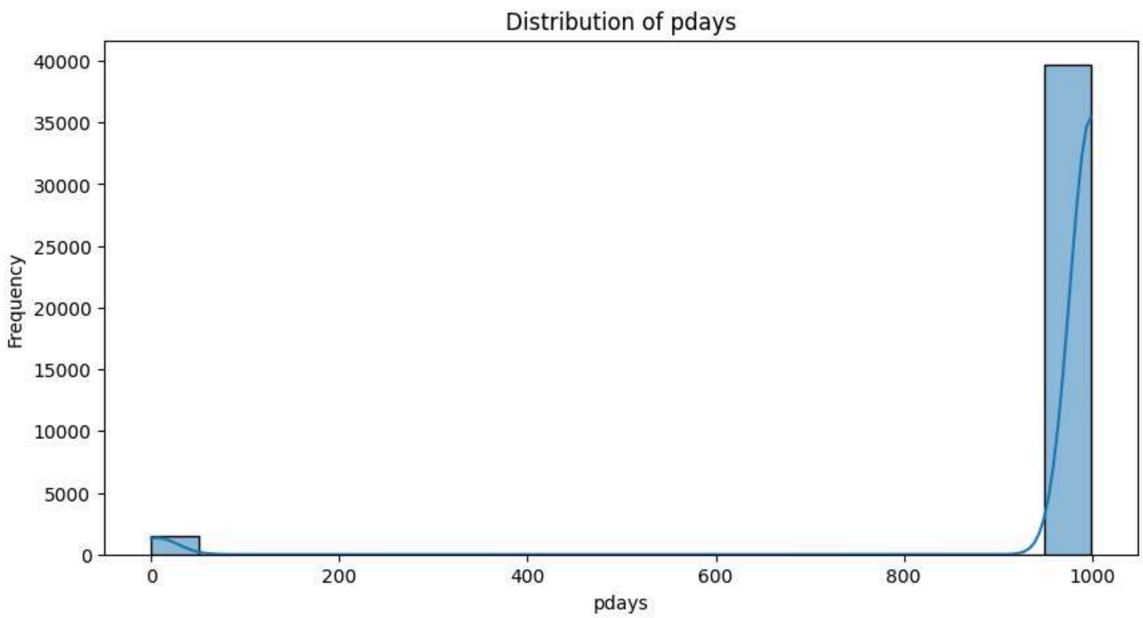
```
       count      mean       std  min  25%  50%  75%   max
y
no   36542.0  2.633271  2.873621  1.0  1.0  2.0  3.0  56.0
yes   4638.0  2.051962  1.666532  1.0  1.0  2.0  2.0  23.0
```

P-Days

Something went wrong while rendering the block. Please refresh the browser or **Send report**.

```
# Analyze the 'campaign' variable
analyze_column(marketing, 'pdays')
```

```
Summary of 'pdays':
count    41180.000000
mean       962.516707
std        186.809028
min          0.000000
25%        999.000000
50%        999.000000
75%        999.000000
max        999.000000
Name: pdays, dtype: float64

Unique values in 'pdays':
[999   6   4   3   5   1   0  10   7   8   9  11   2  12  13  14  15  16
  21  17  18  22  25  26  19  27  20]
```




```
# Create a new column for pdays categories
marketing['pdays_category'] = marketing['pdays'].apply(lambda x: 'Not Contacted (-1)' if x == -1 else 'Contacted Previous

# Display the counts of each category
print(marketing['pdays_category'].value_counts())
```

```
Contacted Previously    41180
Name: pdays_category, dtype: int64
```

```
original_data['pdays'].unique()
```

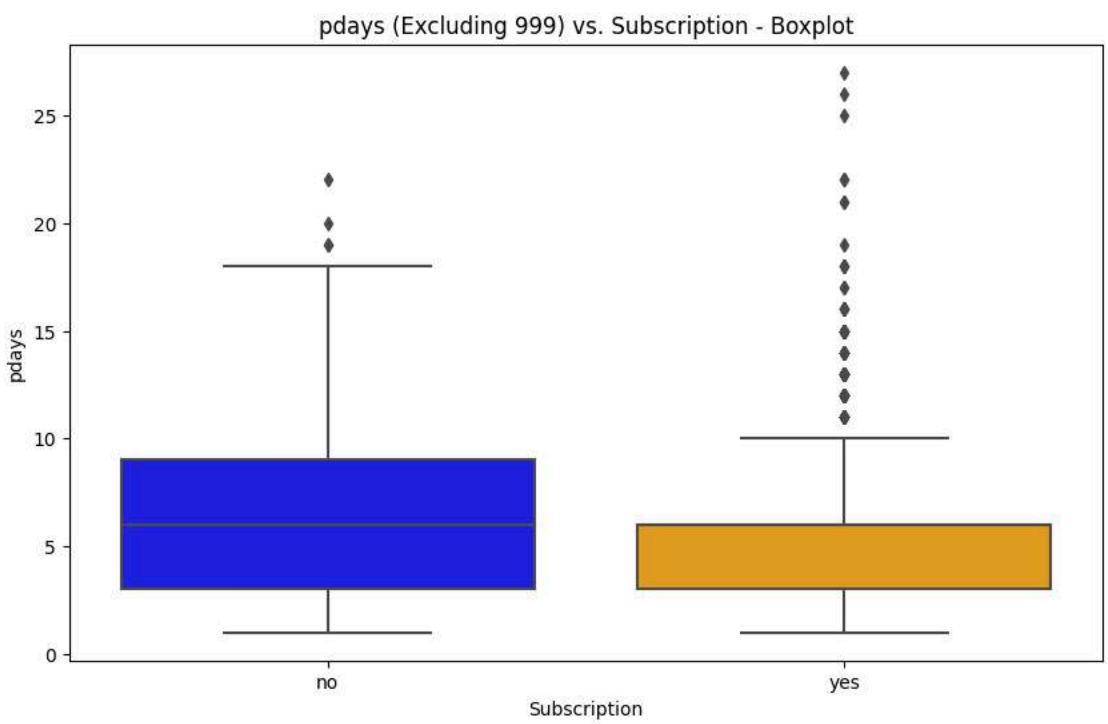
```
array(['999', '6', '4', '3', '5', '1', '0', '10', '7', '8', '9', '11',
       '2', '12', '13', '14', '15', '16', '21', '17', '18', '22', '25',
       '26', '19', '27', '20'], dtype=object)
```

The value for 999 seems to be the -1 which explains when there is no call we should analyze it differently

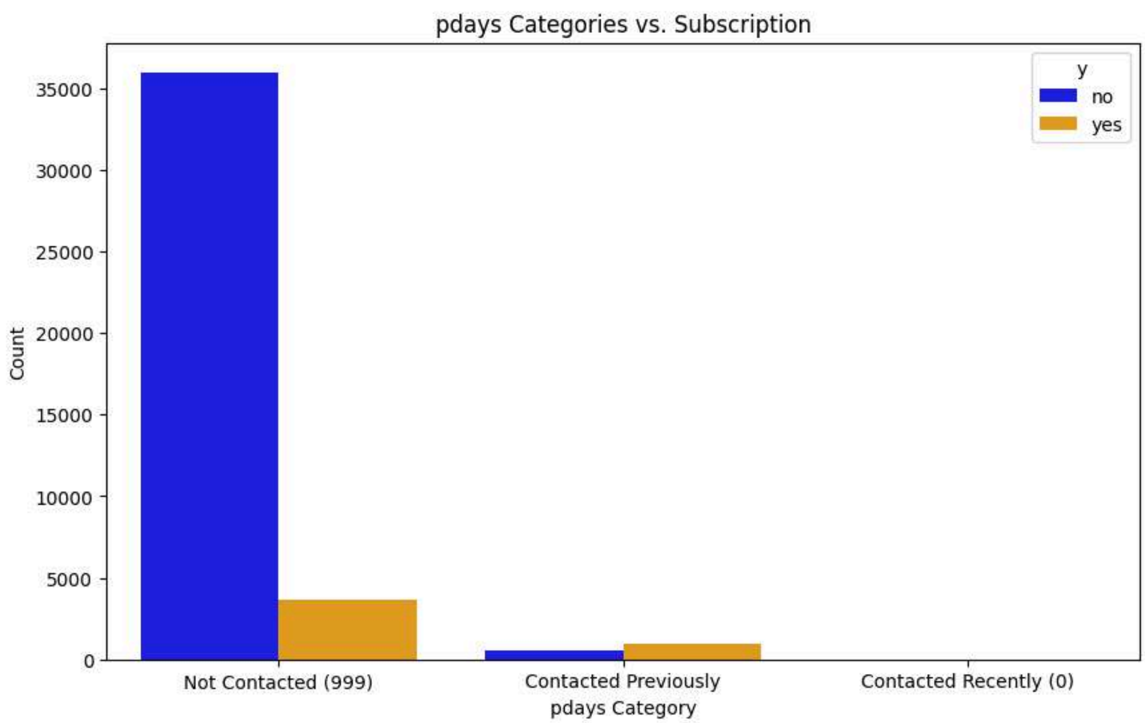
```
marketing['pdays_category'] = marketing['pdays'].apply(lambda x: 'Not Contacted (999)' if x == 999 else ('Contacted Recen

# Display the counts of each category
print(marketing['pdays_category'].value_counts())
```

```
Not Contacted (999)      39667
Contacted Previously      1498
Contacted Recently (0)      15
Name: pdays_category, dtype: int64
```

```
# Filter out 'pdays = 999' and possibly 'pdays = 0'
filtered_pdays = marketing[(marketing['pdays'] > 0) & (marketing['pdays'] < 999)]

# Re-plot the box plot after filtering out 'pdays = 999'
plt.figure(figsize=(10, 6))
sns.boxplot(data=filtered_pdays, x='y', y='pdays', palette=['blue', 'orange'])
plt.title('pdays (Excluding 999) vs. Subscription - Boxplot')
plt.xlabel('Subscription')
plt.ylabel('pdays')
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.countplot(x='pdays_category', hue='y', data=marketing, palette=['blue', 'orange'])
plt.title('pdays Categories vs. Subscription')
plt.xlabel('pdays Category')
plt.ylabel('Count')
plt.show()
```



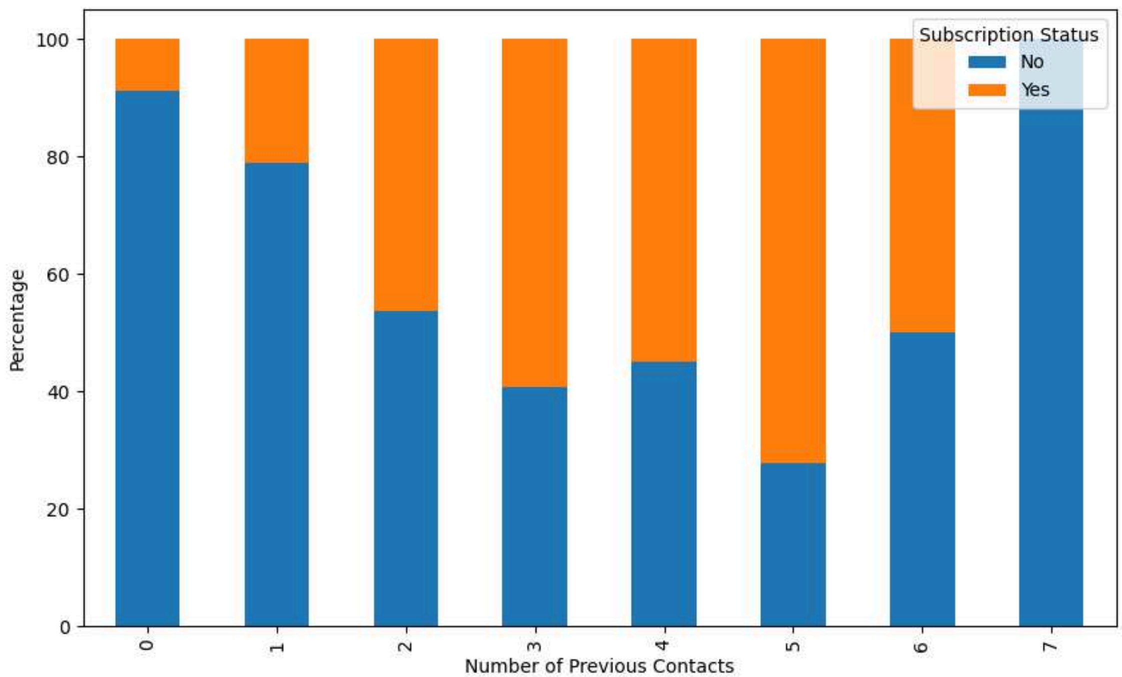

```
# Group the data by 'pdays_category' and 'y', and calculate the count
summary_table = marketing.groupby(['pdays_category', 'y']).size().unstack(fill_value=0)

# Rename the columns for better understanding
summary_table.columns = ['No_count', 'Yes_count']

# Calculate percentages for 'yes' and 'no' in each category
summary_table['No_percent'] = summary_table['No_count'] / (summary_table['No_count'] + summary_table['Yes_count']) * 100
summary_table['Yes_percent'] = summary_table['Yes_count'] / (summary_table['No_count'] + summary_table['Yes_count']) * 10

# Display the summary table
summary_table.reset_index(inplace=True)
summary_table
```

	pdays_category	No_count	Yes_count	No_percent	Yes_percent
0	Contacted Previously	543	955	36.248331	63.751669
1	Contacted Recently (0)	5	10	33.333333	66.666667
2	Not Contacted (999)	35994	3673	90.740414	9.259586

Previous

Something went wrong while rendering the block. Please refresh the browser or **Send report**.

```
# Group the data by 'previous' and 'y', count the occurrences, and normalize for percentages
grouped_data = marketing.groupby(['previous', 'y']).size().unstack(fill_value=0)

# Calculate percentages
grouped_data_percentage = grouped_data.div(grouped_data.sum(axis=1), axis=0) * 100

# Plot the stacked bar chart with percentages
grouped_data_percentage.plot(kind='bar', stacked=True, figsize=(10, 6), color=['#1f77b4', '#ff7f0e'])
#plt.title('Number of Previous Contacts vs. Subscription Status (Percentage)')
plt.xlabel('Number of Previous Contacts')
plt.ylabel('Percentage')
plt.legend(title='Subscription Status', labels=['No', 'Yes'])
plt.show()
```



```
# Group the data by 'previous' and 'y' and calculate counts
previous_summary = marketing.groupby(['previous', 'y']).size().unstack(fill_value=0)

# Rename the columns for better understanding
previous_summary.columns = ['No_count', 'Yes_count']

# Calculate percentages for 'yes' and 'no' in each 'previous' category
previous_summary['No_percent'] = previous_summary['No_count'] / (previous_summary['No_count'] + previous_summary['Yes_cou
previous_summary['Yes_percent'] = previous_summary['Yes_count'] / (previous_summary['No_count'] + previous_summary['Yes_c

# Reset the index to make it a clean table
previous_summary.reset_index(inplace=True)

# Display the summary table
previous_summary
```

	previous	No_count	Yes_count	No_percent	Yes_percent
0	0	32418	3141	91.166793	8.833207
1	1	3593	966	78.811143	21.188857
2	2	404	350	53.580902	46.419098
3	3	88	128	40.740741	59.259259
4	4	31	38	44.927536	55.072464
5	5	5	13	27.777778	72.222222
6	6	2	2	50.000000	50.000000
7	7	1	0	100.000000	0.000000

Feature Selection

The feature selection is based on the EDA analysis and the 2 correlation methods used before. For numerical the biserial correlation and for the categorical the Crammer V methodology. We can use some statistical tests to confirm this.

```
numerical_cols_discard = ['age']  ## cols with treshold correlation of 0.05
cat_cols_discard = ['loan','housing','day_of_week'] ## cols with treshold correlation of 0.05
```

A second approach, will be discarding the variables using multicolinearity. If you see the correlation matrix above, there are 2 variables that have high correlation among each other. This can generate some bias and we can just leave the ones that they have high relationship with.

1. cons.price
2. emp.var.rate

Present high correlation with euriborn and number of employees (and it is normal because they represent same information: on one hand economic financial situation and on the other hand employee information of the ccompaby).

```
numerical_cols_discard = numerical_cols_discard + ['cons.price.idx', 'emp.var.rate']
```

```
marketing_eng = marketing.copy()
marketing_eng.drop(columns=numerical_cols_discard + cat_cols_discard, inplace=True)
marketing_eng.head()
```

	job	marital	education	default	contact	month	duration	campaign	pdays	p
0	admin.	married	basic.6y	no	telephone	may	151	1	999	0
1	services	married	high.school	no	telephone	may	307	1	999	0
2	services	married	basic.9y	unknown	telephone	may	198	1	999	0
3	admin.	married	professional.course	no	telephone	may	139	1	999	0
4	blue-collar	married	unknown	unknown	telephone	may	217	1	999	0

```
marketing_eng.shape
```

```
(41180, 15)
```

Data Cleaning

Missing Value Check

```
marketing_clean = marketing_eng.copy()
marketing_clean.isnull().sum().sum()
```

```
0
```

Duplicates

```
marketing_clean[marketing_clean.duplicated()]
```

	job	marital	education	default	contact	month	duration	campaign	pdays
232	blue-collar	married	basic.4y	no	telephone	may	136	1	999
345	blue-collar	married	basic.4y	no	telephone	may	164	2	999
489	blue-collar	married	basic.9y	unknown	telephone	may	46	1	999
867	blue-collar	married	high.school	no	telephone	may	294	1	999
982	blue-collar	married	basic.6y	no	telephone	may	123	1	999
...	...	...	...	...	...	...	...	...	...
35994	blue-collar	married	basic.9y	no	cellular	may	135	1	999
36590	admin.	married	university.degree	no	cellular	jun	151	1	999
36947	admin.	married	university.degree	no	cellular	jul	252	1	999
38277	retired	single	university.degree	no	telephone	oct	120	1	999
40249	admin.	married	university.degree	no	telephone	jul	7	1	999

194 rows × 15 columns

We cannot assess duplicates beccause we do not have a column that identifies customers to differentiate them. And the customers can have the same parameters and be different results.

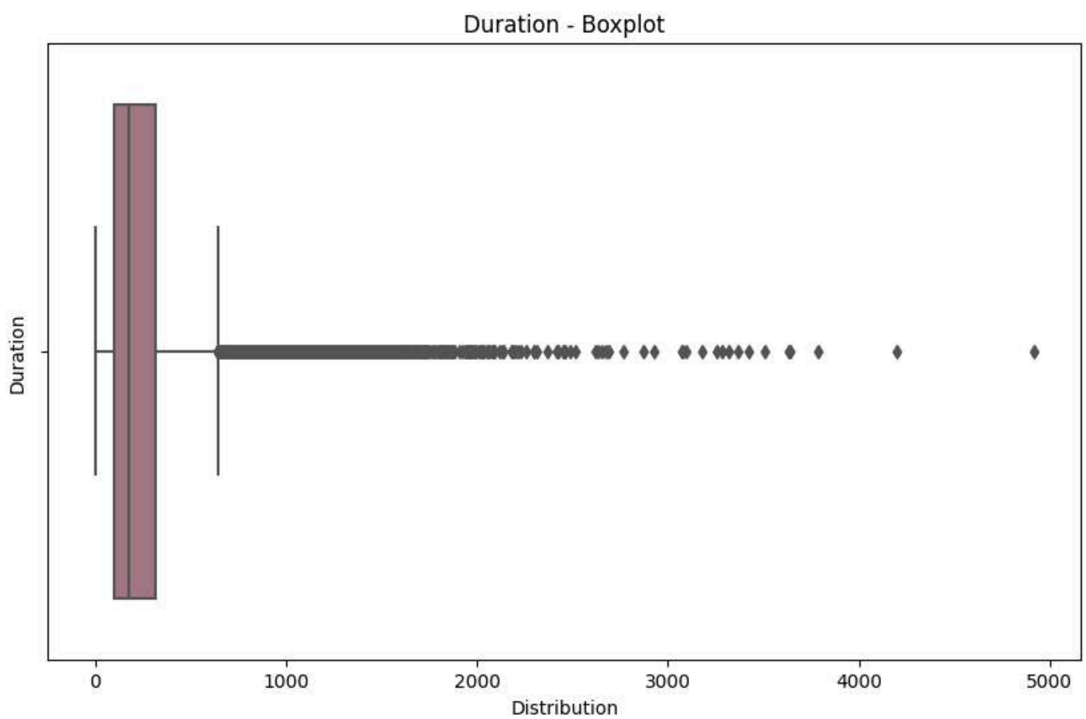
Outliers Detection Analysis

```
#checking for outliers
outliers_dict = {}

for column in marketing_clean.select_dtypes(include=['number']).columns:
    z_scores = np.abs((marketing_clean[column] - marketing_clean[column].mean()) / marketing_clean[column].std())
    outliers = z_scores > 4
    outliers_dict[column] = outliers.sum()

outliers_data = pd.DataFrame(list(outliers_dict.items()), columns=['Column', 'Outliers'])

outliers_data
```

	Column	Outliers
0	duration	386
1	campaign	475
2	pdays	1513
3	previous	308
4	cons.conf.idx	0
5	euribor3m	0
6	nr.employed	0
7	y	0

Duration Analysis

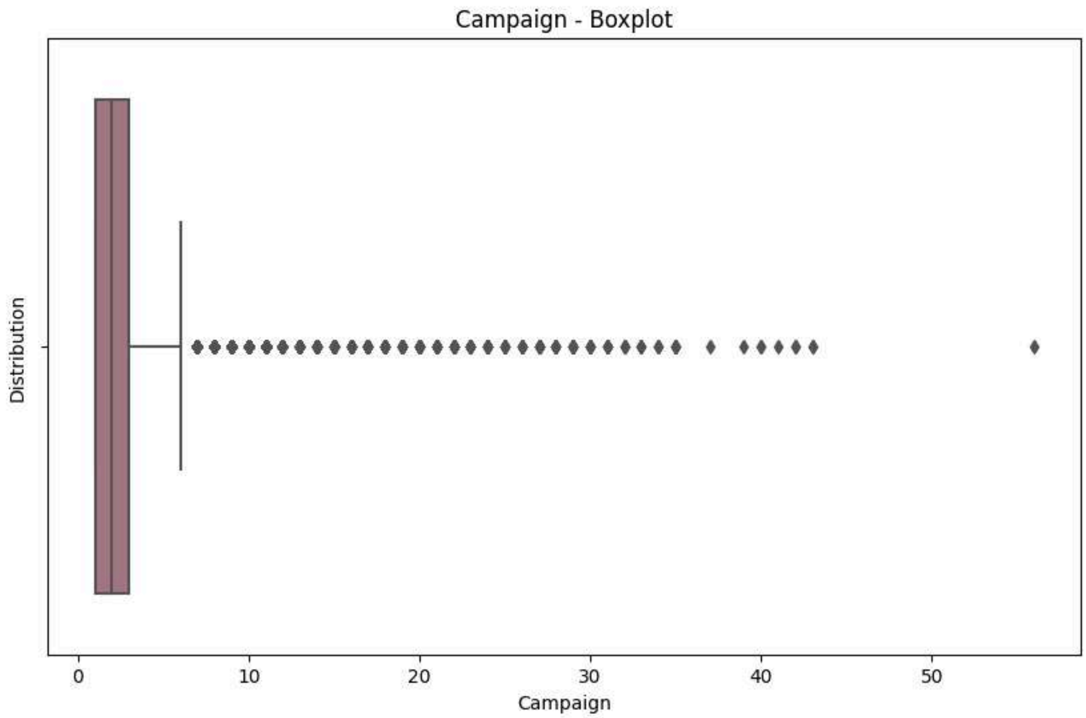
```
plt.figure(figsize=(10, 6))

# Sample a discrete palette from the cubehelix colormap
palette = sns.color_palette("cubehelix", 10)  # Adjust the number of colors if needed

# Plot the boxplot with the discrete palette
sns.boxplot(data=marketing_clean, x='duration', color = '#ac6f82')

# Add titles and labels
plt.title('Duration - Boxplot')
plt.xlabel('Distribution')
plt.ylabel('Duration')
plt.show()
```




```
## lets drop duration because we cannot use it as predictor because we dont know the duration of the call before the call
marketing_clean.drop(columns=['duration'], inplace=True)
```

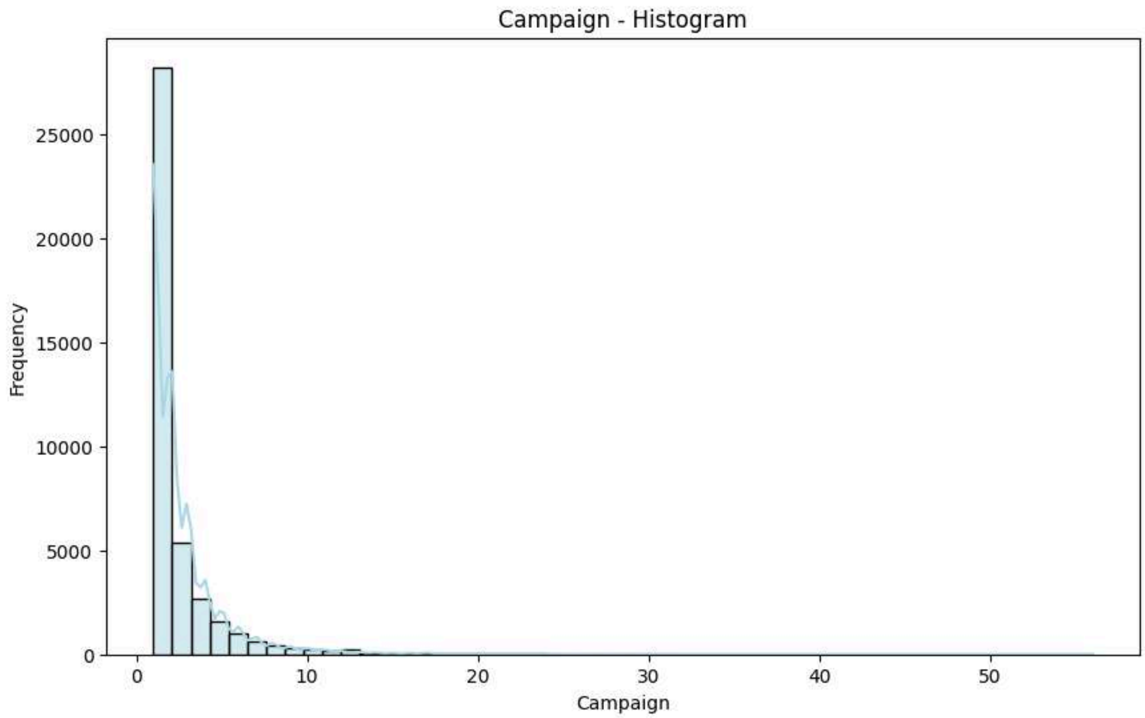
Campaign

```
## plot boxplot of campaign
plt.figure(figsize=(10, 6))
sns.boxplot(data=marketing_clean, x='campaign', color='#ac6f82')
plt.title('Campaign - Boxplot')
plt.xlabel('Campaign')
plt.ylabel('Distribution')
plt.show()
```



For this feature we will generate another category that can repesetn the feature binning. So in this sense it would be good to leave the variable as it is. Beccause too many contacts. At the end we can discard it and only usee the

```
## now the hist
plt.figure(figsize=(10, 6))
sns.histplot(marketing_clean['campaign'], kde=True, bins=50, color='#ADD8E6')
plt.title('Campaign - Histogram')
plt.xlabel('Campaign')
plt.ylabel('Frequency')
plt.show()
```



We should generate feature engineering with this variable. Doing a log transformation is not gpod approach variable is ordinal.

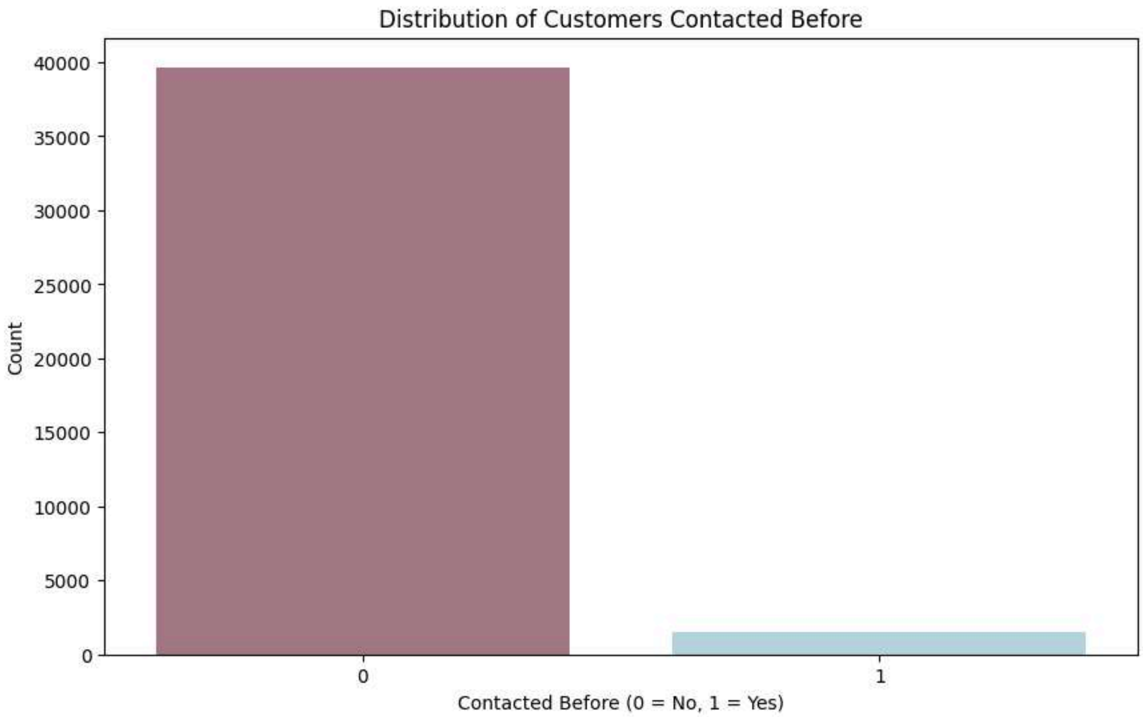
P-days

We should do feature engineering and generate a variable that tells if the person was or not contacted before.

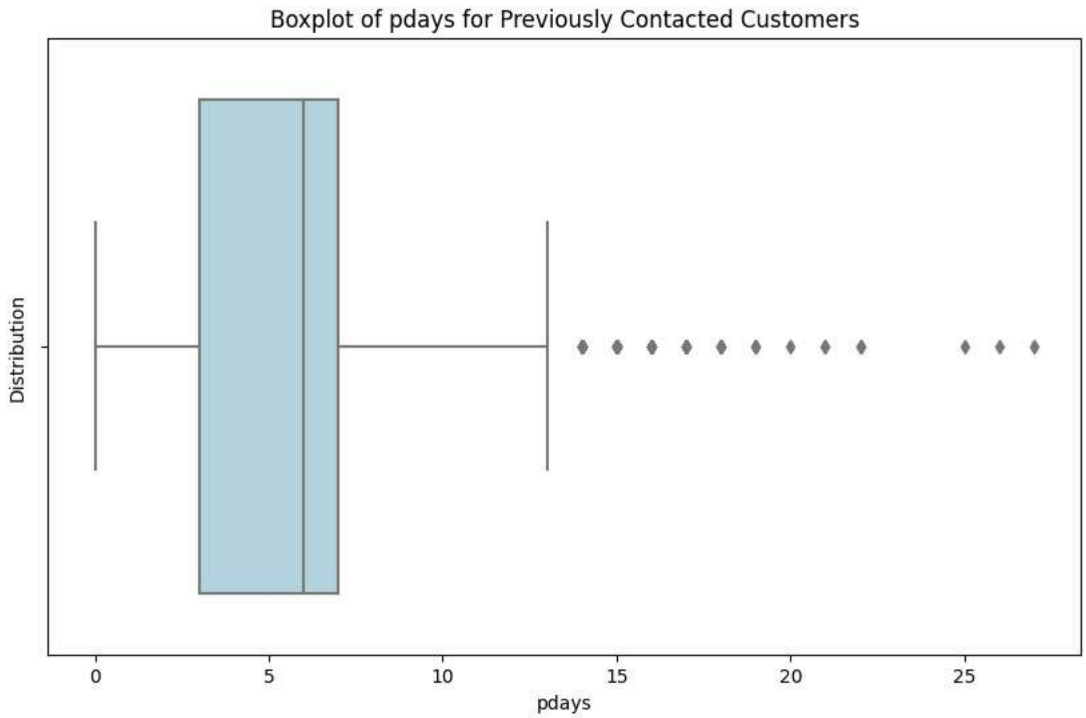
```
## We need to change 999 which means not contacted ti -1,
marketing_clean['pdays'] = marketing_clean['pdays'].replace(999, -1)
```

```
marketing_clean['contacted_before'] = marketing_clean['pdays'].apply(lambda x: 1 if x != -1 else 0)
```

```
plt.figure(figsize=(10, 6))
sns.countplot(data=marketing_clean, x='contacted_before', palette=['#ac6f82','#ADD8E6'])
plt.title('Distribution of Customers Contacted Before')
plt.xlabel('Contacted Before (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```



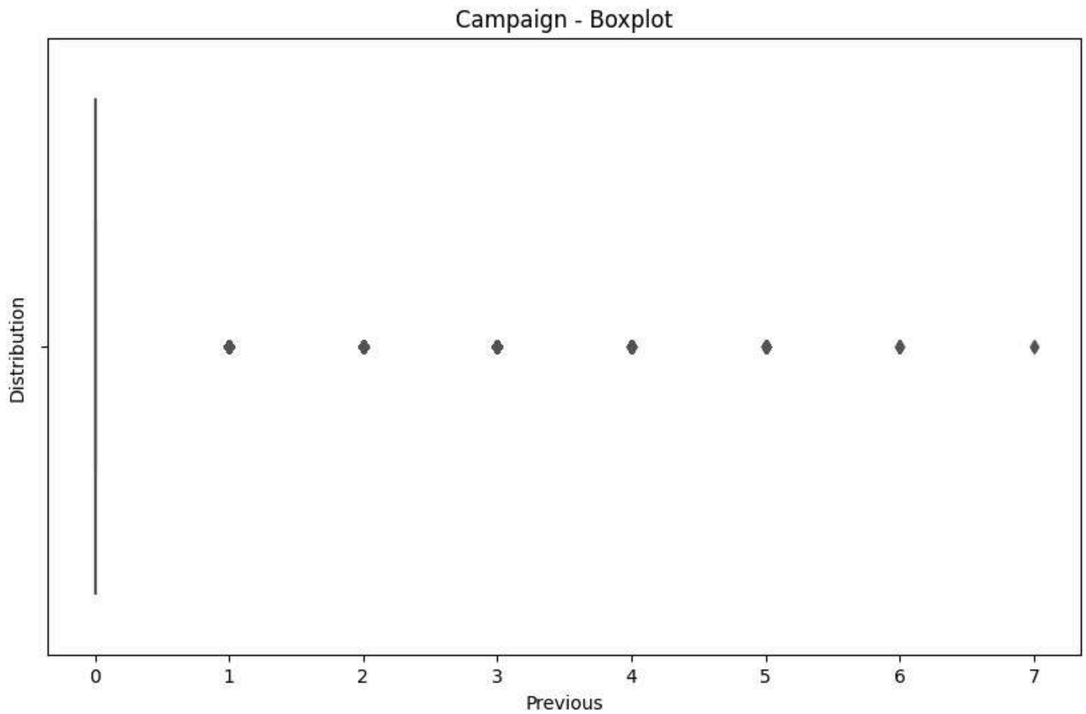
```python
contacted_df = marketing_clean[marketing_clean['pdays'] != -1]
plt.figure(figsize=(10, 6))
sns.boxplot(data=contacted_df, x='pdays', color='#ADD8E6')
plt.title('Boxplot of pdays for Previously Contacted Customers')
plt.xlabel('pdays')
plt.ylabel('Distribution')
plt.show()
```



In this sense, the outliers were not so much as seen before there were the people that were contacted before because most of the people were not contacted. In this way we can assess the outliers.

Previous

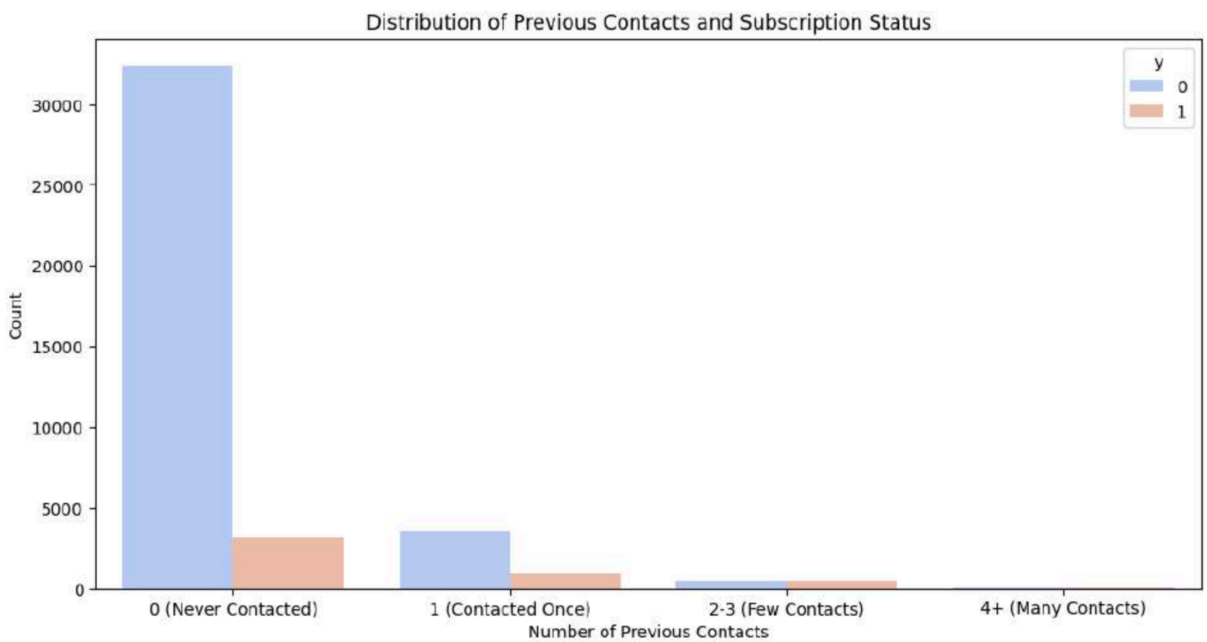
```
## plot boxplot of campaign
plt.figure(figsize=(10, 6))
sns.boxplot(data=marketing_clean, x='previous', color='#ac6f82')
plt.title('Campaign - Boxplot')
plt.xlabel('Previous')
plt.ylabel('Distribution')
plt.show()
```



Feature Engineering

Previous - Binning contacts

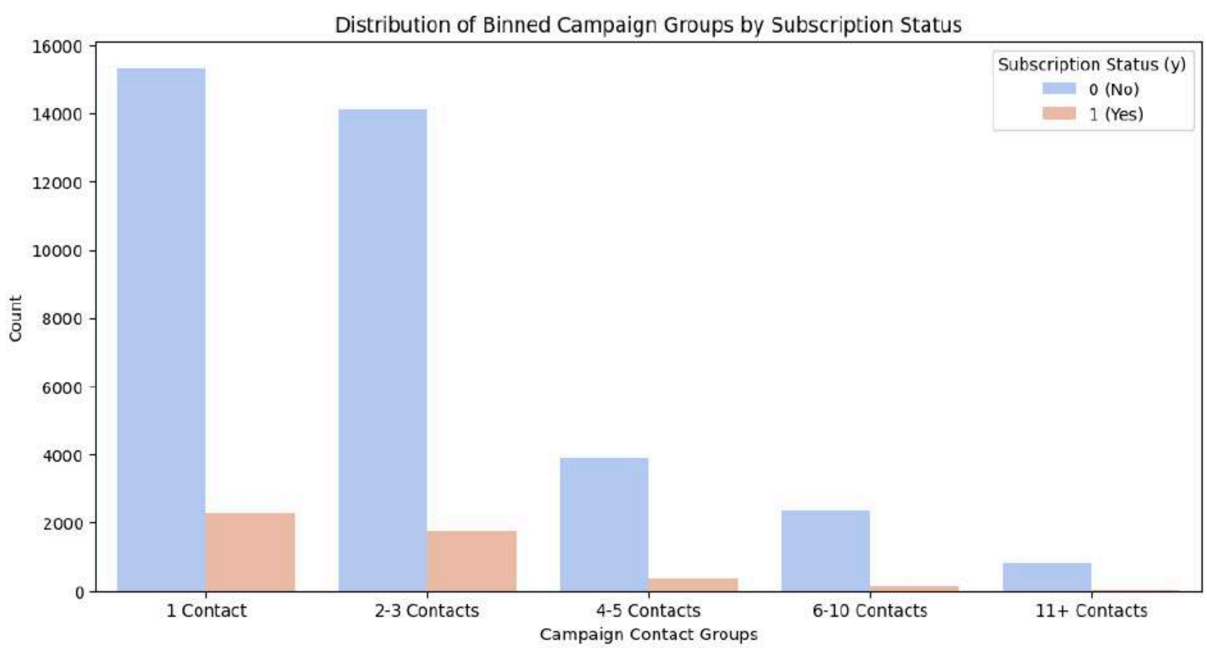
```
marketing_clean['previous_binned'] = pd.cut(
    marketing_clean['previous'],
    bins=[-1, 0, 1, 3, float('inf')],
    labels=['0 (Never Contacted)', '1 (Contacted Once)', '2-3 (Few Contacts)', '4+ (Many Contacts)']
)


plt.figure(figsize=(12, 6))
sns.countplot(data=marketing_clean, x='previous_binned', hue='y', palette='coolwarm')
plt.title('Distribution of Previous Contacts and Subscription Status')
plt.xlabel('Number of Previous Contacts')
plt.ylabel('Count')
plt.show()
```



Campaign - Binning contacts

```
# Generate binned groups for the 'campaign' variable
marketing_clean['campaign_binned'] = pd.cut(
    marketing_clean['campaign'],
    bins=[0, 1, 3, 5, 10, float('inf')],
    labels=['1 Contact', '2-3 Contacts', '4-5 Contacts', '6-10 Contacts', '11+ Contacts'],
    right=True
)

# Check the distribution of the new binned variable by subscription status
print(marketing_clean.groupby('campaign_binned')['y'].value_counts(normalize=True))

# Plot the distribution of the binned variable with respect to subscription status
plt.figure(figsize=(12, 6))
sns.countplot(data=marketing_clean, x='campaign_binned', hue='y', palette='coolwarm')
plt.title('Distribution of Binned Campaign Groups by Subscription Status')
plt.xlabel('Campaign Contact Groups')
plt.ylabel('Count')
plt.legend(title='Subscription Status (y)', labels=['0 (No)', '1 (Yes)'])
plt.show()
```

```
campaign_binned  y
1 Contact        0    0.869649
                 1    0.130351
2-3 Contacts     0    0.887855
                 1    0.112145
4-5 Contacts     0    0.913176
                 1    0.086824
6-10 Contacts    0    0.936804
                 1    0.063196
11+ Contacts     0    0.968930
                 1    0.031070
Name: y, dtype: float64
```




```
## lets drop the campagin and previous and rename the binned ones

marketing_clean = marketing_clean.drop(['previous','campaign'], axis = 1)
marketing_clean = marketing_clean.rename(columns={'previous_binned': 'previous', 'campaign_binned': 'campaign'})
```

```
marketing_clean.columns
```

```
Index(['job', 'marital', 'education', 'default', 'contact', 'month', 'pdays',
       'poutcome', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y',
       'previous', 'campaign'],
      dtype='object')
```

```
marketing_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41180 entries, 0 to 41179
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   job            41180 non-null  object
 1   marital        41180 non-null  object
 2   education      41180 non-null  object
 3   default        41180 non-null  object
 4   contact        41180 non-null  object
 5   month          41180 non-null  object
 6   pdays          41180 non-null  int64
 7   poutcome       41180 non-null  object
 8   cons.conf.idx  41180 non-null  float64
 9   euribor3m      41180 non-null  float64
 10  nr.employed    41180 non-null  float64
 11  y              41180 non-null  int64
 12  previous       41180 non-null  category
 13  campaign       41180 non-null  category
dtypes: category(2), float64(3), int64(2), object(7)
memory usage: 3.8+ MB
```

Data Transformation

```
marketing_transf = marketing_clean.copy()

y = marketing_transf.pop('y')
X = marketing_transf



X.head()
```

	job	marital	education	default	contact	month	pdays	poutcome	cons.conf.
0	admin.	married	basic.6y	no	telephone	may	999	nonexistent	-36.4
1	services	married	high.school	no	telephone	may	999	nonexistent	-36.4
2	services	married	basic.9y	unknown	telephone	may	999	nonexistent	-36.4
3	admin.	married	professional.course	no	telephone	may	999	nonexistent	-36.4
4	blue-collar	married	unknown	unknown	telephone	may	999	nonexistent	-36.4

Standard Scaling

```
numerical_columns = X.select_dtypes(include=['int64', 'float64']).columns
print("Numerical columns:")
print(numerical_columns)
```

```
Numerical columns:
Index(['pdays', 'cons.conf.idx', 'euribor3m', 'nr.employed'], dtype='object')
```

```
from sklearn.preprocessing import StandardScaler
numerical_columns = ['pdays', 'cons.conf.idx', 'euribor3m', 'nr.employed']

scaler = StandardScaler()
X[numerical_columns] = scaler.fit_transform(X[numerical_columns])

X.head()
```

	job	marital	education	default	contact	month	pdays	poutcome	cons.conf
0	admin.	married	basic.6y	no	telephone	may	0.1953	nonexistent	0.886477
1	services	married	high.school	no	telephone	may	0.1953	nonexistent	0.886477
2	services	married	basic.9y	unknown	telephone	may	0.1953	nonexistent	0.886477
3	admin.	married	professional.course	no	telephone	may	0.1953	nonexistent	0.886477
4	blue-collar	married	unknown	unknown	telephone	may	0.1953	nonexistent	0.886477

```
X.poutcome.value_counts()
```

```
nonexistent    35559
failure         4250
success         1371
Name: poutcome, dtype: int64
```

```
categorical_columns = X.select_dtypes(include=['object','category']).columns
print(categorical_columns)
```

```
Index(['job', 'marital', 'education', 'default', 'contact', 'month',
       'poutcome', 'previous', 'campaign'],
      dtype='object')
```

One Hot Encoding

```
categorical_columns = ['job', 'marital', 'education', 'default', 'contact', 'month', 'previous', 'campaign','poutcome']

X = pd.get_dummies(X, columns=categorical_columns, drop_first=True)


X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41180 entries, 0 to 41179
Data columns (total 46 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   pdays                          41180 non-null  float64
 1   cons.conf.idx                  41180 non-null  float64
 2   euribor3m                      41180 non-null  float64
 3   nr.employed                    41180 non-null  float64
 4   job_blue-collar                41180 non-null  uint8
 5   job_entrepreneur               41180 non-null  uint8
 6   job_housemaid                  41180 non-null  uint8
 7   job_management                 41180 non-null  uint8
 8   job_retired                    41180 non-null  uint8
 9   job_self-employed              41180 non-null  uint8
 10  job_services                   41180 non-null  uint8
 11  job_student                    41180 non-null  uint8
 12  job_technician                 41180 non-null  uint8
 13  job_unemployed                 41180 non-null  uint8
 14  job_unknown                    41180 non-null  uint8
 15  marital_married                41180 non-null  uint8
 16  marital_single                 41180 non-null  uint8
 17  marital_unknown                41180 non-null  uint8
 18  education_basic.6y             41180 non-null  uint8
 19  education_basic.9y             41180 non-null  uint8
 20  education_high.school          41180 non-null  uint8
 21  education_illiterate           41180 non-null  uint8
 22  education_professional.course  41180 non-null  uint8
 23  education_university.degree    41180 non-null  uint8
 24  education_unknown              41180 non-null  uint8
```

```
X.shape
```

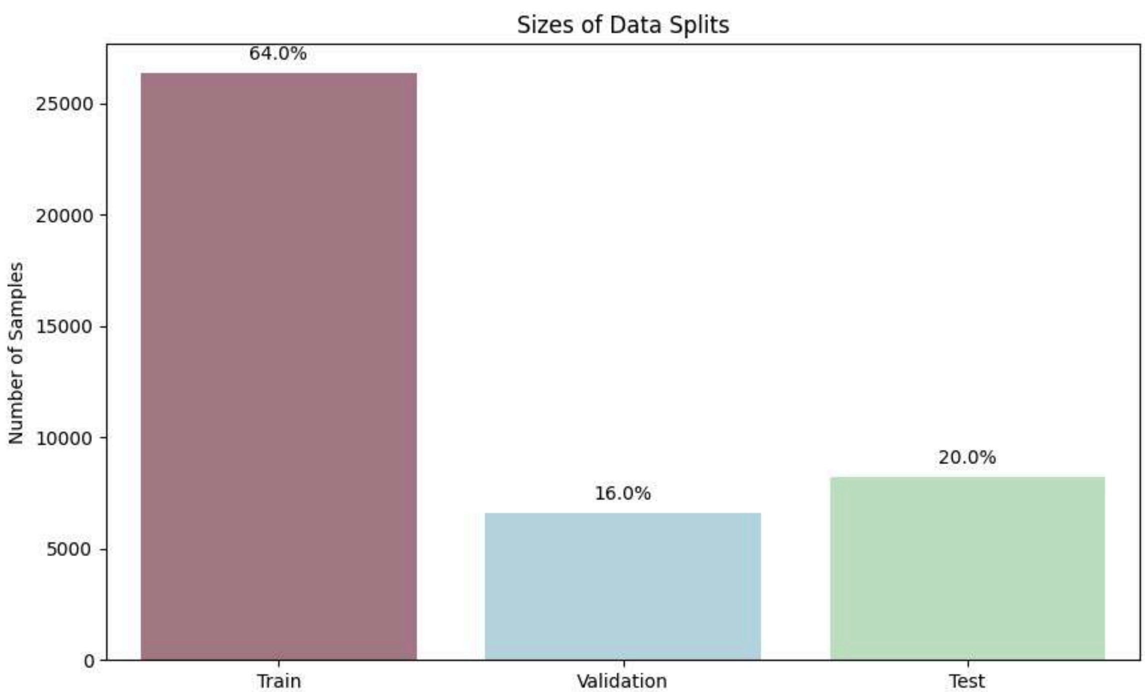
```
(41180, 46)
```

Splitting (Train Validation & Test)

```
from sklearn.model_selection import train_test_split
# Split the data into train and test (80% train, 20% test)
X_test, X_temp, y_test, y_temp = train_test_split(X, y, test_size=0.8, random_state=33, stratify=y)

# Split the temporary set into validation and test (50% of 20% = 10% of original data)
X_val, X_train, y_val, y_train = train_test_split(X_temp, y_temp, test_size=0.8, random_state=33, stratify=y_temp)

# Display the sizes of each split
print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Training set size: 26356
Validation set size: 6588
Test set size: 8236
```

```python
# Define the palette and figure size
palette = ['#ac6f82', '#ADD8E6', '#BAE4BC']
plt.figure(figsize=(10, 6))

# Calculate the sizes and percentages of each split
split_names = ['Train', 'Validation', 'Test']
split_sizes = [X_train.shape[0], X_val.shape[0], X_test.shape[0]]
total_size = sum(split_sizes)
percentages = [size / total_size * 100 for size in split_sizes]

# Create the bar plot
sns.barplot(x=split_names, y=split_sizes, palette=palette)

# Annotate each bar with the percentage
for i, (size, percent) in enumerate(zip(split_sizes, percentages)):
    plt.text(i, size + total_size * 0.01, f'{percent:.1f}%', ha='center', va='bottom')

# Add titles and labels
plt.title('Sizes of Data Splits')
plt.ylabel('Number of Samples')
plt.show()
```



Modeling

Logistic Regression - Statistical Analysis (Stats Model)

```python
import statsmodels.api as sm

X_train_sm = sm.add_constant(X_train)  # Add a constant for intercept


# Fit the logistic regression model using statsmodels
logit_model_sm = sm.Logit(y_train, X_train_sm).fit()

logistic_stats_table = logit_model_sm.summary()
logistic_stats_table
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.275610
         Iterations: 35
```

```
/usr/local/lib/python3.10/site-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed
  warnings.warn("Maximum Likelihood optimization failed to "
```

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	26356
Model:	Logit	**Df Residuals:**	26310
Method:	MLE	**Df Model:**	45
Date:	Sun, 10 Nov 2024	**Pseudo R-squ.:**	0.2169
Time:	13:42:05	**Log-Likelihood:**	-7264.0
converged:	False	**LL-Null:**	-9275.8
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.9468	nan	nan	nan	nan	nan
pdays	-0.2240	0.047	-4.799	0.000	-0.315	-0.132
cons.conf.idx	0.0909	0.025	3.690	0.000	0.043	0.139
euribor3m	-0.0518	0.081	-0.640	0.522	-0.210	0.107
nr.employed	-0.7669	0.072	-10.656	0.000	-0.908	-0.626
job_blue-collar	-0.1164	0.086	-1.349	0.177	-0.286	0.053
job_entrepreneur	0.0180	0.131	0.137	0.891	-0.238	0.274
job_housemaid	-0.1290	0.162	-0.794	0.427	-0.447	0.189
job_management	0.0069	0.092	0.075	0.940	-0.174	0.188
job_retired	0.2885	0.103	2.792	0.005	0.086	0.491
job_self-employed	-0.0452	0.128	-0.354	0.724	-0.295	0.205
job_services	-0.0923	0.093	-0.993	0.321	-0.274	0.090

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score, f1_score, recall_score, make_scorer, precision_score,
```

```
# Define the scoring metrics for GridSearchCV
scoring = {
    'precision': make_scorer(precision_score),
    'f1': make_scorer(f1_score)
}

# Initialize the logistic regression model
log_reg = LogisticRegression(max_iter=1000, class_weight='balanced')

# Define the hyperparameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'fit_intercept': [True, False],
    'penalty': ['l2'],
    'solver': ['lbfgs', 'liblinear', 'saga']  # Add more solvers for evaluation
}

# Configure GridSearchCV
grid_search = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid,
    scoring=scoring,
    refit='precision',
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1,  # Use all available cores
    return_train_score=True  # Ensure training scores are included
)

# Fit the model on the training set
grid_search.fit(X_train, y_train)

# Create a summary DataFrame for the metrics
results = pd.DataFrame(grid_search.cv_results_)

# Check available columns for validation
print(results.columns)

# Extract relevant columns for the table
summary_table = results[['param_C', 'param_fit_intercept', 'mean_train_precision', 'mean_train_f1',
                         'mean_test_precision', 'mean_test_f1']].copy()

# Rename columns for better readability
summary_table.columns = ['C', 'Fit Intercept', 'Precision Train', 'F1 Score Train',
                         'Precision Validation', 'F1 Score Validation']

# Sort the table by Precision Validation (or change to F1 Score if needed)
summary_table = summary_table.sort_values(by='Precision Validation', ascending=False)

# Display the summary table
print(summary_table.head(10))  # Show top 10 models sorted by Precision Validation

# Displaying the confusion matrix for the best model
best_model = grid_search.best_estimator_
y_val_pred = best_model.predict(X_val)

print("Confusion Matrix for Best Model on Validation Set:")
print(confusion_matrix(y_val, y_val_pred))
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=liblinear; total time=   0.5s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=liblinear; total time=   0.5s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=lbfgs; total time=   0.6s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=lbfgs; total time=   0.6s[CV] END C=0.01, fit_intercept=True, penalty=l

[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=lbfgs; total time=   0.6s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=lbfgs; total time=   0.6s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=liblinear; total time=   0.4s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=liblinear; total time=   0.3s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=liblinear; total time=   0.3s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=lbfgs; total time=   0.3s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=lbfgs; total time=   0.3s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=lbfgs; total time=   0.3s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=lbfgs; total time=   0.3s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=liblinear; total time=   0.2s
```

```
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=saga; total time=   1.0s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=saga; total time=   0.9s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=lbfgs; total time=   0.3s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=saga; total time=   1.0s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=saga; total time=   1.0s
[CV] END C=0.01, fit_intercept=True, penalty=l2, solver=saga; total time=   1.0s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=liblinear; total time=   0.3s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=liblinear; total time=   0.4s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=liblinear; total time=   0.4s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=liblinear; total time=   0.4s
[CV] END C=0.01, fit_intercept=False, penalty=l2, solver=saga; total time=   1.1s
[CV] END C=0.1, fit_intercept=True, penalty=l2, solver=lbfgs; total time=   0.8s
[CV] END C=0.1, fit_intercept=True, penalty=l2, solver=lbfgs; total time=   0.8s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END .C=100, fit_intercept=True, penalty=l2, solver=saga; total time=  40.9s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END .C=100, fit_intercept=True, penalty=l2, solver=saga; total time=  40.9s
[CV] END .C=100, fit_intercept=True, penalty=l2, solver=saga; total time=  40.9s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END .C=100, fit_intercept=True, penalty=l2, solver=saga; total time=  41.2s
[CV] END .C=100, fit_intercept=True, penalty=l2, solver=saga; total time=  41.3s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END C=100, fit_intercept=False, penalty=l2, solver=saga; total time=  38.2s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END C=100, fit_intercept=False, penalty=l2, solver=saga; total time=  37.2s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END C=100, fit_intercept=False, penalty=l2, solver=saga; total time=  36.6s
```

```
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which mean
  warnings.warn(
```

```
[CV] END C=100, fit_intercept=False, penalty=l2, solver=saga; total time=  22.9s
[CV] END C=100, fit_intercept=False, penalty=l2, solver=saga; total time=  22.7s
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_C', 'param_fit_intercept', 'param_penalty', 'param_solver',
       'params', 'split0_test_precision', 'split1_test_precision',
       'split2_test_precision', 'split3_test_precision',
       'split4_test_precision', 'mean_test_precision', 'std_test_precision',
       'rank_test_precision', 'split0_train_precision',
       'split1_train_precision', 'split2_train_precision',
       'split3_train_precision', 'split4_train_precision',
       'mean_train_precision', 'std_train_precision', 'split0_test_f1',
       'split1_test_f1', 'split2_test_f1', 'split3_test_f1', 'split4_test_f1',
       'mean_test_f1', 'std_test_f1', 'rank_test_f1', 'split0_train_f1',
```

```
       'split1_train_f1', 'split2_train_f1', 'split3_train_f1',
       'split4_train_f1', 'mean_train_f1', 'std_train_f1'],
      dtype='object')
      C Fit Intercept  Precision Train  F1 Score Train  Precision Validation  \
15    1         False         0.356538        0.456409              0.358123
16    1         False         0.356551        0.456442              0.358123
17    1         False         0.356551        0.456442              0.358123
28  100         False         0.355944        0.456093              0.357681
26  100          True         0.355944        0.456093              0.357681
29  100         False         0.355944        0.456093              0.357681
25  100          True         0.355913        0.456046              0.357632
24  100          True         0.355930        0.456060              0.357632
18   10          True         0.355974        0.456140              0.357553
19   10          True         0.355943        0.456093              0.357553

    F1 Score Validation
```

```python
# Extract relevant columns for the table, including all parameter columns
param_columns = [col for col in results.columns if col.startswith('param_')]
metrics_columns = ['mean_train_precision', 'mean_train_f1', 'mean_test_precision', 'mean_test_f1']

# Combine parameter columns and metrics columns
summary_table = results[param_columns + metrics_columns].copy()

# Rename metric columns for readability
summary_table.columns = [col.replace('param_', '').title() for col in param_columns] + [
    'Precision Train', 'F1 Score Train', 'Precision Validation', 'F1 Score Validation'
]

# Sort the table by Precision Validation (or change to F1 Score if needed)
summary_table = summary_table.sort_values(by='Precision Validation', ascending=False)
summary_table
```

	C	Fit_Intercept	Penalty	Solver	Precision Train	F1 Score Train	Precision Validation	F1 !
15	1	False	l2	lbfgs	0.356538	0.456409	0.358123	0.4!
16	1	False	l2	liblinear	0.356551	0.456442	0.358123	0.4!
17	1	False	l2	saga	0.356551	0.456442	0.358123	0.4!
28	100	False	l2	liblinear	0.355944	0.456093	0.357681	0.4!
26	100	True	l2	saga	0.355944	0.456093	0.357681	0.4!
29	100	False	l2	saga	0.355944	0.456093	0.357681	0.4!
25	100	True	l2	liblinear	0.355913	0.456046	0.357632	0.4!
24	100	True	l2	lbfgs	0.355930	0.456060	0.357632	0.4!
18	10	True	l2	lbfgs	0.355974	0.456140	0.357553	0.4!
19	10	True	l2	liblinear	0.355943	0.456093	0.357553	0.4!
20	10	True	l2	saga	0.355943	0.456093	0.357553	0.4!
27	100	False	l2	lbfgs	0.355946	0.456074	0.357507	0.4!
12	1	True	l2	lbfgs	0.355649	0.455807	0.357345	0.4!
22	10	False	l2	liblinear	0.355956	0.456126	0.357290	0.4!
23	10	False	l2	saga	0.355891	0.456050	0.357290	0.4!
21	10	False	l2	lbfgs	0.355891	0.456050	0.357290	0.4!
13	1	True	l2	liblinear	0.355900	0.455993	0.357288	0.4!
14	1	True	l2	saga	0.355618	0.455760	0.357271	0.4!
9	0.1	False	l2	lbfgs	0.354797	0.455020	0.356553	0.4!

Model Performance Logistic Regression

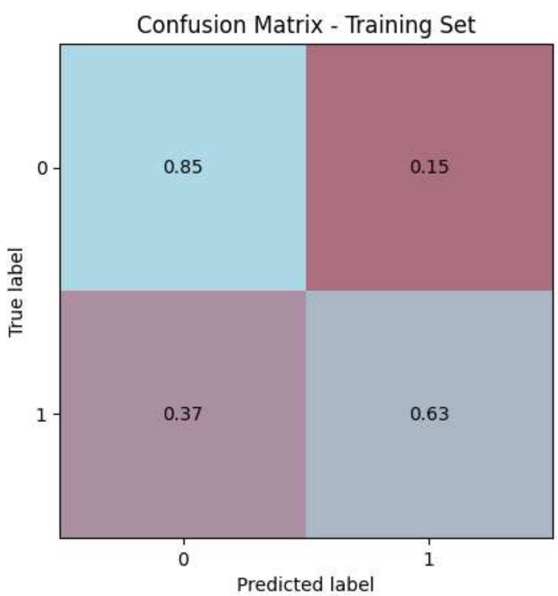
```
print(grid_search.best_params_)
```

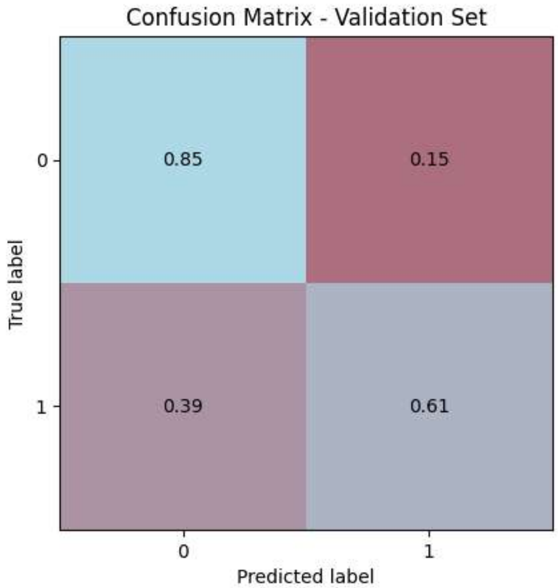
```
{'C': 1, 'fit_intercept': False, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
# model performance
best_model = grid_search.best_estimator_
y_train_preds = best_model.predict(X_train)
y_val_preds = best_model.predict(X_val)

#Display the f1_score and precision for best model on training
print(f'f1_score on training data: {f1_score(y_train, y_train_preds)}')
print(f'precision on training data: {precision_score(y_train, y_train_preds)}')

#Display the f1_score and precision for best model on validation
print(f'f1_score on validation data: {f1_score(y_val, y_val_preds)}')
print(f'precision on validition data: {precision_score(y_val, y_val_preds)}')
```

```
f1_score on training data: 0.45549357264128065
precision on training data: 0.3558165971959075
f1_score on validation data: 0.43062200956937796
precision on validition data: 0.33382789317507416
```

```python
from matplotlib.colors import LinearSegmentedColormap
from sklearn.metrics import ConfusionMatrixDisplay

# Create a custom colormap using the given palette
palette = ['#ac6f82', '#ADD8E6']
custom_cmap = LinearSegmentedColormap.from_list('custom_cmap', palette, N=256)

# Function to plot confusion matrix with black text annotations
def plot_confusion_matrix_with_black_text(estimator, X, y, title, cmap):
    disp = ConfusionMatrixDisplay.from_estimator(
        estimator,
        X,
        y,
        normalize='true',
        cmap=cmap,
        colorbar=False  # Disable the default colorbar
    )
    plt.title(title)
    # Change text color to black for better visibility
    for text in disp.text_.ravel():
        text.set_color('black')
    plt.show()

# Plot confusion matrix for the training set
plot_confusion_matrix_with_black_text(best_model, X_train, y_train, 'Confusion Matrix - Training Set', custom_cmap)

# Plot confusion matrix for the validation set
plot_confusion_matrix_with_black_text(best_model, X_val, y_val, 'Confusion Matrix - Validation Set', custom_cmap)
```






```python
# Ensure that the best model is extracted
best_model = grid_search.best_estimator_

# Get the feature names from the training set (X_train)
feature_names = X_train.columns

# Extract the coefficients from the model
coefficients = best_model.coef_[0]  # Assuming binary classification with one set of coefficients

# Create a DataFrame to display the coefficients with their respective features
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients,
    'Absolute Coefficient': np.abs(coefficients)  # To show the magnitude
})

# Sort the DataFrame by the absolute value of coefficients for better visualization
coef_df = coef_df.sort_values(by='Absolute Coefficient', ascending=False)

# Display the top features based on the absolute value of their coefficients
print(coef_df.head(15))  # Display the top 15 features

# Analyze the intercept
intercept = best_model.intercept_[0]
print(f"\nIntercept of the model: {intercept}")
```

```
                      Feature  Coefficient  Absolute Coefficient
43        campaign_11+ Contacts    -1.414491              1.414491
32                    month_mar     1.052882              1.052882
39  previous_4+ (Many Contacts)    -0.906284              0.906284
38  previous_2-3 (Few Contacts)    -0.809605              0.809605
3                   nr.employed    -0.755242              0.755242
33                    month_may    -0.716138              0.716138
45             poutcome_success     0.533380              0.533380
37  previous_1 (Contacted Once)    -0.510510              0.510510
35                    month_oct     0.477635              0.477635
36                    month_sep    -0.465681              0.465681
21          education_illiterate     0.420712              0.420712
17               marital_unknown     0.420168              0.420168
8                   job_retired     0.370194              0.370194
27            contact_telephone    -0.318159              0.318159
34                    month_nov    -0.293195              0.293195

Intercept of the model: 0.0
```

Random Forrest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score, f1_score, recall_score, make_scorer
import pandas as pd

# Define the scoring metrics for GridSearchCV
scoring = {
    'precision': make_scorer(precision_score),
    'f1': make_scorer(f1_score)
}

# Initialize the Random Forest model
rf_model = RandomForestClassifier(class_weight='balanced', random_state=42)

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5,10, 15],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [2, 4],
    'bootstrap': [True]
}

# Configure GridSearchCV
grid_search_rf = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    scoring=scoring,
    refit='precision',
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1,  # Use all available cores
    return_train_score=True  # Ensure training scores are included
)

# Fit the model on the training set
grid_search_rf.fit(X_train, y_train)

# Create a summary DataFrame for the metrics
results_rf = pd.DataFrame(grid_search_rf.cv_results_)

# Check available columns for validation
print(results_rf.columns)

# Extract relevant columns for the table
param_columns = [col for col in results_rf.columns if col.startswith('param_')]
metrics_columns = ['mean_train_precision', 'mean_train_f1', 'mean_test_precision', 'mean_test_f1']

# Combine parameter columns and metrics columns
summary_table_rf = results_rf[param_columns + metrics_columns].copy()

# Rename metric columns for readability
summary_table_rf.columns = [col.replace('param_', '').title() for col in param_columns] + [
    'Precision Train', 'F1 Score Train', 'Precision Validation', 'F1 Score Validation'
]

# Sort the table by Precision Validation (or change to F1 Score if needed)
summary_table_rf = summary_table_rf.sort_values(by='Precision Validation', ascending=False)

# Display the summary table
print(summary_table_rf.head(10))  # Show top 10 models sorted by Precision Validation

# Displaying the confusion matrix for the best model
best_model = grid_search_rf.best_estimator_
y_val_pred = best_model.predict(X_val)

print("Confusion Matrix for Best Model on Validation Set:")
print(confusion_matrix(y_val, y_val_pred))

# Feature importance analysis
def display_feature_importance(model, feature_names):
    """
    Display the feature importance from the trained Random Forest model.

    Parameters:
    - model: Trained RandomForestClassifier.
    - feature_names: List of feature names.

    Returns:
```

```
    - DataFrame of feature importances sorted by importance.
    """
    feature_importances = pd.DataFrame({
        'Feature': feature_names,
        'Importance': model.feature_importances_
    })
    feature_importances = feature_importances.sort_values(by='Importance', ascending=False)
    print("\nFeature Importances:")
    print(feature_importances.head(15))
    return feature_importances

# Call the function to display feature importances
feature_importances_rf = display_feature_importance(best_model, X_train.columns)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=   2.9s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=   2.9s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=   2.9s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=   3.1s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=200; total time=   3.1s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=   4.6s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=   4.6s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=   4.6s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   1.7s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   1.7s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   1.8s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=   5.1s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=5, n_estimators=300; total time=   5.1s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   2.0s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   3.8s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   3.8s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   3.9s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   3.8s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   3.8s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   2.2s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   2.2s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   2.1s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time=   6.1s
```

```
summary_table_rf['Precision Gap'] = summary_table_rf['Precision Train'] - summary_table_rf['Precision Validation']
summary_table_rf = summary_table_rf.sort_values(by='Precision Gap', ascending=True)
summary_table_rf
```

	Bootstrap	Max_Depth	Min_Samples_Leaf	Min_Samples_Split	N_Estimators	Precision Train	F1 Score Tra
0	True	5	2	5	100	0.370864	0.469239
2	True	5	2	5	300	0.372854	0.470307
1	True	5	2	5	200	0.372171	0.469490
11	True	5	4	10	300	0.370524	0.468610
5	True	5	2	10	300	0.370854	0.468897
10	True	5	4	10	200	0.370158	0.468713
3	True	5	2	10	100	0.369032	0.468040
6	True	5	4	5	100	0.365508	0.465978
7	True	5	4	5	200	0.370771	0.469430
9	True	5	4	10	100	0.364914	0.465592
8	True	5	4	5	300	0.369606	0.468851
4	True	5	2	10	200	0.370808	0.469152
18	True	10	4	5	100	0.401390	0.494112
20	True	10	4	5	300	0.401529	0.494840
19	True	10	4	5	200	0.401307	0.494471
21	True	10	4	10	100	0.404160	0.496491
16	True	10	2	10	200	0.403892	0.496377
22	True	10	4	10	200	0.402129	0.495050
15	True	10	2	10	100	0.404637	0.497540

```
first_table_rf = summary_table.copy()
first_table_rf.to_csv('first_table_rf.csv', index=False)
```

```
# Initialize the Random Forest model
rf_model = RandomForestClassifier(class_weight='balanced', random_state=42)

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100],
    'max_depth': [5],
    'min_samples_split': [4],
    'min_samples_leaf': [2],
    'bootstrap': [True]
}

# Configure GridSearchCV
grid_search_rf = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    scoring=scoring,
    refit='precision',
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1,  # Use all available cores
    return_train_score=True  # Ensure training scores are included
)

# Fit the model on the training set
grid_search_rf.fit(X_train, y_train)

# Create a summary DataFrame for the metrics
results_rf = pd.DataFrame(grid_search_rf.cv_results_)

# Check available columns for validation
print(results_rf.columns)

# Extract relevant columns for the table
param_columns = [col for col in results_rf.columns if col.startswith('param_')]
metrics_columns = ['mean_train_precision', 'mean_train_f1', 'mean_test_precision', 'mean_test_f1']

# Combine parameter columns and metrics columns
summary_table_rf = results_rf[param_columns + metrics_columns].copy()

# Rename metric columns for readability
summary_table_rf.columns = [col.replace('param_', '').title() for col in param_columns] + [
    'Precision Train', 'F1 Score Train', 'Precision Validation', 'F1 Score Validation'
]

# Sort the table by Precision Validation (or change to F1 Score if needed)
summary_table_rf = summary_table_rf.sort_values(by='Precision Validation', ascending=False)

# Display the summary table
print(summary_table_rf.head(10))  # Show top 10 models sorted by Precision Validation

# Displaying the confusion matrix for the best model
best_model = grid_search_rf.best_estimator_
y_val_pred = best_model.predict(X_val)

print("Confusion Matrix for Best Model on Validation Set:")
print(confusion_matrix(y_val, y_val_pred))

# Feature importance analysis
def display_feature_importance(model, feature_names):
    """
    Display the feature importance from the trained Random Forest model.

    Parameters:
    - model: Trained RandomForestClassifier.
    - feature_names: List of feature names.

    Returns:
    - DataFrame of feature importances sorted by importance.
    """
    feature_importances = pd.DataFrame({
        'Feature': feature_names,
        'Importance': model.feature_importances_
    })
    feature_importances = feature_importances.sort_values(by='Importance', ascending=False)
    print("\nFeature Importances:")
    print(feature_importances.head(15))
    return feature_importances
```

```
# Call the function to display feature importances
feature_importances_rf = display_feature_importance(best_model, X_train.columns)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=4, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=4, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=4, n_estimators=100; total time=   1.6s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=4, n_estimators=100; total time=   1.5s
[CV] END bootstrap=True, max_depth=5, min_samples_leaf=2, min_samples_split=4, n_estimators=100; total time=   1.5s
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_bootstrap', 'param_max_depth', 'param_min_samples_leaf',
       'param_min_samples_split', 'param_n_estimators', 'params',
       'split0_test_precision', 'split1_test_precision',
       'split2_test_precision', 'split3_test_precision',
       'split4_test_precision', 'mean_test_precision', 'std_test_precision',
       'rank_test_precision', 'split0_train_precision',
       'split1_train_precision', 'split2_train_precision',
       'split3_train_precision', 'split4_train_precision',
       'mean_train_precision', 'std_train_precision', 'split0_test_f1',
       'split1_test_f1', 'split2_test_f1', 'split3_test_f1', 'split4_test_f1',
       'mean_test_f1', 'std_test_f1', 'rank_test_f1', 'split0_train_f1',
       'split1_train_f1', 'split2_train_f1', 'split3_train_f1',
       'split4_train_f1', 'mean_train_f1', 'std_train_f1'],
      dtype='object')
  Bootstrap Max_Depth Min_Samples_Leaf Min_Samples_Split N_Estimators  \
0      True         5                2                 4          100

   Precision Train  F1 Score Train  Precision Validation  F1 Score Validation
0         0.368798        0.467965              0.366774             0.464466
Confusion Matrix for Best Model on Validation Set:
[[5071  775]
 [ 295  447]]
```

```
summary_table_rf['Precision Gap'] = summary_table_rf['Precision Train'] - summary_table_rf['Precision Validation']
summary_table_rf = summary_table_rf.sort_values(by='Precision Gap', ascending=True)
summary_table_rf
```

	Bootstrap	Max_Depth	Min_Samples_Leaf	Min_Samples_Split	N_Estimators	Precision Train	F1 Score Train
0	True	5	2	4	100	0.368798	0.467965

Model Technical Performance

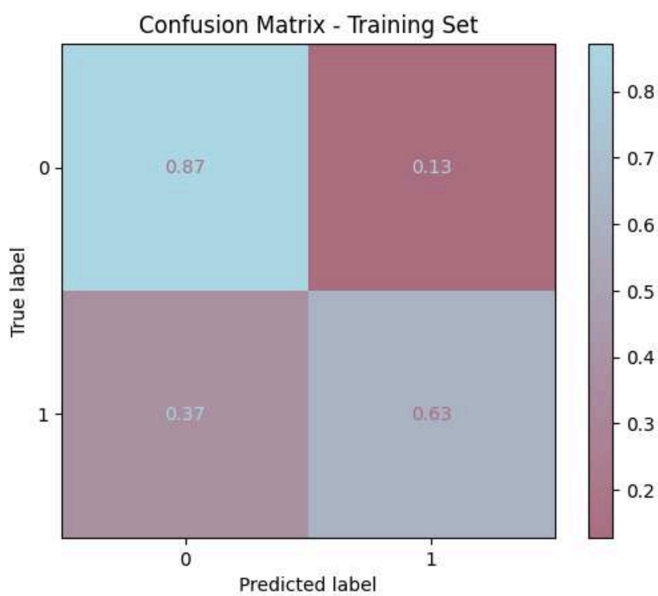
```
best_params = grid_search_rf.best_params_
print(best_params)
```

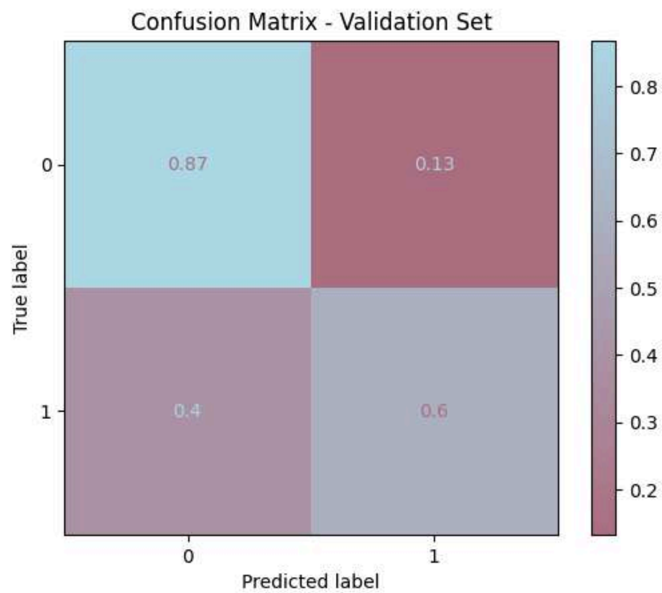
```
{'bootstrap': True, 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 100}
```

```
# model performance
best_model = grid_search_rf.best_estimator_
y_train_preds = best_model.predict(X_train)
y_val_preds = best_model.predict(X_val)

#Display the f1_score and precision for best model on training
print(f'f1_score on training data: {f1_score(y_train, y_train_preds)}')
print(f'precision on training data: {precision_score(y_train, y_train_preds)}')

#Display the f1_score and precision for best model on validation
print(f'f1_score on validation data: {f1_score(y_val, y_val_preds)}')
print(f'precision on validition data: {precision_score(y_val, y_val_preds)}')
```

```
f1_score on training data: 0.476239011339024
precision on training data: 0.3829133374308543
f1_score on validation data: 0.45519348268839105
precision on validition data: 0.3657937806873977
```

```python
from matplotlib.colors import LinearSegmentedColormap

# Create a custom colormap using the given palette
## lets use palle
palette = ['#ac6f82', '#ADD8E6']

custom_cmap = LinearSegmentedColormap.from_list('custom_cmap', palette, N=256)

# Plot confusion matrix for the training set
ConfusionMatrixDisplay.from_estimator(
    best_model,
    X_train,
    y_train,
    normalize='true',
    cmap=custom_cmap
)
plt.title('Confusion Matrix - Training Set')
plt.show()

# Plot confusion matrix for the validation set
ConfusionMatrixDisplay.from_estimator(
    best_model,
    X_val,
    y_val,
    normalize='true',
    cmap=custom_cmap
)
plt.title('Confusion Matrix - Validation Set')
plt.show()
```





Model Comparison

```
best_model_rf = grid_search_rf.best_estimator_ ## this is the random forrest one
best_model = grid_search.best_estimator_ ## this is the logistic one
```

```python
# Import necessary libraries
from sklearn.model_selection import cross_val_score, StratifiedKFold
from scipy.stats import mannwhitneyu
import numpy as np

# Define cross-validation strategy
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Generate cross-validation scores for precision
precision_scores_rf = cross_val_score(best_model_rf, X_val, y_val, cv=cv, scoring=make_scorer(precision_score, average='b
precision_scores_logistic = cross_val_score(best_model, X_val, y_val, cv=cv, scoring=make_scorer(precision_score, average

# Generate cross-validation scores for F1
f1_scores_rf = cross_val_score(best_model_rf, X_val, y_val, cv=cv, scoring=make_scorer(f1_score, average='binary'))
f1_scores_logistic = cross_val_score(best_model, X_val, y_val, cv=cv, scoring=make_scorer(f1_score, average='binary'))

# Perform the Mann-Whitney U test for Precision
precision_stat, precision_p_value = mannwhitneyu(precision_scores_rf, precision_scores_logistic, alternative='two-sided')

# Perform the Mann-Whitney U test for F1 Score
f1_stat, f1_p_value = mannwhitneyu(f1_scores_rf, f1_scores_logistic, alternative='two-sided')

# Print the results
print("Mann-Whitney U Test Results:")
print(f"Precision - U statistic: {precision_stat}, p-value: {precision_p_value}")
print(f"F1 Score - U statistic: {f1_stat}, p-value: {f1_p_value}")

# Interpretation
if precision_p_value < 0.05:
    print("The Precision scores of the two models are significantly different (p < 0.05).")
else:
    print("The Precision scores of the two models are not significantly different (p >= 0.05).")

if f1_p_value < 0.05:
    print("The F1 scores of the two models are significantly different (p < 0.05).")
else:
    print("The F1 scores of the two models are not significantly different (p >= 0.05).")
```

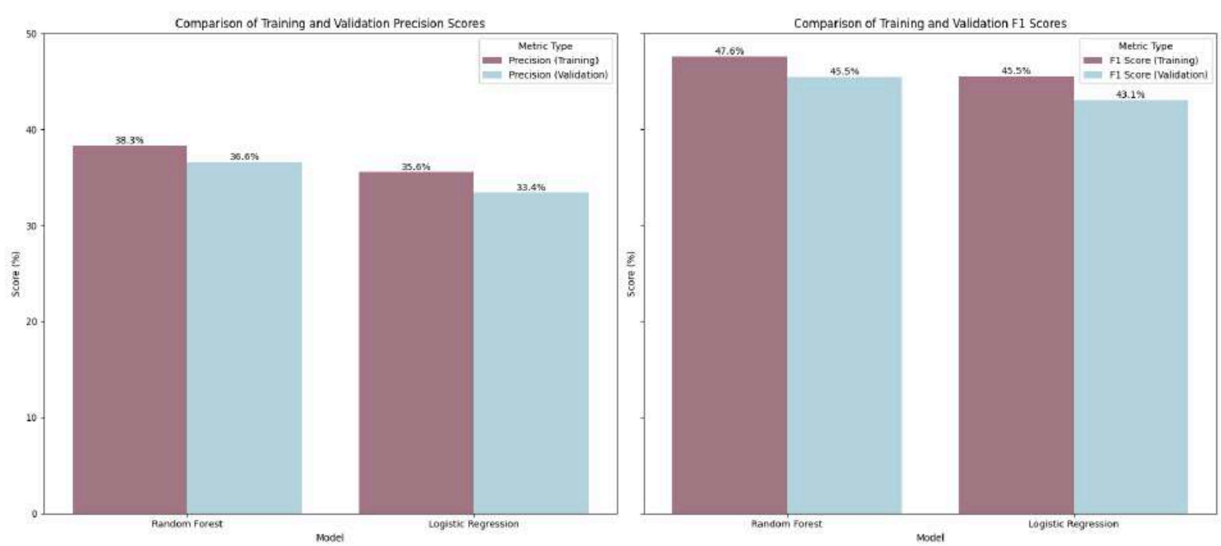
```
Mann-Whitney U Test Results:
Precision - U statistic: 80.0, p-value: 0.025748080821108063
F1 Score - U statistic: 69.0, p-value: 0.16197241048012612
The Precision scores of the two models are significantly different (p < 0.05).
The F1 scores of the two models are not significantly different (p >= 0.05).
```

Because we are based on the premise that the Recall is our most relevant parameter and the statistical Man Whitney U Test gives a p value of 0.025 which is lower than alpha of 0.05, then we reject the null hypothesis which means the two models are sigfnificantly different. In this sense we need to select our model based on the Precision

```
# Model performance for best Random Forest model
best_model_rf = grid_search_rf.best_estimator_
y_train_preds_rf = best_model_rf.predict(X_train)
y_val_preds_rf = best_model_rf.predict(X_val)

# Model performance for best Logistic Regression model
best_model_logistic = grid_search.best_estimator_
y_train_preds_logistic = best_model_logistic.predict(X_train)
y_val_preds_logistic = best_model_logistic.predict(X_val)

# Create a DataFrame to display the F1 Score and Precision for both models
data = {
    'Model': ['Random Forest', 'Logistic Regression'],
    'F1 Score (Training)': [
        f1_score(y_train, y_train_preds_rf),
        f1_score(y_train, y_train_preds_logistic)
    ],
    'Precision (Training)': [
        precision_score(y_train, y_train_preds_rf),
        precision_score(y_train, y_train_preds_logistic)
    ],
    'F1 Score (Validation)': [
        f1_score(y_val, y_val_preds_rf),
        f1_score(y_val, y_val_preds_logistic)
    ],
    'Precision (Validation)': [
        precision_score(y_val, y_val_preds_rf),
        precision_score(y_val, y_val_preds_logistic)
    ]
}

results_df = pd.DataFrame(data)

# Calculate the gaps between training and validation for F1 Score and Precision
results_df['F1 Score Gap'] = results_df['F1 Score (Training)'] - results_df['F1 Score (Validation)']
results_df['Precision Gap'] = results_df['Precision (Training)'] - results_df['Precision (Validation)']

# Display the table
results_df
```

	Model	F1 Score (Training)	Precision (Training)	F1 Score (Validation)	Precision (Validation)
0	Random Forest	0.476239	0.382913	0.455193	0.365794
1	Logistic Regression	0.455494	0.355817	0.430622	0.333828

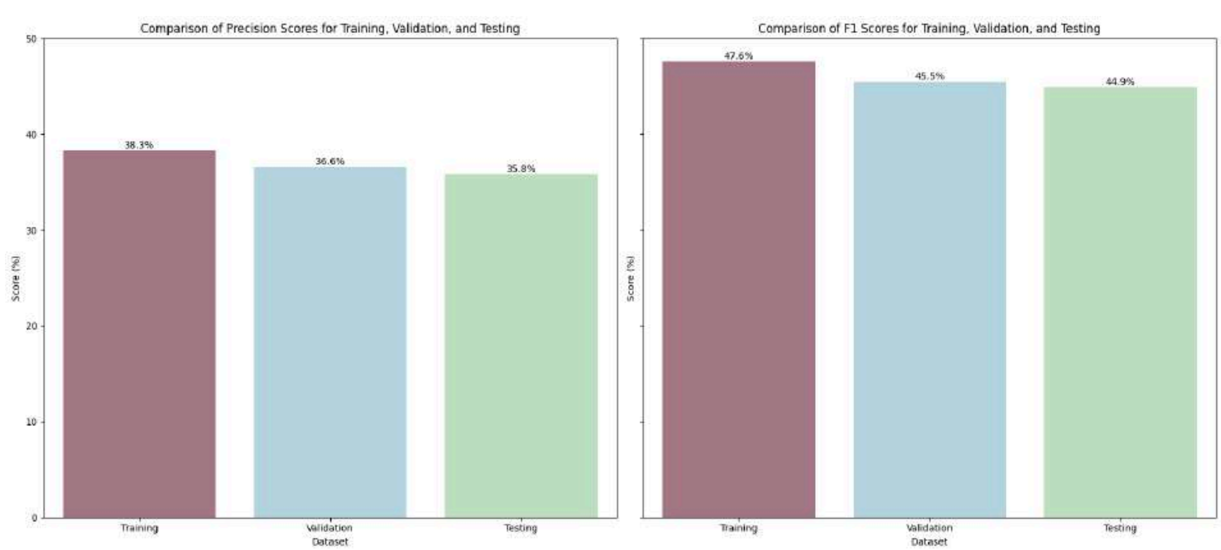
```python
# Convert scores to percentages
results_df[['F1 Score (Training)', 'Precision (Training)', 'F1 Score (Validation)', 'Precision (Validation)']] *= 100

# Plotting the results with subplots for Precision and F1 Score
fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey=True)
palette = ['#ac6f82', '#ADD8E6']

# Plot Precision scores
precision_df = results_df.melt(id_vars=['Model'], value_vars=['Precision (Training)', 'Precision (Validation)'],
                               var_name='Metric Type', value_name='Score')
sns.barplot(x='Model', y='Score', hue='Metric Type', data=precision_df, ax=axes[0], palette=palette)
axes[0].set_title("Comparison of Training and Validation Precision Scores")
axes[0].set_ylabel("Score (%)")

# Annotate the bars with values
for p in axes[0].patches:
    axes[0].annotate(f'{p.get_height():.1f}%',
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)

# Plot F1 scores
f1_df = results_df.melt(id_vars=['Model'], value_vars=['F1 Score (Training)', 'F1 Score (Validation)'],
                        var_name='Metric Type', value_name='Score')
sns.barplot(x='Model', y='Score', hue='Metric Type', data=f1_df, ax=axes[1], palette=palette)
axes[1].set_title("Comparison of Training and Validation F1 Scores")
axes[1].set_ylabel("Score (%)")

# Annotate the bars with values
for p in axes[1].patches:
    axes[1].annotate(f'{p.get_height():.1f}%',
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()
```



Based on the precision I select Random Forrest to be the best model of the two and the one that can be selected if we want to predict.

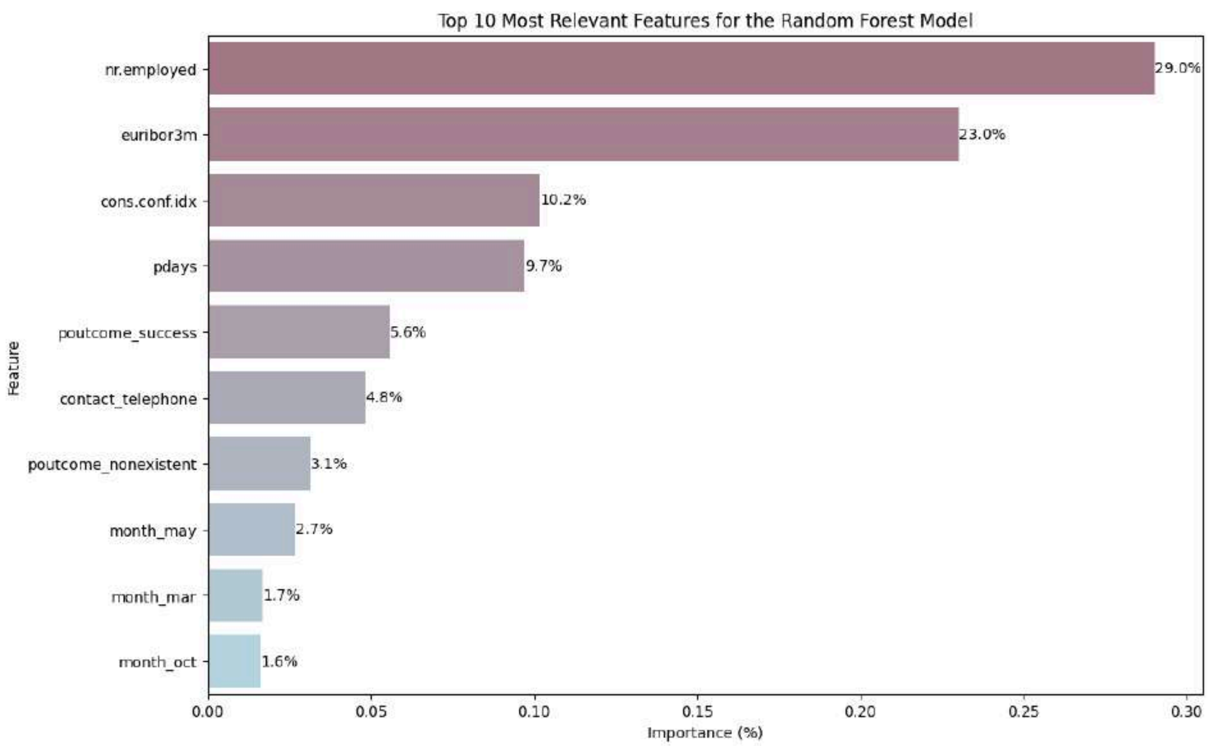
Asssessing the Best Model on Testing

```python
# Model performance for best Random Forest model
best_model_rf = grid_search_rf.best_estimator_
y_train_preds_rf = best_model_rf.predict(X_train)
y_val_preds_rf = best_model_rf.predict(X_val)
y_test_preds_rf = best_model_rf.predict(X_test)

# Create a DataFrame to display the F1 Score and Precision for the Random Forest model
data = {
    'Dataset': ['Training', 'Validation', 'Testing'],
    'F1 Score': [
        f1_score(y_train, y_train_preds_rf) * 100,
        f1_score(y_val, y_val_preds_rf) * 100,
        f1_score(y_test, y_test_preds_rf) * 100
    ],
    'Precision': [
        precision_score(y_train, y_train_preds_rf) * 100,
        precision_score(y_val, y_val_preds_rf) * 100,
        precision_score(y_test, y_test_preds_rf) * 100
    ]
}

results_df = pd.DataFrame(data)

# Plotting the results
palette = ['#ac6f82', '#ADD8E6', '#BAE4BC']
fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey=True)

# Plot Precision scores
sns.barplot(x='Dataset', y='Precision', data=results_df, ax=axes[0], palette=palette)
axes[0].set_title("Comparison of Precision Scores for Training, Validation, and Testing")
axes[0].set_ylabel("Score (%)")

# Annotate the bars with values
for p in axes[0].patches:
    axes[0].annotate(f'{p.get_height():.1f}%',
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)

# Plot F1 scores
sns.barplot(x='Dataset', y='F1 Score', data=results_df, ax=axes[1], palette=palette)
axes[1].set_title("Comparison of F1 Scores for Training, Validation, and Testing")
axes[1].set_ylabel("Score (%)")

# Annotate the bars with values
for p in axes[1].patches:
    axes[1].annotate(f'{p.get_height():.1f}%',
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()
```

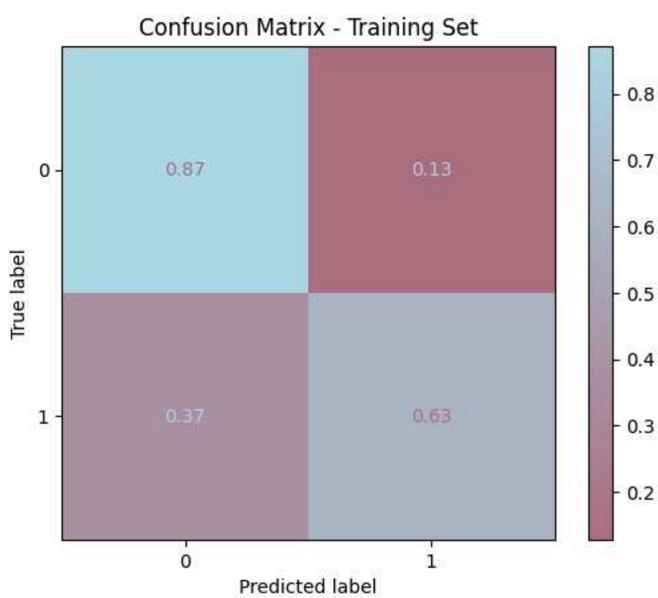


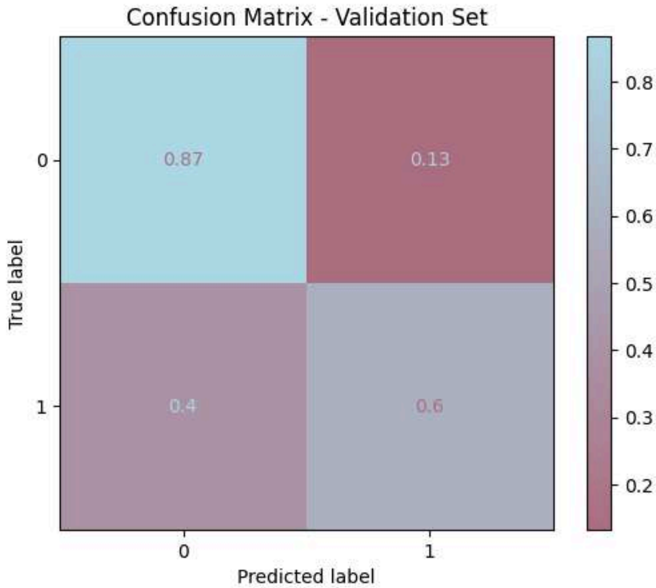
Feature Importance

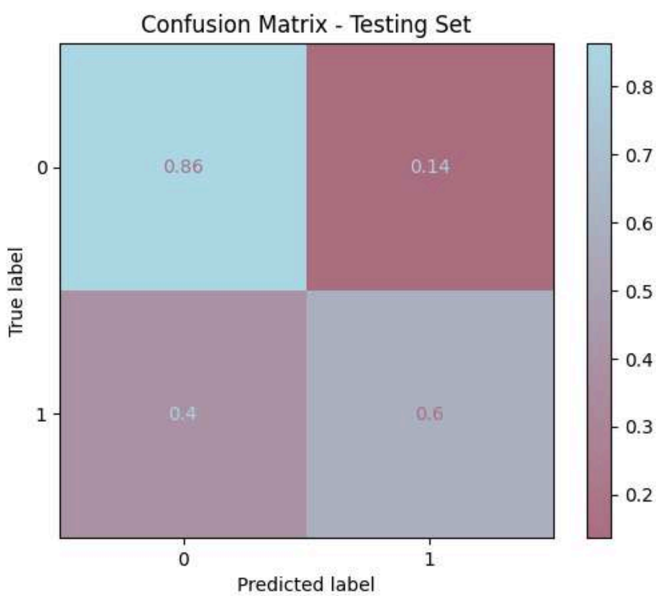
```
# Plot feature importance for the Random Forest model
feature_importances = best_model_rf.feature_importances_
features = X_train.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False).head(10)  # Top 10 most important features

# Generate a gradient color palette using the same blue and red shades
gradient_palette = sns.blend_palette(['#ac6f82', '#ADD8E6'], n_colors=10, as_cmap=False)

# Plotting the feature importances with custom gradient colors
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette=gradient_palette)
plt.title("Top 10 Most Relevant Features for the Random Forest Model")
plt.xlabel("Importance (%)")
plt.ylabel("Feature")

# Annotate the bars with values
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_width() * 100:.1f}%',
                       (p.get_width(), p.get_y() + p.get_height() / 2.),
                       ha='left', va='center', fontsize=10)

plt.show()
```



```python
# Create a custom colormap using the given palette
palette = ['#ac6f82', '#ADD8E6']
custom_cmap = LinearSegmentedColormap.from_list('custom_cmap', palette, N=256)

# Plot confusion matrix for the training set
ConfusionMatrixDisplay.from_estimator(
    best_model_rf,  # Using the Random Forest model
    X_train,
    y_train,
    normalize='true',
    cmap=custom_cmap
)
plt.title('Confusion Matrix - Training Set')
plt.show()

# Plot confusion matrix for the validation set
ConfusionMatrixDisplay.from_estimator(
    best_model_rf,  # Using the Random Forest model
    X_val,
    y_val,
    normalize='true',
    cmap=custom_cmap
)
plt.title('Confusion Matrix - Validation Set')
plt.show()

# Plot confusion matrix for the testing set
ConfusionMatrixDisplay.from_estimator(
    best_model_rf,  # Using the Random Forest model
    X_test,
    y_test,
    normalize='true',
    cmap=custom_cmap
)
plt.title('Confusion Matrix - Testing Set')
plt.show()
```




Confusion Matrix - Validation Set
True label
Predicted label
0
1
0.87
0.13
0.4
0.6
0.8
0.7
0.6
0.5
0.4
0.3
0.2


Confusion Matrix - Testing Set
True label
Predicted label
0
1
0.86
0.14
0.4
0.6
0.8
0.7
0.6
0.5
0.4
0.3
0.2