1. Load and Import Libraries

[ ] ↳ *1 cell hidden*

2. Load the Dataset

```
import pandas as pd
data = pd.read_csv("https://raw.githubusercontent.com/vihanga-induwara/Bank-Marketing/mai
```

3. Explore the Dataset

Inspect the first few rows with .head().

```
data.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_o
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	

5 rows × 21 columns

Check for missing values using .isnull().sum()

```
data.isnull().sum()
```

	0
age	0
job	0
marital	0
education	0
default	0
housing	0
loan	0
contact	0
month	0
day_of_week	0
duration	0
campaign	0
pdays	0
previous	0
poutcome	0
emp.var.rate	0
cons.price.idx	0
cons.conf.idx	0
euribor3m	0
nr.employed	0
y	0

dtype: int64

Check for class imbalance using .value_counts() on the target variable.

```
data['y'].value_counts()
```

	count
y	
no	36548
yes	4640

dtype: int64

Use .describe() for statistical overview of the dataset.

```
data.describe()
```

	age	duration	campaign	pdays	previous	emp.var.rat
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.00000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.08188
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.57096
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.40000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.80000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.10000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.40000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.40000

Examine feature types (categorical, continuous) using .dtypes.

```
data.dtypes
```

	0
age	int64
job	object
marital	object
education	object
default	object
housing	object
loan	object
contact	object
month	object
day_of_week	object
duration	int64
campaign	int64
pdays	int64
previous	int64
poutcome	object
emp.var.rate	float64
cons.price.idx	float64
cons.conf.idx	float64
euribor3m	float64
nr.employed	float64
y	object

dtype: object

4. Data Preprocessing

4.1 Each colum Data Preprocessing

01.age - Client's age (numeric)

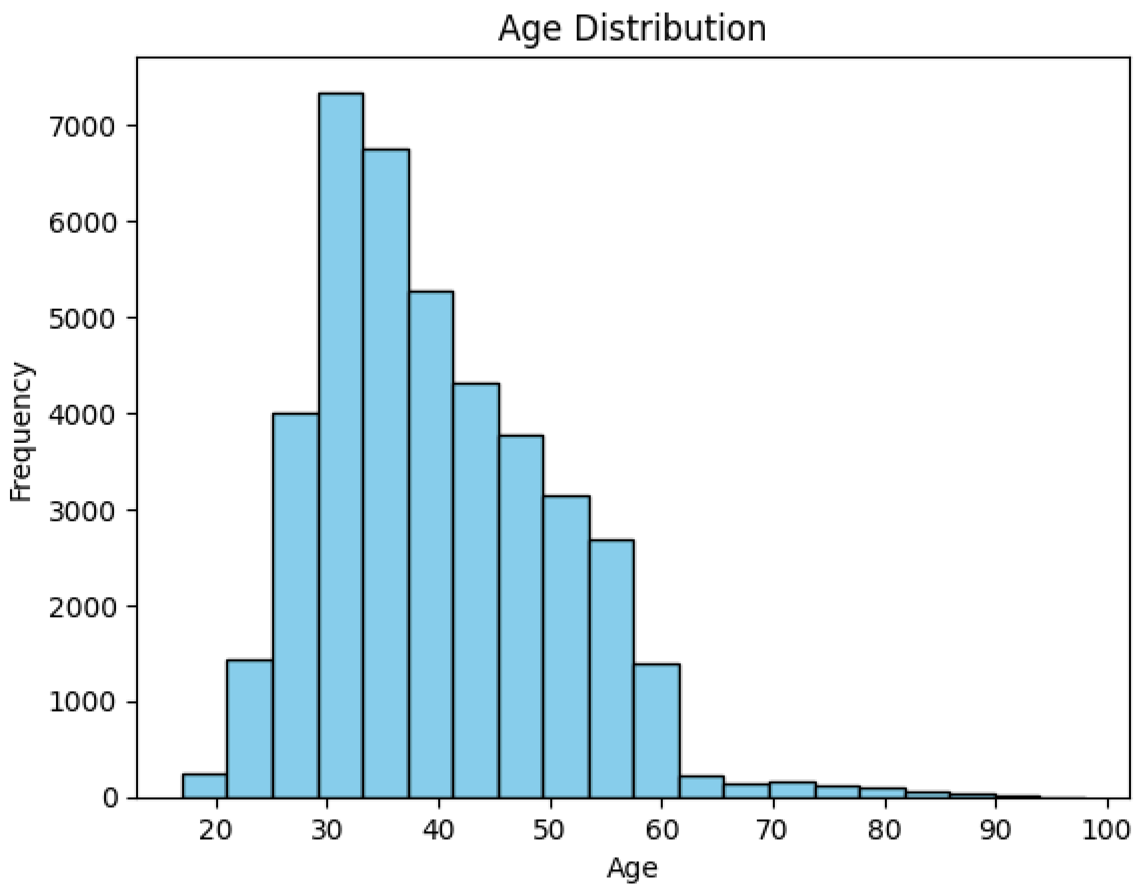
```
data["age"].value_counts()
```

age	count
31	1947
32	1846
33	1833
36	1780
35	1759
...	...
89	2
91	2
94	1
87	1
95	1

78 rows × 1 columns

dtype: int64

```
import matplotlib.pyplot as plt

# Basic histogram for age distribution
plt.hist(data["age"], bins=20, color="skyblue", edgecolor="black")
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

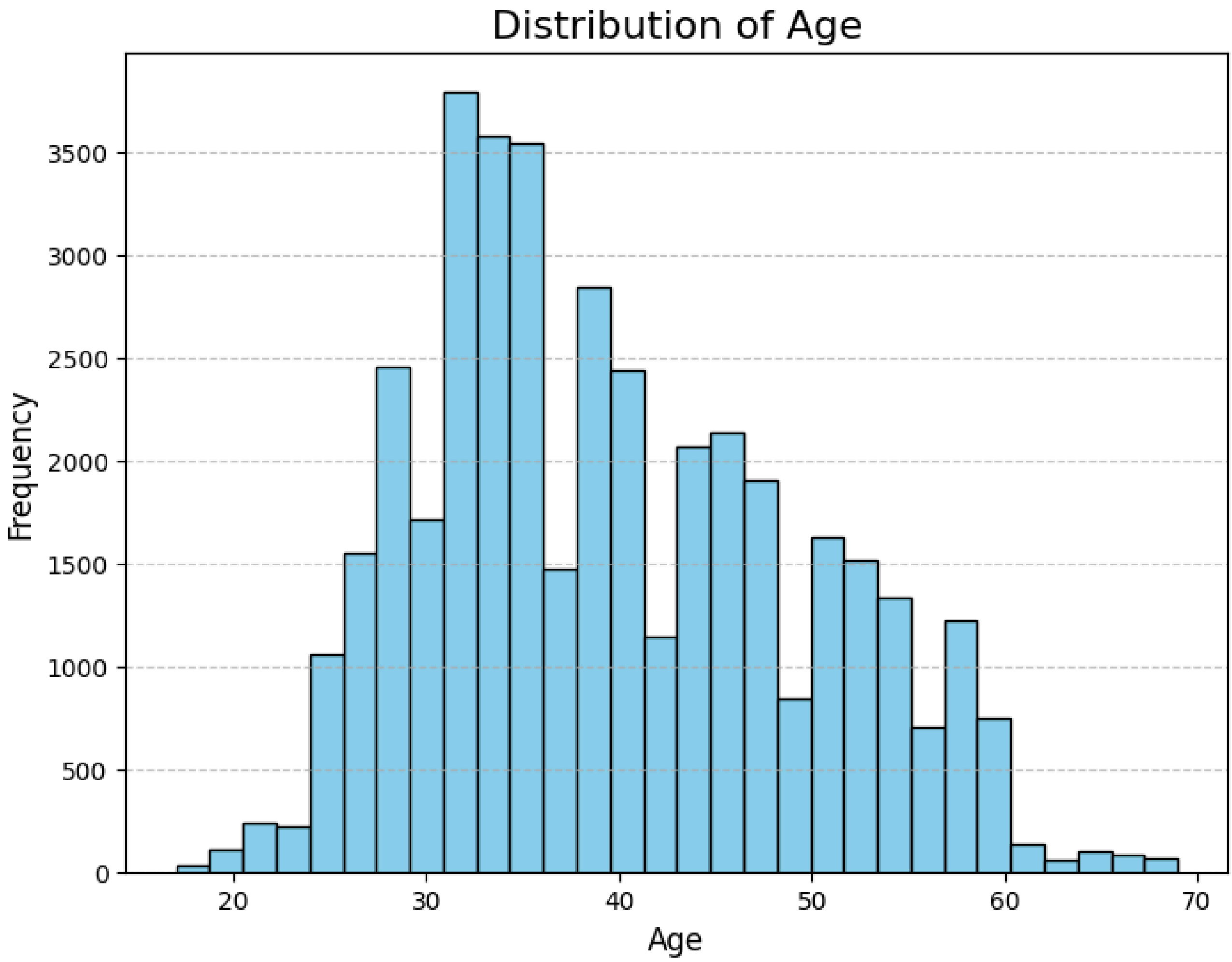



```
import pandas as pd

# Calculate Q1 (25th percentile), Q3 (75th percentile), and IQR
Q1 = data['age'].quantile(0.25)
Q3 = data['age'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

print(f"Lower Bound: {lower_bound}, Upper Bound: {upper_bound}")

# Filter the data to exclude outliers
data = data[(data['age'] >= lower_bound) & (data['age'] <= upper_bound)]

# Print the shape of the dataset before and after removing outliers
print(f"Original data shape: {data.shape}")
print(f"Data shape after removing outliers: {data.shape}")
```

```
Lower Bound: 9.5, Upper Bound: 69.5
Original data shape: (40719, 21)
Data shape after removing outliers: (40719, 21)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.hist(data["age"], bins=30, color='skyblue', edgecolor='black')
```

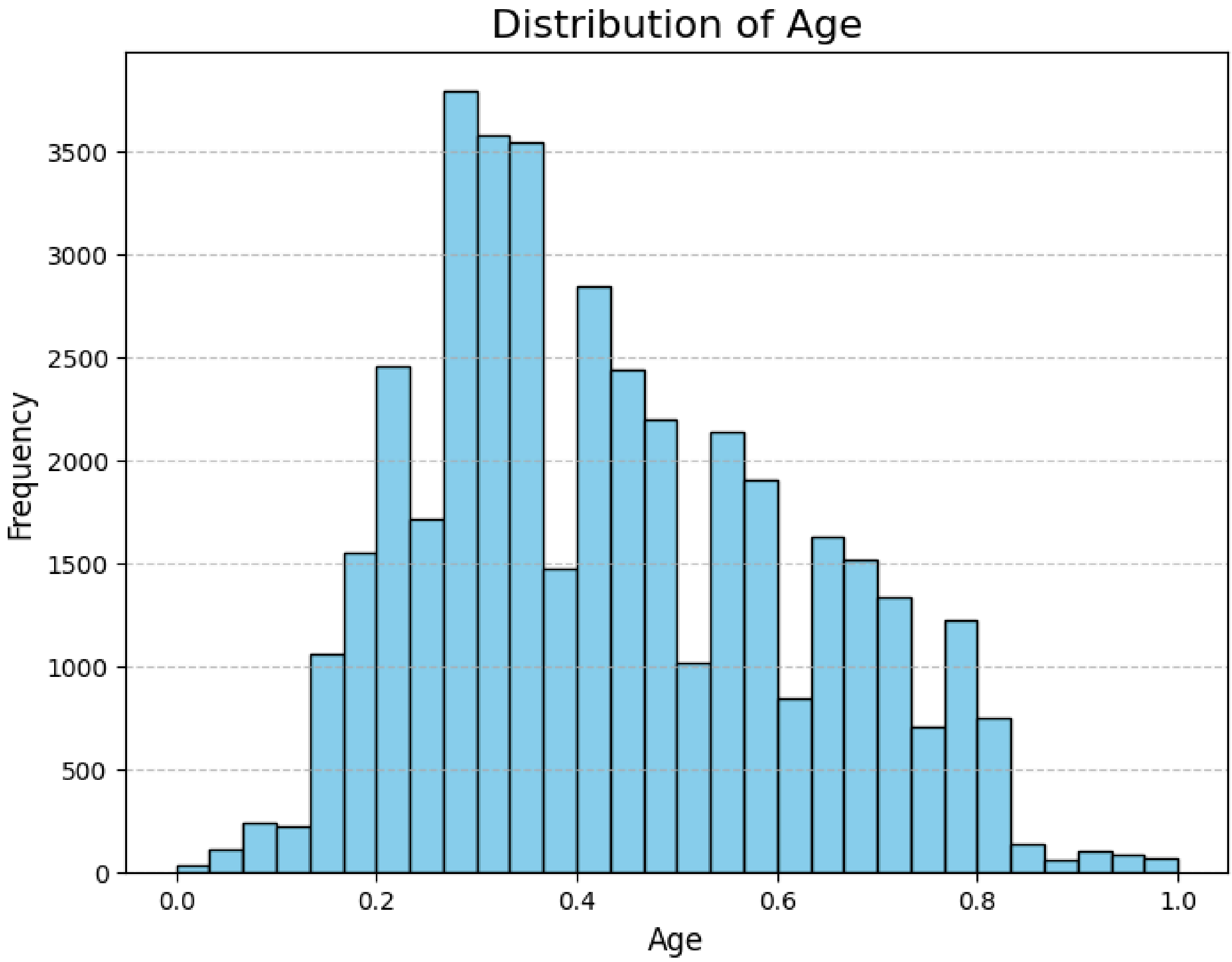
```
plt.title("Distribution of Age", fontsize=16)
plt.xlabel("Age", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```
from sklearn.preprocessing import MinMaxScaler

# Create a scaler instance
scaler = MinMaxScaler()

# Normalize the 'age' column
data['age'] = scaler.fit_transform(data[['age']])

# Display the first few rows of the normalized column
print(data[['age']].head())
```

```
        age
0  0.750000
1  0.769231
2  0.384615
3  0.442308
4  0.750000
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.hist(data["age"], bins=30, color='skyblue', edgecolor='black')
plt.title("Distribution of Age", fontsize=16)
plt.xlabel("Age", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



02.job - Type of job (categorical).

```
data["job"].value_counts()
```

job	count
admin.	10414
blue-collar	9251
technician	6742
services	3969
management	2918
entrepreneur	1456
self-employed	1420
retired	1301
housemaid	1035
unemployed	1014
student	875
unknown	324

dtype: int64

```
import pandas as pd

data_encoded = pd.get_dummies(data['job'], prefix='job', drop_first=False)

data = pd.concat([data, data_encoded], axis=1)

# Display the first few rows of the updated dataframe
print(data.head(2))
```

```
        age        job  marital    education  default housing loan    contact  \
0  0.750000  housemaid  married     basic.4y       no      no   no  telephone
1  0.769231   services  married  high.school  unknown      no   no  telephone

  month day_of_week  ...  job_entrepreneur  job_housemaid  job_management  \
0   may         mon  ...             False           True           False
1   may         mon  ...             False          False           False

   job_retired job_self-employed  job_services  job_student  job_technician  \
0        False             False         False        False           False
1        False             False          True        False           False

   job_unemployed  job_unknown
0           False        False
1           False        False

[2 rows x 33 columns]
```

```
# Drop the 'job' column in-place
data.drop(columns=['job'], inplace=True)

# Verify if the column is removed
print(data.head())
```

```
        age  marital    education  default housing loan    contact month  \
0  0.750000  married     basic.4y       no      no   no  telephone   may   
1  0.769231  married  high.school  unknown      no   no  telephone   may   
2  0.384615  married  high.school       no     yes   no  telephone   may   
3  0.442308  married     basic.6y       no      no   no  telephone   may   
4  0.750000  married  high.school       no      no  yes  telephone   may   

  day_of_week  duration  ...  job_entrepreneur  job_housemaid  job_management  \
0         mon       261  ...             False           True           False   
1         mon       149  ...             False          False           False   
2         mon       226  ...             False          False           False   
3         mon       151  ...             False          False           False   
4         mon       307  ...             False          False           False   

  job_retired  job_self-employed  job_services  job_student  job_technician  \
0       False              False         False        False           False   
1       False              False          True        False           False   
2       False              False          True        False           False   
3       False              False         False        False           False   
4       False              False          True        False           False   

   job_unemployed job_unknown  
0           False       False  
1           False       False  
2           False       False  
3           False       False  
4           False       False  

[5 rows x 32 columns]
```

03.marital - Marital status (categorical).

```
data["marital"].value_counts()
```

	count
marital	
married	24610
single	11553
divorced	4476
unknown	80

dtype: int64

```
# Replace unknown with the most frequent category
most_frequent_marital = data['marital'].mode()[0]
```

```
data['marital'].replace('unknown', most_frequent_marital, inplace=True)
```

```
data["marital"].value_counts()
```

marital	count
married	24690
single	11553
divorced	4476

dtype: int64

```
# One-hot encode the 'marital' column
data = pd.get_dummies(data, columns=['marital'], drop_first=True)

# Check the result
print(data.head())
```

```
        age    education  default housing loan    contact month day_of_week  \
0  0.750000     basic.4y       no      no   no  telephone   may         mon
1  0.769231  high.school  unknown      no   no  telephone   may         mon
2  0.384615  high.school       no     yes   no  telephone   may         mon
3  0.442308     basic.6y       no      no   no  telephone   may         mon
4  0.750000  high.school       no      no  yes  telephone   may         mon

   duration  campaign  ...  job_management  job_retired job_self-employed  \
0       261         1  ...           False        False             False
1       149         1  ...           False        False             False
2       226         1  ...           False        False             False
3       151         1  ...           False        False             False
4       307         1  ...           False        False             False

   job_services  job_student  job_technician  job_unemployed  job_unknown  \
0         False        False           False           False        False
1          True        False           False           False        False
2          True        False           False           False        False
3         False        False           False           False        False
4          True        False           False           False        False

  marital_married  marital_single
0            True           False
1            True           False
2            True           False
3            True           False
4            True           False

[5 rows x 33 columns]
```

04.education - Education level (categorical).

```
data["education"].value_counts()
```

education	count
university.degree	12105
high.school	9481
basic.9y	6018
professional.course	5201
basic.4y	3935
basic.6y	2279
unknown	1683
illiterate	17

dtype: int64

```
# Find the most frequent category in the 'education' column
most_frequent_education = data['education'].mode()[0]

# Replace 'unknown' with the most frequent category
data['education'] = data['education'].replace('unknown', most_frequent_education)


# Check the value counts after replacement
print(data['education'].value_counts())
```

```
education
university.degree      13788
high.school             9481
basic.9y                6018
professional.course     5201
basic.4y                3935
basic.6y                2279
illiterate                17
Name: count, dtype: int64
```

```
# One-hot encode the 'education' column
education_encoded = pd.get_dummies(data['education'], prefix='education')

# Join the one-hot encoded columns back to the original DataFrame
data = pd.concat([data, education_encoded], axis=1)

# Drop the original 'education' column
data.drop('education', axis=1, inplace=True)

# Print the updated DataFrame
print(data.head())
```

```
        age  default housing loan    contact month day_of_week  duration  \
0  0.750000       no      no   no  telephone   may         mon       261
1  0.769231  unknown      no   no  telephone   may         mon       149
2  0.384615       no     yes   no  telephone   may         mon       226
3  0.442308       no      no   no  telephone   may         mon       151
4  0.750000       no      no  yes  telephone   may         mon       307

   campaign  pdays  ...  job_unknown marital_married  marital_single  \
0         1    999  ...        False            True           False
1         1    999  ...        False            True           False
2         1    999  ...        False            True           False
3         1    999  ...        False            True           False
4         1    999  ...        False            True           False

   education_basic.4y  education_basic.6y  education_basic.9y  \
0                True               False               False
1               False               False               False
2               False               False               False
3               False                True               False
4               False               False               False

   education_high.school education_illiterate  education_professional.course  \
0                  False                False                          False
1                   True                False                          False
2                   True                False                          False
3                  False                False                          False
4                   True                False                          False

   education_university.degree
0                        False
1                        False
2                        False
3                        False
4                        False

[5 rows x 39 columns]
```

05.default - Has credit in default? (binary).

```
data["default"].value_counts()
```

	count
default	
no	32162
unknown	8554
yes	3

dtype: int64

```
# Drop the 'default' column if it doesn't provide useful information
data = data.drop('default', axis=1)
```

```
# Check the remaining columns
print(data.columns)
```

```
Index(['age', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'duration',
       'campaign', 'pdays', 'previous', 'poutcome', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed', 'y',
       'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
       'marital_married', 'marital_single', 'education_basic.4y',
       'education_basic.6y', 'education_basic.9y', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree'],
      dtype='object')
```

06.balance - Average yearly balance in euros (numeric).

Start coding or generate with AI.

07.housing - Has housing loan? (binary).

```
data["housing"].value_counts()
```

housing	count
yes	21319
no	18419
unknown	981

dtype: int64

```
# Replace 'unknown' with the most frequent value ('yes')
data['housing'] = data['housing'].replace('unknown', 'yes')

# Check the updated value counts
print(data['housing'].value_counts())
```

```
housing
yes    22300
no     18419
Name: count, dtype: int64
```

```
# One-hot encode the 'housing' column
housing_encoded = pd.get_dummies(data['housing'], prefix='housing')

# Join the encoded columns back to the original dataframe
```

```
data = pd.concat([data, housing_encoded], axis=1)

# Drop the original 'housing' column if it's no longer needed
data = data.drop(columns=['housing'])

# Check the updated dataframe
print(data.head())
```

```
        age loan    contact month day_of_week  duration  campaign  pdays  \
0  0.750000   no  telephone   may         mon       261         1    999
1  0.769231   no  telephone   may         mon       149         1    999
2  0.384615   no  telephone   may         mon       226         1    999
3  0.442308   no  telephone   may         mon       151         1    999
4  0.750000  yes  telephone   may         mon       307         1    999

   previous     poutcome  ...  marital_single  education_basic.4y  \
0         0  nonexistent  ...           False                True
1         0  nonexistent  ...           False               False
2         0  nonexistent  ...           False               False
3         0  nonexistent  ...           False               False
4         0  nonexistent  ...           False               False

   education_basic.6y  education_basic.9y  education_high.school  \
0               False               False                  False
1               False               False                   True
2               False               False                   True
3                True               False                  False
4               False               False                   True

  education_illiterate  education_professional.course  \
0                False                          False
1                False                          False
2                False                          False
3                False                          False
4                False                          False

   education_university.degree  housing_no  housing_yes
0                        False        True        False
1                        False        True        False
2                        False       False         True
3                        False        True        False
4                        False        True        False

[5 rows x 39 columns]
```

08.loan - Has personal loan? (binary).

```
data["loan"].value_counts()
```

	count
loan	
no	33560
yes	6178
unknown	981

dtype: int64

```
# Replace 'unknown' with the most frequent value ('yes')
data['loan'] = data['loan'].replace('unknown', 'yes')

# Check the updated value counts
print(data['loan'].value_counts())
```

```
loan
no     33560
yes     7159
Name: count, dtype: int64
```

```
# One-hot encode the 'loan' column
loan_encoded = pd.get_dummies(data['loan'], prefix='loan')

# Join the encoded columns back to the original dataframe
data = pd.concat([data, loan_encoded], axis=1)

# Drop the original 'loan' column if it's no longer needed
data = data.drop(columns=['loan'])

# Check the updated dataframe
print(data.head())
```

```
        age    contact month day_of_week  duration  campaign  pdays  previous  \
0  0.750000  telephone   may         mon       261         1    999         0
1  0.769231  telephone   may         mon       149         1    999         0
2  0.384615  telephone   may         mon       226         1    999         0
3  0.442308  telephone   may         mon       151         1    999         0
4  0.750000  telephone   may         mon       307         1    999         0

      poutcome  emp.var.rate  ...  education_basic.6y  education_basic.9y  \
0  nonexistent           1.1  ...               False               False
1  nonexistent           1.1  ...               False               False
2  nonexistent           1.1  ...               False               False
3  nonexistent           1.1  ...                True               False
4  nonexistent           1.1  ...               False               False

   education_high.school  education_illiterate education_professional.course  \
0                  False                 False                          False
1                   True                 False                          False
2                   True                 False                          False
3                  False                 False                          False
4                   True                 False                          False
```

```
   education_university.degree  housing_no  housing_yes  loan_no  loan_yes
0                        False        True        False     True     False
1                        False        True        False     True     False
2                        False       False         True     True     False
3                        False        True        False     True     False
4                        False        True        False    False      True

[5 rows x 40 columns]
```

09.contact - Communication type for last contact (categorical).

```
data["contact"].value_counts()
```

contact	count
cellular	25724
telephone	14995

dtype: int64

```
# Drop the 'contact' column
data = data.drop(columns=['contact'])

# Check the updated dataframe
print(data.head())
```

```
        age month day_of_week  duration  campaign  pdays  previous  \
0  0.750000   may         mon       261         1    999         0
1  0.769231   may         mon       149         1    999         0
2  0.384615   may         mon       226         1    999         0
3  0.442308   may         mon       151         1    999         0
4  0.750000   may         mon       307         1    999         0

      poutcome  emp.var.rate  cons.price.idx  ...  education_basic.6y  \
0  nonexistent           1.1          93.994  ...               False
1  nonexistent           1.1          93.994  ...               False
2  nonexistent           1.1          93.994  ...               False
3  nonexistent           1.1          93.994  ...                True
4  nonexistent           1.1          93.994  ...               False

   education_basic.9y  education_high.school education_illiterate  \
0               False                  False                False
1               False                   True                False
2               False                   True                False
3               False                  False                False
4               False                   True                False

   education_professional.course  education_university.degree  housing_no  \
0                          False                        False        True
1                          False                        False        True
2                          False                        False       False
3                          False                        False        True
```

```
4                          False                          False      True

   housing_yes  loan_no  loan_yes
0        False     True     False
1        False     True     False
2         True     True     False
3        False     True     False
4        False    False      True

[5 rows x 39 columns]
```

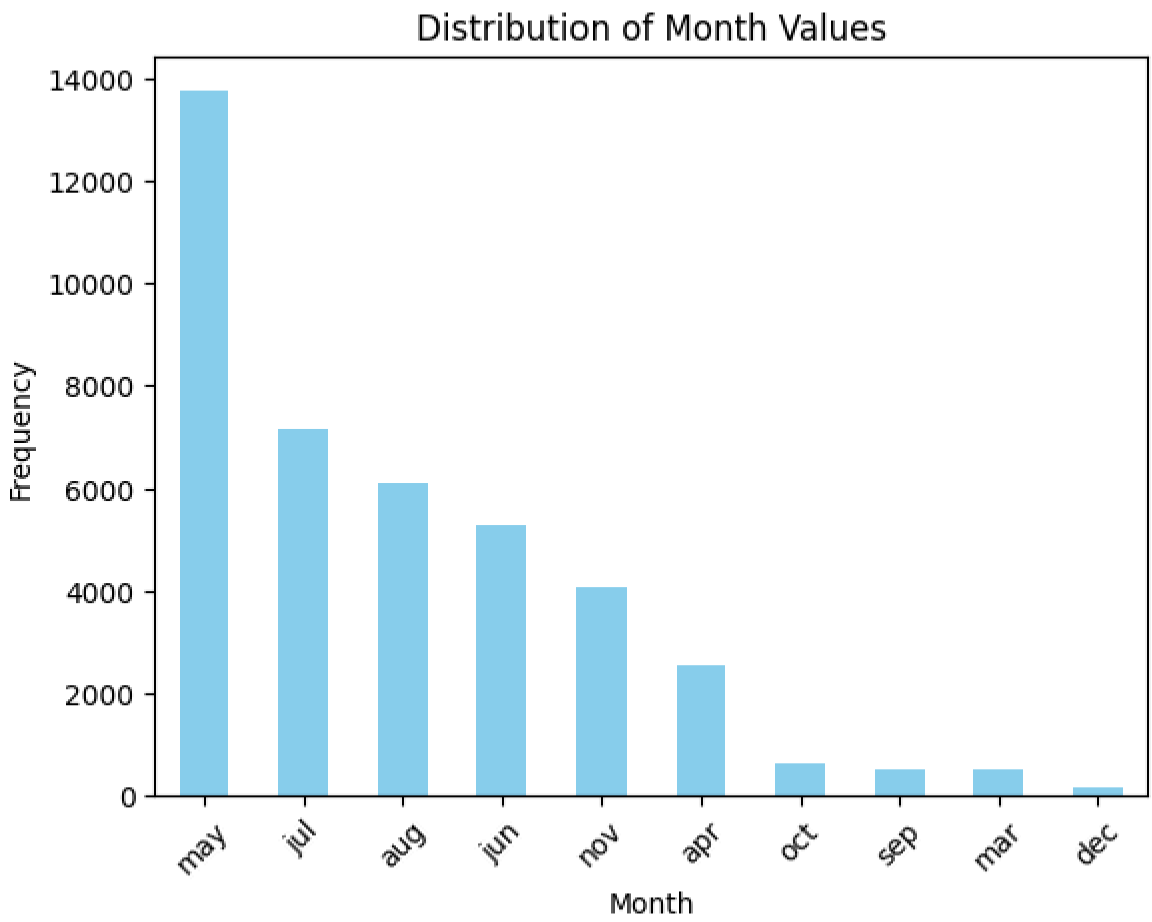
10.day - Last contact day of the month (numeric).

```
data["month"].value_counts()
```

month	count
may	13736
jul	7141
aug	6091
jun	5301
nov	4064
apr	2562
oct	648
sep	513
mar	503
dec	160

dtype: int64

```
import matplotlib.pyplot as plt

# Plot the value counts of the 'month' column
data['month'].value_counts().plot(kind='bar', color='skyblue')

# Set labels and title
plt.title('Distribution of Month Values')
plt.xlabel('Month')
plt.ylabel('Frequency')

# Show the plot
plt.xticks(rotation=45)
plt.show()
```




```
import pandas as pd

data = pd.get_dummies(data, columns=['month'], drop_first=False)

# Display the transformed data to check the result
print(data.head())
```

```
        age day_of_week  duration  campaign  pdays  previous     poutcome  \
0  0.750000         mon       261         1    999         0  nonexistent
1  0.769231         mon       149         1    999         0  nonexistent
2  0.384615         mon       226         1    999         0  nonexistent
3  0.442308         mon       151         1    999         0  nonexistent
4  0.750000         mon       307         1    999         0  nonexistent

  emp.var.rate  cons.price.idx  cons.conf.idx  ...  month_apr  month_aug  \
0          1.1          93.994          -36.4  ...      False      False
1          1.1          93.994          -36.4  ...      False      False
2          1.1          93.994          -36.4  ...      False      False
3          1.1          93.994          -36.4  ...      False      False
4          1.1          93.994          -36.4  ...      False      False

  month_dec  month_jul  month_jun  month_mar  month_may  month_nov  month_oct  \
0     False      False      False      False       True      False      False
1     False      False      False      False       True      False      False
2     False      False      False      False       True      False      False
3     False      False      False      False       True      False      False
4     False      False      False      False       True      False      False

   month_sep
0      False
```

```
1      False
2      False
3      False
4      False

[5 rows x 48 columns]
```

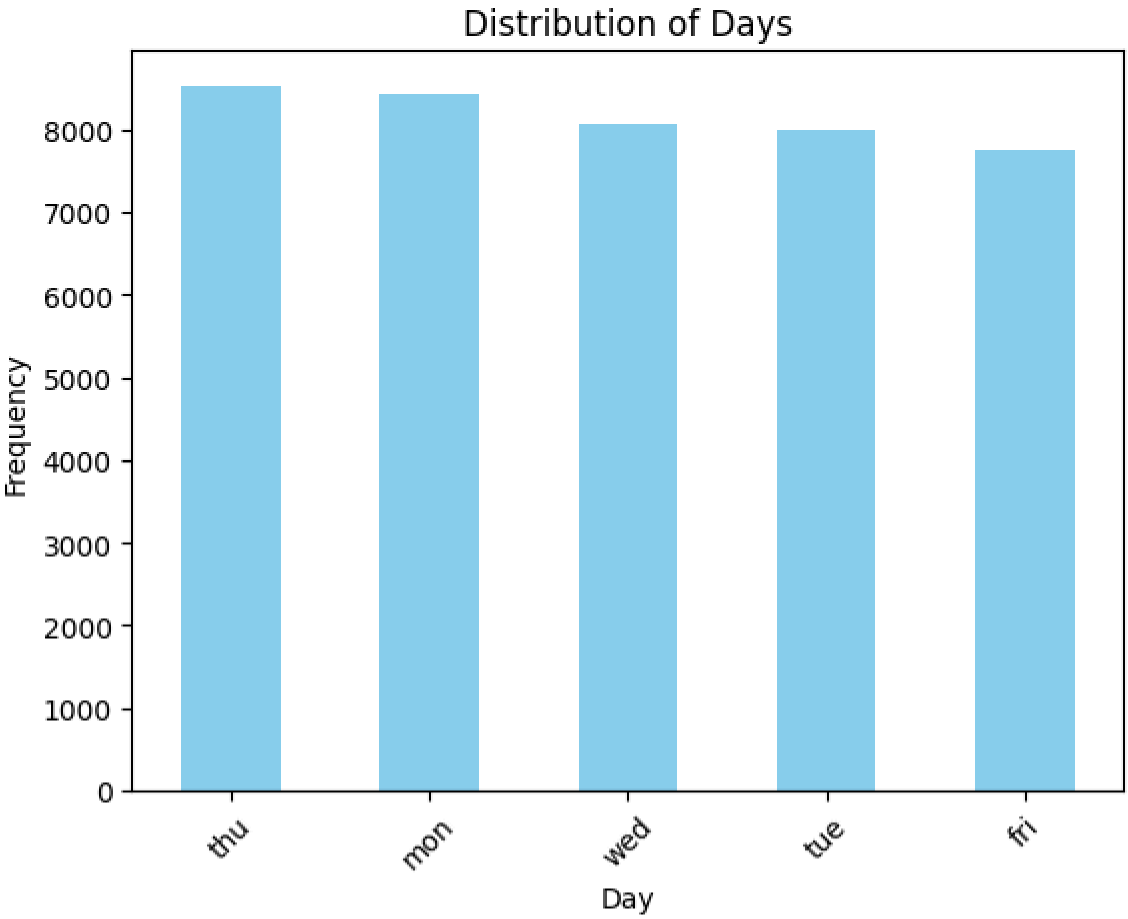
11.month - Last contact month (categorical).

```
data["day_of_week"].value_counts()
```

day_of_week	count
thu	8522
mon	8426
wed	8052
tue	7980
fri	7739

dtype: int64

```
import matplotlib.pyplot as plt

# Plot the value counts of the 'day_of_week' column
data['day_of_week'].value_counts().plot(kind='bar', color='skyblue')

# Set labels and title
plt.title('Distribution of Days')
plt.xlabel('Day')
plt.ylabel('Frequency')

# Show the plot
plt.xticks(rotation=45)
plt.show()
```




```
# Drop the 'day_of_week' column in-place
data.drop(columns=['day_of_week'], inplace=True)

# Verify if the column is removed
print(data.head())
```

```
        age  duration  campaign  pdays  previous     poutcome  emp.var.rate  \
0  0.750000       261         1    999         0  nonexistent           1.1
1  0.769231       149         1    999         0  nonexistent           1.1
2  0.384615       226         1    999         0  nonexistent           1.1
3  0.442308       151         1    999         0  nonexistent           1.1
4  0.750000       307         1    999         0  nonexistent           1.1

   cons.price.idx  cons.conf.idx  euribor3m  ...  month_apr month_aug  \
0          93.994          -36.4      4.857  ...      False     False
1          93.994          -36.4      4.857  ...      False     False
2          93.994          -36.4      4.857  ...      False     False
3          93.994          -36.4      4.857  ...      False     False
4          93.994          -36.4      4.857  ...      False     False

   month_dec  month_jul  month_jun  month_mar  month_may  month_nov  \
0      False      False      False      False       True      False
1      False      False      False      False       True      False
2      False      False      False      False       True      False
3      False      False      False      False       True      False
4      False      False      False      False       True      False

   month_oct  month_sep
0      False      False
1      False      False
```

```
2      False      False
3      False      False
4      False      False

[5 rows x 47 columns]
```

12.duration - Last contact duration in seconds (numeric).

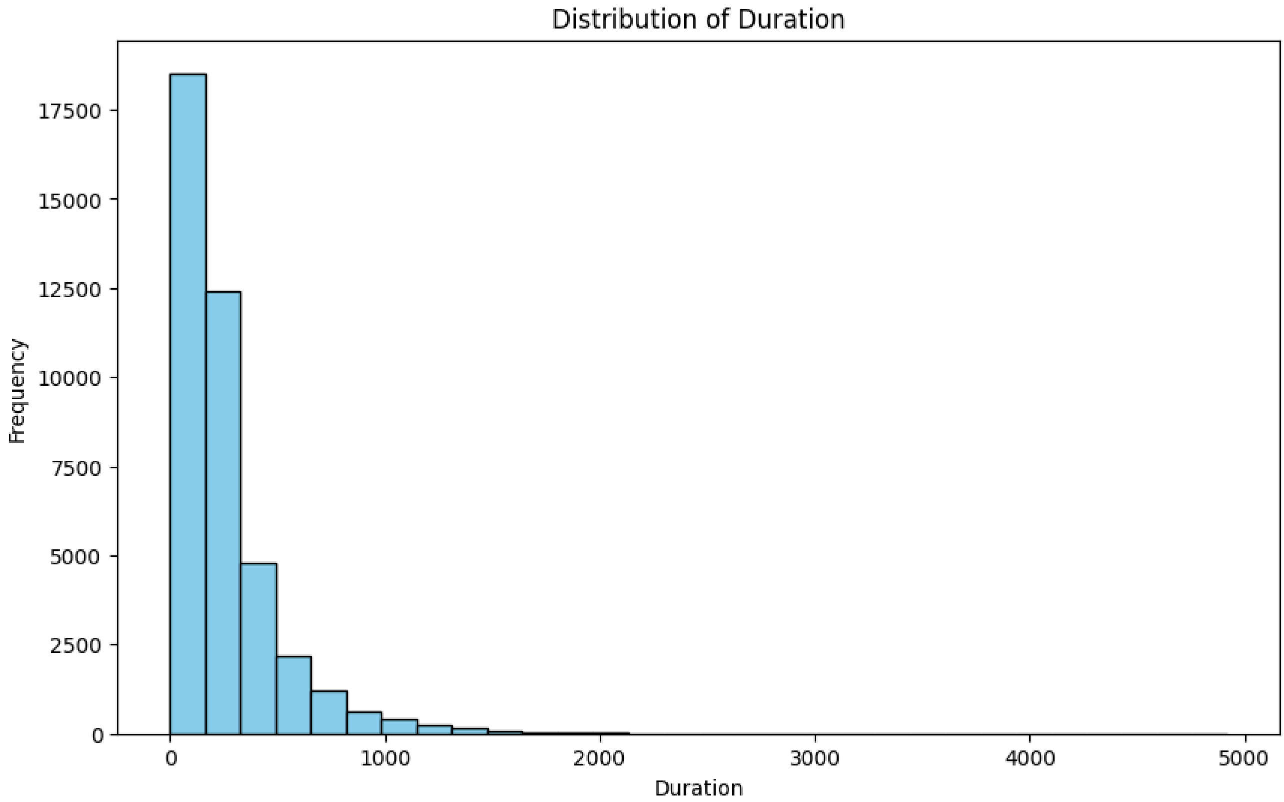
```
data["duration"].value_counts()
```

duration	count
85	168
136	167
90	167
73	166
124	163
...	...
1275	1
1473	1
1432	1
1412	1
1868	1

1542 rows × 1 columns

dtype: int64

```
import matplotlib.pyplot as plt

# Plot the distribution of the 'duration' column
plt.figure(figsize=(10, 6))
plt.hist(data['duration'], bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Duration')
plt.xlabel('Duration')
plt.ylabel('Frequency')
plt.show()
```

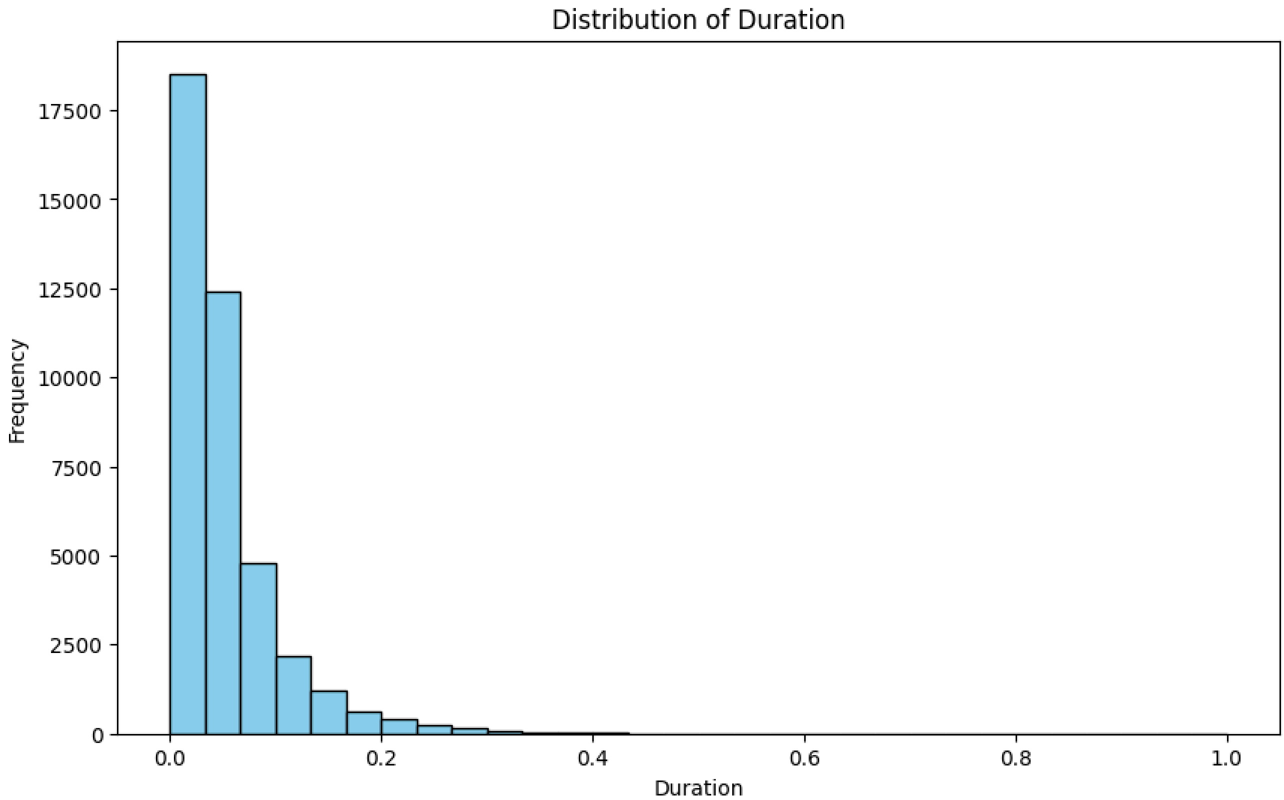



```
from sklearn.preprocessing import MinMaxScaler

# Reshape the data as it is a single column
scaler = MinMaxScaler()
data['duration'] = scaler.fit_transform(data[['duration']])

# Check the result
print(data[['duration']].head())
```

```
   duration
0  0.053070
1  0.030297
2  0.045954
3  0.030704
4  0.062424
```

```
import matplotlib.pyplot as plt

# Plot the distribution of the 'duration' column
plt.figure(figsize=(10, 6))
plt.hist(data['duration'], bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Duration')
plt.xlabel('Duration')
```

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



13.campaign - Number of contacts in this campaign (numeric)

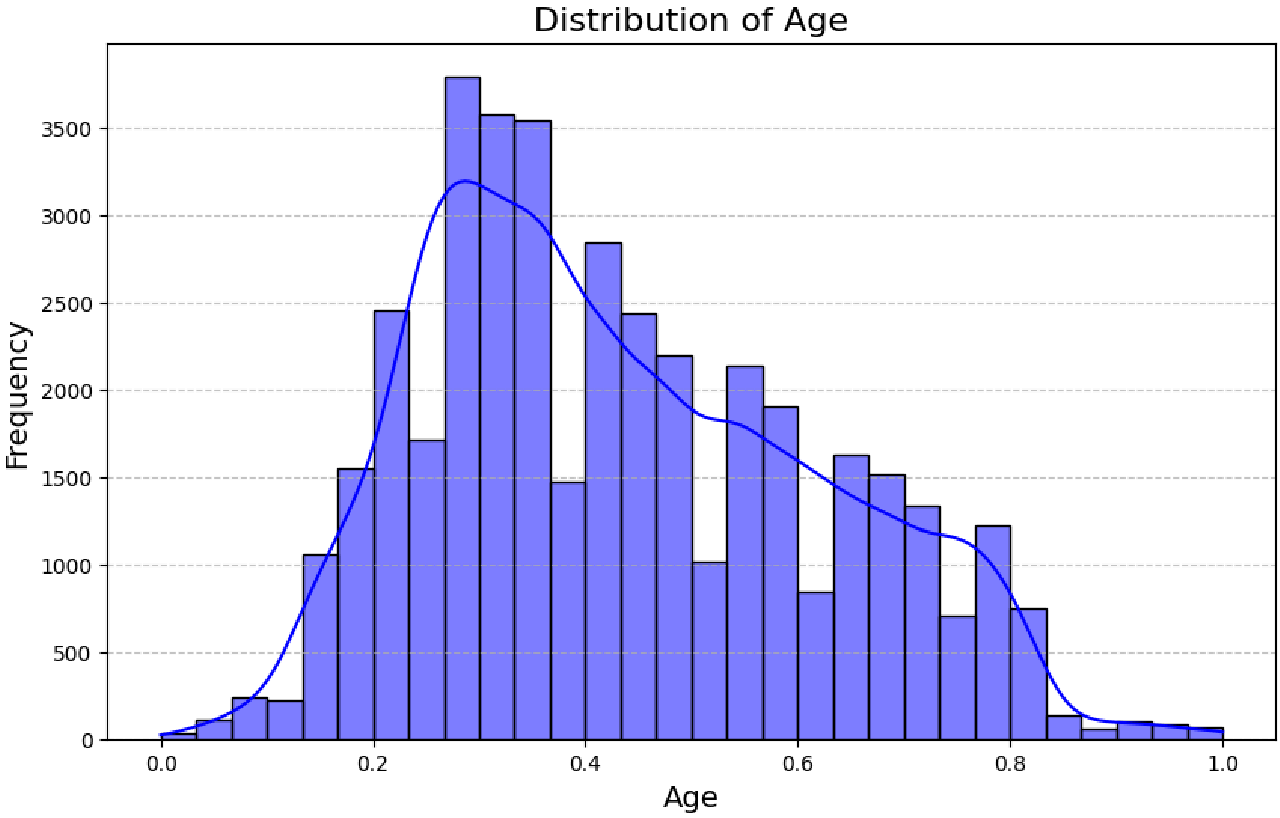
```
data["campaign"].value_counts()
```

	count
campaign	
1	17388
2	10444
3	5300
4	2631
5	1594
6	970
7	624
8	396
9	280
10	225
11	177
12	124
13	92
14	69
17	58
15	51
16	50
18	33
20	30
19	26
21	24
22	17
23	16
24	15
27	11
29	10
25	8
28	8
26	8
30	7
31	7

35	5
32	4
33	4
34	3
42	2
40	2
43	2
56	1
39	1
41	1
37	1

dtype: int64

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of age
plt.figure(figsize=(10, 6))
sns.histplot(data['age'], bins=30, kde=True, color='blue')  # KDE adds a smooth curve
plt.title('Distribution of Age', fontsize=16)
plt.xlabel('Age', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```
from sklearn.preprocessing import MinMaxScaler

# Reshape the data as it is a single column
scaler = MinMaxScaler()
data['campaign'] = scaler.fit_transform(data[['campaign']])

# Check the result
print(data[['campaign']].head())
```

```
   campaign
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
```

14.pdays - Days passed since the last contact in a previous campaign (numeric, -1 for no previous contact).

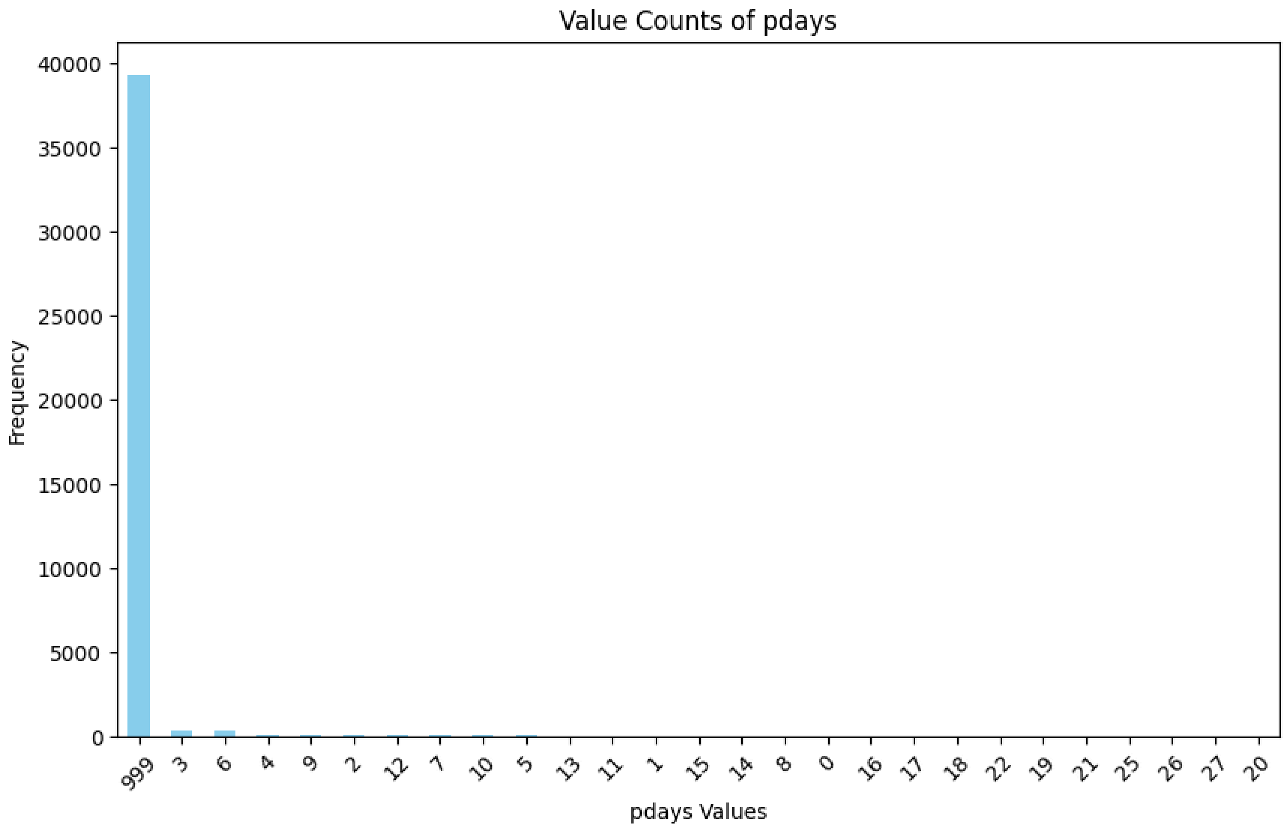
```
data["pdays"].value_counts()
```

pdays	count
999	39302
3	399
6	382
4	111
9	60
2	60
12	56
7	55
10	51
5	45
13	34
11	28
1	26
15	22
14	18
8	17
0	15
16	11
17	8
18	7
22	3
19	3
21	2
25	1
26	1
27	1
20	1

dtype: int64

```
import matplotlib.pyplot as plt

# Get the value counts of the 'pdays' column
```

```
pdays_counts = data["pdays"].value_counts()

# Plot the result
pdays_counts.plot(kind='bar', figsize=(10,6), color='skyblue')

# Adding labels and title
plt.xlabel('pdays Values')
plt.ylabel('Frequency')
plt.title('Value Counts of pdays')
plt.xticks(rotation=45)
plt.show()
```




```
# Drop the 'pdays' column
# data = data.drop(columns=['pdays'])
# print(data.head())
```

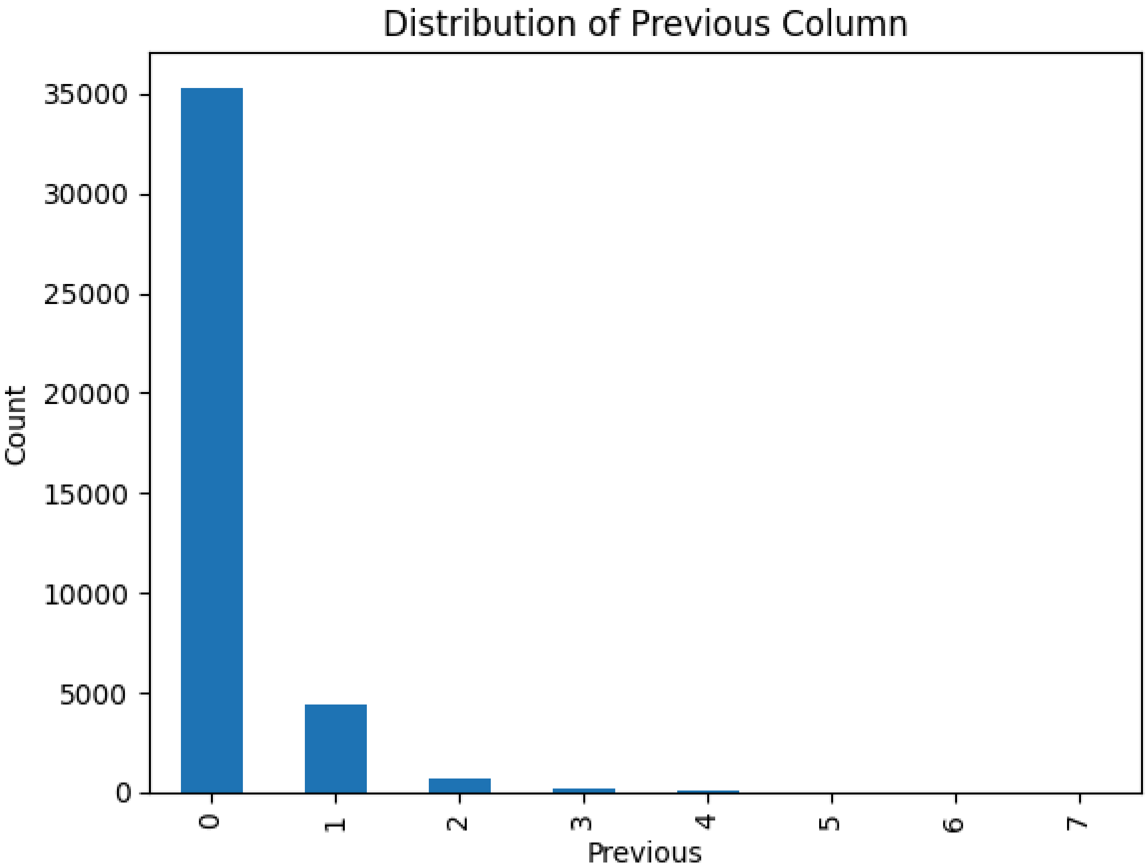
15.previous - Number of contacts before the current campaign (numeric).

```
data["previous"].value_counts()
```

previous	count
0	35296
1	4439
2	700
3	200
4	61
5	18
6	4
7	1

dtype: int64

```
import matplotlib.pyplot as plt

# Plot the value counts for the 'previous' column
data['previous'].value_counts().sort_index().plot(kind='bar')

# Adding labels and title
plt.xlabel('Previous')
plt.ylabel('Count')
plt.title('Distribution of Previous Column')

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




```
# Drop the 'pdays' column
# data = data.drop(columns=['previous'])
# print(data.head())
```

16.poutcome - Outcome of the previous campaign (categorical).

```
data["poutcome"].value_counts()
```

poutcome	count
nonexistent	35296
failure	4141
success	1282

dtype: int64

```
import pandas as pd

# Perform one-hot encoding on 'poutcome'
one_hot_encoded = pd.get_dummies(data['poutcome'], prefix='poutcome')

# Add the one-hot encoded columns back to the original dataset
data = pd.concat([data, one_hot_encoded], axis=1)
```

```
# Display counts for the one-hot encoded columns
print("One-Hot Encoded Counts:")
print(one_hot_encoded.sum())
```

```
One-Hot Encoded Counts:
poutcome_failure         4141
poutcome_nonexistent    35296
poutcome_success         1282
dtype: int64
```

```
# Drop the 'poutcome' column
data = data.drop(columns=['poutcome'])
```

17.cons.price.idx - Numeric: Consumer price index.

```
data["cons.price.idx"].value_counts()
```

cons.price.idx	count
93.994	7763
93.918	6685
92.893	5785
93.444	5175
94.465	4374
93.200	3616
93.075	2418
92.963	710
92.201	704
92.431	392
92.649	322
94.215	279
94.199	278
92.843	254
93.369	249
92.379	235
94.055	217
94.027	212
94.601	189
93.876	188
92.469	177
92.713	150
93.749	144
94.767	126
93.798	67
92.756	10

dtype: int64

18.cons.conf.idx - Numeric: Consumer confidence index.

```
data["cons.conf.idx"].value_counts()
```

cons.conf.idx	count
-36.4	7763
-42.7	6685
-46.2	5785
-36.1	5175
-41.8	4374
-42.0	3616
-47.1	2418
-40.8	710
-31.4	704
-26.9	392
-30.1	322
-40.3	279
-37.5	278
-50.0	254
-34.8	249
-29.8	235
-39.8	217
-38.3	212
-49.5	189
-40.0	188
-33.6	177
-33.0	150
-34.6	144
-50.8	126
-40.4	67
-45.9	10

dtype: int64

19.emp.var.rate - Numeric: Employment variation rate.

```
data["emp.var.rate"].value_counts()
```

emp.var.rate	count
1.4	16234
-1.8	9038
1.1	7763
-0.1	3683
-2.9	1591
-3.4	949
-1.7	708
-1.1	593
-3.0	150
-0.2	10

dtype: int64

20.euribor3m - Numeric: Euribor 3-month rate.

```
data["euribor3m"].value_counts()
```

euribor3m	count
4.857	2868
4.962	2613
4.963	2487
4.961	1902
4.856	1210
...	...
3.743	1
3.282	1
3.669	1
3.488	1
0.956	1

316 rows × 1 columns

dtype: int64

21.nr.employed - Numeric: Number of employees.

```
data["nr.employed"].value_counts()
```

nr.employed	count
5228.1	16234
5099.1	8457
5191.0	7763
5195.8	3683
5076.2	1591
5017.5	949
4991.6	708
4963.6	593
5008.7	581
5023.5	150
5176.3	10

dtype: int64

22.y - Client subscribed to a term deposit? (binary).

```
data["y"].value_counts()
```

y	count
no	36300
yes	4419

dtype: int64

```
import pandas as pd

data['y'] = data['y'].map({'no': 0, 'yes': 1})


data["y"].value_counts()
```

	count
y	
0	36300
1	4419

dtype: int64

4.2 Encode Categorical Variables

Start coding or generate with AI.

4.3 Feature Engineering

```
data.dtypes
```

	0
age	float64
duration	float64
campaign	float64
pdays	int64
previous	int64
emp.var.rate	float64
cons.price.idx	float64
cons.conf.idx	float64
euribor3m	float64
nr.employed	float64
y	int64
job_admin.	bool
job_blue-collar	bool
job_entrepreneur	bool
job_housemaid	bool
job_management	bool
job_retired	bool
job_self-employed	bool
job_services	bool
job_student	bool
job_technician	bool
job_unemployed	bool
job_unknown	bool
marital_married	bool
marital_single	bool
education_basic.4y	bool
education_basic.6y	bool
education_basic.9y	bool
education_high.school	bool
education_illiterate	bool
education_professional.course	bool
education_university.degree	bool

housing_no	bool
housing_yes	bool
loan_no	bool
loan_yes	bool
month_apr	bool
month_aug	bool
month_dec	bool
month_jul	bool
month_jun	bool
month_mar	bool
month_may	bool
month_nov	bool
month_oct	bool
month_sep	bool
poutcome_failure	bool
poutcome_nonexistent	bool
poutcome_success	bool

dtype: object

4.3.1 Feature Extraction (Principal Component Analysis (PCA))

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Create a DataFrame for the relevant columns
data_pca = data[['pdays', 'previous', 'poutcome_failure', 'poutcome_nonexistent', 'poutco

# Standardize the data (PCA is sensitive to scale)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_pca)

# Apply PCA - reducing to 1 component
pca = PCA(n_components=1)
pca_result = pca.fit_transform(data_scaled)

# Add the PCA result as a new column in the dataset
data['pca_1'] = pca_result

# Drop the original columns (pdays, previous, and the one-hot encoded columns)
data.drop(columns=['pdays', 'previous', 'poutcome_failure', 'poutcome_nonexistent', 'pout

# Display the first few rows of the updated dataset
print(data.head())
```

```
        age  duration  campaign  emp.var.rate  cons.price.idx  cons.conf.idx  \
0  0.750000  0.053070       0.0           1.1          93.994          -36.4
1  0.769231  0.030297       0.0           1.1          93.994          -36.4
2  0.384615  0.045954       0.0           1.1          93.994          -36.4
3  0.442308  0.030704       0.0           1.1          93.994          -36.4
4  0.750000  0.062424       0.0           1.1          93.994          -36.4

   euribor3m  nr.employed  y  job_admin.  ...  month_aug  month_dec  \
0      4.857       5191.0  0       False  ...      False      False
1      4.857       5191.0  0       False  ...      False      False
2      4.857       5191.0  0       False  ...      False      False
3      4.857       5191.0  0        True  ...      False      False
4      4.857       5191.0  0       False  ...      False      False

   month_jul  month_jun  month_mar  month_may  month_nov  month_oct  \
0      False      False      False       True      False      False
1      False      False      False       True      False      False
2      False      False      False       True      False      False
3      False      False      False       True      False      False
4      False      False      False       True      False      False

   month_sep     pca_1
0      False -0.653294
1      False -0.653294
2      False -0.653294
3      False -0.653294
4      False -0.653294
```

```
[5 rows x 45 columns]
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Step 1: Select the columns for PCA
pca_columns = ['cons.price.idx', 'cons.conf.idx', 'emp.var.rate', 'euribor3m', 'nr.employ
pca_data = data[pca_columns]

# Step 2: Standardize the data (PCA requires standardization)
scaler = StandardScaler()
pca_data_scaled = scaler.fit_transform(pca_data)

# Step 3: Apply PCA
pca = PCA(n_components=1)  # Reduce to 1 component for pca_02
data['pca_02'] = pca.fit_transform(pca_data_scaled)

# Step 4: Drop the original columns
data = data.drop(columns=pca_columns)

# Display the updated data
print(data.head())
```

```
        age  duration  campaign  y  job_admin.  job_blue-collar  \
0  0.750000  0.053070       0.0  0       False            False
1  0.769231  0.030297       0.0  0       False            False
2  0.384615  0.045954       0.0  0       False            False
3  0.442308  0.030704       0.0  0        True            False
4  0.750000  0.062424       0.0  0       False            False

   job_entrepreneur  job_housemaid  job_management  job_retired  ...  \
0             False           True           False        False  ...
1             False          False           False        False  ...
2             False          False           False        False  ...
3             False          False           False        False  ...
4             False          False           False        False  ...

   month_dec  month_jul  month_jun  month_mar  month_may  month_nov  \
0      False      False      False      False       True      False
1      False      False      False      False       True      False
2      False      False      False      False       True      False
3      False      False      False      False       True      False
4      False      False      False      False       True      False

   month_oct  month_sep     pca_1    pca_02
0      False      False -0.653294  1.302342
1      False      False -0.653294  1.302342
2      False      False -0.653294  1.302342
3      False      False -0.653294  1.302342
4      False      False -0.653294  1.302342

[5 rows x 41 columns]
```

Start coding or generate with AI.

4.3.2 Feature Selection:

```
Start coding or generate with AI.
```

4.4 Normalize/Scale Numerical Features

```
data.dtypes
```

	0
age	float64
duration	float64
campaign	float64
y	int64
job_admin.	bool
job_blue-collar	bool
job_entrepreneur	bool
job_housemaid	bool
job_management	bool
job_retired	bool
job_self-employed	bool
job_services	bool
job_student	bool
job_technician	bool
job_unemployed	bool
job_unknown	bool
marital_married	bool
marital_single	bool
education_basic.4y	bool
education_basic.6y	bool
education_basic.9y	bool
education_high.school	bool
education_illiterate	bool
education_professional.course	bool
education_university.degree	bool
housing_no	bool
housing_yes	bool
loan_no	bool
loan_yes	bool
month_apr	bool
month_aug	bool
month_dec	bool

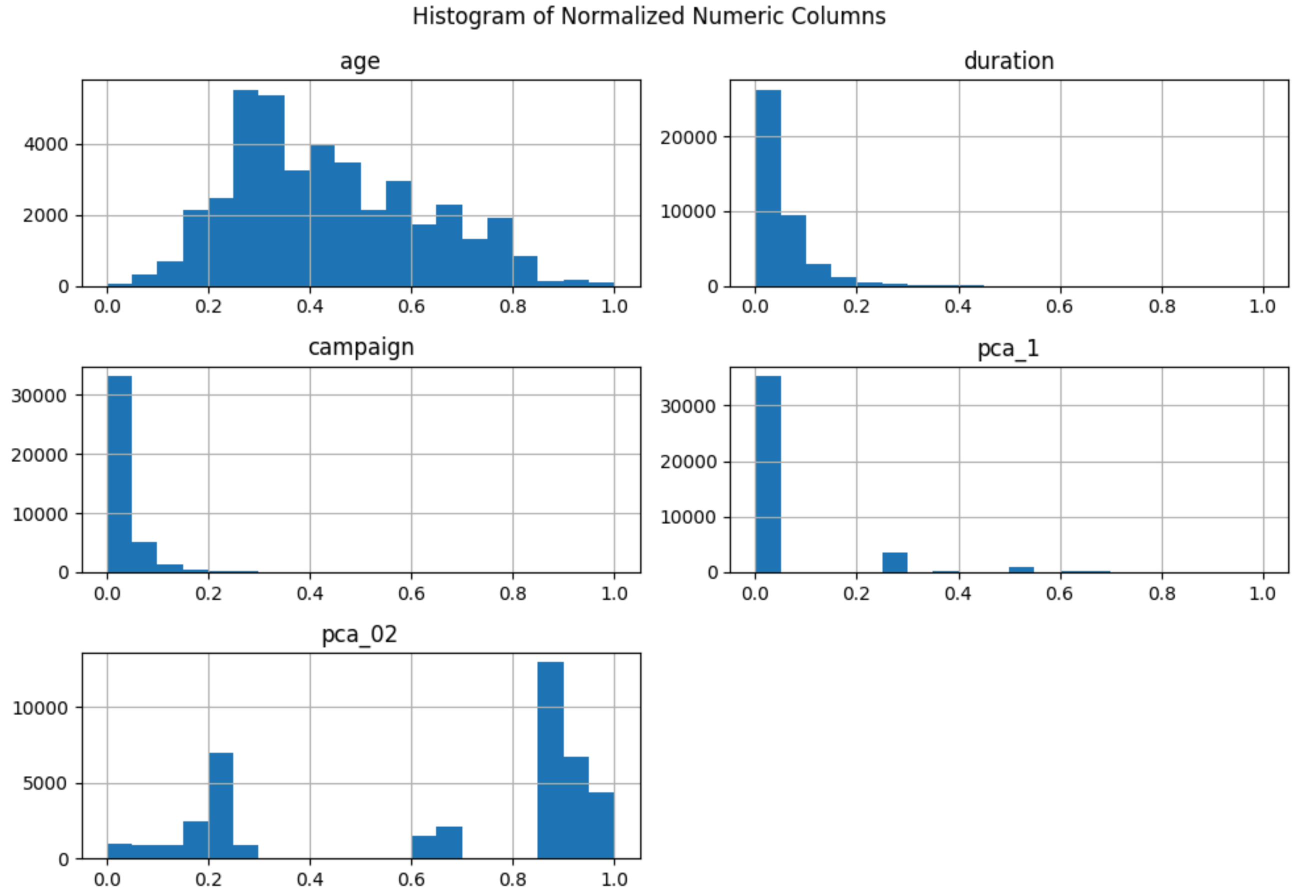
month_jul	bool
month_jun	bool
month_mar	bool
month_may	bool
month_nov	bool
month_oct	bool
month_sep	bool
pca_1	float64
pca_02	float64

dtype: object

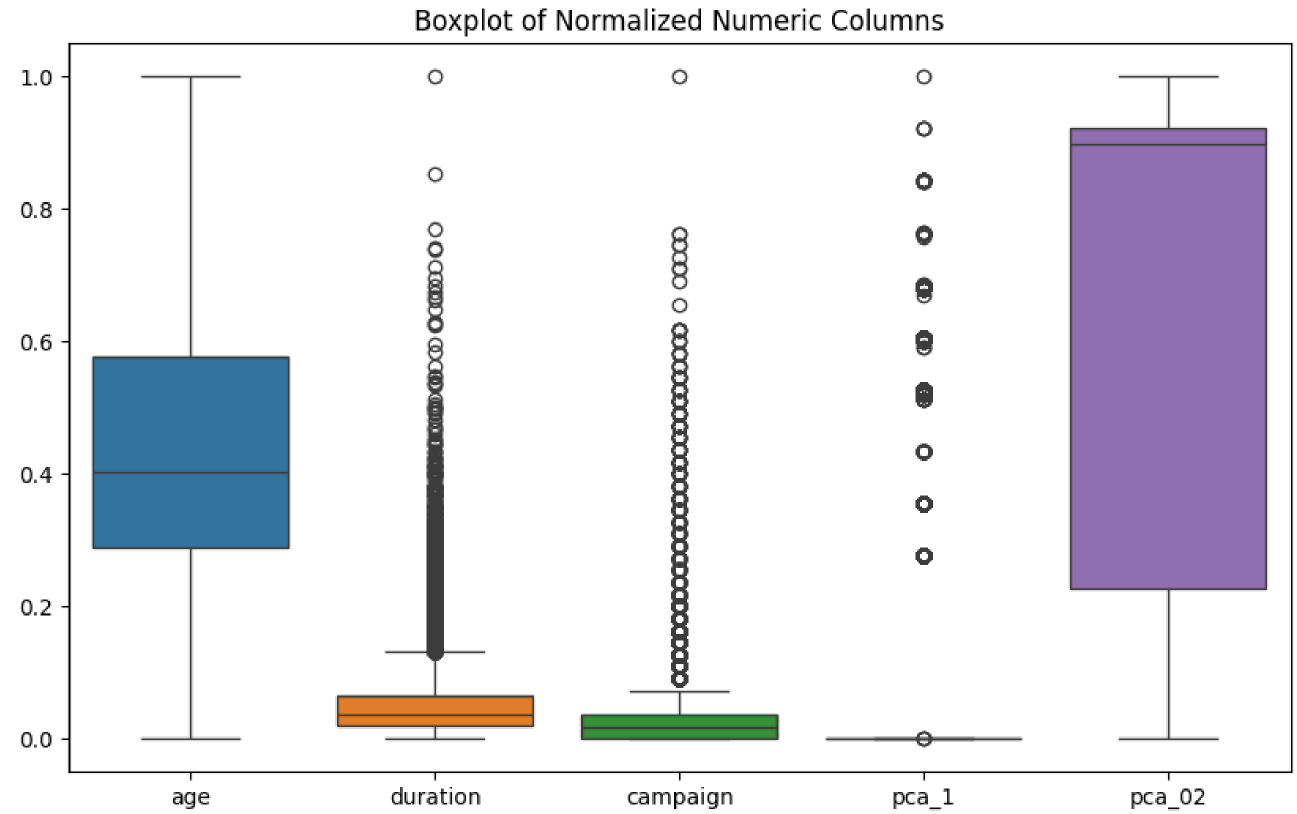
```
from sklearn.preprocessing import MinMaxScaler

# Select numeric columns for normalization
numeric_columns = ['age', 'duration', 'campaign', 'pca_1', 'pca_02']

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize the selected numeric columns
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])

# Display the updated data
print(data[numeric_columns].head())
```

```
        age  duration  campaign  pca_1    pca_02
0  0.750000  0.053070       0.0    0.0  0.898881
1  0.769231  0.030297       0.0    0.0  0.898881
2  0.384615  0.045954       0.0    0.0  0.898881
3  0.442308  0.030704       0.0    0.0  0.898881
4  0.750000  0.062424       0.0    0.0  0.898881
```

```
import matplotlib.pyplot as plt

# Select numeric columns for normalization
numeric_columns = ['age', 'duration', 'campaign', 'pca_1', 'pca_02']

# Plot histograms for each numeric column
data[numeric_columns].hist(bins=20, figsize=(10, 7))
plt.suptitle('Histogram of Normalized Numeric Columns')
plt.tight_layout()
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Select numeric columns for visualization
numeric_columns = ['age', 'duration', 'campaign', 'pca_1', 'pca_02']

# Create a boxplot to visualize distribution and outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=data[numeric_columns])
plt.title('Boxplot of Normalized Numeric Columns')
plt.show()
```



4.5 Handle Class Imbalance

```
data.dtypes
```

	0
age	float64
duration	float64
campaign	float64
y	int64
job_admin.	bool
job_blue-collar	bool
job_entrepreneur	bool
job_housemaid	bool
job_management	bool
job_retired	bool
job_self-employed	bool
job_services	bool
job_student	bool
job_technician	bool
job_unemployed	bool
job_unknown	bool
marital_married	bool
marital_single	bool
education_basic.4y	bool
education_basic.6y	bool
education_basic.9y	bool
education_high.school	bool
education_illiterate	bool
education_professional.course	bool
education_university.degree	bool
housing_no	bool
housing_yes	bool
loan_no	bool
loan_yes	bool
month_apr	bool
month_aug	bool
month_dec	bool

month_jul	bool
month_jun	bool
month_mar	bool
month_may	bool
month_nov	bool
month_oct	bool
month_sep	bool
pca_1	float64
pca_02	float64

dtype: object

oversampling the minority class

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN
from sklearn.model_selection import train_test_split
import pandas as pd

X = data.drop('y', axis=1)  # Features
y = data['y']               # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,

# First, oversample the minority class
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X_train, y_train)

# Next, undersample the majority class from the oversampled dataset
undersampler = RandomUnderSampler(random_state=42)
X_balanced, y_balanced = undersampler.fit_resample(X_resampled, y_resampled)

# Alternatively, use SMOTEENN for a hybrid approach (Optional)
smoteenn = SMOTEENN(random_state=42)
X_smoteenn, y_smoteenn = smoteenn.fit_resample(X_train, y_train)

# Display the class distribution after balancing
from collections import Counter
print("Class distribution after oversampling and undersampling:", Counter(y_balanced))
print("Class distribution after SMOTEENN:", Counter(y_smoteenn))
```

```
# You can now train your model on X_balanced or X_smoteenn
```

```
Class distribution after oversampling and undersampling: Counter({0: 25410, 1: 25410}
Class distribution after SMOTEENN: Counter({1: 21694, 0: 20559})
```

undersampling the majority class

Start coding or generate with AI.

4.6 Split the Dataset

```
from sklearn.model_selection import train_test_split

# Split into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X_smoteenn, y_smoteenn, test_size=0.2, random_state=42, stratify=y_smoteenn
)

# Display the shape of the resulting splits
print("Training set size:", X_train.shape, y_train.shape)
print("Testing set size:", X_test.shape, y_test.shape)

# Check the class distribution in the training and testing sets
from collections import Counter
print("Class distribution in training set:", Counter(y_train))
print("Class distribution in testing set:", Counter(y_test))
```

```
Training set size: (33802, 40) (33802,)
Testing set size: (8451, 40) (8451,)
Class distribution in training set: Counter({1: 17355, 0: 16447})
Class distribution in testing set: Counter({1: 4339, 0: 4112})
```

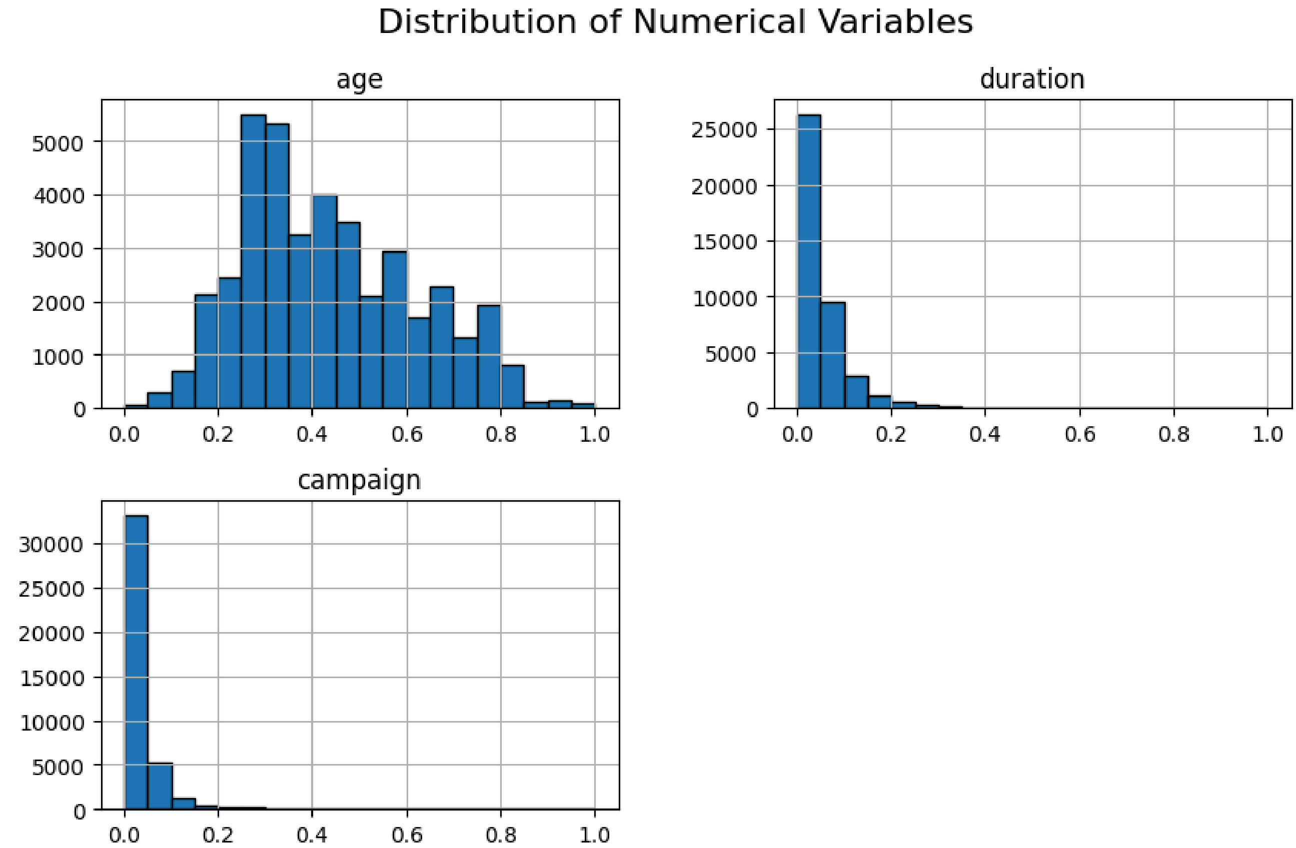
5. visualizations

Histogram for a numerical feature

distribution of numerical features

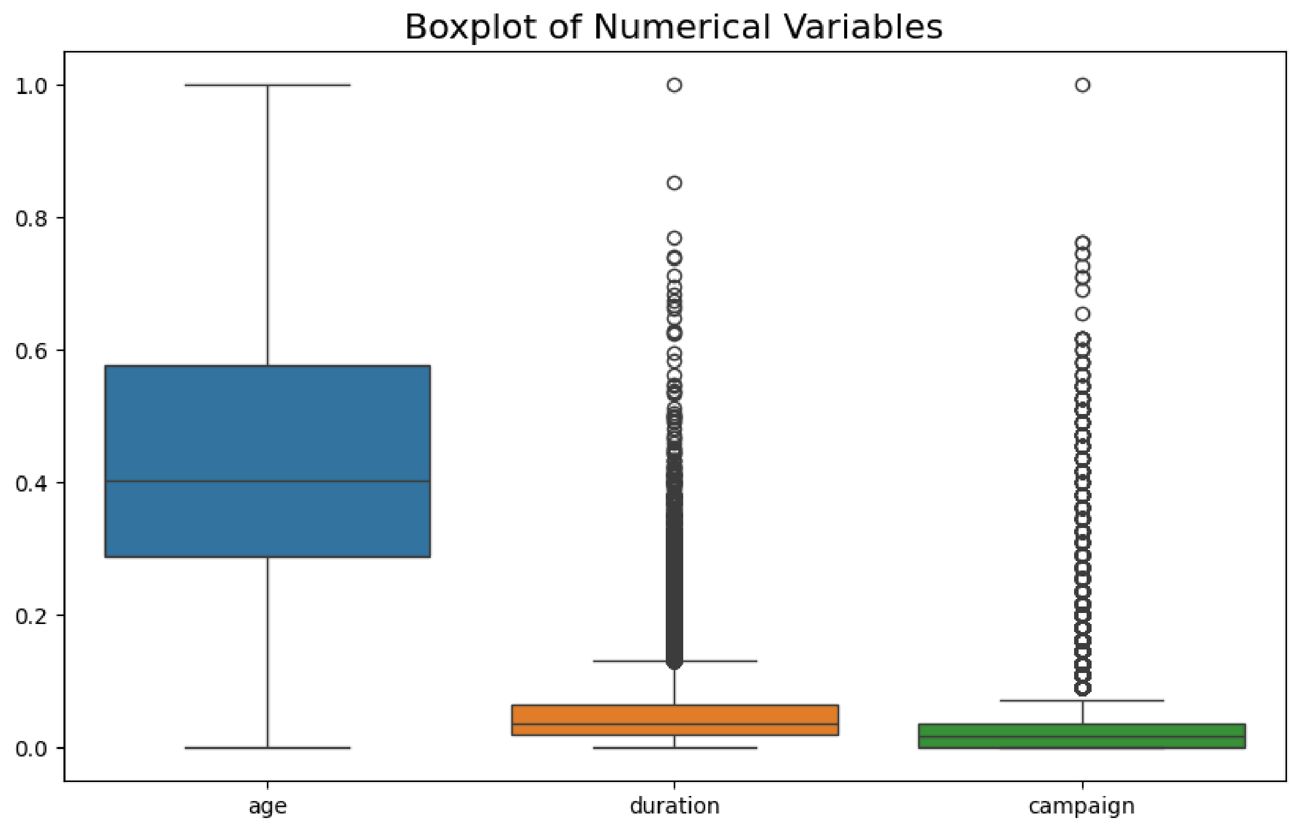
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Visualize the distribution of numerical features
data[['age', 'duration', 'campaign']].hist(figsize=(10, 6), bins=20, edgecolor='black')
plt.suptitle('Distribution of Numerical Variables', fontsize=16)
plt.show()
```



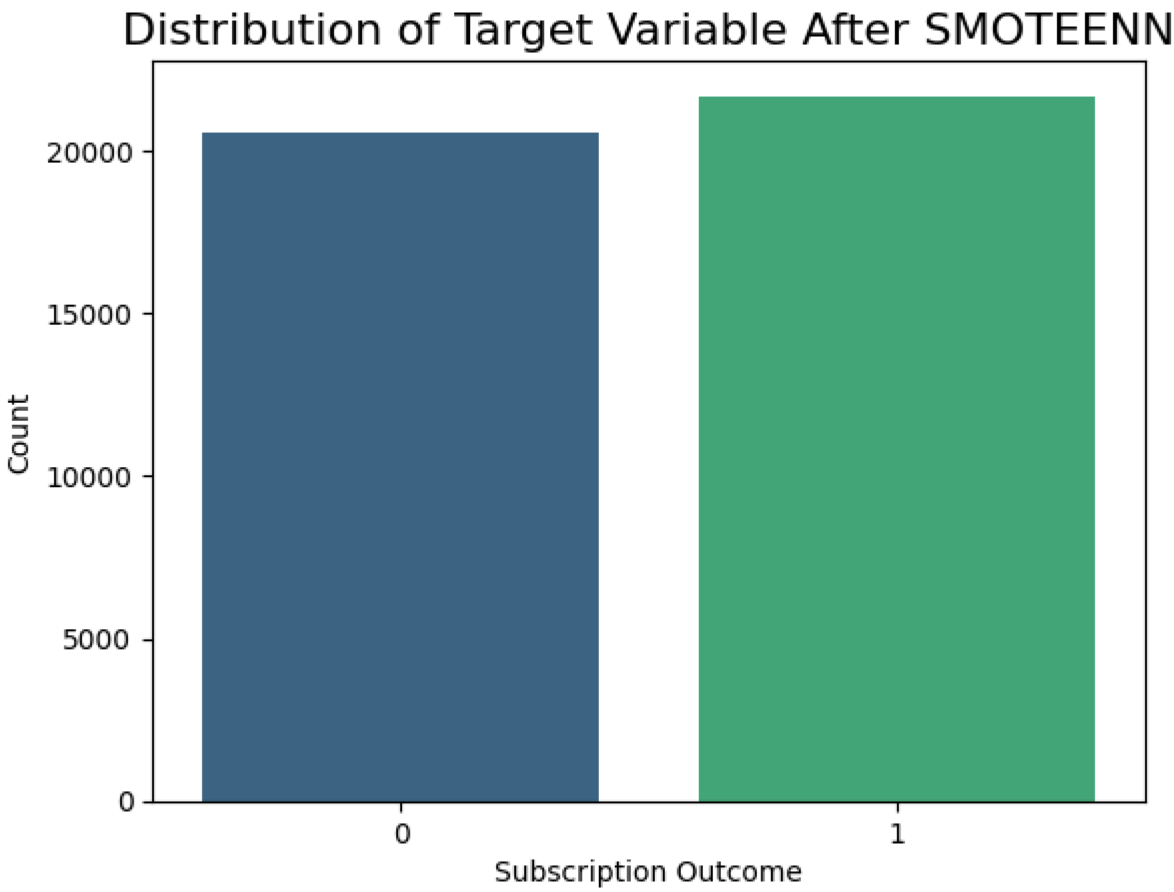
Boxplot for numerical variables

```
# Boxplot for numerical variables
plt.figure(figsize=(10, 6))
sns.boxplot(data=data[['age', 'duration', 'campaign']])
plt.title('Boxplot of Numerical Variables', fontsize=16)
plt.show()
```



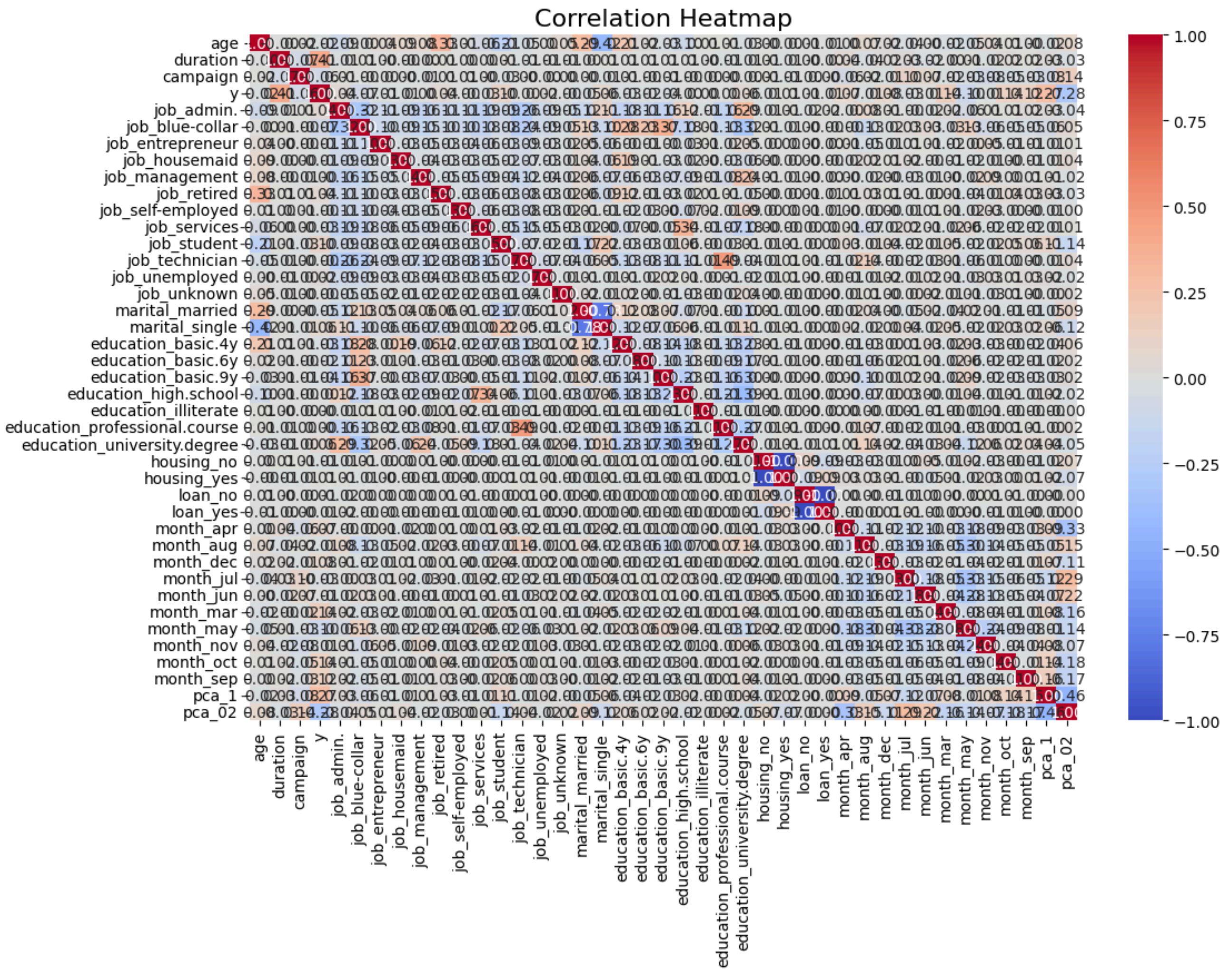
Bar plot for the target variable

```
# Bar plot for the target variable after SMOTEENN
sns.countplot(x=y_smoteenn, palette='viridis')
plt.title('Distribution of Target Variable After SMOTEENN', fontsize=16)
plt.xlabel('Subscription Outcome')
plt.ylabel('Count')
plt.show()
```



correlation matrix

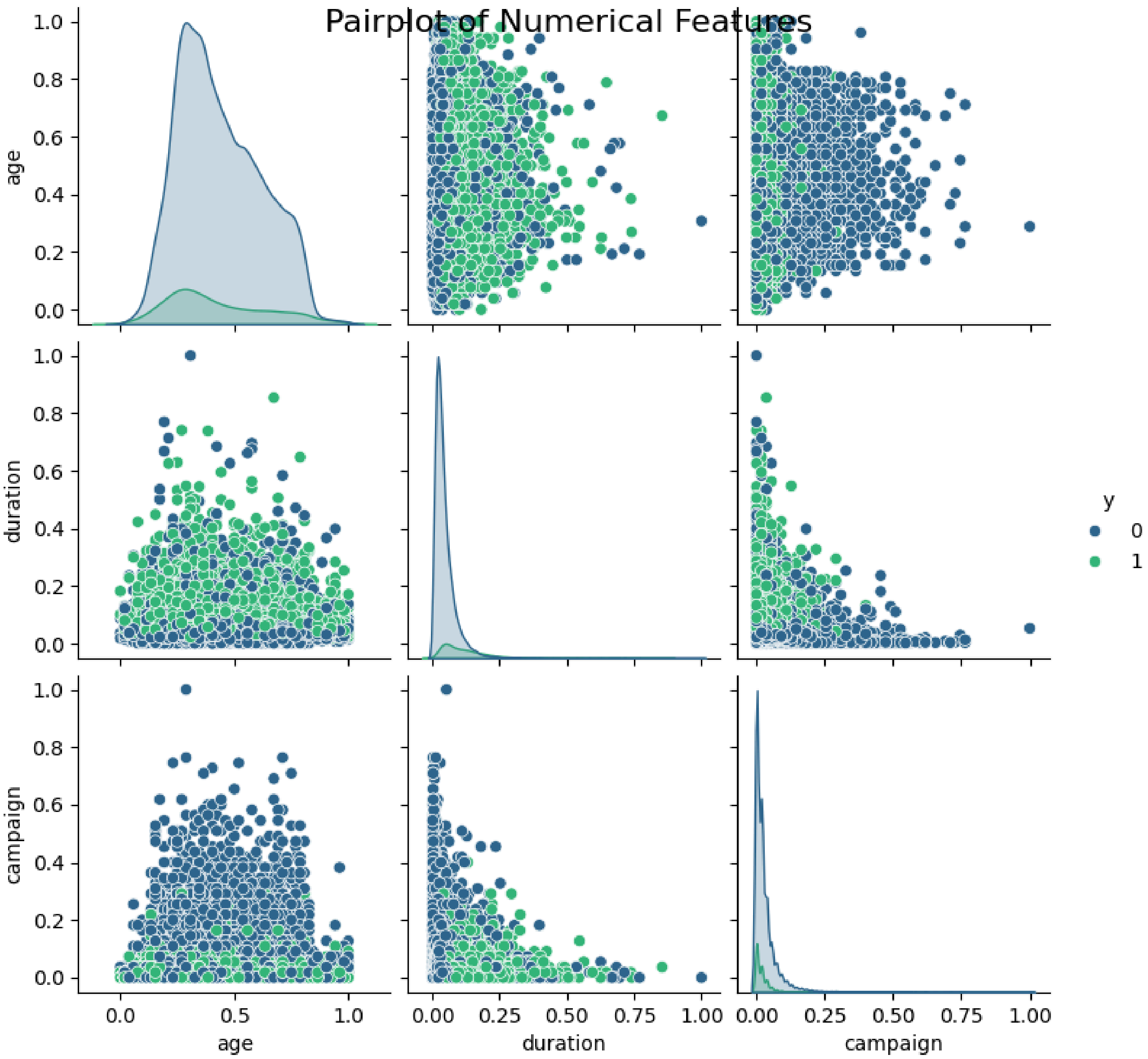
```
# Calculate the correlation matrix for numerical features
correlation_matrix = data.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```



Pair plot

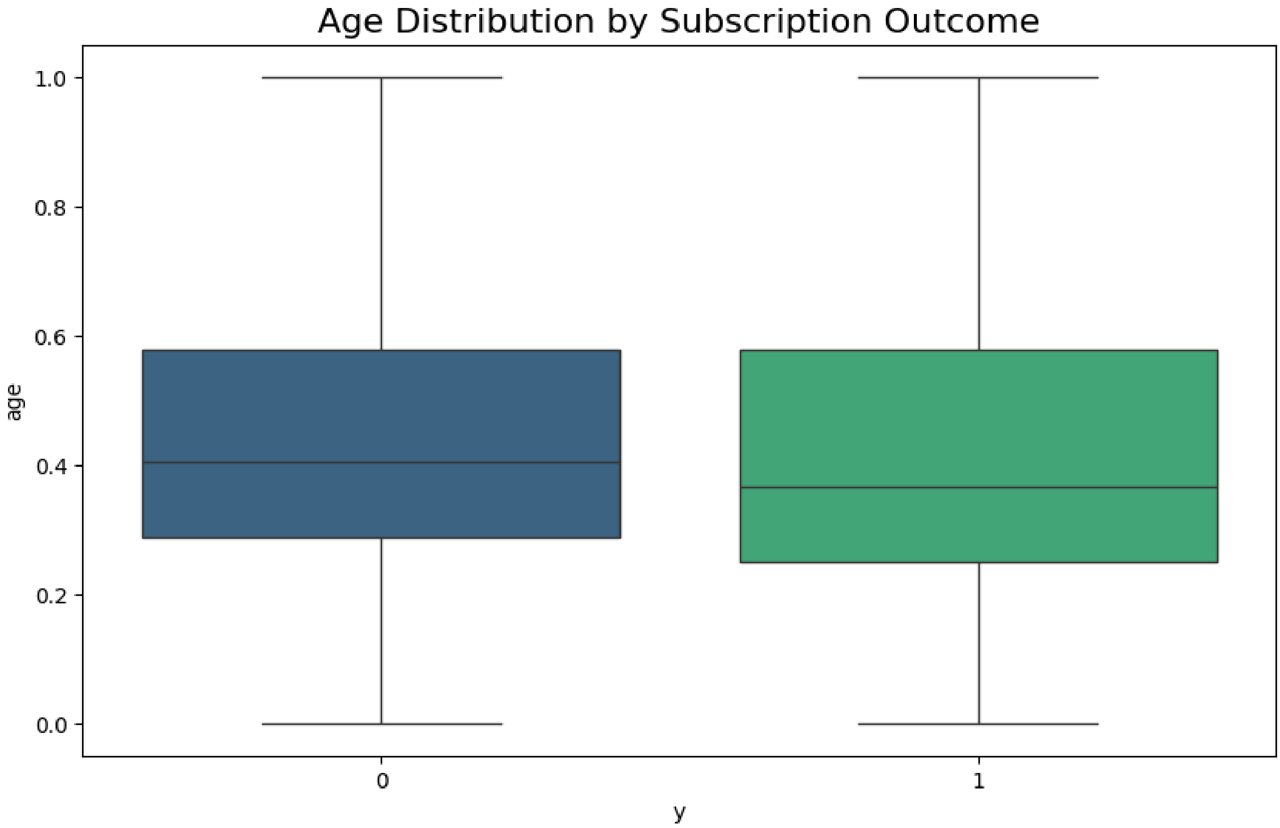
```
# Pair plot for continuous variables
sns.pairplot(data[['age', 'duration', 'campaign', 'y']], hue='y', palette='viridis')
```

```
plt.suptitle('Pairplot of Numerical Features', fontsize=16)
plt.show()
```



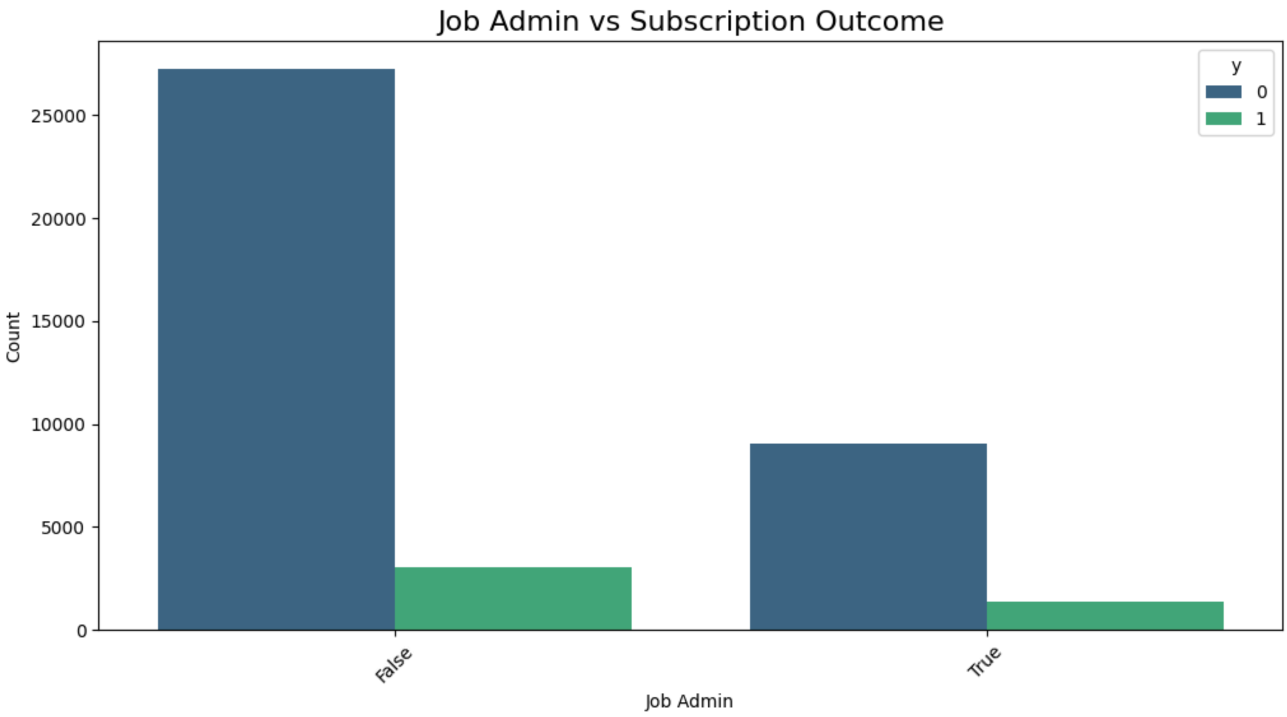
Boxplot of age

```
# Boxplot of age by target variable (y)
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='y', y='age', palette='viridis')
plt.title('Age Distribution by Subscription Outcome', fontsize=16)
plt.show()
```



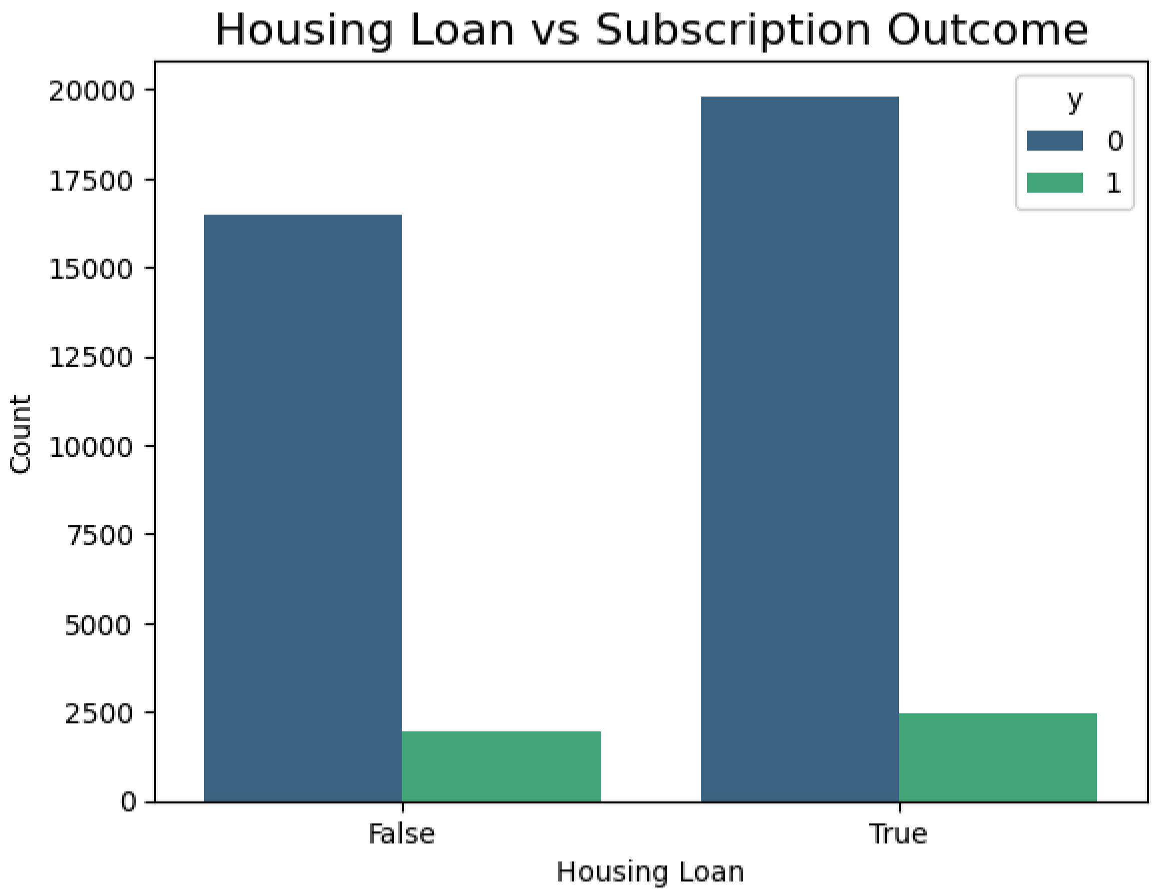
plot for 'job' against 'y'

```
# Count plot for 'job' against 'y'
plt.figure(figsize=(12, 6))
sns.countplot(data=data, x='job_admin.', hue='y', palette='viridis')
plt.title('Job Admin vs Subscription Outcome', fontsize=16)
plt.xlabel('Job Admin')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



Count plot for housing loans

```
# Count plot for housing loans
sns.countplot(data=data, x='housing_yes', hue='y', palette='viridis')
plt.title('Housing Loan vs Subscription Outcome', fontsize=16)
plt.xlabel('Housing Loan')
plt.ylabel('Count')
plt.show()
```



7. Random Forest Model

```
X_train.columns
```

```
Index(['age', 'duration', 'campaign', 'job_admin.', 'job_blue-collar',
       'job_entrepreneur', 'job_housemaid', 'job_management', 'job_retired',
       'job_self-employed', 'job_services', 'job_student', 'job_technician',
       'job_unemployed', 'job_unknown', 'marital_married', 'marital_single',
       'education_basic.4y', 'education_basic.6y', 'education_basic.9y',
       'education_high.school', 'education_illiterate',
       'education_professional.course', 'education_university.degree',
       'housing_no', 'housing_yes', 'loan_no', 'loan_yes', 'month_apr',
       'month_aug', 'month_dec', 'month_jul', 'month_jun', 'month_mar',
       'month_may', 'month_nov', 'month_oct', 'month_sep', 'pca_1', 'pca_02'],
      dtype='object')
```

```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predict on the test set
```

```
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

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

           0       0.99      0.97      0.98      4112
           1       0.97      0.99      0.98      4339

    accuracy                           0.98      8451
   macro avg       0.98      0.98      0.98      8451
weighted avg       0.98      0.98      0.98      8451
```

8. Neural Network Model

```
# Import necessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import classification_report, accuracy_score

# Define the Neural Network architecture
nn_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

# Compile the model
nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = nn_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_tes

# Evaluate the model
y_pred_nn = (nn_model.predict(X_test) > 0.5).astype("int32")
print("Neural Network Classification Report:")
print(classification_report(y_test, y_pred_nn))
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7939 - loss: 0.4248 - val_ac
Epoch 2/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9455 - loss: 0.1551 - val_ac
Epoch 3/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9527 - loss: 0.1346 - val_ac
Epoch 4/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9559 - loss: 0.1237 - val_ac
```

```
Epoch 5/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9573 - loss: 0.1154 - val_ac
Epoch 6/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9563 - loss: 0.1171 - val_ac
Epoch 7/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9622 - loss: 0.1064 - val_ac
Epoch 8/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9613 - loss: 0.1060 - val_ac
Epoch 9/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9639 - loss: 0.0995 - val_ac
Epoch 10/10
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9635 - loss: 0.0987 - val_ac
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Neural Network Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97      4112
           1       0.95      0.98      0.97      4339

    accuracy                           0.97      8451
   macro avg       0.97      0.97      0.97      8451
weighted avg       0.97      0.97      0.97      8451
```

9. Hyperparameter Tuning

9.1 Random Forest Tuning

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Define the parameter grid for Random Forest
param_dist = {
    'n_estimators': [50, 100, 150, 200, 250],           # Number of trees
    'max_depth': [None, 10, 20, 30, 40],                 # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],                     # Minimum samples required to sp
    'min_samples_leaf': [1, 2, 4],                       # Minimum samples required at ea
    'bootstrap': [True, False]                            # Whether to use bootstrapping
}

# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Perform RandomizedSearchCV for faster hyperparameter tuning
random_search = RandomizedSearchCV(estimator=rf_model,
                                   param_distributions=param_dist,
                                   n_iter=100,         # Number of random combinations to
                                   cv=5,               # Number of cross-validation folds
                                   scoring='accuracy',
                                   verbose=2,
```

```
                                   n_jobs=-1,         # Use all available CPUs
                                   random_state=42)

# Fit the model using the sampled training data
random_search.fit(X_train, y_train)

# Best parameters and model from the search
print("Best parameters for Random Forest:", random_search.best_params_)
best_rf_model = random_search.best_estimator_

# Predict on the test set using the best model
y_pred_rf = best_rf_model.predict(X_test)

# Evaluate the tuned model
print("Tuned Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-283-d282a7c69f67> in <cell line: 28>()
     26
     27 # Fit the model using the sampled training data
---> 28 random_search.fit(X_train, y_train)
     29
     30 # Best parameters and model from the search

                          7 frames
/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in _retrieve(self)
   1760                     (self._jobs[0].get_status(
   1761                         timeout=self.timeout) == TASK_PENDING)):
-> 1762                 time.sleep(0.01)
   1763                 continue
   1764

KeyboardInterrupt:
```

9.2 Neural Network Tuning

```
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# Function to create the model for KerasClassifier
def create_nn_model(optimizer='adam', activation='relu', units=64):
    model = Sequential([
        Dense(units, activation=activation, input_shape=(X_train.shape[1],)),
        Dense(32, activation=activation),
        Dense(1, activation='sigmoid')  # Binary classification
    ])

    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

```
    return model

# Wrap the model using KerasClassifier
model = KerasClassifier(build_fn=create_nn_model, epochs=10, batch_size=32, verbose=1)

# Define the parameter grid
param_grid = {
    'optimizer': ['adam', 'rmsprop'],  # Optimizer to test
    'activation': ['relu', 'tanh'],    # Activation function to test
    'units': [32, 64, 128],            # Number of units in the first layer
    'batch_size': [16, 32],            # Batch size to test
    'epochs': [10, 20]                 # Number of epochs
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, n_jobs=-1, verbc

# Fit the grid search
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and score
print("Best parameters for Neural Network:", grid_search.best_params_)
print("Best score for Neural Network:", grid_search.best_score_)

# Evaluate on the test set using the best model
best_nn_model = grid_search.best_estimator_

# Predict and evaluate the model
y_pred_nn = (best_nn_model.predict(X_test) > 0.5).astype("int32")
print("Tuned Neural Network Classification Report:")
print(classification_report(y_test, y_pred_nn))
```

10. Model Evaluation

Random Forest Model - Model Evaluation

1. Classification Report (Precision, Recall, F1-Score, Support)

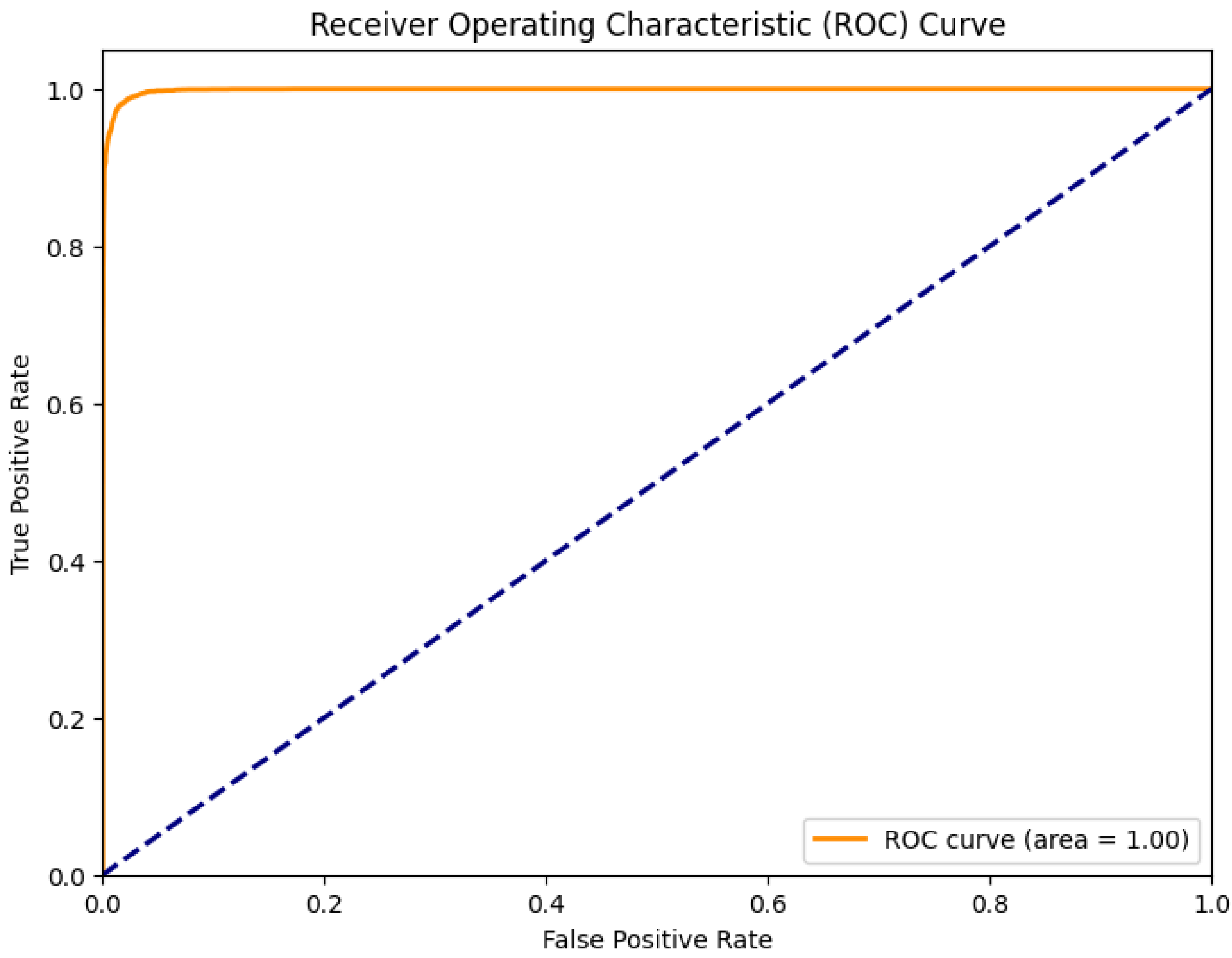
```
from sklearn.metrics import classification_report

# Evaluate the model with precision, recall, f1-score, and support
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

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

```
           0       0.99      0.97      0.98      4112
           1       0.97      0.99      0.98      4339

    accuracy                           0.98      8451
   macro avg       0.98      0.98      0.98      8451
weighted avg       0.98      0.98      0.98      8451
```

2. ROC Curve

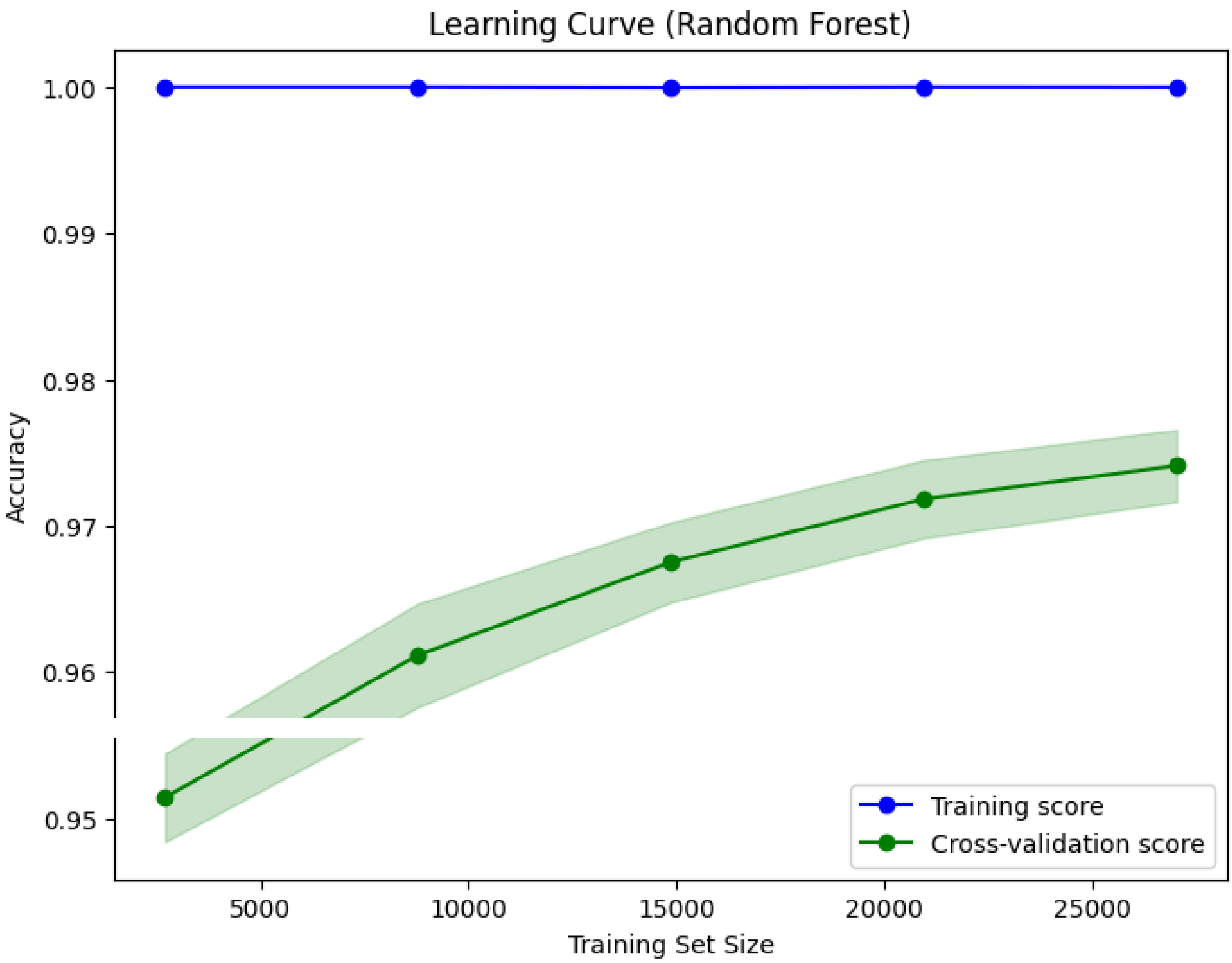
```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model with training data
rf_model.fit(X_train, y_train)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, rf_model.predict_proba(X_test)[:, 1])

# Compute AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



3. Underfitting and Overfitting Check using Learning Curves

```
from sklearn.model_selection import learning_curve
import numpy as np

# Generate learning curves
train_sizes, train_scores, test_scores = learning_curve(
    rf_model, X_train, y_train, cv=5, n_jobs=-1,
    train_sizes=np.linspace(0.1, 1.0, 5), scoring='accuracy')

# Calculate mean and standard deviation
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot the learning curves
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', label='Training score')
plt.plot(train_sizes, test_mean, color='green', marker='o', label='Cross-validation score

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='blue
```

```
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color='green',

plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve (Random Forest)')
plt.legend(loc='best')
plt.show()
```



4.training and testing error (MSE)