project-bank-term-deposit

January 11, 2024

BANK TERM DEPOSIT PREDICTION

This dataset, titled Direct Marketing Campaigns for Bank Term Deposits, is a collection of data related to the direct marketing campaigns conducted by a Portuguese banking institution. These campaigns primarily involved phone calls with customers, and the objective was to determine whether or not a customer would subscribe to a term deposit offered by the bank.

The dataset contains various features that provide insights into customer attributes and campaign outcomes. These features include:

```
* Age: The age of the customer.
* Job: The occupation of the customer.
* Marital Status: The marital status of the customer.
* Education: The education level of the customer.
* Default: Whether or not the customer has credit in default.
* Balance: The balance of the customer's account.
* Housing Loan: Whether or not the customer has a housing loan.
* Loan: Whether the customer has a loan or not
* Contact Communication Type: The method used to contact the customer(e.g., telephone, cellula
* Day: The day of the month when the last contact with the customers was made.
* Month: Last contact month of year
* Duration: The duration (in seconds) of the last contact with customers during a campaign.
* Campaign Contacts Count: Number of contacts performed during this campaign for each customer
* Pdays : number days passed since previously contacted form previous camapign
* Previous: Number of contacts performed before this campaign and for this client.
* Poutcome : outcome from previous marketing campaign
* y: Has the client subscribed a term deposit
```

The purpose behind this dataset is to train a predictive model that can determine if a given customer will subscribe to a term deposit based on these various features.

In addition to training data, there is also test data included in this dataset. This test data can be used to evaluate how well our trained predictive model performs when applied to new, unseen instances.

By utilizing this dataset and applying machine learning techniques, businesses in similar domains can better understand their target audience and optimize their marketing efforts towards potential subscribers who are more likely to respond positively to these campaigns.

Importing Required Libraries

[1677]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading Datasets

[1678]:
```python
df_train=pd.read_csv('/home/saniga/Documents/train_bank.csv')
df_test=pd.read_csv('/home/saniga/Documents/test_bank.csv')
```

[1679]:
```python
#Train dataset
df_train
```

[1679]:
```
       age           job  marital  education default  balance housing loan  \
0       58    management  married   tertiary      no     2143     yes   no
1       44    technician   single  secondary      no       29     yes   no
2       33  entrepreneur  married  secondary      no        2     yes  yes
3       47   blue-collar  married    unknown      no     1506     yes   no
4       33       unknown   single    unknown      no        1      no   no
…     …           …        …        …          …       …      …   …
45206   51    technician  married   tertiary      no      825      no   no
45207   71       retired divorced    primary      no     1729      no   no
45208   72       retired  married  secondary      no     5715      no   no
45209   57   blue-collar  married  secondary      no      668      no   no
45210   37  entrepreneur  married  secondary      no     2971      no   no

         contact  day month  duration  campaign  pdays  previous poutcome    y
0        unknown    5   may       261         1     -1         0  unknown   no
1        unknown    5   may       151         1     -1         0  unknown   no
2        unknown    5   may        76         1     -1         0  unknown   no
3        unknown    5   may        92         1     -1         0  unknown   no
4        unknown    5   may       198         1     -1         0  unknown   no
…            …   …    …       …         …      …        …      …
45206   cellular   17   nov       977         3     -1         0  unknown  yes
45207   cellular   17   nov       456         2     -1         0  unknown  yes
45208   cellular   17   nov      1127         5    184         3  success  yes
45209  telephone   17   nov       508         4     -1         0  unknown   no
45210   cellular   17   nov       361         2    188        11    other   no

[45211 rows x 17 columns]
```

[1680]:
```python
print('Records:',df_train.shape[0],'\nColumns:',df_train.shape[1])
```

```
Records: 45211
Columns: 17
```

[1681]:

```
#Test dataset
df_test
```

[1681]:

	age	job	marital	education	default	balance	housing	loan	\
0	30	unemployed	married	primary	no	1787	no	no	
1	33	services	married	secondary	no	4789	yes	yes	
2	35	management	single	tertiary	no	1350	yes	no	
3	30	management	married	tertiary	no	1476	yes	yes	
4	59	blue-collar	married	secondary	no	0	yes	no	
…	…	…	…	…	…	…	…	…	
4516	33	services	married	secondary	no	-333	yes	no	
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	
4518	57	technician	married	secondary	no	295	no	no	
4519	28	blue-collar	married	secondary	no	1137	no	no	
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	

	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	cellular	19	oct	79	1	-1	0	unknown	no
1	cellular	11	may	220	1	339	4	failure	no
2	cellular	16	apr	185	1	330	1	failure	no
3	unknown	3	jun	199	4	-1	0	unknown	no
4	unknown	5	may	226	1	-1	0	unknown	no
…	…	…	…	…	…	…	…	..	
4516	cellular	30	jul	329	5	-1	0	unknown	no
4517	unknown	9	may	153	1	-1	0	unknown	no
4518	cellular	19	aug	151	11	-1	0	unknown	no
4519	cellular	6	feb	129	4	211	3	other	no
4520	cellular	3	apr	345	2	249	7	other	no

[4521 rows x 17 columns]

[1682]:

```
print('Records:',df_test.shape[0],'\nColumns:',df_test.shape[1])
```

```
Records: 4521
Columns: 17
```

[1683]:

```
#First 10 values
df_train.head(10)
```

[1683]:

	age	job	marital	education	default	balance	housing	loan	\
0	58	management	married	tertiary	no	2143	yes	no	
1	44	technician	single	secondary	no	29	yes	no	
2	33	entrepreneur	married	secondary	no	2	yes	yes	
3	47	blue-collar	married	unknown	no	1506	yes	no	
4	33	unknown	single	unknown	no	1	no	no	
5	35	management	married	tertiary	no	231	yes	no	
6	28	management	single	tertiary	no	447	yes	yes	

```
7   42  entrepreneur  divorced   tertiary     yes        2      yes   no
8   58       retired   married    primary      no      121      yes   no
9   43    technician    single  secondary      no      593      yes   no

   contact  day month  duration  campaign  pdays  previous poutcome   y
0  unknown    5   may       261         1     -1         0  unknown  no
1  unknown    5   may       151         1     -1         0  unknown  no
2  unknown    5   may        76         1     -1         0  unknown  no
3  unknown    5   may        92         1     -1         0  unknown  no
4  unknown    5   may       198         1     -1         0  unknown  no
5  unknown    5   may       139         1     -1         0  unknown  no
6  unknown    5   may       217         1     -1         0  unknown  no
7  unknown    5   may       380         1     -1         0  unknown  no
8  unknown    5   may        50         1     -1         0  unknown  no
9  unknown    5   may        55         1     -1         0  unknown  no
```

[1684]:
```python
#Last 10 values
df_train.tail(10)
```

[1684]:
```
       age           job   marital  education default  balance housing loan  \
45201   53    management   married   tertiary      no      583      no   no
45202   34        admin.    single  secondary      no      557      no   no
45203   23       student    single   tertiary      no      113      no   no
45204   73       retired   married  secondary      no     2850      no   no
45205   25    technician    single  secondary      no      505      no  yes
45206   51    technician   married   tertiary      no      825      no   no
45207   71       retired  divorced    primary      no     1729      no   no
45208   72       retired   married  secondary      no     5715      no   no
45209   57   blue-collar   married  secondary      no      668      no   no
45210   37  entrepreneur   married  secondary      no     2971      no   no

         contact  day month  duration  campaign  pdays  previous poutcome    y
45201   cellular   17   nov       226         1    184         4  success  yes
45202   cellular   17   nov       224         1     -1         0  unknown  yes
45203   cellular   17   nov       266         1     -1         0  unknown  yes
45204   cellular   17   nov       300         1     40         8  failure  yes
45205   cellular   17   nov       386         2     -1         0  unknown  yes
45206   cellular   17   nov       977         3     -1         0  unknown  yes
45207   cellular   17   nov       456         2     -1         0  unknown  yes
45208   cellular   17   nov      1127         5    184         3  success  yes
45209  telephone   17   nov       508         4     -1         0  unknown   no
45210   cellular   17   nov       361         2    188        11    other   no
```

[1685]:
```python
#Column of train dataset
df_train.columns
```

[1685]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y'],
      dtype='object')
```

[1686]:
```
#Datatype of train dataset
df_train.dtypes
```

[1686]:
```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
y            object
dtype: object
```

[1687]:
```
#Train dataset description
print(df_train.describe())
```

```
                age        balance           day      duration      campaign  \
count  45211.000000   45211.000000  45211.000000  45211.000000  45211.000000
mean      40.936210    1362.272058     15.806419    258.163080      2.763841
std       10.618762    3044.765829      8.322476    257.527812      3.098021
min       18.000000   -8019.000000      1.000000      0.000000      1.000000
25%       33.000000      72.000000      8.000000    103.000000      1.000000
50%       39.000000     448.000000     16.000000    180.000000      2.000000
75%       48.000000    1428.000000     21.000000    319.000000      3.000000
max       95.000000  102127.000000     31.000000   4918.000000     63.000000

              pdays      previous
count  45211.000000  45211.000000
mean      40.197828      0.580323
std      100.128746      2.303441
min       -1.000000      0.000000
25%       -1.000000      0.000000
50%       -1.000000      0.000000
```

```
75%       -1.000000     0.000000
max      871.000000   275.000000
```

[1688]:
```python
#Number of unique values in train dataset
df_train.nunique()
```

[1688]:
```
age            77
job            12
marital         3
education       4
default         2
balance      7168
housing         2
loan            2
contact         3
day            31
month          12
duration     1573
campaign       48
pdays         559
previous       41
poutcome        4
y               2
dtype: int64
```

[1689]:
```python
#Train dataset information
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
```

```
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

[1690]:
```
#Checking null values in train dataset
df_train.isna().sum()
```

[1690]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

[1691]:
```
#Combining train dataset and test dataset.
df = pd.concat([df_train,df_test],axis=0,ignore_index=True)
```

Data Visualization

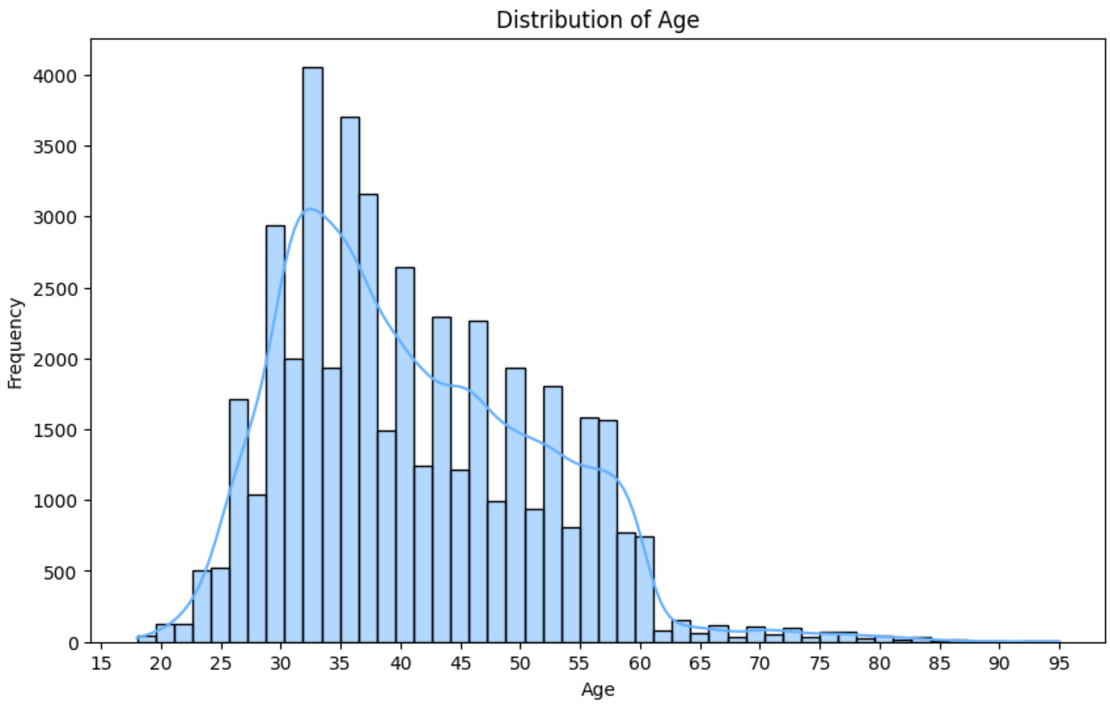
Age Feature

Age of the Customer

[1692]:
```
# Get statistical analysis
df_train['age'].describe()
```

[1692]:
```
count    45211.000000
mean        40.936210
std         10.618762
min         18.000000
25%         33.000000
50%         39.000000
75%         48.000000
max         95.000000
Name: age, dtype: float64
```

[1693]:
```python
# Define figure size
plt.figure(figsize=(10, 6))

# Plot the histogram
plot = sns.histplot(df_train['age'], bins=50, kde=True, color = '#66B2FF')

# Add labels and title
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.xticks([i for i in range(15, 100, 5)])

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



That is,most prevalent age range is (30 - 47)

Job Feature

The occupation/employment status of the Customer.

[1694]:
```python
df['job'].value_counts()
```

[1694]:
```
job
blue-collar      10678
```

```
management       10427
technician        8365
admin.            5649
services          4571
retired           2494
self-employed     1762
entrepreneur      1655
unemployed        1431
housemaid         1352
student           1022
unknown            326
Name: count, dtype: int64
```

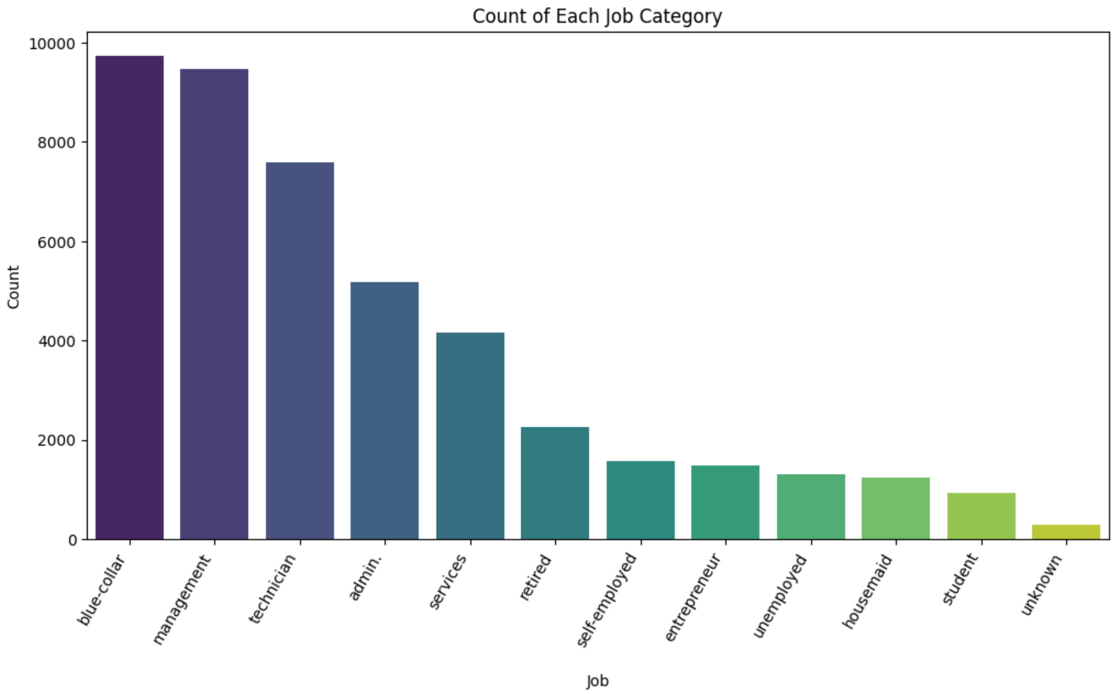
Renaming 'unknown' values with 'others'

[1695]:
```
df['job'] = df['job'].replace('unknown','others')
df['job'].value_counts()
```

[1695]:
```
job
blue-collar      10678
management       10427
technician        8365
admin.            5649
services          4571
retired           2494
self-employed     1762
entrepreneur      1655
unemployed        1431
housemaid         1352
student           1022
others             326
Name: count, dtype: int64
```

[1696]:
```
# Define counts
job_counts = df_train['job'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))

# Plot bar chart
sns.barplot(x=job_counts.index, y=job_counts.values,hue = job_counts.
 ↪index,legend = False, palette='viridis')

# Add labels and title
plt.title('Count of Each Job Category')
plt.xlabel('\nJob')
plt.ylabel('Count')
```

```
plt.xticks(rotation=60, ha='right')

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



Marital-Status Feature

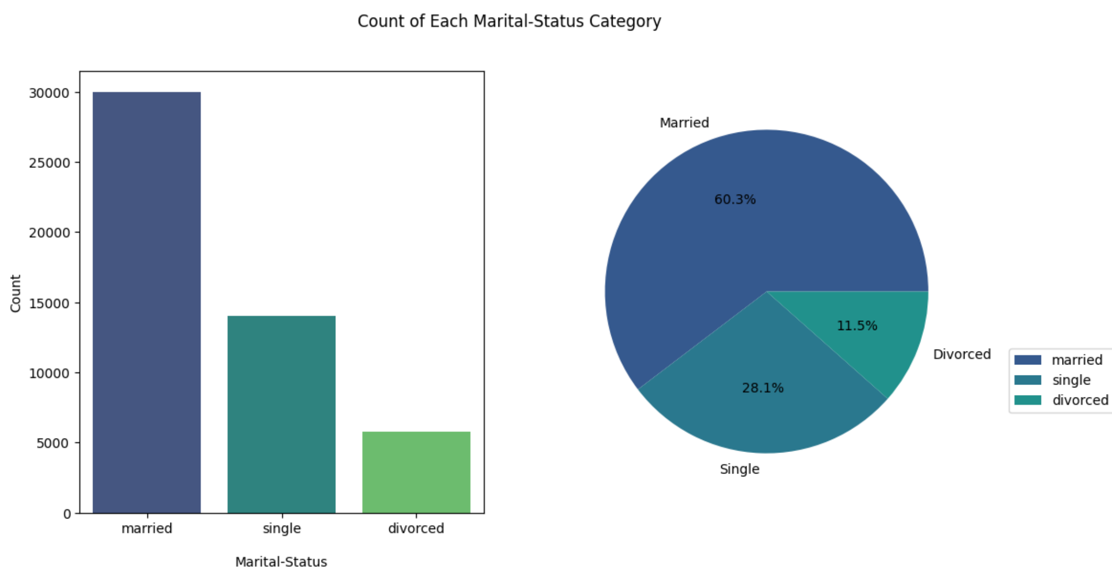
The marital status of the Customer.

[1697]:
```
df['marital'].value_counts()
```

[1697]:
```
marital
married     30011
single      13986
divorced     5735
Name: count, dtype: int64
```

[1698]:
```
# Define counts
marital_counts = df['marital'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))
plt.suptitle('Count of Each Marital-Status Category')

# Plot bar chart
```

```
plt.subplot(1,2,1)
sns.barplot(x = marital_counts.index, y = marital_counts.values,hue =␣
↪marital_counts.index,legend = False, palette = 'viridis')

# Add labels and title
plt.xlabel('\nMarital-Status')
plt.ylabel('Count')

plt.subplot(1,2,2)
palette = ['#35598E','#2A788E','#21918C']
plt.pie(marital_counts,labels = ['Married','Single','Divorced'],autopct = '%1.
↪1f%%',colors = palette)
plt.legend(marital_counts.index,loc=(1.1,0.2))

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



Educational Feature

The education level of the customer.

[1699]:
```
df['education'].value_counts()
```

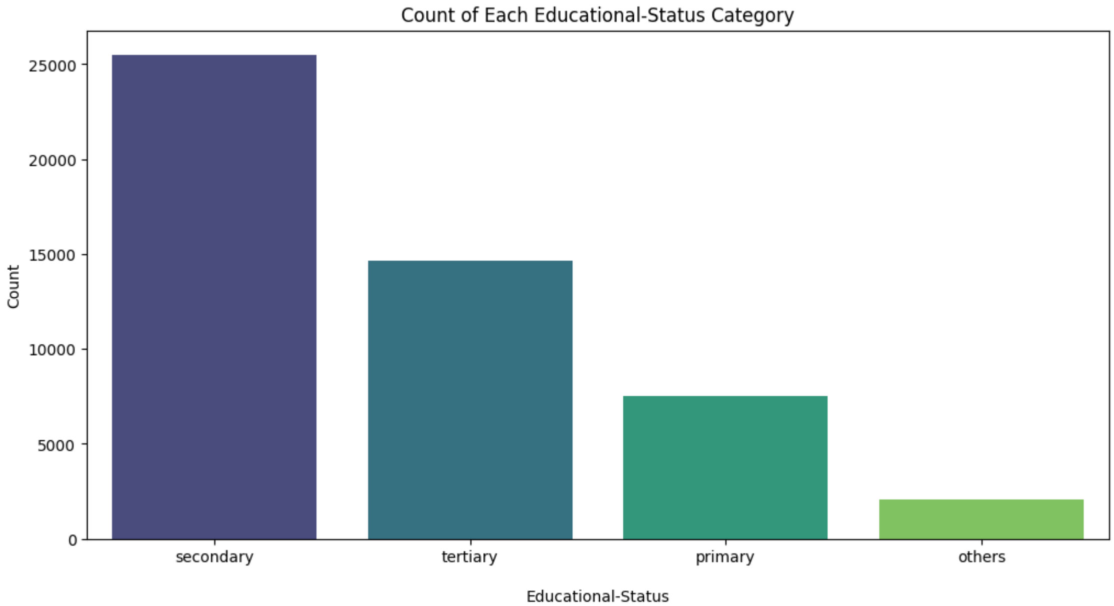
[1699]:
```
education
secondary    25508
tertiary     14651
primary       7529
unknown       2044
Name: count, dtype: int64
```

Rename "unknown" values with "others"

[1700]:
```python
df['education'] = df['education'].replace('unknown', 'others')
df['education'].value_counts()
```

[1700]:
```
education
secondary    25508
tertiary     14651
primary       7529
others        2044
Name: count, dtype: int64
```

[1701]:
```python
# Define Counts
education_counts = df['education'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))

# Plot bar chart
sns.barplot(x = education_counts.index, y = education_counts.values,hue =␣
↪education_counts.index,legend = False, palette = 'viridis')

# Add labels and title
plt.title('Count of Each Educational-Status Category')
plt.xlabel('\nEducational-Status')
plt.ylabel('Count')

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



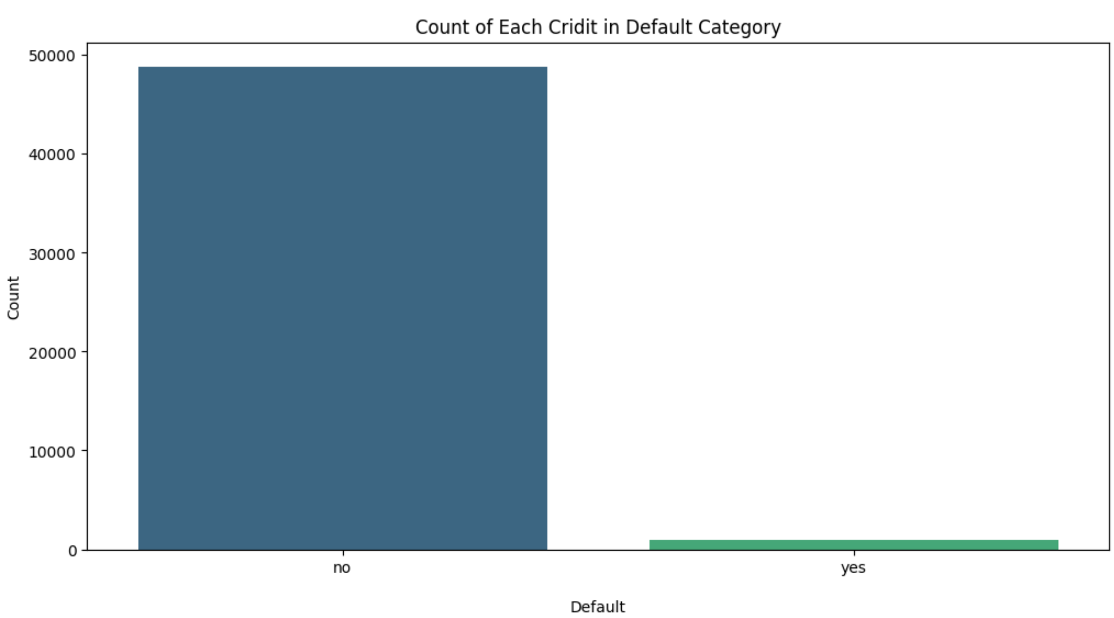
Credit in Default Feature

Whether the customer has credit in default or not.

[1702]:
```
df['default'].value_counts()
```

[1702]:
```
default
no     48841
yes      891
Name: count, dtype: int64
```

[1703]:
```
# Define counts
default_counts = df['default'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))

# Plot bar chart
sns.barplot(x=default_counts.index, y=default_counts.values,hue =␣
 ↪default_counts.index,legend = False, palette='viridis')

# Add labels and title
plt.title('Count of Each Cridit in Default Category')
plt.xlabel('\nDefault')
plt.ylabel('Count')

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



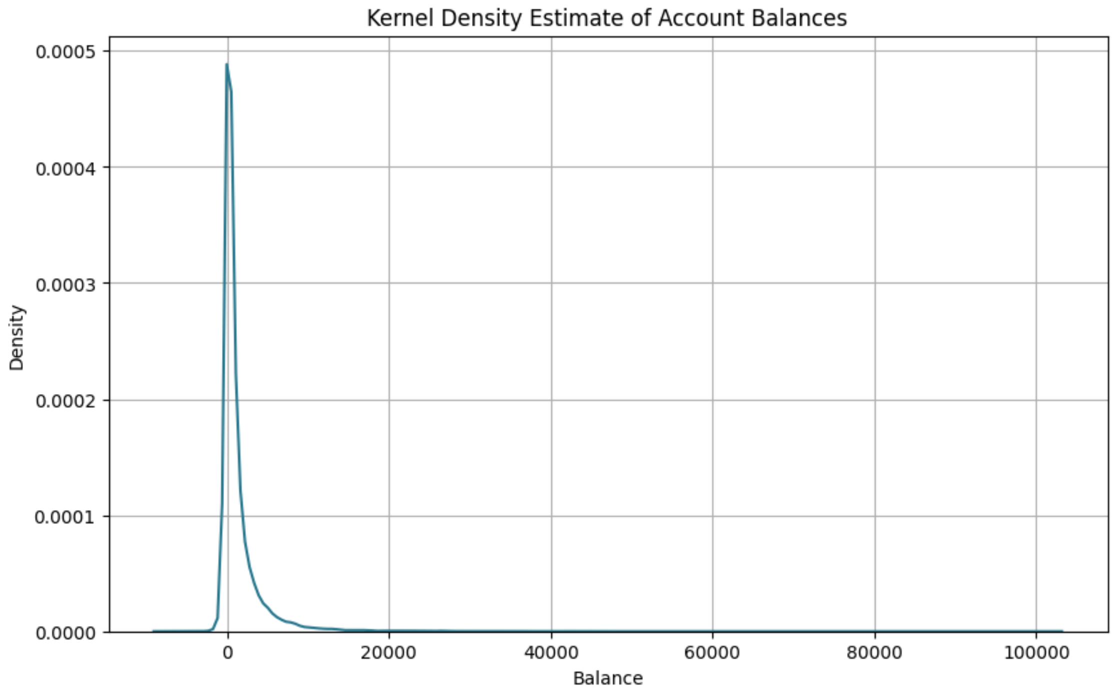
Balance Feature

The balance in the customer's account.

[1704]:
```
df['balance'].describe()
```

[1704]:
```
count     49732.000000
mean       1367.761562
std        3041.608766
min       -8019.000000
25%          72.000000
50%         448.000000
75%        1431.000000
max      102127.000000
Name: balance, dtype: float64
```

[1705]:
```
# Define figure size
plt.figure(figsize=(10, 6))

# Plot the histogram
sns.kdeplot(df['balance'], color = '#2A788E')

# Add labels and title
plt.title('Kernel Density Estimate of Account Balances')
plt.xlabel('Balance')
plt.ylabel('Density')

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



Check for balance under zero

[1706]:
```
df[df['balance'] <= 0]['balance'].count()
```

[1706]: 8003

Define the percentile threshold for outliers - 95%

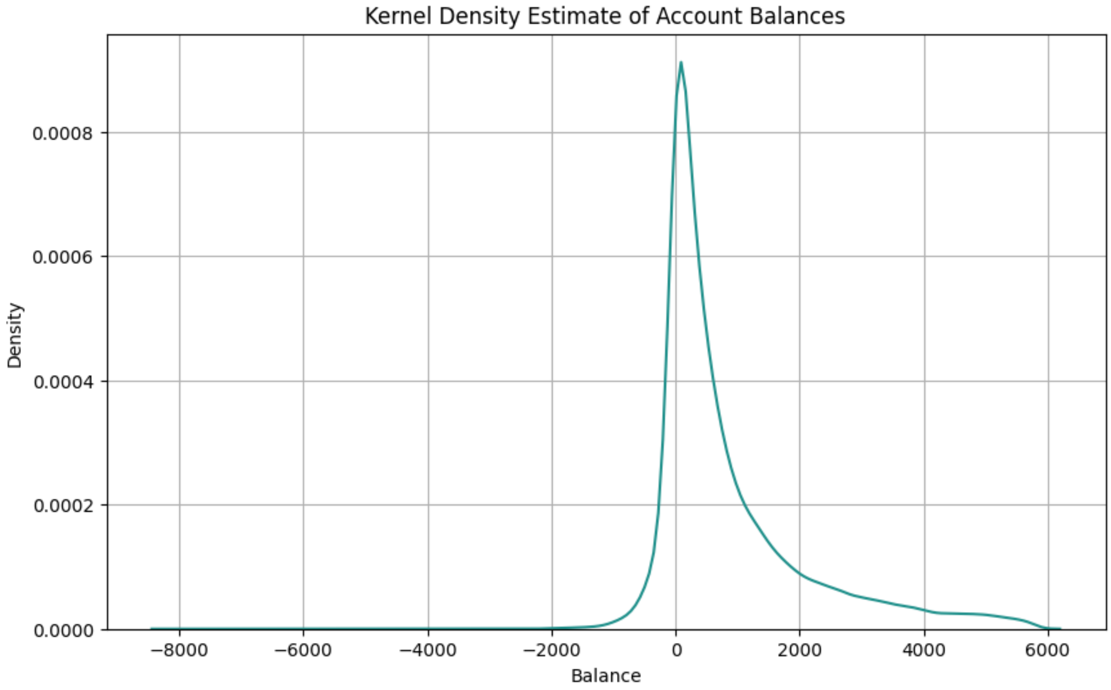
[1707]:
```
# Define the percentile threshold
percentile_threshold = 95

# Calculate the specified percentile
percentile_value = int(np.percentile(df['balance'], percentile_threshold))

# Identify potential outliers
outliers = df[df['balance'] > percentile_value]

print(f'{percentile_threshold}th Percentile Value: {percentile_value}')
print(f'Number of Potential Outliers: {len(outliers)}')
```

```
95th Percentile Value: 5798
Number of Potential Outliers: 2487
```

The maximum value of 102127 is considerably higher than the 95th percentile (5768). Hence drop the values that above 5768.

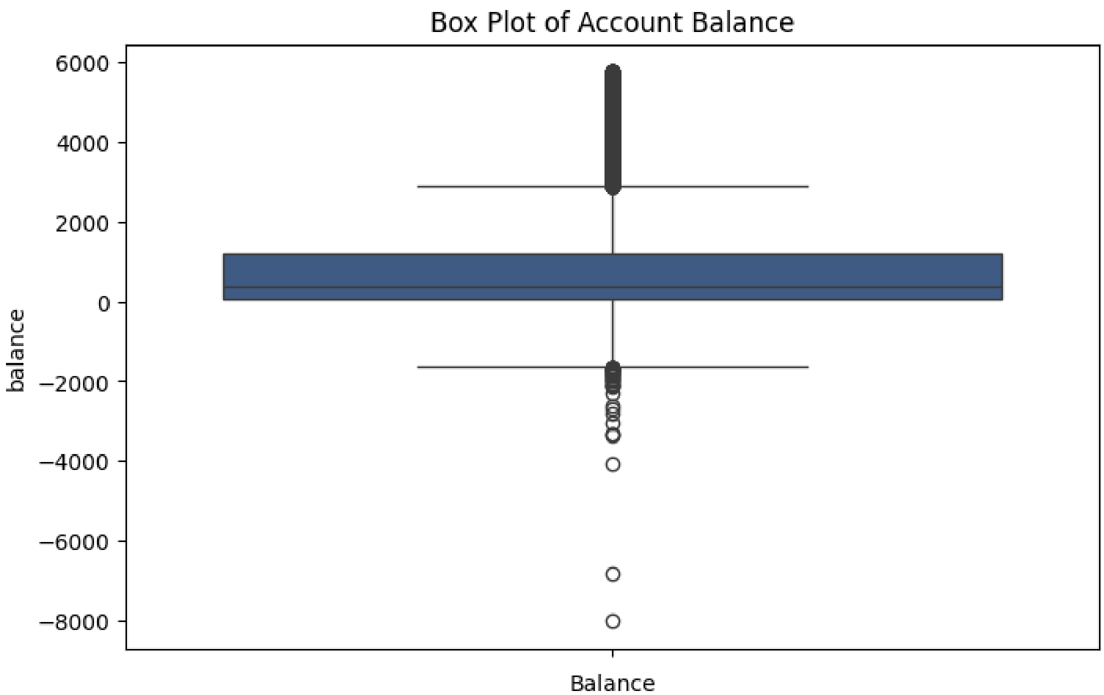
[1708]:
```python
df = df[df['balance'] <= 5768]
```

[1709]:
```python
# Define figure size
plt.figure(figsize = (10, 6))

# Plot the histogram
sns.kdeplot(df['balance'],color = '#21918C')

# Add labels and title
plt.title('Kernel Density Estimate of Account Balances')
plt.xlabel('Balance')
plt.ylabel('Density')

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



[1710]:
```python
# Define figure size
plt.figure(figsize = (8, 5))

# Plot the boxplot
sns.boxplot(df['balance'], color = '#35598E')

# Add labels and title
```

```
plt.title('Box Plot of Account Balance')
plt.xlabel('Balance')

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



Define the percentile threshold for outliers - 5%

[1711]:
```
# Define the percentile threshold
percentile_threshold = 5

# Calculate the specified percentile
percentile_value = int(np.percentile(df['balance'], percentile_threshold))

# Identify potential outliers
outliers = df[df['balance'] < percentile_value]

print(f'{percentile_threshold}th Percentile Value: {percentile_value}')
print(f'Number of Potential Outliers: {len(outliers)}')
```

```
5th Percentile Value: -191
Number of Potential Outliers: 2355
```

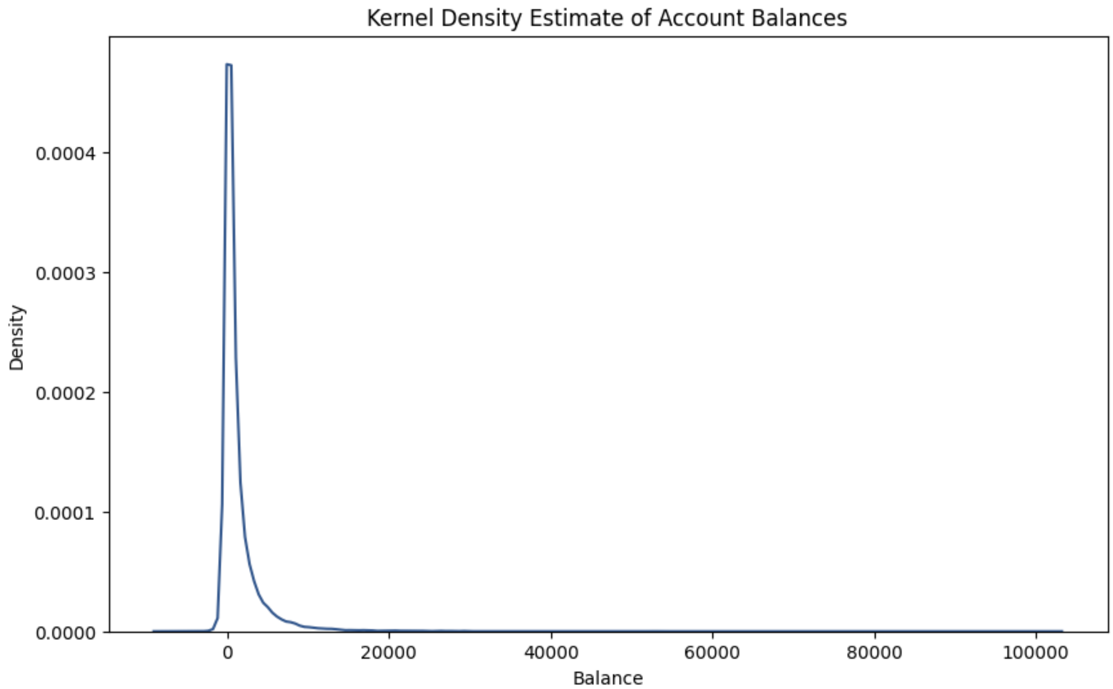
The minimum value of -8019 is considerably lower than the 5th percentile (-191). I'll drop the values that under -191.

[1712]:

```
df = df[df['balance'] > -191]
```

[1713]:

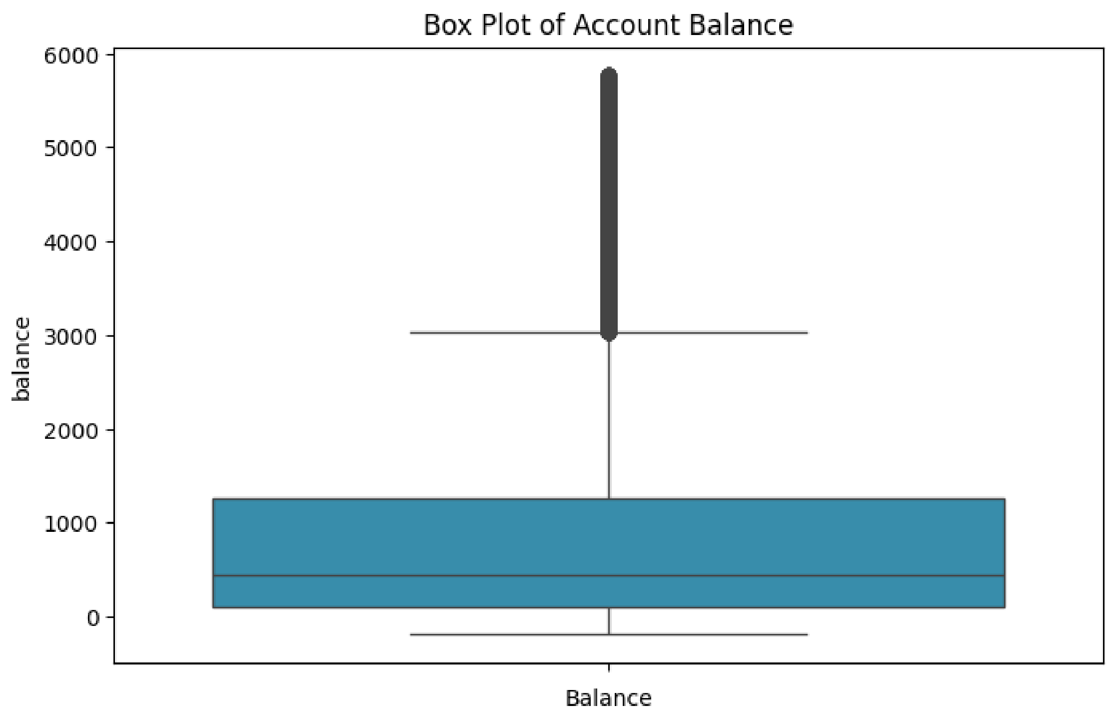
```
# Define figure size
plt.figure(figsize = (10, 6))

# Plot the histogram
sns.kdeplot(df_train['balance'], color = '#35598E')

# Add labels and title
plt.title('Kernel Density Estimate of Account Balances')
plt.xlabel('Balance')
plt.ylabel('Density')

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



[1714]:

```
# Define figure size
plt.figure(figsize=(8, 5))

# Plot the boxplot
sns.boxplot(df['balance'], color = '#2596be')

# Add labels and title
plt.title('Box Plot of Account Balance')
```

```
plt.xlabel('Balance')

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



Housing Loan Feature

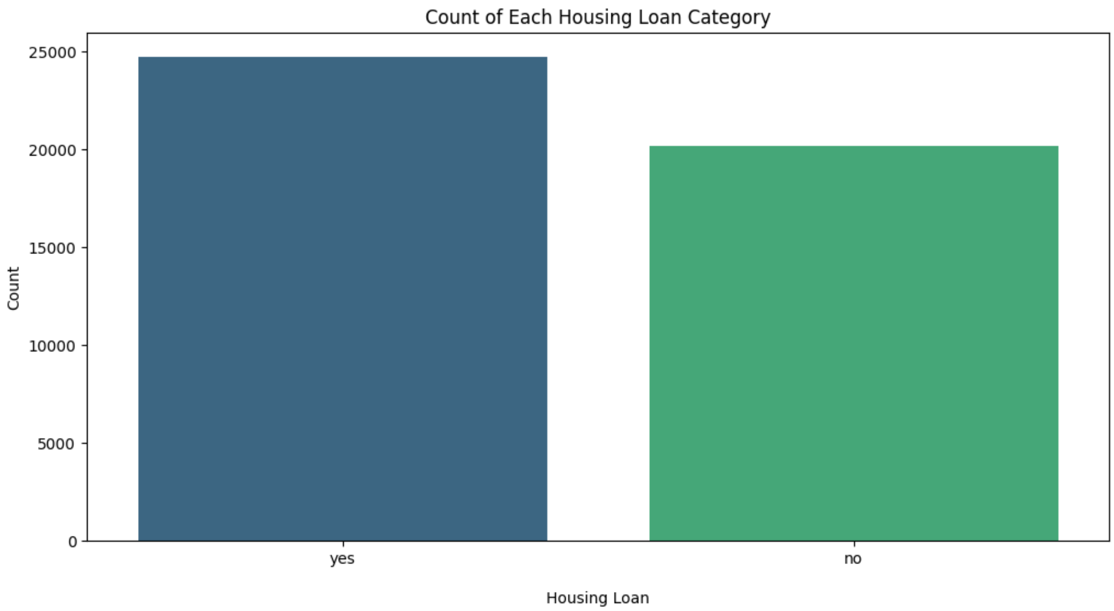
Whether the customer has a housing loan or not.

[1715]:
```
df['housing'].value_counts()
```

[1715]:
```
housing
yes    24718
no     20138
Name: count, dtype: int64
```

[1716]:
```
# Define counts
housing_counts = df['housing'].value_counts()

# Define figure size
plt.figure(figsize = (12, 6))

# Plot bar chart
sns.countplot(x = df['housing'],data = df,hue = df['housing'],legend = False,
  palette = 'viridis')
```

```
# Add labels and title
plt.title('Count of Each Housing Loan Category')
plt.xlabel('\nHousing Loan')
plt.ylabel('Count')

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



Loan Feature

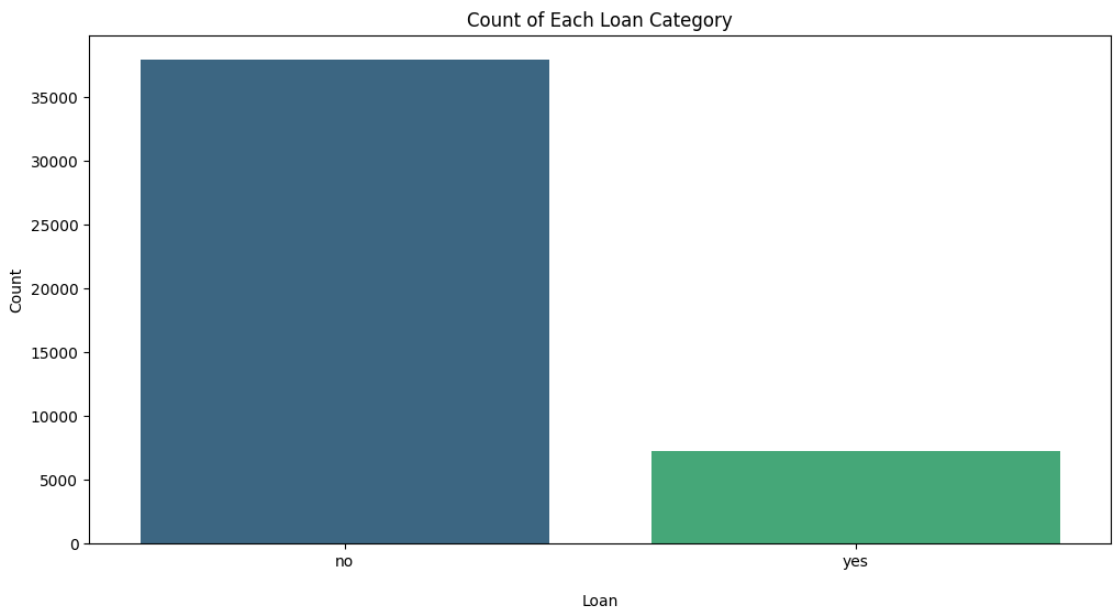
Whether the customer has a loan or not.

[1717]:
```
df['loan'].value_counts()
```

[1717]:
```
loan
no     37906
yes     6950
Name: count, dtype: int64
```

[1718]:
```
# Define counts
loan_counts = df_train['loan'].value_counts()

# Define figure size
plt.figure(figsize = (12, 6))

# Plot bar chart
```

```
sns.barplot(x = loan_counts.index, y = loan_counts.values,hue = loan_counts.
 ↪index,legend = False, palette = 'viridis')

# Add labels and title
plt.title('Count of Each Loan Category')
plt.xlabel('\nLoan')
plt.ylabel('Count')

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



Contact Feature

Type of communication used to contact customers

[1719]:
```
df['contact'].value_counts()
```

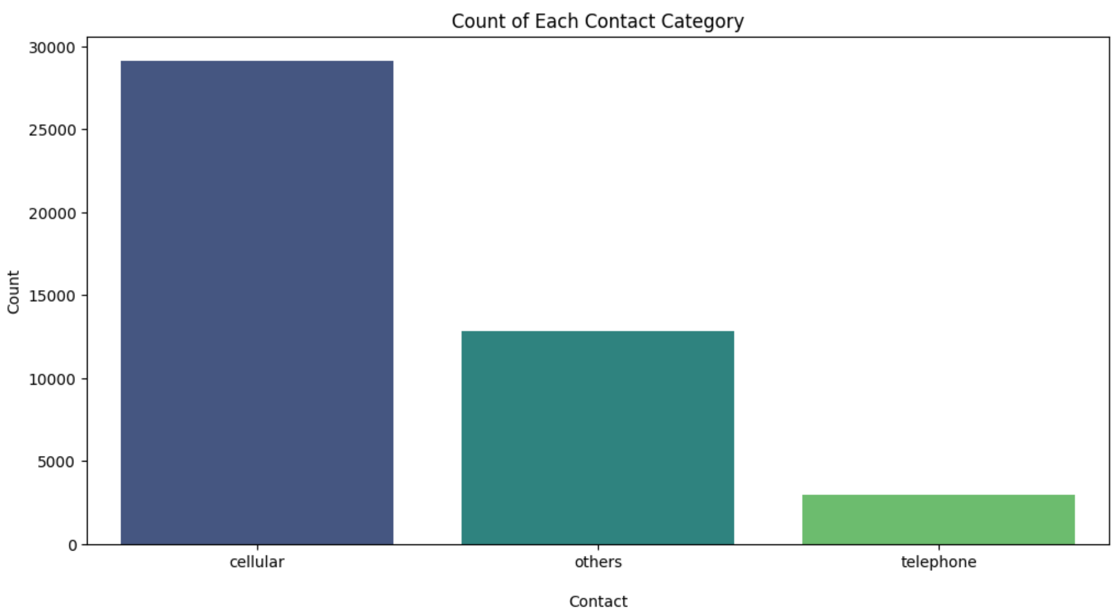
[1719]:
```
contact
cellular     29144
unknown      12796
telephone     2916
Name: count, dtype: int64
```

Rename "unknown" values with "others"

[1720]:
```
df['contact'] = df['contact'].replace('unknown', 'others')
df['contact'].value_counts()
```

[1720]:
```
contact
cellular     29144
others       12796
telephone     2916
Name: count, dtype: int64
```

[1721]:
```
# Define counts
contact_counts = df['contact'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))

# Plot bar chart
sns.barplot(x=contact_counts.index, y=contact_counts.values,hue =␣
↪contact_counts.index,legend = False, palette='viridis')

# Add labels and title
plt.title('Count of Each Contact Category')
plt.xlabel('\nContact')
plt.ylabel('Count')

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



Day Feature

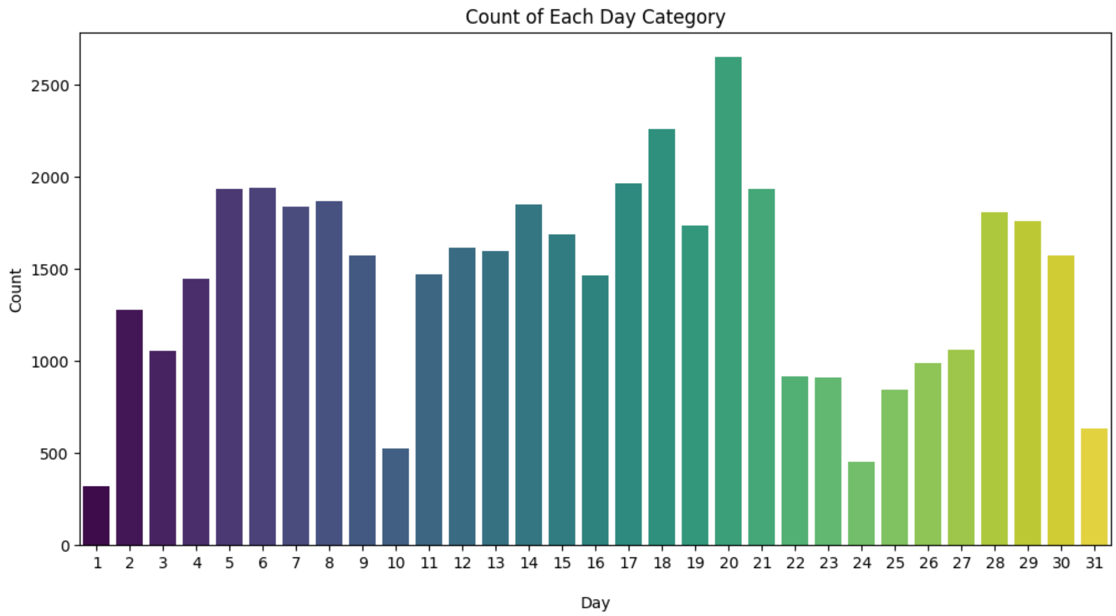
Day of the month when customers were last contacted

[1722]:
```
df['day'].value_counts()
```

[1722]:
```
day
20    2651
18    2257
17    1961
6     1938
5     1933
21    1930
8     1867
14    1844
7     1832
28    1806
29    1758
19    1730
15    1684
12    1613
13    1596
9     1571
30    1570
11    1470
16    1461
4     1444
2     1272
27    1059
3     1054
26     983
22     915
23     904
25     841
31     628
10     520
24     450
1      314
Name: count, dtype: int64
```

[1723]:
```
# Define counts
day_counts = df['day'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))

# Plot bar chart
sns.barplot(x = day_counts.index, y = day_counts.values,hue = day_counts.
 ↪index,legend = False, palette = 'viridis')

# Add labels and title
```

```
plt.title('Count of Each Day Category')
plt.xlabel('\nDay')
plt.ylabel('Count')

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



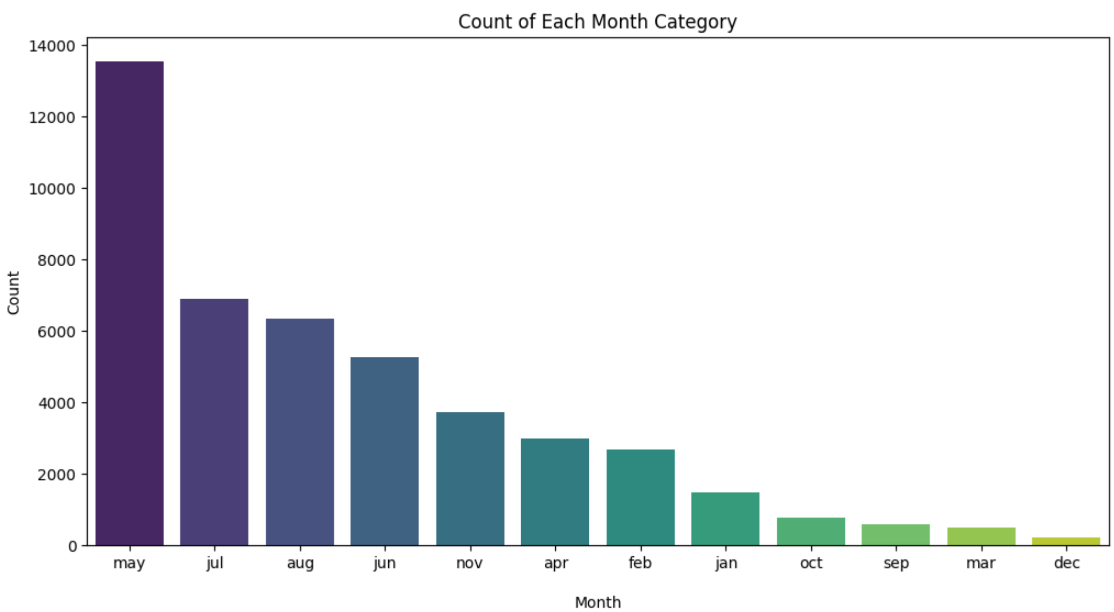
Month Feature

Last contact month of year.

[1724]:
```
df['month'].value_counts()
```

[1724]:
```
month
may    13550
jul     6878
aug     6342
jun     5264
nov     3712
apr     2981
feb     2658
jan     1464
oct      742
sep      581
mar      472
dec      212
Name: count, dtype: int64
```

[1725]:
```
# Define counts
month_counts = df['month'].value_counts()

# Define figure size
plt.figure(figsize = (12, 6))

# Plot bar chart
sns.barplot(x = month_counts.index, y = month_counts.values,hue = month_counts.
↪index,legend = False, palette = 'viridis')

# Add labels and title
plt.title('Count of Each Month Category')
plt.xlabel('\nMonth')
plt.ylabel('Count')

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



Duration Feature

last contact duration, in seconds

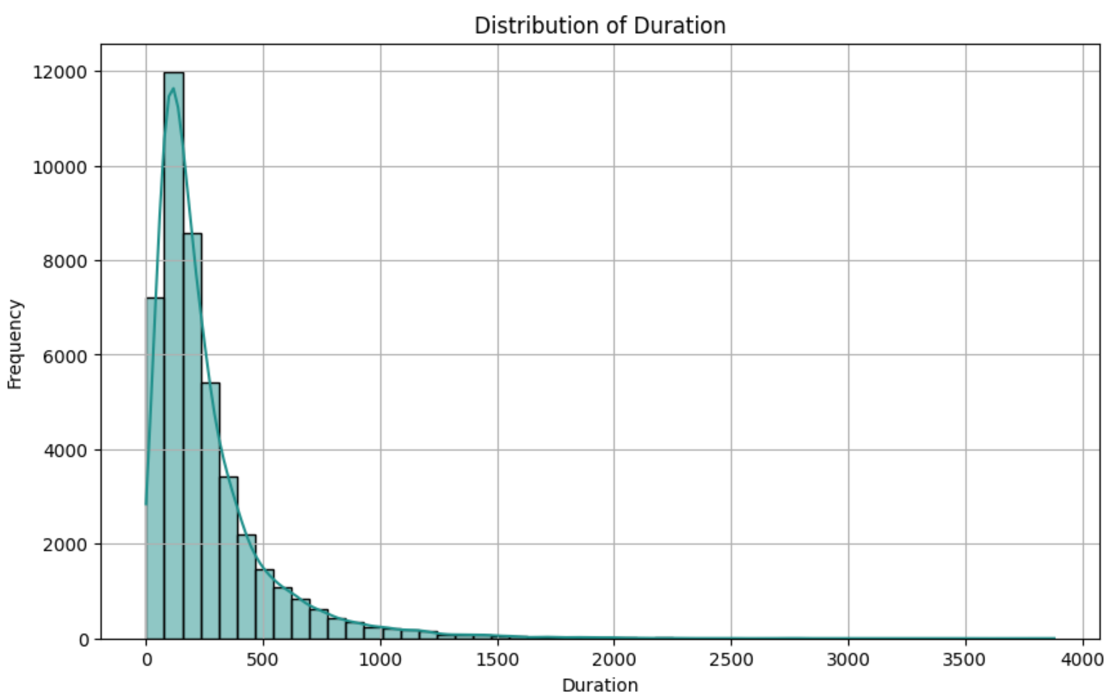
[1726]:
```
df['duration'].value_counts()
```

[1726]:
```
duration
124     184
119     180
121     179
```

```
104     178
112     178
       …
1508      1
1833      1
1545      1
1352      1
1556      1
Name: count, Length: 1541, dtype: int64
```

[1727]:

```
#Define figure size
plt.figure(figsize = (10, 6))
plt.grid(True)
# Plot the histogram
ax = sns.histplot(df['duration'], bins = 50, kde = True,color = '#21918C')

# Add labels and title
plt.title('Distribution of Duration')
plt.xlabel('Duration')
plt.ylabel('Frequency')
# plt.xticks([i for i in range(15, 100, 5)])


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



Campaign Feature

Number of contacts performed during this campaign and for this client

[1728]:
```
df['campaign'].value_counts()
```

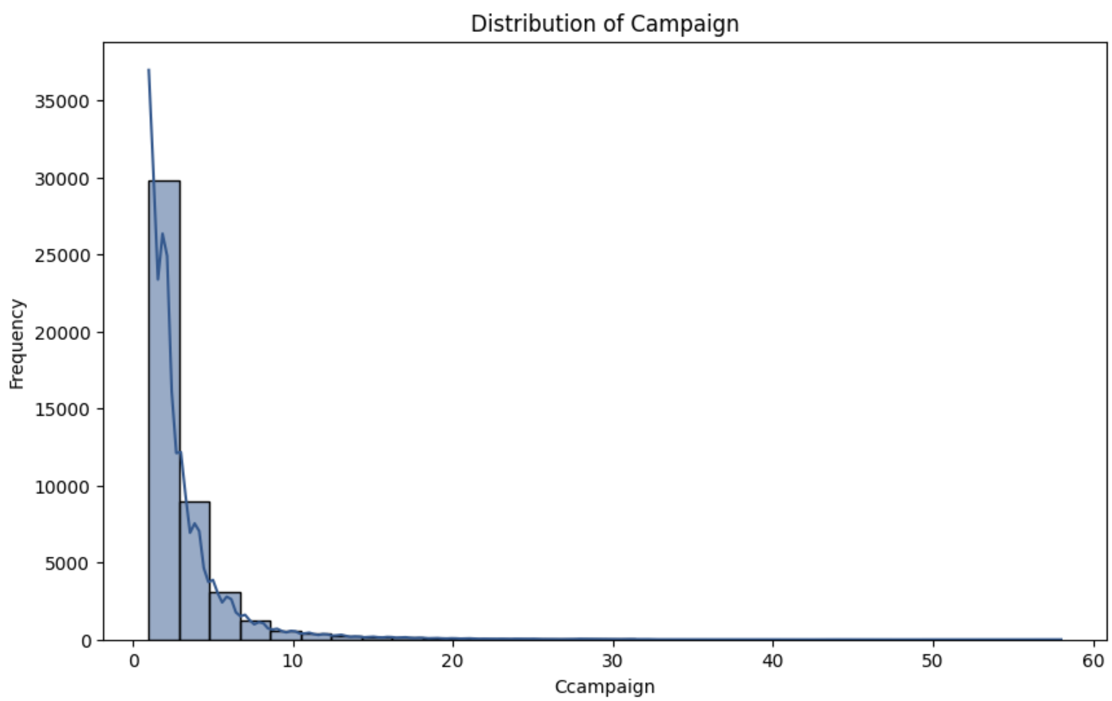
[1728]:
```
campaign
1     17384
2     12406
3      5468
4      3508
5      1745
6      1307
7       728
8       529
9       317
10      263
11      203
12      165
13      141
14       93
15       84
16       78
17       68
18       52
19       43
20       37
21       34
22       22
24       21
23       21
25       20
28       19
29       14
31       12
26       12
32       11
30        8
27        8
34        4
36        4
33        4
35        4
38        3
50        3
44        2
```

```
41        2
37        2
43        2
39        1
55        1
58        1
51        1
46        1
Name: count, dtype: int64
```

[1729]:
```
df['campaign'].describe()
```

[1729]:
```
count    44856.000000
mean         2.764959
std          3.084754
min          1.000000
25%          1.000000
50%          2.000000
75%          3.000000
max         58.000000
Name: campaign, dtype: float64
```

[1730]:
```
# Define figure size
plt.figure(figsize = (10, 6))

# Plot the histogram
ax = sns.histplot(df['campaign'], bins = 30, kde = True, color = '#35598E')

# Add labels and title
plt.title('Distribution of Campaign')
plt.xlabel('Ccampaign')
plt.ylabel('Frequency')
# plt.xticks([i for i in range(15, 100, 5)])


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



Passed Days

Number of days that passed by after the client was last contacted from a previous campaign (-1

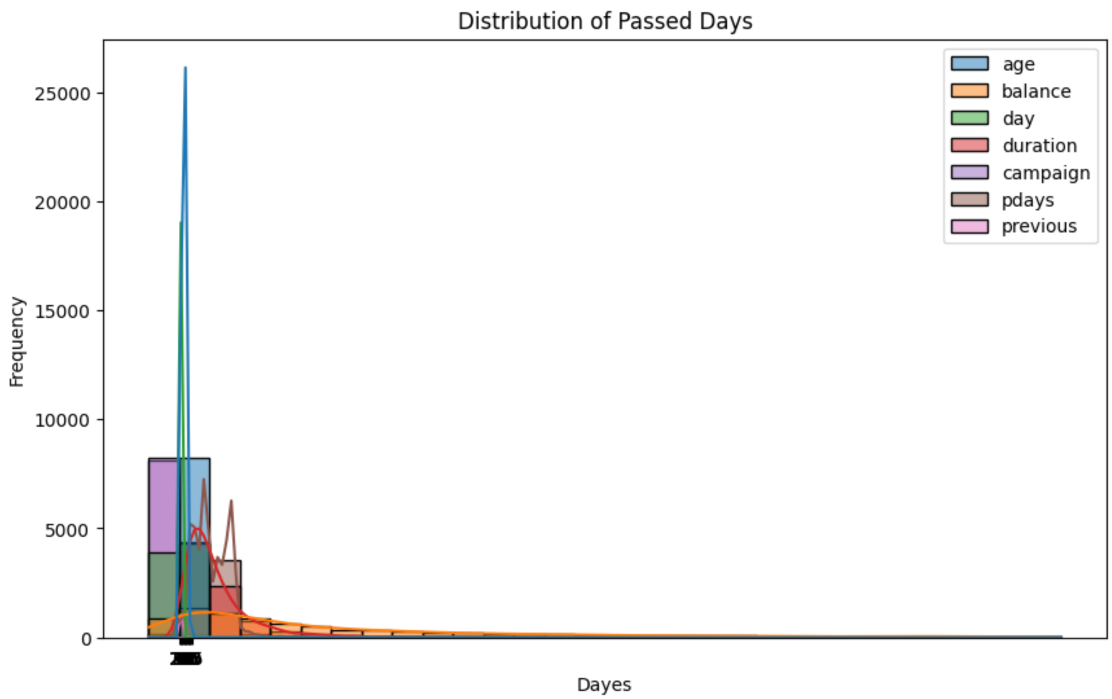
[1731]:
```
# Get the values that is not -1
filtered_data = df[df['pdays'] != -1]
```

[1732]:
```
# Get statistical summary
filtered_data['pdays'].describe()
```

[1732]:
```
count    8227.000000
mean      224.626352
std       116.207673
min         1.000000
25%       132.000000
50%       194.000000
75%       327.000000
max       871.000000
Name: pdays, dtype: float64
```

[1733]:
```
# Define figure size
plt.figure(figsize = (10, 6))

# Plot the histogram
ax = sns.histplot(filtered_data,bins = 30,kde = True,color = '#21918C')
```

```python
# Add labels and title
plt.title('Distribution of Passed Days')
plt.xlabel('Dayes')
plt.ylabel('Frequency')
plt.xticks([i for i in range(15, 100, 5)])


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



Previous Contacts

Number of contacts performed before this campaign and for this client.

[1734]:
```python
df['previous'].value_counts()
```

[1734]:
```
previous
0       36629
1        2779
2        2074
3        1133
4         720
5         465
6         275
```

```
7         205
8         142
9          89
10         66
11         63
12         45
13         31
14         20
15         19
17         14
16         12
19         11
20          8
23          8
22          7
24          6
27          5
18          4
25          4
21          4
29          3
38          2
37          2
30          2
51          1
275         1
26          1
58          1
28          1
32          1
40          1
55          1
41          1
Name: count, dtype: int64
```

Since most of values have 0 value,this feature is not necessary.Hence drop it

[1735]:
```python
df.drop(columns = ['previous'], inplace = True)
```

Previous Outcome

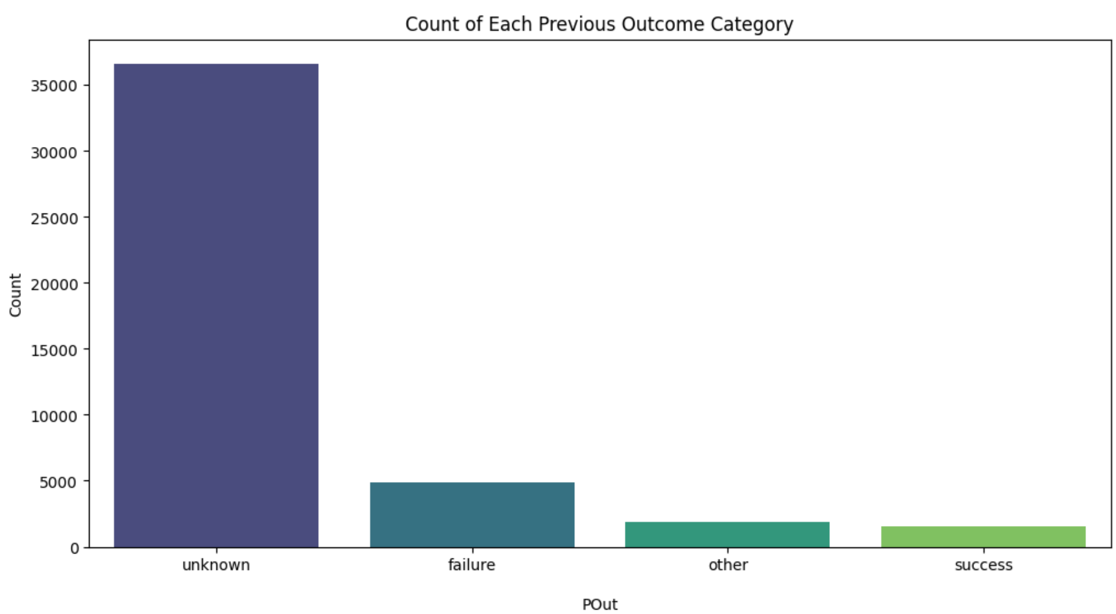
Outcome from previous marketing campaign.

[1736]:
```python
df['poutcome'].value_counts()
```

[1736]:
```
poutcome
unknown    36634
failure     4839
```

```
other        1861
success      1522
Name: count, dtype: int64
```

[1737]:
```
# Define counts
pout_counts = df['poutcome'].value_counts()

# Define figure size
plt.figure(figsize=(12, 6))

# Plot bar chart
sns.barplot(x = pout_counts.index, y = pout_counts.values,hue = pout_counts.
 ↪index,legend = False, palette='viridis')

# Add labels and title
plt.title('Count of Each Previous Outcome Category')
plt.xlabel('\nPOut')
plt.ylabel('Count')

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



The most of values are unknowns. Hence drop it

[1738]:
```
df.drop(columns = ['poutcome'], inplace = True)
```

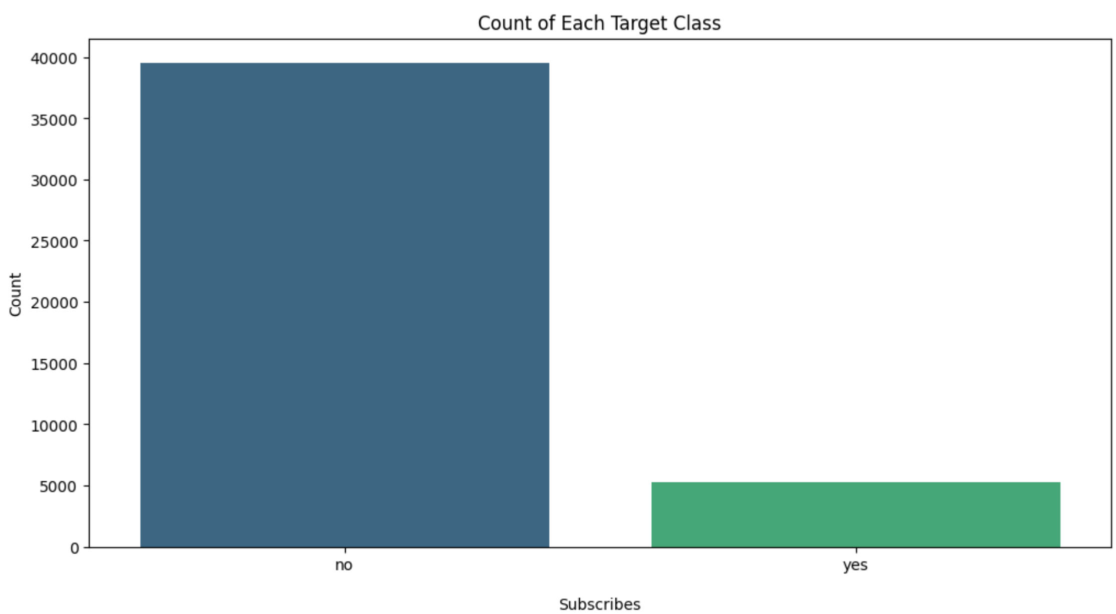
Target Column

Has the client subscribed a term deposit.

[1739]:
```
df['y'].value_counts()
```

[1739]:
```
y
no     39555
yes     5301
Name: count, dtype: int64
```

[1740]:
```
# Define counts
target_counts = df['y'].value_counts()

# Define figure size
plt.figure(figsize = (12, 6))

# Plot bar chart
sns.barplot(x = target_counts.index, y = target_counts.values,hue =␣
↪target_counts.index,legend = False, palette = 'viridis')

# Add labels and title
plt.title('Count of Each Target Class')
plt.xlabel('\nSubscribes')
plt.ylabel('Count')

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



Create a Label Encoder model to convert the categorical values into numeric

```
[1741]: #Converting categorical values into numeric
        from sklearn.preprocessing import LabelEncoder
        df = df.apply(LabelEncoder().fit_transform)
```

Features Correlation

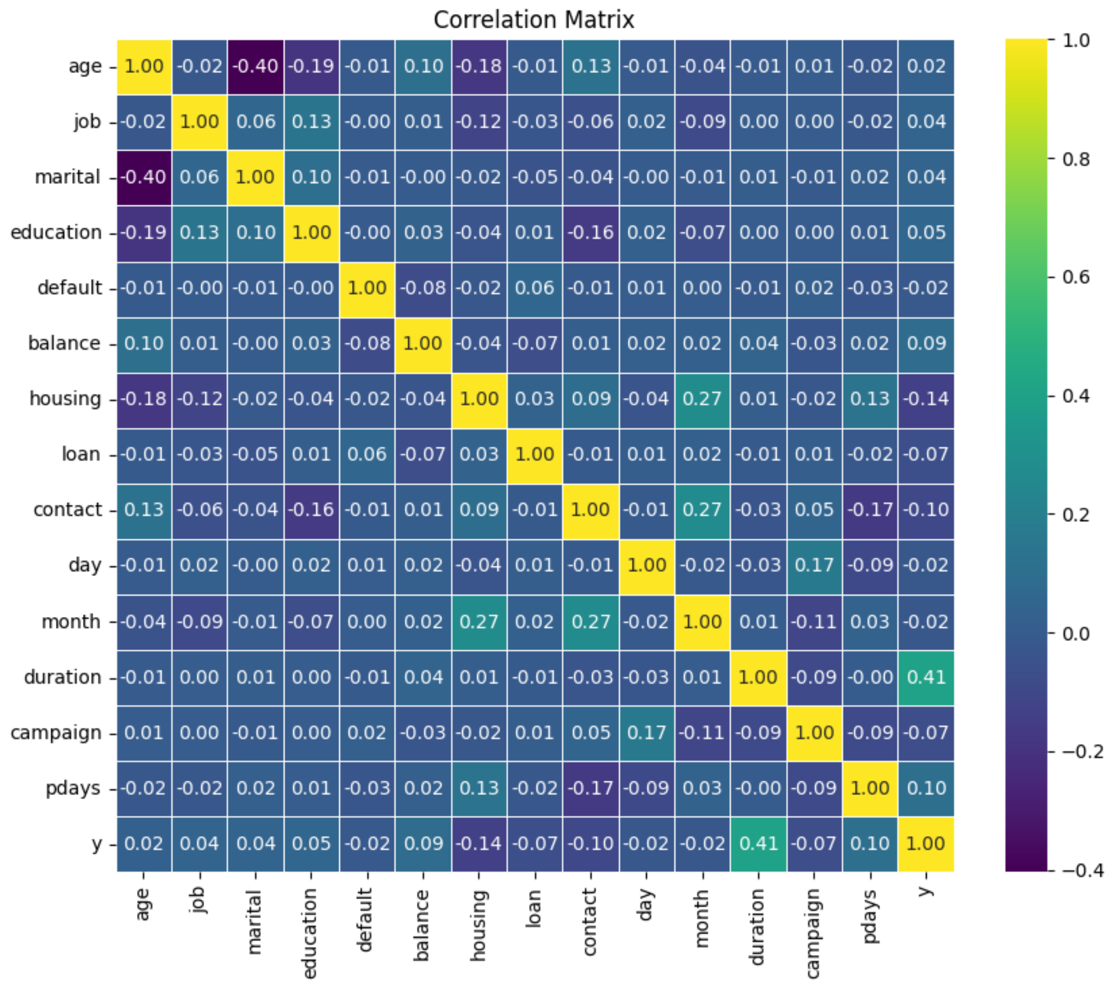
```
[1742]: df.corr()
```

```
[1742]:                 age       job   marital  education   default   balance  \
        age        1.000000 -0.021360 -0.403838  -0.185506 -0.013095  0.104719
        job       -0.021360  1.000000  0.062147   0.134183 -0.001519  0.008464
        marital   -0.403838  0.062147  1.000000   0.100429 -0.008042 -0.002250
        education -0.185506  0.134183  0.100429   1.000000 -0.003797  0.029984
        default   -0.013095 -0.001519 -0.008042  -0.003797  1.000000 -0.081977
        balance    0.104719  0.008464 -0.002250   0.029984 -0.081977  1.000000
        housing   -0.182568 -0.118689 -0.016263  -0.037160 -0.023848 -0.039418
        loan      -0.010979 -0.025799 -0.046146   0.008807  0.058560 -0.074309
        contact    0.127149 -0.061655 -0.044446  -0.161464 -0.009202  0.012160
        day       -0.006487  0.022386 -0.001454   0.022913  0.006337  0.017742
        month     -0.042910 -0.089837 -0.007680  -0.074536  0.004793  0.022937
        duration  -0.008671  0.003689  0.010857   0.002833 -0.011533  0.042326
        campaign   0.006158  0.003911 -0.009901   0.002069  0.018025 -0.028071
        pdays     -0.023039 -0.023951  0.018294   0.005296 -0.026260  0.016237
        y          0.022676  0.040831  0.044276   0.054307 -0.018002  0.088249

                    housing      loan   contact       day     month  duration  \
        age       -0.182568 -0.010979  0.127149 -0.006487 -0.042910 -0.008671
        job       -0.118689 -0.025799 -0.061655  0.022386 -0.089837  0.003689
        marital   -0.016263 -0.046146 -0.044446 -0.001454 -0.007680  0.010857
        education -0.037160  0.008807 -0.161464  0.022913 -0.074536  0.002833
        default   -0.023848  0.058560 -0.009202  0.006337  0.004793 -0.011533
        balance   -0.039418 -0.074309  0.012160  0.017742  0.022937  0.042326
        housing    1.000000  0.033591  0.091829 -0.036472  0.273604  0.010063
        loan       0.033591  1.000000 -0.014250  0.007692  0.022070 -0.014128
        contact    0.091829 -0.014250  1.000000 -0.010352  0.268092 -0.032752
        day       -0.036472  0.007692 -0.010352  1.000000 -0.019664 -0.030453
        month      0.273604  0.022070  0.268092 -0.019664  1.000000  0.006743
        duration   0.010063 -0.014128 -0.032752 -0.030453  0.006743  1.000000
        campaign  -0.024498  0.011943  0.046142  0.165496 -0.107868 -0.087676
        pdays      0.131484 -0.022723 -0.171631 -0.091659  0.030697 -0.001100
        y         -0.136640 -0.067817 -0.100141 -0.024196 -0.024707  0.405191

                   campaign     pdays         y
        age        0.006158 -0.023039  0.022676
        job        0.003911 -0.023951  0.040831
        marital   -0.009901  0.018294  0.044276
        education  0.002069  0.005296  0.054307
```

```
default    0.018025 -0.026260 -0.018002
balance   -0.028071  0.016237  0.088249
housing   -0.024498  0.131484 -0.136640
loan       0.011943 -0.022723 -0.067817
contact    0.046142 -0.171631 -0.100141
day        0.165496 -0.091659 -0.024196
month     -0.107868  0.030697 -0.024707
duration  -0.087676 -0.001100  0.405191
campaign   1.000000 -0.090658 -0.073342
pdays     -0.090658  1.000000  0.101960
y         -0.073342  0.101960  1.000000
```

[1743]:

```
#Calculate the correlation matrix
correlation_matrix = df.corr()

# Create a heatmap
plt.figure(figsize = (10, 8))
sns.heatmap(correlation_matrix, annot = True, cmap = 'viridis', fmt = ".2f",
 ↪linewidths = .5)
plt.title('Correlation Matrix')
plt.show()
```



Top 5 Most Positively Correlated

[1744]:
```python
print("Top 5 Most Positively Correlated to the Output 'y'")
correlation_matrix['y'].sort_values(ascending = False).head(5)
```

```
Top 5 Most Positively Correlated to the Output 'y'
```

[1744]:
```
y            1.000000
duration     0.405191
pdays        0.101960
balance      0.088249
education    0.054307
Name: y, dtype: float64
```

Top 5 Most Negatively Correlated

[1745]:
```
print("Top 5 Most Negatively Correlated to the Output 'y'")
correlation_matrix['y'].sort_values(ascending = True).head(5)
```

```
Top 5 Most Negatively Correlated to the Output 'y'
```

[1745]:
```
housing    -0.136640
contact    -0.100141
campaign   -0.073342
loan       -0.067817
month      -0.024707
Name: y, dtype: float64
```

Define Features x and Target y

[1746]:
```
#Input feature
x = df.iloc[:, :-1]

#Target
y = df.iloc[:, -1]
```

Feature Selection

[1747]:
```
#Feature selection using chi square test
from sklearn.feature_selection import chi2
score=chi2(x,y)
score
```

[1747]:
```
(array([1.13471714e+02, 2.13846126e+02, 2.78259375e+01, 3.89752111e+01,
        1.43496402e+01, 3.85666632e+05, 3.75986066e+02, 1.74333595e+02,
        4.03847331e+02, 1.23251215e+02, 4.55004879e+01, 1.64313599e+06,
        1.28614931e+03, 1.08905346e+05]),
 array([1.70090544e-026, 1.98968972e-048, 1.32734647e-007, 4.29221261e-010,
        1.51808707e-004, 0.00000000e+000, 9.30828782e-084, 8.37016303e-040,
        8.00624826e-090, 1.22865031e-028, 1.52600815e-011, 0.00000000e+000,
        1.15660270e-281, 0.00000000e+000]))
```

[1748]:
```
#f score values
f_score=pd.Series(score[0],index = x.columns)
f_score.sort_values(ascending=False)
```

[1748]:
```
duration     1.643136e+06
balance      3.856666e+05
pdays        1.089053e+05
campaign     1.286149e+03
contact      4.038473e+02
housing      3.759861e+02
job          2.138461e+02
loan         1.743336e+02
```

```
day          1.232512e+02
age          1.134717e+02
month        4.550049e+01
education    3.897521e+01
marital      2.782594e+01
default      1.434964e+01
dtype: float64
```

[1749]:
```
#p values
p_value=pd.Series(score[1],index=x.columns)
p_value.sort_values(ascending=False)
```

[1749]:
```
default       1.518087e-04
marital       1.327346e-07
education     4.292213e-10
month         1.526008e-11
age           1.700905e-26
day           1.228650e-28
loan          8.370163e-40
job           1.989690e-48
housing       9.308288e-84
contact       8.006248e-90
campaign     1.156603e-281
balance       0.000000e+00
duration      0.000000e+00
pdays         0.000000e+00
dtype: float64
```

Columns with high p value can be removed

[1750]:
```
#Dropping columns with high p_value
x.drop(columns=['contact'],inplace=True)
x.drop(columns=['housing'],inplace=True)
x.drop(columns=['loan'],inplace=True)
```

```
/tmp/ipykernel_5659/1526967643.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x.drop(columns=['contact'],inplace=True)
/tmp/ipykernel_5659/1526967643.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x.drop(columns=['housing'],inplace=True)
/tmp/ipykernel_5659/1526967643.py:4: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x.drop(columns=['loan'],inplace=True)
```

Splitting Training data and Testing data

[1751]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪30,random_state=42)
#Training data==>70%
#Testing data==>30%
```

[1752]:
```
#Smoting to avoid over sampling
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='auto',random_state=42)
x_train, y_train = smote.fit_resample(x_train,y_train)
x_test, y_test = smote.fit_resample(x_test,y_test)
```

[1753]:
```
#Display the shapes of the resulting datasets
print("x_train shape:",x_train.shape,"\nx_test shape:",x_test.shape,"\ny_train␣
 ↪shape:",y_train.shape,"\ny_test shape",y_test.shape)
```

```
x_train shape: (55344, 11)
x_test shape: (23766, 11)
y_train shape: (55344,)
y_test shape (23766,)
```

[1754]:
```
#Normalization Step
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
```

[1755]:
```
y_train.value_counts()
```

[1755]:
```
y
0    27672
1    27672
Name: count, dtype: int64
```

[1756]:
```
y_train.unique()
```

[1756]:
```
array([0, 1])
```

[1757]:
```
y_test.value_counts()
```

[1757]: y
0   11883
1   11883
Name: count, dtype: int64

[1758]:
```
y_test.unique()
```

[1758]: array([0, 1])

Machine Learning Algorithms

Model Building and Analysis

Logistic Regression

[1759]:
```
# Logistic Regression model
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

#Create a logistic regression model
lr = LogisticRegression()

#Define hyperparameter grid
param = {'penalty':['l1','l2'],'C':[0.001,0.01,0.1,1,10,100],'max_iter':
 ↪[50,100,200,500,1000,5000]}

#Use gridsearchCV to search for the best combination of hyperparameters
grid_search = GridSearchCV(estimator = lr,param_grid = param,cv = 2,scoring =␣
 ↪'accuracy')
grid_search.fit(x_train,y_train)

#Print the best hyperparameters
print("Best Hyperparameters:",grid_search.best_params_)

#Get the best model
best_lr_model = grid_search.best_estimator_
```

```
Best Hyperparameters: {'C': 10, 'max_iter': 50, 'penalty': 'l2'}

/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
72 fits failed out of a total of 144.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
```

```
72 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/base.py", line 1152, in
wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/linear_model/_logistic.py", line 1169, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/linear_model/_logistic.py", line 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py:979:
UserWarning: One or more of the test scores are non-finite: [        nan
0.75298135        nan 0.75298135        nan 0.75298135
        nan 0.75298135        nan 0.75298135        nan 0.75298135
        nan 0.75299942        nan 0.75299942        nan 0.75299942
        nan 0.75299942        nan 0.75299942        nan 0.75299942
        nan 0.75359569        nan 0.75359569        nan 0.75359569
        nan 0.75359569        nan 0.75359569        nan 0.75359569
        nan 0.75366797        nan 0.75366797        nan 0.75366797
        nan 0.75366797        nan 0.75366797        nan 0.75366797
        nan 0.75368604        nan 0.75368604        nan 0.75368604
        nan 0.75368604        nan 0.75368604        nan 0.75368604
        nan 0.75368604        nan 0.75368604        nan 0.75368604
        nan 0.75368604        nan 0.75368604        nan 0.75368604]
  warnings.warn(
```

[1760]:
```
lr_reg = LogisticRegression(penalty = 'l2',C = 10,max_iter = 50)


# Train the model
lr_reg.fit(x_train, y_train)
```

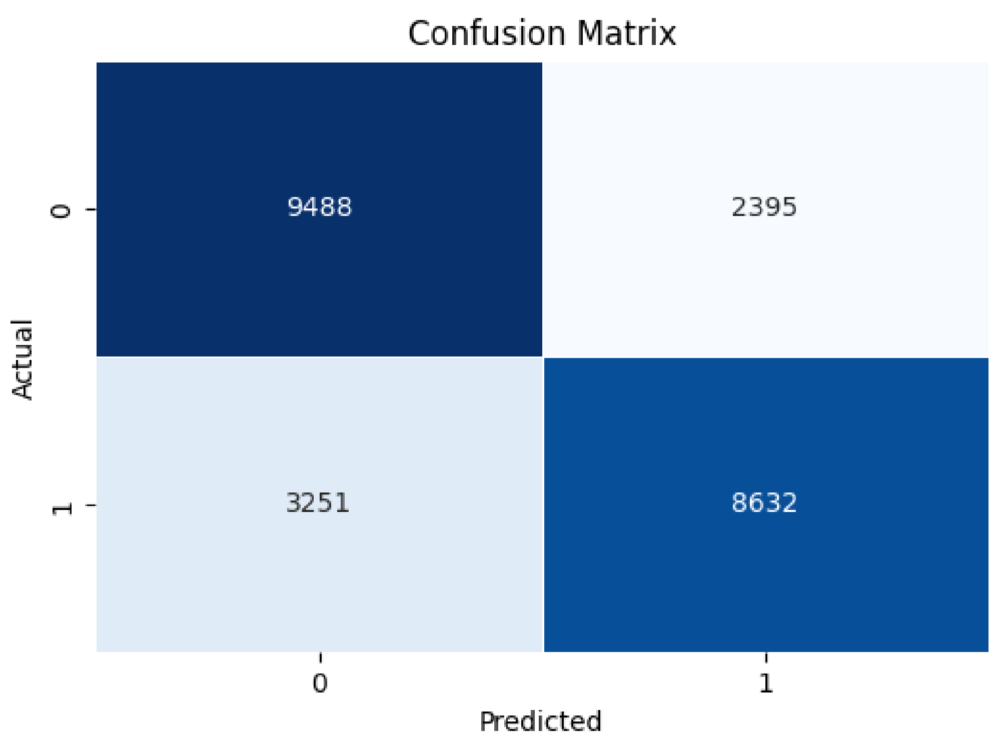
[1760]:
```
LogisticRegression(C=10, max_iter=50)
```

[1761]:
```
# Train Score
from sklearn.metrics import accuracy_score
print(lr_reg.score(x_train, y_train))

# Test Score
print(lr_reg.score(x_test, y_test))
```

```
0.7564686325527609
0.7624337288563494
```

[1762]:
```python
# Make predictions on the test set
y_pred = lr_reg.predict(x_test)
```

[1763]:
```python
# Create a confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix with a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', linewidths=.5,
  cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[1764]:
```python
#Confusion matrix
print(conf_matrix)
```

```
[[9488 2395]
 [3251 8632]]
```

[1765]:
```
#Accuracy score
print("Logistic Regression Accuracy Score:",accuracy_score(y_test,y_pred))
```

```
Logistic Regression Accuracy Score: 0.7624337288563494
```

[1766]:
```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.80      0.77     11883
           1       0.78      0.73      0.75     11883

    accuracy                           0.76     23766
   macro avg       0.76      0.76      0.76     23766
weighted avg       0.76      0.76      0.76     23766
```

Decision Tree Classifier

[1767]:
```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

#Create a Decision Tree Classifier
clf = DecisionTreeClassifier()

#Define the hyperparameter grid
param_grid={'max_depth':[None,10,20],'min_samples_split':
 ↪[2,5,10],'max_features':['auto','sqrt','log2']}

#Use gridsearchCV to search for the best combination of hyperparameters
grid_search = GridSearchCV(estimator =␣
 ↪clf,param_grid=param_grid,cv=2,scoring='accuracy')
grid_search.fit(x_train,y_train)

#Print the best hyperparameters
print("Best Hyperparameters:",grid_search.best_params_)

#Get the best model
best_dt_model = grid_search.best_estimator_
```

```
Best Hyperparameters: {'max_depth': None, 'max_features': 'log2',
'min_samples_split': 2}
```

```
/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
18 fits failed out of a total of 54.
```

```
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
18 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/base.py", line 1145, in
wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.8/dist-packages/sklearn/base.py", line 638, in
_validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/utils/_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of DecisionTreeClassifier must be an int in the range [1, inf), a
float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py:979:
UserWarning: One or more of the test scores are non-finite: [       nan
nan        nan 0.84805941 0.8426207  0.83991038
 0.84903513 0.84623446 0.83237569        nan        nan        nan
 0.81067505 0.81226511 0.80975354 0.81524646 0.81461405 0.80825383
        nan        nan        nan 0.84764383 0.84637901 0.84305435
 0.84758962 0.84075961 0.84224125]
  warnings.warn(
```

[1776]:
```python
dt_clf = DecisionTreeClassifier(max_depth = 20,min_samples_split =
 ↪2,max_features = 'log2')

#Train the model
dt_clf.fit(x_train,y_train)
```

[1776]:
```
DecisionTreeClassifier(max_depth=20, max_features='log2')
```

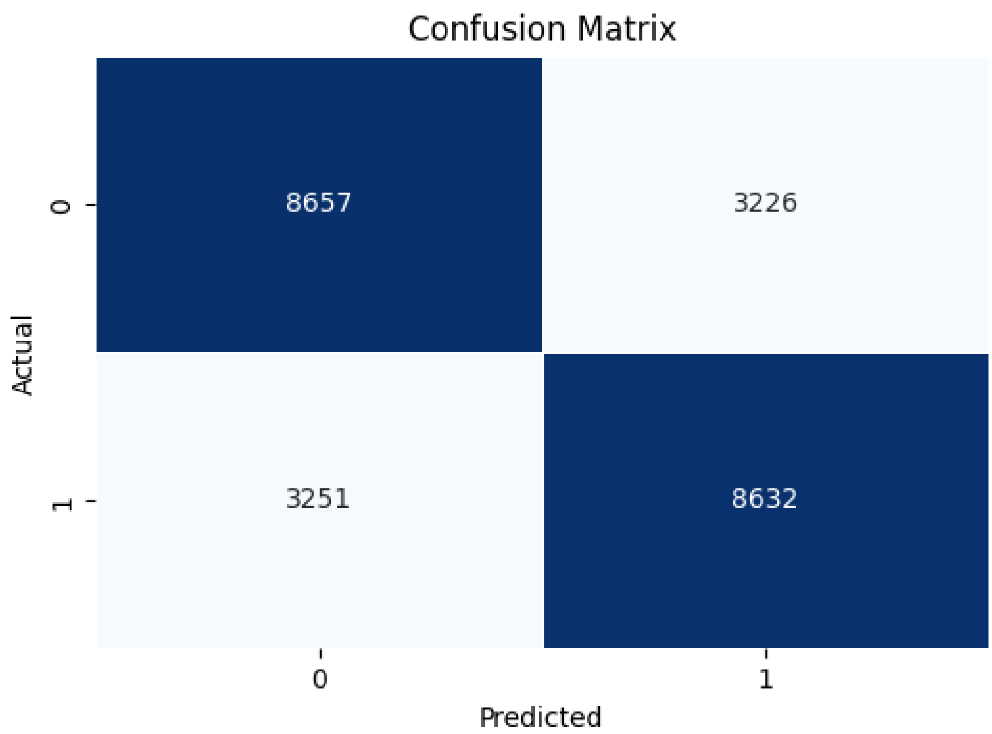
[1777]:
```python
# Train Score
print('Train score',dt_clf.score(x_train, y_train))
```

```
# Test Score
print('Test score',dt_clf.score(x_test, y_test))
```

```
Train score 0.9805941023417173
Test score 0.7274678111587983
```

[1778]:
```
# Make predictions on the test set
y_pred = dt_clf.predict(x_test)
```

[1779]:
```
# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix with a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = 'Blues', linewidths=.
↪5, cbar = False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[1780]:
```
#Confusion matrix
print(conf_matrix)
```

```
[[8657 3226]
 [3251 8632]]
```

[1781]:
```
#Accuracy score
print("Decision Tree Accuracy score:",accuracy_score(y_test,y_pred))
```

```
Decision Tree Accuracy score: 0.7274678111587983
```

[1782]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.73      0.73     11883
           1       0.73      0.73      0.73     11883

    accuracy                           0.73     23766
   macro avg       0.73      0.73      0.73     23766
weighted avg       0.73      0.73      0.73     23766
```

Random Forest Classifier

[1783]:
```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

#Create a Random Forest Classifier
RF_clf = RandomForestClassifier()

#Define the hyperparameter grid
param = {'n_estimators':[50,100,200],'max_depth':
 ↪[None,10,20],'min_samples_split':[2,5,10],'max_features':
 ↪['auto','sqrt','log2']}

#Use GridsearchCV to search for the best combination of hyperparameters
grid_search = GridSearchCV(estimator=RF_clf,param_grid=param,cv =␣
 ↪2,scoring='accuracy')
grid_search.fit(x_train,y_train)

#Print the best hyperparameters
print("Best Hyperparameter:",grid_search.best_params_)

#Get the best model
best_rf_model = grid_search.best_estimator_
```

```
/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
54 fits failed out of a total of 162.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
54 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/base.py", line 1145, in
wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.8/dist-packages/sklearn/base.py", line 638, in
_validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/utils/_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf), a
float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py:979:
UserWarning: One or more of the test scores are non-finite: [       nan
nan        nan        nan        nan        nan
        nan        nan        nan 0.90960176 0.91139058 0.91153513
 0.90403657 0.90602414 0.90634938 0.89693553 0.89800159 0.89946516
 0.90875253 0.90989086 0.91081237 0.90418112 0.90490387 0.90624097
 0.89708008 0.89843524 0.89955551        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.8606353  0.8617375  0.86267707 0.86119543 0.86065337 0.86229763
 0.85817794 0.85823215 0.85897297 0.86045461 0.86079792 0.86224342
 0.86032813 0.86114123 0.86117736 0.85964151 0.86045461 0.85920786
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.90665655 0.90920425 0.90858991
 0.90387395 0.90528332 0.9048316  0.89511058 0.89697167 0.89823648
 0.90757806 0.90907777 0.90958369 0.90441602 0.90553628 0.9054098
 0.89633926 0.89801966 0.8989231 ]
  warnings.warn(
```

```
Best Hyperparameter: {'max_depth': None, 'max_features': 'sqrt',
'min_samples_split': 2, 'n_estimators': 200}
```

[1784]:
```python
rf_clf = RandomForestClassifier(max_depth = None,max_features =
 ↪'sqrt',min_samples_split = 2,n_estimators = 200)

#Train the model
rf_clf.fit(x_train,y_train)
```

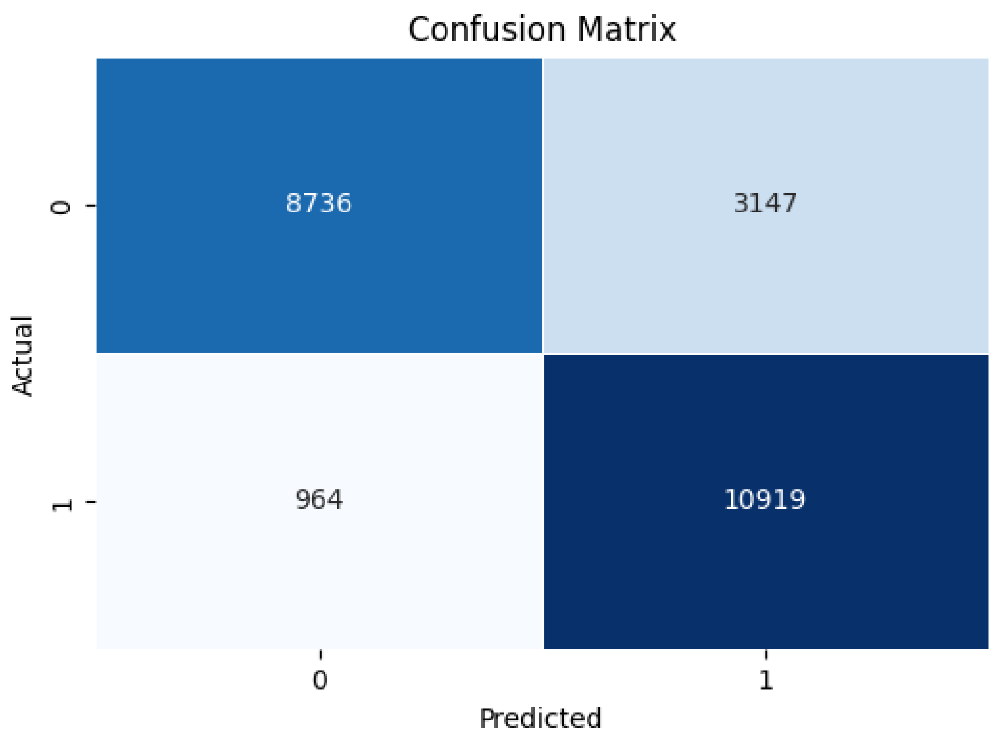
[1784]:
```
RandomForestClassifier(n_estimators=200)
```

[1785]:
```python
# Train Score
print('Train score',rf_clf.score(x_train, y_train))

# Test Score
print('Test score',rf_clf.score(x_test, y_test))
```

```
Train score 1.0
Test score 0.8270217958428007
```

[1786]:
```python
#Make predictions on test data
y_pred = rf_clf.predict(x_test)
```

[1787]:
```python
# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix with a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', linewidths=.5,
 ↪cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[1788]:
```
#Confusion matrix
print(conf_matrix)
```

```
[[ 8736  3147]
 [  964 10919]]
```

[1789]:
```
#Accuracy score
print("Random Forest Accuracy score:",accuracy_score(y_test,y_pred))
```

```
Random Forest Accuracy score: 0.8270217958428007
```

[1790]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.74      0.81     11883
           1       0.78      0.92      0.84     11883

    accuracy                           0.83     23766
   macro avg       0.84      0.83      0.83     23766
weighted avg       0.84      0.83      0.83     23766
```

[1792]:

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

models = {'Logistic Regression': LogisticRegression(penalty='l2', C=10,␣
 ↪max_iter=50),
          'Decision Tree': DecisionTreeClassifier(max_depth =␣
 ↪20,min_samples_split = 2,max_features = 'log2'),
          'Random Forest': RandomForestClassifier(max_depth = None,max_features␣
 ↪= 'sqrt',min_samples_split = 2,n_estimators = 200)}

best_model = None
best_score = 0

for model_name,model in models.items():
   model.fit(x_train,y_train)
   y_pred = model.predict(x_test)

   #Evaluate the model
   score = accuracy_score(y_test,y_pred)
   report = classification_report(y_test,y_pred)
   submit = pd.DataFrame()
   submit['Actual_value'] = y_test
   submit['Predicted_value'] = y_pred
   submit = submit.reset_index()
   score = accuracy_score(y_test,y_pred)

   if score > best_score:
      best_score = score
      best_model = model_name

      print(f'{model_name}:')  #f string formatting is used to embed variables␣
 ↪directly into the strings.
      print(f'Accuracy Score:{score:.2f}')
      print(f'Classification Report:{report:.2}')  #2f ==> till 2 decimal␣
 ↪points will be displayed
      print(submit.head(5))

print('----------------------------------------------------------------------')
print(f"The best performing model is:{best_model} with accuracy:{best_score:.
 ↪2f}")
```

```
Linear Regression:
Accuracy Score:0.76
Classification Report:
   index  Actual_value  Predicted_value
0      0             0                0
```

```
1      1             0                 0
2      2             0                 0
3      3             1                 1
4      4             0                 0
Random Forest:
Accuracy Score:0.83
Classification Report:
   index  Actual_value  Predicted_value
0      0             0                 1
1      1             0                 0
2      2             0                 0
3      3             1                 1
4      4             0                 0
--------------------------------------------------------------------------
The best performing model is:Random Forest with accuracy:0.83
```

Conclusion

Best Model:

```
* The Random Forest model appears to outperform the Decision tree and Linear regression models
* It achieved perfect predictions on the test data, indicating on almost match between predict
*  The Random Forest model has an accuracy score of 0.83, means it almost explains all the vari
```