tugas-3-eda-klasifikasi

November 8, 2024

0.1 # Tugas 3 Data Mining - EDA + Klasifikasi

List Anggota Kelompok 9 * Cahaya Aulia Firdausyah (2006304) * Anderfa Jalu Kawani (2102671) * Sabila Rosad (2106000)

[1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
```

[2]:
```
drive.mount('/content/drive')

df = pd.read_csv("/content/drive/MyDrive/Data Mining/Tugas 3/transact_train.
↪txt", delimiter="|")
```

```
Mounted at /content/drive
```

[3]:
```
df.head()
```

[3]:
```
   sessionNo  startHour  startWeekday  duration  cCount cMinPrice cMaxPrice  \
0          1          6             5     0.000       1     59.99     59.99
1          1          6             5    11.940       1     59.99     59.99
2          1          6             5    39.887       1     59.99     59.99
3          2          6             5     0.000       0         ?         ?
4          2          6             5    15.633       0         ?         ?

  cSumPrice  bCount bMinPrice  …          availability customerNo maxVal  \
0     59.99       1     59.99  …                     ?          1    600
1     59.99       1     59.99  …  completely orderable          1    600
2     59.99       1     59.99  …  completely orderable          1    600
3         ?       0         ?  …  completely orderable          ?      ?
4         ?       0         ?  …  completely orderable          ?      ?

  customerScore accountLifetime payments age address lastOrder order
0            70              21        1  43       1        49     y
1            70              21        1  43       1        49     y
2            70              21        1  43       1        49     y
```

```
3             ?               ?        ?   ?       ?         ?     y
4             ?               ?        ?   ?       ?         ?     y

[5 rows x 24 columns]
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 429013 entries, 0 to 429012
Data columns (total 24 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   sessionNo        429013 non-null  int64
 1   startHour        429013 non-null  int64
 2   startWeekday     429013 non-null  int64
 3   duration         429013 non-null  float64
 4   cCount           429013 non-null  int64
 5   cMinPrice        429013 non-null  object
 6   cMaxPrice        429013 non-null  object
 7   cSumPrice        429013 non-null  object
 8   bCount           429013 non-null  int64
 9   bMinPrice        429013 non-null  object
 10  bMaxPrice        429013 non-null  object
 11  bSumPrice        429013 non-null  object
 12  bStep            429013 non-null  object
 13  onlineStatus     429013 non-null  object
 14  availability     429013 non-null  object
 15  customerNo       429013 non-null  object
 16  maxVal           429013 non-null  object
 17  customerScore    429013 non-null  object
 18  accountLifetime  429013 non-null  object
 19  payments         429013 non-null  object
 20  age              429013 non-null  object
 21  address          429013 non-null  object
 22  lastOrder        429013 non-null  object
 23  order            429013 non-null  object
dtypes: float64(1), int64(5), object(18)
memory usage: 78.6+ MB
```

[5]:
```
df.describe()
```

[5]:
```
            sessionNo      startHour   startWeekday       duration  \
count  429013.000000  429013.000000  429013.000000  429013.000000
mean    25274.631293      14.617061       5.924839    1573.901640
std     14441.366146       4.485914       0.790930    2427.123356
min         1.000000       0.000000       5.000000       0.000000
25%     12731.000000      11.000000       5.000000     225.070000
```

```
50%     25470.000000      15.000000        6.000000     738.199000
75%     37542.000000      18.000000        7.000000    1880.265000
max     50000.000000      23.000000        7.000000   21580.092000

              cCount          bCount
count  429013.000000  429013.000000
mean       24.140317       4.135168
std        30.398164       4.451778
min         0.000000       0.000000
25%         5.000000       1.000000
50%        13.000000       3.000000
75%        31.000000       5.000000
max       200.000000     108.000000
```

[6]:
```python
# Check the data types
print(df.dtypes)
```

```
sessionNo            int64
startHour            int64
startWeekday         int64
duration           float64
cCount               int64
cMinPrice           object
cMaxPrice           object
cSumPrice           object
bCount               int64
bMinPrice           object
bMaxPrice           object
bSumPrice           object
bStep               object
onlineStatus        object
availability        object
customerNo          object
maxVal              object
customerScore       object
accountLifetime     object
payments            object
age                 object
address             object
lastOrder           object
order               object
dtype: object
```

[7]:
```python
# Delete rows with '?' in any column
df = df[(df != '?').all(axis=1)]
```

[8]:
```
# Convert numeric columns to float
numeric_cols = ['sessionNo', 'startHour', 'startWeekday', 'duration', 'cCount',␣
 ↪'bCount', 'maxVal', 'customerScore', 'accountLifetime', 'payments', 'age',␣
 ↪'lastOrder']
df[numeric_cols] = df[numeric_cols].astype(float)

# Convert categorical columns to string
categorical_cols = ['cMinPrice', 'cMaxPrice', 'cSumPrice', 'bMinPrice',␣
 ↪'bMaxPrice', 'bSumPrice', 'bStep', 'onlineStatus', 'availability', 'order',␣
 ↪'customerNo', 'address']
df[categorical_cols] = df[categorical_cols].astype(str)

# Verify the data types
print(df.dtypes)
```

```
sessionNo          float64
startHour          float64
startWeekday       float64
duration           float64
cCount             float64
cMinPrice           object
cMaxPrice           object
cSumPrice           object
bCount             float64
bMinPrice           object
bMaxPrice           object
bSumPrice           object
bStep               object
onlineStatus        object
availability        object
customerNo          object
maxVal             float64
customerScore      float64
accountLifetime    float64
payments           float64
age                float64
address             object
lastOrder          float64
order               object
dtype: object
```

[9]:
```
numeric_cols = ['sessionNo', 'startHour', 'startWeekday', 'duration', 'cCount',␣
 ↪'bCount', 'maxVal', 'customerScore', 'accountLifetime', 'payments', 'age',␣
 ↪'lastOrder']
categorical_cols = ['cMinPrice', 'cMaxPrice', 'cSumPrice', 'bMinPrice',␣
 ↪'bMaxPrice', 'bSumPrice', 'bStep', 'onlineStatus', 'availability', 'order',␣
 ↪'customerNo', 'address']
```

[10]:
```python
from sklearn.impute import SimpleImputer

# Impute missing values in numeric columns using mean
numeric_imputer = SimpleImputer(strategy='mean')
df[numeric_cols] = numeric_imputer.fit_transform(df[numeric_cols])
```

[11]:
```python
# Impute missing values in categorical columns using mode
from collections import Counter

for col in categorical_cols:
    mode_value = df[col].mode().iloc[0]
    df[col] = df[col].fillna(mode_value)
```

[12]:
```python
# Check for remaining '?' values
print(df.isin(['?']).sum())
```

```
sessionNo          0
startHour          0
startWeekday       0
duration           0
cCount             0
cMinPrice          0
cMaxPrice          0
cSumPrice          0
bCount             0
bMinPrice          0
bMaxPrice          0
bSumPrice          0
bStep              0
onlineStatus       0
availability       0
customerNo         0
maxVal             0
customerScore      0
accountLifetime    0
payments           0
age                0
address            0
lastOrder          0
order              0
dtype: int64
```

[13]:
```python
# Check for NaN values
print(df.isna().sum())
```

```
sessionNo          0
startHour          0
startWeekday       0
```

```
duration          0
cCount            0
cMinPrice         0
cMaxPrice         0
cSumPrice         0
bCount            0
bMinPrice         0
bMaxPrice         0
bSumPrice         0
bStep             0
onlineStatus      0
availability      0
customerNo        0
maxVal            0
customerScore     0
accountLifetime   0
payments          0
age               0
address           0
lastOrder         0
order             0
dtype: int64
```

[14]:
```
# Visually inspect the dataset
print(df.head())
```

```
    sessionNo  startHour  startWeekday  duration  cCount cMinPrice cMaxPrice  \
1         1.0        6.0           5.0    11.940     1.0     59.99     59.99
11        3.0        6.0           5.0   324.278    11.0      9.99     29.99
20        5.0        6.0           5.0  2738.467    45.0     12.99    179.95
21        5.0        6.0           5.0  2797.247    45.0     12.99    179.95
27        7.0        6.0           5.0   268.713     6.0       3.0      20.0

   cSumPrice  bCount bMinPrice  …         availability customerNo  maxVal  \
1      59.99     1.0     59.99  …  completely orderable          1   600.0
11    109.95     2.0      9.99  …  completely orderable          3  1800.0
20   1093.72     4.0     19.99  …  completely orderable          4   800.0
21   1093.72     4.0     19.99  …  completely orderable          4   800.0
27      73.0     1.0       3.0  …  completely orderable          5   900.0

   customerScore accountLifetime payments   age  address  lastOrder  order
1           70.0            21.0      1.0  43.0        1       49.0      y
11         475.0           302.0     12.0  45.0        1       11.0      y
20         503.0            18.0      1.0  46.0        1       40.0      y
21         503.0            18.0      1.0  46.0        1       40.0      y
27         575.0            35.0     10.0  31.0        2       10.0      y

[5 rows x 24 columns]
```

[15]:
```
# Summarize the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 141163 entries, 1 to 428972
Data columns (total 24 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   sessionNo        141163 non-null  float64
 1   startHour        141163 non-null  float64
 2   startWeekday     141163 non-null  float64
 3   duration         141163 non-null  float64
 4   cCount           141163 non-null  float64
 5   cMinPrice        141163 non-null  object
 6   cMaxPrice        141163 non-null  object
 7   cSumPrice        141163 non-null  object
 8   bCount           141163 non-null  float64
 9   bMinPrice        141163 non-null  object
 10  bMaxPrice        141163 non-null  object
 11  bSumPrice        141163 non-null  object
 12  bStep            141163 non-null  object
 13  onlineStatus     141163 non-null  object
 14  availability     141163 non-null  object
 15  customerNo       141163 non-null  object
 16  maxVal           141163 non-null  float64
 17  customerScore    141163 non-null  float64
 18  accountLifetime  141163 non-null  float64
 19  payments         141163 non-null  float64
 20  age              141163 non-null  float64
 21  address          141163 non-null  object
 22  lastOrder        141163 non-null  float64
 23  order            141163 non-null  object
dtypes: float64(12), object(12)
memory usage: 26.9+ MB
None
```

[16]:
```
df.head()
```

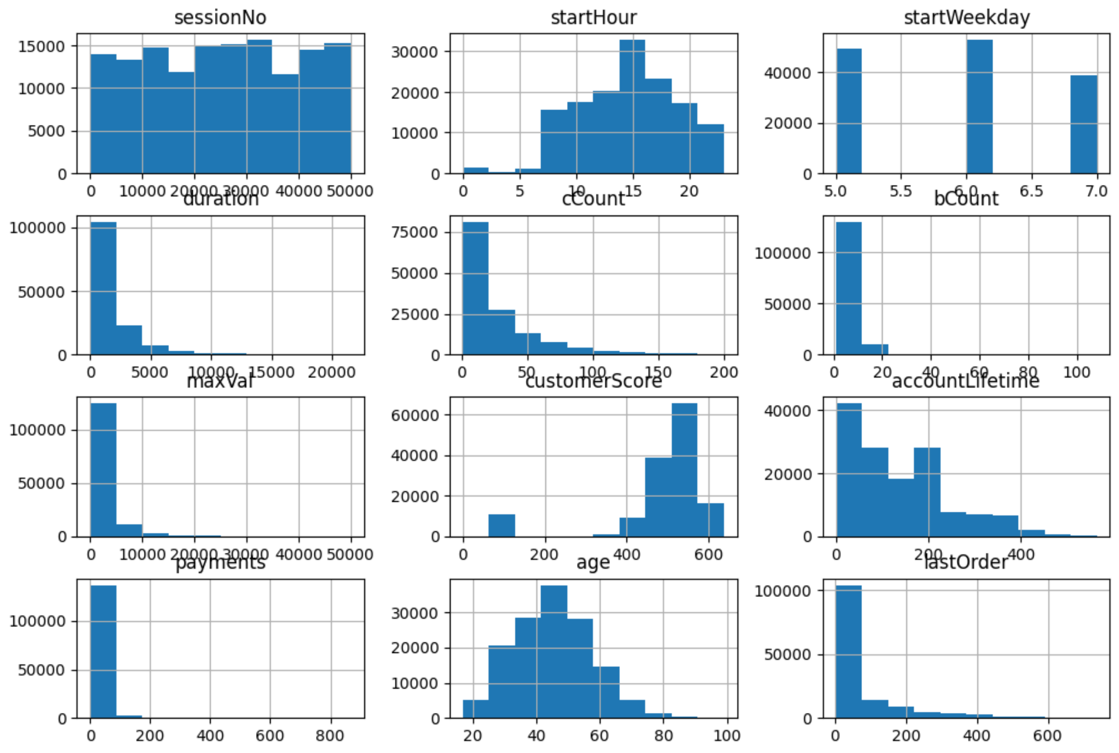
[16]:
```
    sessionNo  startHour  startWeekday  duration  cCount cMinPrice cMaxPrice  \
1         1.0        6.0           5.0    11.940     1.0     59.99     59.99
11        3.0        6.0           5.0   324.278    11.0      9.99     29.99
20        5.0        6.0           5.0  2738.467    45.0     12.99    179.95
21        5.0        6.0           5.0  2797.247    45.0     12.99    179.95
27        7.0        6.0           5.0   268.713     6.0       3.0      20.0

   cSumPrice  bCount bMinPrice  …          availability customerNo  maxVal  \
1      59.99     1.0     59.99  …  completely orderable          1   600.0
```

```
11    109.95      2.0       9.99  …  completely orderable          3  1800.0
20   1093.72      4.0      19.99  …  completely orderable          4   800.0
21   1093.72      4.0      19.99  …  completely orderable          4   800.0
27      73.0      1.0        3.0  …  completely orderable          5   900.0

   customerScore accountLifetime payments   age  address  lastOrder  order
1           70.0            21.0      1.0  43.0        1       49.0      y
11         475.0           302.0     12.0  45.0        1       11.0      y
20         503.0            18.0      1.0  46.0        1       40.0      y
21         503.0            18.0      1.0  46.0        1       40.0      y
27         575.0            35.0     10.0  31.0        2       10.0      y

[5 rows x 24 columns]
```

[17]:
```python
# Summary statistics for numeric columns
print(df[numeric_cols].describe())

# Visualize the distributions of numeric columns
import matplotlib.pyplot as plt
df[numeric_cols].hist(figsize=(12, 8))
plt.show()
```

```
           sessionNo      startHour   startWeekday       duration  \
count  141163.000000  141163.000000  141163.000000  141163.000000
mean    25271.805494      14.662865       5.924555    1838.816338
std     14442.609194       4.324934       0.787167    2512.450329
min         1.000000       0.000000       5.000000       0.062000
25%     12702.000000      11.000000       5.000000     383.329000
50%     25482.000000      15.000000       6.000000     992.864000
75%     37533.000000      18.000000       7.000000    2245.432500
max     49995.000000      23.000000       7.000000   21553.323000

              cCount         bCount         maxVal  customerScore  \
count  141163.000000  141163.000000  141163.000000  141163.000000
mean       28.235557       4.865347    2636.787260     486.201823
std        32.808797       4.728091    3241.472901     128.959337
min         1.000000       1.000000       0.000000       0.000000
25%         6.000000       2.000000     600.000000     481.000000
50%        16.000000       3.000000    1600.000000     520.000000
75%        37.000000       6.000000    4000.000000     554.000000
max       200.000000     108.000000   50000.000000     638.000000

       accountLifetime       payments            age      lastOrder
count    141163.000000  141163.000000  141163.000000  141163.000000
mean        138.734031      17.082479      45.247593      77.973208
std         110.553214      38.547387      11.943082     113.282558
min           0.000000       0.000000      17.000000       3.000000
```

```
25%          45.000000     3.000000    37.000000    14.000000
50%         113.000000     9.000000    45.000000    32.000000
75%         220.000000    16.000000    53.000000    81.000000
max         564.000000   868.000000    99.000000   738.000000
```



[18]:
```
# Explore the categorical columns
for col in categorical_cols:
    print(f"Column: {col}")

    # Check number of unique values
    unique_values = df[col].unique()
    print(f"Number of unique values: {len(unique_values)}")

    # Print the top 10 most frequent values
    value_counts = df[col].value_counts()
    print("Most frequent values:")
    print(value_counts.head(10))

    # Check for any unexpected or invalid values
    unusual_values = value_counts[value_counts < 10].index
    if len(unusual_values) > 0:
        print("Unusual/Infrequent values:")
        print(unusual_values)
    print("---")
```

```
Column: cMinPrice
Number of unique values: 498
Most frequent values:
cMinPrice
9.99     19123
3.99     15828
19.99     7420
12.99     6209
4.99      5905
14.99     5861
7.99      4730
29.99     4410
3.0       4010
6.99      3928
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['1799.99', '79.0', '309.99', '4.6', '51.96', '13.95', '13.9', '149.95',
       '759.99', '36.85',
       …
       '999.9', '180.0', '819.0', '134.99', '12.27', '489.99', '1699.99',
       '18.5', '38.64', '8.75'],
      dtype='object', name='cMinPrice', length=204)
---
Column: cMaxPrice
Number of unique values: 635
Most frequent values:
cMaxPrice
29.99    11639
19.99     9533
39.99     8769
49.99     8423
24.99     6236
59.99     6145
99.99     4514
79.99     4098
59.95     4027
34.99     3909
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['949.0', '69.96', '7.5', '85.0', '3394.03', '45.9', '75.9', '100.5',
       '36.85', '1739.0',
       …
       '82.99', '15.9', '1039.99', '1599.0', '709.9', '689.99', '9.5', '53.5',
       '919.99', '259.0'],
      dtype='object', name='cMaxPrice', length=204)
```

```
---
Column: cSumPrice
Number of unique values: 26254
Most frequent values:
cSumPrice
39.98    774
59.98    465
39.99    464
59.97    410
89.97    403
49.98    378
79.98    375
19.98    364
29.99    357
29.98    319
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['185.87', '1266.51', '116.71', '387.8', '4005.85', '1849.97',
       '15908.84', '164.91', '4029.08', '716.3',
       …
       '2021.56', '2696.93', '2797.78', '1904.01', '2134.94', '1485.86',
       '386.92', '3910.68', '1613.29', '319.6'],
      dtype='object', name='cSumPrice', length=23216)
---
Column: bMinPrice
Number of unique values: 522
Most frequent values:
bMinPrice
9.99     20313
3.99     14762
19.99     8764
14.99     7318
12.99     6950
29.99     5872
24.99     4297
4.99      4297
7.99      3660
39.99     3656
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['144.99', '31.99', '4.6', '389.99', '79.0', '110.0', '58.82', '28.99',
       '1299.0', '150.0',
       …
       '72.0', '50.0', '2.77', '779.0', '180.0', '1.2', '48.96', '190.0',
       '2799.99', '65.95'],
      dtype='object', name='bMinPrice', length=207)
---
Column: bMaxPrice
```

```
Number of unique values: 548
Most frequent values:
bMaxPrice
29.99    14379
19.99    12846
39.99     9593
49.99     8494
24.99     7636
59.99     5550
9.99      4597
14.99     4135
59.95     4071
34.99     3676
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['51.96', '10.95', '52.99', '6.95', '183.96', '140.0', '161.8', '15.49',
       '144.99', '53.99',
       …
       '1039.99', '409.99', '819.0', '1059.9', '6.59', '2799.99', '959.9',
       '53.0', '1119.0', '224.95'],
      dtype='object', name='bMaxPrice', length=170)
---
Column: bSumPrice
Number of unique values: 8462
Most frequent values:
bSumPrice
29.99    2318
19.99    2128
39.99    1823
24.99    1515
9.99     1405
49.99    1349
39.98    1209
59.98    1169
59.99    1045
49.98    1027
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['186.86', '1099.97', '261.86', '1049.3', '189.86', '238.81', '173.87',
       '256.83', '106.94', '151.93',
       …
       '451.97', '81.75', '606.66', '521.69', '385.72', '319.74', '327.89',
       '309.9', '101.34', '30.59'],
      dtype='object', name='bSumPrice', length=5913)
---
Column: bStep
Number of unique values: 5
Most frequent values:
```

```
bStep
1    54046
2    31480
4    28676
3    17888
5     9073
Name: count, dtype: int64
---
Column: onlineStatus
Number of unique values: 2
Most frequent values:
onlineStatus
y    139488
n      1675
Name: count, dtype: int64
---
Column: availability
Number of unique values: 7
Most frequent values:
availability
completely orderable          134756
mainly orderable                4185
mixed                            753
completely not determinable      582
completely not orderable         555
mainly not orderable             205
mainly not determinable          127
Name: count, dtype: int64
---
Column: order
Number of unique values: 2
Most frequent values:
order
y    114781
n     26382
Name: count, dtype: int64
---
Column: customerNo
Number of unique values: 21164
Most frequent values:
customerNo
5464     268
7394     124
16740    115
4118     100
5981      96
15503     92
16132     89
```

```
5336      89
10777     87
4034      87
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['15452', '15975', '15446', '17193', '15950', '15892', '15400', '6531',
       '16690', '15973',
       …
       '15500', '15501', '15504', '15512', '15519', '15520', '15524', '15528',
       '15547', '25037'],
      dtype='object', name='customerNo', length=16958)
---
Column: address
Number of unique values: 3
Most frequent values:
address
2    103294
1     37726
3       143
Name: count, dtype: int64
---
```

[19]:
```
# Cross-validate columns
print(df.loc[df['cCount'] > df['bCount']])
```

```
        sessionNo  startHour  startWeekday  duration  cCount cMinPrice  \
11            3.0        6.0           5.0   324.278    11.0      9.99
20            5.0        6.0           5.0  2738.467    45.0     12.99
21            5.0        6.0           5.0  2797.247    45.0     12.99
27            7.0        6.0           5.0   268.713     6.0       3.0
28            7.0        6.0           5.0   274.297     6.0       3.0
…             …        …             …         …         …       …
428953    49993.0       18.0           7.0  3866.511    69.0      9.99
428954    49993.0       18.0           7.0  3915.585    69.0      9.99
428955    49993.0       18.0           7.0  4094.847    69.0      9.99
428956    49993.0       18.0           7.0  4113.213    69.0      9.99
428972    49995.0       18.0           7.0   572.544    22.0      9.99

       cMaxPrice cSumPrice  bCount bMinPrice  …          availability  \
11         29.99    109.95     2.0      9.99  …  completely orderable
20        179.95   1093.72     4.0     19.99  …  completely orderable
21        179.95   1093.72     4.0     19.99  …  completely orderable
27          20.0      73.0     1.0       3.0  …  completely orderable
28          20.0      73.0     1.0       3.0  …  completely orderable
…             …        …       …          …  …                      …
428953     24.99    971.31    15.0      9.99  …  completely orderable
428954     24.99    971.31    15.0      9.99  …  completely orderable
428955     24.99    971.31    15.0      9.99  …  completely orderable
```

```
428956      24.99    971.31    15.0      9.99  …  completely orderable
428972      19.99     319.6     2.0      9.99  …  completely orderable

       customerNo  maxVal customerScore accountLifetime payments  age  \
11              3  1800.0         475.0           302.0     12.0 45.0
20              4   800.0         503.0            18.0      1.0 46.0
21              4   800.0         503.0            18.0      1.0 46.0
27              5   900.0         575.0            35.0     10.0 31.0
28              5   900.0         575.0            35.0     10.0 31.0
…             …     …             …               …        …   …
428953      25036   300.0         503.0            25.0      0.0 54.0
428954      25036   300.0         503.0            25.0      0.0 54.0
428955      25036   300.0         503.0            25.0      0.0 54.0
428956      25036   300.0         503.0            25.0      0.0 54.0
428972      25037   800.0         522.0            63.0      2.0 42.0

        address  lastOrder  order
11            1       11.0      y
20            1       40.0      y
21            1       40.0      y
27            2       10.0      y
28            2       10.0      y
…           …         …    …
428953        2       45.0      n
428954        2       45.0      n
428955        2       45.0      n
428956        2       45.0      n
428972        2        9.0      n

[124393 rows x 24 columns]
```

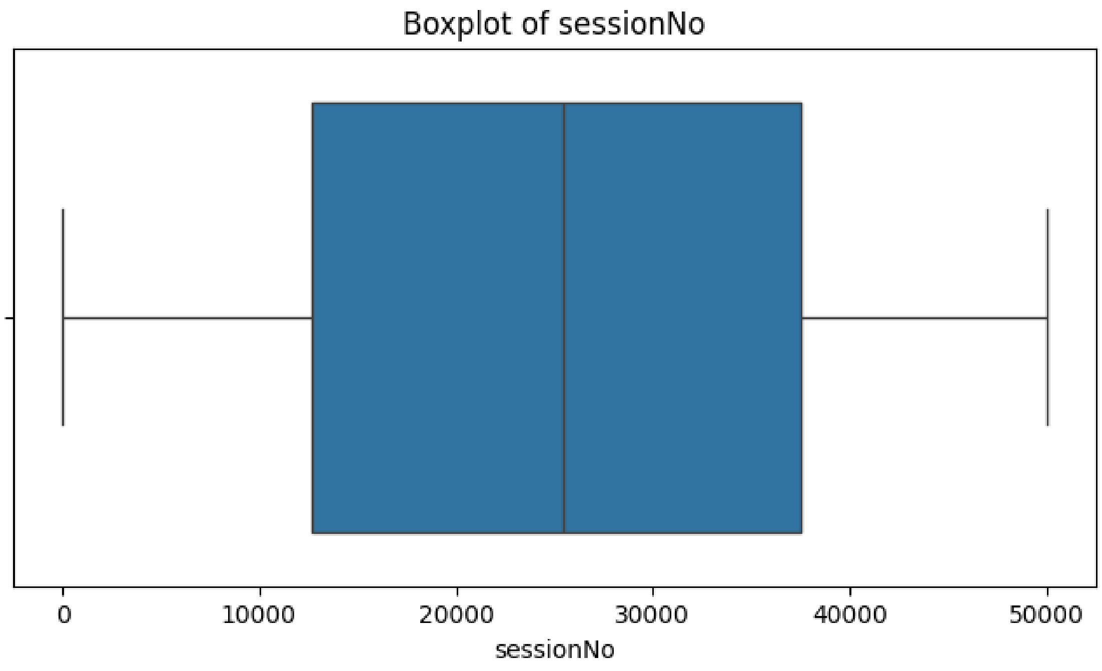
[20]:
```python
# Identify outliers using z-score
from scipy.stats import zscore

z = np.abs(zscore(df[numeric_cols]))

# Create a boolean mask for outlier rows, considering any outlier across columns
outlier_mask = (z > 3).any(axis=1)

# Filter the DataFrame using the outlier mask
outliers = df[outlier_mask]

print(outliers)
```

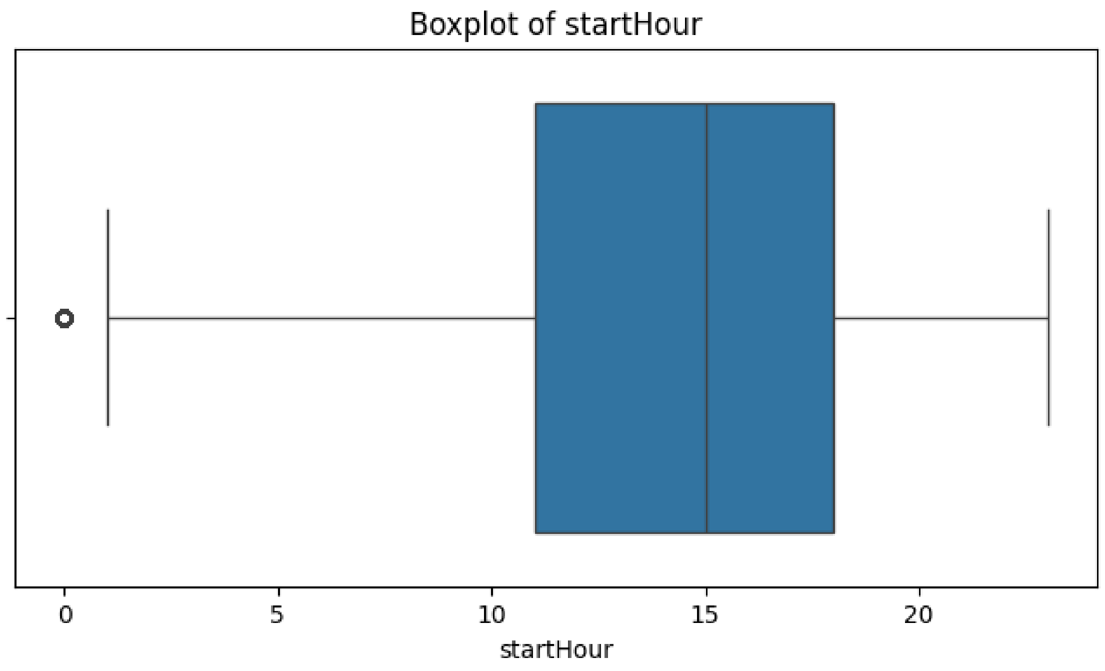
```
        sessionNo  startHour  startWeekday  duration  cCount cMinPrice  \
1             1.0        6.0           5.0    11.940     1.0     59.99
77           12.0        6.0           5.0   555.557    14.0      5.99
78           12.0        6.0           5.0   594.719    14.0      5.99
```

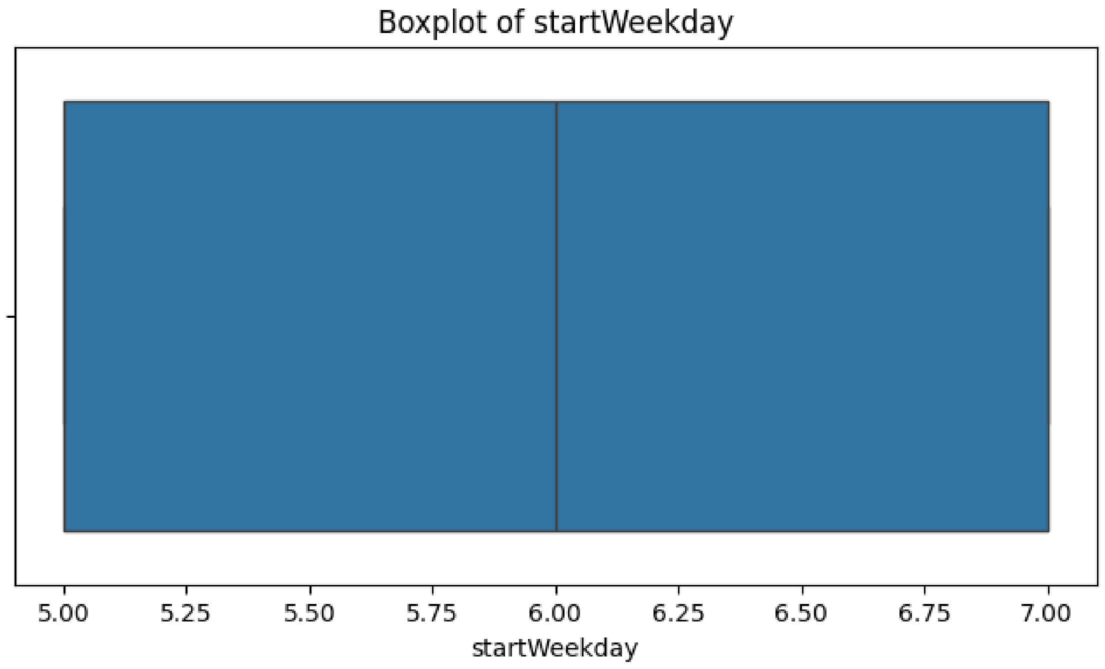
```
79           12.0        6.0           5.0   638.904    14.0      5.99
80           12.0        6.0           5.0   735.665    14.0      5.99
…            …        …            …       …      …       …
428739    49975.0       18.0           7.0   992.672    17.0       5.0
428740    49975.0       18.0           7.0  1054.158    17.0       5.0
428741    49975.0       18.0           7.0  1075.531    17.0       5.0
428742    49975.0       18.0           7.0  1127.911    17.0       5.0
428743    49975.0       18.0           7.0  1183.038    17.0       5.0

       cMaxPrice cSumPrice  bCount bMinPrice  …         availability  \
1          59.99     59.99     1.0     59.99  …  completely orderable
77          52.5    317.82     7.0      5.99  …  completely orderable
78          52.5    317.82     7.0      5.99  …  completely orderable
79          52.5    317.82     7.0      5.99  …  completely orderable
80          52.5    317.82     7.0      5.99  …  completely orderable
…            …       …       …         …  …                   …
428739    199.99     430.9     4.0      9.99  …  completely orderable
428740    199.99     430.9     4.0      9.99  …  completely orderable
428741    199.99     430.9     4.0      9.99  …  completely orderable
428742    199.99     430.9     4.0      9.99  …  completely orderable
428743    199.99     430.9     4.0      9.99  …  completely orderable

       customerNo  maxVal customerScore accountLifetime payments   age  \
1               1   600.0          70.0            21.0      1.0  43.0
77              8  2000.0         546.0           364.0     11.0  86.0
78              8  2000.0         546.0           364.0     11.0  86.0
79              8  2000.0         546.0           364.0     11.0  86.0
80              8  2000.0         546.0           364.0     11.0  86.0
…             …     …           …             …       …    …
428739      25024   600.0          70.0            98.0      0.0  47.0
428740      25024   600.0          70.0            98.0      0.0  47.0
428741      25024   600.0          70.0            98.0      0.0  47.0
428742      25024   600.0          70.0            98.0      0.0  47.0
428743      25024   600.0          70.0            98.0      0.0  47.0

        address  lastOrder  order
1             1       49.0      y
77            2       37.0      y
78            2       37.0      y
79            2       37.0      y
80            2       37.0      y
…           …         …    …
428739        2      488.0      y
428740        2      488.0      y
428741        2      488.0      y
428742        2      488.0      y
428743        2      488.0      y
```

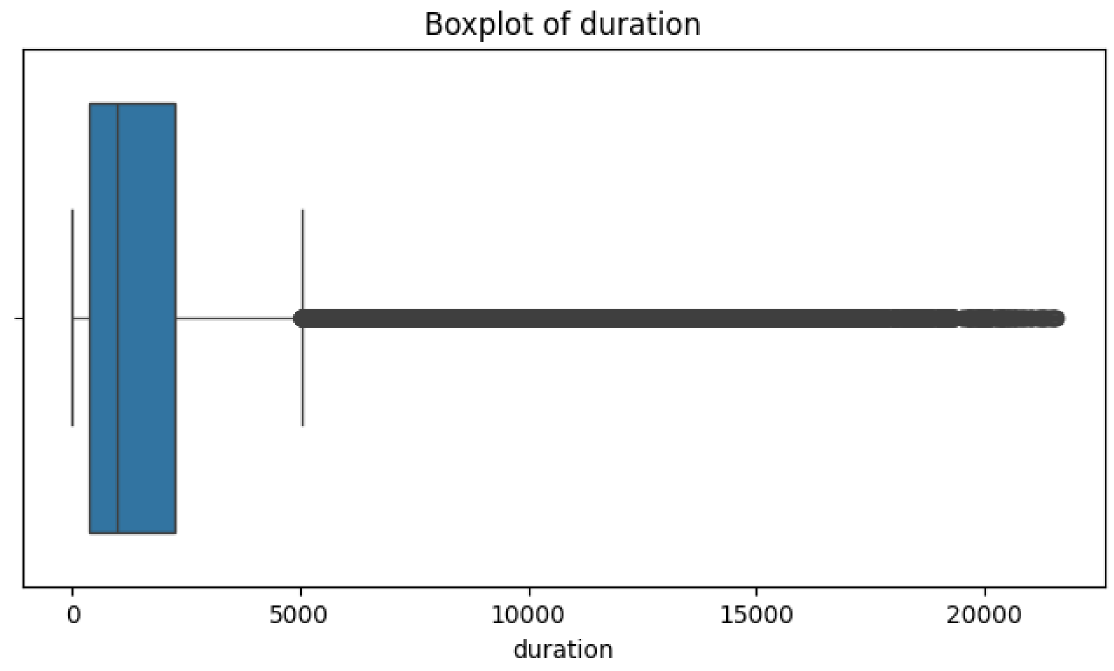
```
[24284 rows x 24 columns]
```

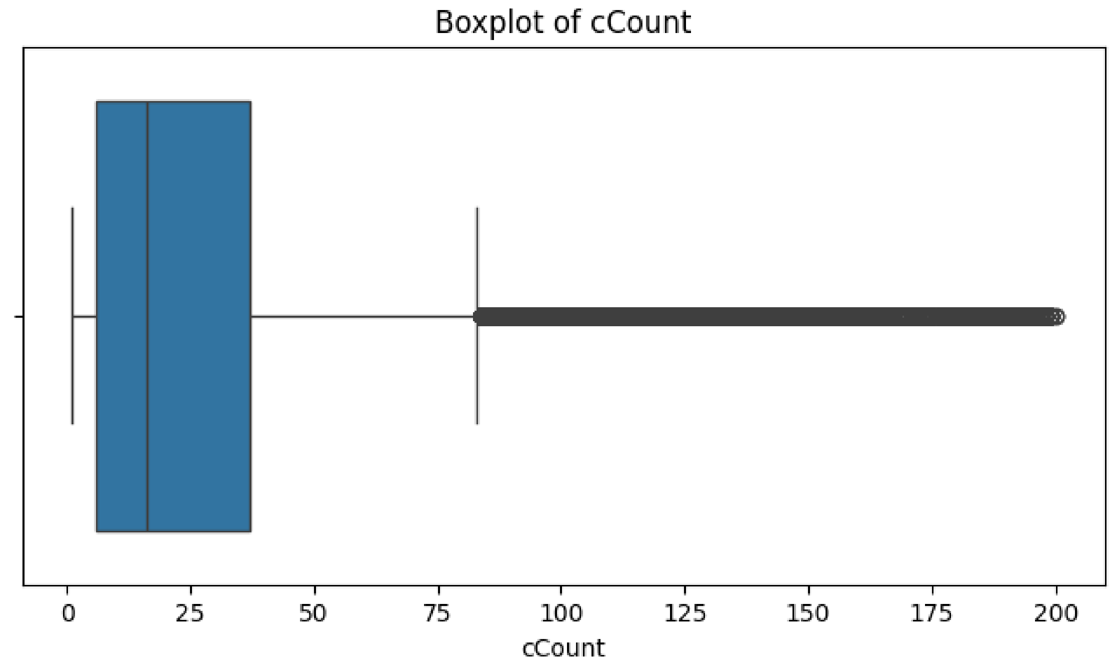
[21]:
```
# Boxplots for each numeric column to spot outliers
for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()
```

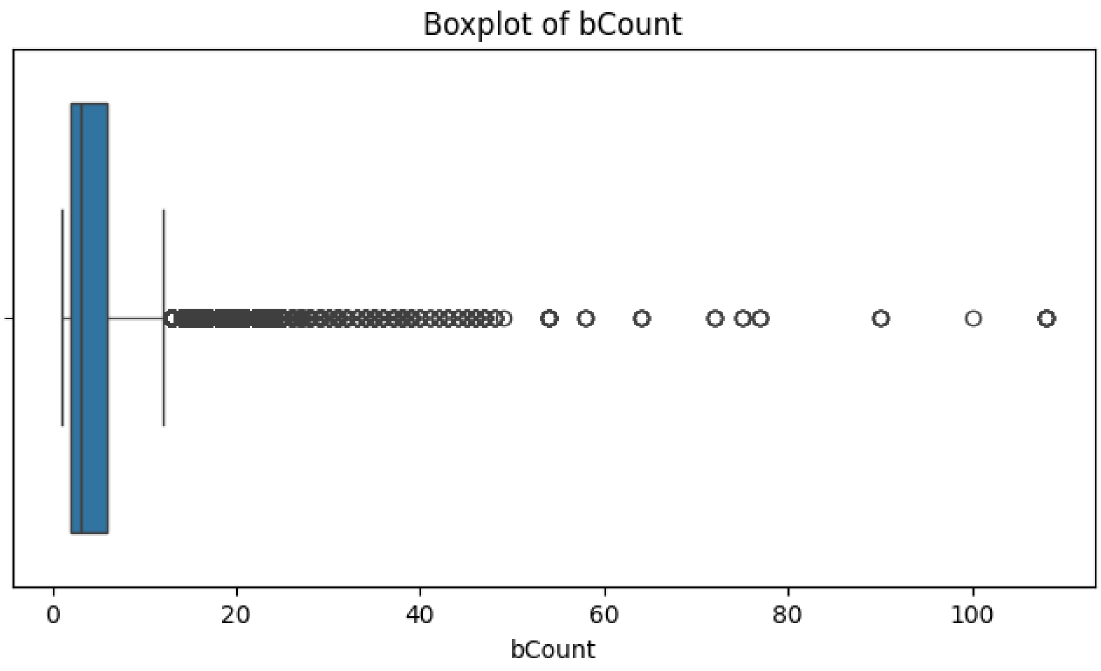


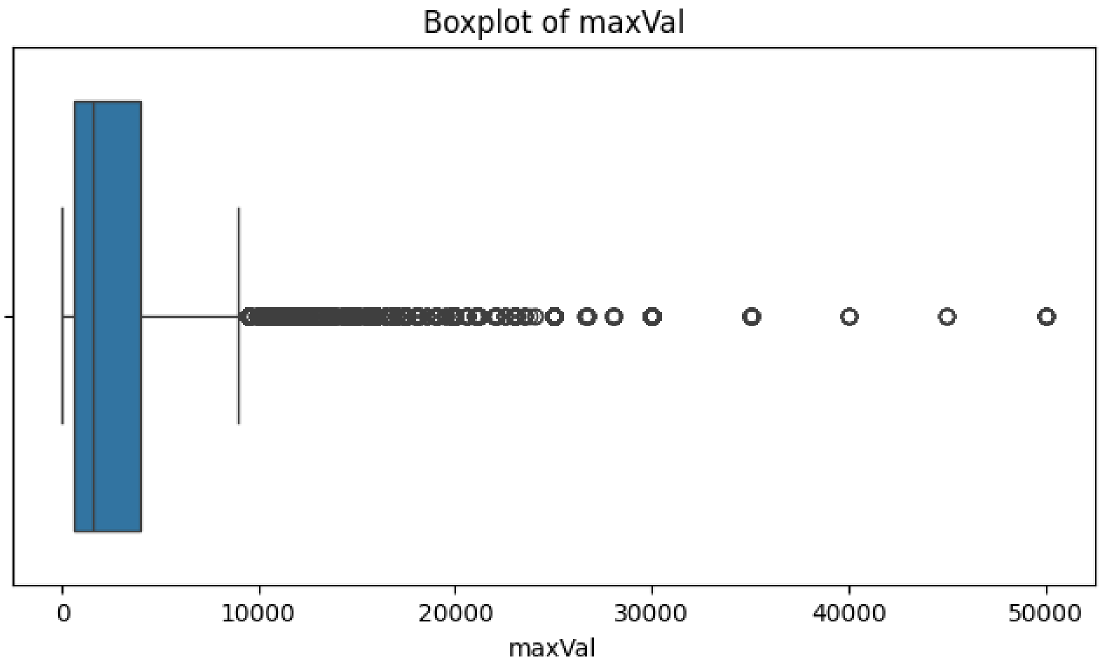

Boxplot of startHour
0
5
10
15
20
startHour

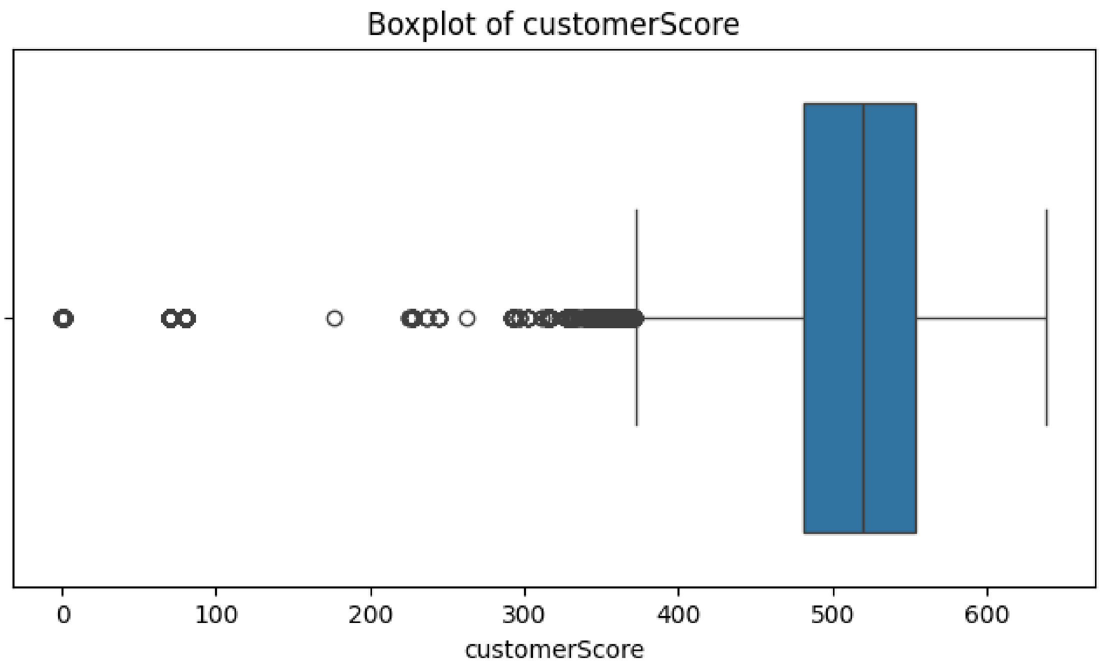

Boxplot of startWeekday
5.00
5.25
5.50
5.75
6.00
6.25
6.50
6.75
7.00
startWeekday

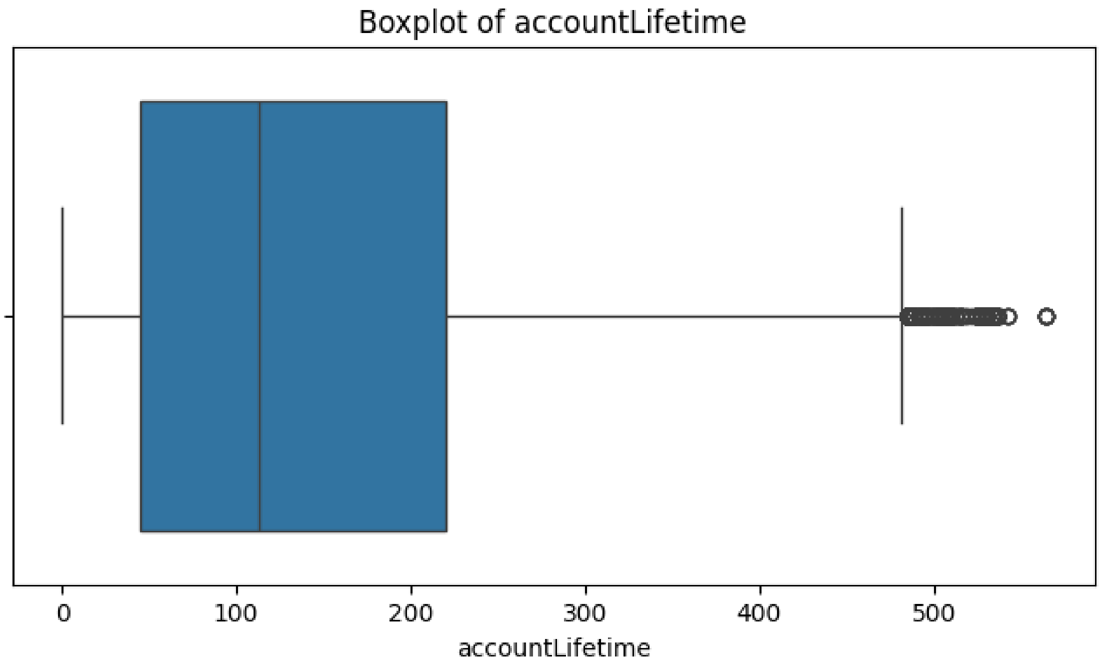

Boxplot of duration
0
5000
10000
15000
20000
duration

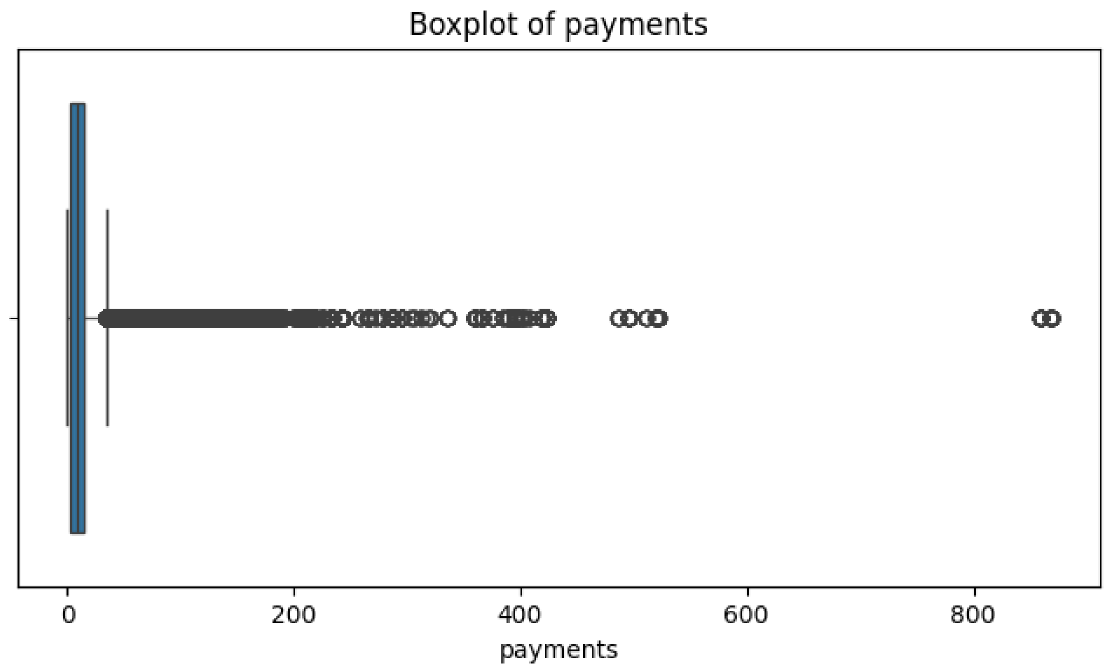

Boxplot of cCount
0
25
50
75
100
125
150
175
200
cCount


Boxplot of bCount
0
20
40
60
80
100
bCount

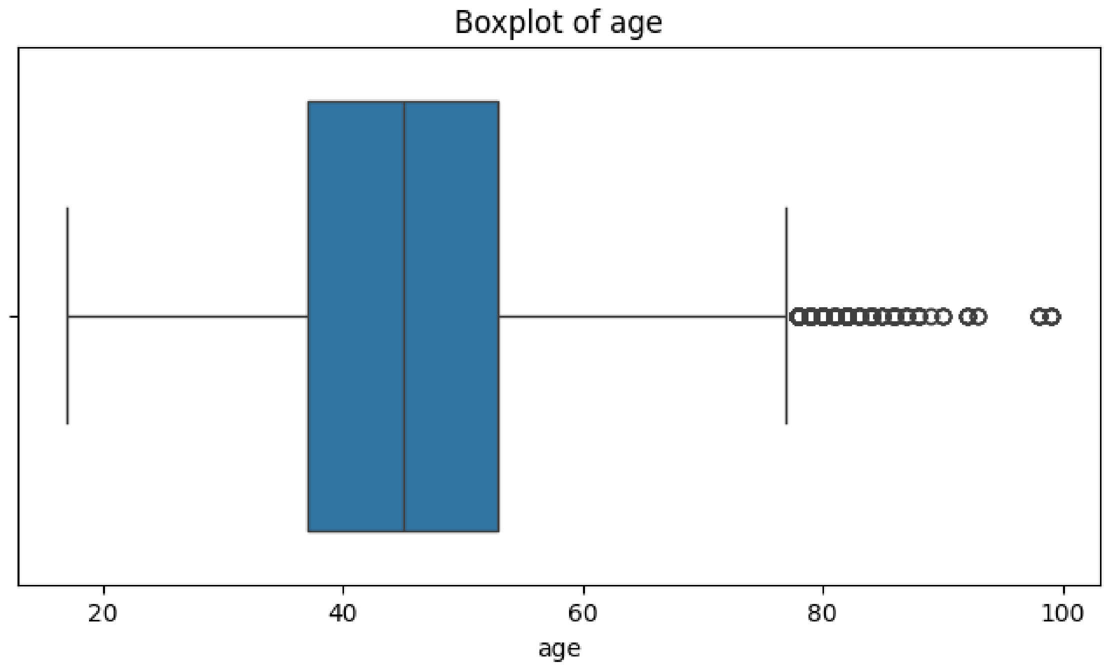

Boxplot of maxVal
0
10000
20000
30000
40000
50000
maxVal


Boxplot of customerScore
0
100
200
300
400
500
600
customerScore

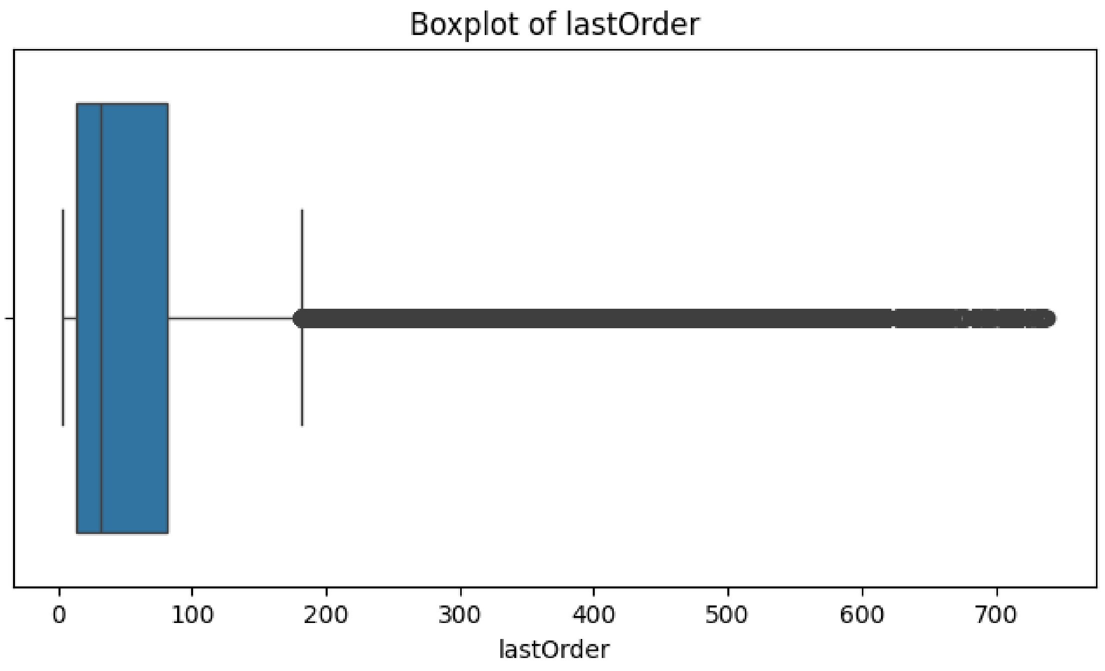

Boxplot of accountLifetime
0
100
200
300
400
500
accountLifetime



Boxplot of payments
0
200
400
600
800
payments



Boxplot of age
20
40
60
80
100
age



[22]:
```python
# Get unique values in categorical columns
for col in categorical_cols:
    print(f"Unique values in {col}:\n", df[col].unique())
```

```
Unique values in cMinPrice:
 ['59.99' '9.99' '12.99' '3.0' '5.99' '19.99' '0.8' '49.99' '14.99' '4.99'
 '29.99' '24.99' '139.99' '7.99' '5.0' '7.0' '3.99' '47.99' '49.95'
 '39.99' '2.99' '16.99' '44.99' '17.99' '1.5' '1.0' '54.99' '39.95' '13.9'
 '40.99' '349.99' '29.97' '23.99' '0.0' '79.82' '15.0' '599.0' '249.99'
 '4.97' '6.99' '69.99' '999.99' '159.99' '11.99' '15.99' '39.9' '6.47'
 '179.99' '12.0' '309.99' '279.99' '149.99' '11.98' '6.0' '299.99' '45.0'
 '27.99' '10.0' '469.99' '229.99' '10.99' '8.99' '18.0' '34.99' '0.19'
 '1199.0' '17.95' '79.99' '399.99' '34.95' '949.99' '169.99' '14.95'
 '14.0' '89.99' '4.0' '2.95' '16.8' '129.99' '13.99' '29.95' '379.99'
 '8.0' '129.0' '89.9' '119.95' '27.85' '99.99' '8.95' '499.99' '499.0'
 '20.0' '6.96' '22.99' '0.99' '62.99' '19.95' '189.99' '29.9' '2.75'
 '25.99' '29.0' '12.95' '449.99' '4.5' '9.0' '28.0' '8.39' '199.99'
 '56.99' '269.99' '7.95' '419.99' '8.9' '9.95' '19.0' '1.99' '119.0'
 '79.9' '15.95' '25.2' '729.99' '599.99' '24.9' '42.99' '79.95' '2.49'
 '25.0' '699.0' '4.95' '119.99' '59.95' '149.0' '46.0' '159.0' '67.99'
 '12.9' '10.92' '20.99' '699.99' '65.99' '10.95' '9.74' '489.99' '269.0'
 '609.99' '129.95' '1299.99' '11.9' '6.95' '19.9' '1799.99' '13.0' '24.97'
 '16.75' '7.9' '549.99' '2799.99' '799.0' '45.99' '16.95' '379.0' '69.95'
 '669.99' '649.99' '719.99' '179.0' '99.0' '9.2' '2.0' '99.95' '6.59'
 '64.95' '74.99' '24.95' '169.0' '12.49' '7.5' '1.95' '3.95' '3.5' '9.9'
 '209.99' '64.99' '18.99' '999.0' '13.49' '299.95' '13.19' '899.0'
 '2099.0' '359.99' '59.0' '1499.0' '36.85' '219.0' '12.8' '749.0' '19.97'
 '22.5' '154.99' '1.4' '32.99' '54.95' '109.99' '199.0' '319.99' '44.0'
 '17.5' '35.99' '39.96' '21.99' '31.99' '222.0' '889.99' '14.9' '5.95'
 '79.96' '70.99' '89.95' '7.49' '30.0' '21.0' '170.0' '259.99' '329.0'
 '159.95' '74.0' '333.0' '16.9' '5.5' '8.49' '19.32' '74.95' '9.06' '3.49'
 '749.9' '799.99' '749.99' '71.99' '22.0' '399.95' '139.0' '49.0' '11.95'
 '67.21' '9.69' '2.5' '899.99' '299.98' '6.39' '18.5' '36.99' '16.0'
 '329.99' '12.5' '180.0' '84.99' '27.95' '17.49' '16.79' '99.9' '9.6'
 '37.99' '19.3' '429.99' '69.0' '26.99' '44.9' '1189.0' '11.24' '219.99'
 '299.0' '49.9' '44.95' '529.0' '51.96' '4.9' '859.0' '5.49' '849.99'
 '28.76' '1739.0' '42.5' '1149.99' '55.95' '14.97' '1499.99' '42.95'
 '89.0' '44.85' '99.79' '1049.99' '17.0' '539.99' '339.99' '666.0' '50.0'
 '55.75' '750.0' '7.75' '58.82' '149.95' '35.95' '6.9' '5.9' '14.24'
 '33.74' '76.99' '33.99' '72.95' '1849.99' '7.96' '52.0' '1099.99'
 '1399.99' '2199.99' '261.99' '279.0' '99.8' '38.99' '38.64' '239.99'
 '35.0' '459.99' '144.99' '649.0' '19.5' '75.95' '12.98' '39.0' '12.97'
 '139.9' '289.99' '69.9' '6.5' '94.99' '22.95' '8.5' '9.56' '49.97'
 '359.0' '137.69' '479.0' '98.0' '579.99' '17.9' '759.9' '519.99' '798.99'
 '23.96' '4.75' '48.74' '1899.99' '11.49' '29.4' '659.99' '969.0' '752.52'
 '120.0' '34.96' '119.96' '819.0' '59.97' '10.49' '6.49' '6.75' '1.45'
 '48.96' '82.99' '30.99' '33.61' '8.75' '759.99' '1599.99' '72.99'
 '1199.99' '229.0' '289.0' '12.76' '859.99' '479.99' '23.95' '119.9'
 '16.49' '11.0' '1119.0' '1399.0' '65.6' '47.5' '3.9' '13.95' '259.0'
 '159.96' '221.73' '579.0' '85.99' '34.9' '55.96' '155.99' '500.0' '399.0'
 '23.9' '779.99' '9.97' '27.96' '77.0' '11.17' '79.0' '34.97' '2049.99'
 '639.99' '22.36' '25.95' '21.8' '32.49' '37.5' '75.99' '94.95' '429.0'
 '57.99' '629.0' '103.96' '28.99' '14.5' '2.77' '18.49' '559.99' '18.9'
 '13.96' '10.74' '16.5' '39.4' '5.7' '369.99' '1349.99' '10.9' '9.75'
 '369.0' '577.45' '41.95' '22.49' '21.95' '4.6' '32.95' '7.97' '33.98'
 '389.99' '4.45' '15.5' '41.96' '150.0' '39.97' '42.01' '46.99' '23.0'
 '190.0' '1699.0' '229.9' '9.8' '18.8' '79.79' '38.0' '591.0' '31.96'
 '134.99' '110.0' '15.9' '12.27' '26.24' '1699.99' '24.8' '10.5' '9.5'
 '215.0' '999.9' '55.99' '100.0' '1.25' '100.83' '40.0' '52.99' '4.49'
 '10.2' '70.0' '439.99' '11.96' '15.96' '679.99' '7.57' '53.99' '37.0'
 '919.0' '1599.0' '529.99' '159.2' '29.75' '18.95']
Unique values in cMaxPrice:
 ['59.99' '29.99' '179.95' '20.0' '40.0' '52.5' '34.99' '499.99' '14.99'
 '19.99' '33.99' '44.99' '139.99' '39.99' '24.99' '79.99' '29.95' '27.96'
 '47.99' '79.95' '120.0' '5.99' '9.99' '180.0' '18.99' '69.99' '329.99'
 '12.99' '49.99' '799.99' '4.99' '22.0' '169.95' '1399.99' '399.99'
 '39.95' '349.99' '79.9' '13.9' '89.99' '469.99' '69.95' '134.95' '159.99'
 '199.99' '99.99' '699.0' '299.99' '34.97' '45.99' '3.0' '85.99' '99.79'
 '7.99' '99.95' '189.95' '169.99' '999.99' '19.0' '15.99' '379.99' '22.99'
 '179.99' '739.99' '1099.0' '74.99' '10.0' '17.99' '309.99' '279.99'
 '89.95' '129.99' '49.95' '5.0' '479.99' '45.0' '27.99' '699.99' '229.99'
 '8.99' '12.0' '59.95' '16.99' '189.99' '64.99' '46.0' '1199.0' '119.99'
 '19.95' '34.95' '550.0' '54.99' '15.0' '949.99' '64.95' '219.0' '1399.0'
```

```
'319.99' '429.99' '45.95' '48.99' '119.95' '599.0' '28.76' '54.95'
'1299.99' '549.99' '164.99' '289.99' '529.99' '54.9' '1499.99' '719.99'
'459.99' '199.95' '899.0' '999.0' '249.99' '25.99' '59.9' '24.8' '17.95'
'7.0' '8.39' '3.99' '110.0' '4.0' '595.0' '16.79' '76.45' '359.0'
'449.99' '16.8' '29.9' '109.95' '899.99' '15.95' '100.83' '84.99' '29.0'
'104.99' '60.0' '28.0' '95.0' '21.0' '149.9' '30.99' '369.99' '140.0'
'419.99' '32.99' '149.99' '42.99' '69.9' '750.0' '35.99' '1049.99' '30.0'
'44.95' '119.0' '10.99' '94.95' '599.99' '729.99' '849.99' '34.9' '74.95'
'62.99' '129.95' '239.99' '149.0' '31.99' '159.0' '79.0' '749.0' '8.95'
'26.99' '35.95' '65.99' '1199.99' '489.99' '499.0' '269.0' '609.99'
'75.0' '159.95' '55.0' '149.95' '25.95' '1799.99' '38.97' '10.92' '21.99'
'91.5' '23.99' '579.99' '17.0' '32.0' '16.75' '135.0' '2799.99' '39.0'
'190.0' '799.0' '669.99' '89.9' '959.99' '27.95' '179.0' '58.82' '99.0'
'11.95' '39.9' '12.9' '289.0' '59.0' '105.0' '169.0' '219.99' '25.2'
'34.0' '299.0' '779.99' '269.99' '55.75' '65.0' '209.99' '500.0' '78.99'
'79.82' '74.0' '569.99' '399.0' '13.99' '14.8' '333.0' '51.99' '2599.0'
'759.99' '959.9' '1499.0' '36.85' '1599.99' '12.8' '1749.99' '85.0'
'6.99' '154.99' '35.0' '25.0' '150.0' '3.95' '29.4' '14.0' '129.0'
'199.0' '37.99' '14.95' '1899.99' '11.99' '52.0' '22.95' '649.99' '100.0'
'889.99' '20.99' '111.99' '775.0' '1999.99' '749.99' '42.01' '50.0'
'88.99' '6.0' '9.95' '170.0' '14.9' '94.99' '329.0' '109.99' '38.64'
'949.0' '144.99' '69.0' '99.9' '36.99' '65.95' '3.49' '63.99' '154.9'
'798.99' '929.99' '2099.0' '1449.9' '139.0' '639.0' '9.0' '27.5' '67.21'
'8.0' '1029.99' '1569.0' '299.98' '1099.99' '38.99' '47.96' '16.0'
'24.95' '18.5' '299.95' '2499.99' '379.0' '28.99' '591.0' '679.99' '0.0'
'250.0' '7.95' '1049.0' '2399.99' '31.96' '53.99' '44.9' '39.96' '6.95'
'44.0' '1449.99' '259.99' '13.49' '450.0' '1809.9' '137.69' '49.9'
'139.95' '279.0' '529.0' '888.0' '539.99' '51.96' '409.99' '161.8'
'859.0' '659.99' '1.99' '1739.0' '1149.99' '239.0' '24.9' '349.0' '249.0'
'33.74' '89.0' '2999.99' '16.95' '261.99' '27.85' '103.96' '26.9' '55.96'
'24.0' '16.49' '2.99' '7.5' '37.95' '79.96' '339.99' '666.0' '1999.9'
'29.98' '389.99' '134.99' '709.95' '19.3' '24.5' '109.0' '499.9'
'2249.99' '222.0' '70.0' '19.9' '12.95' '52.99' '37.0' '119.96' '1849.99'
'55.99' '32.49' '125.0' '32.4' '2199.99' '32.95' '96.85' '577.45' '12.49'
'17.5' '99.8' '75.62' '124.95' '199.9' '25.16' '19.32' '1.0' '359.99'
'67.99' '40.85' '189.0' '9.74' '41.0' '36.0' '919.99' '39.4' '126.04'
'26.24' '53.5' '100.5' '41.99' '12.98' '139.9' '13.0' '64.75' '3299.99'
'37.5' '135.96' '879.99' '124.99' '22.9' '15.5' '80.0' '49.0' '9.5'
'439.99' '1449.0' '1699.0' '1122.0' '98.0' '71.95' '759.9' '519.99'
'129.9' '17.9' '229.0' '249.95' '33.61' '859.9' '629.99' '15.9' '247.12'
'549.9' '188.99' '3394.03' '27.0' '1759.99' '2299.99' '869.0' '67.22'
'74.9' '819.0' '52.95' '49.5' '26.95' '130.0' '185.0' '18.0' '1549.99'
'165.0' '819.99' '869.99' '969.99' '58.0' '5.95' '6999.99' '59.96'
'2449.99' '90.0' '91.99' '57.85' '1009.95' '72.99' '1998.99' '3499.99'
'4999.99' '69.74' '34.5' '859.99' '83.96' '46.99' '69.96' '79.79' '159.9'
'45.9' '1119.0' '42.95' '649.0' '47.5' '159.96' '177.0' '259.0' '17.49'
'999.9' '183.96' '221.73' '579.0' '1695.0' '47.95' '155.99' '21.95'
'1299.0' '13.95' '1349.99' '14.97' '9.69' '559.99' '167.99' '20.76'
```

```
 '33.3' '2049.99' '177.99' '639.99' '33.0' '619.99' '379.9' '260.49'
 '21.8' '449.0' '42.9' '19.5' '469.0' '1699.99' '67.95' '57.99' '1199.9'
 '31.95' '55.95' '42.5' '39.16' '2299.0' '44.4' '28.5' '2700.0' '629.0'
 '26.0' '2.0' '48.0' '16.5' '23.96' '195.0' '29.97' '319.32' '160.0'
 '2099.99' '19.97' '184.95' '1159.9' '82.99' '11.9' '23.0' '9.9' '1559.99'
 '15.96' '114.99' '849.0' '198.0' '369.0' '75.95' '73.0' '86.99' '359.95'
 '1519.99' '43.99' '229.9' '1039.99' '598.0' '85.95' '1799.0' '33.98'
 '10.95' '759.0' '2149.99' '84.95' '5999.99' '64.9' '34.36' '70.99'
 '1249.99' '32.9' '449.9' '59.97' '30.95' '1079.99' '151.25' '42.7'
 '197.47' '200.0' '116.99' '15.49' '602.0' '21.9' '239.95' '684.0' '38.0'
 '107.95' '909.9' '1066.03' '689.99' '709.9' '1599.0' '21.5' '209.95'
 '37.49' '66.0' '164.0' '80.99' '3999.99' '37.81' '45.49' '215.0' '1059.9'
 '75.9' '581.03' '115.0' '84.49' '77.99' '829.99' '20.49' '43.96' '269.95'
 '709.99' '1229.99' '919.0' '50.41' '85.9' '35.9' '43.5' '225.0' '75.99']
Unique values in cSumPrice:
 ['59.99' '109.95' '1093.72' … '951.32' '971.31' '319.6']
Unique values in bMinPrice:
 ['59.99' '9.99' '19.99' '3.0' '5.99' '29.99' '1.99' '49.99' '14.99' '6.0'
 '24.99' '27.99' '139.99' '12.99' '5.0' '3.99' '47.99' '49.95' '39.99'
 '2.99' '7.99' '16.99' '44.99' '12.49' '1.5' '1.0' '39.95' '13.9' '40.99'
 '349.99' '34.9' '23.99' '99.95' '79.95' '79.82' '699.0' '249.99' '7.0'
 '15.99' '17.99' '6.99' '69.99' '149.99' '999.99' '159.99' '11.99'
 '229.99' '69.9' '15.0' '179.99' '159.0' '12.0' '10.0' '10.99' '18.99'
 '309.99' '279.99' '199.99' '4.99' '299.99' '8.39' '45.0' '469.99' '19.0'
 '8.99' '429.99' '19.95' '18.0' '34.99' '149.0' '1199.0' '17.95' '79.99'
 '399.99' '34.95' '99.99' '54.99' '949.99' '14.0' '599.0' '4.0' '2.95'
 '1299.99' '17.5' '16.8' '249.0' '13.99' '799.99' '59.95' '29.95' '719.99'
 '379.99' '459.99' '8.0' '499.0' '129.0' '119.96' '27.85' '22.99' '9.95'
 '9.0' '499.99' '24.8' '20.0' '20.96' '0.99' '595.0' '89.9' '62.99'
 '76.45' '189.99' '29.9' '2.75' '65.95' '25.99' '0.0' '29.0' '449.99'
 '20.99' '28.0' '8.95' '94.99' '21.0' '30.99' '269.99' '32.0' '419.99'
 '45.95' '29.4' '750.0' '529.99' '89.99' '16.0' '119.0' '79.9' '15.95'
 '21.99' '129.99' '25.2' '729.99' '599.99' '74.99' '24.9' '42.99' '2.49'
 '71.99' '25.0' '4.95' '7.95' '119.99' '67.99' '10.92' '64.99' '749.0'
 '699.99' '65.99' '13.0' '54.9' '9.74' '489.99' '269.0' '609.99' '129.95'
 '159.95' '14.95' '11.9' '9.9' '999.0' '6.95' '1799.99' '46.0' '47.95'
 '24.97' '45.99' '16.75' '2799.99' '40.0' '799.0' '193.99' '379.0' '69.95'
 '669.99' '959.99' '179.0' '99.0' '9.2' '14.9' '12.9' '6.59' '64.95' '8.9'
 '19.9' '37.95' '169.0' '5.49' '13.95' '39.9' '3.5' '209.99' '549.99'
 '14.36' '13.19' '899.0' '3.95' '79.0' '2099.0' '7.5' '389.99' '759.99'
 '4.5' '1499.0' '24.95' '36.85' '219.0' '439.99' '12.8' '899.99' '154.99'
 '32.99' '109.99' '5.95' '199.0' '319.99' '54.95' '35.99' '22.95' '39.96'
 '63.99' '649.99' '100.0' '5.9' '31.99' '18.5' '889.99' '37.99' '79.96'
 '70.99' '299.0' '10.95' '170.0' '169.99' '259.99' '329.0' '178.95' '74.0'
 '16.9' '19.32' '74.95' '36.99' '3.49' '809.9' '119.95' '239.99' '22.0'
 '139.0' '2.0' '279.0' '67.21' '1569.0' '299.98' '329.99' '333.0' '12.95'
 '16.24' '25.95' '180.0' '84.99' '30.0' '16.79' '99.9' '19.3' '26.99'
 '44.9' '1189.0' '8.49' '219.99' '3.9' '749.99' '49.9' '89.95' '529.0'
 '51.96' '5.5' '859.0' '7.96' '2.5' '849.99' '1739.0' '52.0' '1149.99'
 '11.95' '1599.99' '42.95' '9.56' '20.9' '89.0' '44.85' '1999.99' '16.95'
 '34.97' '26.9' '10.5' '1049.99' '539.99' '339.99' '666.0' '50.0' '55.75'
 '58.82' '17.0' '149.95' '35.95' '11.49' '0.8' '72.0' '33.74' '17.97'
 '222.0' '76.99' '59.9' '59.0' '4.9' '444.0' '1849.99' '1099.99' '1399.99'
 '44.95' '125.0' '33.99' '31.96' '2199.99' '137.69' '99.8' '38.99' '65.0'
 '38.64' '35.0' '144.99' '649.0' '359.99' '28.95' '19.5' '32.49' '479.99'
 '12.98' '39.0' '33.61' '169.95' '53.5' '139.9' '1.95' '15.9' '1099.0'
 '299.95' '21.95' '879.99' '289.99' '224.95' '6.9' '150.0' '359.0' '7.16'
 '98.0' '579.99' '22.9' '759.9' '519.99' '369.99' '798.99' '7.9' '120.0'
 '8.5' '409.99' '4.75' '66.0' '48.74' '1899.99' '9.97' '659.99' '969.0'
 '6.96' '6.5' '819.0' '7.49' '17.9' '229.0' '10.49' '6.49' '165.0' '82.99'
 '629.99' '134.99' '239.0' '8.75' '17.49' '888.0' '55.0' '15.5' '72.99'
 '1998.99' '2699.99' '24.0' '13.96' '859.99' '51.99' '48.99' '46.99'
 '51.95' '119.9' '16.49' '44.0' '11.0' '1119.0' '1399.0' '47.5' '26.24'
 '259.0' '159.96' '221.73' '579.0' '85.99' '43.5' '1049.0' '155.99'
 '1299.0' '500.0' '399.0' '22.49' '9.69' '779.99' '591.0' '161.8' '0.19'
 '27.96' '11.17' '13.49' '261.99' '2049.99' '639.99' '1199.99' '619.99'
 '22.36' '55.96' '21.8' '104.96' '7.97' '37.5' '75.99' '94.95' '10.36'
 '75.95' '45.9' '57.99' '31.95' '2.77' '1.45' '19.97' '100.83' '779.0'
 '1499.99' '16.5' '23.96' '39.4' '5.7' '109.0' '149.9' '11.5' '104.99'
 '1349.99' '10.9' '369.0' '139.95' '577.45' '28.99' '1.2' '96.99' '11.96'
 '4.6' '32.95' '33.98' '9.75' '42.01' '48.96' '23.9' '110.0' '5.87'
 '190.0' '1699.0' '49.0' '229.9' '27.95' '99.79' '79.79' '21.9' '38.0'
 '84.49' '69.0' '629.0' '18.36' '18.16' '42.7' '12.27' '53.0' '37.49'
 '18.49' '215.0' '1059.9' '6999.99' '10.2' '70.0' '15.96' '709.99' '7.57'
 '53.99' '37.0' '919.0' '1599.0' '39.16' '36.0' '26.95' '29.8' '18.95']
Unique values in bMaxPrice:
 ['59.99' '29.99' '27.85' '3.0' '52.5' '19.99' '49.99' '14.99' '7.99'
 '12.99' '139.99' '39.99' '24.99' '25.95' '47.99' '71.96' '89.99' '2.99'
 '9.99' '150.0' '18.99' '69.99' '299.99' '799.99' '4.99' '22.0' '39.95'
 '349.99' '13.9' '40.99' '399.99' '34.99' '49.95' '33.99' '99.95' '5.0'
 '79.99' '199.99' '699.0' '249.99' '34.97' '43.99' '12.49' '44.99' '39.9'
 '149.99' '999.99' '15.0' '16.99' '159.99' '29.95' '229.99' '22.99' '69.9'
 '179.99' '8.99' '159.0' '69.95' '74.99' '10.0' '21.99' '17.99' '19.32'
 '309.99' '279.99' '15.99' '64.99' '45.0' '27.99' '469.99' '19.0' '36.99'
 '429.99' '169.99' '19.97' '3.99' '59.95' '149.0' '1199.0' '119.99' '59.9'
 '5.99' '19.95' '34.95' '299.0' '54.99' '7.0' '949.99' '64.95' '219.0'
 '48.99' '599.0' '54.95' '1299.99' '99.99' '16.8' '249.0' '529.99' '54.9'
 '79.95' '719.99' '459.99' '119.96' '45.99' '10.99' '6.95' '25.99'
 '499.99' '50.0' '24.8' '20.0' '17.95' '169.95' '70.99' '20.96' '4.0'
 '595.0' '62.99' '16.79' '76.45' '189.99' '29.9' '109.95' '65.95' '1.0'
 '699.99' '15.95' '100.83' '29.0' '449.99' '89.95' '27.95' '129.99' '28.0'
 '94.99' '21.0' '30.99' '32.0' '369.99' '419.99' '45.95' '42.99' '25.2'
 '55.0' '750.0' '35.99' '30.0' '44.95' '119.0' '79.9' '219.99' '729.99'
 '599.99' '319.99' '74.95' '71.99' '75.95' '14.95' '24.9' '6.99' '749.0'
 '8.95' '65.99' '84.99' '55.75' '71.95' '489.99' '269.0' '609.99' '129.95'
 '49.9' '60.0' '159.95' '999.0' '26.99' '1799.99' '46.0' '47.95' '38.97'
 '10.92' '1099.99' '17.0' '16.75' '85.99' '1399.99' '2799.99' '40.0'
 '261.99' '190.0' '799.0' '193.99' '379.0' '669.99' '79.0' '959.99'
 '179.0' '58.82' '99.0' '479.99' '11.95' '9.2' '37.99' '12.9' '899.0'
 '32.99' '59.0' '37.95' '13.99' '169.0' '579.99' '379.99' '12.0' '269.99'
 '9.9' '209.99' '549.99' '500.0' '79.82' '14.36' '569.99' '499.0' '3.95'
 '333.0' '3.9' '2099.0' '759.99' '959.9' '1499.0' '36.85' '12.8' '1749.99'
 '899.99' '154.99' '16.0' '74.0' '29.4' '129.0' '109.99' '97.99' '199.0'
 '25.0' '22.95' '1149.99' '11.99' '849.99' '649.99' '100.0' '889.99'
 '20.99' '183.96' '775.0' '749.99' '31.99' '6.2' '105.0' '42.01' '6.0'
 '9.95' '170.0' '14.9' '259.99' '329.0' '178.95' '198.95' '44.0' '38.64'
 '14.0' '34.9' '37.81' '24.95' '69.0' '329.99' '99.79' '3.49' '909.9'
 '23.99' '8.39' '119.95' '239.99' '18.0' '27.5' '139.0' '35.0' '279.0'
 '67.21' '9.0' '42.0' '1569.0' '299.98' '12.95' '38.99' '43.96' '109.0'
 '35.95' '18.5' '0.0' '180.0' '189.0' '7.95' '99.9' '44.9' '1189.0'
 '13.49' '137.69' '139.95' '529.0' '82.0' '51.96' '5.5' '859.0' '8.49'
 '1.99' '47.96' '539.99' '1739.0' '52.0' '161.8' '1599.99' '63.96'
 '798.99' '89.0' '28.99' '1999.99' '16.95' '26.9' '22.36' '199.95'
 '289.99' '47.5' '1049.99' '149.95' '7.5' '79.96' '339.99' '666.0' '749.9'
 '29.98' '389.99' '189.95' '709.95' '19.3' '22.9' '55.96' '134.99' '11.49'
 '8.0' '222.0' '79.79' '76.99' '19.9' '125.0' '52.99' '39.96' '37.0' '4.9'
 '444.0' '1849.99' '55.99' '1199.99' '32.49' '1499.99' '39.0' '11.9'
 '13.96' '2199.99' '24.0' '89.9' '577.45' '17.5' '99.8' '65.0' '23.96'
 '140.0' '144.99' '649.0' '359.99' '28.95' '67.99' '19.5' '40.85' '80.0'
 '299.95' '9.74' '8.9' '33.61' '39.4' '26.24' '53.5' '41.99' '12.98'
 '139.9' '16.24' '1099.0' '659.99' '13.0' '21.95' '879.99' '124.99'
 '224.95' '49.0' '359.0' '98.0' '759.9' '519.99' '1039.99' '129.9' '120.0'
 '17.9' '409.99' '89.71' '1899.99' '969.0' '369.0' '67.22' '819.0' '52.95'
 '49.5' '229.0' '130.0' '165.0' '82.99' '629.99' '6999.99' '239.0' '888.0'
 '12.56' '72.99' '1998.99' '2699.99' '3499.99' '31.96' '18.95' '859.99'
 '51.99' '83.96' '46.99' '15.9' '159.9' '45.9' '10.95' '1119.0' '29.75'
 '1399.0' '259.0' '17.49' '159.96' '221.73' '32.95' '579.0' '1049.0'
 '155.99' '94.95' '1299.0' '54.0' '13.95' '14.97' '9.69' '779.99' '591.0'
 '11.17' '20.76' '15.49' '2049.99' '639.99' '24.74' '33.0' '619.99'
 '399.0' '15.96' '21.8' '5.95' '37.5' '549.9' '10.36' '116.99' '57.99'
 '31.95' '6.59' '55.95' '179.95' '28.5' '629.0' '2.0' '779.0' '16.5' '7.9'
 '177.99' '42.7' '1449.0' '184.95' '1159.9' '23.0' '149.9' '439.99'
 '104.99' '1349.99' '42.95' '6.9' '103.96' '1099.9' '85.95' '4.95' '33.98'
 '15.5' '75.0' '23.9' '559.0' '110.0' '59.97' '30.95' '1699.0' '229.9'
 '21.9' '27.96' '38.0' '84.49' '1066.03' '2.95' '689.99' '679.99' '21.5'
 '51.95' '126.04' '34.0' '53.0' '37.49' '10.5' '215.0' '1059.9' '581.03'
 '777.0' '77.99' '66.99' '20.49' '0.8' '164.9' '709.99' '53.99' '919.0'
 '36.0' '39.16' '35.9' '43.5' '84.95' '34.5' '75.99' '26.95' '29.8']
Unique values in bSumPrice:
 ['59.99' '39.98' '103.54' … '93.44' '329.96' '30.59']
Unique values in bStep:
 ['2' '4' '1' '3' '5']
Unique values in onlineStatus:
 ['y' 'n']
```

```
Unique values in availability:
 ['completely orderable' 'mainly orderable' 'completely not orderable'
 'mixed' 'mainly not orderable' 'completely not determinable'
 'mainly not determinable']
Unique values in order:
 ['y' 'n']
Unique values in customerNo:
 ['1' '3' '4' … '25035' '25036' '25037']
Unique values in address:
 ['1' '2' '3']
```

[23]:
```
#Convert all to lowercase
df[categorical_cols] = df[categorical_cols].apply(lambda x: x.str.lower())
```

[24]:
```
# Check logical consistency between min and max price columns
inconsistent_prices = df[df['cMinPrice'] > df['cMaxPrice']]
print("Rows with inconsistent price values:\n", inconsistent_prices)
```

```
Rows with inconsistent price values:
         sessionNo  startHour  startWeekday  duration  cCount cMinPrice  \
11            3.0        6.0           5.0   324.278    11.0      9.99
27            7.0        6.0           5.0   268.713     6.0       3.0
28            7.0        6.0           5.0   274.297     6.0       3.0
29            7.0        6.0           5.0   286.562     6.0       3.0
31            7.0        6.0           5.0   304.672     6.0       3.0
…               …          …             …         …       …
428953    49993.0       18.0           7.0  3866.511    69.0      9.99
428954    49993.0       18.0           7.0  3915.585    69.0      9.99
428955    49993.0       18.0           7.0  4094.847    69.0      9.99
428956    49993.0       18.0           7.0  4113.213    69.0      9.99
428972    49995.0       18.0           7.0   572.544    22.0      9.99

       cMaxPrice cSumPrice  bCount bMinPrice  …          availability  \
11         29.99    109.95     2.0      9.99  …  completely orderable
27          20.0      73.0     1.0       3.0  …  completely orderable
28          20.0      73.0     1.0       3.0  …  completely orderable
29          20.0      73.0     1.0       3.0  …  completely orderable
31          20.0      73.0     1.0       3.0  …  completely orderable
…            …        …      …         …  …                   …
428953     24.99    971.31    15.0      9.99  …  completely orderable
428954     24.99    971.31    15.0      9.99  …  completely orderable
428955     24.99    971.31    15.0      9.99  …  completely orderable
428956     24.99    971.31    15.0      9.99  …  completely orderable
428972     19.99     319.6     2.0      9.99  …  completely orderable

       customerNo  maxVal customerScore accountLifetime payments   age  \
11              3  1800.0         475.0           302.0     12.0  45.0
27              5   900.0         575.0            35.0     10.0  31.0
```

```
28              5   900.0         575.0            35.0     10.0  31.0
29              5   900.0         575.0            35.0     10.0  31.0
31              5   900.0         575.0            35.0     10.0  31.0
…             …   …           …             …       …   …
428953      25036   300.0         503.0            25.0      0.0  54.0
428954      25036   300.0         503.0            25.0      0.0  54.0
428955      25036   300.0         503.0            25.0      0.0  54.0
428956      25036   300.0         503.0            25.0      0.0  54.0
428972      25037   800.0         522.0            63.0      2.0  42.0

        address  lastOrder  order
11            1       11.0      y
27            2       10.0      y
28            2       10.0      y
29            2       10.0      y
31            2       10.0      y
…           …        …    …
428953        2       45.0      n
428954        2       45.0      n
428955        2       45.0      n
428956        2       45.0      n
428972        2        9.0      n

[54477 rows x 24 columns]
```

```
[25]: # Correct inconsistent price values
df.loc[df['cMinPrice'] > df['cMaxPrice'], ['cMinPrice', 'cMaxPrice']] = df.
 ↪loc[df['cMinPrice'] > df['cMaxPrice'], ['cMaxPrice', 'cMinPrice']].values
```

```
[26]: print(df.describe(include='all'))
```

```
            sessionNo      startHour   startWeekday       duration  \
count   141163.000000  141163.000000  141163.000000  141163.000000
unique            NaN            NaN            NaN            NaN
top               NaN            NaN            NaN            NaN
freq              NaN            NaN            NaN            NaN
mean     25271.805494      14.662865       5.924555    1838.816338
std      14442.609194       4.324934       0.787167    2512.450329
min          1.000000       0.000000       5.000000       0.062000
25%      12702.000000      11.000000       5.000000     383.329000
50%      25482.000000      15.000000       6.000000     992.864000
75%      37533.000000      18.000000       7.000000    2245.432500
max      49995.000000      23.000000       7.000000   21553.323000

               cCount cMinPrice cMaxPrice cSumPrice         bCount bMinPrice  \
count   141163.000000    141163    141163    141163  141163.000000    141163
unique            NaN       611       587     26254            NaN       522
top               NaN     19.99      9.99     39.98            NaN      9.99
```

```
freq          NaN     12792     19512       774           NaN     20313
mean    28.235557       NaN       NaN       NaN      4.865347       NaN
std     32.808797       NaN       NaN       NaN      4.728091       NaN
min      1.000000       NaN       NaN       NaN      1.000000       NaN
25%      6.000000       NaN       NaN       NaN      2.000000       NaN
50%     16.000000       NaN       NaN       NaN      3.000000       NaN
75%     37.000000       NaN       NaN       NaN      6.000000       NaN
max    200.000000       NaN       NaN       NaN    108.000000       NaN

        …          availability customerNo         maxVal  customerScore  \
count   …                141163     141163  141163.000000  141163.000000
unique  …                     7      21164            NaN            NaN
top     …  completely orderable       5464            NaN            NaN
freq    …                134756        268            NaN            NaN
mean    …                   NaN        NaN    2636.787260     486.201823
std     …                   NaN        NaN    3241.472901     128.959337
min     …                   NaN        NaN       0.000000       0.000000
25%     …                   NaN        NaN     600.000000     481.000000
50%     …                   NaN        NaN    1600.000000     520.000000
75%     …                   NaN        NaN    4000.000000     554.000000
max     …                   NaN        NaN   50000.000000     638.000000

       accountLifetime       payments            age  address      lastOrder  \
count    141163.000000  141163.000000  141163.000000   141163  141163.000000
unique             NaN            NaN            NaN        3            NaN
top                NaN            NaN            NaN        2            NaN
freq               NaN            NaN            NaN   103294            NaN
mean        138.734031      17.082479      45.247593      NaN      77.973208
std         110.553214      38.547387      11.943082      NaN     113.282558
min           0.000000       0.000000      17.000000      NaN       3.000000
25%          45.000000       3.000000      37.000000      NaN      14.000000
50%         113.000000       9.000000      45.000000      NaN      32.000000
75%         220.000000      16.000000      53.000000      NaN      81.000000
max         564.000000     868.000000      99.000000      NaN     738.000000

         order
count   141163
unique       2
top          y
freq    114781
mean       NaN
std        NaN
min        NaN
25%        NaN
50%        NaN
75%        NaN
max        NaN
```

```
[11 rows x 24 columns]
```

[33]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix


# Initialize LabelEncoder
encoder = LabelEncoder()

# Iterate through columns of X_train and encode object (string) types
for col in X_train.select_dtypes(include=['object']).columns:
    # Fit on the combined unique values from both training and testing data
    all_values = pd.concat([X_train[col], X_test[col]]).unique()
    encoder.fit(all_values)

    X_train[col] = encoder.transform(X_train[col])
    X_test[col] = encoder.transform(X_test[col]) # Apply the same encoding to␣
 ↪X_test

# Create and train the RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on the test data  #This line is added to get predictions␣
 ↪from the model
y_pred_rf = rf_model.predict(X_test)

# Hitung akurasi
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy (Random Forest):", accuracy_rf)

# Tampilkan laporan klasifikasi dan matriks konfusi
print("Classification Report (Random Forest):\n", classification_report(y_test,␣
 ↪y_pred_rf))
print("Confusion Matrix (Random Forest):\n", confusion_matrix(y_test,␣
 ↪y_pred_rf))
```

```
Accuracy (Random Forest): 0.957283524994687
Classification Report (Random Forest):
               precision    recall  f1-score   support

           0       0.99      0.78      0.87      7884
           1       0.95      1.00      0.97     34465

    accuracy                           0.96     42349
```

```
   macro avg       0.97      0.89      0.92     42349
weighted avg       0.96      0.96      0.96     42349

Confusion Matrix (Random Forest):
 [[ 6117  1767]
 [   42 34423]]
```

0.2 Analisis Model Klasifikasi

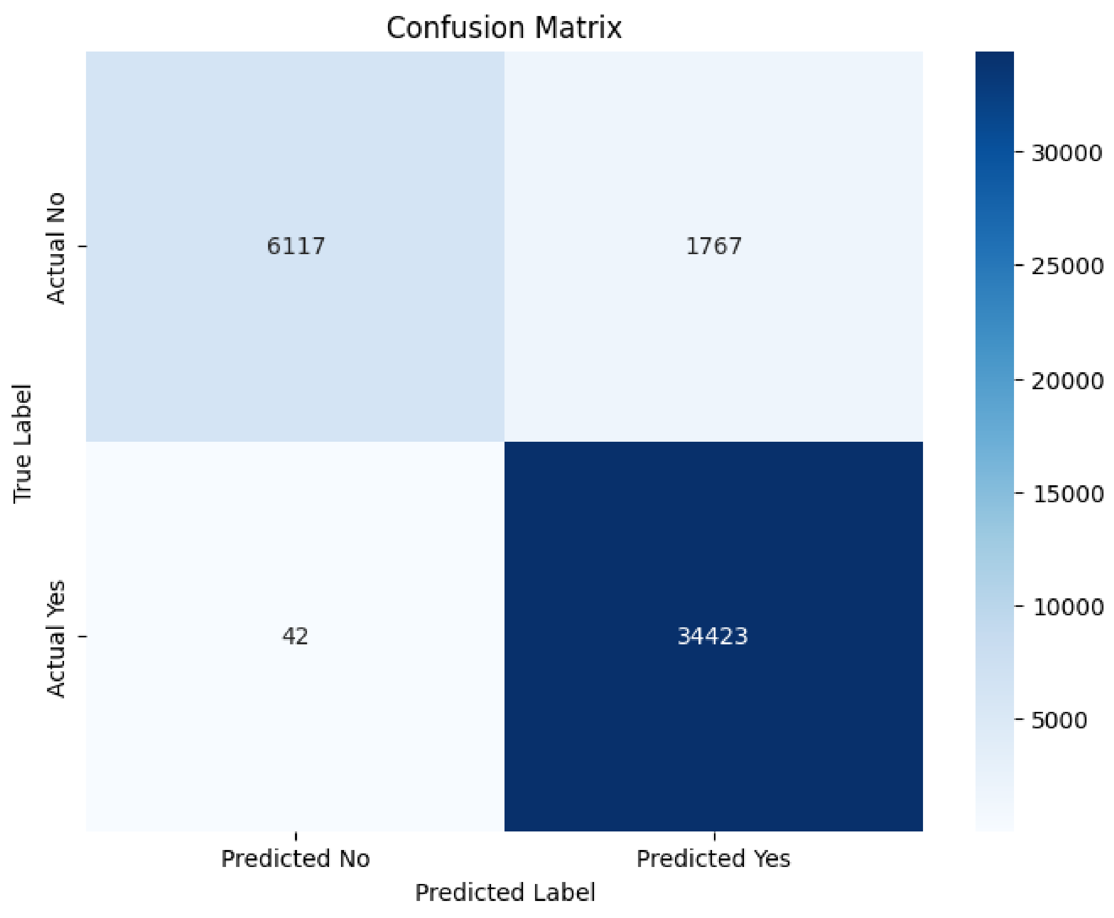
Model Random Forest menunjukkan kinerja yang sangat baik dalam memprediksi kategori "Order," dengan akurasi keseluruhan sebesar 95,7%. Untuk kategori "Order," model ini memiliki precision sebesar 95% dan recall 100%, yang berarti model sangat efektif dalam mendeteksi data yang benar-benar melakukan "Order." Di sisi lain, untuk kategori "Tidak Order," precision mencapai 99%, namun recall-nya lebih rendah, yaitu 78%. Hal ini menunjukkan bahwa beberapa data yang sebenarnya "Tidak Order" salah diprediksi sebagai "Order."

Berdasarkan confusion matrix, terdapat 6117 prediksi benar untuk "Tidak Order" dan 34423 untuk "Order." Namun, model juga menghasilkan 1767 data "Tidak Order" yang salah diklasifikasikan sebagai "Order," serta 42 data "Order" yang salah diprediksi sebagai "Tidak Order."

Dengan demikian, model ini sangat baik untuk mendeteksi kategori "Order," namun perlu sedikit perbaikan pada kategori "Tidak Order," yang bisa dilakukan dengan balancing data atau tuning parameter lebih lanjut.

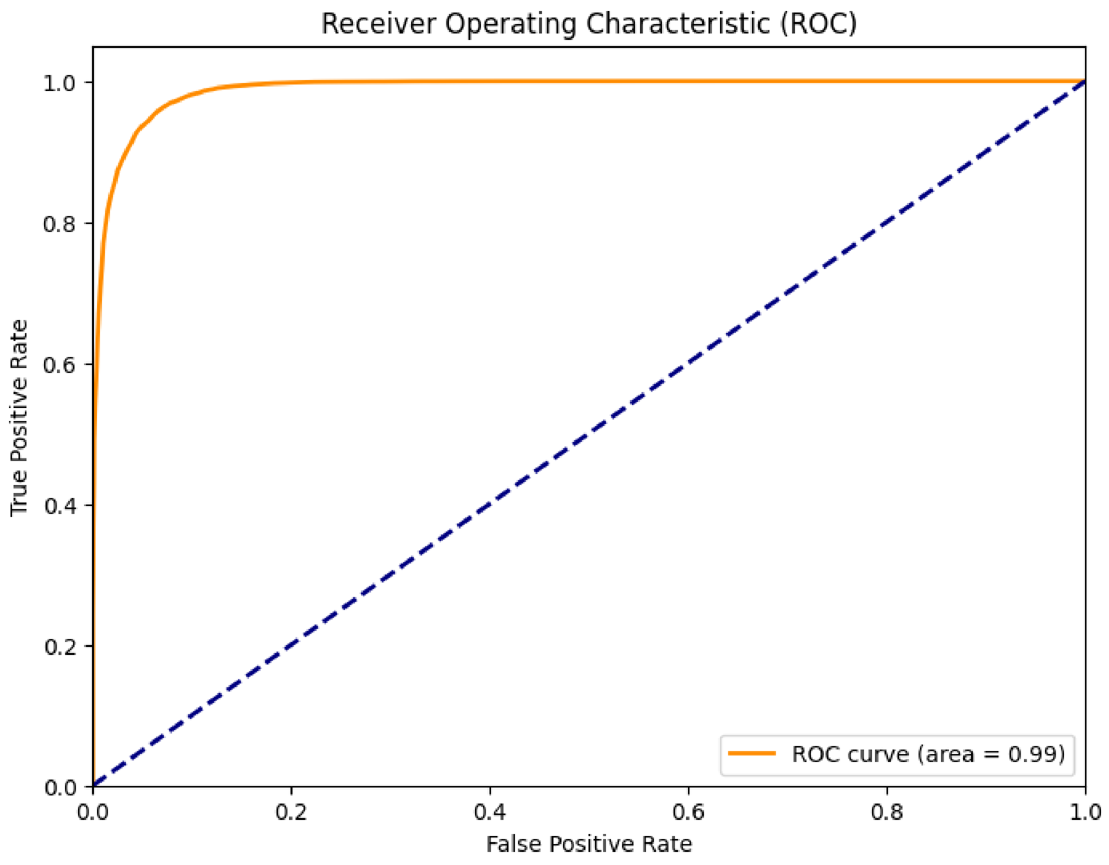
0.3 Visualisasi Confusion Matrix

[34]:
```python
cm = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted No', 'Predicted Yes'],
            yticklabels=['Actual No', 'Actual Yes'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



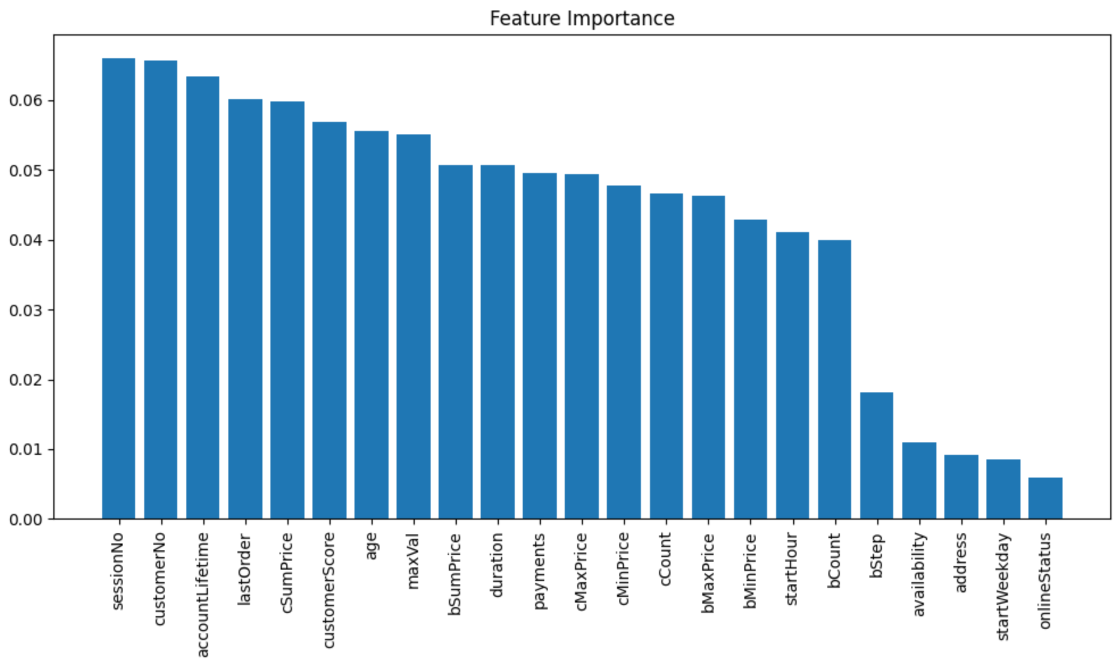
0.4 Visualisasi ROC Curve

[36]:
```python
from sklearn.metrics import roc_curve, auc

# Assuming you have probabilities for the positive class (e.g., from␣
↪predict_proba)
y_probs = rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

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)')
plt.legend(loc="lower right")
plt.show()
```



0.5 Visualisasi Feature Importance

[37]:
```
importances = rf_model.feature_importances_
feature_names = X_train.columns

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importance")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), feature_names[indices], rotation=90)
plt.tight_layout()
plt.show()
```


Feature Importance
0.00
0.01
0.02
0.03
0.04
0.05
0.06
sessionNo
customerNo
accountLifetime
lastOrder
cSumPrice
customerScore
age
maxVal
bSumPrice
duration
payments
cMaxPrice
cMinPrice
cCount
bMaxPrice
bMinPrice
startHour
bCount
bStep
availability
address
startWeekday
onlineStatus