tugas-4

November 22, 2024

0.1 # Tugas 3 Data Mining - EDA + Klasifikasi

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

0.2 Pendahuluan

Analisis Data Eksplorasi (EDA) dan klasifikasi memainkan peran penting dalam memahami pola historis dan membuat prediksi akurat. Pada kompetisi DMC tahun ini, task pertama berfokus pada analisis data sesi historis toko untuk memprediksi kemungkinan terjadinya pesanan (order). Data historis mencakup sekitar 50.000 sesi dengan atribut target "order" yang memiliki dua nilai: "y" untuk pesanan yang dilakukan dan "n" untuk tidak ada pesanan.

Melalui EDA, analisis awal terhadap data dilakukan untuk mengidentifikasi distribusi, pola, dan hubungan antar fitur yang berkontribusi pada kemungkinan pesanan. Informasi ini membantu membangun model prediktif berbasis machine learning untuk menentukan probabilitas [0,1] bagi setiap sesi baru. Evaluasi model dilakukan berdasarkan tingkat kesalahan terhadap hasil aktual dari sekitar 5.000 sesi yang disediakan. Dengan pendekatan ini, diharapkan solusi yang akurat dapat dihasilkan untuk mengoptimalkan prediksi pesanan di masa depan.

0.3 Import Library

Import library yang akan digunakan

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

##Load Dataset

[ ]:
```
# Mount Google Drive to Colab
drive.mount('/content/drive')

# Read the transact_train.txt file from Google Drive into a DataFrame
df = pd.read_csv("/content/drive/MyDrive/Data Mining/Tugas 3/transact_train.
 ↪txt", delimiter="|")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
# Display the first 5 rows of the DataFrame to preview the data
df.head()
```

[ ]:
```
   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]
```

[ ]:
```
# Show a summary of the DataFrame, including column data types, non-null counts, and memory usage
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
```

[ ]:
```python
# Generate summary statistics (mean, median, standard deviation, etc.) for␣
↪numerical columns
df.describe()
```

[ ]:
```
            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
```

[ ]:
```python
# Identify columns containing "?"
columns_with_question_mark = [col for col in df.columns if df[col].astype(str).
↪str.contains('\?').any()]

print("Columns containing '?':", columns_with_question_mark)
```

```
Columns containing '?': ['cMinPrice', 'cMaxPrice', 'cSumPrice', 'bMinPrice',
```

```
'bMaxPrice', 'bSumPrice', 'bStep', 'onlineStatus', 'availability', 'customerNo',
'maxVal', 'customerScore', 'accountLifetime', 'payments', 'age', 'address',
'lastOrder']
```

[ ]:
```
# Convert numeric columns to float, handling '?' values
numeric_cols = ['sessionNo', 'startHour', 'startWeekday', 'duration', 'cCount',␣
 ↪'bCount', 'maxVal', 'customerScore', 'accountLifetime', 'payments', 'age',␣
 ↪'lastOrder']
for col in numeric_cols:
    # Replace '?' with NaN before converting to float
    df[col] = df[col].replace('?', np.nan).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
```

[ ]:
```
# Define a list of columns that are numeric (i.e., continuous or quantitative␣
↪values)
numeric_cols = ['sessionNo', 'startHour', 'startWeekday', 'duration', 'cCount',␣
↪'bCount', 'maxVal', 'customerScore', 'accountLifetime', 'payments', 'age',␣
↪'lastOrder']

# Define a list of columns that are categorical (i.e., discrete or qualitative␣
↪values)
categorical_cols = ['cMinPrice', 'cMaxPrice', 'cSumPrice', 'bMinPrice',␣
↪'bMaxPrice', 'bSumPrice', 'bStep', 'onlineStatus', 'availability', 'order',␣
↪'customerNo', 'address']
```

[ ]:
```
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])
```

[ ]:
```
# 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)
```

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

```
sessionNo              0
startHour              0
startWeekday           0
duration               0
cCount                 0
cMinPrice           2765
cMaxPrice           2765
cSumPrice           2765
bCount                 0
bMinPrice           5130
bMaxPrice           5130
bSumPrice           5130
bStep             191333
onlineStatus      160379
availability      165255
customerNo        151098
maxVal                 0
customerScore          0
accountLifetime        0
```

```
payments                0
age                     0
address            151098
lastOrder               0
order                   0
dtype: int64
```

[ ]:
```
# 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
```

[ ]:
```
# Display the first 5 rows of the DataFrame to preview the data
df.head()
```

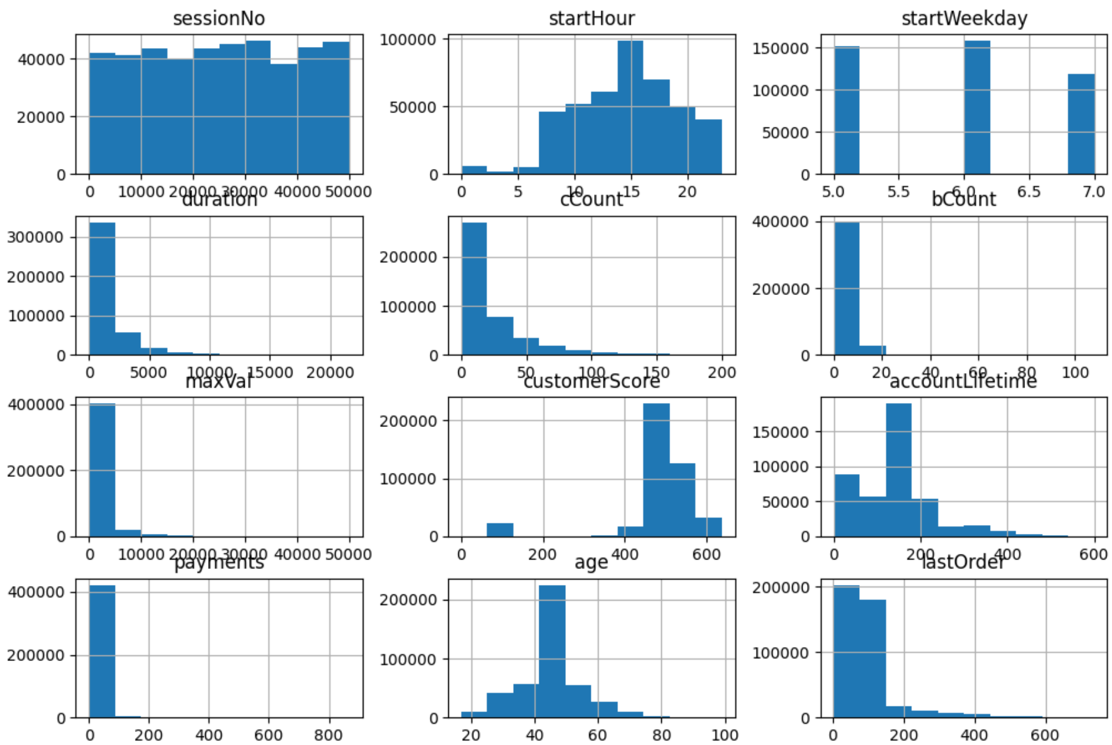
[ ]:
```
   sessionNo  startHour  startWeekday  duration  cCount cMinPrice cMaxPrice  \
0        1.0        6.0           5.0     0.000     1.0     59.99     59.99
1        1.0        6.0           5.0    11.940     1.0     59.99     59.99
2        1.0        6.0           5.0    39.887     1.0     59.99     59.99
3        2.0        6.0           5.0     0.000     0.0         ?         ?
4        2.0        6.0           5.0    15.633     0.0         ?         ?

  cSumPrice  bCount bMinPrice  …          availability customerNo  \
```

```
0     59.99     1.0     59.99  …                     ?          1
1     59.99     1.0     59.99  …  completely orderable          1
2     59.99     1.0     59.99  …  completely orderable          1
3         ?     0.0         ?  …  completely orderable          ?
4         ?     0.0         ?  …  completely orderable          ?

      maxVal customerScore accountLifetime   payments        age  address  \
0   600.00000     70.000000       21.000000   1.000000  43.000000        1
1   600.00000     70.000000       21.000000   1.000000  43.000000        1
2   600.00000     70.000000       21.000000   1.000000  43.000000        1
3  2486.35827    485.298449      135.557403  15.218016  44.919861        ?
4  2486.35827    485.298449      135.557403  15.218016  44.919861        ?

   lastOrder  order
0  49.000000      y
1  49.000000      y
2  49.000000      y
3  79.883975      y
4  79.883975      y

[5 rows x 24 columns]
```

[ ]:
```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  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         maxVal  customerScore  \
count  429013.000000  429013.000000  429013.000000  429013.000000
mean       24.140317       4.135168    2486.358270     485.298449
std        30.398164       4.451778    2433.856317     104.956462
min         0.000000       0.000000       0.000000       0.000000
25%         5.000000       1.000000     900.000000     485.298449
50%        13.000000       3.000000    2486.358270     485.298449
```

```
75%         31.000000       5.000000     2500.000000      535.000000
max        200.000000     108.000000    50000.000000      638.000000

       accountLifetime       payments            age       lastOrder
count    429013.000000  429013.000000  429013.000000  429013.000000
mean        135.557403      15.218016      44.919861       79.883975
std          87.774074      28.083922       9.601616       91.111767
min           0.000000       0.000000      17.000000        3.000000
25%          75.000000       6.000000      41.000000       23.000000
50%         135.557403      15.218016      44.919861       79.883975
75%         156.000000      15.218016      48.000000       79.883975
max         600.000000     868.000000      99.000000      738.000000
```




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

    # 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
```

```python
    unusual_values = value_counts[value_counts < 10].index
    if len(unusual_values) > 0:
        print("Unusual/Infrequent values:")
        print(unusual_values)
    print("---")
```

```
Column: cMinPrice
Most frequent values:
cMinPrice
9.99     55667
3.99     35395
19.99    23352
12.99    19007
14.99    18227
29.99    15337
4.99     15227
6.99     13211
7.99     12986
24.99    11050
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['65.6', '34.96', '119.9', '46.99', '19.96', '569.99', '98.0', '110.0',
       '48.74', '18.5',
       …
       '37.49', '31.95', '888.0', '23.5', '51.99', '159.9', '1190.0', '519.0',
       '30.95', '54.9'],
      dtype='object', name='cMinPrice', length=263)
---
Column: cMaxPrice
Most frequent values:
cMaxPrice
29.99    33194
19.99    27636
49.99    26032
39.99    24772
24.99    17868
59.99    16131
99.99    12498
79.99    11288
59.95    10973
34.99    10499
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['28.9', '28.5', '247.12', '165.99', '42.5', '185.0', '57.85', '859.99',
       '449.0', '675.0',
       …
       '34.96', '18.16', '11.0', '1190.0', '35.96', '23.5', '95.99', '4.75',
```

```
       '15.3', '201.99'],
      dtype='object', name='cMaxPrice', length=245)
---
Column: cSumPrice
Most frequent values:
cSumPrice
49.99    3942
?        2765
6.99     2754
39.98    2461
59.98    1692
89.97    1686
19.98    1602
59.97    1577
79.98    1553
49.98    1368
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['605.65', '749.57', '270.88', '314.79', '526.63', '1651.81', '1674.84',
       '172.88', '919.88', '367.81',
       …
       '836.53', '900.49', '167.24', '198.15', '228.13', '258.12', '2275.91',
       '553.86', '650.61', '5253.28'],
      dtype='object', name='cSumPrice', length=64510)
---
Column: bMinPrice
Most frequent values:
bMinPrice
9.99     56187
3.99     32670
19.99    27620
14.99    21855
12.99    20079
29.99    19388
24.99    13461
39.99    11275
49.99    11187
6.99     10548
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['1039.99', '1.45', '134.99', '82.49', '36.0', '366.0', '67.49', '32.0',
       '57.99', '1.25',
       …
       '51.97', '13.5', '36.76', '37.96', '7.8', '549.95', '43.0', '62.96',
       '9.79', '159.9'],
      dtype='object', name='bMinPrice', length=239)
---
Column: bMaxPrice
```

```
Most frequent values:
bMaxPrice
29.99    40203
19.99    36399
39.99    25841
49.99    25754
24.99    21253
59.99    14910
9.99     14629
14.99    13085
59.95    10684
34.99    10018
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['6.59', '71.95', '909.9', '2799.99', '28.5', '26.95', '2199.99',
       '387.03', '67.49', '6999.99',
       …
       '4.65', '154.9', '598.0', '62.0', '69.74', '349.95', '299.9', '1619.0',
       '1549.99', '7.8'],
      dtype='object', name='bMaxPrice', length=216)
---
Column: bSumPrice
Most frequent values:
bSumPrice
29.99    8733
19.99    8285
49.99    8082
39.99    6129
9.99     5879
24.99    5413
?        5130
14.99    4005
39.98    3886
59.98    3489
Name: count, dtype: int64
Unusual/Infrequent values:
Index(['228.52', '324.38', '2264.0', '212.94', '421.76', '151.97', '1291.59',
       '452.81', '397.86', '161.61',
       …
       '49.36', '102.83', '122.82', '241.82', '160.81', '210.76', '235.75',
       '273.69', '772.96', '211.86'],
      dtype='object', name='bSumPrice', length=15207)
---
Column: bStep
Most frequent values:
bStep
?    191333
1     90058
```

```
2     60682
4     41142
3     30062
5     15736
Name: count, dtype: int64
---
Column: onlineStatus
Most frequent values:
onlineStatus
y    265625
?    160379
n      3009
Name: count, dtype: int64
---
Column: availability
Most frequent values:
availability
completely orderable           253692
?                              165255
mainly orderable                 5756
completely not orderable         1491
mixed                            1284
completely not determinable      1017
mainly not orderable              320
mainly not determinable           198
Name: count, dtype: int64
---
Column: order
Most frequent values:
order
y    290030
n    138983
Name: count, dtype: int64
---
Column: customerNo
Most frequent values:
customerNo
?        151098
47         1248
5464        345
4953        196
16740       155
7394        152
5488        149
4034        136
15769       123
4118        118
Name: count, dtype: int64
```

```
Unusual/Infrequent values:
Index(['15891', '12761', '22717', '14906', '14882', '12456', '22729', '12736',
       '14524', '20945',
       …
       '11690', '11665', '11494', '11626', '11570', '11544', '11540', '11537',
       '11512', '10461'],
      dtype='object', name='customerNo', length=13625)
---
Column: address
Most frequent values:
address
2    203570
?    151098
1     74058
3       287
Name: count, dtype: int64
---
```

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

```
        sessionNo  startHour  startWeekday  duration  cCount cMinPrice  \
8             3.0        6.0           5.0   181.477     9.0     29.99
9             3.0        6.0           5.0   297.018    11.0      9.99
10            3.0        6.0           5.0   310.967    11.0      9.99
11            3.0        6.0           5.0   324.278    11.0      9.99
12            3.0        6.0           5.0   341.613    11.0      9.99
…             …          …             …         …       …        …
429006    49998.0       18.0           7.0  2961.909     6.0     59.99
429007    49998.0       18.0           7.0  4700.383    50.0      9.99
429008    49998.0       18.0           7.0  5988.882    77.0      9.99
429009    49999.0       18.0           7.0   675.114     6.0      59.0
429010    49999.0       18.0           7.0   715.341     7.0      59.0

       cMaxPrice cSumPrice  bCount bMinPrice  …          availability  \
8          29.99     89.97     1.0     29.99  …                     ?
9          29.99    109.95     2.0      9.99  …                     ?
10         29.99    109.95     2.0      9.99  …  completely orderable
11         29.99    109.95     2.0      9.99  …  completely orderable
12         29.99    109.95     2.0      9.99  …  completely orderable
…            …         …         …    …                    …
429006     99.99    419.94     1.0     59.99  …                     ?
429007    119.99    2974.6     2.0     59.99  …                     ?
429008    149.99   5253.28     3.0     49.95  …                     ?
429009    199.99    509.96     1.0     89.99  …                     ?
429010    649.99   1159.95     1.0     89.99  …  completely orderable

       customerNo      maxVal customerScore accountLifetime   payments  \
```

```
8               3  1800.00000   475.000000     302.000000  12.000000
9               3  1800.00000   475.000000     302.000000  12.000000
10              3  1800.00000   475.000000     302.000000  12.000000
11              3  1800.00000   475.000000     302.000000  12.000000
12              3  1800.00000   475.000000     302.000000  12.000000
…             …        …            …            …         …
429006          ?  2486.35827   485.298449     135.557403  15.218016
429007          ?  2486.35827   485.298449     135.557403  15.218016
429008          ?  2486.35827   485.298449     135.557403  15.218016
429009      25038  2486.35827   485.298449     135.557403   0.000000
429010      25038  2486.35827   485.298449     135.557403   0.000000

              age  address  lastOrder  order
8       45.000000        1  11.000000      y
9       45.000000        1  11.000000      y
10      45.000000        1  11.000000      y
11      45.000000        1  11.000000      y
12      45.000000        1  11.000000      y
…             …      …         …    …
429006  44.919861        ?  79.883975      n
429007  44.919861        ?  79.883975      n
429008  44.919861        ?  79.883975      n
429009  24.000000        1   4.000000      n
429010  24.000000        1   4.000000      n

[373604 rows x 24 columns]
```

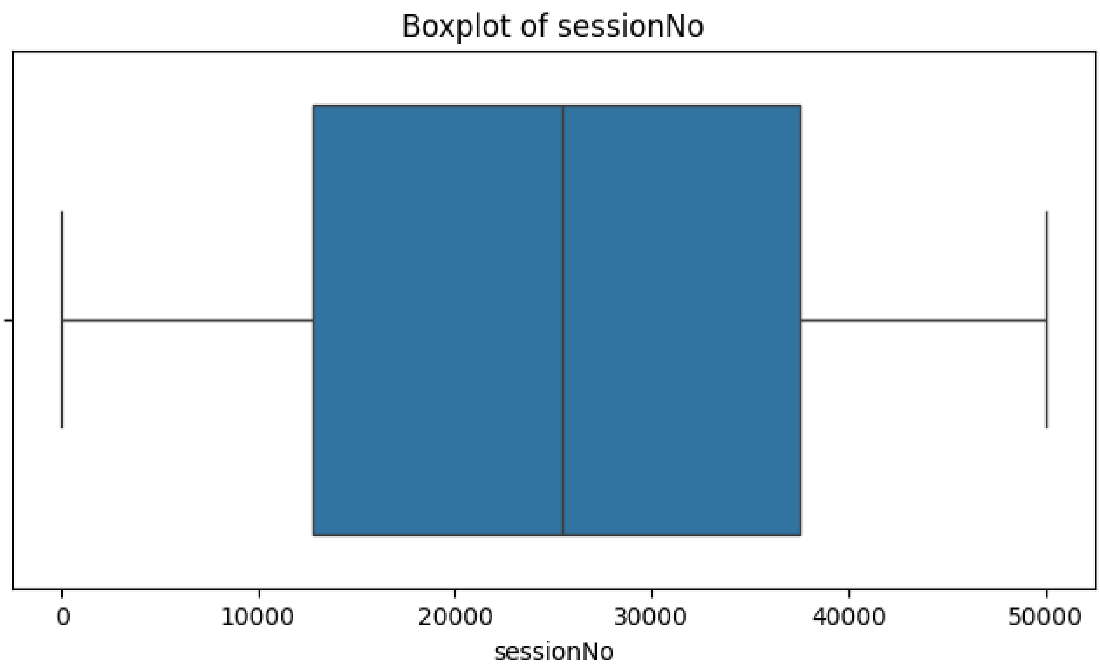
[ ]:

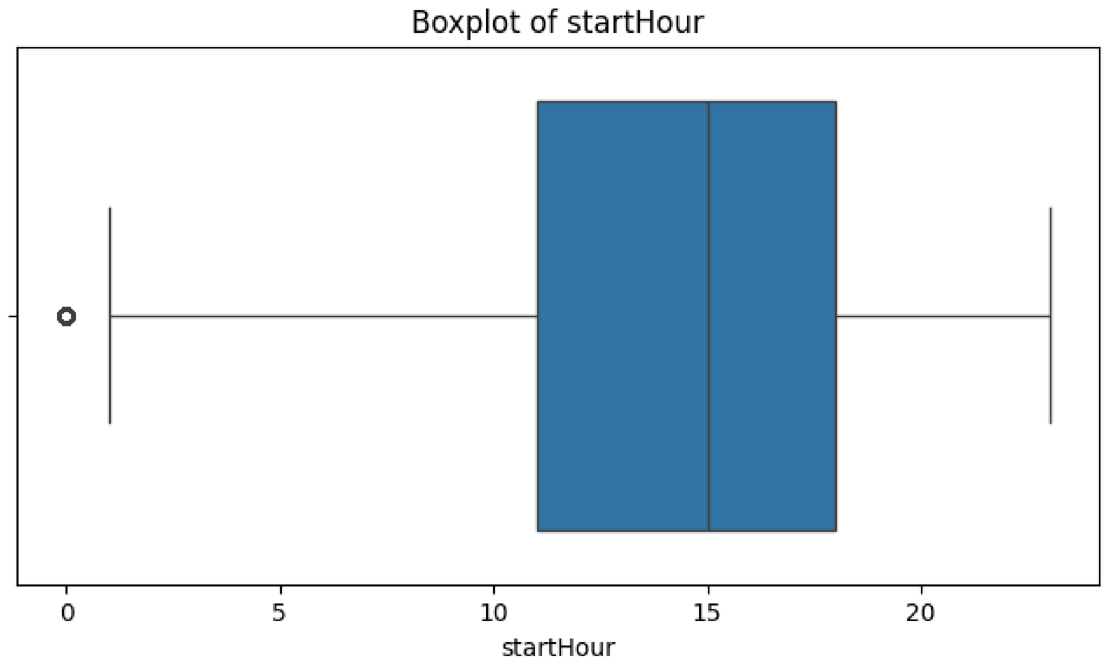
```
# Identify outliers using z-score
from scipy.stats import zscore

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)
```

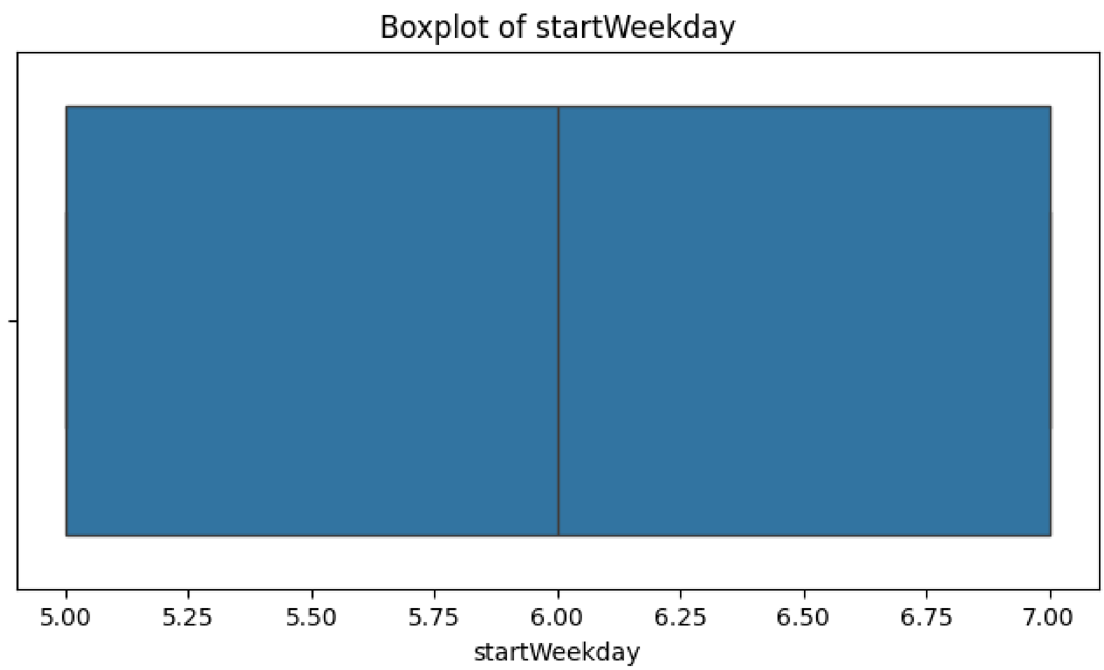
```
        sessionNo  startHour  startWeekday  duration  cCount cMinPrice  \
0             1.0        6.0           5.0     0.000     1.0     59.99
1             1.0        6.0           5.0    11.940     1.0     59.99
2             1.0        6.0           5.0    39.887     1.0     59.99
70           12.0        6.0           5.0     9.220     2.0      9.99
71           12.0        6.0           5.0    91.283     4.0      5.99
…             …         …            …         …        …       …
428999    49996.0       18.0           7.0  7015.682   197.0      6.99
```

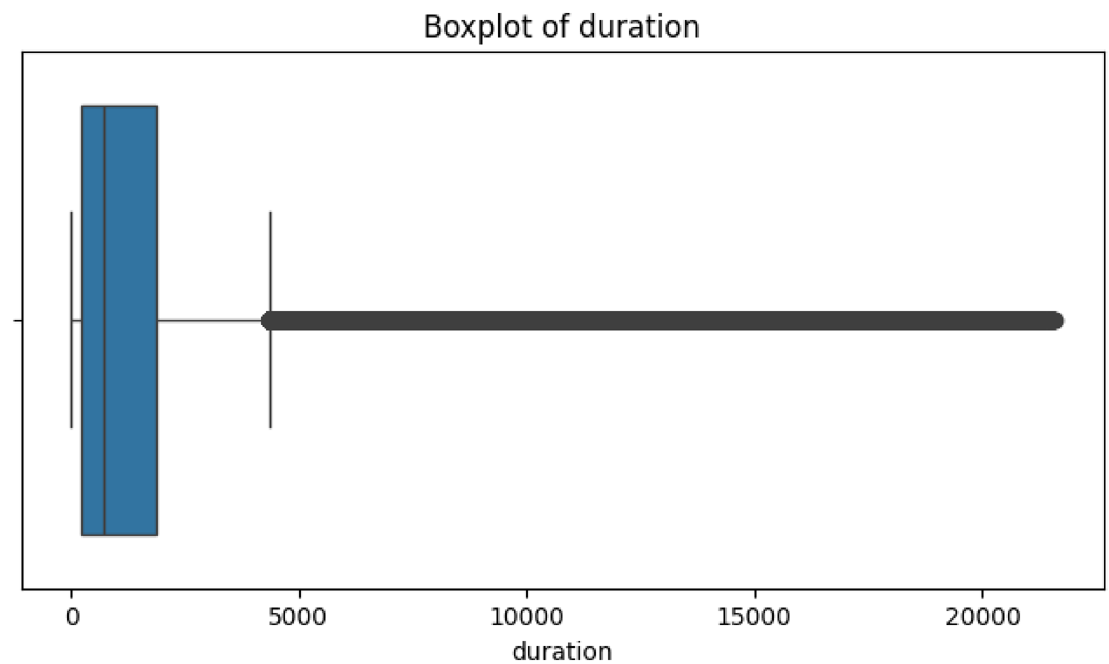
```
429000    49996.0       18.0           7.0  7074.729   197.0      6.99
429001    49996.0       18.0           7.0  7089.360   197.0      6.99
429002    49996.0       18.0           7.0  7170.905   197.0      6.99
429003    49996.0       18.0           7.0  7271.812   197.0      6.99

       cMaxPrice cSumPrice  bCount bMinPrice  …         availability  \
0          59.99     59.99     1.0     59.99  …                    ?
1          59.99     59.99     1.0     59.99  …  completely orderable
2          59.99     59.99     1.0     59.99  …  completely orderable
70          9.99     19.98     1.0      9.99  …                    ?
71          9.99     31.96     2.0      5.99  …                    ?
…            …         …      …         …   …                    …
428999     59.99   4315.03    13.0      9.99  …  completely orderable
429000     59.99   4315.03    13.0      9.99  …  completely orderable
429001     59.99   4315.03    13.0      9.99  …  completely orderable
429002     59.99   4315.03    13.0      9.99  …  completely orderable
429003     59.99   4315.03    13.0      9.99  …  completely orderable

       customerNo      maxVal customerScore accountLifetime   payments  \
0               1   600.00000     70.000000       21.000000   1.000000
1               1   600.00000     70.000000       21.000000   1.000000
2               1   600.00000     70.000000       21.000000   1.000000
70              8  2000.00000    546.000000      364.000000  11.000000
71              8  2000.00000    546.000000      364.000000  11.000000
…          …            …             …               …          …
428999          ?  2486.35827    485.298449      135.557403  15.218016
429000          ?  2486.35827    485.298449      135.557403  15.218016
429001          ?  2486.35827    485.298449      135.557403  15.218016
429002          ?  2486.35827    485.298449      135.557403  15.218016
429003          ?  2486.35827    485.298449      135.557403  15.218016

              age  address  lastOrder  order
0       43.000000        1  49.000000      y
1       43.000000        1  49.000000      y
2       43.000000        1  49.000000      y
70      86.000000        2  37.000000      y
71      86.000000        2  37.000000      y
…          …        …         …    …
428999  44.919861        ?  79.883975      y
429000  44.919861        ?  79.883975      y
429001  44.919861        ?  79.883975      y
429002  44.919861        ?  79.883975      y
429003  44.919861        ?  79.883975      y

[69869 rows x 24 columns]
```

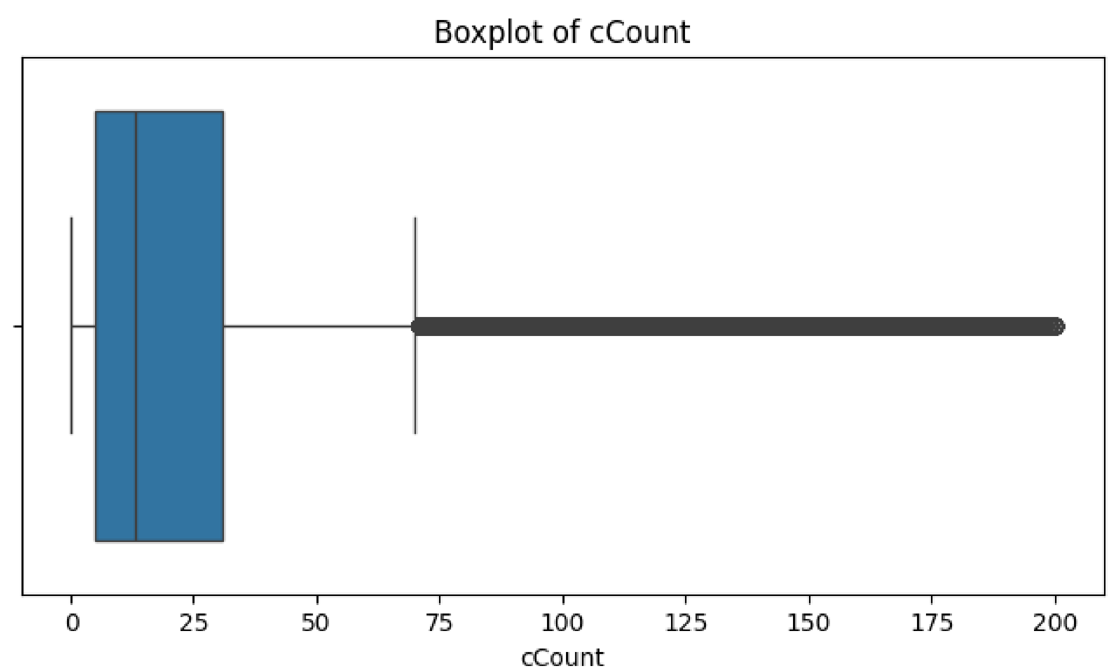
[ ]:
```python
# Boxplots for each numeric column to spot outliers
for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()
```

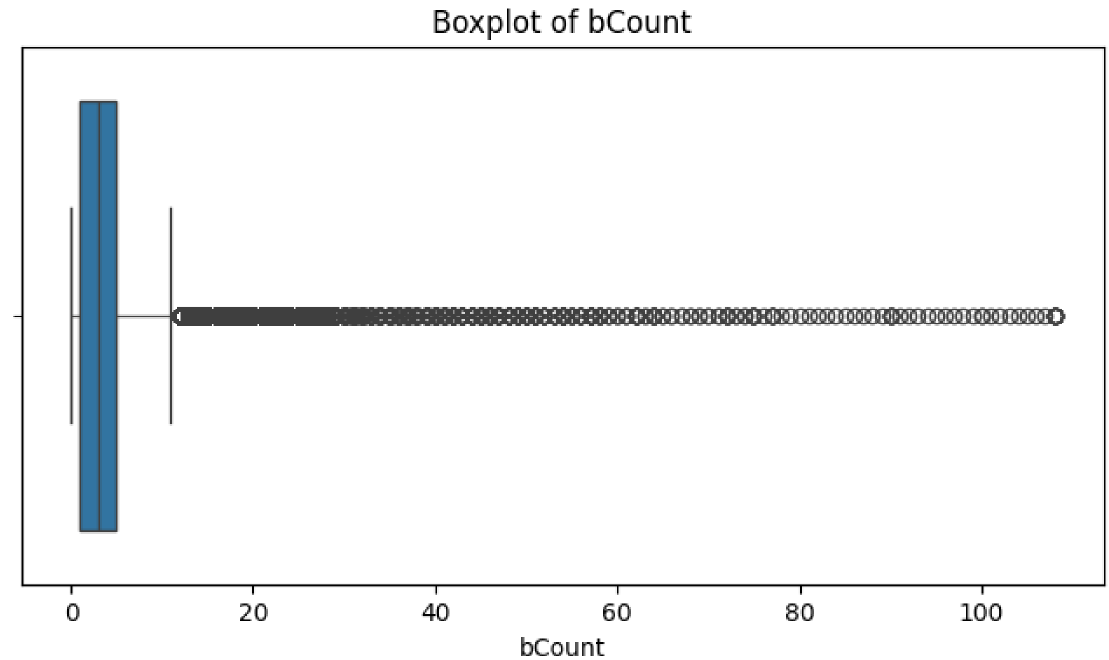


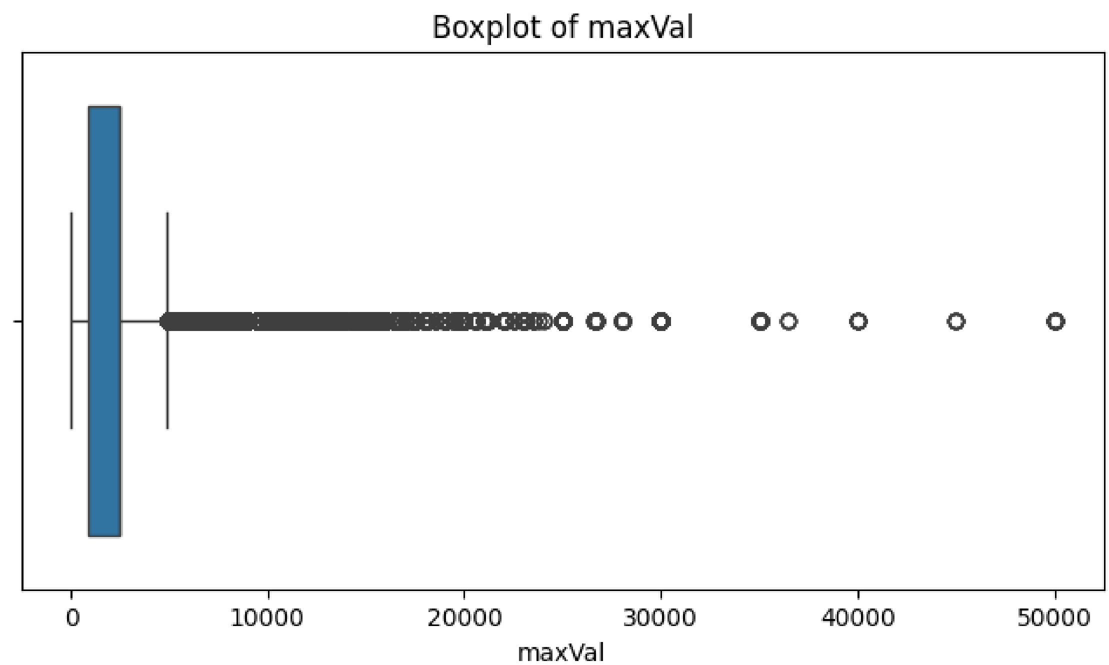

Boxplot of startHour
0
5
10
15
20
startHour

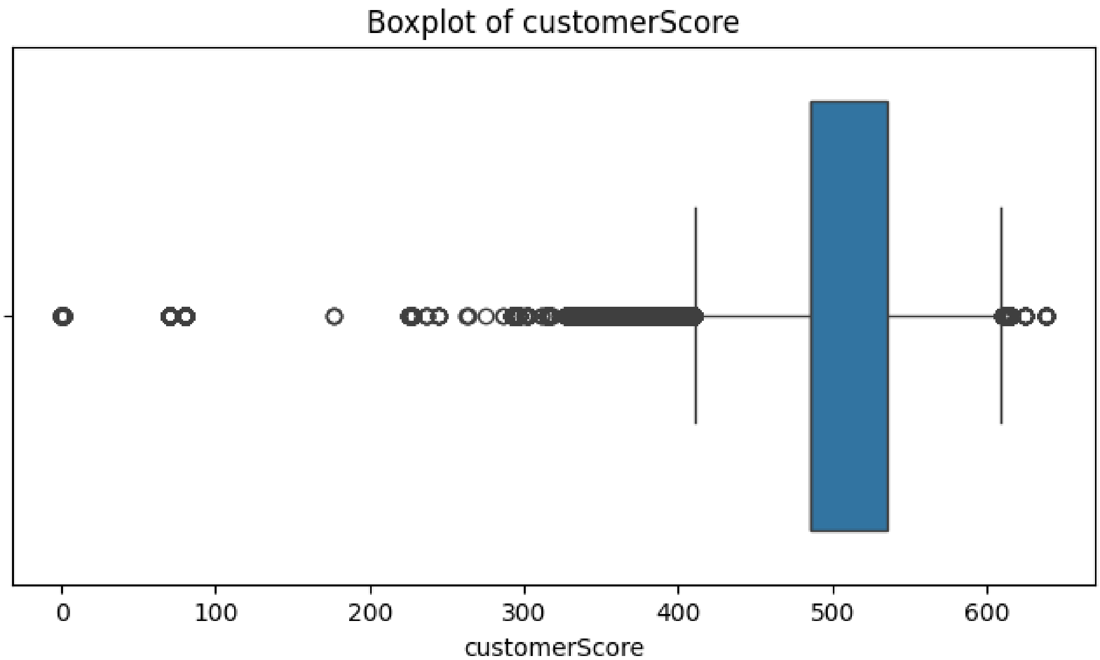

Boxplot of startWeekday
5.00
5.25
5.50
5.75
6.00
6.25
6.50
6.75
7.00
startWeekday

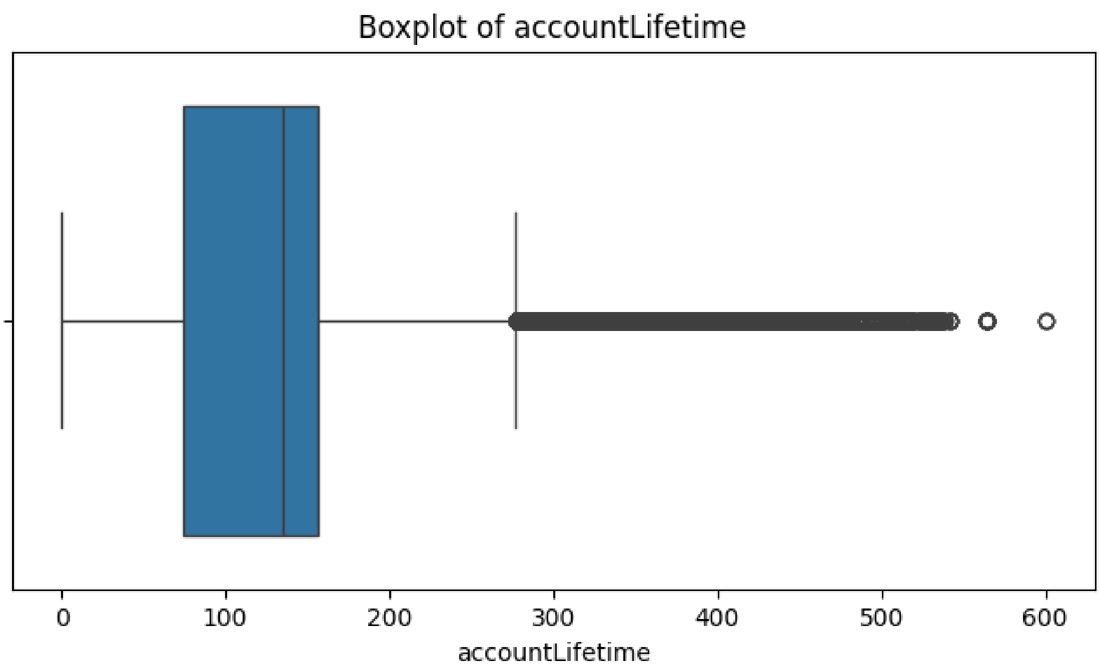

Boxplot of duration
0
5000
10000
15000
20000
duration

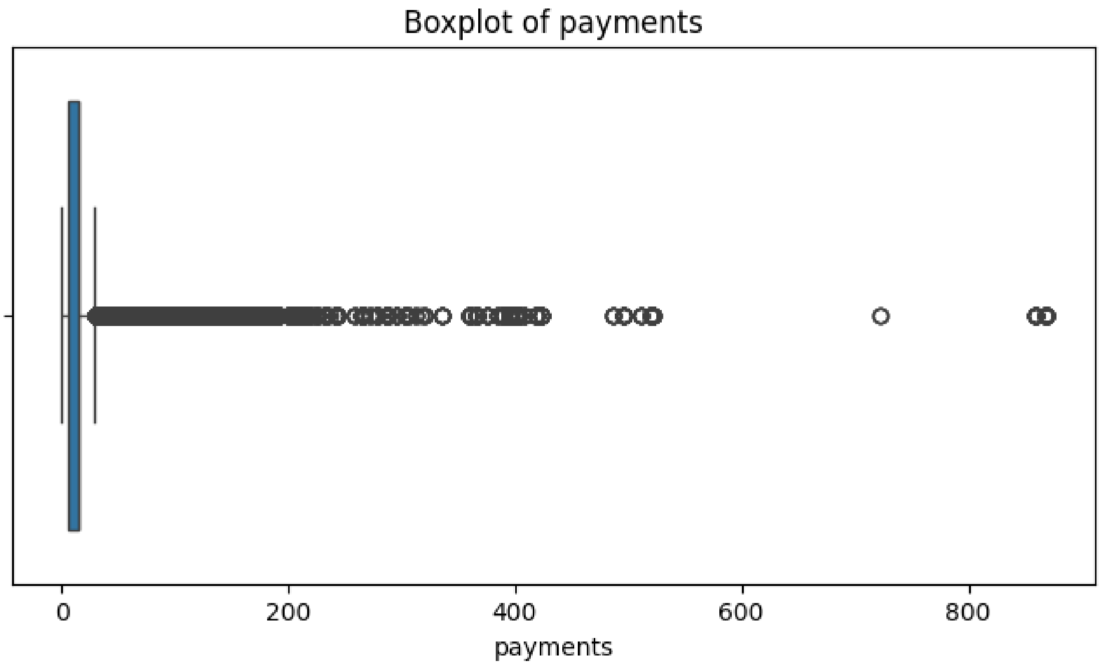

Boxplot of cCount
0
25
50
75
100
125
150
175
200
cCount

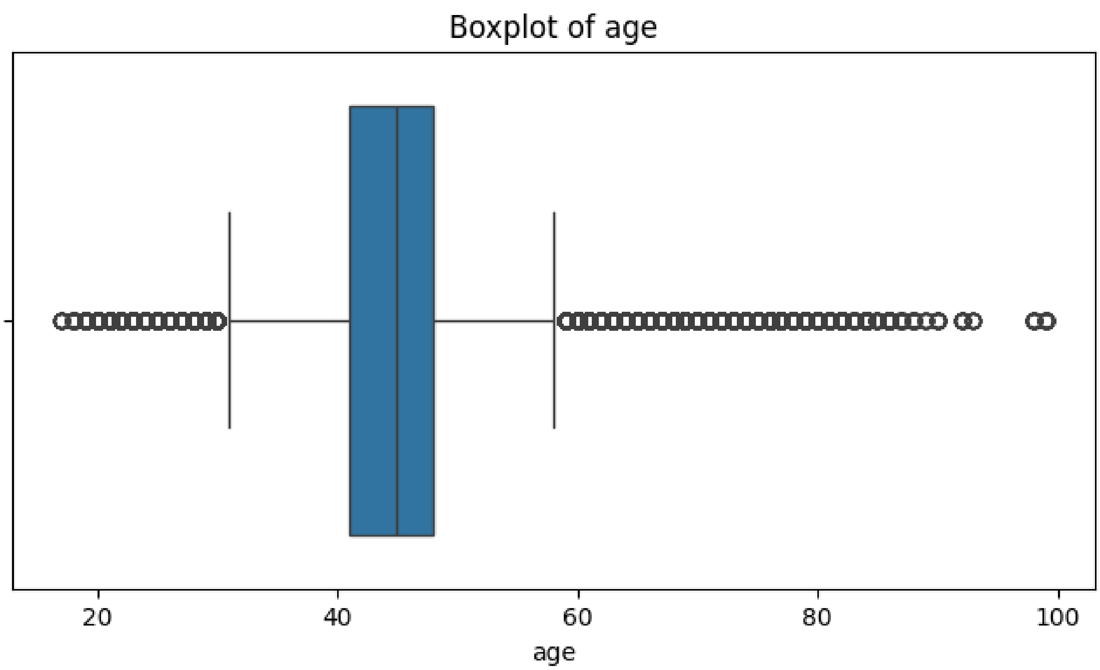

Boxplot of bCount
0
20
40
60
80
100
bCount

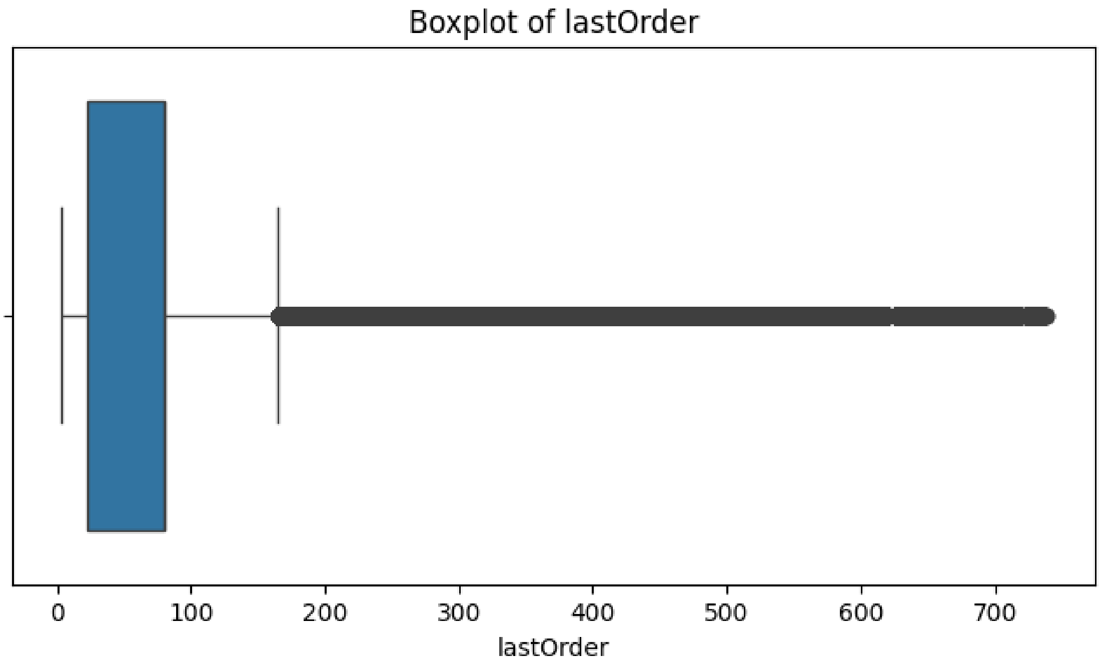

Boxplot of maxVal
0
10000
20000
30000
40000
50000
maxVal


Boxplot of customerScore
0
100
200
300
400
500
600
customerScore


Boxplot of accountLifetime
0
100
200
300
400
500
600
accountLifetime


Boxplot of payments
0
200
400
600
800
payments


Boxplot of age
20
40
60
80
100
age



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

[ ]:
```
# 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  \
9              3.0        6.0           5.0   297.018    11.0      9.99
10             3.0        6.0           5.0   310.967    11.0      9.99
11             3.0        6.0           5.0   324.278    11.0      9.99
12             3.0        6.0           5.0   341.613    11.0      9.99
24             6.0        6.0           5.0     0.000     2.0     99.99
…              …          …            …          …        …       …
429002     49996.0       18.0           7.0  7170.905   197.0      6.99
429003     49996.0       18.0           7.0  7271.812   197.0      6.99
429007     49998.0       18.0           7.0  4700.383    50.0      9.99
429008     49998.0       18.0           7.0  5988.882    77.0      9.99
429009     49999.0       18.0           7.0   675.114     6.0      59.0

       cMaxPrice cSumPrice  bCount bMinPrice  …          availability  \
9          29.99    109.95     2.0      9.99  …                     ?
10         29.99    109.95     2.0      9.99  …  completely orderable
11         29.99    109.95     2.0      9.99  …  completely orderable
```

```
12          29.99    109.95     2.0       9.99  …   completely orderable
24         129.99    229.98     2.0      99.99  …                      ?
…             …        …       …        …    …               …
429002      59.99   4315.03    13.0       9.99  …   completely orderable
429003      59.99   4315.03    13.0       9.99  …   completely orderable
429007     119.99    2974.6     2.0      59.99  …                      ?
429008     149.99   5253.28     3.0      49.95  …                      ?
429009     199.99    509.96     1.0      89.99  …                      ?

       customerNo      maxVal customerScore accountLifetime   payments  \
9               3  1800.00000    475.000000      302.000000  12.000000
10              3  1800.00000    475.000000      302.000000  12.000000
11              3  1800.00000    475.000000      302.000000  12.000000
12              3  1800.00000    475.000000      302.000000  12.000000
24              ?  2486.35827    485.298449      135.557403  15.218016
…            …           …             …              …         …
429002          ?  2486.35827    485.298449      135.557403  15.218016
429003          ?  2486.35827    485.298449      135.557403  15.218016
429007          ?  2486.35827    485.298449      135.557403  15.218016
429008          ?  2486.35827    485.298449      135.557403  15.218016
429009      25038  2486.35827    485.298449      135.557403   0.000000

              age  address  lastOrder  order
9       45.000000        1  11.000000      y
10      45.000000        1  11.000000      y
11      45.000000        1  11.000000      y
12      45.000000        1  11.000000      y
24      44.919861        ?  79.883975      n
…            …       …        …    …
429002  44.919861        ?  79.883975      y
429003  44.919861        ?  79.883975      y
429007  44.919861        ?  79.883975      n
429008  44.919861        ?  79.883975      n
429009  24.000000        1   4.000000      n

[150046 rows x 24 columns]
```

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

[ ]:
```python
# Display summary statistics for all columns in the DataFrame, including both␣
 ↪numeric and categorical columns
print(df.describe(include='all'))
```

```
            sessionNo      startHour   startWeekday       duration  \
count   429013.000000  429013.000000  429013.000000  429013.000000
unique            NaN            NaN            NaN            NaN
```

```
top            NaN           NaN           NaN           NaN
freq           NaN           NaN           NaN           NaN
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 cMinPrice cMaxPrice cSumPrice         bCount bMinPrice  \
count  429013.000000    429013    429013    429013  429013.000000    429013
unique           NaN       839       825     72990            NaN       748
top              NaN     19.99      9.99     49.99            NaN      9.99
freq             NaN     36762     57229      3942            NaN     56187
mean       24.140317       NaN       NaN       NaN       4.135168       NaN
std        30.398164       NaN       NaN       NaN       4.451778       NaN
min         0.000000       NaN       NaN       NaN       0.000000       NaN
25%         5.000000       NaN       NaN       NaN       1.000000       NaN
50%        13.000000       NaN       NaN       NaN       3.000000       NaN
75%        31.000000       NaN       NaN       NaN       5.000000       NaN
max       200.000000       NaN       NaN       NaN     108.000000       NaN

        …          availability customerNo         maxVal  customerScore  \
count   …                429013     429013  429013.000000  429013.000000
unique  …                     8      25038            NaN            NaN
top     …   completely orderable          ?            NaN            NaN
freq    …                253692     151098            NaN            NaN
mean    …                   NaN        NaN    2486.358270     485.298449
std     …                   NaN        NaN    2433.856317     104.956462
min     …                   NaN        NaN       0.000000       0.000000
25%     …                   NaN        NaN     900.000000     485.298449
50%     …                   NaN        NaN    2486.358270     485.298449
75%     …                   NaN        NaN    2500.000000     535.000000
max     …                   NaN        NaN   50000.000000     638.000000

       accountLifetime       payments            age  address      lastOrder  \
count    429013.000000  429013.000000  429013.000000   429013  429013.000000
unique             NaN            NaN            NaN        4            NaN
top                NaN            NaN            NaN        2            NaN
freq               NaN            NaN            NaN   203570            NaN
mean        135.557403      15.218016      44.919861      NaN      79.883975
std          87.774074      28.083922       9.601616      NaN      91.111767
min           0.000000       0.000000      17.000000      NaN       3.000000
25%          75.000000       6.000000      41.000000      NaN      23.000000
50%         135.557403      15.218016      44.919861      NaN      79.883975
75%         156.000000      15.218016      48.000000      NaN      79.883975
max         600.000000     868.000000      99.000000      NaN     738.000000
```

```
        order
count  429013
unique      2
top         y
freq   290030
mean      NaN
std       NaN
min       NaN
25%       NaN
50%       NaN
75%       NaN
max       NaN

[11 rows x 24 columns]
```

##Model Klasifikasi

```
[ ]: import pandas as pd #Make sure pandas is imported
from sklearn.model_selection import train_test_split #Import train_test_split␣
 ↪to split the data
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Assuming 'df' is your DataFrame containing all the data

X = df.drop(columns=['order'])
y = df['order']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# 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)

# Calculate accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy (Random Forest):", accuracy_rf)

# show classification reports and confusion matrix
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): 0.9122524853443352
Classification Report (Random Forest):
               precision    recall  f1-score   support

           n       0.89      0.83      0.86     27700
           y       0.92      0.95      0.94     58103

    accuracy                           0.91     85803
   macro avg       0.91      0.89      0.90     85803
weighted avg       0.91      0.91      0.91     85803

Confusion Matrix (Random Forest):
 [[22983  4717]
 [ 2812 55291]]
```

0.4 Analisis Model Klasifikasi

1. Akurasi:

- Akurasi model adalah 0.9122, yang berarti model memprediksi kelas dengan benar sekitar 91.22% dari waktu.
- Ini merupakan nilai akurasi yang cukup tinggi, menunjukkan bahwa model secara umum berkinerja baik.

2. Classification Report:

- Precision:
  - Precision untuk kelas "n" adalah 0.89, yang berarti dari semua data yang diprediksi sebagai "n", sekitar 89% benar-benar "n".
  - Precision untuk kelas "y" adalah 0.92, yang berarti dari semua data yang diprediksi sebagai "y", sekitar 92% benar-benar "y".

- Recall:
  - Recall untuk kelas "n" adalah 0.83, yang berarti model berhasil mengidentifikasi sekitar 83% dari semua data "n" yang sebenarnya.
  - Recall untuk kelas "y" adalah 0.95, yang berarti model berhasil mengidentifikasi sekitar 95% dari semua data "y" yang sebenarnya.
- F1-score:
  - F1-score adalah rata-rata harmonik antara precision dan recall.
  - F1-score untuk kelas "n" adalah 0.86, dan untuk kelas "y" adalah 0.94.
  - Nilai F1-score yang tinggi menunjukkan keseimbangan yang baik antara precision dan recall.
- Support:
  - Support menunjukkan jumlah data aktual untuk setiap kelas.
  - Ada 27700 data untuk kelas "n" dan 58103 data untuk kelas "y".

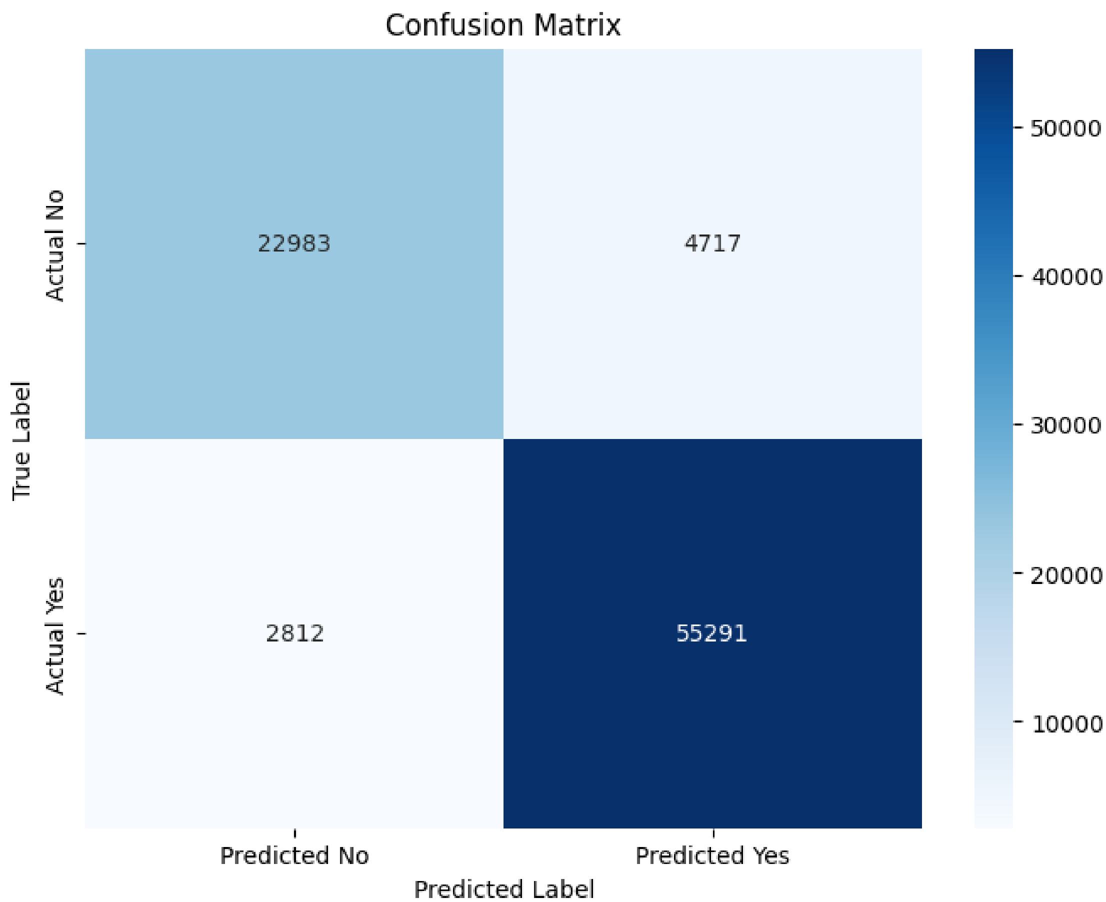
3. Confusion Matrix:

- Confusion matrix memberikan gambaran lebih detail tentang kinerja model:
  - True Positive (TP): 55291 (model memprediksi "y" dengan benar)
  - True Negative (TN): 22983 (model memprediksi "n" dengan benar)
  - False Positive (FP): 4717 (model memprediksi "y" secara salah)
  - False Negative (FN): 2812 (model memprediksi "n" secara salah)

0.5 Visualisasi Confusion Matrix

- Predicted No, Actual No (True Negative - TN): 22,983. Model memprediksi "No" dengan benar untuk 22,983 sampel.
- Predicted Yes, Actual No (False Positive - FP): 4,717. Model salah memprediksi "Yes" padahal seharusnya "No".
- Predicted No, Actual Yes (False Negative - FN): 2,812. Model salah memprediksi "No" padahal seharusnya "Yes".
- Predicted Yes, Actual Yes (True Positive - TP): 55,291. Model memprediksi "Yes" dengan benar untuk 55,291 sampel.

[ ]:
```
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()
```

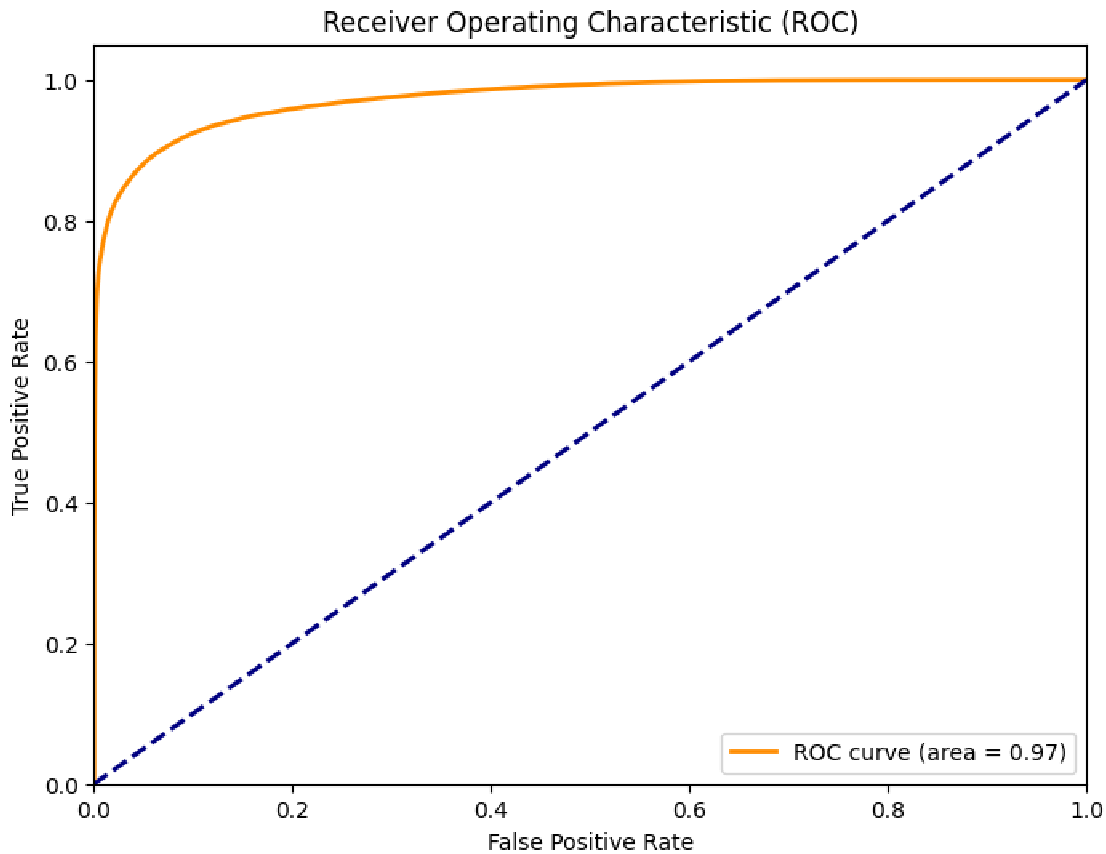


0.6 Visualisasi ROC Curve

- Dengan AUC sebesar 0.97, model memiliki kemampuan diskriminasi yang sangat baik untuk membedakan antara kelas positif dan negatif.
- Pada berbagai threshold, model mampu mempertahankan keseimbangan yang baik antara TPR (benar memprediksi positif) dan FPR (meminimalkan prediksi positif palsu).

[ ]:
```
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]

# Convert 'y_test' to numerical format where 'y' is 1 and 'n' is 0
y_test_numeric = [1 if value == 'y' else 0 for value in y_test]

# Use the numerical y_test in roc_curve
fpr, tpr, thresholds = roc_curve(y_test_numeric, 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()
```

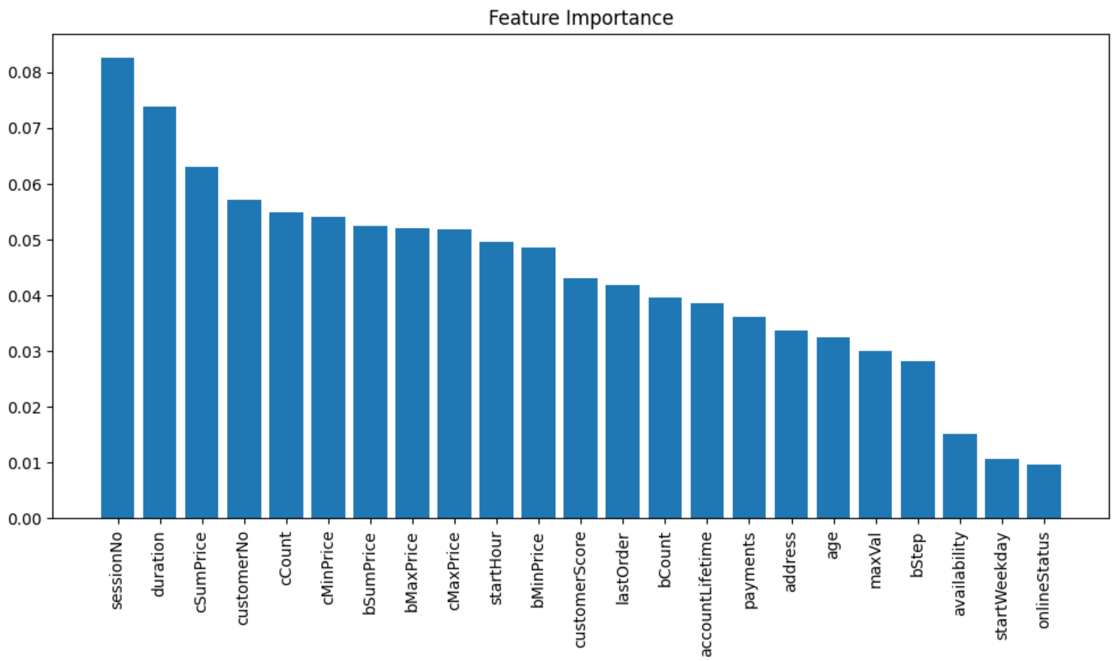


0.7 Visualisasi Feature Importance

- sessionNo, duration, dan cSumPrice memiliki nilai feature importance tertinggi. Hal ini menunjukkan bahwa fitur-fitur tersebut memiliki pengaruh paling besar dalam menentukan output model.

- availability, startWeekday, dan onlineStatus memiliki kontribusi yang sangat kecil terhadap prediksi model. Fitur ini mungkin tidak terlalu relevan atau penting bagi model.

[ ]:
```python
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()
```



0.8 Kesimpulan

1. **Akurasi Model**: Model ini memiliki akurasi **91.22%**, yang menunjukkan bahwa model memprediksi kelas dengan benar sekitar 91.22% dari waktu.
2. **Classification Report**:
    - **Precision**: Untuk kelas "n" (Tidak Order), precision mencapai **0.89**, dan untuk kelas "y" (Order), precision mencapai **0.92**.
    - **Recall**: Untuk kelas "n", recall adalah **0.83**, sedangkan untuk kelas "y", recall mencapai **0.95**.

- **F1-score**: Nilai F1-score untuk kelas “n” adalah **0.86**, sementara untuk kelas “y” adalah **0.94**.
- **Support**: Jumlah data untuk kelas “n” adalah **27,700**, sementara kelas “y” memiliki **58,103** data.

3. **Confusion Matrix**:
   - Model menghasilkan **True Positive (TP)** sebanyak **55,291** untuk kelas “y”.
   - Model menghasilkan **True Negative (TN)** sebanyak **22,983** untuk kelas “n”.
   - Terdapat **4,717 False Positive (FP)**, di mana model salah memprediksi kelas “y” padahal seharusnya “n”.
   - Terdapat **2,812 False Negative (FN)**, di mana model salah memprediksi kelas “n” padahal seharusnya “y”.

4. **Visualisasi ROC Curve**: AUC (Area Under the Curve) model adalah **0.97**, menunjukkan bahwa model memiliki kemampuan diskriminasi yang sangat baik dalam membedakan antara kelas positif dan negatif.

5. **Visualisasi Feature Importance**: Fitur yang memiliki kontribusi tertinggi dalam model adalah **sessionNo**, **duration**, dan **cSumPrice**, sementara fitur dengan kontribusi lebih kecil adalah **availability**, **startWeekday**, dan **onlineStatus**.

Secara keseluruhan, model ini menunjukkan performa yang sangat baik, dengan akurasi tinggi, precision dan recall yang seimbang, serta kemampuan yang kuat dalam membedakan antara kelas positif dan negatif.