Untitled

October 12, 2024

1 INDEX

1. Data Inspection
2. Data Preprocessing for EDA
3. Exploratory Data Analysis (EDA)
4. Data Preprocessing for Modelling
5. Modelling

2 1.Data Inspection

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
data_1 = pd.read_csv('NUSW-NB15_features.csv', encoding='cp1252')
data_1
```

[4]:
```
     No.             Name      Type  \
0      1            srcip   nominal
1      2            sport   integer
2      3            dstip   nominal
3      4           dsport   integer
4      5            proto   nominal
5      6            state   nominal
6      7              dur     Float
7      8           sbytes   Integer
8      9           dbytes   Integer
9     10             sttl   Integer
10    11             dttl   Integer
11    12            sloss   Integer
12    13            dloss   Integer
13    14          service   nominal
14    15            Sload     Float
15    16            Dload     Float
16    17            Spkts   integer
17    18            Dpkts   integer
```

18	19	swin	integer
19	20	dwin	integer
20	21	stcpb	integer
21	22	dtcpb	integer
22	23	smeansz	integer
23	24	dmeansz	integer
24	25	trans_depth	integer
25	26	res_bdy_len	integer
26	27	Sjit	Float
27	28	Djit	Float
28	29	Stime	Timestamp
29	30	Ltime	Timestamp
30	31	Sintpkt	Float
31	32	Dintpkt	Float
32	33	tcprtt	Float
33	34	synack	Float
34	35	ackdat	Float
35	36	is_sm_ips_ports	Binary
36	37	ct_state_ttl	Integer
37	38	ct_flw_http_mthd	Integer
38	39	is_ftp_login	Binary
39	40	ct_ftp_cmd	integer
40	41	ct_srv_src	integer
41	42	ct_srv_dst	integer
42	43	ct_dst_ltm	integer
43	44	ct_src_ ltm	integer
44	45	ct_src_dport_ltm	integer
45	46	ct_dst_sport_ltm	integer
46	47	ct_dst_src_ltm	integer
47	48	attack_cat	nominal
48	49	Label	binary

	Description
0	Source IP address
1	Source port number
2	Destination IP address
3	Destination port number
4	Transaction protocol
5	Indicates to the state and its dependent proto…
6	Record total duration
7	Source to destination transaction bytes
8	Destination to source transaction bytes
9	Source to destination time to live value
10	Destination to source time to live value
11	Source packets retransmitted or dropped
12	Destination packets retransmitted or dropped
13	http, ftp, smtp, ssh, dns, ftp-data ,irc and …

```
14                                  Source bits per second
15                             Destination bits per second
16                     Source to destination packet count
17                      Destination to source packet count
18                   Source TCP window advertisement value
19              Destination TCP window advertisement value
20                         Source TCP base sequence number
21                    Destination TCP base sequence number
22  Mean of the ?ow packet size transmitted by the…
23  Mean of the ?ow packet size transmitted by the…
24  Represents the pipelined depth into the connec…
25  Actual uncompressed content size of the data t…
26                                    Source jitter (mSec)
27                               Destination jitter (mSec)
28                                       record start time
29                                        record last time
30                  Source interpacket arrival time (mSec)
31             Destination interpacket arrival time (mSec)
32  TCP connection setup round-trip time, the sum …
33  TCP connection setup time, the time between th…
34  TCP connection setup time, the time between th…
35  If source (1) and destination (3)IP addresses …
36  No. for each state (6) according to specific r…
37  No. of flows that has methods such as Get and …
38  If the ftp session is accessed by user and pas…
39     No of flows that has a command in ftp session.
40  No. of connections that contain the same servi…
41  No. of connections that contain the same servi…
42  No. of connections of the same destination add…
43  No. of connections of the same source address …
44  No of connections of the same source address (…
45  No of connections of the same destination addr…
46  No of connections of the same source (1) and t…
47  The name of each attack category. In this data…
48               0 for normal and 1 for attack records
```

[5]:
```python
data_1.tail(5)
```

[5]:
```
    No.              Name     Type  \
44   45  ct_src_dport_ltm  integer
45   46  ct_dst_sport_ltm  integer
46   47    ct_dst_src_ltm  integer
47   48        attack_cat  nominal
48   49             Label   binary

                                          Description
44  No of connections of the same source address (…
```

```
45   No of connections of the same destination addr…
46   No of connections of the same source (1) and t…
47   The name of each attack category. In this data…
48                 0 for normal and 1 for attack records
```

[6]:
```python
data_2 = pd.read_csv('UNSW_NB15_training-set.csv')
data_2
```

[6]:
```
            id       dur proto service state  spkts  dpkts  sbytes  dbytes  \
0            1  0.000011   udp       -   INT      2      0     496       0
1            2  0.000008   udp       -   INT      2      0    1762       0
2            3  0.000005   udp       -   INT      2      0    1068       0
3            4  0.000006   udp       -   INT      2      0     900       0
4            5  0.000010   udp       -   INT      2      0    2126       0
…          …         …   …       …    …     …      …     …      …
82327  82328  0.000005   udp       -   INT      2      0     104       0
82328  82329  1.106101   tcp       -   FIN     20      8   18062     354
82329  82330  0.000000   arp       -   INT      1      0      46       0
82330  82331  0.000000   arp       -   INT      1      0      46       0
82331  82332  0.000009   udp       -   INT      2      0     104       0

                rate  …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  \
0       90909.090200  …                 1               2             0
1      125000.000300  …                 1               2             0
2      200000.005100  …                 1               3             0
3      166666.660800  …                 1               3             0
4      100000.002500  …                 1               3             0
…                …  …               …             …           …
82327  200000.005100  …                 1               2             0
82328      24.410067  …                 1               1             0
82329       0.000000  …                 1               1             0
82330       0.000000  …                 1               1             0
82331  111111.107200  …                 1               1             0

       ct_ftp_cmd  ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  \
0               0                 0           1           2                0
1               0                 0           1           2                0
2               0                 0           1           3                0
3               0                 0           2           3                0
4               0                 0           2           3                0
…             …               …         …         …              …
82327           0                 0           2           1                0
82328           0                 0           3           2                0
82329           0                 0           1           1                1
82330           0                 0           1           1                1
82331           0                 0           1           1                0
```

```
       attack_cat  label
0          Normal      0
1          Normal      0
2          Normal      0
3          Normal      0
4          Normal      0
…             …    …
82327      Normal      0
82328      Normal      0
82329      Normal      0
82330      Normal      0
82331      Normal      0

[82332 rows x 45 columns]
```

[7]:
```
data_2.columns
```

[7]:
```
Index(['id', 'dur', 'proto', 'service', 'state', 'spkts', 'dpkts', 'sbytes',
       'dbytes', 'rate', 'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss',
       'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb', 'dwin',
       'tcprtt', 'synack', 'ackdat', 'smean', 'dmean', 'trans_depth',
       'response_body_len', 'ct_srv_src', 'ct_state_ttl', 'ct_dst_ltm',
       'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',
       'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'ct_src_ltm',
       'ct_srv_dst', 'is_sm_ips_ports', 'attack_cat', 'label'],
      dtype='object')
```

[8]:
```
data_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82332 entries, 0 to 82331
Data columns (total 45 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 82332 non-null  int64
 1   dur                82332 non-null  float64
 2   proto              82332 non-null  object
 3   service            82332 non-null  object
 4   state              82332 non-null  object
 5   spkts              82332 non-null  int64
 6   dpkts              82332 non-null  int64
 7   sbytes             82332 non-null  int64
 8   dbytes             82332 non-null  int64
 9   rate               82332 non-null  float64
 10  sttl               82332 non-null  int64
 11  dttl               82332 non-null  int64
 12  sload              82332 non-null  float64
 13  dload              82332 non-null  float64
```

```
 14  sloss              82332 non-null  int64
 15  dloss              82332 non-null  int64
 16  sinpkt             82332 non-null  float64
 17  dinpkt             82332 non-null  float64
 18  sjit               82332 non-null  float64
 19  djit               82332 non-null  float64
 20  swin               82332 non-null  int64
 21  stcpb              82332 non-null  int64
 22  dtcpb              82332 non-null  int64
 23  dwin               82332 non-null  int64
 24  tcprtt             82332 non-null  float64
 25  synack             82332 non-null  float64
 26  ackdat             82332 non-null  float64
 27  smean              82332 non-null  int64
 28  dmean              82332 non-null  int64
 29  trans_depth        82332 non-null  int64
 30  response_body_len  82332 non-null  int64
 31  ct_srv_src         82332 non-null  int64
 32  ct_state_ttl       82332 non-null  int64
 33  ct_dst_ltm         82332 non-null  int64
 34  ct_src_dport_ltm   82332 non-null  int64
 35  ct_dst_sport_ltm   82332 non-null  int64
 36  ct_dst_src_ltm     82332 non-null  int64
 37  is_ftp_login       82332 non-null  int64
 38  ct_ftp_cmd         82332 non-null  int64
 39  ct_flw_http_mthd   82332 non-null  int64
 40  ct_src_ltm         82332 non-null  int64
 41  ct_srv_dst         82332 non-null  int64
 42  is_sm_ips_ports    82332 non-null  int64
 43  attack_cat         82332 non-null  object
 44  label              82332 non-null  int64
dtypes: float64(11), int64(30), object(4)
memory usage: 28.3+ MB
```

[9]:
```
data_2.describe()
```

[9]:
```
                 id           dur         spkts         dpkts        sbytes  \
count  82332.000000  82332.000000  82332.000000  82332.000000  8.233200e+04
mean   41166.500000      1.006756     18.666472     17.545936  7.993908e+03
std    23767.345519      4.710444    133.916353    115.574086  1.716423e+05
min        1.000000      0.000000      1.000000      0.000000  2.400000e+01
25%    20583.750000      0.000008      2.000000      0.000000  1.140000e+02
50%    41166.500000      0.014138      6.000000      2.000000  5.340000e+02
75%    61749.250000      0.719360     12.000000     10.000000  1.280000e+03
max    82332.000000     59.999989  10646.000000  11018.000000  1.435577e+07

             dbytes          rate          sttl          dttl         sload  \
```

```
count  8.233200e+04  8.233200e+04  82332.000000  82332.000000  8.233200e+04
mean   1.323379e+04  8.241089e+04    180.967667     95.713003  6.454902e+07
std    1.514715e+05  1.486204e+05    101.513358    116.667722  1.798618e+08
min    0.000000e+00  0.000000e+00      0.000000      0.000000  0.000000e+00
25%    0.000000e+00  2.860611e+01     62.000000      0.000000  1.120247e+04
50%    1.780000e+02  2.650177e+03    254.000000     29.000000  5.770032e+05
75%    9.560000e+02  1.111111e+05    254.000000    252.000000  6.514286e+07
max    1.465753e+07  1.000000e+06    255.000000    253.000000  5.268000e+09

       …  ct_src_dport_ltm  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  \
count  …      82332.000000      82332.000000    82332.000000  82332.000000
mean   …          4.928898          3.663011        7.456360      0.008284
std    …          8.389545          5.915386       11.415191      0.091171
min    …          1.000000          1.000000        1.000000      0.000000
25%    …          1.000000          1.000000        1.000000      0.000000
50%    …          1.000000          1.000000        3.000000      0.000000
75%    …          4.000000          3.000000        6.000000      0.000000
max    …         59.000000         38.000000       63.000000      2.000000

         ct_ftp_cmd  ct_flw_http_mthd    ct_src_ltm    ct_srv_dst  \
count  82332.000000      82332.000000  82332.000000  82332.000000
mean       0.008381          0.129743      6.468360      9.164262
std        0.092485          0.638683      8.543927     11.121413
min        0.000000          0.000000      1.000000      1.000000
25%        0.000000          0.000000      1.000000      2.000000
50%        0.000000          0.000000      3.000000      5.000000
75%        0.000000          0.000000      7.000000     11.000000
max        2.000000         16.000000     60.000000     62.000000

       is_sm_ips_ports         label
count     82332.000000  82332.000000
mean          0.011126      0.550600
std           0.104891      0.497436
min           0.000000      0.000000
25%           0.000000      0.000000
50%           0.000000      1.000000
75%           0.000000      1.000000
max           1.000000      1.000000

[8 rows x 41 columns]
```

[10]:

```
data_2.describe(include='object')
```

[10]:

```
        proto service  state attack_cat
count   82332   82332  82332      82332
unique    131      13      7         10
top       tcp       -    FIN     Normal
```

```
freq     43095    47153   39339       37000
```

3 Data Preprocessing for EDA

[12]:
```python
data_2.isna()
#method returns the dataframe of the same shape as data_2 where each element is␣
↪boolean(true for missing and false for non-missing values)
```

[12]:
```
          id    dur  proto  service  state  spkts  dpkts  sbytes  dbytes  \
0      False  False  False    False  False  False  False   False   False
1      False  False  False    False  False  False  False   False   False
2      False  False  False    False  False  False  False   False   False
3      False  False  False    False  False  False  False   False   False
4      False  False  False    False  False  False  False   False   False
…        …     …      …       …      …      …      …       …       …
82327  False  False  False    False  False  False  False   False   False
82328  False  False  False    False  False  False  False   False   False
82329  False  False  False    False  False  False  False   False   False
82330  False  False  False    False  False  False  False   False   False
82331  False  False  False    False  False  False  False   False   False

        rate  …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  ct_ftp_cmd  \
0      False  …             False           False         False       False
1      False  …             False           False         False       False
2      False  …             False           False         False       False
3      False  …             False           False         False       False
4      False  …             False           False         False       False
…        …  …                 …               …             …           …
82327  False  …             False           False         False       False
82328  False  …             False           False         False       False
82329  False  …             False           False         False       False
82330  False  …             False           False         False       False
82331  False  …             False           False         False       False

       ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  attack_cat  \
0                 False       False       False            False       False
1                 False       False       False            False       False
2                 False       False       False            False       False
3                 False       False       False            False       False
4                 False       False       False            False       False
…                     …           …           …                …           …
82327             False       False       False            False       False
82328             False       False       False            False       False
82329             False       False       False            False       False
82330             False       False       False            False       False
82331             False       False       False            False       False
```

```
        label
0       False
1       False
2       False
3       False
4       False
…       …
82327   False
82328   False
82329   False
82330   False
82331   False

[82332 rows x 45 columns]
```

[13]:

```
data_2.isna().sum()
#it is used for finding the number of missing values
```

[13]:

```
id                   0
dur                  0
proto                0
service              0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sttl                 0
dttl                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
synack               0
ackdat               0
smean                0
```

```
dmean                  0
trans_depth            0
response_body_len      0
ct_srv_src             0
ct_state_ttl           0
ct_dst_ltm             0
ct_src_dport_ltm       0
ct_dst_sport_ltm       0
ct_dst_src_ltm         0
is_ftp_login           0
ct_ftp_cmd             0
ct_flw_http_mthd       0
ct_src_ltm             0
ct_srv_dst             0
is_sm_ips_ports        0
attack_cat             0
label                  0
dtype: int64
```

[14]:
```python
data_2.dropna(inplace=True)
data_2.isna().sum().sum()
```

[14]: 0

[15]:
```python
# Replacing '-' in state and service for 'other'
data_2['state'] = data_2['state'].replace('-','other')
data_2['service'] = data_2['service'].replace('-','other')
```

[16]:
```python
data_2
```

[16]:
```
          id       dur proto service state  spkts  dpkts  sbytes  dbytes  \
0          1  0.000011   udp   other   INT      2      0     496       0
1          2  0.000008   udp   other   INT      2      0    1762       0
2          3  0.000005   udp   other   INT      2      0    1068       0
3          4  0.000006   udp   other   INT      2      0     900       0
4          5  0.000010   udp   other   INT      2      0    2126       0
…      …         …   …      …   …     …     …      …      …
82327  82328  0.000005   udp   other   INT      2      0     104       0
82328  82329  1.106101   tcp   other   FIN     20      8   18062     354
82329  82330  0.000000   arp   other   INT      1      0      46       0
82330  82331  0.000000   arp   other   INT      1      0      46       0
82331  82332  0.000009   udp   other   INT      2      0     104       0

               rate  …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  \
0       90909.090200  …                 1               2             0
1      125000.000300  …                 1               2             0
2      200000.005100  …                 1               3             0
```

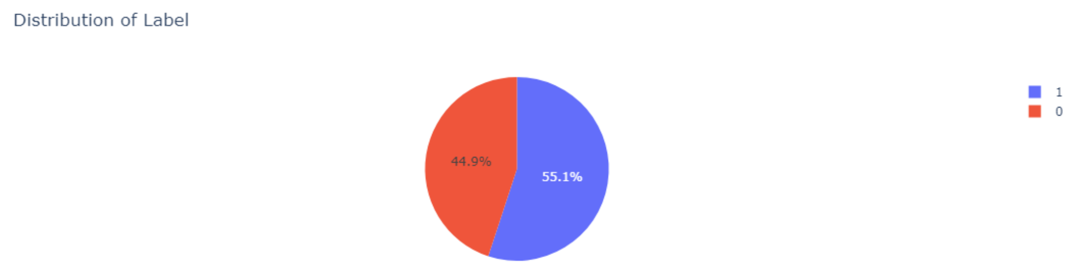
```
3      166666.660800  …                 1              3             0
4      100000.002500  …                 1              3             0
…                 …   …           …              …             …
82327  200000.005100  …                 1              2             0
82328      24.410067  …                 1              1             0
82329       0.000000  …                 1              1             0
82330       0.000000  …                 1              1             0
82331  111111.107200  …                 1              1             0

       ct_ftp_cmd  ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  \
0               0                 0           1           2                0
1               0                 0           1           2                0
2               0                 0           1           3                0
3               0                 0           2           3                0
4               0                 0           2           3                0
…             …                …          …           …              …
82327           0                 0           2           1                0
82328           0                 0           3           2                0
82329           0                 0           1           1                1
82330           0                 0           1           1                1
82331           0                 0           1           1                0

       attack_cat  label
0          Normal      0
1          Normal      0
2          Normal      0
3          Normal      0
4          Normal      0
…             …    …
82327      Normal      0
82328      Normal      0
82329      Normal      0
82330      Normal      0
82331      Normal      0

[82332 rows x 45 columns]
```

4 EDA

[18]:
```python
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

[19]:
```python
counts = data_2['label'].value_counts()
fig = px.pie(counts, values=counts.values, names=counts.index,␣
 ↪title='Distribution of Label')
```

```
fig.show()
```



[20]:
```
data_2[data_2['attack_cat']!='normal'].groupby('attack_cat').size()
#The UNSW-NB15 dataset contains both normal network traffic and a wide variety
 ↪of malicious network traffic (attacks), and the "attack_cat" field in the
 ↪dataset categorizes the type of attack. There are nine major attack
 ↪categories in the dataset:
```

[20]:
```
attack_cat
Analysis            677
Backdoor            583
DoS                4089
Exploits          11132
Fuzzers            6062
Generic           18871
Normal            37000
Reconnaissance     3496
Shellcode           378
Worms                44
dtype: int64
```

- Fuzzers: Sending malformed or semi-malformed data to a system in order to crash it or discover vulnerabilities.
- Analysis: Various tools and activities such as port scanning, spam, and mail bomb attacks.
- Backdoor: Attacks where a system is remotely accessed without authorization, bypassing normal authentication procedures.
- DoS: Denial of Service attacks aimed at making a system unavailable to its intended users by overwhelming it with traffic.
- Exploits: Taking advantage of bugs or vulnerabilities in software or hardware systems.
- Generic: Cryptography attacks that work on all block ciphers, independent of the structure of the cipher.
- Reconnaissance: Gathering information about the target system to find vulnerabilities, often via scanning and probing.
- Shellcode: Exploiting a system by injecting executable code into a vulnerable program to give

an attacker control.
- Worms: Self-replicating malware that spreads across systems, typically through the network

4.1 Grouping "backdoor" and "backdoors" into a single category for anomaly detection in the UNSW-NB15 dataset, or in general, can be important for the following reasons:

1. Generalization of Attack Behavior: • Backdoor attacks refer to any unauthorized access or control over a system by bypassing standard authentication mechanisms. Whether it's labeled as "backdoor" or "backdoors," the underlying behavior is the same: exploiting a vulnerability for remote control. • By grouping similar attack types (like backdoor and backdoors), anomaly detection systems can generalize the attack signatures and patterns, making the model more robust in detecting various instances of the same category.
2. Simplification of Classification: • Anomaly detection models often work better with fewer, clearly defined categories because it reduces noise and redundancy in the data. Grouping closely related or identical attack types reduces the complexity of the classification task, leading to more efficient model training and better detection rates. • In this case, treating "backdoor" and "backdoors" as separate categories would add unnecessary duplication, as the difference between them is minimal or only semantic.
3. Better Detection Performance: • By grouping similar types of attacks together, you increase the amount of training data available for each attack category. This can lead to better anomaly detection performance since machine learning models typically perform better with larger datasets. • Combining similar attack categories helps avoid under-representation of individual categories, which could cause the model to misclassify or overlook important patterns.
4. Clarity in Attack Categorization: • From a data management perspective, having multiple names for essentially the same type of attack can cause confusion. Grouping them under a single category ensures clarity and consistency when labeling and analyzing attack types. • Backdoor attacks, whether singular or plural in their description, generally exploit similar vulnerabilities or weaknesses in systems, so they naturally belong in the same category.
5. Reduced Overlap and Ambiguity: • In anomaly detection, minimizing overlap and ambiguity between different categories is critical to improving classification accuracy. If two attack types are very similar but are kept as separate categories, there might be significant overlap in their features, leading to poor model performance. Grouping them reduces this issue.

[23]:
```
data_2['attack_cat'] = data_2['attack_cat'].apply(lambda item: 'Backdoor' if␣
↪item =='Backdoors' else item)
```

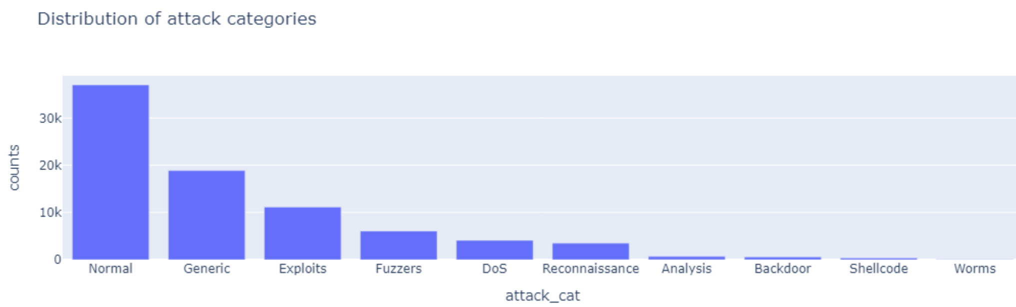
[24]:
```
attack_by_cat = data_2[data_2['attack_cat']!='normal'].groupby('attack_cat').
↪size().reset_index(name='counts')
attack_by_cat
```

[24]:

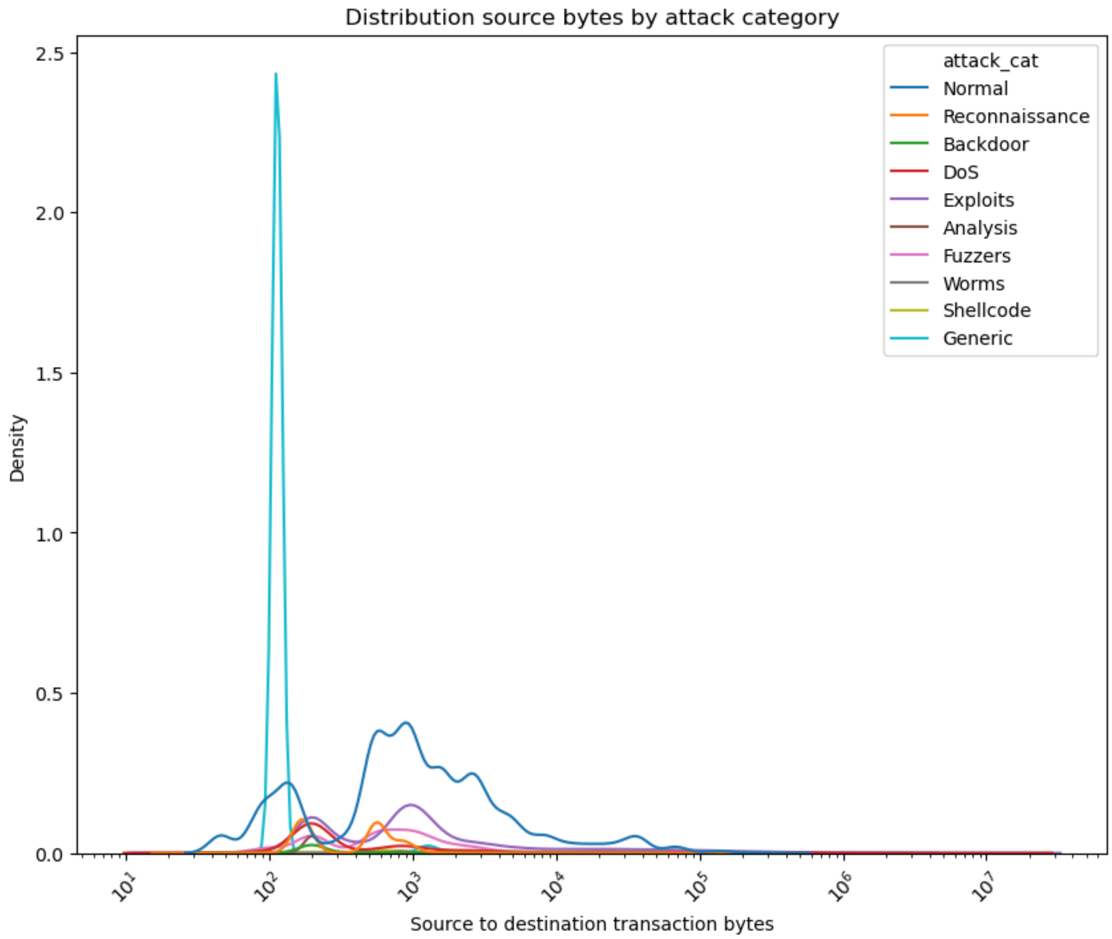
	attack_cat	counts
0	Analysis	677
1	Backdoor	583
2	DoS	4089
3	Exploits	11132
4	Fuzzers	6062

```
5          Generic   18871
6           Normal   37000
7   Reconnaissance    3496
8        Shellcode     378
9            Worms      44
```

[25]:
```
attack_by_cat = data_2[data_2['attack_cat']!='normal'].groupby('attack_cat').
↪size().reset_index(name='counts').sort_values(by='counts', ascending = False)
fig = px.bar(attack_by_cat, x='attack_cat', y='counts', title='Distribution of␣
↪attack categories')
fig.show()
```



[26]:
```
plt.figure(figsize=(10,8))
ax = sns.
↪kdeplot(x='sbytes',hue='attack_cat',log_scale=True,data=data_2[data_2['attack_cat']!
↪='normal'])
ax.set_title('Distribution source bytes by attack category')
ax.set_xlabel('Source to destination transaction bytes')
ax.set_ylabel('Density')
plt.xticks(rotation=45)
plt.show()
```

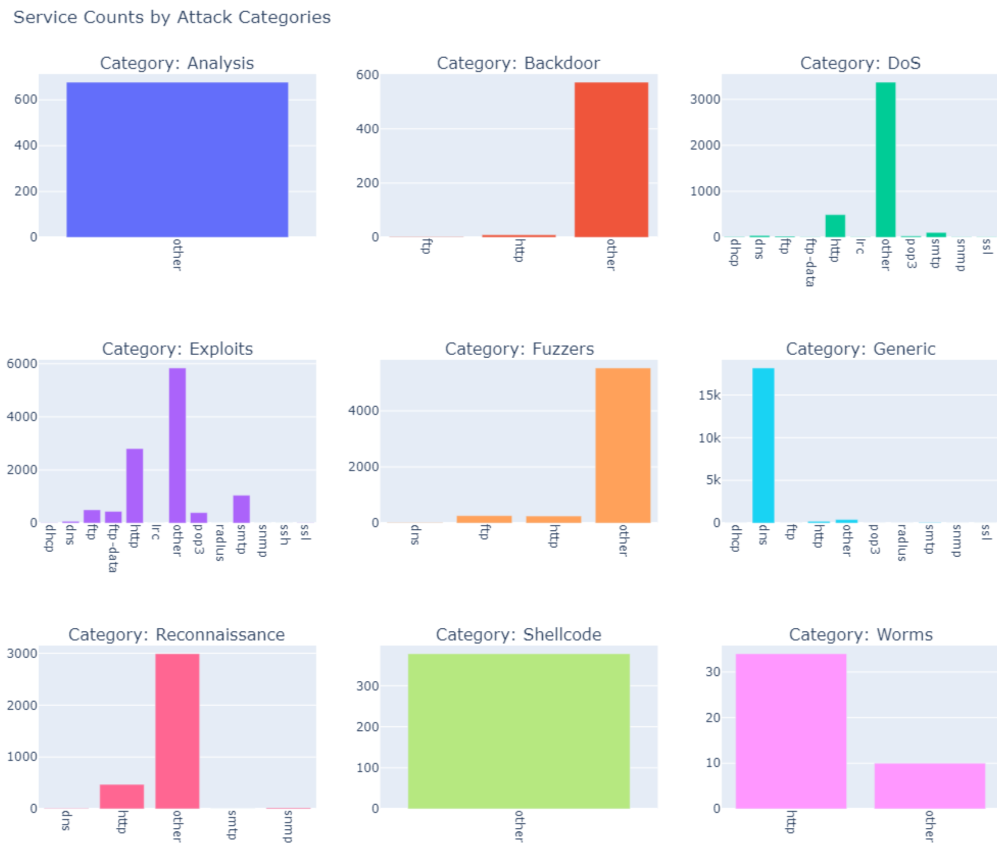


4.2 Most Used Services by Attack Category:* HTTP (Port 80, 8080) – Common in Exploits, Backdoor, and Fuzzers

- DNS (Port 53) – Targeted in Reconnaissance and DoS attacks.
- SSH (Port 22) – Attacked for Backdoor and Exploits.
- FTP (Port 21) – Vulnerable to Fuzzers, Exploits, and Backdoor.
- Telnet (Port 23) – Targeted for Backdoor and Exploits.
- MySQL (Port 3306) – Used in SQL Injection and Exploits.
- SNMP (Port 161) – Used for Reconnaissance and DoS.
- RDP (Port 3389) – Targeted for Backdoor and Exploits.
- NTP (Port 123) – Common in DoS attacks. These services represent common targets for attackers and are heavily exploited based on their functionalities and vulnerabilities in different network environments vironmentsfferent network environments.

[28]:
```
data_2_attacks_cat_services = data_2[(data_2['attack_cat']!='normal')].
↪groupby(['attack_cat','service']).size().reset_index(name='Count')
```

[29]:
```python
# Create a 3x3 subplot grid
fig = make_subplots(rows=3, cols=3,
                    subplot_titles=('Category: Analysis', 'Category: Backdoor',␣
↪'Category: DoS',
                                    'Category: Exploits', 'Category: Fuzzers',␣
↪'Category: Generic',
                                    'Category: Reconnaissance', 'Category:␣
↪Shellcode', 'Category: Worms'))

# Create a list of attack categories to loop through
categories = ['Analysis', 'Backdoor', 'DoS', 'Exploits', 'Fuzzers', 'Generic',␣
↪'Reconnaissance', 'Shellcode', 'Worms']
# Define row and column positions for each plot
positions = [(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)]

# Loop through the categories and create a bar plot for each one
for i, category in enumerate(categories):
    # Filter the dataframe for each attack category
    data_2_filtered =␣
↪data_2_attacks_cat_services[data_2_attacks_cat_services['attack_cat'] ==␣
↪category]

    # Add bar plot to the subplot
    fig.add_trace(go.Bar(x=data_2_filtered['service'],␣
↪y=data_2_filtered['Count'], name=category,
                         marker_color=px.colors.qualitative.Plotly[i]),
                  row=positions[i][0], col=positions[i][1])
    # Update layout
fig.update_layout(height=900, width=900, title_text="Service Counts by Attack␣
↪Categories", showlegend=False)

# Update x-axis for all subplots
fig.update_xaxes(tickangle=90)

# Show the figure
fig.show()
```



4.3 Top 5 protocols used by attack category

In the UNSW-NB15 dataset, various attack categories utilize different network protocols. The top 5 protocols most commonly used by different attack categories can be identified by analyzing the dataset based on the frequency of protocols associated with specific attack types. Here is a typical breakdown of the top 5 protocols used by attack categories based on the UNSW-NB15 dataset: 1. TCP (Transmission Control Protocol) • Attack Categories: Most attack types, including Backdoor, DoS, Exploits, and Worms, heavily utilize TCP because it is the most widely used protocol for reliable, connection-oriented communication. • Usage: TCP is often targeted in network-based attacks (e.g., port scanning, denial-of-service) because it handles core services like web traffic (HTTP/HTTPS), email (SMTP), and file transfers (FTP). 2. UDP (User Datagram Protocol) • Attack Categories: DoS and Reconnaissance attacks often exploit UDP since it is connectionless and offers less security than TCP. • Usage: UDP is common in attacks targeting services like DNS, DHCP, and streaming applications because it doesn't require a connection setup, making it faster but easier to exploit for flooding attacks. 3. ICMP (Internet Control Message Protocol) • Attack Categories: Primarily used in Reconnaissance (scanning) and DoS attacks. • Usage: ICMP is used for sending error messages and operational information, but it is also used for ping sweepsand traceroute during reconnaissance or flooding attacks in a DoS context (e.g., ICMP flood). 4. HTTP (HyperText Transfer Protocol) • Attack Categories: Exploits, Backdoor, and Fuzzers. • Usage:

Many exploits and backdoors target web servers, making HTTP one of the top protocols used in such attacks. Web-based attacks like SQL injections, cross-site scripting (XSS), and backdoor connections often use HTTP. 5. DNS (Domain Name System) • Attack Categories: Reconnaissance and DoS. • Usage: DNS is often targeted for reconnaissance attacks (e.g., DNS zone transfers to gather information) and also for DNS amplification attacks in DoS scenarios.

[31]:
```
data_2_attacks_cat_proto = data_2[(data_2['attack_cat']!='normal')].
 ↪groupby(['attack_cat','proto']).size().reset_index(name='Count')
```

[32]:
```
# Create a 3x3 subplot grid
fig = make_subplots(rows=3, cols=3,
                    subplot_titles=('Category: Analysis', 'Category: Backdoor',␣
 ↪'Category: DoS',
                                    'Category: Exploits', 'Category: Fuzzers',␣
 ↪'Category: Generic',
                                    'Category: Reconnaissance', 'Category:␣
 ↪Shellcode', 'Category: Worms'))

# Create a list of attack categories to loop through
categories = ['Analysis', 'Backdoor', 'DoS', 'Exploits', 'Fuzzers', 'Generic',␣
 ↪'Reconnaissance', 'Shellcode', 'Worms']

# Define row and column positions for each plot
positions = [(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)]

# Loop through the categories and create a bar plot for each one (top 5␣
 ↪protocols)
for i, category in enumerate(categories):
    # Filter the dataframe for each attack category and take the top 5 protocols
    data_2_filtered =␣
 ↪data_2_attacks_cat_proto[data_2_attacks_cat_proto['attack_cat'] ==␣
 ↪category][:5]

    # Add bar plot to the subplot
    fig.add_trace(go.Bar(x=data_2_filtered['proto'],␣
 ↪y=data_2_filtered['Count'], name=category,
                         marker_color=px.colors.qualitative.Plotly[i]),
                  row=positions[i][0], col=positions[i][1])

# Update layout
fig.update_layout(height=900, width=900, title_text="Top 5 Protocols by Attack␣
 ↪Categories", showlegend=False)

# Update x-axis for all subplots
fig.update_xaxes(tickangle=90)

# Show the figure
```

```
fig.show()
```

4.4 Distribution of states by attack category

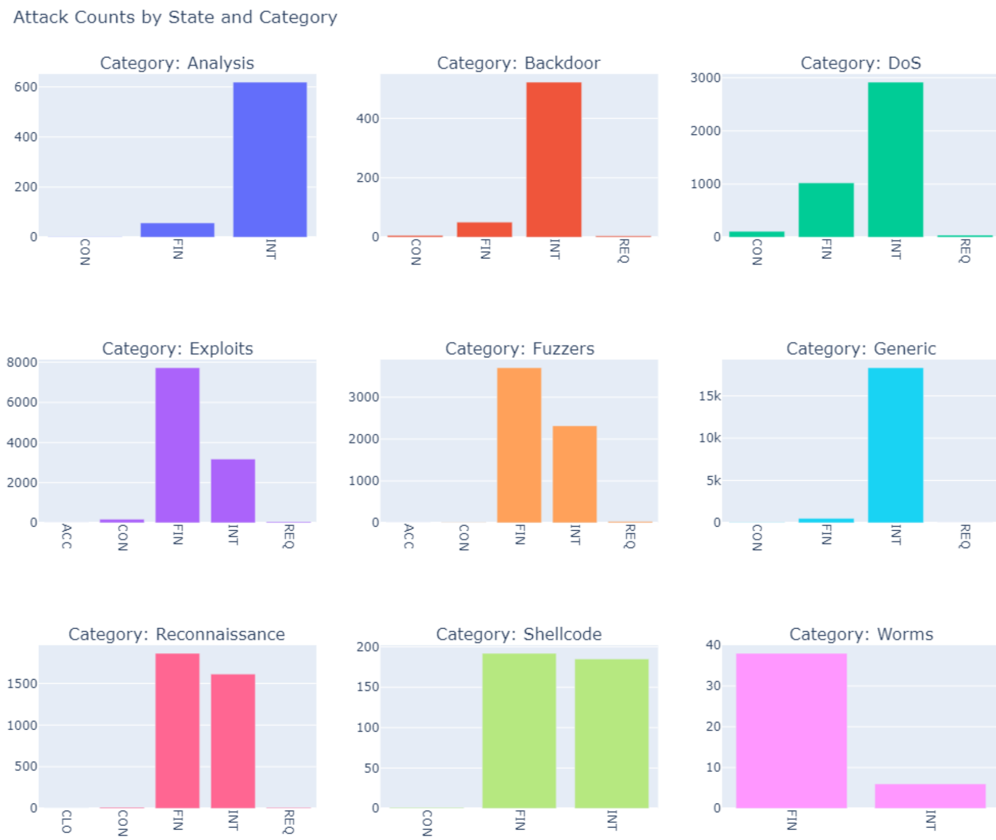
4.4.1 Why distribution is required?

- Distributing states by attack category in anomaly detection allows for better attack recognition, improves detection accuracy, facilitates more effective responses, and enhances the overall security of the system. It helps security systems and professionals better understand the nature of attacks, prioritize resources, and reduce false positives or negatives.

[34]:
```
data_2_attacks_cat_state = data_2[(data_2['attack_cat']!='normal')].
↪groupby(['attack_cat','state']).size().reset_index(name='Count')
```

[35]:
```
# Create a 3x3 subplot grid
fig = make_subplots(rows=3, cols=3,
                    subplot_titles=('Category: Analysis', 'Category: Backdoor',␣
↪'Category: DoS',
```

```
                                     'Category: Exploits', 'Category: Fuzzers',␣
↪'Category: Generic',
                                     'Category: Reconnaissance', 'Category:␣
↪Shellcode', 'Category: Worms'))

# List of attack categories
categories = ['Analysis', 'Backdoor', 'DoS', 'Exploits', 'Fuzzers', 'Generic',␣
↪'Reconnaissance', 'Shellcode', 'Worms']

# Define the row and column positions for the plots
positions = [(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)]

# Loop through the attack categories and create a bar plot for each one
for i, category in enumerate(categories):
    # Filter the dataframe for each attack category
    data_2_filtered =␣
↪data_2_attacks_cat_state[data_2_attacks_cat_state['attack_cat'] == category]

    # Add bar plot to the subplot
    fig.add_trace(go.Bar(x=data_2_filtered['state'],␣
↪y=data_2_filtered['Count'], name=category,
                         marker_color=px.colors.qualitative.Plotly[i]),
                  row=positions[i][0], col=positions[i][1])

# Update the layout of the entire figure
fig.update_layout(height=900, width=900, title_text="Attack Counts by State and␣
↪Category", showlegend=False)

# Rotate x-axis labels for better readability
fig.update_xaxes(tickangle=90)

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



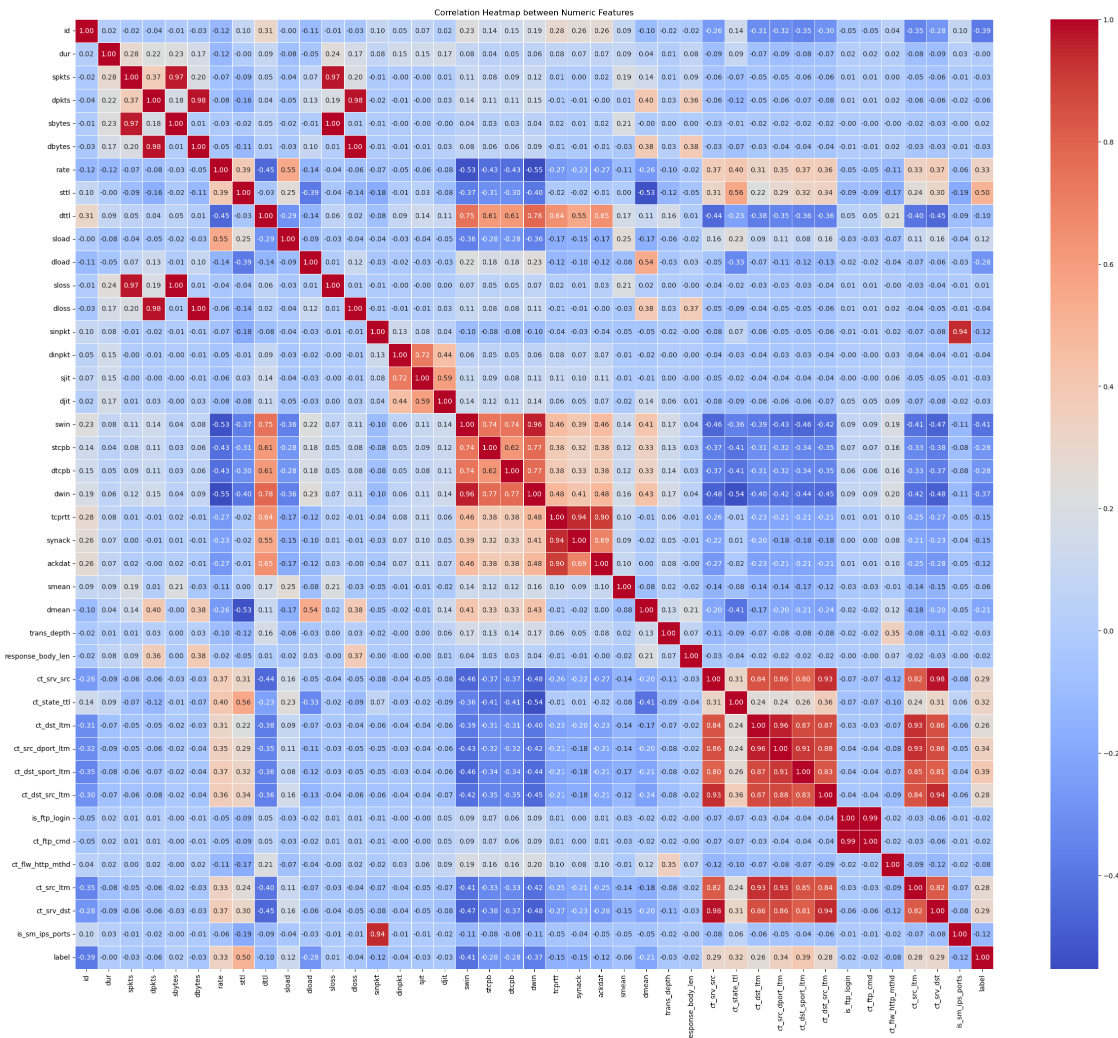
4.5 Summary of EDA

The Exploratory Data Analysis (EDA) on this dataset helps in understanding its structure, distributions, relationships between features, and patterns in attack behavior. Here's a summarized EDA process for the dataset: *Class Imbalance: The dataset is imbalanced, with attack categories like Generic and Exploits being more common. This requires attention during model training, such as using resampling techniques.* Feature Relationships: Certain features, like protocols and services, are strongly associated with specific attack types, which can be useful for feature selection and model interpretation. * Outliers and Skewed Features: Several features contain outliers and extreme skewness, which should be addressed in model preprocessing

The distribution of attack types is uneven, with the majority being generic. Similarly, the distribution of source bytes sent by attack type follows a comparable pattern, showing no significant differences. Most attacks utilize the HTTP service (excluding the 'other' category). 3PC and Argus are protocols frequently employed in distributed computing environments, commonly used by most attack types alongside TCP and UDP. The majority of connections are in the INT and FIN states.

.

[37]:
```
numerical_columns = [col for col in data_2.columns if data_2[col].dtypes !=␣
↪"object"]
# Calculate the correlation matrix for numerical columns
corr = data_2[numerical_columns].corr()

plt.figure(figsize=(30, 25))
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5, fmt='.2f')
plt.title('Correlation Heatmap between Numeric Features')
plt.show()
```



[38]:
```
# Calculate the absolute correlation matrix
corr_matrix = data_2[numerical_columns].corr().abs()

# Extract the upper triangle of the correlation matrix
```

```
upper_triangle = corr_matrix.mask(np.tril(np.ones(corr_matrix.shape), k=0).
 ↪astype(bool))

# Stack the upper triangle and reset the index to create a DataFrame
high_corr_pairs = upper_triangle.unstack().dropna().reset_index()

# Rename the columns for clarity
high_corr_pairs.columns = ['Feature 1', 'Feature 2', 'Correlation']

# Filter the pairs with a correlation higher than 0.80
high_corr_pairs = high_corr_pairs[high_corr_pairs['Correlation'] > 0.80]

# Display the result
high_corr_pairs
```

[38]:

	Feature 1	Feature 2	Correlation
8	sbytes	spkts	0.965750
13	dbytes	dpkts	0.976419
57	sloss	spkts	0.973644
59	sloss	sbytes	0.995027
69	dloss	dpkts	0.981506
71	dloss	dbytes	0.997109
207	dwin	swin	0.960125
252	synack	tcprtt	0.939473
274	ackdat	tcprtt	0.897688
463	ct_dst_ltm	ct_srv_src	0.842195
493	ct_src_dport_ltm	ct_srv_src	0.855472
495	ct_src_dport_ltm	ct_dst_ltm	0.960401
524	ct_dst_sport_ltm	ct_srv_src	0.801164
526	ct_dst_sport_ltm	ct_dst_ltm	0.872185
527	ct_dst_sport_ltm	ct_src_dport_ltm	0.911637
556	ct_dst_src_ltm	ct_srv_src	0.933795
558	ct_dst_src_ltm	ct_dst_ltm	0.868150
559	ct_dst_src_ltm	ct_src_dport_ltm	0.876030
560	ct_dst_src_ltm	ct_dst_sport_ltm	0.830993
629	ct_ftp_cmd	is_ftp_login	0.994341
694	ct_src_ltm	ct_srv_src	0.822486
696	ct_src_ltm	ct_dst_ltm	0.932252
697	ct_src_ltm	ct_src_dport_ltm	0.933172
698	ct_src_ltm	ct_dst_sport_ltm	0.847314
699	ct_src_ltm	ct_dst_src_ltm	0.840012
731	ct_srv_dst	ct_srv_src	0.977849
733	ct_srv_dst	ct_dst_ltm	0.855158
734	ct_srv_dst	ct_src_dport_ltm	0.863614
735	ct_srv_dst	ct_dst_sport_ltm	0.810954
736	ct_srv_dst	ct_dst_src_ltm	0.941047
740	ct_srv_dst	ct_src_ltm	0.822407

```
754    is_sm_ips_ports            sinpkt     0.944506
```

[39]:
```
# List of features to drop from the DataFrame
features_to_drop = ['ct_state_ttl', 'ct_dst_sport_ltm', 'ct_src_ltm']

# Dropping the specified features (columns) from the DataFrame
# 'axis=1' specifies that we are dropping columns, not rows
data_2 = data_2.drop(features_to_drop, axis=1)

# The DataFrame 'data_2' now no longer contains the dropped columns
```

5 4. Preprocessing for the Model

- Preprocessing ensures that the UNSW-NB15 dataset is clean, well-structured, and in the right format for training a machine learning model. It helps avoid biases and ensures that the model learns effectively from the data.

5.0.1 Encoding

- Encoding is an essential step in preprocessing the UNSW-NB15 dataset because many machine learning models require numerical input. The dataset contains several categorical features, and encoding transforms these categorical variables into a numerical format, which models can process
- Without encoding, the model would either ignore or incorrectly interpret important features, leading to suboptimal performance

[42]:
```
# Select the columns in the DataFrame that are of categorical data type (object)
categorical_columns = data_2.select_dtypes(include=['object']).columns

# The 'categorical_columns' variable now contains the names of all categorical␣
↪columns in the DataFrame
categorical_columns
```

[42]:
```
Index(['proto', 'service', 'state', 'attack_cat'], dtype='object')
```

[43]:
```
# Apply one-hot encoding to the 'proto' and 'service' columns in the DataFrame
# The 'drop_first=True' parameter ensures that the first category is dropped to␣
↪avoid multicollinearity
data_2 = pd.get_dummies(data_2, columns=['proto', 'service'], drop_first=True)

# The DataFrame 'data_2' now includes one-hot encoded versions of the 'proto'␣
↪and 'service' columns
```

[44]:
```
from sklearn.preprocessing import LabelEncoder
# The LabelEncoder class from the sklearn.preprocessing module is an essential␣
↪tool in the machine learning workflow, particularly for handling categorical␣
↪data
```

[45]:
```
# Initialize the label encoder for transforming categorical variables into␣
↪numerical form
label_encoder = LabelEncoder()

# Apply label encoding to the 'state' column from the original DataFrame and␣
↪store the result in 'data_2'
data_2['state'] = label_encoder.fit_transform(data_2['state'])

# Similarly, apply label encoding to the 'attack_cat' column and store it in␣
↪'data_2'
data_2['attack_cat'] = label_encoder.fit_transform(data_2['attack_cat'])

# The 'state' and 'attack_cat' columns in 'data_2' are now encoded as numerical␣
↪values
```

5.1 Feature Engeneering

- feature engineering in the UNSW-NB15 dataset is vital for building effective intrusion detection models. It allows for enhanced model performance, better interpretability, and more robust detection capabilities by transforming and creating features that capture the underlying patterns in the data.

[47]:
```
# Create a new feature in the DataFrame called 'load_interaction'
# This feature is the product of 'sload' (source load) and 'dload' (destination␣
↪load)
data_2['load_interaction'] = data_2['sload'] * data_2['dload']

# This code calculates the interaction between source load and destination load␣
↪by multiplying their respective values and stores the result in a new column␣
↪called load_interaction
```

[48]:
```
# Calculate the total number of transaction bytes exchanged between the source␣
↪and destination
# The new feature 'total_bytes' is created by summing the 'sbytes' (source␣
↪bytes) and 'dbytes' (destination bytes)
data_2['total_bytes'] = data_2['sbytes'] + data_2['dbytes']

# The 'total_bytes' column now contains the total bytes transmitted in each␣
↪transaction
```

This code creates a new column called total_bytes in the data_2 DataFrame, representing the sum of source bytes and destination bytes for each transaction.

[50]:
```
# Calculate the packet flow ratio between source and destination
data_2['pkt_flow_ratio'] = data_2['spkts'] / (data_2['dpkts'] + 1)

# Calculate the difference in bytes between source and destination
data_2['bytes_diff'] = data_2['sbytes'] - data_2['dbytes']
```

```
# Calculate the ratio of source bytes to destination bytes
data_2['bytes_ratio'] = data_2['sbytes'] / (data_2['dbytes'] + 1)

# Calculate the difference in Time to Live (TTL) between source and destination
data_2['ttl_diff'] = data_2['sttl'] - data_2['dttl']

# Calculate the difference in jitter between source and destination
data_2['jitter_diff'] = data_2['sjit'] - data_2['djit']

# Calculate the ratio of source jitter to destination jitter
data_2['jitter_ratio'] = data_2['sjit'] / (data_2['djit'] + 1)

# Calculate the difference between SYN-ACK and ACK data times
data_2['tcp_time_diff'] = data_2['synack'] - data_2['ackdat']

# Display a random sample of 10 rows from the updated DataFrame
data_2.sample(10)
```

[50]:

```
          id       dur  state  spkts  dpkts  sbytes  dbytes           rate  \
53823  53824  4.554701      3    186     44  220187    2740      50.277726
13674  13675  0.000009      4      2      0     114       0  111111.107200
3315    3316  1.438045      3      8      8     364     510      10.430828
26023  26024  0.006131      3     14      6    8928     320    3099.005114
2890    2891  0.000005      4      2      0     200       0  200000.005100
30118  30119  1.006176      3     14     18    1684   10168      30.809719
19702  19703  0.000007      4      2      0     114       0  142857.140900
72271  72272  0.683291      3     10      6     582     268      21.952579
59833  59834  0.566213      3     14      6    8854     268      33.556276
31994  31995  0.018476      3     16     18    1540    1644    1786.100882

       sttl  dttl  …  service_ssl  load_interaction  total_bytes  \
53823    62   252  …        False      1.809351e+09       222927
13674   254     0  …        False      0.000000e+00          114
3315     62   252  …        False      4.412992e+06          874
26023    31    29  …        False      3.769082e+12         9248
2890    254     0  …        False      0.000000e+00          200
30118    31    29  …        False      9.495568e+08        11852
19702   254     0  …        False      0.000000e+00          114
72271   254   252  …        False      1.608970e+07          850
59833   254   252  …        False      3.676595e+08         9122
31994    31    29  …        False      4.204387e+11         3184

       pkt_flow_ratio  bytes_diff  bytes_ratio  ttl_diff   jitter_diff  \
53823        4.133333      217447    80.330901      -190   2475.005107
13674        2.000000         114   114.000000       254      0.000000
3315         0.888889        -146     0.712329      -190  13569.645061
```

```
26023        2.000000       8608    27.813084          2       21.826970
2890         2.000000        200   200.000000        254        0.000000
30118        0.736842      -8484     0.165601          2     1091.504531
19702        2.000000        114   114.000000        254        0.000000
72271        1.428571        314     2.163569          2   756141.746900
59833        2.000000       8586    32.914498          2     2394.610993
31994        0.842105       -104     0.936170          2       71.610908

       jitter_ratio  tcp_time_diff
53823     19.249577      -0.060822
13674      0.000000       0.000000
3315      40.462983      -0.015380
26023      9.464225       0.000322
2890       0.000000       0.000000
30118      1.142075       0.000351
19702      0.000000       0.000000
72271     26.828202      -0.031256
59833     16.931137       0.014209
31994     23.801816       0.000340

[10 rows x 191 columns]
```

Now we will be making two seprate copies of Dataframe, which is Data_2 allowing us to manipulate data1 and data2 independently without affecting the original data

[52]:
```
# Create a copy of the original DataFrame 'data_2' for data1
D1 = data_2.copy()

# Create another copy of the original DataFrame 'data_' for data2
D2 = data_2.copy()
```

This code performs various calculations to derive new features related to packet flow, byte differences, TTL, jitter, and TCP timing, and it stores these in the data_2 DataFrame. It also shows a random sample of 10 rows from the DataFrame at the end.

6 5.Modelling

Data 1 -> Without Over Sampling

Now for modeling we will use two different approaches for handling class imbalance during the modeling phase. * D1 Without Over Sampling * D2 SMOTE (Synthetic Minority Over-sampling Technique)

6.1 D1 Without Over Sampling

- This refers to using the dataset as it is, without applying any techniques to balance the classes. ## D2 SMOTE (Synthetic Minority Over-sampling Technique)
- SMOTE is a popular technique used to address class imbalance. It works by generating synthetic samples for the minority class based on the existing data points.

In a nutshell D1 serves as a reference point for assessing the model's performance on an imbalanced dataset, whereas D2 illustrates the impact of applying SMOTE to achieve balance. By comparing these two methods, one can gain insights into the influence of class imbalance on model effectiveness and determine if the creation of synthetic data enhances anomaly detection within the UNSW-NB15 dataset."

[57]:
```
# Import necessary libraries for model training and evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, RobustScaler

# Separate features and target variable from the dataset Data_1
X = D1.drop('label', axis=1)  # Features (all columns except 'label')
y = D1['label']                 # Target variable (the 'label' column)

# Define a list of scalers to normalize the feature data
scalers = [
    MinMaxScaler(),  # Scales features to a range between 0 and 1
    RobustScaler()   # Scales features using statistics that are robust to␣
 ↪outliers
]

# Install the XGBoost library if not already installed
!pip install xgboost

# Import the XGBoost library for building the model
import xgboost as xgb
```

```
Requirement already satisfied: xgboost in c:\users\braha\anaconda3\lib\site-
packages (2.1.1)
Requirement already satisfied: numpy in c:\users\braha\anaconda3\lib\site-
packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\braha\anaconda3\lib\site-
packages (from xgboost) (1.13.1)
```

[58]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
models = [
    ('KNN', KNeighborsClassifier(), {}),
    ('Random Forest', RandomForestClassifier(), {
        'n_estimators': [50, 100, 200],
```

```
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    }),
    ('XGBoost', xgb.XGBClassifier(eval_metric='mlogloss'), {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 6, 9]
    })
]
```

[59]:

```
def summarize_metrics(model_name,y_train,t_test,y_train_predict,y_test_predict,␣
↪existing_data_2=None):
    """
    This method is used to generate classification report for both train and␣
↪test and generate a dataframe that summarizes different
    models metrics, it takes the precited values for each model and scaling␣
↪method name, model name, and an existing dataframe to append
    different results
    """

    scaler_names = {
        'MinMaxScaler': 'MinMaxScaler',
        'StandardScaler': 'StandardScaler',
        'RobustScaler': 'RobustScaler',
    }

    # Determine the name of the scaler
    scaler_name = scaler_names.get(type(scaler).__name__, 'No Scaling')


    # Generating classification reports
    train_report_dict = classification_report(y_train, y_train_pred,␣
↪output_dict=True)
    test_report_dict = classification_report(y_test, y_test_pred,␣
↪output_dict=True)

    # Extracting and rounding metrics for train
    accuracy_train = round(train_report_dict['accuracy'], 2)
    macro_avg_train = train_report_dict['macro avg']
    precision_train = round(macro_avg_train['precision'], 2)
    recall_train = round(macro_avg_train['recall'], 2)
    f1_train = round(macro_avg_train['f1-score'], 2)

    # Extracting and rounding metrics for test
    accuracy_test = round(test_report_dict['accuracy'], 2)
    macro_avg_test = test_report_dict['macro avg']
    precision_test = round(macro_avg_test['precision'], 2)
```

```
    recall_test = round(macro_avg_test['recall'], 2)
    f1_test = round(macro_avg_test['f1-score'], 2)

    # Create a summary dictionary
    summary_dict = {
        'Model': model_name,
        'Scaling Method': scaler_name,
        'Train Accuracy': accuracy_train,
        'Test Accuracy': accuracy_test,
        'Train Precision': precision_train,
        'Test Precision': precision_test,
        'Train Recall': recall_train,
        'Test Recall': recall_test,
        'Train F1-Score': f1_train,
        'Test F1-Score': f1_test
    }

    summary_data_2 = pd.DataFrame([summary_dict])

    # Append to existing DataFrame or return new DataFrame
    if existing_data_2 is not None:
        return pd.concat([existing_data_2, summary_data_2], ignore_index=True)
    else:
        return summary_data_2
```

[60]:
```
from sklearn.model_selection import GridSearchCV

def tune_model(model, param_grid, X_train, y_train, X_test, y_test):
    """
    This function takes a machine learning model and a parameter grid, performs␣
 ↪hyperparameter tuning
    using GridSearchCV, and returns the best estimator based on cross-validated␣
 ↪accuracy.

    Parameters:
    - model: The machine learning model (e.g., LogisticRegression,␣
 ↪RandomForestClassifier)
    - param_grid: Dictionary with parameters to tune (e.g., {'C': [0.1, 1, 10]})
    - X_train: Training data features
    - y_train: Training data labels
    - X_test: Test data features
    - y_test: Test data labels

    Returns:
    - best_model: The best model after hyperparameter tuning
    """
```

```
    # Perform hyperparameter tuning using GridSearchCV
    # - cv=5 means 5-fold cross-validation
    # - scoring='accuracy' to evaluate models based on accuracy
    # - n_jobs=-1 means all processors will be used for parallel computation
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy',␣
 ↪n_jobs=-1)

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

    # Extract the best model (i.e., the one with the optimal hyperparameters)
    best_model = grid_search.best_estimator_

    # Make predictions using the best model on both training and testing data
    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)

    # Display the distribution of class labels in the training dataset␣
 ↪(replacing 'df' with 'data_2')
    print("Values for class:", data_2['y_train'].value_counts())  # Assuming␣
 ↪'data_2' contains the target variable

    # ============== Model Evaluation ===============

    # Calculate the accuracy of the best model on training and testing datasets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # Print out the best hyperparameters and the training/testing accuracy
    print("Best Parameters:", grid_search.best_params_)
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")

    return best_model  # Return the tuned best model
results_data_2 = None
```

[61]:
```
def tune_model(model, param_grid, X_train, y_train, X_test, y_test):
    """
    This function takes a model and a parameter grid, performs hyperparameter␣
 ↪tuning using gridsearch.
    and returns the best esimator
    """
    # Fitting the data with GridSearchCV
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy',␣
 ↪n_jobs=-1)
    grid_search.fit(X_train, y_train)
```

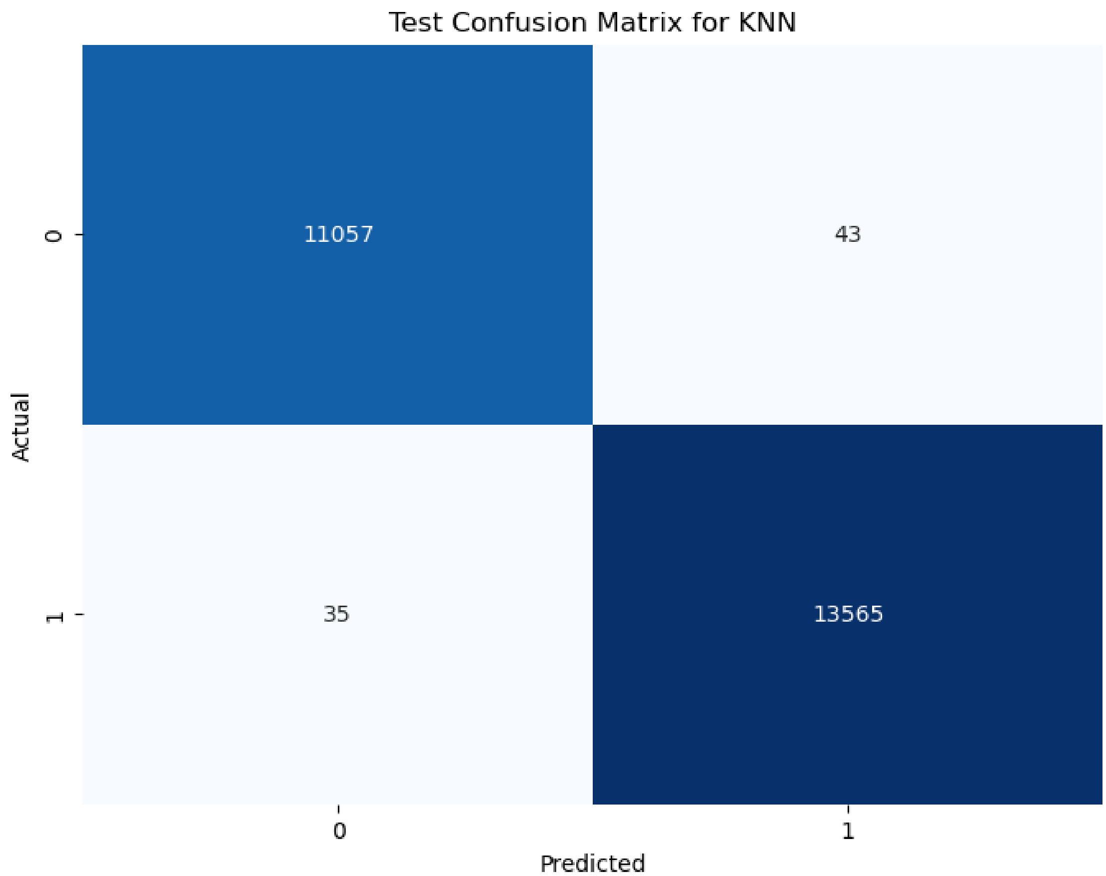
```
    best_model = grid_search.best_estimator_ # Best model

    # Predictions
    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)
    print("Values for class", y_train.value_counts())

    # ============== Evaluation ===============
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    print("Best Parameters:", grid_search.best_params_)
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")

    return best_model
```

[62]:
```
results_data_2 = None
```

[63]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

def plot_confusion_matrix(y_true, y_pred, title):
    conf_matrix = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(title)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

imputer = SimpleImputer(strategy='mean')

for scaler in scalers:
    # ========= scaling ==========
    X_scaled = scaler.fit_transform(X)
    X_scaled = imputer.fit_transform(X_scaled)
    # Train Test Split
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,␣
↪test_size=0.3, random_state=42, stratify=y)

    for model_name, model, param_grid in models:
        print(f"Running model: {model_name} with {scaler.__class__.__name__}")
```

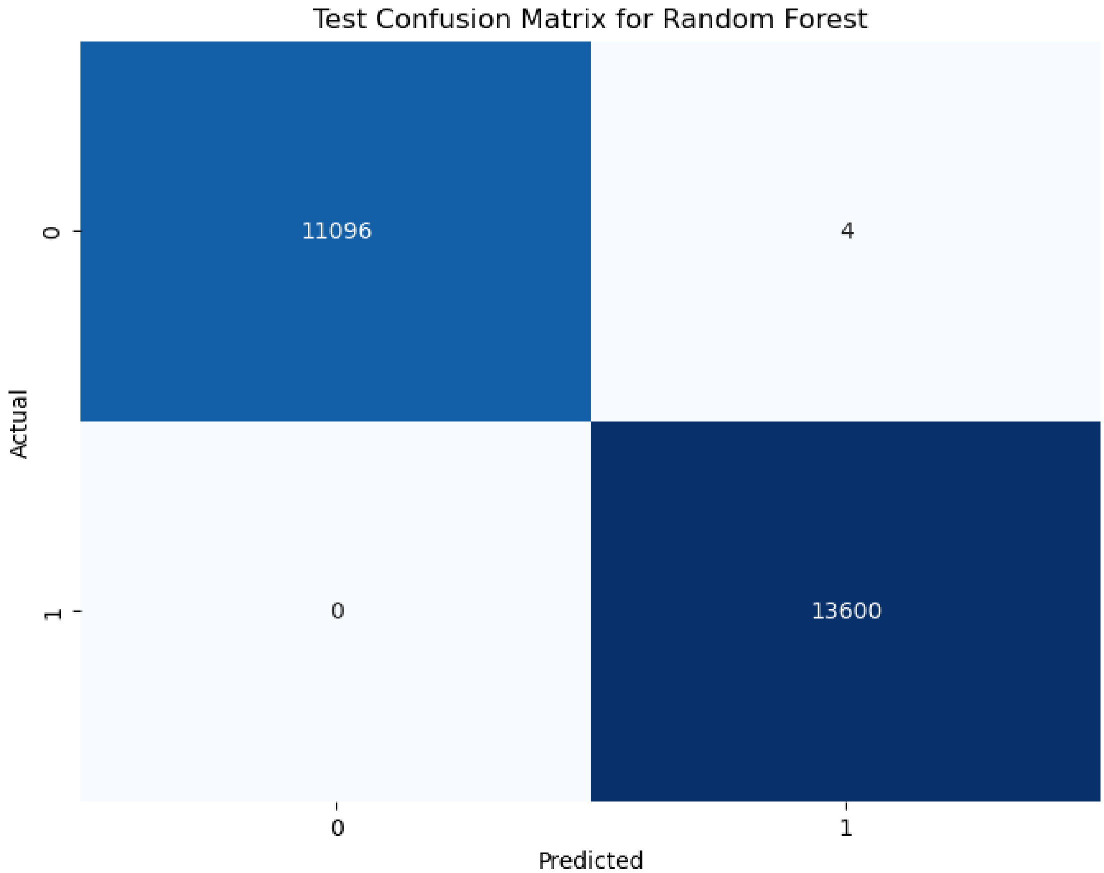
```
    # ============== Hyperparameter Tuning (Finding best parameters)␣
↪==========
    best_model = tune_model(model, param_grid, X_train, y_train, X_test,␣
↪y_test)

    # Predict using best model
    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)

    # ======== Summarize and Evaluate ==============
    results_data_2 = summarize_metrics(
        model_name,
        y_train, y_test,
        y_train_pred, y_test_pred,
        existing_data_2=results_data_2
    )
      # ======== Visualize Confusion Matrices ==============
    plot_confusion_matrix(y_test, y_test_pred, title=f"Test Confusion␣
↪Matrix for {model_name}")
    print("\n=======================================================\n")

            # ======== Print Classification Report ==============
    print(f"Classification Report for {model_name} on Test Set")
    print(classification_report(y_test, y_test_pred))

    print("\n=======================================================\n")
```

```
Running model: KNN with MinMaxScaler
Values for class label
1    31732
0    25900
Name: count, dtype: int64
Best Parameters: {}
Training Accuracy: 0.9982
Test Accuracy: 0.9968
```



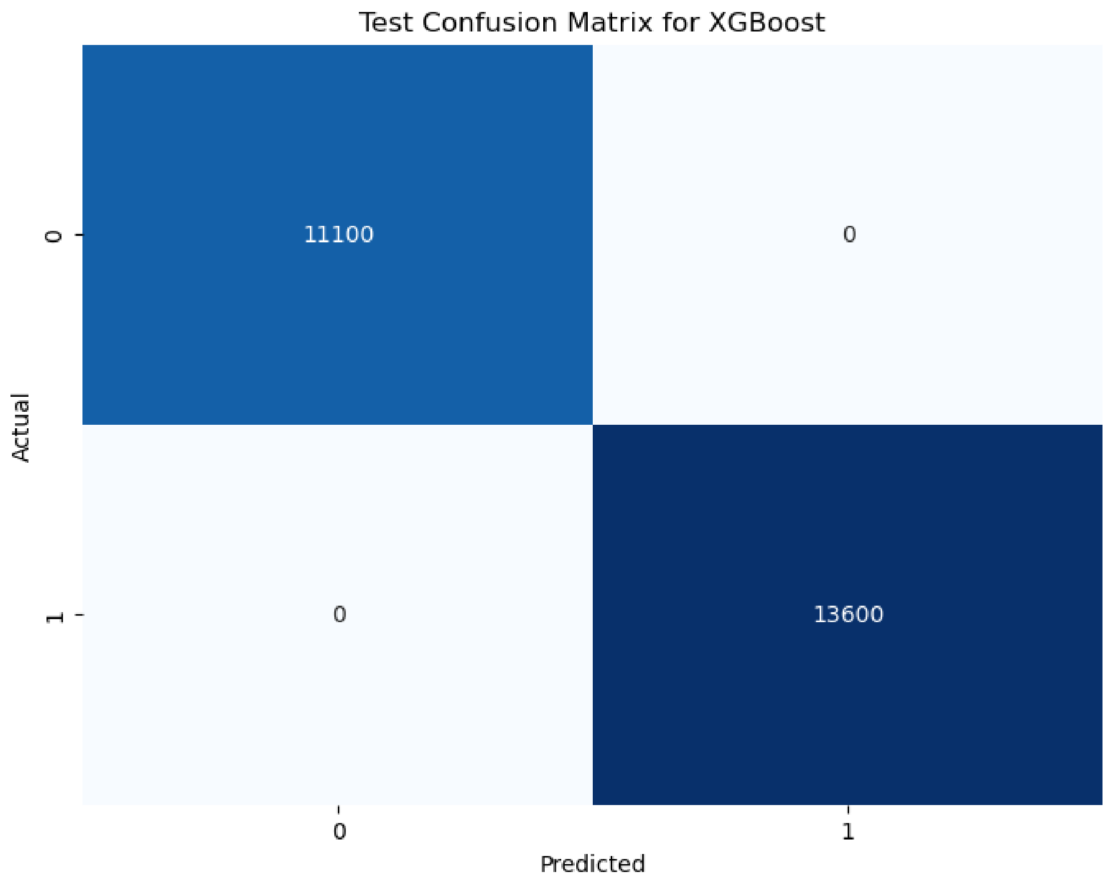

```
=======================================================

Classification Report for KNN on Test Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11100
           1       1.00      1.00      1.00     13600

    accuracy                           1.00     24700
   macro avg       1.00      1.00      1.00     24700
weighted avg       1.00      1.00      1.00     24700


=======================================================

Running model: Random Forest with MinMaxScaler
Values for class label
1    31732
0    25900
```

```
Name: count, dtype: int64
Best Parameters: {'max_depth': 30, 'min_samples_split': 5, 'n_estimators': 200}
Training Accuracy: 1.0000
Test Accuracy: 0.9998
```



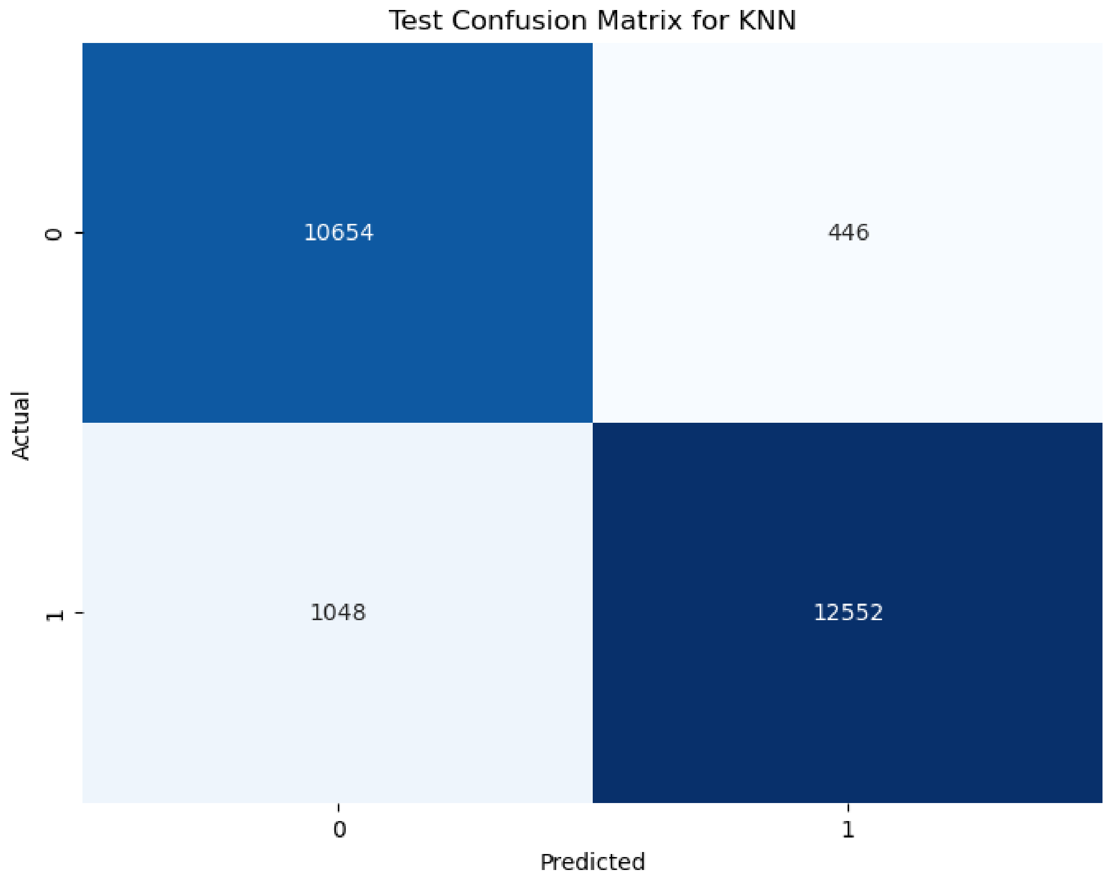

```
============================================================

Classification Report for Random Forest on Test Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11100
           1       1.00      1.00      1.00     13600

    accuracy                           1.00     24700
   macro avg       1.00      1.00      1.00     24700
weighted avg       1.00      1.00      1.00     24700


============================================================
```

```
Running model: XGBoost with MinMaxScaler
Values for class label
1    31732
0    25900
Name: count, dtype: int64
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
Training Accuracy: 1.0000
Test Accuracy: 1.0000
```



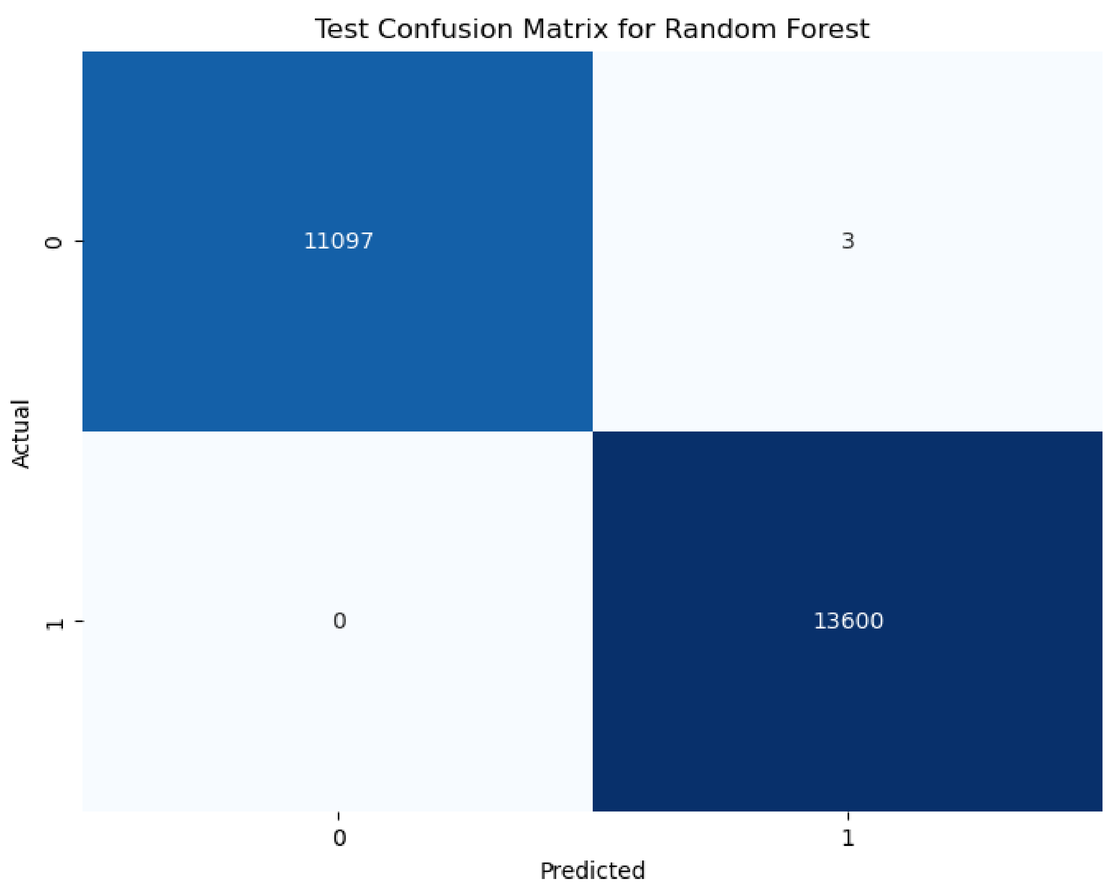

```
=======================================================

Classification Report for XGBoost on Test Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11100
           1       1.00      1.00      1.00     13600

    accuracy                           1.00     24700
```

```
   macro avg       1.00      1.00      1.00     24700
weighted avg       1.00      1.00      1.00     24700


=======================================================

Running model: KNN with RobustScaler
Values for class label
1    31732
0    25900
Name: count, dtype: int64
Best Parameters: {}
Training Accuracy: 0.9606
Test Accuracy: 0.9395
```

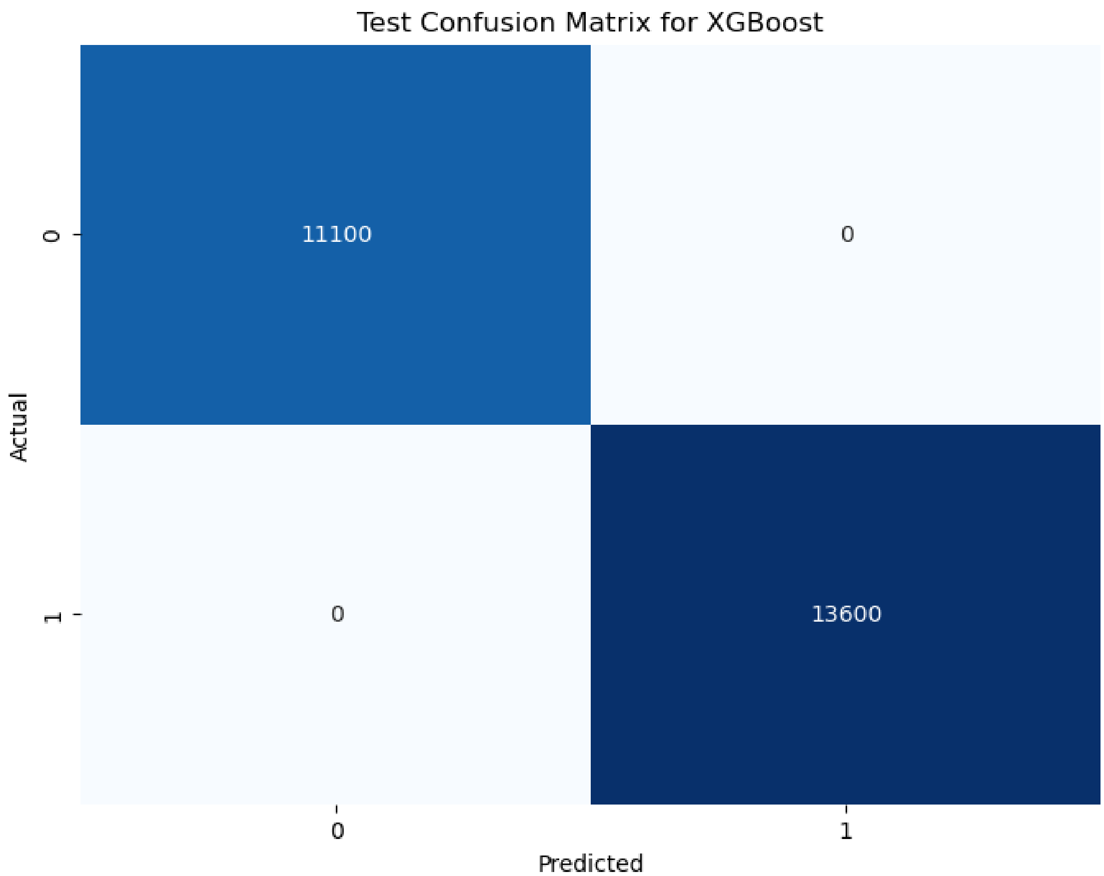



```
=======================================================

Classification Report for KNN on Test Set
              precision    recall  f1-score   support
```

```
           0       0.91      0.96      0.93     11100
           1       0.97      0.92      0.94     13600

    accuracy                           0.94     24700
   macro avg       0.94      0.94      0.94     24700
weighted avg       0.94      0.94      0.94     24700


=======================================================

Running model: Random Forest with RobustScaler
Values for class label
1    31732
0    25900
Name: count, dtype: int64
Best Parameters: {'max_depth': 30, 'min_samples_split': 2, 'n_estimators': 200}
Training Accuracy: 1.0000
Test Accuracy: 0.9999
```



```
=======================================================

Classification Report for Random Forest on Test Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11100
           1       1.00      1.00      1.00     13600

    accuracy                           1.00     24700
   macro avg       1.00      1.00      1.00     24700
weighted avg       1.00      1.00      1.00     24700


=======================================================

Running model: XGBoost with RobustScaler
Values for class label
1    31732
0    25900
Name: count, dtype: int64
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
Training Accuracy: 1.0000
Test Accuracy: 1.0000
```




```
==========================================================

Classification Report for XGBoost on Test Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11100
           1       1.00      1.00      1.00     13600

    accuracy                           1.00     24700
   macro avg       1.00      1.00      1.00     24700
weighted avg       1.00      1.00      1.00     24700


==========================================================
```

[64]:
```
results_data_2
```

[64]:

```
           Model Scaling Method  Train Accuracy  Test Accuracy  \
0            KNN   MinMaxScaler            1.00           1.00
1  Random Forest   MinMaxScaler            1.00           1.00
2        XGBoost   MinMaxScaler            1.00           1.00
3            KNN   RobustScaler            0.96           0.94
4  Random Forest   RobustScaler            1.00           1.00
5        XGBoost   RobustScaler            1.00           1.00

   Train Precision  Test Precision  Train Recall  Test Recall  Train F1-Score  \
0             1.00            1.00          1.00         1.00            1.00
1             1.00            1.00          1.00         1.00            1.00
2             1.00            1.00          1.00         1.00            1.00
3             0.96            0.94          0.96         0.94            0.96
4             1.00            1.00          1.00         1.00            1.00
5             1.00            1.00          1.00         1.00            1.00

   Test F1-Score
0           1.00
1           1.00
2           1.00
3           0.94
4           1.00
5           1.00
```

7 Data 2 -> Smote

[66]:

```python
from sklearn.model_selection import train_test_split
X = D1.drop('label', axis=1)
y = D1['label']
```

[67]:

```python
from sklearn.preprocessing import StandardScaler
scalers = [
    MinMaxScaler(),
    StandardScaler()
]
```

[68]:

```python
!pip install xgboost
import xgboost as xgb # Import the xgboost library
```

```
Requirement already satisfied: xgboost in c:\users\braha\anaconda3\lib\site-
packages (2.1.1)
Requirement already satisfied: numpy in c:\users\braha\anaconda3\lib\site-
packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\braha\anaconda3\lib\site-
packages (from xgboost) (1.13.1)
```

[69]:
```python
models = [
    # A tuple representing the K-Nearest Neighbors (KNN) model with no␣
↪hyperparameters to tune
    ('KNN', KNeighborsClassifier(), {}),

    # A tuple representing the Random Forest model with hyperparameters to tune:
    # 'n_estimators': Number of trees in the forest
    # 'max_depth': Maximum depth of the trees
    # 'min_samples_split': Minimum number of samples required to split an␣
↪internal node
    ('Random Forest', RandomForestClassifier(), {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    }),

    # A tuple representing the XGBoost model with hyperparameters to tune:
    # 'n_estimators': Number of boosting rounds
    # 'learning_rate': Step size shrinkage used in update to prevent overfitting
    # 'max_depth': Maximum depth of a tree
    ('XGBoost', xgb.XGBClassifier(eval_metric='mlogloss'), {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 6, 9]
    })
]
```

7.0.1 Key Points:

1. **GridSearchCV**:
   - Used to perform an exhaustive search over a parameter grid with cross-validation.
   - Finds the best combination of hyperparameters for the model.
2. **Predictions**:
   - After finding the best model, predictions are made on both the training and test sets.
   - These predictions are used for evaluating model performance.
3. **Evaluation**:
   - The function calculates and prints the accuracy for both the training and test sets.
   - Outputs the best hyperparameters and returns the best model found during the search. both training and test sets and returns the best model.

[71]:
```python
def summarize_metrics(model_name,y_train,t_test,y_train_predict,y_test_predict,␣
↪existing_data_2=None):
    """
    This method is used to generate classification report for both train and␣
↪test and generate a dataframe that summarizes different
    models metrics, it takes the precited values for each model and scaling␣
↪method name, model name, and an existing dataframe to append
```

```
    different results
    """

    scaler_names = {
        'MinMaxScaler': 'MinMaxScaler',
        'StandardScaler': 'StandardScaler',
        'RobustScaler': 'RobustScaler',
    }
    # Determine the name of the scaler
    scaler_name = scaler_names.get(type(scaler).__name__, 'No Scaling')


    # Generating classification reports
    train_report_dict = classification_report(y_train, y_train_pred,␣
↪output_dict=True)
    test_report_dict = classification_report(y_test, y_test_pred,␣
↪output_dict=True)

    # Extracting and rounding metrics for train
    accuracy_train = round(train_report_dict['accuracy'], 2)
    macro_avg_train = train_report_dict['macro avg']
    precision_train = round(macro_avg_train['precision'], 2)
    recall_train = round(macro_avg_train['recall'], 2)
    f1_train = round(macro_avg_train['f1-score'], 2)
    # Extracting and rounding metrics for test
    accuracy_test = round(test_report_dict['accuracy'], 2)
    macro_avg_test = test_report_dict['macro avg']
    precision_test = round(macro_avg_test['precision'], 2)
    recall_test = round(macro_avg_test['recall'], 2)
    f1_test = round(macro_avg_test['f1-score'], 2)

    # Create a summary dictionary
    summary_dict = {
        'Model': model_name,
        'Scaling Method': scaler_name,
        'Train Accuracy': accuracy_train,
        'Test Accuracy': accuracy_test,
        'Train Precision': precision_train,
        'Test Precision': precision_test,
        'Train Recall': recall_train,
        'Test Recall': recall_test,
        'Train F1-Score': f1_train,
        'Test F1-Score': f1_test
    }

    summary_data_2 = pd.DataFrame([summary_dict])
```

```python
    # Append to existing DataFrame or return new DataFrame
    if existing_data_2 is not None:
        return pd.concat([existing_data_2, summary_data_2], ignore_index=True)
    else:
        return summary_data_2
```

7.0.2 Summary of the summarize_metrics Function

The summarize_metrics function generates performance metrics for a classification model, including accuracy, precision, recall, and F1-score for both the training and test datasets.

7.0.3 Key Features:

1. **Scalability Detection**: Automatically identifies and records the scaling method used (e.g., MinMaxScaler, StandardScaler). If no scaler is applied, it marks the result as 'No Scaling'.
2. **Metrics Calculation**: Uses the classification_report function from sklearn to extract and round key metrics for both training and test sets:
   - Accuracy
   - Precision
   - Recall
   - F1-Score
3. **data_2 DataFrame Output**:
   - The function creates a summary of the model's performance metrics and appends it to an existing data_2 DataFrame.
   - If no existing data_2 DataFrame is provided, the function generates a new one containing the summarized metrics.
4. **Return Value**: Returns a DataFrame with the following columns:
   - Model name
   - Scaling method used
   - Train and test accuracy
   - Train and test precision
   - Train and test recall
   - Train and test F1-score

[73]:
```python
def tune_model(model, param_grid, X_train, y_train, X_test, y_test):
    """
    This function takes a model and a parameter grid, performs hyperparameter␣
    ↪tuning using GridSearchCV,
    and returns the best estimator.
    """

    # ============== Hyperparameter Tuning ==============
    # Perform grid search with cross-validation (cv=5) on the model to find the␣
    ↪best combination of hyperparameters.
    # 'param_grid' contains the hyperparameter space to search.
    # 'scoring=accuracy' evaluates models based on accuracy.
```

```
    # 'n_jobs=-1' allows the use of all available CPU cores for parallel␣
↪processing.
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy',␣
↪n_jobs=-1)

    # Fit the model on the training data using GridSearchCV to search for the␣
↪best parameters.
    grid_search.fit(X_train, y_train)

    # Extract the best model found during grid search based on accuracy score.
    best_model = grid_search.best_estimator_

    # ============== Predictions ==============
    # Use the best model to predict on the training data.
    y_train_pred = best_model.predict(X_train)

    # Use the best model to predict on the test data.
    y_test_pred = best_model.predict(X_test)

    # Display the distribution of classes in the training set (useful for␣
↪classification tasks).
    print("Values for class", y_train.value_counts())

    # ============== Evaluation ==============
    # Compute the accuracy of the model on the training data.
    train_accuracy = accuracy_score(y_train, y_train_pred)

    # Compute the accuracy of the model on the test data.
    test_accuracy = accuracy_score(y_test, y_test_pred)

    # Output the best hyperparameters found during the grid search.
    print("Best Parameters:", grid_search.best_params_)

    # Print the accuracy of the best model on the training set.
    print(f"Training Accuracy: {train_accuracy:.4f}")

    # Print the accuracy of the best model on the test set.
    print(f"Test Accuracy: {test_accuracy:.4f}")

    # Return the best model from the grid search.
    return best_model
```

7.0.4 Summary of the `tune_model` Function:

The `tune_model` function performs hyperparameter tuning on a given machine learning model using `GridSearchCV`. It searches for the best combination of hyperparameters, evaluates the model's performance on both training and test datasets, and returns the best-performing model.

7.0.5 Key Features:

1. **Hyperparameter Tuning**:
   - Utilizes `GridSearchCV` to perform an exhaustive search over the provided hyperparameter grid (`param_grid`) with 5-fold cross-validation.
   - Automatically selects the model with the best performance based on accuracy.
2. **Predictions**:
   - After finding the best model, predictions are made on both the training and test sets.
3. **Evaluation**:
   - The function computes and prints the accuracy for both the training and test datasets.
   - Outputs the best hyperparameters found during the tuning process.
4. **Return Value**:
   - Returns the best model (`best_model`) found during the grid search.

7.0.6 Usage:

This function is useful for automating the hyperparameter tuning process, making it easy to identify the optimal model configuration and evaluate its performance on different datasets.

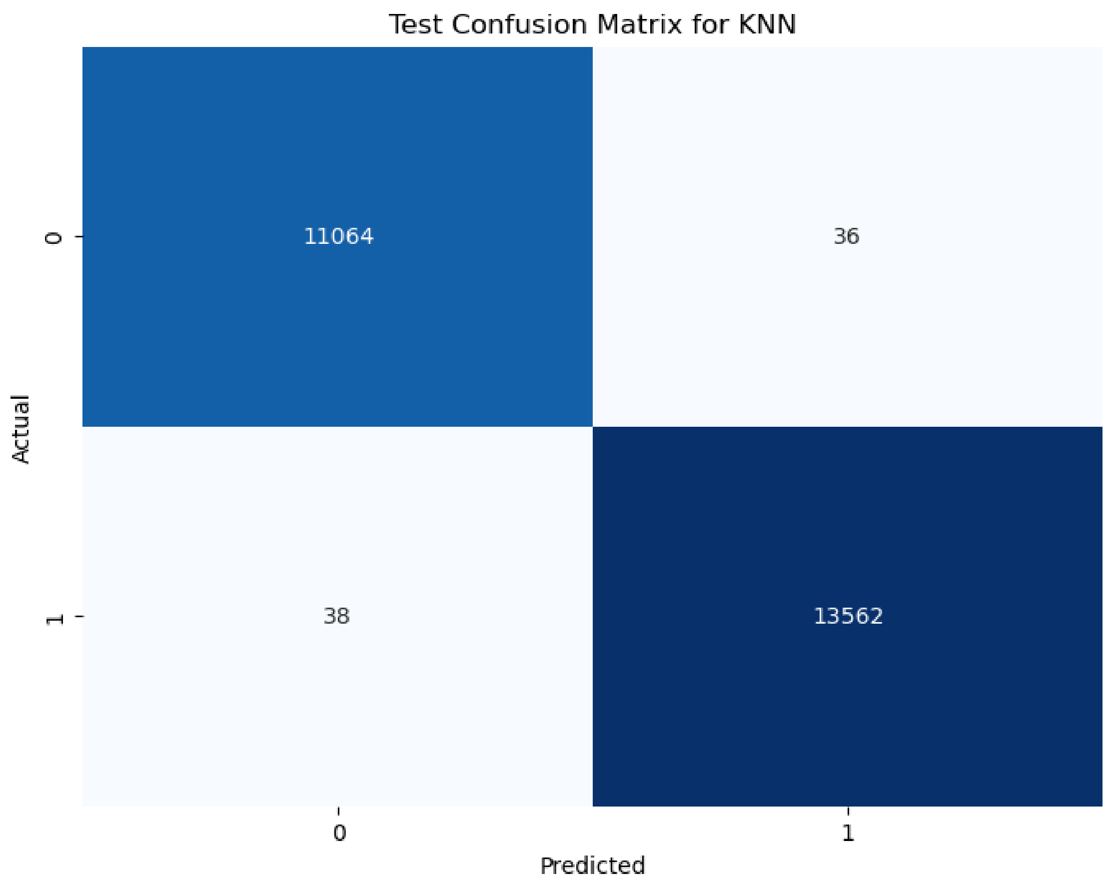
[75]:
```
results_data_2 = None
```

[76]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE

def plot_confusion_matrix(y_true, y_pred, title):
    """
    Plots a confusion matrix using the true and predicted values.

    Parameters:
    - y_true: True labels.
    - y_pred: Predicted labels.
    - title: Title for the plot.
    """
    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_true, y_pred)

    # Set up the figure for the heatmap
    plt.figure(figsize=(8, 6))

    # Create a heatmap for the confusion matrix
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(title)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
```

```
    plt.show()

    # Initialize a SimpleImputer to fill missing values (if any)
    imputer = SimpleImputer(strategy='mean')

# Loop through different scalers
for scaler in scalers:
    # ========= Scaling ==========
    # Apply the scaler to the features and impute any missing values
    X_scaled = scaler.fit_transform(X)
    X_scaled = imputer.fit_transform(X_scaled)

    # Split the scaled data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,␣
↪test_size=0.3, random_state=42, stratify=y)

    # Initialize SMOTE to handle class imbalance
    smote = SMOTE(random_state=42)

    # Fit SMOTE to the training data and resample
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    # Loop through each model and its corresponding hyperparameter grid
    for model_name, model, param_grid in models:
        print(f"Running model: {model_name} with {scaler.__class__.__name__}")

        # ============== Hyperparameter Tuning (Finding best parameters)␣
↪==========
        # Tune the model using the training data
        best_model = tune_model(model, param_grid, X_train_resampled,␣
↪y_train_resampled, X_test, y_test)

        # Make predictions using the best model
        y_train_pred = best_model.predict(X_train_resampled)
        y_test_pred = best_model.predict(X_test)

        # ======== Summarize and Evaluate ==============
        # Summarize the model's metrics and append to results
        results_data_2 = summarize_metrics(
            model_name,
            y_train_resampled, y_test,
            y_train_pred, y_test_pred,
            existing_data_2 = results_data_2
        )

        # ======== Visualize Confusion Matrices =============
        # Plot the confusion matrix for the test set predictions
```

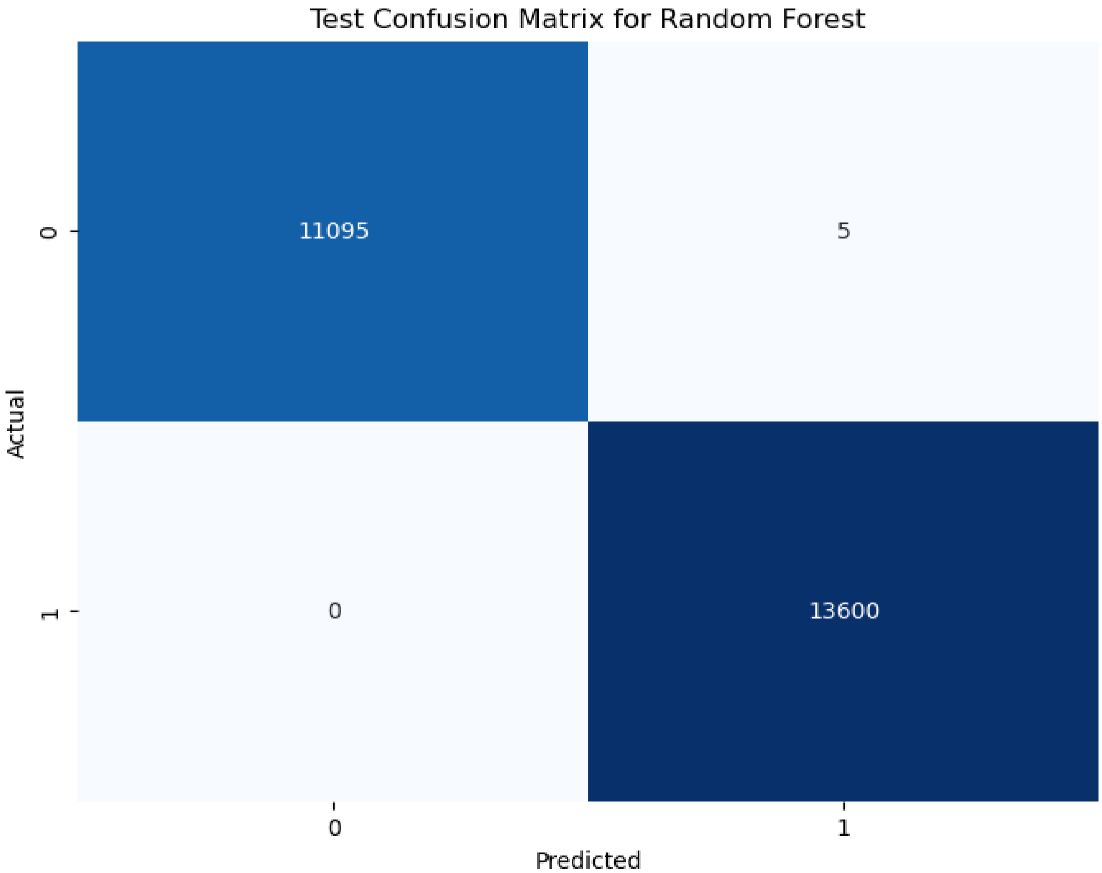
```
        plot_confusion_matrix(y_test, y_test_pred, title=f"Test Confusion␣
↪Matrix for {model_name}")
        print("\n========================================================\n")
```

```
Running model: KNN with MinMaxScaler
Values for class label
0    31732
1    31732
Name: count, dtype: int64
Best Parameters: {}
Training Accuracy: 0.9984
Test Accuracy: 0.9970
```

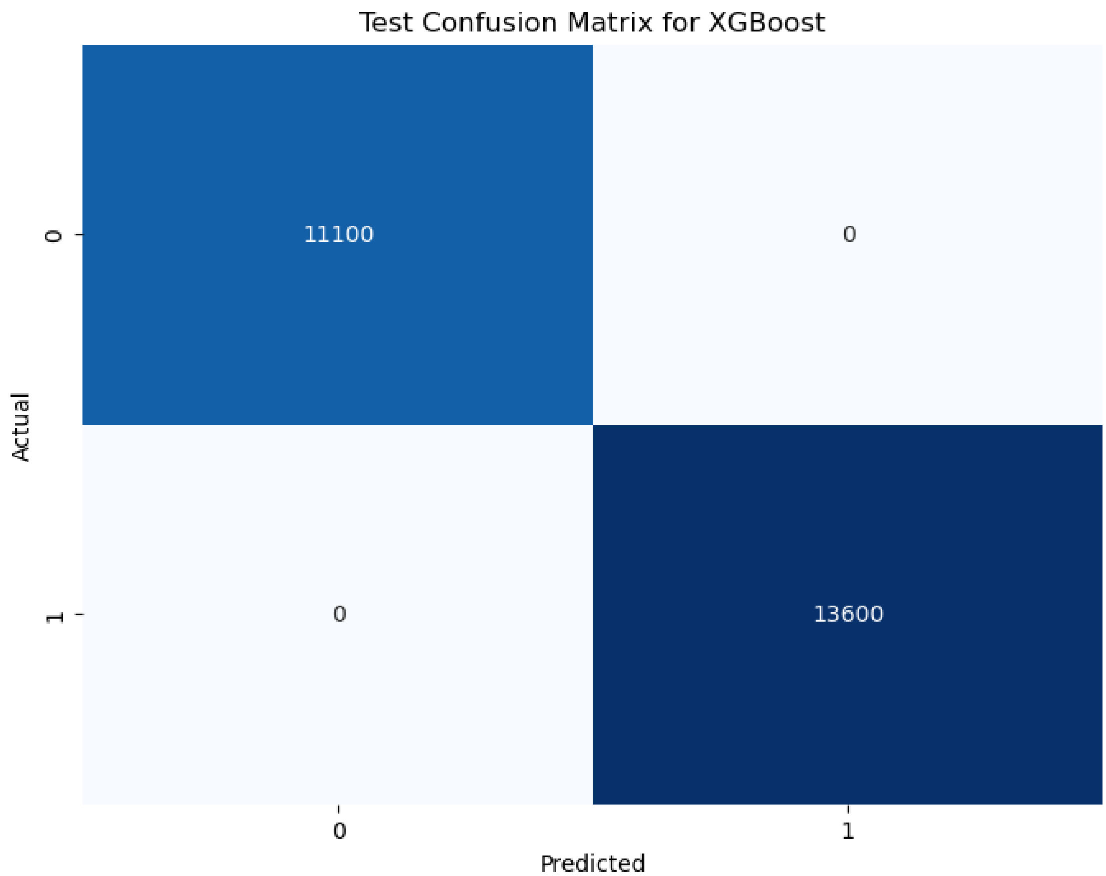



```
========================================================

Running model: Random Forest with MinMaxScaler
Values for class label
0    31732
1    31732
```

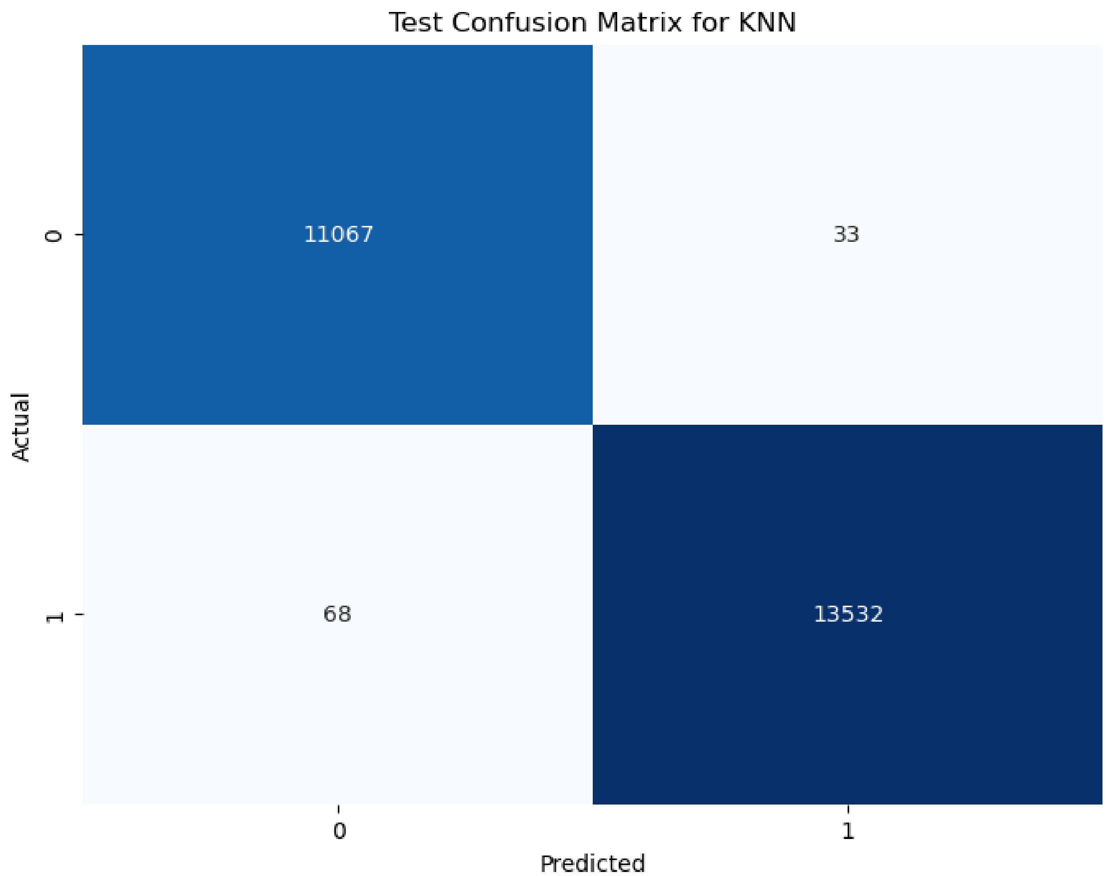
```
Name: count, dtype: int64
Best Parameters: {'max_depth': 20, 'min_samples_split': 10, 'n_estimators': 100}
Training Accuracy: 1.0000
Test Accuracy: 0.9998
```



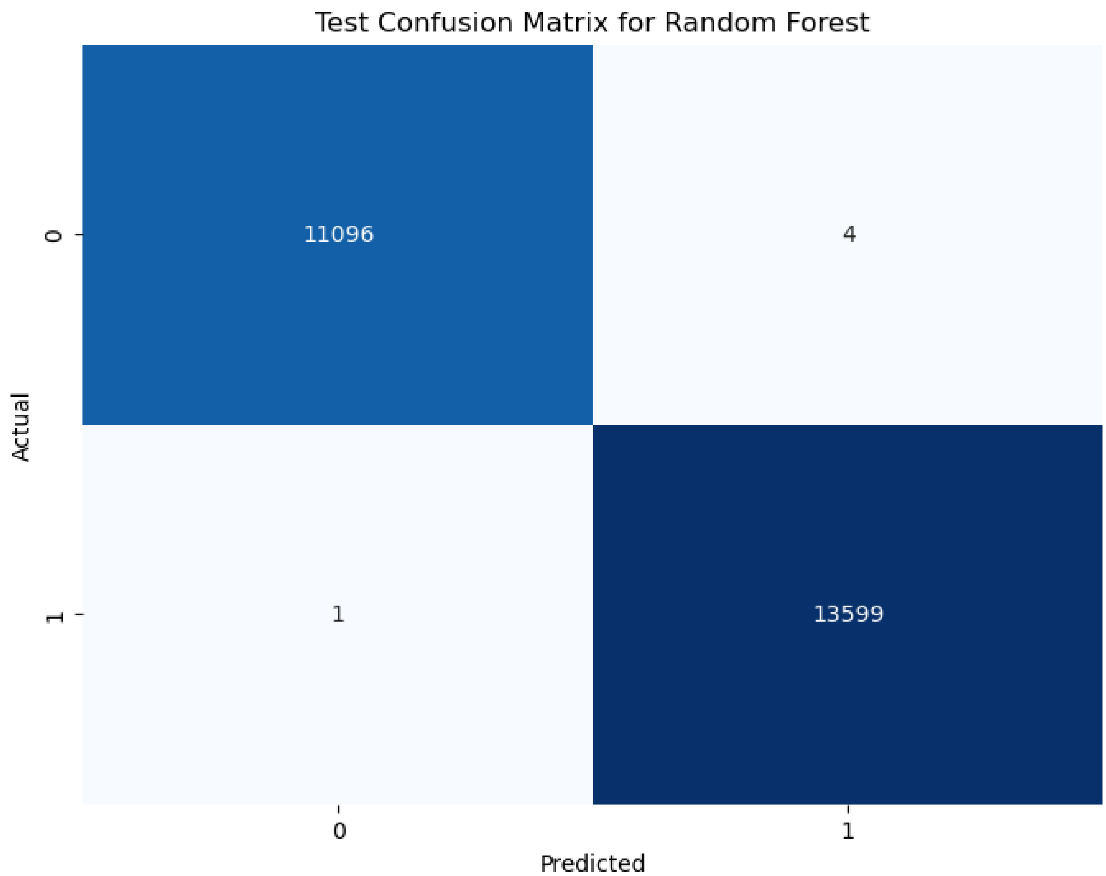

```
========================================================

Running model: XGBoost with MinMaxScaler
Values for class label
0    31732
1    31732
Name: count, dtype: int64
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
Training Accuracy: 1.0000
Test Accuracy: 1.0000
```



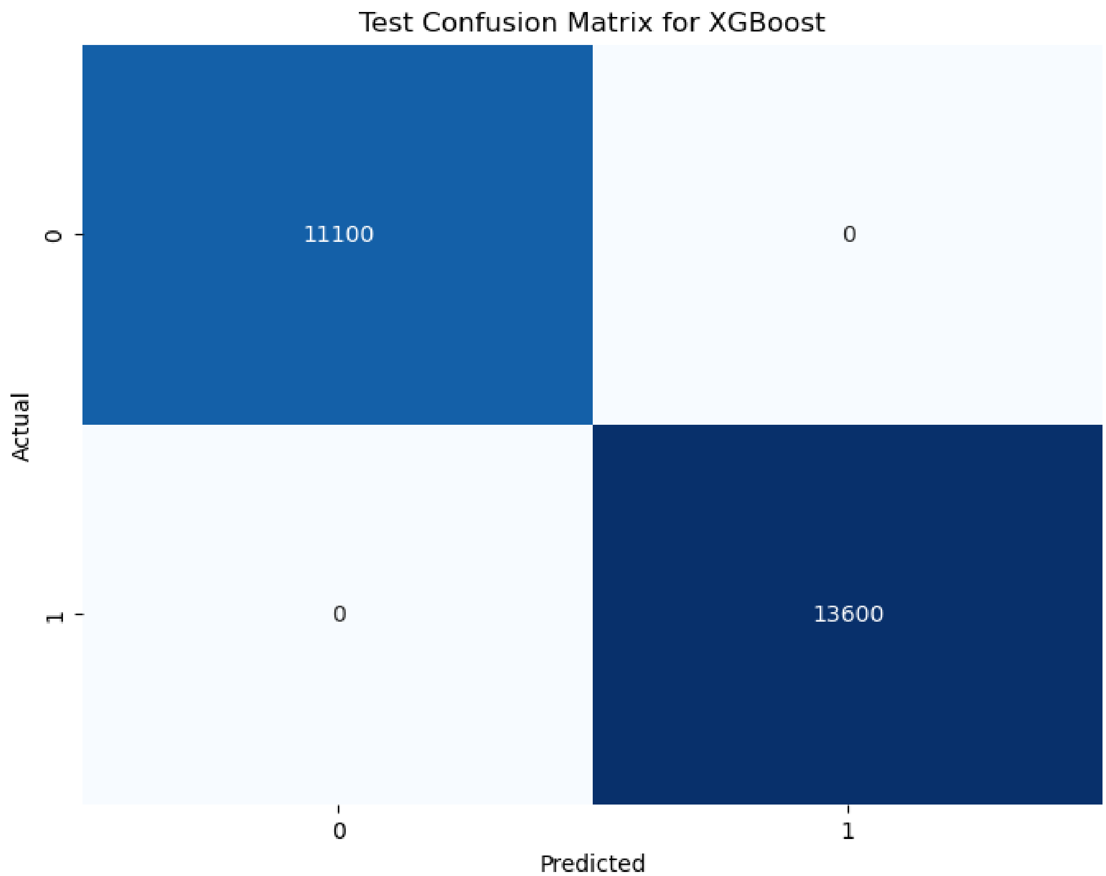

```
=======================================================

Running model: KNN with StandardScaler
Values for class label
0    31732
1    31732
Name: count, dtype: int64
Best Parameters: {}
Training Accuracy: 0.9974
Test Accuracy: 0.9959
```




```
============================================================

Running model: Random Forest with StandardScaler
Values for class label
0    31732
1    31732
Name: count, dtype: int64
Best Parameters: {'max_depth': 30, 'min_samples_split': 5, 'n_estimators': 200}
Training Accuracy: 1.0000
Test Accuracy: 0.9998
```




```
==========================================================

Running model: XGBoost with StandardScaler
Values for class label
0    31732
1    31732
Name: count, dtype: int64
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
Training Accuracy: 1.0000
Test Accuracy: 1.0000
```



==

7.0.7 Summary of the Model Training and Evaluation Process

The provided code implements a systematic approach for training and evaluating multiple machine learning models using various data preprocessing techniques. Key components of the process include:

1. **Imports**: Essential libraries for data visualization (Matplotlib and Seaborn), model evaluation (scikit-learn), data imputation, and handling class imbalance (SMOTE) are imported.

2. **Confusion Matrix Plotting**: The `plot_confusion_matrix` function visualizes the confusion matrix, which compares true labels against predicted labels to assess model performance.

3. **Data Preprocessing Loop**:

    - **Scaling**: The code iterates through a list of scalers, applying each to the feature dataset (`X`) and imputing any missing values using a `SimpleImputer`.
    - **Train-Test Split**: The scaled data is split into training and test sets, maintaining the proportion of target classes using stratified sampling.

4. **Handling Class Imbalance**:
   - **SMOTE**: The SMOTE technique is applied to the training data to generate synthetic samples, thus addressing any class imbalance issues.
5. **Model Training Loop**:
   - The code iterates over a list of models and their corresponding hyperparameter grids. For each model:
     - **Hyperparameter Tuning**: The `tune_model` function is called to find the best hyperparameters for the model using the resampled training data.
     - **Predictions**: Predictions are made for both the training and test datasets using the best model.
6. **Metrics Summarization**:
   - The `summarize_metrics` function is utilized to compile key performance metrics (accuracy, precision, recall, F1-score) for each model and append the results to an existing DataFrame (`results_data_2`).
7. **Visualization of Results**:
   - Finally, confusion matrices for the test predictions are plotted to visually represent the model's performance.

This systematic process ensures that multiple models are evaluated comprehensively, facilitating comparisons of their performance based on various scaling methods and handling of class imbalances.

[78]:
```
results_data_2
```

[78]:
```
           Model  Scaling Method  Train Accuracy  Test Accuracy  \
0            KNN    MinMaxScaler             1.0            1.0
1  Random Forest    MinMaxScaler             1.0            1.0
2        XGBoost    MinMaxScaler             1.0            1.0
3            KNN  StandardScaler             1.0            1.0
4  Random Forest  StandardScaler             1.0            1.0
5        XGBoost  StandardScaler             1.0            1.0

   Train Precision  Test Precision  Train Recall  Test Recall  Train F1-Score  \
0              1.0             1.0           1.0          1.0             1.0
1              1.0             1.0           1.0          1.0             1.0
2              1.0             1.0           1.0          1.0             1.0
3              1.0             1.0           1.0          1.0             1.0
4              1.0             1.0           1.0          1.0             1.0
5              1.0             1.0           1.0          1.0             1.0

   Test F1-Score
0            1.0
1            1.0
2            1.0
```

```
3            1.0
4            1.0
5            1.0
```

7.1 6.User behavior

[176]:
```python
userdata = pd.read_csv('UNSW_NB15_training-set.csv', encoding='cp1252')
userdata
```

[176]:
```
         ï»¿id       dur proto service state  spkts  dpkts  sbytes  dbytes  \
0            1  0.000011   udp       -   INT      2      0     496       0
1            2  0.000008   udp       -   INT      2      0    1762       0
2            3  0.000005   udp       -   INT      2      0    1068       0
3            4  0.000006   udp       -   INT      2      0     900       0
4            5  0.000010   udp       -   INT      2      0    2126       0
…          …         …   …     …    …      …      …     …     …
82327    82328  0.000005   udp       -   INT      2      0     104       0
82328    82329  1.106101   tcp       -   FIN     20      8   18062     354
82329    82330  0.000000   arp       -   INT      1      0      46       0
82330    82331  0.000000   arp       -   INT      1      0      46       0
82331    82332  0.000009   udp       -   INT      2      0     104       0

                rate  …  ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  \
0       90909.090200  …                 1               2             0
1      125000.000300  …                 1               2             0
2      200000.005100  …                 1               3             0
3      166666.660800  …                 1               3             0
4      100000.002500  …                 1               3             0
…                …  …               …             …           …
82327  200000.005100  …                 1               2             0
82328      24.410067  …                 1               1             0
82329       0.000000  …                 1               1             0
82330       0.000000  …                 1               1             0
82331  111111.107200  …                 1               1             0

       ct_ftp_cmd  ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  \
0               0                 0           1           2                0
1               0                 0           1           2                0
2               0                 0           1           3                0
3               0                 0           2           3                0
4               0                 0           2           3                0
…             …                …          …          …               …
82327           0                 0           2           1                0
82328           0                 0           3           2                0
82329           0                 0           1           1                1
82330           0                 0           1           1                1
82331           0                 0           1           1                0
```

```
       attack_cat  label
0          Normal      0
1          Normal      0
2          Normal      0
3          Normal      0
4          Normal      0
…             …     …
82327      Normal      0
82328      Normal      0
82329      Normal      0
82330      Normal      0
82331      Normal      0

[82332 rows x 45 columns]
```

7.1.1 we are using UNSW_NB15_training data set because it contain the relevent features for user behavior analysis

[178]:
```python
print(userdata.columns)
```

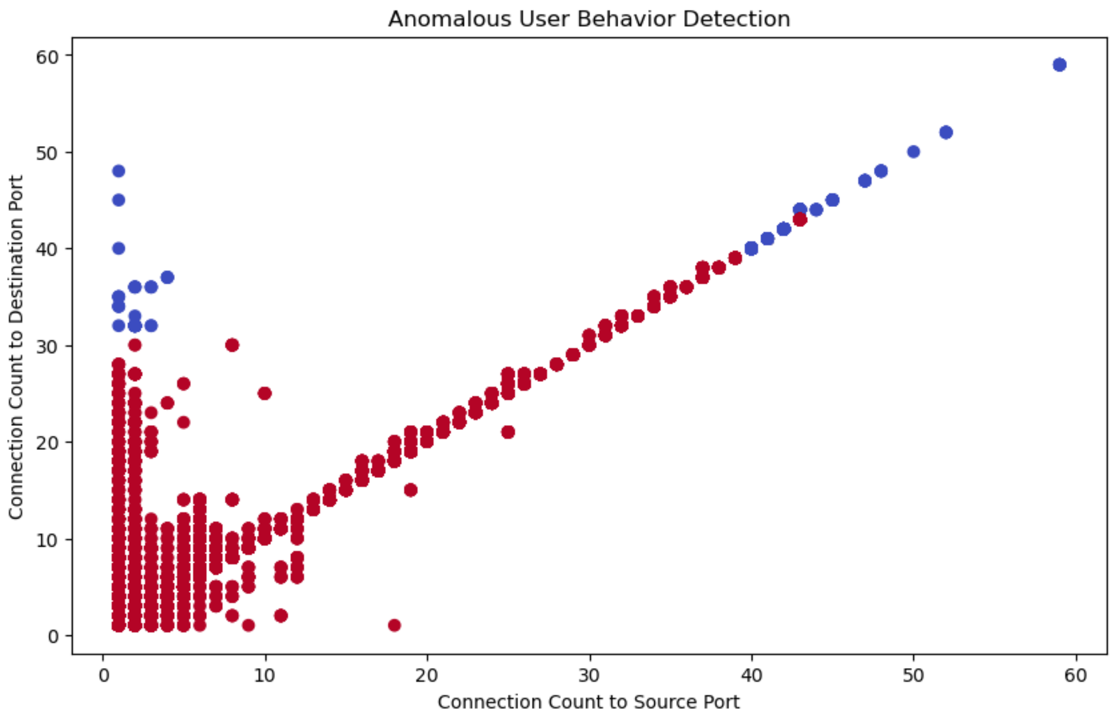
```
Index(['ï»¿id', 'dur', 'proto', 'service', 'state', 'spkts', 'dpkts', 'sbytes',
       'dbytes', 'rate', 'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss',
       'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb', 'dwin',
       'tcprtt', 'synack', 'ackdat', 'smean', 'dmean', 'trans_depth',
       'response_body_len', 'ct_srv_src', 'ct_state_ttl', 'ct_dst_ltm',
       'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',
       'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'ct_src_ltm',
       'ct_srv_dst', 'is_sm_ips_ports', 'attack_cat', 'label'],
      dtype='object')
```

[190]:
```python
# Selecting relevant features for user behavior analysis
features = [ 'ct_src_dport_ltm', 'ct_dst_ltm']
userdata = userdata[features]
```

- **`ct_src_dport_ltm` (Source Port Connection Count)**: This feature tracks how often a source port (from which a user's device sends data) is accessed within a time window.
  - *Use Case*: If a user initiates connections to many source ports quickly, it could indicate abnormal behavior such as network scanning or malicious activities like Distributed Denial-of-Service (DDoS) attacks.
- **`ct_dst_ltm` (Destination Port Connection Count)**: This tracks the number of connections to destination ports over a given time window.
  - *Use Case*: An unusually high number of connections to certain destination ports may indicate suspicious activity, such as attempts to exploit vulnerabilities or perform brute-force attacks.

[192]:
```python
# Handle missing values and normalize data
userdata = userdata.fillna(0)
```

```python
scaler = StandardScaler()
userdata_scaled = scaler.fit_transform(userdata[features])

# Train an Isolation Forest for anomaly detection
model = IsolationForest(contamination=0.01)  # Assume 1% of data is anomalous
model.fit(userdata_scaled)

# Predict anomalies
userdata['anomaly'] = model.predict(userdata_scaled)

# Plot results
plt.figure(figsize=(10, 6))
plt.scatter(userdata['ct_src_dport_ltm'], userdata['ct_dst_ltm'],
    c=userdata['anomaly'], cmap='coolwarm')
plt.xlabel('Connection Count to Source Port')
plt.ylabel('Connection Count to Destination Port')
plt.title('Anomalous User Behavior Detection')
plt.show()
```



8 Visualization of Connection Counts (`ct_src_dport_ltm` vs `ct_dst_ltm`)

We are visualizing the connection counts to source ports (`ct_src_dport_ltm`) against the connection counts to destination ports (`ct_dst_ltm`) to detect user behavior anomalies. Here's a breakdown of the key insights:

8.1 1. Red and Blue Points:

- **Red Points**: These represent "normal" data points as identified by the Isolation Forest model.
- **Blue Points**: These are "anomalies," flagged as outliers by the model.

8.2 2. Concentration of Red Points:

- The majority of red points, representing normal behavior, are clustered in the lower ranges of both source and destination port connection counts, particularly where the values are below 10.
- There seems to be a **linear relationship** between `ct_src_dport_ltm` and `ct_dst_ltm` for normal points, indicating that as the number of connections to source ports increases, the number of connections to destination ports increases as well.

8.3 3. Anomalies (Blue Points):

- Blue points are scattered, primarily at **higher values** of both source and destination connection counts, particularly in regions with fewer normal data points.
- These anomalies occur in regions where connection patterns deviate from the majority, suggesting abnormal or unexpected behavior.
- Some anomalies occur at **low values of `ct_src_dport_ltm` but high values of `ct_dst_ltm`**, indicating that even with a relatively low number of source port connections, there are abnormally high destination port connections.

8.4 Interpretation:

- **Normal Behavior**: Most users or devices show a balanced, relatively low number of connections to both source and destination ports, which is typical in normal network traffic.
- **Anomalous Behavior**: Users or devices classified as anomalies exhibit abnormal connection patterns. This could include unusually high connection counts, which may signal a potential security threat such as:
  - **DDoS attacks**: Large volumes of traffic directed toward specific ports.
  - **Port Scanning**: Excessive probing of network ports by potential attackers.
  - **Other Suspicious Network Activities**.

These anomalies should be further investigated to determine if they correspond to specific types of malicious activities within the dataset.