Project work on Network Intrusion Detection. The tools used in generating this injected dataset are; Sqlite, Zenmap Nmap, ID2T and Python.The ID2T toolkit targets the injection of attacks into existing network datasets. First, it analyzes a given dataset and collects statistics from it. These statistics are stored in a local database (Sqlite). Next, these statistics can be used to define attack parameters for the injection of one or multiple attacks. Finally, the application creates the required attack packets and injects them into the existing file. Resulting in a new PCAP with the injected attacks and a label file indicating the position (timestamps) of the first and last attack packet. Nping is a multifunctional tool, perfect for generating RAW packages. It has an "echo mode" that enables advanced detection and troubleshooting. Echo mode allows both the destination and source computers to see how network packets change during transmission.

Basically, this mode splits nping into its two components: echo server and echo client. An echo server is a network service for capturing packets and echoing them over a side channel to the originating client. The Echo client takes over generating packets and sending them to the server. This element is also responsible for receiving the echo version. I like echo mode because it perfectly understands the difference between sending and receiving packets.

In [3]:
```
## importing the necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import sklearn
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import pycountry_convert as pc
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
import xgboost
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from xgboost import XGBRegressor
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
%matplotlib inline
```

In [5]:
```
#Reading the loaded files
data_1=pd.read_csv('network_dataset_1.csv')
data_2=pd.read_csv('network_dataset_2.csv')
data_3=pd.read_csv('network_dataset_3.csv')
data_4=pd.read_csv('network_dataset_4.csv')
data_5=pd.read_csv('network_dataset_5.csv')
data_6=pd.read_excel('network_dataset_6.xlsx')
```

In [6]:
```
#joining all 6 files to become one
data=pd.concat([data_1,data_2,data_3,data_4,data_5,data_6],axis=0)
data.shape
```

Out[6]:
```
(94200, 30)
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 94200 entries, 0 to 4999
Data columns (total 30 columns):
 #   Column                           Non-Null Count  Dtype 
---  ------                           --------------  ----- 
 0   Source IP                        94200 non-null  object
 1   Destination IP                   94200 non-null  object
 2   Source Port                      94200 non-null  int64 
 3   Destination Port                 94200 non-null  int64 
 4   Protocol                         94200 non-null  object
 5   Packet Size                      94200 non-null  int64 
 6   Timestamp                        94200 non-null  object
 7   pktsSent                         94200 non-null  int64 
 8   kbytesSent                       94200 non-null  int64 
 9   kbytesReceived                   94200 non-null  int64 
 10  TTL (Time to Live) Value         94200 non-null  int64 
 11  Flag                             94200 non-null  object
 12  VLAN ID                          94200 non-null  int64 
 13  QoS (Quality of Service)         94200 non-null  object
 14  AS (Autonomous System) Number    94200 non-null  int64 
 15  Geolocation                      94200 non-null  object
 16  Application                      94200 non-null  object
 17  Threat Score                     94200 non-null  int64 
 18  Payload                          94200 non-null  object
 19  Packet ID                        94200 non-null  int64 
 20  Time to Live (TTL)               94200 non-null  int64 
 21  Quality of Service (QoS) Class   94200 non-null  object
 22  Fragmentation                    94200 non-null  bool  
 23  Type of Service (ToS)            94200 non-null  int64 
 24  Hop Count                        94200 non-null  int64 
 25  Error Codes                      94200 non-null  int64 
 26  Flow ID                          94200 non-null  object
 27  Routing Information              94200 non-null  object
 28  Packet Capture Timestamp         94200 non-null  object
 29  Attack Type                      94200 non-null  object
dtypes: bool(1), int64(15), object(14)
memory usage: 21.7+ MB
```

In [8]:
```
# to get columns with missing values
missing_values = data.isna().sum()
missing_columns = missing_values[missing_values > 1]
```

```python
# print the columns with missing values
if not missing_columns.empty:
    print("Columns with missing values:")
    print(missing_columns)
else:
    print("No columns have more than one missing value.")
```

```
No columns have more than one missing value.
```

In [9]:
```python
data.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```python
data.head()
```

Out[10]:

	Source IP	Destination IP	Source Port	Destination Port	Protocol	Packet Size	Timestamp	pktsSent	kbytesSent	kbytesI
0	105.89.111.120	125.39.118.75	28847	32584	TCP	1120	2023-04-26 03:26:18	376	1424	
1	67.162.41.35	96.65.28.109	4666	14817	TCP	481	2023-05-02 20:55:23	773	588	
2	21.12.248.67	221.80.136.139	44942	59301	UDP	152	2023-05-24 20:54:31	294	1834	
3	4.92.166.209	34.96.37.72	63574	4929	ICMP	144	2022-09-25 09:53:40	904	1507	
4	42.96.78.99	8.99.218.138	4431	22529	TCP	860	2022-08-06 14:24:11	861	1330	

In [11]:
```python
data.describe(include='all')
```

Out[11]:

	Source IP	Destination IP	Source Port	Destination Port	Protocol	Packet Size	Timestamp	pktsSen
count	94200	94200	94200.000000	94200.000000	94200	94200.000000	94200	94200.00000
unique	94199	94200	NaN	NaN	3	NaN	92262	Na
top	170.168.163.38	125.39.118.75	NaN	NaN	UDP	NaN	26/12/2022 09:32	Na
freq	2	1	NaN	NaN	31508	NaN	4	Na
mean	NaN	NaN	32681.652739	32785.473312	NaN	779.501274	NaN	501.79107
std	NaN	NaN	18954.108222	18914.681025	NaN	415.360703	NaN	287.88371
min	NaN	NaN	2.000000	2.000000	NaN	64.000000	NaN	1.00000
25%	NaN	NaN	16205.000000	16417.000000	NaN	418.000000	NaN	253.00000
50%	NaN	NaN	32651.000000	32751.500000	NaN	778.000000	NaN	501.50000
75%	NaN	NaN	49135.250000	49189.000000	NaN	1139.000000	NaN	752.00000
max	NaN	NaN	65535.000000	65535.000000	NaN	1500.000000	NaN	1000.00000

In [12]:

```
data.columns
```

Out[12]:

```
Index(['Source IP', 'Destination IP', 'Source Port', 'Destination Port',
       'Protocol', 'Packet Size', 'Timestamp', 'pktsSent', 'kbytesSent',
       'kbytesReceived', 'TTL (Time to Live) Value', 'Flag', 'VLAN ID',
       'QoS (Quality of Service)', 'AS (Autonomous System) Number',
       'Geolocation', 'Application', 'Threat Score', 'Payload', 'Packet ID',
       'Time to Live (TTL)', 'Quality of Service (QoS) Class', 'Fragmentation',
       'Type of Service (ToS)', 'Hop Count', 'Error Codes', 'Flow ID',
       'Routing Information', 'Packet Capture Timestamp', 'Attack Type'],
      dtype='object')
```

In [13]:

```
#Dropping columns after feature selection
data.drop(['Source IP', 'Destination IP','Timestamp','Packet Capture Timestamp','Payload','Flow I
```

In [14]:

```
data.head()
```

Out[14]:

	Source Port	Destination Port	Protocol	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	Flag	VLAN ID	QoS (Quality of Service)
0	28847	32584	TCP	1120	376	1424	1994	110	FIN	7	Gold
1	4666	14817	TCP	481	773	588	972	59	ACK	3	Gold
2	44942	59301	UDP	152	294	1834	1895	121	FIN	8	Platinum
3	63574	4929	ICMP	144	904	1507	694	36	ACK	1	Platinum
4	4431	22529	TCP	860	861	1330	867	84	ACK	3	Platinum

In [15]:

```
#Attack is multi-class, we want to only deal with binary class, i.e, malicious or benign attack t
data['Attack Type'].value_counts()
```

Out[15]:

```
SMBLoris Attack            5130
DoS                        5081
Infiltration               5055
MS17Scan Attack            5034
MemcrashedSpoofer Attack   5001
None                       4993
JoomlaRegPrivesc Exploit   4984
Heart-bleed                4975
DDoS                       4972
P2PBotnet                  4963
Portscan Attack            4938
SQLi Attack                4932
EternalBlue Exploit        4898
Sality Botnet              4897
SMBScan Attack             4894
FTPWinaXe Exploit          4878
Brute force                4861
DDoS Attack                4860
Web-based                  4854
Name: Attack Type, dtype: int64
```

In [16]:

```
data.describe()
```

Out[16]:

	Source Port	Destination Port	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	V
count	94200.000000	94200.000000	94200.000000	94200.000000	94200.000000	94200.000000	94200.000000	9420
mean	32681.652739	32785.473312	779.501274	501.791072	1026.240743	1025.474501	64.434565	5
std	18954.108222	18914.681025	415.360703	287.883713	590.361131	590.371901	36.914171	2
min	2.000000	2.000000	64.000000	1.000000	1.000000	1.000000	1.000000	1
25%	16205.000000	16417.000000	418.000000	253.000000	519.000000	515.000000	33.000000	3
50%	32651.000000	32751.500000	778.000000	501.500000	1026.000000	1028.000000	64.000000	5
75%	49135.250000	49189.000000	1139.000000	752.000000	1537.000000	1536.000000	96.000000	8
max	65535.000000	65535.000000	1500.000000	1000.000000	2048.000000	2048.000000	128.000000	10

In [17]:

```
data.dtypes
```

Out[17]:

```
Source Port                          int64
Destination Port                     int64
Protocol                            object
Packet Size                          int64
pktsSent                             int64
kbytesSent                           int64
kbytesReceived                       int64
TTL (Time to Live) Value             int64
Flag                                object
VLAN ID                              int64
QoS (Quality of Service)            object
AS (Autonomous System) Number        int64
Geolocation                         object
Application                         object
Threat Score                         int64
Time to Live (TTL)                   int64
Quality of Service (QoS) Class      object
Fragmentation                         bool
Type of Service (ToS)                int64
Hop Count                            int64
Error Codes                          int64
Attack Type                         object
dtype: object
```

In [18]:

```
for cols in data.columns:
    if data[cols].dtypes == 'int64':
        print (cols)
```

```
Source Port
Destination Port
Packet Size
pktsSent
kbytesSent
kbytesReceived
TTL (Time to Live) Value
VLAN ID
AS (Autonomous System) Number
Threat Score
Time to Live (TTL)
Type of Service (ToS)
Hop Count
Error Codes
```

In [19]:
```
for cols in data.columns:
    if data[cols].dtypes == 'object':
        print (cols)
```

```
Protocol
Flag
QoS (Quality of Service)
Geolocation
Application
Quality of Service (QoS) Class
Attack Type
```

In [20]:
```
data['Attack Type'].value_counts()
```

Out[20]:
```
SMBLoris Attack             5130
DoS                         5081
Infiltration                5055
MS17Scan Attack             5034
MemcrashedSpoofer Attack    5001
None                        4993
JoomlaRegPrivesc Exploit    4984
Heart-bleed                 4975
DDoS                        4972
P2PBotnet                   4963
Portscan Attack             4938
SQLi Attack                 4932
EternalBlue Exploit         4898
Sality Botnet               4897
SMBScan Attack              4894
FTPWinaXe Exploit           4878
Brute force                 4861
DDoS Attack                 4860
Web-based                   4854
Name: Attack Type, dtype: int64
```

Attack is multi-class, we want to only deal with binary class, i.e, malicious or benign attack type

In [21]:
```
data.replace({'SMBLoris Attack':'Malicious Attack','DoS':'Malicious Attack', 'Infiltration':'Mali
              'MS17Scan Attack':'Malicious Attack', 'MemcrashedSpoofer Attack':'Malicious Attack
              'JoomlaRegPrivesc Exploit':'Malicious Attack','Heart-bleed':'Malicious Attack','DD
              'P2PBotnet':'Malicious Attack','Portscan Attack':'Malicious Attack','SQLi Attack':
              'EternalBlue Exploit':'Malicious Attack','Sality Botnet':'Malicious Attack','SMBSc
              'FTPWinaXe Exploit':'Malicious Attack','Brute force':'Malicious Attack','DDoS Atta
              'Web-based':'Malicious Attack','None':'Benign'}, inplace=True)
```

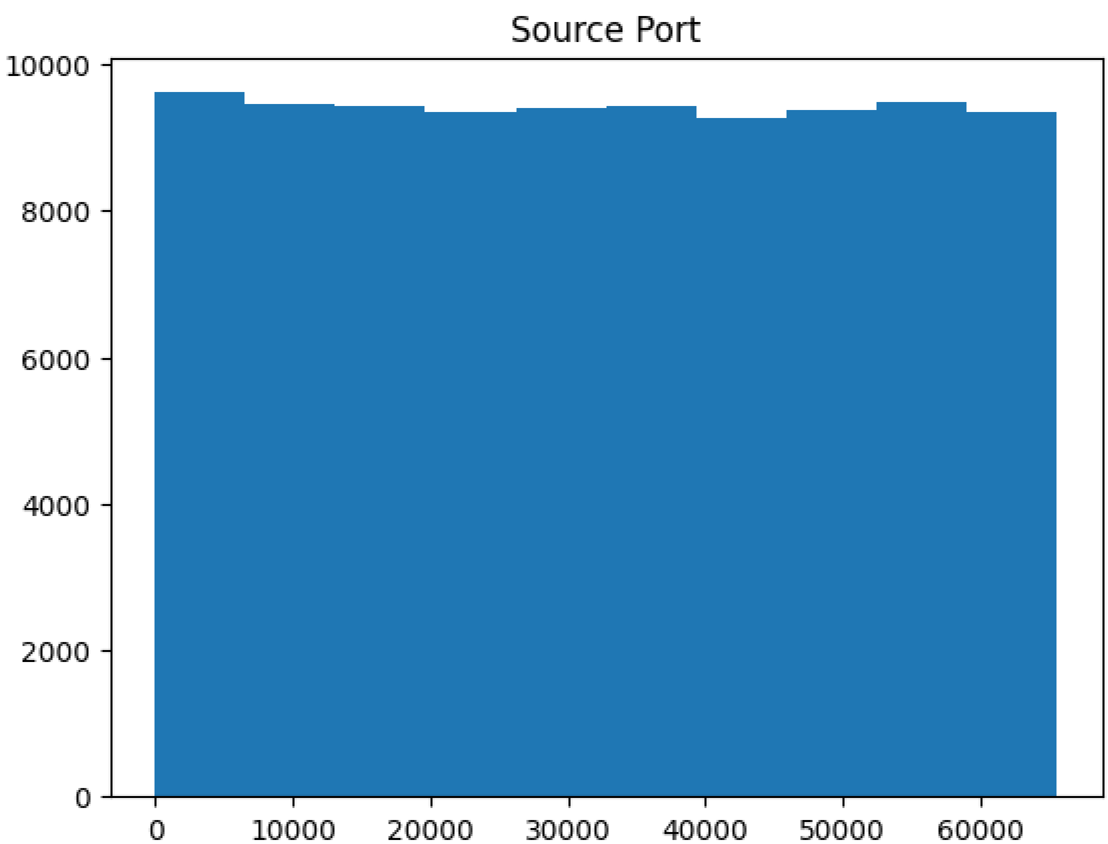
In [22]:
```
data['Attack Type'].value_counts()
```

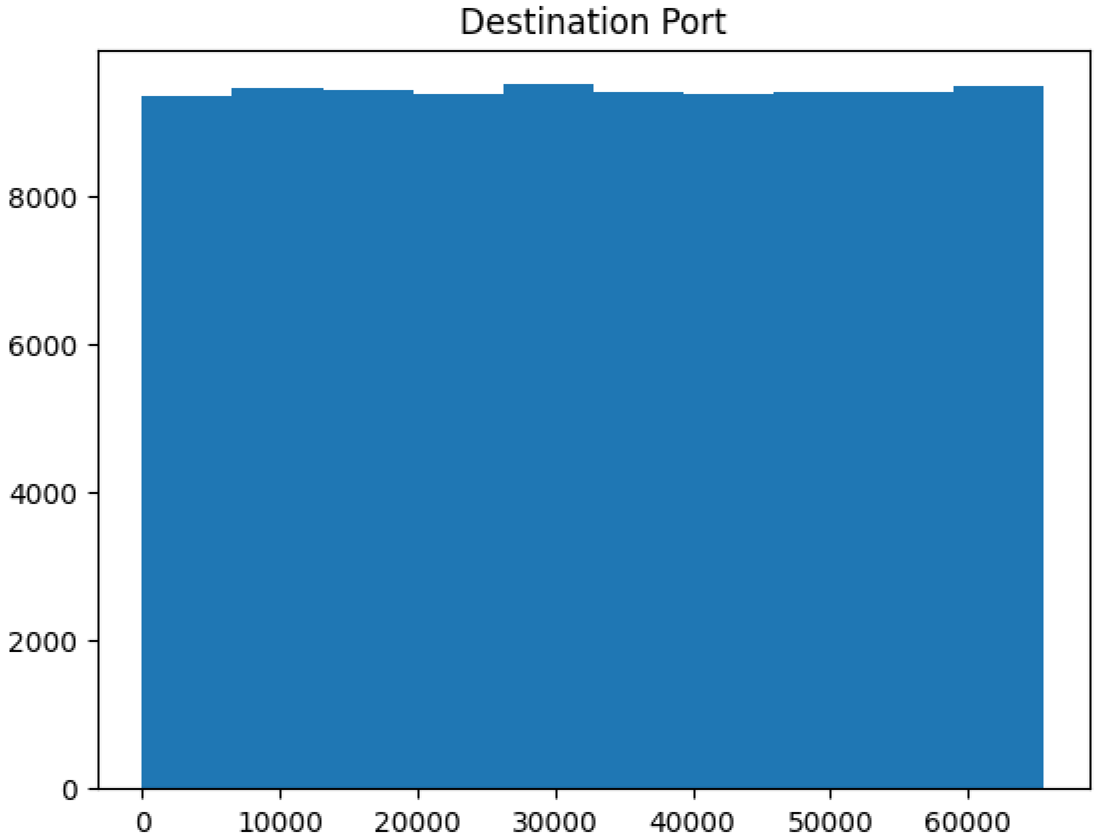
Out[22]:
```
Malicious Attack    89207
Benign               4993
Name: Attack Type, dtype: int64
```

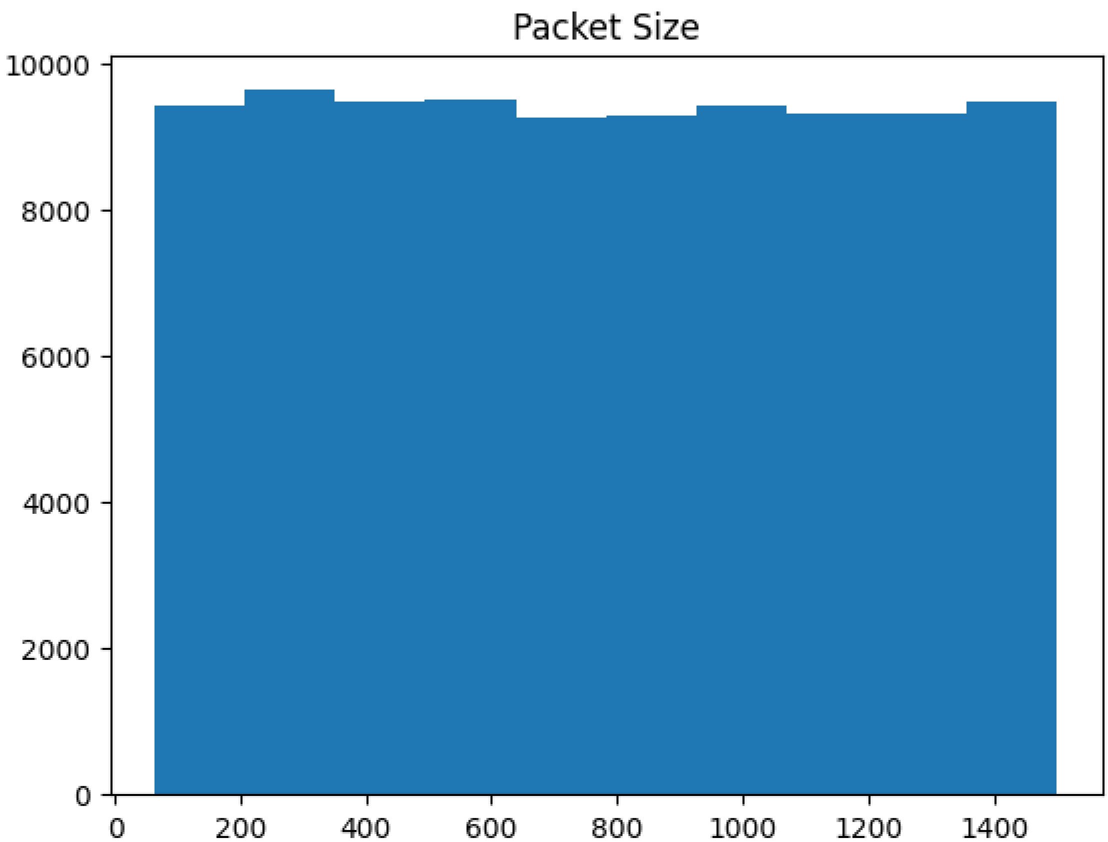
Explorative Data Analysis

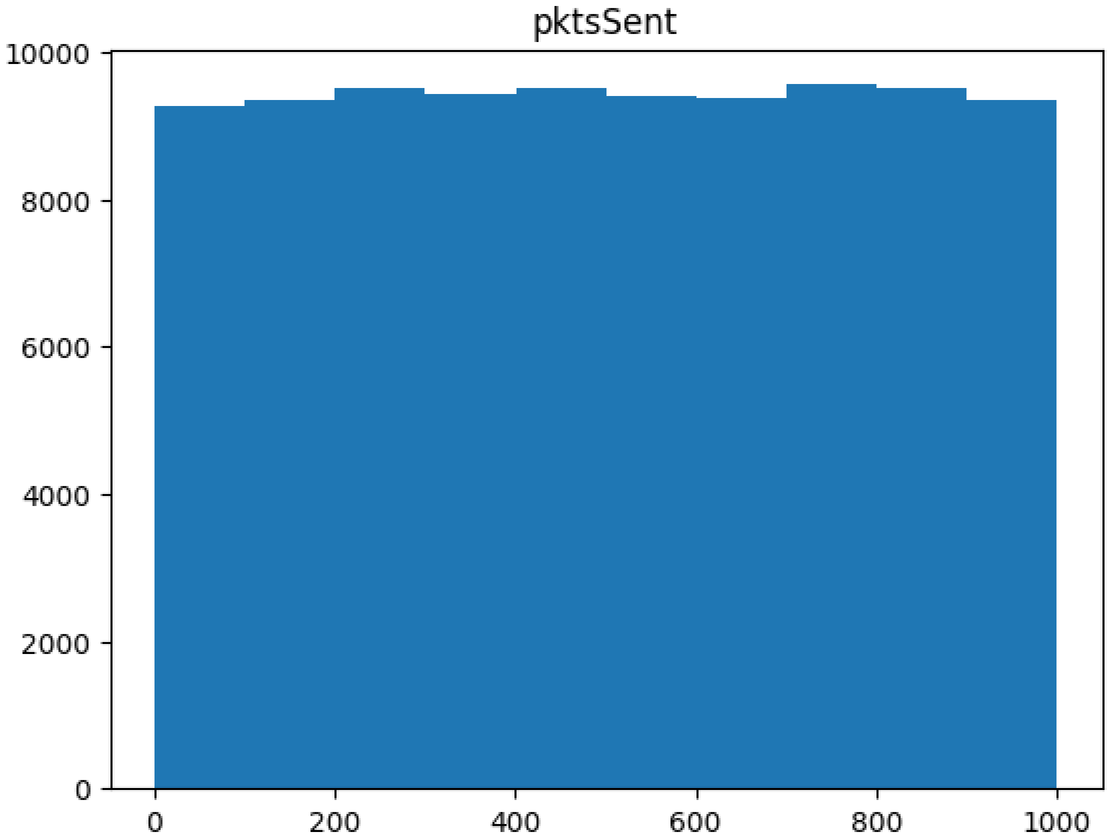
In [23]:
```
num_cols=data[['Source Port','Destination Port','Packet Size','pktsSent','kbytesSent','kbytesRec
'VLAN ID','AS (Autonomous System) Number','Threat Score','Time to Live (TTL)','Type of Service (

cat_cols=data[['Protocol','Flag','QoS (Quality of Service)','Geolocation','Application','Quality
'Attack Type']]

num_colss=['Source Port','Destination Port','Packet Size','pktsSent','kbytesSent','kbytesReceive
'VLAN ID','AS (Autonomous System) Number','Threat Score','Time to Live (TTL)','Type of Service (
```

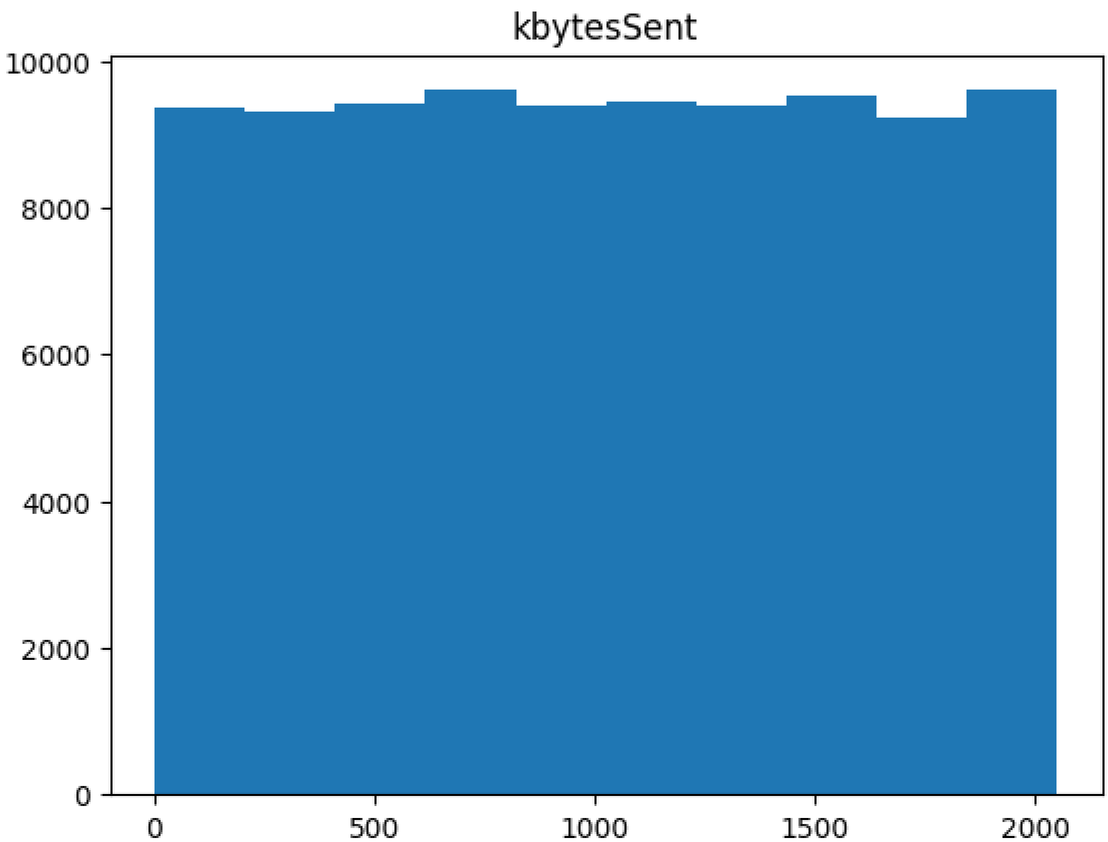
In [24]:
```
for jeu in num_cols.columns:
    plt.hist(num_cols[jeu])
    plt.title(jeu)
    plt.show()
```

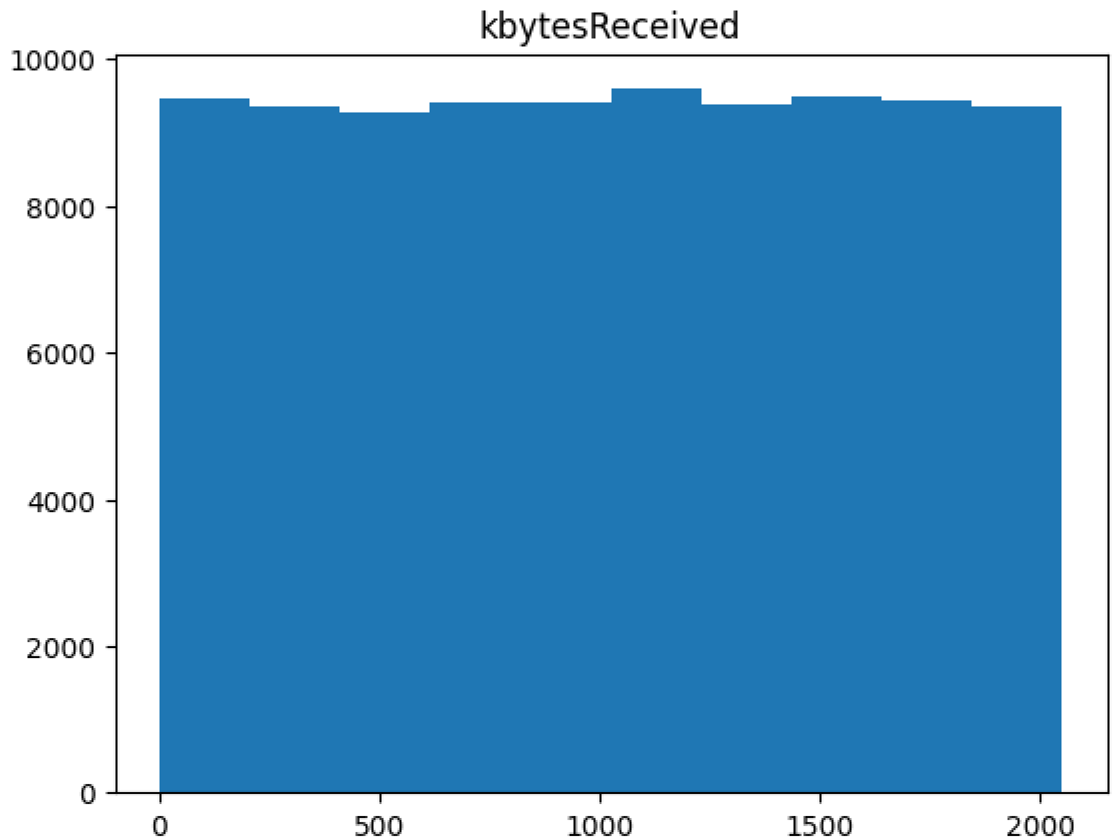


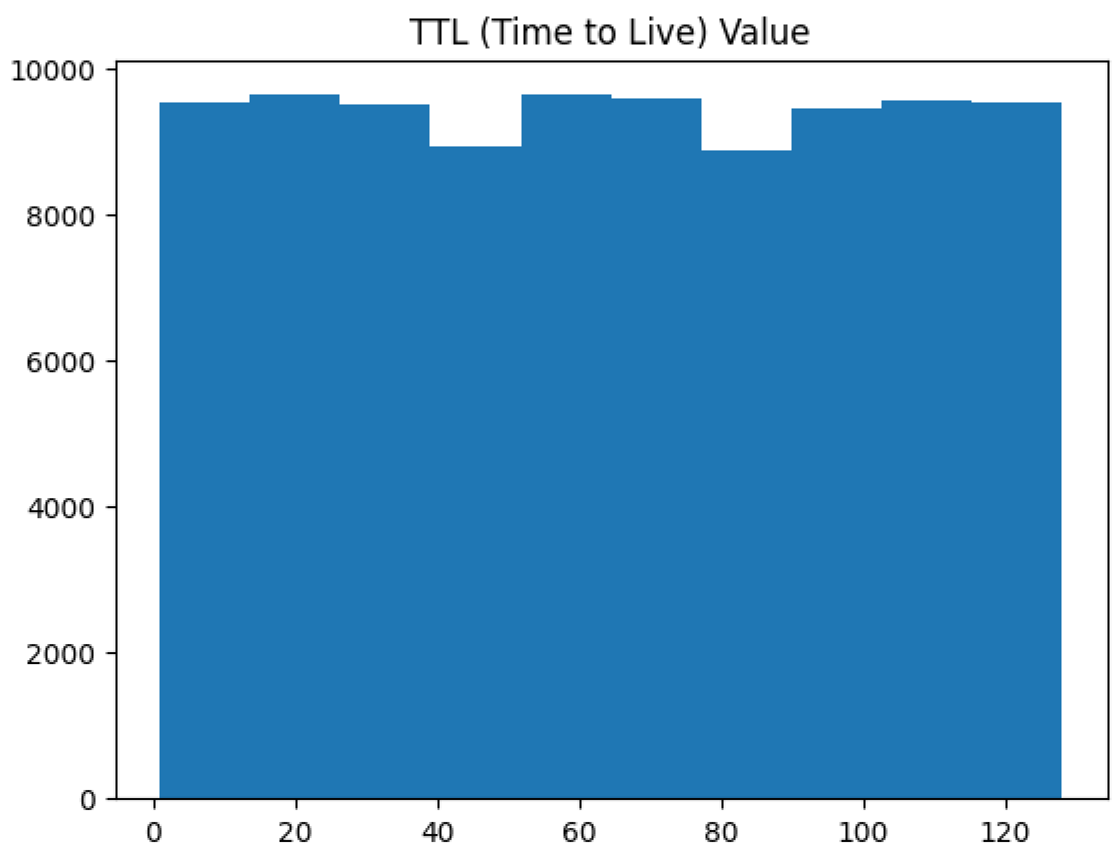

Destination Port
0
2000
4000
6000
8000
0
10000
20000
30000
40000
50000
60000

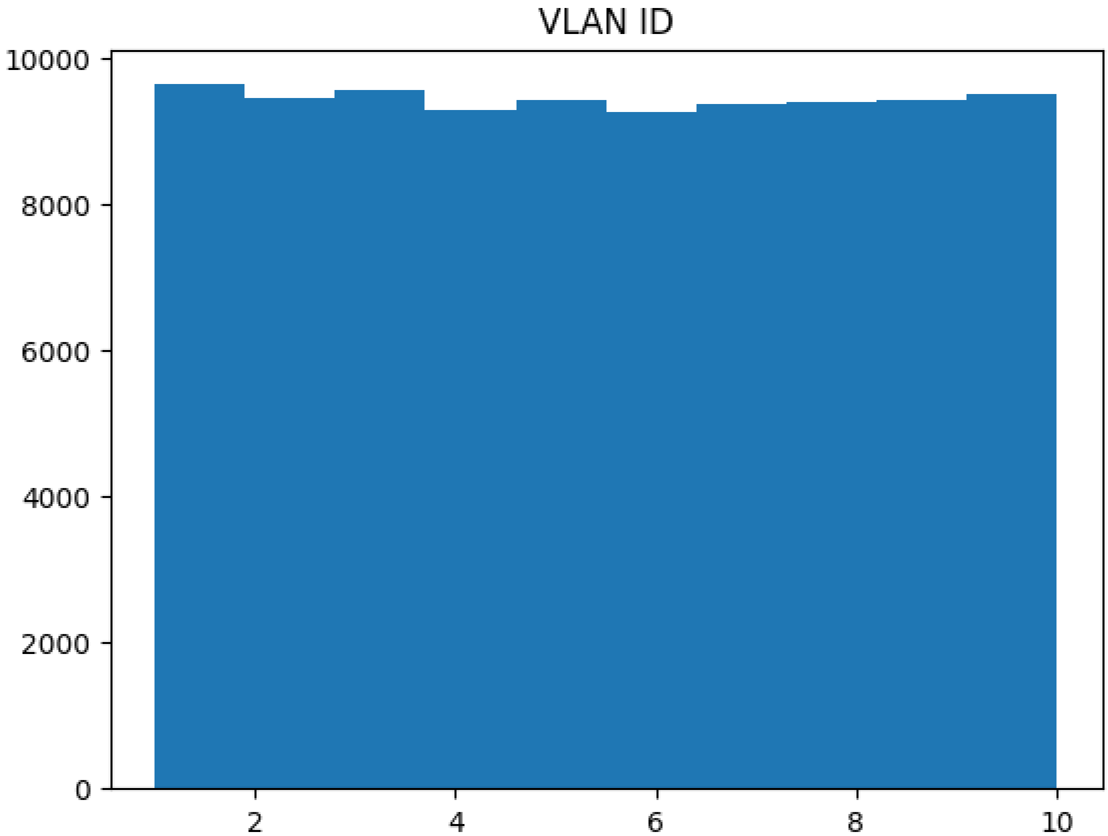

Packet Size
0
2000
4000
6000
8000
10000
0
200
400
600
800
1000
1200
1400

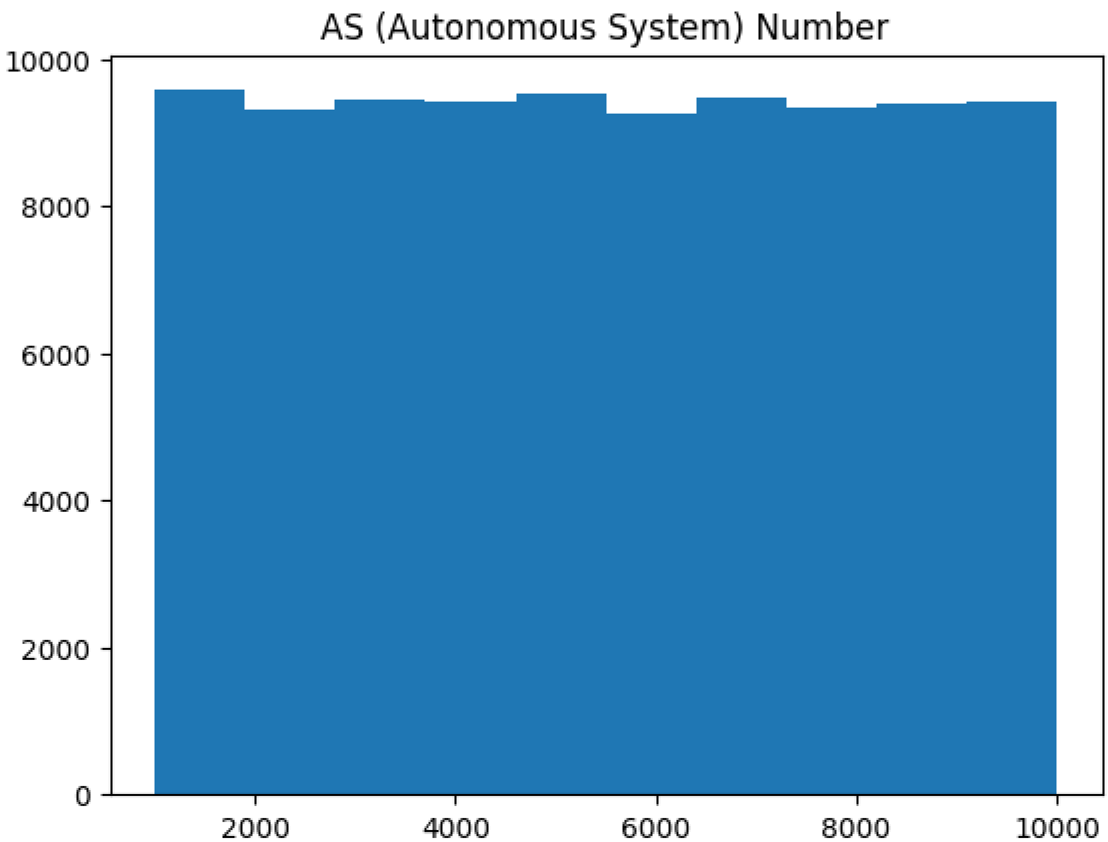

pktsSent
10000
8000
6000
4000
2000
0
0
200
400
600
800
1000

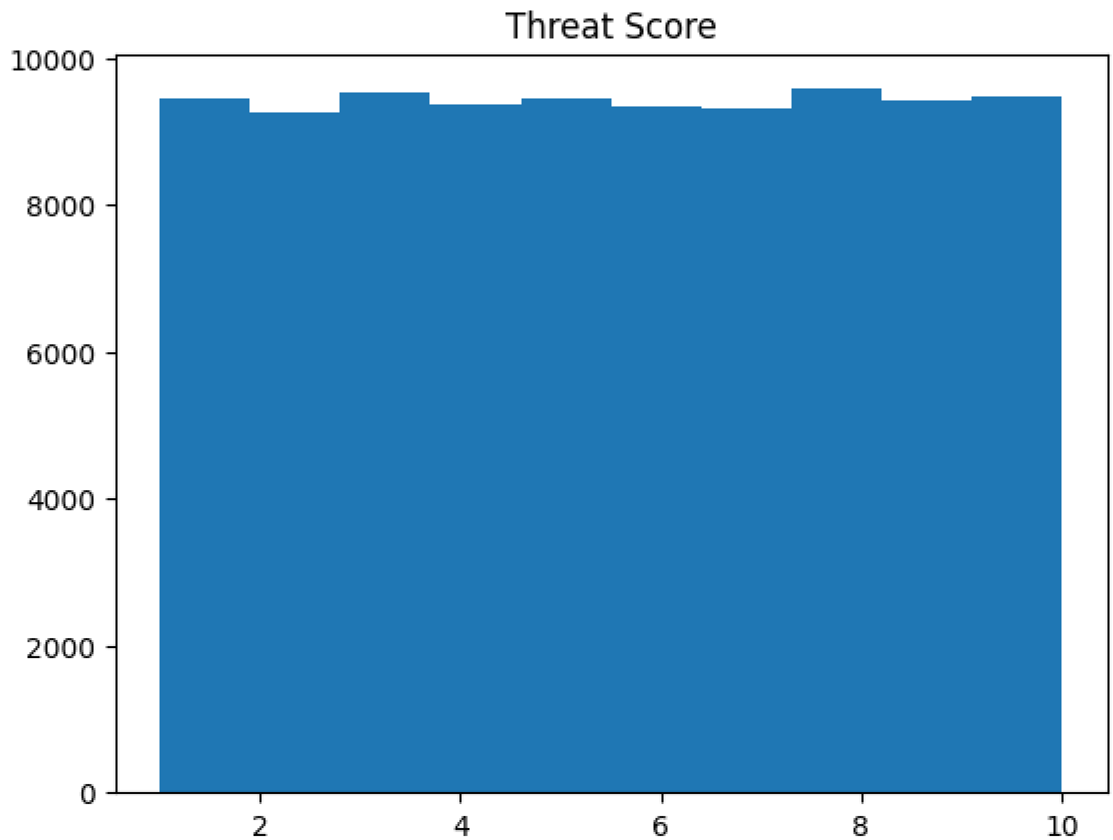

kbytesSent
10000
8000
6000
4000
2000
0
0
500
1000
1500
2000

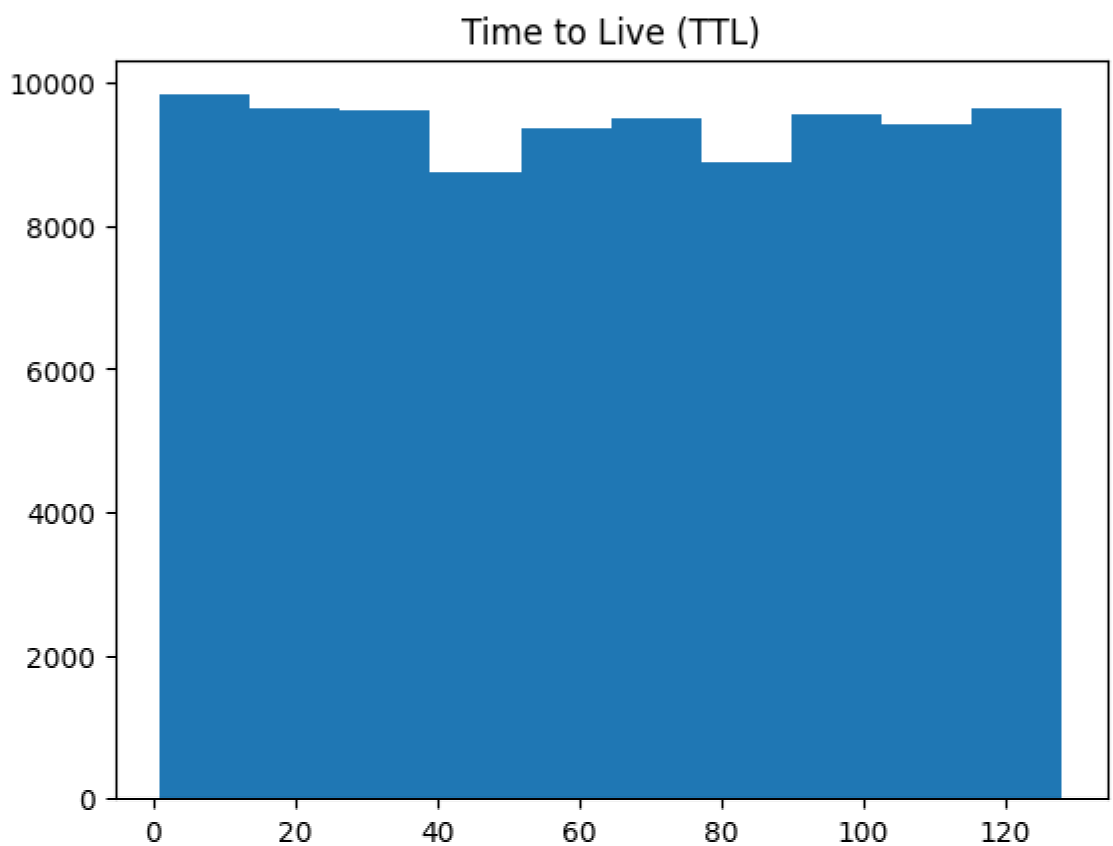

kbytesReceived
0
2000
4000
6000
8000
10000
0
500
1000
1500
2000

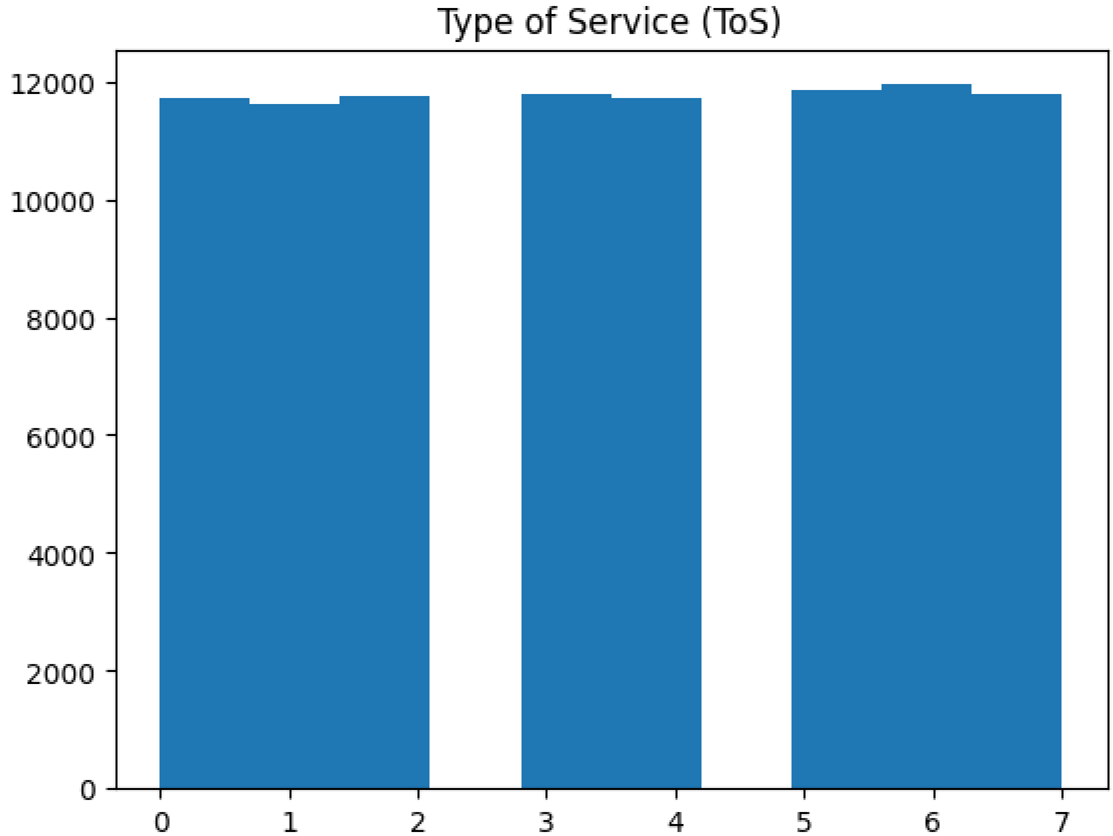

TTL (Time to Live) Value
0
2000
4000
6000
8000
10000
0
20
40
60
80
100
120


VLAN ID
10000
8000
6000
4000
2000
0
2
4
6
8
10

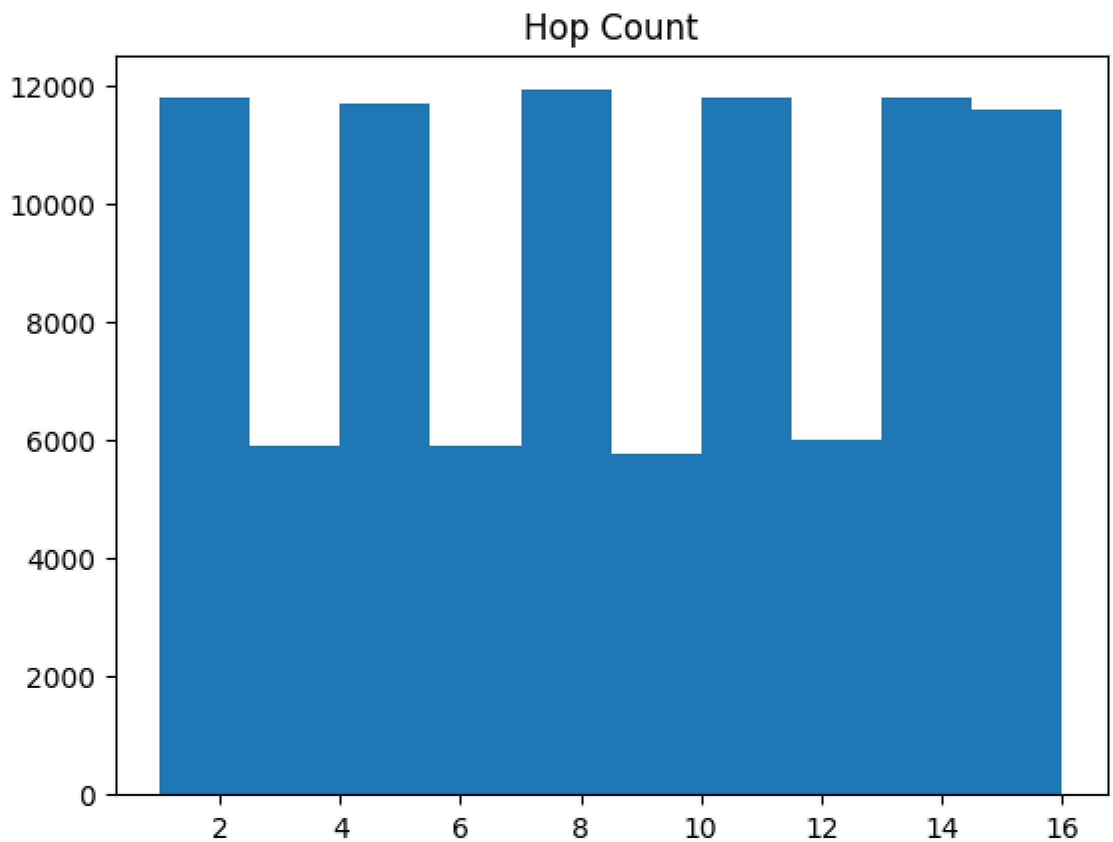

AS (Autonomous System) Number
10000
8000
6000
4000
2000
0
2000
4000
6000
8000
10000


Threat Score
10000
8000
6000
4000
2000
0
2
4
6
8
10

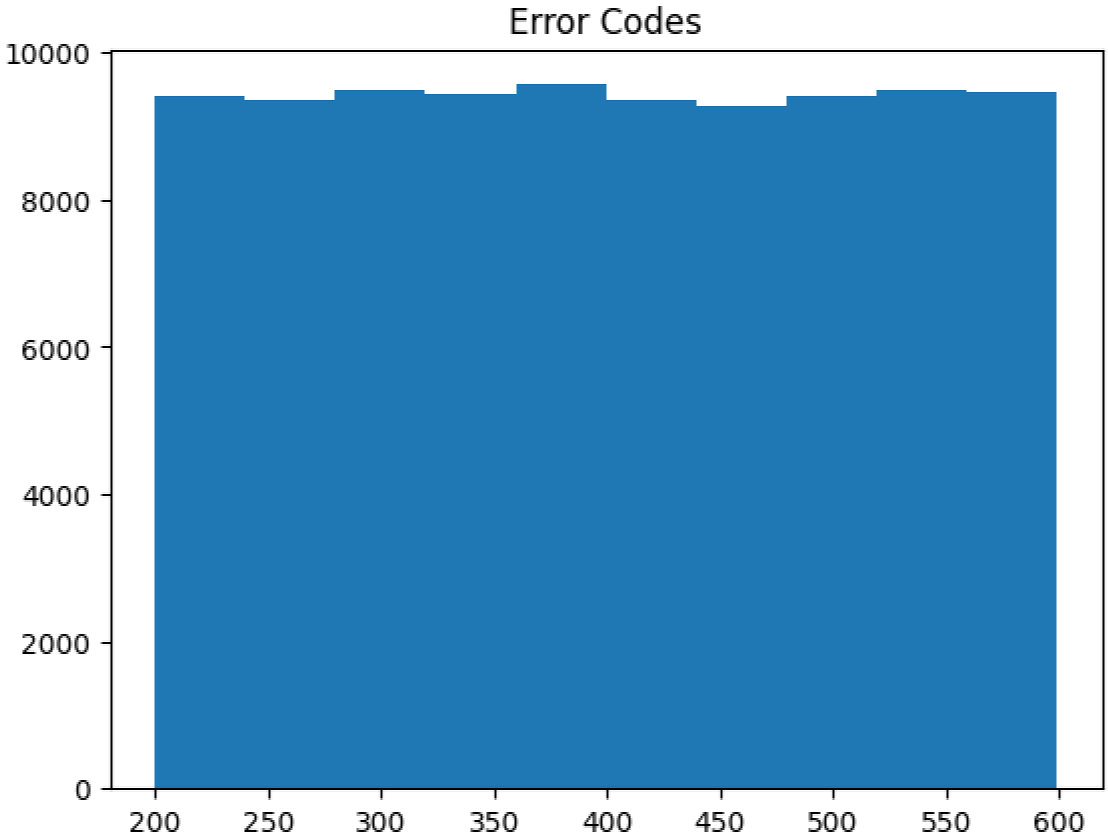

Time to Live (TTL)
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120



Type of Service (ToS)
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7



Hop Count
12000
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16



In [25]:
```
num_cols.corr()
```

Out[25]:

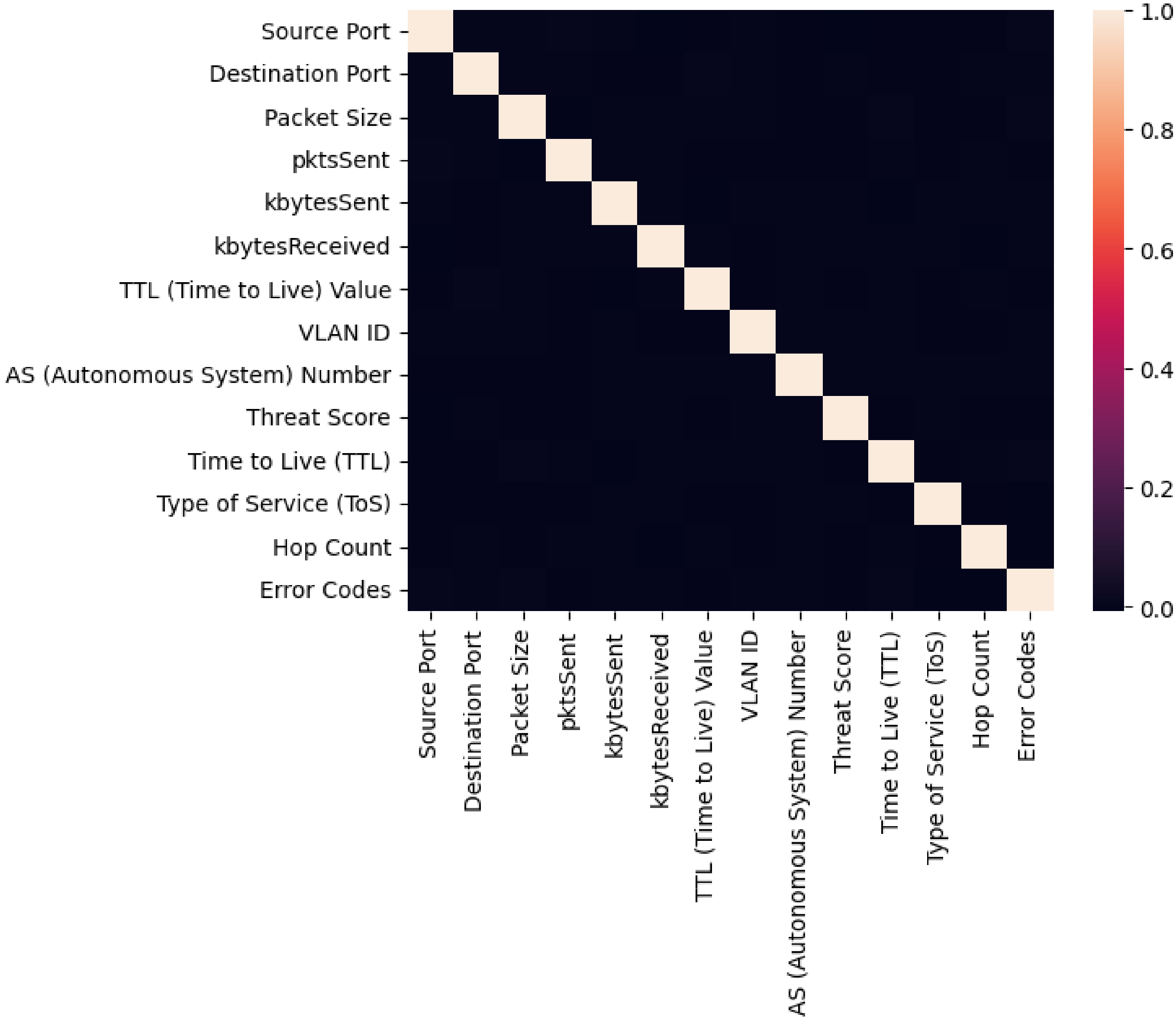
	Source Port	Destination Port	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	VLAN ID	(A
Source Port	1.000000	0.002343	0.002564	0.004883	0.003763	-0.007799	-0.001072	0.001663	
Destination Port	0.002343	1.000000	0.000691	0.003877	-0.001283	-0.000083	0.007869	0.002176	
Packet Size	0.002564	0.000691	1.000000	-0.004067	0.002093	0.000616	0.001806	0.000967	
pktsSent	0.004883	0.003877	-0.004067	1.000000	0.003129	0.000578	-0.003596	-0.002198	
kbytesSent	0.003763	-0.001283	0.002093	0.003129	1.000000	0.002742	-0.007069	0.000434	
kbytesReceived	-0.007799	-0.000083	0.000616	0.000578	0.002742	1.000000	0.001198	-0.000728	
TTL (Time to Live) Value	-0.001072	0.007869	0.001806	-0.003596	-0.007069	0.001198	1.000000	-0.000265	
VLAN ID	0.001663	0.002176	0.000967	-0.002198	0.000434	-0.000728	-0.000265	1.000000	
AS (Autonomous System) Number	-0.000611	-0.000289	-0.000049	-0.001332	0.001463	0.000307	0.001019	0.003922	
Threat Score	-0.004285	0.000987	-0.000099	0.000065	0.001449	0.001543	-0.000752	0.001603	
Time to Live (TTL)	-0.002721	-0.002663	0.005204	0.000683	-0.004228	0.002189	0.003807	0.000818	
Type of Service (ToS)	-0.000878	-0.000729	-0.002832	-0.003142	0.000338	0.002420	-0.000703	-0.000259	
Hop Count	-0.000130	0.003468	-0.000544	0.002991	0.000277	-0.004387	0.000114	-0.000299	
Error Codes	0.004333	0.002536	0.006336	-0.001598	0.002619	0.003628	-0.003753	0.001686	

In [26]:

```
sns.heatmap(num_cols.corr())
```

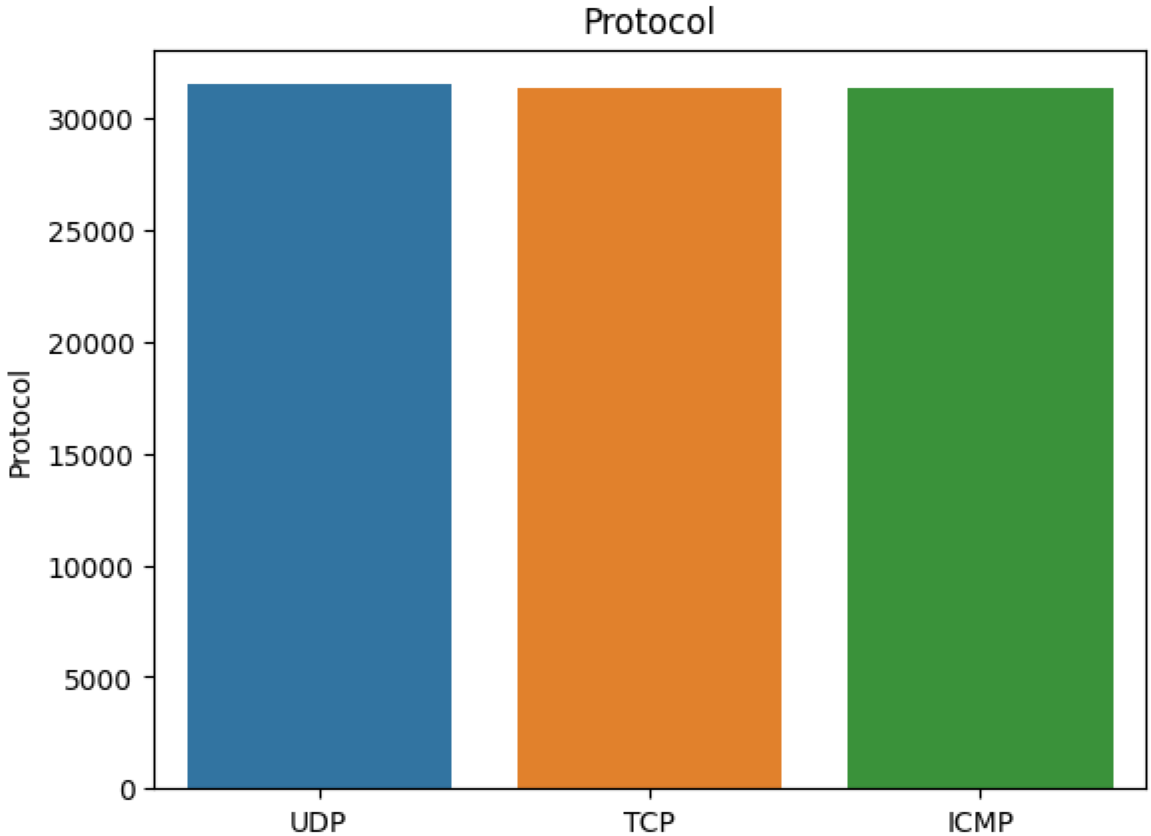
Out[26]:

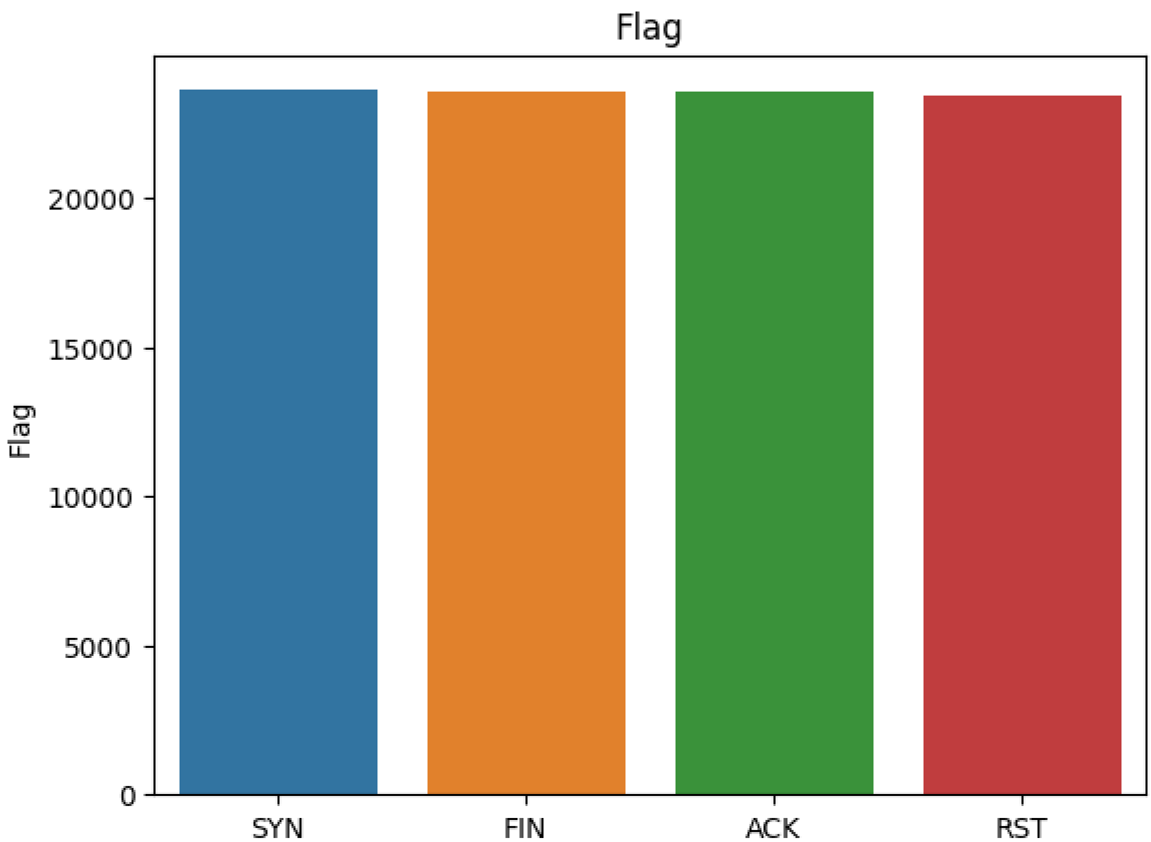
```
<Axes: >
```

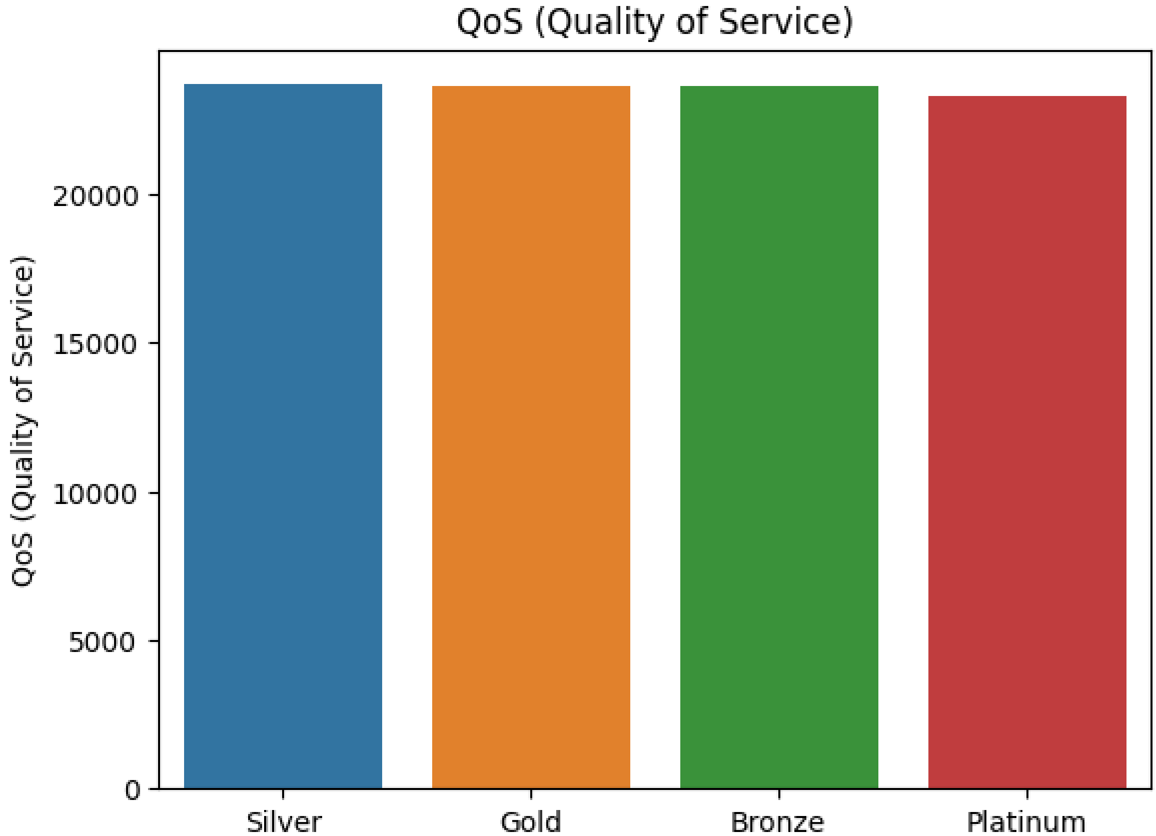


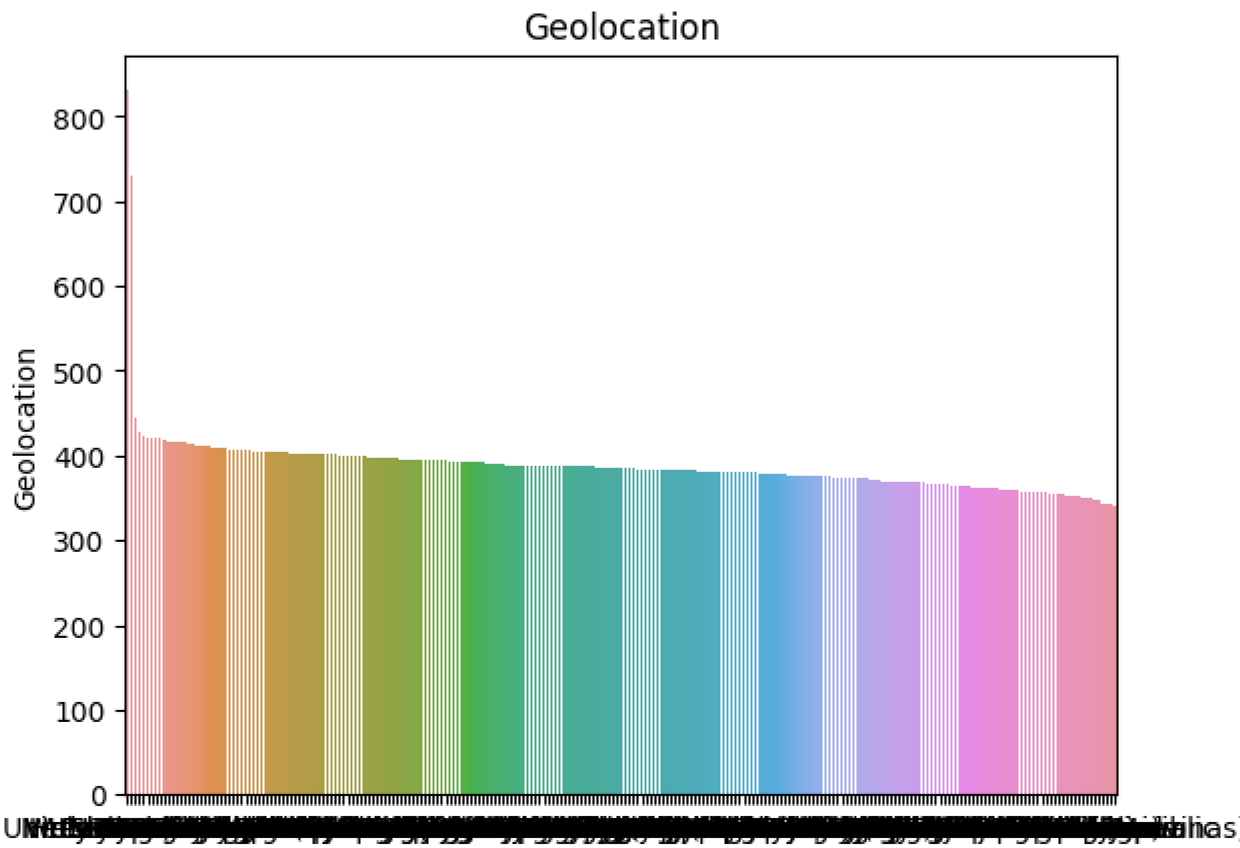
In [27]:

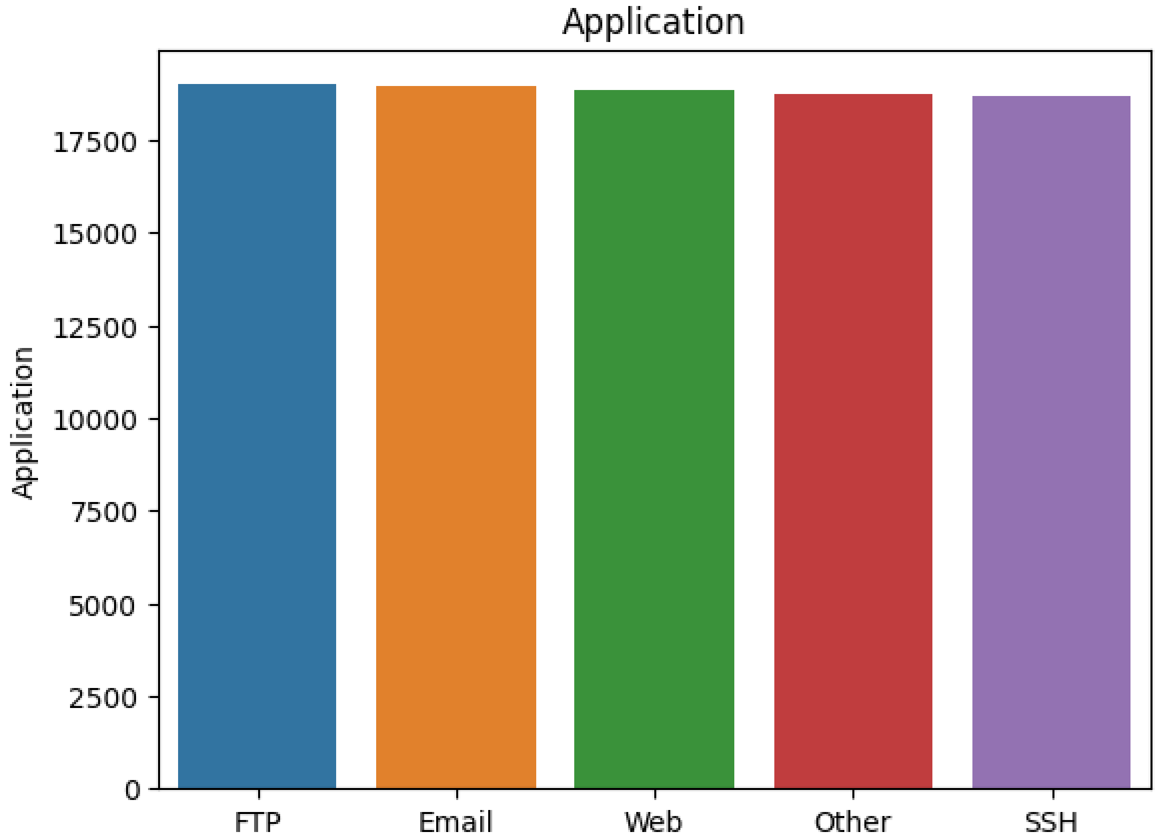
```
for i in cat_cols.columns:
    sns.barplot(x=cat_cols[i].value_counts().index,y=cat_cols[i].value_counts()).set_title(i)
    plt.show()
```

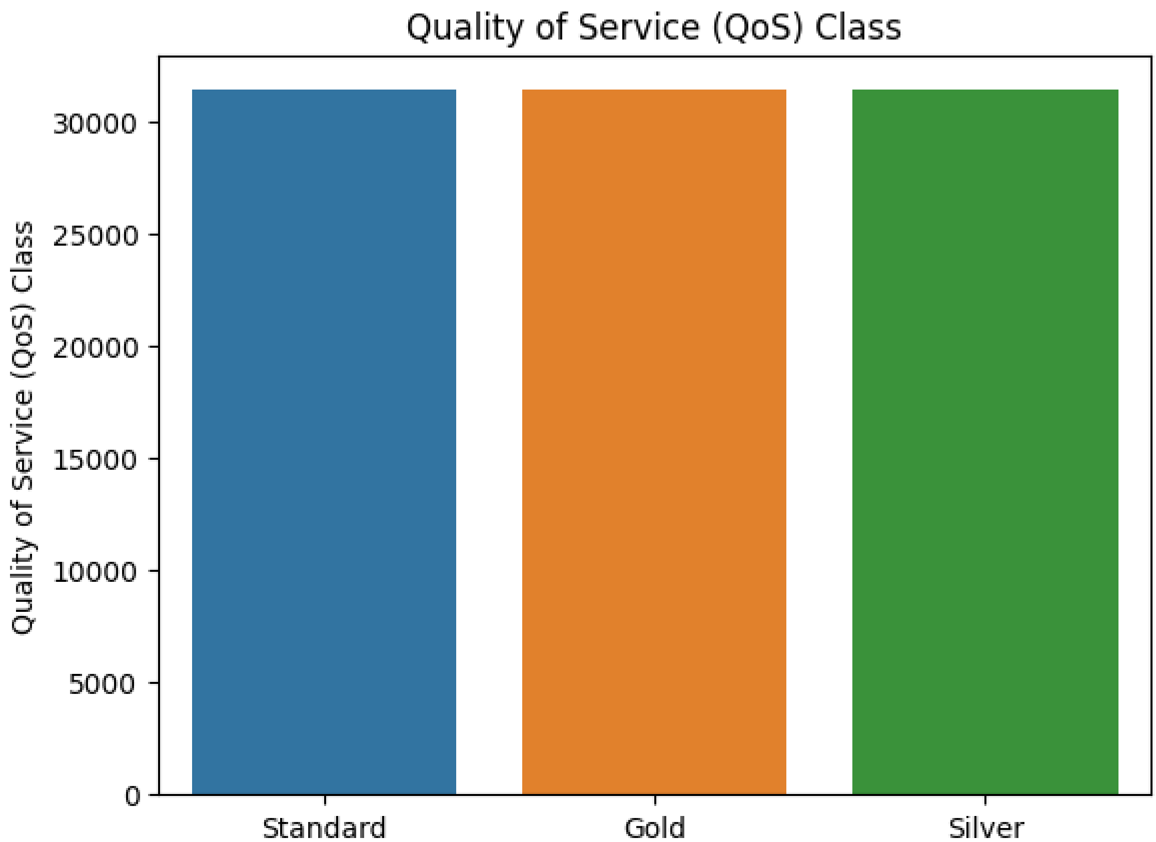

Protocol
30000
25000
20000
15000
10000
5000
0
Protocol
UDP
TCP
ICMP

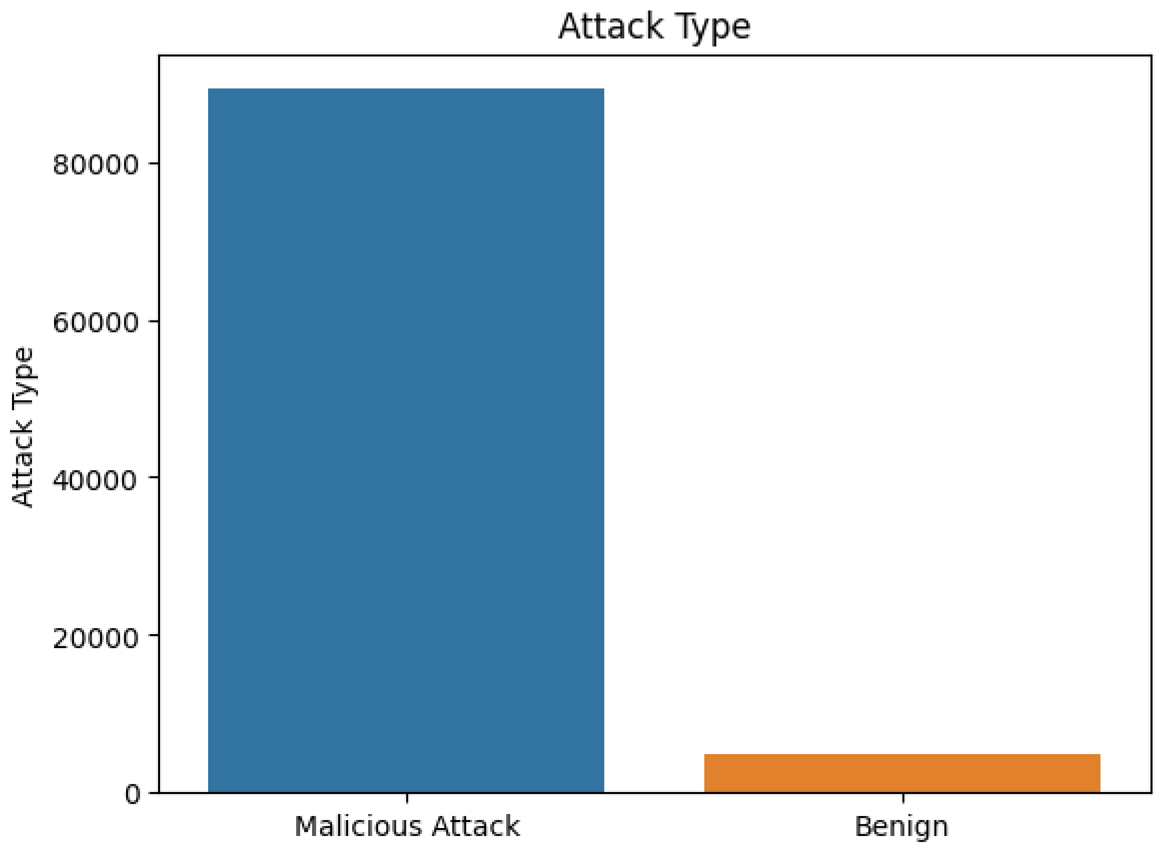

Flag
20000
15000
10000
5000
0
Flag
SYN
FIN
ACK
RST


QoS (Quality of Service)
QoS (Quality of Service)
0
5000
10000
15000
20000
Silver
Gold
Bronze
Platinum


Geolocation
Geolocation
0
100
200
300
400
500
600
700
800


Application
Application
0
2500
5000
7500
10000
12500
15000
17500
FTP
Email
Web
Other
SSH


Quality of Service (QoS) Class
Quality of Service (QoS) Class
0
5000
10000
15000
20000
25000
30000
Standard
Gold
Silver



From the charts above, we can see that the attack types are imbalanced

In [28]:
```python
data.describe()
```

Out[28]:

	Source Port	Destination Port	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	V
count	94200.000000	94200.000000	94200.000000	94200.000000	94200.000000	94200.000000	94200.000000	9420
mean	32681.652739	32785.473312	779.501274	501.791072	1026.240743	1025.474501	64.434565	5
std	18954.108222	18914.681025	415.360703	287.883713	590.361131	590.371901	36.914171	2
min	2.000000	2.000000	64.000000	1.000000	1.000000	1.000000	1.000000	1
25%	16205.000000	16417.000000	418.000000	253.000000	519.000000	515.000000	33.000000	3
50%	32651.000000	32751.500000	778.000000	501.500000	1026.000000	1028.000000	64.000000	5
75%	49135.250000	49189.000000	1139.000000	752.000000	1537.000000	1536.000000	96.000000	8
max	65535.000000	65535.000000	1500.000000	1000.000000	2048.000000	2048.000000	128.000000	10

In [29]:
```python
data.head()
```

Out[29]:

	Source Port	Destination Port	Protocol	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	Flag	VLAN ID	QoS (Quality of Service)	(A
0	28847	32584	TCP	1120	376	1424	1994	110	FIN	7	Gold	
1	4666	14817	TCP	481	773	588	972	59	ACK	3	Gold	
2	44942	59301	UDP	152	294	1834	1895	121	FIN	8	Platinum	
3	63574	4929	ICMP	144	904	1507	694	36	ACK	1	Platinum	
4	4431	22529	TCP	860	861	1330	867	84	ACK	3	Platinum	

In [30]:
```
scaler=MinMaxScaler()
data[num_colss]=scaler.fit_transform(data[num_colss])
```

In [31]:
```
data.sample(3)
```

Out[31]:

	Source Port	Destination Port	Protocol	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	Flag	VLAN ID
19240	0.513970	0.596570	TCP	0.830780	0.407407	0.846116	0.700537	0.677165	ACK	0.666667
9168	0.808661	0.806891	TCP	0.713788	0.869870	0.652662	0.083048	0.937008	SYN	0.222222
10068	0.353929	0.019700	TCP	0.316852	0.450450	0.011236	0.280899	0.732283	ACK	0.555556

In this section of the code, we want to convert each countries to their various continents as a form of feature engineering. Islands on the Antartica are mapped to countries that own them

In [32]:
```
data['Geolocation'].nunique()
```

Out[32]: 243

In [33]:
```
data['Geolocation'].replace({'Palestinian Territory':'Palestine'},inplace=True)
data['Geolocation'].replace({'Pitcairn Islands':'Australia'},inplace=True)
data['Geolocation'].replace({'Holy See (Vatican City State)':'Italy'},inplace=True)
data['Geolocation'].replace({'Western Sahara':'Morocco'},inplace=True)
data['Geolocation'].replace({'Korea':'South Korea'},inplace=True)
data['Geolocation'].replace({'Reunion':'France'},inplace=True)
data['Geolocation'].replace({'Slovakia (Slovak Republic)':'Slovakia'},inplace=True)
data['Geolocation'].replace({'Saint Barthelemy':'Cuba'},inplace=True)
data['Geolocation'].replace({'Timor-Leste':'India'},inplace=True)
data['Geolocation'].replace({'Netherlands Antilles':'Netherlands'},inplace=True)
data['Geolocation'].replace({'British Indian Ocean Territory (Chagos Archipelago)':'Mauritius'},i
data['Geolocation'].replace({"Cote d'Ivoire":'Ivory Coast'},inplace=True)
```

```python
data['Geolocation'].replace({"Svalbard & Jan Mayen Islands":'Norway'},inplace=True)
data['Geolocation'].replace({"United States Minor Outlying Islands":'United States'},inplace=Tru
data['Geolocation'].replace({"Libyan Arab Jamahiriya":'Libya'},inplace=True)
```

In [34]:
```python
data.drop(data[data['Geolocation']=='Antarctica (the territory South of 60 deg S)'].index,axis=0
data.drop(data[data['Geolocation']=='French Southern Territories'].index,axis=0,inplace=True)
data.drop(data[data['Geolocation']=='Bouvet Island (Bouvetoya)'].index,axis=0,inplace=True)
data.drop(data[data['Geolocation']=='Saint Helena'].index,axis=0,inplace=True)
```

In [35]:
```python
#Getting the country code
def convert(row):
    cn_code=pc.country_name_to_country_alpha2(row.Geolocation,cn_name_format='default')
    conti_code=pc.country_alpha2_to_continent_code(cn_code)
    return conti_code
```

In [36]:
```python
data['Continent']=data.apply(convert, axis=1)
data
```

In [ ]:
```python
data.Continent
```

In [ ]:
```python
data.isna().sum()
```

In [ ]:
```python
data['Continent'].value_counts()
```

In [ ]:
```python
conti_names={
    "AF":"Africa",
    "AS":"Asian",
    "EU": "Europe",
    "NA": "North America",
    "OC": "Oceania",
    "SA": "South America",
    "AN": "Antarctica"
}
data['Continent']=data['Continent'].map(conti_names)
data['Continent'].value_counts()
```

In [42]:
```python
data.drop(['Geolocation'], axis=1,inplace=True)
```

In [43]:
```python
data.head()
```

Out[43]:

	Source Port	Destination Port	Protocol	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	Flag	VLAN ID	(Qu Serv
1	0.071170	0.226069	TCP	0.290390	0.772773	0.286761	0.474353	0.456693	ACK	0.222222	(
2	0.685761	0.904872	UDP	0.061281	0.293293	0.895457	0.925256	0.944882	FIN	0.777778	Plati
4	0.067584	0.343750	TCP	0.554318	0.860861	0.649243	0.423058	0.653543	ACK	0.222222	Plati
5	0.810233	0.753849	UDP	0.437326	0.327327	0.317538	0.516365	0.850394	FIN	0.888889	(
6	0.005463	0.540049	ICMP	0.920613	0.316316	0.638984	0.209575	0.417323	FIN	0.888889	Br

In [44]:
```
data['Fragmentation'].replace({False:0,True:1},inplace=True)
data['Attack Type'].replace({'Malicious Attack':0,'Benign':1}, inplace=True)
```

In [45]:
```
data['Flag'].nunique()
```

Out[45]: 4

In [46]:
```
data['Application'].nunique()
```

Out[46]: 5

In [47]:
```
data=pd.get_dummies(data=data,columns=['Flag','QoS (Quality of Service)','Application','Continen
```

In [47]:
```

```

In [48]:
```
data.head()
```

Out[48]:

	Source Port	Destination Port	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	VLAN ID	AS (Autonomous System) Number	Thr Sc
1	0.071170	0.226069	0.290390	0.772773	0.286761	0.474353	0.456693	0.222222	0.408045	0.333
2	0.685761	0.904872	0.061281	0.293293	0.895457	0.925256	0.944882	0.777778	0.386932	0.666
4	0.067584	0.343750	0.554318	0.860861	0.649243	0.423058	0.653543	0.222222	0.147350	0.444
5	0.810233	0.753849	0.437326	0.327327	0.317538	0.516365	0.850394	0.888889	0.542838	0.555
6	0.005463	0.540049	0.920613	0.316316	0.638984	0.209575	0.417323	0.888889	0.544394	1.000

In [49]:
```
data.shape
```

Out[49]: (87538, 42)

In [50]:
```
data.dtypes
```

Out[50]:
```
Source Port                                  float64
Destination Port                             float64
Packet Size                                  float64
pktsSent                                     float64
kbytesSent                                   float64
kbytesReceived                               float64
TTL (Time to Live) Value                     float64
VLAN ID                                      float64
AS (Autonomous System) Number                float64
Threat Score                                 float64
Time to Live (TTL)                           float64
Fragmentation                                  int64
Type of Service (ToS)                        float64
Hop Count                                    float64
Error Codes                                  float64
Attack Type                                    int64
Flag_ACK                                       uint8
Flag_FIN                                       uint8
Flag_RST                                       uint8
Flag_SYN                                       uint8
QoS (Quality of Service)_Bronze                uint8
QoS (Quality of Service)_Gold                  uint8
QoS (Quality of Service)_Platinum              uint8
QoS (Quality of Service)_Silver                uint8
Application_Email                              uint8
Application_FTP                                uint8
Application_Other                              uint8
Application_SSH                                uint8
Application_Web                                uint8
Continent_Africa                               uint8
Continent_Antarctica                           uint8
Continent_Asian                                uint8
Continent_Europe                               uint8
Continent_North America                        uint8
Continent_Oceania                              uint8
Continent_South America                        uint8
Quality of Service (QoS) Class_Gold            uint8
Quality of Service (QoS) Class_Silver          uint8
Quality of Service (QoS) Class_Standard        uint8
Protocol_ICMP                                  uint8
Protocol_TCP                                   uint8
Protocol_UDP                                   uint8
dtype: object
```

Majority Undersampling as a form of handling imbalanced dataset

In [51]:
```
data_count_0=data[data['Attack Type']==0]
data_count_1=data[data['Attack Type']==1]
```

In [52]:
```
count_class_0,count_class_1=data['Attack Type'].value_counts()
count_class_0,count_class_1
```

Out[52]:
```
(82904, 4634)
```

In [53]:
```
data_count_0.shape,data_count_1.shape
```

Out[53]: ((82904, 42), (4634, 42))

In [54]:
```
data_under_sample0 = data_count_0.sample(count_class_1)

data_under=pd.concat([data_under_sample0,data_count_1])
```

In [55]:
```
data_under['Attack Type'].value_counts()
```

Out[55]:
```
0    4634
1    4634
Name: Attack Type, dtype: int64
```

In [56]:
```
x=data_under.drop('Attack Type',axis=1)
y=data_under['Attack Type']
```

In [57]:
```
x.head()
```

Out[57]:

	Source Port	Destination Port	Packet Size	pktsSent	kbytesSent	kbytesReceived	TTL (Time to Live) Value	VLAN ID	AS (Autonomous System) Number
11187	0.514336	0.823265	0.089833	0.874875	0.054226	0.271128	0.992126	0.777778	0.784865
2581	0.101399	0.813773	0.038301	0.810811	0.987787	0.844651	0.110236	0.222222	0.690743
1089	0.591183	0.302504	0.288301	0.401401	0.714704	0.812408	0.188976	0.666667	0.458384
16141	0.555873	0.930920	0.096797	0.664665	0.710308	0.507572	0.976378	0.777778	0.648850
2557	0.602246	0.398456	0.700557	0.468468	0.008793	0.829995	0.188976	0.000000	0.833093

In [58]:
```
y.head()
```

Out[58]:
```
11187    0
2581     0
1089     0
16141    0
2557     0
Name: Attack Type, dtype: int64
```

In [59]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0,stratify=y)
```

Logistic Regression Undersampling

In [60]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

Out[60]:
```
▾ LogisticRegression
LogisticRegression()
```

In [61]:
```
model.score(x_test,y_test)
```

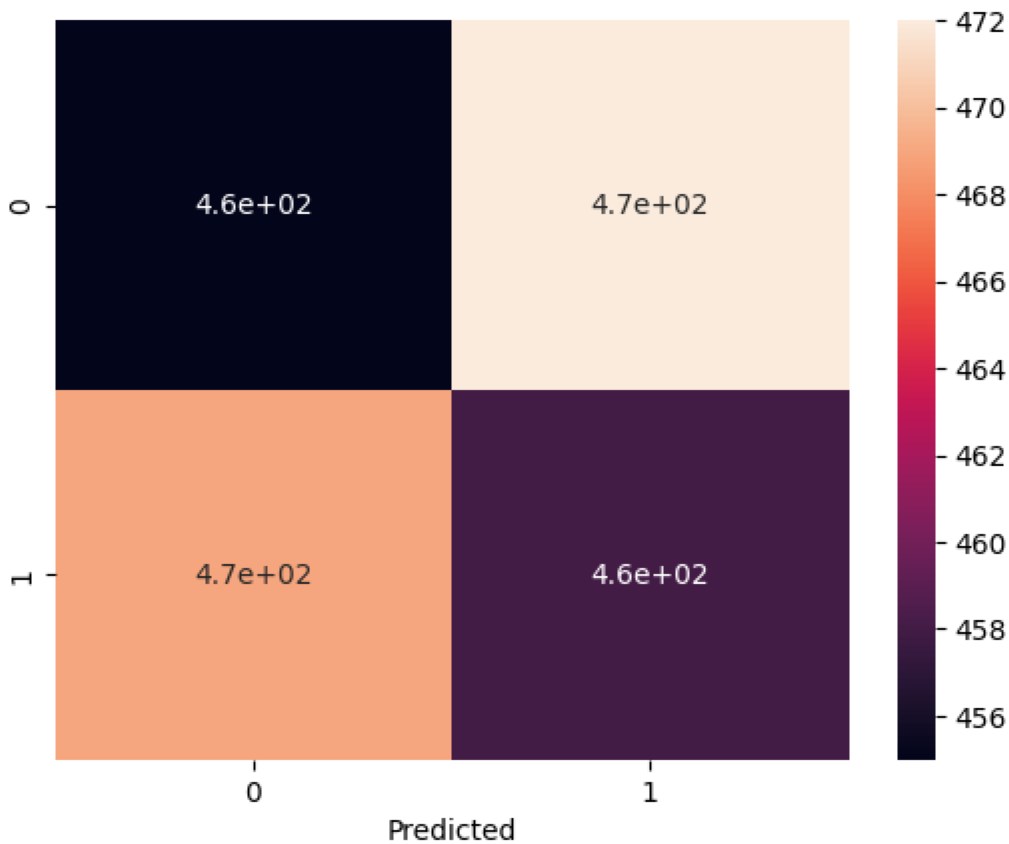
Out[61]: 0.4924487594390507

In [62]:
```
y_pred=model.predict(x_test)
```

In [63]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[63]:
```
array([[455, 472],
       [469, 458]])
```

In [64]:
```
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel=('True')
```



In [65]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.49      0.49      0.49       927
           1       0.49      0.49      0.49       927

    accuracy                           0.49      1854
   macro avg       0.49      0.49      0.49      1854
weighted avg       0.49      0.49      0.49      1854
```

Decision Tree Undersampling

In [66]:
```
model=tree.DecisionTreeClassifier()
model.fit(x_train,y_train)
```

Out[66]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [67]:
```
model.score(x_test,y_test)
```

Out[67]:
```
0.5102481121898598
```

In [68]:
```
y_pred=model.predict(x_test)
```

In [69]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[69]:
```
array([[487, 440],
       [468, 459]])
```

In [70]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.53      0.52       927
           1       0.51      0.50      0.50       927

    accuracy                           0.51      1854
   macro avg       0.51      0.51      0.51      1854
weighted avg       0.51      0.51      0.51      1854
```

Random Forest Undersampling

In [71]:
```
model=RandomForestClassifier()
model.fit(x_train,y_train)
```

Out[71]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [72]:
```
model.score(x_test,y_test)
```

Out[72]:
```
0.49083063646170444
```

In [73]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.53      0.52       927
           1       0.51      0.50      0.50       927

    accuracy                           0.51      1854
   macro avg       0.51      0.51      0.51      1854
weighted avg       0.51      0.51      0.51      1854
```

XGB Classifier Undersampling

In [74]:
```
model=XGBClassifier()
model.fit(x_train,y_train)
```

Out[74]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [75]:
```
model.score(x_test,y_test)
```

Out[75]: 0.5194174757281553

In [76]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.53      0.52       927
           1       0.51      0.50      0.50       927

    accuracy                           0.51      1854
   macro avg       0.51      0.51      0.51      1854
weighted avg       0.51      0.51      0.51      1854
```

Using Deep Learning. Artifical Neural Network Undersampling

In [77]:
```
from keras.engine.training import optimizer
from keras.api._v2.keras import activations
import tensorflow as tf
from tensorflow import keras
model=keras.Sequential([
    keras.layers.Dense(41,input_shape=(41,),activation='relu'),
    keras.layers.Dense(10,activation='relu'),
    keras.layers.Dense(1,activation='sigmoid')

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

model.fit(x_train,y_train,epochs=100)
```

```
Epoch 1/100
232/232 [==============================] - 2s 2ms/step - loss: 0.6958 - accuracy: 0.4968
Epoch 2/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6933 - accuracy: 0.5109
Epoch 3/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6919 - accuracy: 0.5209
Epoch 4/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6907 - accuracy: 0.5301
Epoch 5/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6894 - accuracy: 0.5329
Epoch 6/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6880 - accuracy: 0.5464
Epoch 7/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6858 - accuracy: 0.5503
Epoch 8/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6843 - accuracy: 0.5499
Epoch 9/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6817 - accuracy: 0.5598
Epoch 10/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6784 - accuracy: 0.5722
Epoch 11/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6760 - accuracy: 0.5788
Epoch 12/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6725 - accuracy: 0.5883
Epoch 13/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6690 - accuracy: 0.5936
Epoch 14/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6653 - accuracy: 0.5955
Epoch 15/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6620 - accuracy: 0.5975
Epoch 16/100
232/232 [==============================] - 1s 3ms/step - loss: 0.6583 - accuracy: 0.6068
Epoch 17/100
232/232 [==============================] - 1s 3ms/step - loss: 0.6551 - accuracy: 0.6155
Epoch 18/100
232/232 [==============================] - 1s 3ms/step - loss: 0.6511 - accuracy: 0.6186
Epoch 19/100
232/232 [==============================] - 1s 3ms/step - loss: 0.6483 - accuracy: 0.6207
Epoch 20/100
232/232 [==============================] - 1s 3ms/step - loss: 0.6441 - accuracy: 0.6265
Epoch 21/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6398 - accuracy: 0.6354
Epoch 22/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6376 - accuracy: 0.6373
Epoch 23/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6335 - accuracy: 0.6422
Epoch 24/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6311 - accuracy: 0.6422
Epoch 25/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6265 - accuracy: 0.6420
Epoch 26/100
232/232 [==============================] - 1s 2ms/step - loss: 0.6243 - accuracy: 0.6488
Epoch 27/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6207 - accuracy: 0.6523
Epoch 28/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6175 - accuracy: 0.6586
Epoch 29/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6145 - accuracy: 0.6577
Epoch 30/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6115 - accuracy: 0.6582
Epoch 31/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6075 - accuracy: 0.6675
```

```
Epoch 32/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6060 - accuracy: 0.6633
Epoch 33/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6047 - accuracy: 0.6705
Epoch 34/100
232/232 [==============================] - 0s 2ms/step - loss: 0.6004 - accuracy: 0.6726
Epoch 35/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5988 - accuracy: 0.6735
Epoch 36/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5965 - accuracy: 0.6784
Epoch 37/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5944 - accuracy: 0.6764
Epoch 38/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5907 - accuracy: 0.6814
Epoch 39/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5892 - accuracy: 0.6864
Epoch 40/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5874 - accuracy: 0.6830
Epoch 41/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5836 - accuracy: 0.6890
Epoch 42/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5825 - accuracy: 0.6875
Epoch 43/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5800 - accuracy: 0.6900
Epoch 44/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5785 - accuracy: 0.6909
Epoch 45/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5749 - accuracy: 0.6905
Epoch 46/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5739 - accuracy: 0.6925
Epoch 47/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5720 - accuracy: 0.6965
Epoch 48/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5697 - accuracy: 0.6992
Epoch 49/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5677 - accuracy: 0.6989
Epoch 50/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5674 - accuracy: 0.6976
Epoch 51/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5660 - accuracy: 0.6995
Epoch 52/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5635 - accuracy: 0.7042
Epoch 53/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5615 - accuracy: 0.7018
Epoch 54/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5592 - accuracy: 0.6995
Epoch 55/100
232/232 [==============================] - 1s 2ms/step - loss: 0.5569 - accuracy: 0.7089
Epoch 56/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5564 - accuracy: 0.7047
Epoch 57/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5555 - accuracy: 0.7105
Epoch 58/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5532 - accuracy: 0.7104
Epoch 59/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5520 - accuracy: 0.7100
Epoch 60/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5523 - accuracy: 0.7104
Epoch 61/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5491 - accuracy: 0.7126
Epoch 62/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5488 - accuracy: 0.7118
```

```
Epoch 63/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5441 - accuracy: 0.7196
Epoch 64/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5448 - accuracy: 0.7170
Epoch 65/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5432 - accuracy: 0.7185
Epoch 66/100
232/232 [==============================] - 1s 2ms/step - loss: 0.5425 - accuracy: 0.7194
Epoch 67/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5398 - accuracy: 0.7196
Epoch 68/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5405 - accuracy: 0.7199
Epoch 69/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5389 - accuracy: 0.7215
Epoch 70/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5375 - accuracy: 0.7263
Epoch 71/100
232/232 [==============================] - 2s 7ms/step - loss: 0.5348 - accuracy: 0.7227
Epoch 72/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5352 - accuracy: 0.7258
Epoch 73/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5343 - accuracy: 0.7261
Epoch 74/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5316 - accuracy: 0.7277
Epoch 75/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5322 - accuracy: 0.7269
Epoch 76/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5314 - accuracy: 0.7257
Epoch 77/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5290 - accuracy: 0.7285
Epoch 78/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5280 - accuracy: 0.7329
Epoch 79/100
232/232 [==============================] - 1s 2ms/step - loss: 0.5284 - accuracy: 0.7327
Epoch 80/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5252 - accuracy: 0.7328
Epoch 81/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5243 - accuracy: 0.7328
Epoch 82/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5244 - accuracy: 0.7370
Epoch 83/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5240 - accuracy: 0.7336
Epoch 84/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5219 - accuracy: 0.7319
Epoch 85/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5215 - accuracy: 0.7400
Epoch 86/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5216 - accuracy: 0.7339
Epoch 87/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5196 - accuracy: 0.7381
Epoch 88/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5175 - accuracy: 0.7370
Epoch 89/100
232/232 [==============================] - 1s 2ms/step - loss: 0.5179 - accuracy: 0.7402
Epoch 90/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5162 - accuracy: 0.7387
Epoch 91/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5146 - accuracy: 0.7385
Epoch 92/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5142 - accuracy: 0.7402
Epoch 93/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5133 - accuracy: 0.7412
```

```
Epoch 94/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5104 - accuracy: 0.7428
Epoch 95/100
232/232 [==============================] - 1s 3ms/step - loss: 0.5106 - accuracy: 0.7408
Epoch 96/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5111 - accuracy: 0.7466
Epoch 97/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5111 - accuracy: 0.7420
Epoch 98/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5080 - accuracy: 0.7468
Epoch 99/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5080 - accuracy: 0.7476
Epoch 100/100
232/232 [==============================] - 0s 2ms/step - loss: 0.5068 - accuracy: 0.7464
```

Out[77]:
```
<keras.callbacks.History at 0x7c74a8c064a0>
```

In [78]:
```
model.evaluate(x_test,y_test)
```

```
58/58 [==============================] - 0s 2ms/step - loss: 0.9767 - accuracy: 0.5092
```

Out[78]:
```
[0.9767211079597473, 0.509169340133667]
```

In [79]:
```
y_pred=model.predict(x_test)
```

```
58/58 [==============================] - 0s 1ms/step
```

In [80]:
```
y_pred[:5]
```

Out[80]:
```
array([[0.47612187],
       [0.57068855],
       [0.32112825],
       [0.6958266 ],
       [0.25755513]], dtype=float32)
```

In [81]:
```
yp=[]
for i in y_pred:
  if i > 0.5:
    yp.append(1)
  else:
    yp.append(0)
```

In [82]:
```
print(classification_report(y_test,yp))
```

```
              precision    recall  f1-score   support

           0       0.51      0.56      0.53       927
           1       0.51      0.46      0.48       927

    accuracy                           0.51      1854
   macro avg       0.51      0.51      0.51      1854
weighted avg       0.51      0.51      0.51      1854
```

In [83]:
```
print(confusion_matrix(y_test,yp))
```

```
[[519 408]
 [502 425]]
```

Minority Oversampling

In [84]:
```
data_count_0=data[data['Attack Type']==0]
data_count_1=data[data['Attack Type']==1]
```

In [85]:
```
count_class_0,count_class_1=data['Attack Type'].value_counts()
count_class_0,count_class_1
```

Out[85]:
```
(82904, 4634)
```

In [86]:
```
data_over=data_count_1.sample(count_class_0, replace=True)

data_over_1=pd.concat([data_count_0,data_over],axis=0)
data_over_1['Attack Type'].value_counts()
```

Out[86]:
```
0    82904
1    82904
Name: Attack Type, dtype: int64
```

In [87]:
```
x=data_over_1.drop('Attack Type',axis=1)
y=data_over_1['Attack Type']
```

In [88]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0,stratify=y)
```

Logistic Regression Oversampling

In [89]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

Out[89]:
```
▾ LogisticRegression
LogisticRegression()
```

In [90]:
```
model.score(x_test,y_test)
```

Out[90]:
```
0.5099813039020565
```

In [91]:
```
y_pred=model.predict(x_test)
```

In [92]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.51      0.50      0.50     16581
           1       0.51      0.52      0.52     16581

    accuracy                           0.51     33162
   macro avg       0.51      0.51      0.51     33162
weighted avg       0.51      0.51      0.51     33162
```

Decision Tree Oversampling

In [93]:
```
model=tree.DecisionTreeClassifier()
model.fit(x_train,y_train)
```

Out[93]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [94]:
```
model.score(x_train,y_train)
```

Out[94]:
```
1.0
```

In [95]:
```
model.score(x_test,y_test)
```

Out[95]:
```
0.9672818285989988
```

In [96]:
```
y_pred=model.predict(x_test)
y_pred[:5]
```

Out[96]:
```
array([1, 0, 1, 1, 0])
```

In [97]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.97     16581
           1       0.94      1.00      0.97     16581

    accuracy                           0.97     33162
   macro avg       0.97      0.97      0.97     33162
weighted avg       0.97      0.97      0.97     33162
```

Random Forest Oversampling

In [98]:
```
model=RandomForestClassifier()
model.fit(x_train,y_train)
```

Out[98]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [99]:
```
model.score(x_train,y_train)
```

Out[99]:
```
1.0
```

In [100…
```
model.score(x_test,y_test)
```

Out[100]:
```
1.0
```

In [101…
```
y_pred=model.predict(x_test)
```

In [102…
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     16581
           1       1.00      1.00      1.00     16581

    accuracy                           1.00     33162
   macro avg       1.00      1.00      1.00     33162
weighted avg       1.00      1.00      1.00     33162
```

XGB Classifier Oversampling

In [103…
```
model=XGBClassifier()
model.fit(x_train,y_train)
```

Out[103]:
```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [104…
```
model.score(x_test,y_test)
```

Out[104]: 0.8610156202882817

In [105…
```
y_pred=model.predict(x_test)
```

In [106…
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.81      0.85     16581
           1       0.83      0.91      0.87     16581

    accuracy                           0.86     33162
   macro avg       0.87      0.86      0.86     33162
weighted avg       0.87      0.86      0.86     33162
```

Artificial Neural network Oversampling

In [107…
```
model=keras.Sequential([
    keras.layers.Dense(41,input_shape=(41,),activation='relu'),
    keras.layers.Dense(10,activation='relu'),
    keras.layers.Dense(1,activation='sigmoid')

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

```
model.fit(x_train,y_train,epochs=100)
```

```
Epoch 1/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.6895 - accuracy: 0.5307
Epoch 2/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6744 - accuracy: 0.5759
Epoch 3/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6571 - accuracy: 0.6059
Epoch 4/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.6435 - accuracy: 0.6245
Epoch 5/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6321 - accuracy: 0.6390
Epoch 6/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6233 - accuracy: 0.6484
Epoch 7/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.6165 - accuracy: 0.6563
Epoch 8/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6105 - accuracy: 0.6612
Epoch 9/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6051 - accuracy: 0.6666
Epoch 10/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.6001 - accuracy: 0.6700
Epoch 11/100
4146/4146 [==============================] - 12s 3ms/step - loss: 0.5959 - accuracy: 0.6741
Epoch 12/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5923 - accuracy: 0.6771
Epoch 13/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5884 - accuracy: 0.6807
Epoch 14/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5863 - accuracy: 0.6809
Epoch 15/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5836 - accuracy: 0.6829
Epoch 16/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5809 - accuracy: 0.6864
Epoch 17/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5783 - accuracy: 0.6871
Epoch 18/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5761 - accuracy: 0.6884
Epoch 19/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5743 - accuracy: 0.6897
Epoch 20/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5723 - accuracy: 0.6911
Epoch 21/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5701 - accuracy: 0.6925
Epoch 22/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5681 - accuracy: 0.6944
Epoch 23/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5667 - accuracy: 0.6961
Epoch 24/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5647 - accuracy: 0.6964
Epoch 25/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5636 - accuracy: 0.6980
Epoch 26/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5623 - accuracy: 0.6975
Epoch 27/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5606 - accuracy: 0.7000
Epoch 28/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5595 - accuracy: 0.7004
Epoch 29/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5587 - accuracy: 0.7014
Epoch 30/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5575 - accuracy: 0.7019
Epoch 31/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5562 - accuracy: 0.7030
```

```
Epoch 32/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5549 - accuracy: 0.7037
Epoch 33/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5545 - accuracy: 0.7054
Epoch 34/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5539 - accuracy: 0.7057
Epoch 35/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5528 - accuracy: 0.7060
Epoch 36/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5524 - accuracy: 0.7061
Epoch 37/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5519 - accuracy: 0.7071
Epoch 38/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5513 - accuracy: 0.7068
Epoch 39/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5500 - accuracy: 0.7067
Epoch 40/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5499 - accuracy: 0.7065
Epoch 41/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5490 - accuracy: 0.7091
Epoch 42/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5489 - accuracy: 0.7079
Epoch 43/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5475 - accuracy: 0.7093
Epoch 44/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5474 - accuracy: 0.7082
Epoch 45/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5471 - accuracy: 0.7095
Epoch 46/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5461 - accuracy: 0.7106
Epoch 47/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5456 - accuracy: 0.7111
Epoch 48/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5451 - accuracy: 0.7120
Epoch 49/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5447 - accuracy: 0.7122
Epoch 50/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5442 - accuracy: 0.7130
Epoch 51/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5433 - accuracy: 0.7135
Epoch 52/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5436 - accuracy: 0.7148
Epoch 53/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5427 - accuracy: 0.7132
Epoch 54/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5420 - accuracy: 0.7149
Epoch 55/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5417 - accuracy: 0.7147
Epoch 56/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5417 - accuracy: 0.7154
Epoch 57/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5408 - accuracy: 0.7157
Epoch 58/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5405 - accuracy: 0.7164
Epoch 59/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5398 - accuracy: 0.7167
Epoch 60/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5394 - accuracy: 0.7159
Epoch 61/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5391 - accuracy: 0.7160
Epoch 62/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5383 - accuracy: 0.7168
```

```
Epoch 63/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5381 - accuracy: 0.7163
Epoch 64/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5373 - accuracy: 0.7173
Epoch 65/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5374 - accuracy: 0.7177
Epoch 66/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5365 - accuracy: 0.7192
Epoch 67/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5362 - accuracy: 0.7183
Epoch 68/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5366 - accuracy: 0.7194
Epoch 69/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5368 - accuracy: 0.7175
Epoch 70/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5355 - accuracy: 0.7179
Epoch 71/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5353 - accuracy: 0.7202
Epoch 72/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5354 - accuracy: 0.7190
Epoch 73/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5347 - accuracy: 0.7193
Epoch 74/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5349 - accuracy: 0.7200
Epoch 75/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5344 - accuracy: 0.7193
Epoch 76/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5332 - accuracy: 0.7206
Epoch 77/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5332 - accuracy: 0.7203
Epoch 78/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5331 - accuracy: 0.7204
Epoch 79/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5327 - accuracy: 0.7206
Epoch 80/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5319 - accuracy: 0.7201
Epoch 81/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5318 - accuracy: 0.7214
Epoch 82/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5319 - accuracy: 0.7211
Epoch 83/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5315 - accuracy: 0.7216
Epoch 84/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5307 - accuracy: 0.7218
Epoch 85/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5305 - accuracy: 0.7229
Epoch 86/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5298 - accuracy: 0.7219
Epoch 87/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5303 - accuracy: 0.7217
Epoch 88/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5299 - accuracy: 0.7228
Epoch 89/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5293 - accuracy: 0.7235
Epoch 90/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5289 - accuracy: 0.7229
Epoch 91/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5289 - accuracy: 0.7230
Epoch 92/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5290 - accuracy: 0.7226
Epoch 93/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5281 - accuracy: 0.7231
```

```
Epoch 94/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5282 - accuracy: 0.7235
Epoch 95/100
4146/4146 [==============================] - 8s 2ms/step - loss: 0.5273 - accuracy: 0.7244
Epoch 96/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5275 - accuracy: 0.7245
Epoch 97/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5279 - accuracy: 0.7233
Epoch 98/100
4146/4146 [==============================] - 10s 2ms/step - loss: 0.5279 - accuracy: 0.7232
Epoch 99/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5269 - accuracy: 0.7240
Epoch 100/100
4146/4146 [==============================] - 9s 2ms/step - loss: 0.5270 - accuracy: 0.7239
```

Out[107]:
```
<keras.callbacks.History at 0x7c74a4da1900>
```

In [108…
```
model.evaluate(x_test,y_test)
```

```
1037/1037 [==============================] - 2s 2ms/step - loss: 0.5577 - accuracy: 0.7004
```

Out[108]:
```
[0.5576631426811218, 0.7004402875900269]
```

In [109…
```
y_pred=model.predict(x_test)
```

```
1037/1037 [==============================] - 2s 2ms/step
```

In [ ]:
```
yp=[]
for i in y_pred:
  if i > 0.5:
    yp.append(1)
  else:
    yp.append(0)
```

In [111…
```
print(classification_report(y_test,yp))
```

```
              precision    recall  f1-score   support

           0       0.76      0.58      0.66     16581
           1       0.66      0.82      0.73     16581

    accuracy                           0.70     33162
   macro avg       0.71      0.70      0.70     33162
weighted avg       0.71      0.70      0.70     33162
```


```
              precision    recall  f1-score   support

           0       0.76      0.58      0.66     16581
           1       0.66      0.82      0.73     16581

    accuracy                           0.70     33162
   macro avg       0.71      0.70      0.70     33162
weighted avg       0.71      0.70      0.70     33162
```


From our models, we can see that RandomForest has the highest accuracy followed by DecisionTree Classifier.

Also, we could see that, minority oversampling as a method of handling imbalanced dataset performed better than the majority undersampling, this could be cos when undersampling, the dataset lost more data for training ability.