```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
```

```
df = pd.read_csv('Network_anomaly_data.csv')
```

Problem Statement: Identify and predict anamolies in network connection. Provide insights and recommendation to minimze the risk.

```
df.iloc[:, :20]
```

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	hot	numfailedlogins	loggedin	numcompromised	rootshell	suattempted	numroot
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0
1	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0
2	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	0	0
4	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
125968	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0
125969	8	udp	private	SF	105	145	0	0	0	0	0	0	0	0	0	0
125970	0	tcp	smtp	SF	2231	384	0	0	0	0	0	1	0	0	0	0
125971	0	tcp	klogin	S0	0	0	0	0	0	0	0	0	0	0	0	0
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0	0	1	0	0	0	0

125973 rows × 20 columns

```
df.iloc[:, 19:37]
```

	numoutboundcmds	ishostlogin	isguestlogin	count	srvcount	serrorrate	srvserrorrate	rerrorrate	srvrerrorrate	samesrvrate	diffsrvrate	srvdiffhostrate	dsthostcou
0	0	0	0	2	2	0.0	0.0	0.0	0.0	1.00	0.00	0.00	1
1	0	0	0	13	1	0.0	0.0	0.0	0.0	0.08	0.15	0.00	2
2	0	0	0	123	6	1.0	1.0	0.0	0.0	0.05	0.07	0.00	2
3	0	0	0	5	5	0.2	0.2	0.0	0.0	1.00	0.00	0.00	
4	0	0	0	30	32	0.0	0.0	0.0	0.0	1.00	0.00	0.09	2
...	...	...	...	...	...	...	...	...	...	...	...	...	
125968	0	0	0	184	25	1.0	1.0	0.0	0.0	0.14	0.06	0.00	2
125969	0	0	0	2	2	0.0	0.0	0.0	0.0	1.00	0.00	0.00	2
125970	0	0	0	1	1	0.0	0.0	0.0	0.0	1.00	0.00	0.00	2
125971	0	0	0	144	8	1.0	1.0	0.0	0.0	0.06	0.05	0.00	2
125972	0	0	0	1	1	0.0	0.0	0.0	0.0	1.00	0.00	0.00	2

125973 rows × 18 columns

```
df.iloc[:, 36:]
```

	dsthostsrvdiffhostrate	dsthostserrorrate	dsthostsrvserrorrate	dsthostrerrorrate	dsthostsrvrerrorrate	attack	lastflag
0	0.00	0.00	0.00	0.05	0.00	normal	20
1	0.00	0.00	0.00	0.00	0.00	normal	15
2	0.00	1.00	1.00	0.00	0.00	neptune	19
3	0.04	0.03	0.01	0.00	0.01	normal	21
4	0.00	0.00	0.00	0.00	0.00	normal	21
...	...	...	...	...	...	...	...
125968	0.00	1.00	1.00	0.00	0.00	neptune	20
125969	0.00	0.00	0.00	0.00	0.00	normal	21
125970	0.00	0.72	0.00	0.01	0.00	normal	18
125971	0.00	1.00	1.00	0.00	0.00	neptune	20
125972	0.00	0.00	0.00	0.00	0.00	normal	21

125973 rows × 7 columns

EDA

```
df['attack'].value_counts()
```

```
attack
normal             67343
neptune            41214
satan               3633
ipsweep             3599
portsweep           2931
smurf               2646
nmap                1493
back                 956
teardrop             892
warezclient          890
pod                  201
guess_passwd          53
buffer_overflow       30
warezmaster           20
land                  18
imap                  11
rootkit               10
loadmodule             9
ftp_write              8
multihop               7
phf                    4
perl                   3
spy                    2
Name: count, dtype: int64
```

```
df['attack?'] = np.where(df['attack']=="normal",0,1)
```

```
df['attack?']
```

```
0         0
1         0
2         1
3         0
4         0
         ..
125968    1
125969    0
125970    0
125971    1
125972    0
Name: attack?, Length: 125973, dtype: int64
```

```
df['attack?'].value_counts()
## We see that minority class is not too small as compared to the majority class
```

```
attack?
0    67343
1    58630
Name: count, dtype: int64
```

```
df['protocoltype'].value_counts()
```

```
protocoltype
tcp     102689
udp      14993
```

```
icmp      8291
Name: count, dtype: int64
```

```
df['flag'].value_counts()
```

```
flag
SF        74945
S0        34851
REJ       11233
RSTR       2421
RSTO       1562
S1          365
SH          271
S2          127
RSTOS0      103
S3           49
OTH          46
Name: count, dtype: int64
```

```
df['service'].value_counts()
```

```
service
http         40338
private      21853
domain_u      9043
smtp          7313
ftp_data      6860
             ...
tftp_u           3
http_8001        2
aol              2
harvest          2
http_2784        1
Name: count, Length: 70, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125973 entries, 0 to 125972
Data columns (total 44 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   duration                125973 non-null  int64
 1   protocoltype            125973 non-null  object
 2   service                 125973 non-null  object
 3   flag                    125973 non-null  object
 4   srcbytes                125973 non-null  int64
 5   dstbytes                125973 non-null  int64
 6   land                    125973 non-null  int64
 7   wrongfragment           125973 non-null  int64
 8   urgent                  125973 non-null  int64
 9   hot                     125973 non-null  int64
 10  numfailedlogins         125973 non-null  int64
 11  loggedin                125973 non-null  int64
 12  numcompromised          125973 non-null  int64
 13  rootshell               125973 non-null  int64
 14  suattempted             125973 non-null  int64
```

```
 15  numroot                125973 non-null  int64
 16  numfilecreations       125973 non-null  int64
 17  numshells              125973 non-null  int64
 18  numaccessfiles         125973 non-null  int64
 19  numoutboundcmds        125973 non-null  int64
 20  ishostlogin            125973 non-null  int64
 21  isguestlogin           125973 non-null  int64
 22  count                  125973 non-null  int64
 23  srvcount               125973 non-null  int64
 24  serrorrate             125973 non-null  float64
 25  srvserrorrate          125973 non-null  float64
 26  rerrorrate             125973 non-null  float64
 27  srvrerrorrate          125973 non-null  float64
 28  samesrvrate            125973 non-null  float64
 29  diffsrvrate            125973 non-null  float64
 30  srvdiffhostrate        125973 non-null  float64
 31  dsthostcount           125973 non-null  int64
 32  dsthostsrvcount        125973 non-null  int64
 33  dsthostsamesrvrate     125973 non-null  float64
 34  dsthostdiffsrvrate     125973 non-null  float64
 35  dsthostsamesrcportrate  125973 non-null  float64
 36  dsthostsrvdiffhostrate  125973 non-null  float64
 37  dsthostserrorrate      125973 non-null  float64
 38  dsthostsrvserrorrate   125973 non-null  float64
 39  dsthostrerrorrate      125973 non-null  float64
 40  dsthostsrvrerrorrate   125973 non-null  float64
 41  attack                 125973 non-null  object
 42  lastflag               125973 non-null  int64
 43  attack?                125973 non-null  int64
dtypes: float64(15), int64(25), object(4)
memory usage: 42.3+ MB
```

```
df.shape
```

```
(125973, 44)
```

```
df.describe()
```

	duration	srcbytes	dstbytes	land	wrongfragment	urgent	hot	numfailedlogins	loggedin	numcompromised	...	dsthostsamesrvrate	dstl
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	...	125973.000000	
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.022687	0.000111	0.204409	0.001222	0.395736	0.279250	...	0.521242	
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.253530	0.014366	2.149968	0.045239	0.489010	23.942042	...	0.448949	
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.050000	
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.510000	
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	...	1.000000	
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	77.000000	5.000000	1.000000	7479.000000	...	1.000000	

8 rows × 40 columns

```
df['wrongfragment'].value_counts()
```

```
wrongfragment
0    124883
3       884
1       206
Name: count, dtype: int64
```

Univariate Analysis: Distribution of each variable

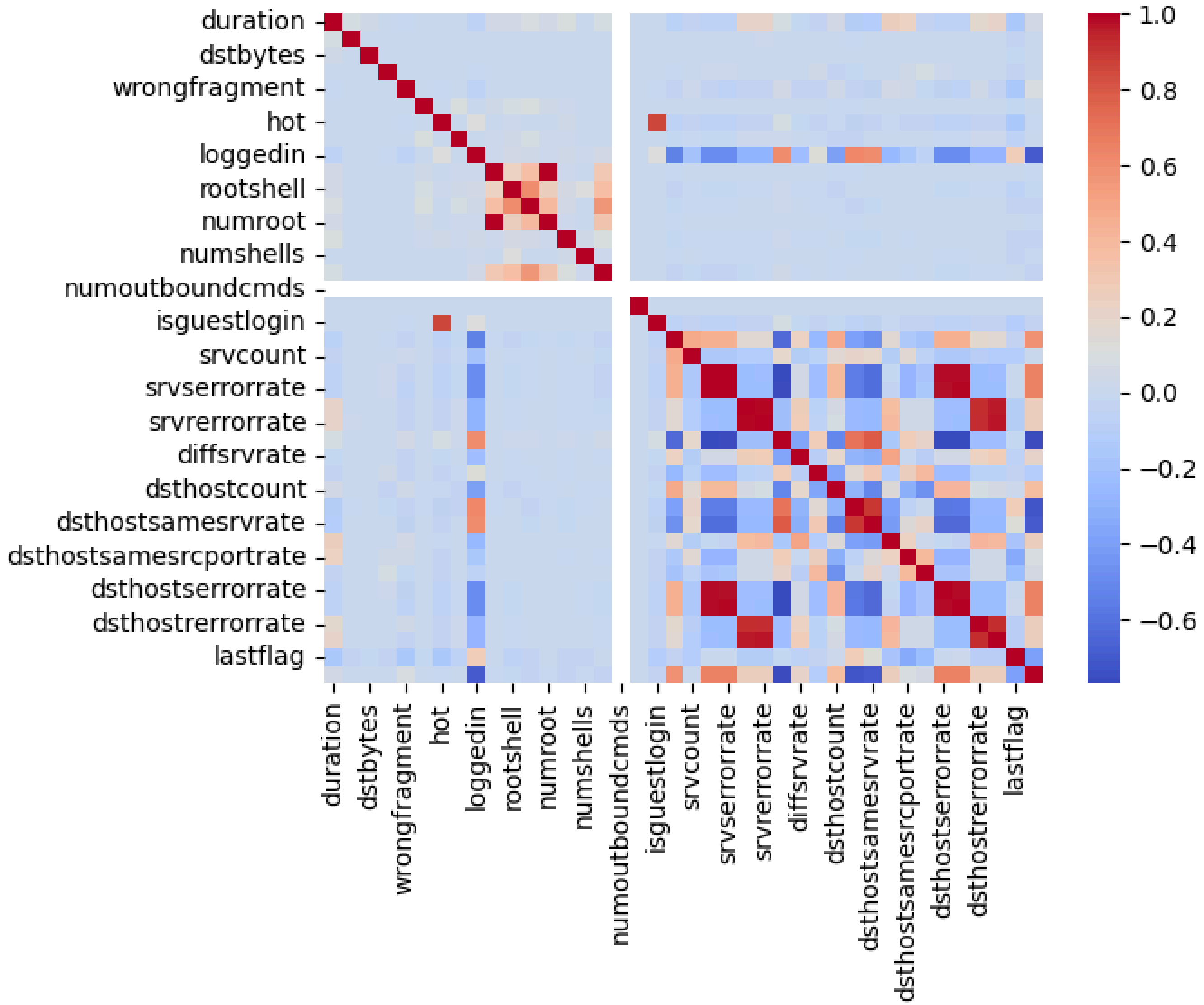
```
for col in df.describe().columns:
    sns.kdeplot(df[col])
    plt.show()

## We see that most of the plots are very much skewed because of outliers
```

```
for col in df.describe().columns:
    sns.boxplot(df[col])
    plt.show()
```

```
## Let us try and transform the data so that the outliers will be nullified and their impact will be reduced
from scipy.stats import boxcox
transformed_data = pd.DataFrame()
for column in df.describe().columns:
    a = df[column] - df[column].min() +1
    transformed_data[column], fitted_lambda = boxcox(a)
    sns.boxplot(transformed_data[column])
    plt.show()
```

```
## Heatmap showing correlation between each column
numeric_col = df.describe().columns
sns.heatmap(df[numeric_col].corr(), cmap = 'coolwarm')
```

```
<Axes: >
```




```
cor = df[numeric_col].corr()
cor
```

	duration	srcbytes	dstbytes	land	wrongfragment	urgent	hot	numfailedlogins	loggedin	numcompromised	...	dsthostsamesrvrate	dsthostdiffsr
duration	1.000000	0.070737	0.034878	-0.001553	-0.009866	0.003830	0.000705	0.009528	-0.064218	0.042679	...	-0.116005	0.2
srcbytes	0.070737	1.000000	0.000204	-0.000109	-0.000693	-0.000059	0.000295	-0.000208	-0.003353	-0.000086	...	-0.006572	0.0
dstbytes	0.034878	0.000204	1.000000	-0.000069	-0.000440	0.000248	-0.000344	0.000504	-0.002894	0.001233	...	-0.004424	0.0
land	-0.001553	-0.000109	-0.000069	1.000000	-0.001261	-0.000109	-0.001340	-0.000381	-0.011402	-0.000164	...	0.011597	-0.0
wrongfragment	-0.009866	-0.000693	-0.000440	-0.001261	1.000000	-0.000692	-0.008508	-0.002418	-0.072418	-0.001044	...	-0.048733	0.0
urgent	0.003830	-0.000059	0.000248	-0.000109	-0.000692	1.000000	0.000293	0.097507	0.007299	0.033329	...	-0.004489	0.0
hot	0.000705	0.000295	-0.000344	-0.001340	-0.008508	0.000293	1.000000	0.003715	0.116435	0.002014	...	-0.036293	-0.0
numfailedlogins	0.009528	-0.000208	0.000504	-0.000381	-0.002418	0.097507	0.003715	1.000000	-0.006439	0.019085	...	-0.001576	-0.0
loggedin	-0.064218	-0.003353	-0.002894	-0.011402	-0.072418	0.007299	0.116435	-0.006439	1.000000	0.014413	...	0.604058	-0.2
numcompromised	0.042679	-0.000086	0.001233	-0.000164	-0.001044	0.033329	0.002014	0.019085	0.014413	1.000000	...	-0.004995	0.0
rootshell	0.052791	-0.000272	0.001069	-0.000516	-0.003280	0.075199	0.015379	0.032567	0.045290	0.224872	...	0.007608	-0.0
suattempted	0.087183	-0.000186	0.001133	-0.000344	-0.002187	0.097710	0.000130	0.073175	0.030196	0.362702	...	-0.015606	0.0
numroot	0.045519	-0.000093	0.001229	-0.000174	-0.001108	0.032470	0.001510	0.018112	0.015304	0.998833	...	-0.005918	0.0
numfilecreations	0.099116	-0.000179	0.000089	-0.000369	-0.002343	0.024918	0.028716	0.021774	0.032283	0.015976	...	-0.017325	0.0
numshells	-0.001593	-0.000134	-0.000083	-0.000262	-0.001665	-0.000144	0.004723	-0.000503	0.022996	0.001338	...	-0.006134	-0.0
numaccessfiles	0.070420	-0.000309	0.000339	-0.000581	-0.003689	0.010803	-0.001987	0.000652	0.050937	0.299631	...	0.006925	0.0
numoutboundcmds	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
ishostlogin	-0.000258	-0.000022	-0.000008	-0.000040	-0.000252	-0.000022	0.001043	-0.000076	0.003482	0.001144	...	-0.002832	0.0
isguestlogin	0.000440	-0.000742	-0.000421	-0.001374	-0.008728	-0.000754	0.860288	0.006446	0.119678	-0.001138	...	-0.054297	-0.0
count	-0.079042	-0.005152	-0.003543	-0.009837	-0.020819	-0.005615	-0.068697	-0.019544	-0.539754	-0.008434	...	-0.473957	0.1
srvcount	-0.039470	-0.002792	-0.001754	-0.005031	0.024457	-0.002848	-0.034575	-0.009880	-0.199744	-0.004279	...	0.181116	-0.1
serrorrate	-0.069873	-0.003228	-0.003059	0.021734	-0.043316	-0.004929	-0.059083	-0.015254	-0.491925	-0.005297	...	-0.622797	-0.0
srvserrorrate	-0.069510	-0.003438	-0.003038	0.022614	-0.056549	-0.004889	-0.058713	-0.015899	-0.490167	-0.005278	...	-0.619130	-0.0
rerrorrate	0.200682	0.013782	0.011176	-0.004096	-0.033052	-0.002896	-0.032382	0.022193	-0.287514	-0.003682	...	-0.257613	0.3
srvrerrorrate	0.199961	0.013975	0.011052	-0.005275	-0.033507	-0.002897	-0.031436	0.021870	-0.283532	-0.003642	...	-0.255565	0.3
samesrvrate	0.074681	0.003899	0.003788	0.008739	0.054759	0.005967	0.069365	0.019477	0.600536	0.008944	...	0.788978	-0.1
diffsrvrate	-0.013738	-0.000432	-0.001703	-0.001551	-0.026638	-0.002705	-0.016212	-0.004438	-0.221323	-0.004035	...	-0.330735	0.4
srvdiffhostrate	-0.040158	-0.002608	-0.001674	0.038102	-0.026247	-0.002898	-0.026781	-0.010122	0.131074	-0.004227	...	0.291418	-0.
dsthostcount	0.050570	-0.005791	0.002528	-0.025499	0.041056	-0.006941	-0.012249	-0.025476	-0.401084	-0.010928	...	-0.518145	0.1
dsthostsrvcount	-0.109776	-0.006861	-0.004224	-0.014159	-0.045240	-0.007897	-0.051864	-0.023053	0.624365	-0.010321	...	0.896663	-0.3
dsthostsamesrvrate	-0.116005	-0.006572	-0.004424	0.011597	-0.048733	-0.004489	-0.036293	-0.001576	0.604058	-0.004995	...	1.000000	-0.4
dsthostdiffsrvrate	0.254195	0.000900	0.011031	-0.004516	0.059797	0.006840	-0.012293	-0.001945	-0.256065	0.002981	...	-0.419341	1.0

dsthostsamesrcportrate	0.228737	0.000431	0.011747	0.033851	0.037177	0.002741	-0.034536	-0.005526	-0.160994	-0.002045	...	0.135946	0.:
dsthostsrvdiffhostrate	-0.026669	-0.001655	-0.001281	0.070474	-0.016252	0.005176	-0.024715	0.003302	-0.055953	0.004252	...	0.199187	0.(
dsthostserrorrate	-0.064948	-0.004503	-0.003024	0.019840	-0.051917	-0.004749	-0.058222	-0.011648	-0.491478	-0.004377	...	-0.639205	-0.(
dsthostsrvserrorrate	-0.064361	-0.003397	-0.002944	0.012276	-0.055917	-0.004834	-0.058214	-0.012299	-0.493264	-0.004898	...	-0.632048	-0.(
dsthostrerrorrate	0.173815	-0.001468	0.011729	-0.005222	0.028890	-0.002999	-0.030555	0.018660	-0.275972	-0.003647	...	-0.257178	0.4
dsthostsrvrerrorrate	0.199024	0.012449	0.011223	-0.005303	-0.033682	-0.002912	-0.031670	0.017359	-0.272806	-0.003219	...	-0.258147	0.:
lastflag	-0.156311	-0.022592	-0.018076	-0.037038	-0.157130	-0.016411	-0.160013	-0.073178	0.269818	-0.006101	...	0.126981	-0.1
attack?	0.048785	0.005921	0.004118	0.007191	0.095905	-0.002787	-0.013083	-0.003755	-0.690171	-0.010198	...	-0.693803	0.:

40 rows × 40 columns

```
## Let us see which columns have the highest correlation with other columns one by one.
for col in cor.columns:
    print(cor[col].sort_values(ascending = False)[1:4])
    print("-------------------------------------")
```

```
dsthostsrvrerrorrate    0.278103
srvrerrorrate           0.275323
Name: diffsrvrate, dtype: float64
-------------------------------------
dsthostsrvdiffhostrate    0.379677
dsthostsamesrvrate        0.291418
samesrvrate               0.277155
Name: srvdiffhostrate, dtype: float64
-------------------------------------
count                   0.468092
dsthostserrorrate       0.410922
dsthostsrvserrorrate    0.407532
Name: dsthostcount, dtype: float64
-------------------------------------
dsthostsamesrvrate    0.896663
samesrvrate           0.705410
loggedin              0.624365
Name: dsthostsrvcount, dtype: float64
-------------------------------------
dsthostsrvcount    0.896663
samesrvrate        0.788978
loggedin           0.604058
```

```
for col in cor.columns:
    print(cor[col].sort_values(ascending = False)[-4:-1])
    print("-------------------------------------")
```

```
-------------------------------------
dsthostsrvcount      -0.567594
dsthostsamesrvrate   -0.632048
samesrvrate          -0.765322
Name: dsthostsrvserrorrate, dtype: float64
-------------------------------------
dsthostsrvcount      -0.250307
dsthostsamesrvrate   -0.257178
loggedin             -0.275972
Name: dsthostrerrorrate, dtype: float64
-------------------------------------
dsthostsrvcount      -0.252848
dsthostsamesrvrate   -0.258147
loggedin             -0.272806
Name: dsthostsrvrerrorrate, dtype: float64
-------------------------------------
dsthostsrvdiffhostrate   -0.232510
dsthostsamesrcportrate   -0.341422
attack?                  -0.379707
Name: lastflag, dtype: float64
-------------------------------------
dsthostsamesrvrate   -0.693803
dsthostsrvcount      -0.722535
samesrvrate          -0.751913
Name: attack?, dtype: float64
-------------------------------------
```

Treating missing values

```
df.isna().sum().sum()
## Fortunately there are no missing values so we need not worry about it
```

```
0
```

Treating outliers and feature engineering

```
df
```

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	hot	...	dsthostdiffsrvrate	dsthostsamesrcportrate	dsthostsrvdiffhostrate	ds
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	0.03	0.17	0.00	
1	0	udp	other	SF	146	0	0	0	0	0	...	0.60	0.88	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.05	0.00	0.00	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	0.00	0.03	0.04	
4	0	tcp	http	SF	199	420	0	0	0	0	...	0.00	0.00	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
125968	0	tcp	private	S0	0	0	0	0	0	0	...	0.06	0.00	0.00	
125969	8	udp	private	SF	105	145	0	0	0	0	...	0.01	0.01	0.00	
125970	0	tcp	smtp	SF	2231	384	0	0	0	0	...	0.06	0.00	0.00	
125971	0	tcp	klogin	S0	0	0	0	0	0	0	...	0.05	0.00	0.00	
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0	...	0.03	0.30	0.00	

125973 rows × 44 columns

```
df.describe().iloc[:, 27:]
```

	srvdiffhostrate	dsthostcount	dsthostsrvcount	dsthostsamesrvrate	dsthostdiffsrvrate	dsthostsamesrcportrate	dsthostsrvdiffhostrate	dsthostserrorrate	dsthostsrvse
count	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	1259
mean	0.097322	182.148945	115.653005	0.521242	0.082951	0.148379	0.032542	0.284452	
std	0.259830	99.206213	110.702741	0.448949	0.188922	0.308997	0.112564	0.444784	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	82.000000	10.000000	0.050000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	255.000000	63.000000	0.510000	0.020000	0.000000	0.000000	0.000000	
75%	0.000000	255.000000	255.000000	1.000000	0.070000	0.060000	0.020000	1.000000	
max	1.000000	255.000000	255.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

We see that there are two types of columns - one where there are continuous numeric data with extremely big outliers and the other where its a binary data. So let's try and treat them separately.

```
non_bin_col = ['duration', 'srcbytes', 'dstbytes', 'hot', 'numcompromised', 'numroot',
'numfilecreations', 'numaccessfiles', 'count', 'srvcount', 'dsthostcount', 'dsthostsrvcount', 'lastflag']
```

```
print(df['numoutboundcmds'].max())
print(df['numoutboundcmds'].min())
```

```
0
0
```

```
## Since 'numoutboundcmds' has just 0 values, we will be dropping it.
df.drop(columns = ['numoutboundcmds'], inplace = True)
```

```
## We have 3 columns 'duration', 'srcbytes' and 'dstbytes' which are continuous numeric data but having big outlier,
## so we will put a cap of its maxima. Also known as clipping
df2 = df.copy()
for col in ['duration', 'srcbytes', 'dstbytes']:
    k = np.percentile(df2[col], 95)
    df2.loc[df2[col]>k, col] = k
```

```
df2.describe()[non_bin_col]
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	dsthostcount	dsthostsr\
count	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.
mean	0.252927	232.155533	981.247418	0.204409	0.279250	0.302192	0.012669	0.004096	84.107555	27.737888	182.148945	115.
std	0.927547	389.404277	2155.660314	2.149968	23.942042	24.399618	0.483935	0.099370	114.508607	72.635840	99.206213	110.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2.000000	82.000000	10.
50%	0.000000	44.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	14.000000	8.000000	255.000000	63.
75%	0.000000	276.000000	516.000000	0.000000	0.000000	0.000000	0.000000	0.000000	143.000000	18.000000	255.000000	255.
max	4.000000	1480.000000	8314.000000	77.000000	7479.000000	7468.000000	43.000000	9.000000	511.000000	511.000000	255.000000	255.

```
df.describe()[non_bin_col]
```

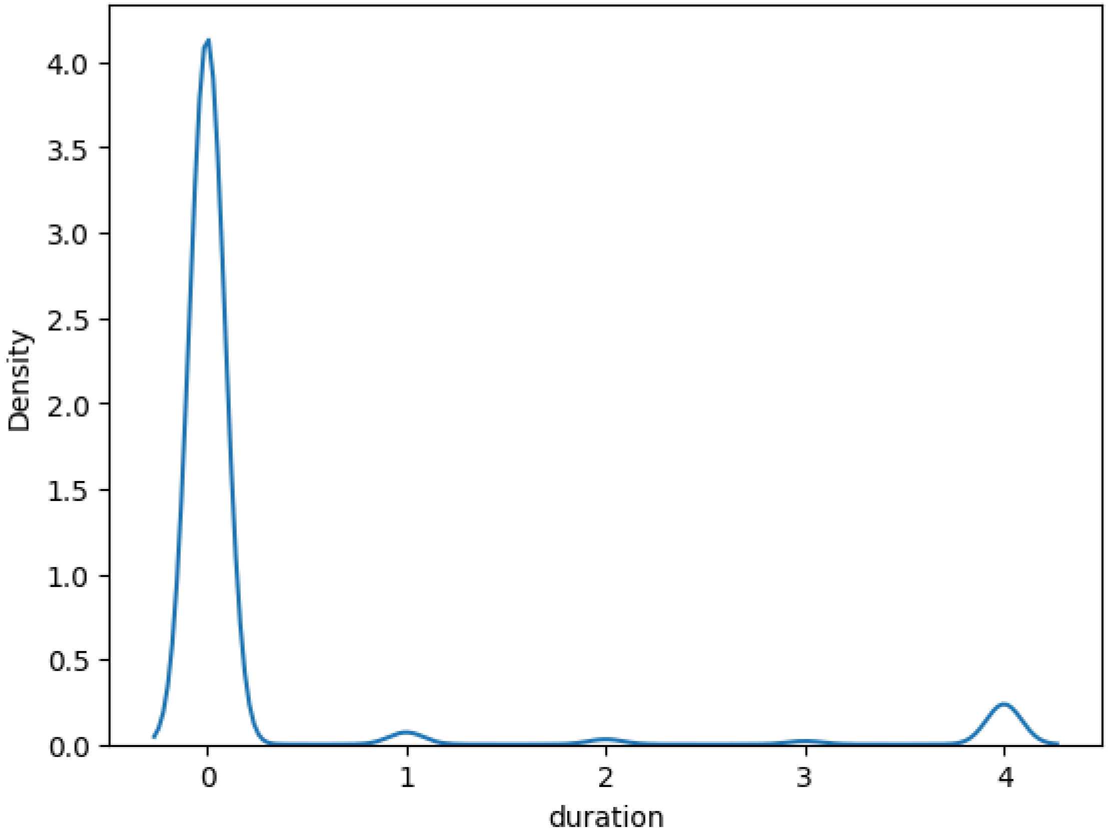
	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	dsthostcount	dsthostsrvco
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000
mean	287.14465	4.556674e+04	1.977911e+04	0.204409	0.279250	0.302192	0.012669	0.004096	84.107555	27.737888	182.148945	115.653
std	2604.51531	5.870331e+06	4.021269e+06	2.149968	23.942042	24.399618	0.483935	0.099370	114.508607	72.635840	99.206213	110.702
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2.000000	82.000000	10.000
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	14.000000	8.000000	255.000000	63.000
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	143.000000	18.000000	255.000000	255.000
max	42908.00000	1.379964e+09	1.309937e+09	77.000000	7479.000000	7468.000000	43.000000	9.000000	511.000000	511.000000	255.000000	255.000

```
for col in non_bin_col:
    sns.kdeplot(df[col])
    plt.show()
```

Outliers for 'duration' column

```
## After clipping, lets transform the data using box-cox transformation on non-binary columns to make sure we treat the outliers.
df3 = df.copy()
k = np.percentile(df['duration'], 95)
df3.loc[df3['duration']>k, 'duration'] = k
sns.kdeplot(df3['duration'])
```

```
<Axes: xlabel='duration', ylabel='Density'>
```




```
df3 = pd.DataFrame()
for col in non_bin_col:
    column, lambda_v = boxcox(df2[col] + 1)
    df3[col] = column
```

```
df3.describe()[non_bin_col]
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	dsthostcount	dsthostsr\
count	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.000000	125973.
mean	0.008694	3.214034	1.994218	0.000815	0.000116	0.000049	0.000007	0.000007	2.700655	1.522415	159.637694	6.
std	0.029579	2.863706	2.212235	0.005536	0.001140	0.000676	0.000147	0.000129	1.612402	0.641201	86.085652	3.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.065146	0.943572	73.971997	3.
50%	0.000000	3.868968	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.510683	1.633984	222.675371	6.
75%	0.000000	5.760722	4.246620	0.000000	0.000000	0.000000	0.000000	0.000000	4.331892	1.994388	222.675371	9.
max	0.109366	7.531932	5.260409	0.038427	0.011342	0.009442	0.003084	0.002378	5.255751	2.920108	222.675371	9.

```
for col in non_bin_col:
    sns.kdeplot(df3[col])
    plt.show()
```

```
bin_col = df.columns[~df.columns.isin(non_bin_col)]
bin_col
```

```
Index(['protocoltype', 'service', 'flag', 'land', 'wrongfragment', 'urgent',
       'numfailedlogins', 'loggedin', 'rootshell', 'suattempted', 'numshells',
       'ishostlogin', 'isguestlogin', 'serrorrate', 'srvserrorrate',
       'rerrorrate', 'srvrerrorrate', 'samesrvrate', 'diffsrvrate',
       'srvdiffhostrate', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',
       'dsthostsamesrcportrate', 'dsthostsrvdiffhostrate', 'dsthostserrorrate',
       'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate',
       'attack', 'attack?'],
      dtype='object')
```

```
# Now we join the non-binary columns with binary columns
df3 = pd.concat([df3, df[bin_col]], axis =1)
```

```
df3.drop_duplicates(inplace = True)
df.drop_duplicates(inplace = True)
```

```
df3
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	...	dsthostsamesrvrate	dsthostdiffsrvrate	dsthostsam
0	0.0	6.364809	0.000000	0.0	0.0	0.0	0.0	0.0	1.065146	0.943572	...	0.17	0.03	
1	0.0	5.097876	0.000000	0.0	0.0	0.0	0.0	0.0	2.451380	0.629115	...	0.00	0.60	
2	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.218551	1.494510	...	0.10	0.05	
3	0.0	5.579399	5.254500	0.0	0.0	0.0	0.0	0.0	1.703883	1.403895	...	1.00	0.00	
4	0.0	5.419534	4.155753	0.0	0.0	0.0	0.0	0.0	3.120817	2.215567	...	1.00	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
125967	0.0	6.035959	4.104679	0.0	0.0	0.0	0.0	0.0	1.333292	1.781873	...	1.00	0.00	
125968	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.519658	2.124293	...	0.10	0.06	
125970	0.0	7.531932	4.115430	0.0	0.0	0.0	0.0	0.0	0.679724	0.629115	...	0.12	0.06	
125971	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.337114	1.633984	...	0.03	0.05	
125972	0.0	5.132779	0.000000	0.0	0.0	0.0	0.0	0.0	0.679724	0.629115	...	0.30	0.03	

124658 rows × 43 columns

Hypothesis Testing

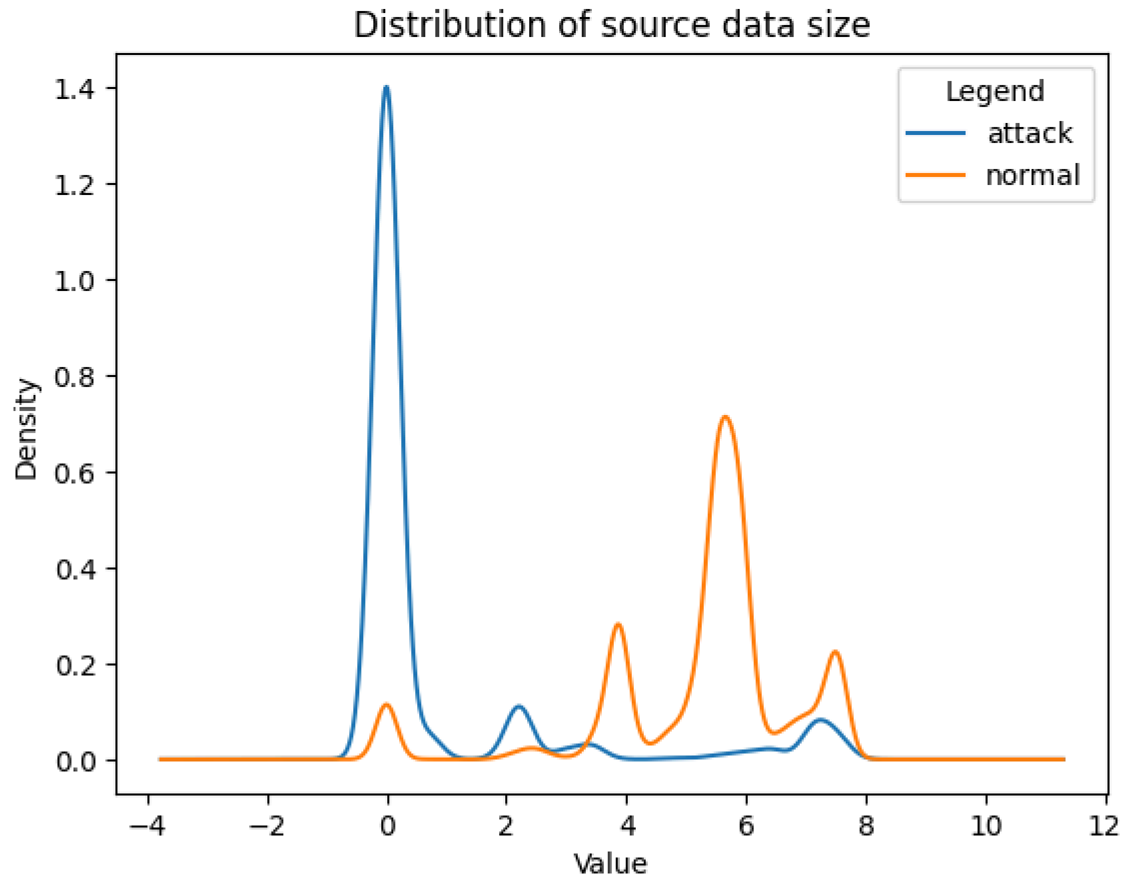
1. Network Traffic Volume and Anomalies:

Hypothesis: Network connections with unusually high or low traffic volume (bytes transferred) are more likely to be anomalous.

Tests: Use t-tests or ANOVA to compare the means of Src_bytes and Dst_bytes in normal versus anomalous connections.

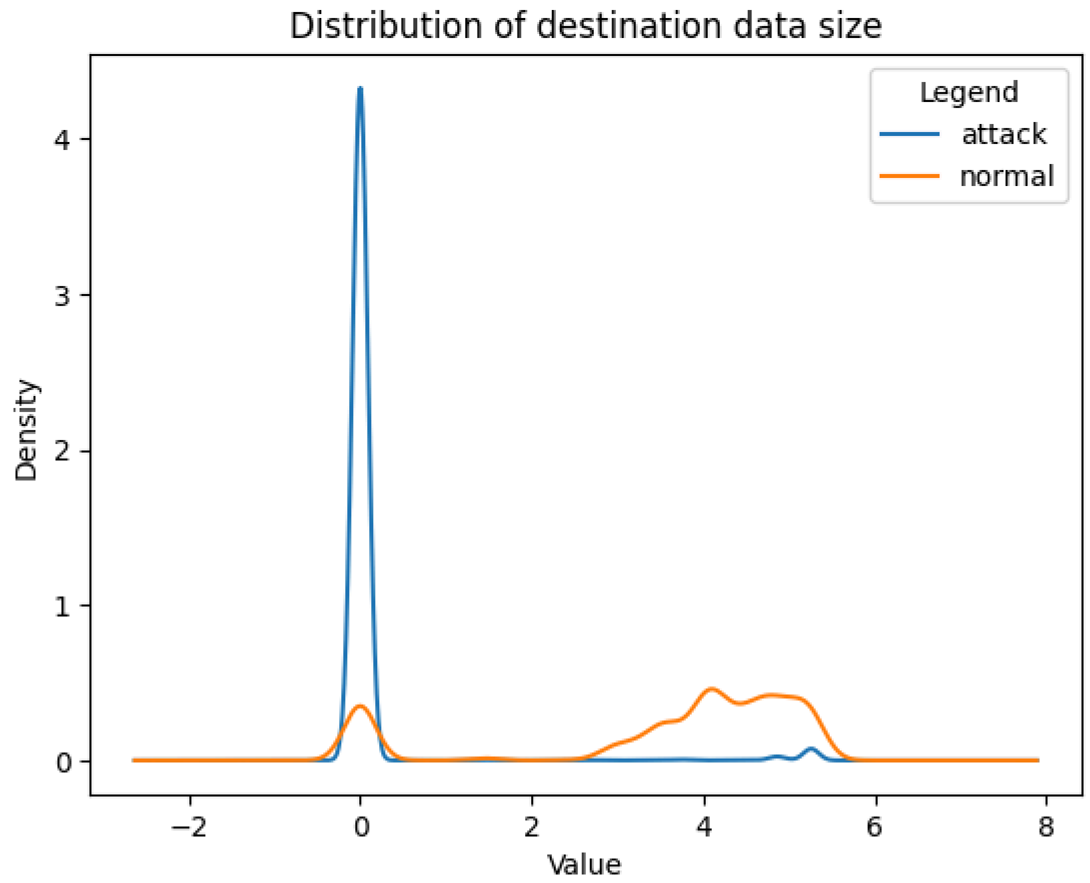
```
a_s = df3.loc[df3["attack?"]==1, "srcbytes"]
n_s = df3.loc[df3["attack?"]==0, "srcbytes"]

a_d = df3.loc[df3["attack?"]==1, "dstbytes"]
n_d = df3.loc[df3["attack?"]==0, "dstbytes"]
```

```
a_s.plot(kind = 'kde', label = "attack")
n_s.plot(kind = 'kde', label = "normal")
plt.title('Distribution of source data size')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend(title='Legend', loc='upper right')  # Custom legend title and location
plt.show()
```




```
a_d.plot(kind = 'kde', label = "attack")
n_d.plot(kind = 'kde', label = "normal")
plt.title('Distribution of destination data size')
plt.xlabel('Value')
```

```
plt.ylabel('Density')
plt.legend(title='Legend', loc='upper right')  # Custom legend title and location
plt.show()
```




```
a_s = df.loc[df["attack?"]==1, "srcbytes"]
n_s = df.loc[df["attack?"]==0, "srcbytes"]

a_d = df.loc[df["attack?"]==1, "dstbytes"]
n_d = df.loc[df["attack?"]==0, "dstbytes"]
```

Null hypothesis: The two sets of data have the same mean.

Alternative hypothesis: The two sets of data have different mean.

Alpha -> Significance level = 0.05

```
from scipy.stats import ttest_ind
t_stat, p_value = ttest_ind(a_s, n_s)
p_value
```

```
0.03558539933331456
```

Result: Since p<0.05, we can conclude that there is a significant difference between the means of the attacked connection and normal connection

Let's repeate the same experiment for destination data

2. Null hypothesis: The two sets of data have the same mean.

Alternative hypothesis: The two sets of data have different mean.

Alpha -> Significance level = 0.05

```
from scipy.stats import ttest_ind
t_stat, p_value = ttest_ind(a_d, n_d)
p_value
```

```
0.14390157812640422
```

Result: Since p>0.05, we can conclude that there is NO significant difference between the means of the attacked connection and normal connection with respect to destintion data source.

```
Start coding or generate with AI.
```

3. Impact of Protocol Type on Anomaly Detection:

Hypothesis: Certain protocols are more frequently associated with network anomalies.

Tests: Chi-square test to determine if the distribution of Protocol_type differs significantly in normal and anomalous connections.

Null hypothesis: The columns - protocol type and attack are independant

Alternative hypothesis: The columns - protocol type and attack are dependant

Alpha -> Significance level = 0.05

```
from scipy.stats import chi2_contingency
data = pd.crosstab(index = df['protocoltype'], columns = df['attack?'])
```

```
data
```

attack?	0	1
protocoltype		
icmp	1309	6982
tcp	53600	49089
udp	12434	2559

```
chi2, p, dof, expected = chi2_contingency(data)
p
```

```
0.0
```

```
expected
```

```
array([[ 4432.22605638,  3858.77394362],
       [54895.77391187, 47793.22608813],
       [ 8015.00003175,  6977.99996825]])
```

Result: since p<<0.05, we can safely conclude that the categoric columns protocol type and attack (anamoly) are dependent.

```
ser = df['service'].value_counts()
ser[ser>100]
```

```
service
http          40338
private       21853
domain_u       9043
smtp           7313
ftp_data       6860
eco_i          4586
other          4359
ecr_i          3077
telnet         2353
finger         1767
ftp            1754
auth            955
Z39_50          862
uucp            780
courier         734
bgp             710
whois           693
uucp_path       689
iso_tsap        687
time            654
imap4           647
nnsp            630
vmnet           617
```

```
urp_i           602
domain          569
ctf             563
csnet_ns        545
supdup          544
discard         538
http_443        530
daytime         521
gopher          518
efs             485
systat          477
link            475
exec            474
hostnames       460
name            451
mtp             439
echo            434
klogin          433
login           429
ldap            410
netbios_dgm     405
sunrpc          381
netbios_ssn     362
netstat         360
netbios_ns      347
ssh             311
kshell          299
nntp            296
pop_3           264
sql_net         245
IRC             187
ntp_u           168
Name: count, dtype: int64
```

```python
df.loc[~df['service'].isin(['http', 'private', 'domain_u', 'smtp', 'ftp_data']), 'service'] = "others"
```

```python
df['service'].value_counts()
```

```
service
others      40566
http        40338
private     21853
domain_u     9043
smtp         7313
ftp_data     6860
Name: count, dtype: int64
```

4. Role of Service in Network Security:

Hypothesis: Specific services are targets of network anomalies more often than others.

Tests: Chi-square test to compare the frequency of services in normal versus anomaly-flagged connections.

Null hypothesis: The columns - service and attack are independant

Alternative hypothesis: The columns - service and attack are dependant

Alpha -> Significance level = 0.05

```
data = pd.crosstab(index = df['service'], columns = df['attack?'])
data
```

attack?	0	1
service		
domain_u	9034	9
ftp_data	4984	1876
http	38049	2289
others	7265	33301
private	982	20871
smtp	7029	284

```
chi2, p, dof, expected = chi2_contingency(data)
p
```

```
0.0
```

```
expected
```

```
array([[ 4834.23232756,  4208.76767244],
       [ 3667.23805895,  3192.76194105],
       [21564.00128599, 18773.99871401],
       [21685.88616608, 18880.11383392],
       [11682.2380907 , 10170.7619093 ],
       [ 3909.40407071,  3403.59592929]])
```

Result: since p<<0.05, we can safely conclude that the categoric columns service and attack (anamoly) are dependent.

```
flag_col = ['flag', 'serrorrate', 'srvserrorrate', 'rerrorrate', 'srvrerrorrate', 'dsthostserrorrate', 'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate']
```

```
df[flag_col]
```

	flag	serrorrate	srvserrorrate	rerrorrate	srvrerrorrate	dsthostserrorrate	dsthostsrvserrorrate	dsthostrerrorrate	dsthostsrvrerrorrate
0	SF	0.0	0.0	0.0	0.0	0.00	0.00	0.05	0.00
1	SF	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00
2	S0	1.0	1.0	0.0	0.0	1.00	1.00	0.00	0.00
3	SF	0.2	0.2	0.0	0.0	0.03	0.01	0.00	0.01
4	SF	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00
...	...	...	...	...	...	...	...	...	...
125968	S0	1.0	1.0	0.0	0.0	1.00	1.00	0.00	0.00
125969	SF	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00
125970	SF	0.0	0.0	0.0	0.0	0.72	0.00	0.01	0.00
125971	S0	1.0	1.0	0.0	0.0	1.00	1.00	0.00	0.00
125972	SF	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.00

125973 rows × 9 columns

Logistic regression to evaluate whether the presence of Urgent packets increases the odds of an anomaly.

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
# We first split the data and then do data preprocessing to avoid data leakage
x_train, x_test, y_train, y_test = train_test_split(df[flag_col], df['attack?'], train_size = 0.8)
```

```
x_train
```

	flag	serrorrate	srvserrorrate	rerrorrate	srvrerrorrate	dsthostserrorrate	dsthostsrvserrorrate	dsthostrerrorrate	dsthostsrvrerrorrate
11060	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
52965	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
64293	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
47513	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
121371	REJ	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.66
...	...	...	...	...	...	...	...	...	...
91283	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
121232	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
76226	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
81488	SF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00
62381	S0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.00

100778 rows × 9 columns

```
# Target encoding of categoric column - 'flag'
ab = pd.concat([x_train['flag'], y_train], axis = 1)
ab
```

	flag	attack?
11060	SF	0
52965	SF	0
64293	SF	0
47513	SF	0
121371	REJ	0
...	...	...
91283	SF	0
121232	SF	1
76226	SF	1
81488	SF	0
62381	S0	1

100778 rows × 2 columns

```
ab = ab.groupby('flag')['attack?'].mean().reset_index()
ab
```

	flag	attack?
0	OTH	0.717949
1	REJ	0.760855
2	RSTO	0.862839
3	RSTOS0	1.000000
4	RSTR	0.938691
5	S0	0.989939
6	S1	0.010490
7	S2	0.060000
8	S3	0.045455
9	SF	0.154716
10	SH	0.990566

```
x_train = x_train.merge(ab, how = 'left', left_on = 'flag', right_on = 'flag').drop(columns = 'flag').rename(columns = {'attack?': 'flag'})
x_train
```

	serrorrate	srvserrorrate	rerrorrate	srvrerrorrate	dsthostserrorrate	dsthostsrvserrorrate	dsthostrerrorrate	dsthostsrvrerrorrate	flag
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
4	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.66	0.760855
...	...	...	...	...	...	...	...	...	...
100773	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100774	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100775	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100776	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100777	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.00	0.989939

100778 rows × 9 columns

```
ab = pd.concat([x_test['flag'], y_test], axis = 1)
ab = ab.groupby('flag')['attack?'].mean().reset_index()
x_test = x_test.merge(ab, how = 'left', left_on = 'flag', right_on = 'flag').drop(columns = 'flag').rename(columns = {'attack?': 'flag'})
```

```
x_train
```

	serrorrate	srvserrorrate	rerrorrate	srvrerrorrate	dsthostserrorrate	dsthostsrvserrorrate	dsthostrerrorrate	dsthostsrvrerrorrate	flag
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
4	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.66	0.760855
...	...	...	...	...	...	...	...	...	...
100773	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100774	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100775	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100776	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.154716
100777	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.00	0.989939

100778 rows × 9 columns

Normalizing the data

```
sc = StandardScaler()
x_train = pd.DataFrame(sc.fit_transform(x_train), columns = x_train.columns)
x_test = pd.DataFrame(sc.fit_transform(x_test), columns = x_test.columns)
```

```
log = LogisticRegression()
log.fit(x_train, y_train)
log.coef_
```

```
array([[ 0.42231764, -0.47549728, -0.95968772,  0.49143106,  0.30342497,
         1.63412975,  0.65479567,  0.27971549,  1.21134583]])
```

```
pd.DataFrame({'columns': x_train.columns, 'coefficients': log.coef_.reshape(-1)})
```

	columns	coefficients
0	serrorrate	0.422318
1	srvserrorrate	-0.475497
2	rerrorrate	-0.959688
3	srvrerrorrate	0.491431
4	dsthostserrorrate	0.303425
5	dsthostsrvserrorrate	1.634130
6	dsthostrerrorrate	0.654796
7	dsthostsrvrerrorrate	0.279715
8	flag	1.211346

```
y_train_pred = log.predict(x_train)
y_test_pred = log.predict(x_test)
```

Results

```
print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 0.879755502192939
test accuracy: 0.8820797777336773
----------------------------------------------------
train precision: 0.8020351131685048
test precision: 0.8071981621713605
----------------------------------------------------
train f1_score: 0.861210371999267
test f1_score: 0.8646160856687174
----------------------------------------------------
train recall: 0.9298132805737603
test recall: 0.9308281004709577
```

```
df['urgent'].describe()
```

```
count    125973.000000
mean          0.000111
std           0.014366
```

```
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           3.000000
Name: urgent, dtype: float64
```

```
df[['urgent', 'attack?']].corr()
```

	urgent	attack?
urgent	1.000000	-0.002787
attack?	-0.002787	1.000000

We can see that urgent column doesn't seem to have any effect on attack column

Let us try to fit a model to predict attacks just based on urgent column

```
model = LogisticRegression()
x_train, x_test, y_train, y_test = train_test_split(df['urgent'], df['attack?'], train_size = 0.8, shuffle = True)
```

```
x_train = pd.DataFrame(np.array(x_train).reshape(-1, 1), columns = ['urgent'])
x_test = pd.DataFrame(np.array(x_test).reshape(-1, 1), columns = ['urgent'])
```

```
sc = StandardScaler()
x_train = pd.DataFrame(sc.fit_transform(x_train), columns = x_train.columns)
x_test = pd.DataFrame(sc.fit_transform(x_test), columns = x_test.columns)
```

```
model.fit(x_train, y_train)
y_train_pred = model.predict(x_train)
y_test_pred = model.predict(x_test)
pd.DataFrame({'columns': x_train.columns, 'coefficients': model.coef_.reshape(-1)})
```

	columns	coefficients
0	urgent	-0.004571

```
y_train_pred = model.predict(x_train)
y_test_pred = model.predict(x_test)
```

```
print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
-----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
```

```
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 0.5338069816825101
test accuracy: 0.5376860488192101
----------------------------------------------------
train precision: 0.0
test precision: 0.0
----------------------------------------------------
train f1_score: 0.0
test f1_score: 0.0
----------------------------------------------------
train recall: 0.0
test recall: 0.0
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, msg_start, len(result))
```

Urgent column alone doesn't have a significant impact on attack

```
y_train_pred.sum()
```

```
0
```

```
df.iloc[:, 29:].head(20)
```

	srvdiffhostrate	dsthostcount	dsthostsrvcount	dsthostsamesrvrate	dsthostdiffsrvrate	dsthostsamesrcportrate	dsthostsrvdiffhostrate	dsthostserrorrate	dsthostsrvserro
0	0.00	150	25	0.17	0.03	0.17	0.00	0.00	
1	0.00	255	1	0.00	0.60	0.88	0.00	0.00	
2	0.00	255	26	0.10	0.05	0.00	0.00	1.00	
3	0.00	30	255	1.00	0.00	0.03	0.04	0.03	
4	0.09	255	255	1.00	0.00	0.00	0.00	0.00	
5	0.00	255	19	0.07	0.07	0.00	0.00	0.00	
6	0.00	255	9	0.04	0.05	0.00	0.00	1.00	
7	0.00	255	15	0.06	0.07	0.00	0.00	1.00	
8	0.00	255	23	0.09	0.05	0.00	0.00	1.00	
9	0.00	255	13	0.05	0.06	0.00	0.00	1.00	
10	0.00	255	12	0.05	0.07	0.00	0.00	0.00	
11	0.00	255	13	0.05	0.07	0.00	0.00	1.00	
12	0.43	8	219	1.00	0.00	0.12	0.03	0.00	
13	0.00	2	20	1.00	0.00	1.00	0.20	0.00	
14	0.00	255	1	0.00	0.07	0.00	0.00	1.00	
15	0.00	255	2	0.01	0.06	0.00	0.00	1.00	
16	0.22	91	255	1.00	0.00	0.01	0.02	0.00	
17	0.00	1	16	1.00	0.00	1.00	1.00	0.00	
18	0.00	66	255	1.00	0.00	0.02	0.03	0.00	
19	0.20	157	255	1.00	0.00	0.01	0.04	0.00	

ML Modeling

Data Cleaning

```
df['srvcount'] = np.where(df['srvcount']> df['count'], df['count'], df['srvcount'])
```

Feature Engineering: Creating new features

Multiplying the rate with the count gives us the actual metric like server count and host count - these are new data points which are not available in the original dataset.

```
df['diffsrvcount_'] = df['diffsrvrate']*df['count']
df['diffhostcount_'] = df['srvdiffhostrate']*df['srvcount']
df['dsthostcount_'] = df['dsthostdiffsrvrate']*df['dsthostcount']
df['sameportcount'] = df['dsthostsamesrcportrate']*df['dsthostsrvcount']
df['diffportcount'] = df['dsthostsrvdiffhostrate']*df['dsthostsrvcount']
```

Treating the outliers by clipping and applying box-cox transformation

```
df4 = df.copy()
k = np.percentile(df4['duration'], 95)
df4.loc[df4['duration']>k, 'duration'] = k
```

```
non_bin_col = ['duration', 'srcbytes', 'dstbytes', 'hot', 'numcompromised', 'numroot',
'numfilecreations', 'numaccessfiles', 'count', 'srvcount', 'dsthostcount', 'dsthostsrvcount', 'lastflag', 'diffsrvcount_',
    'dsthostcount_', 'dsthostcount_', 'dsthostcount_', 'diffportcount', 'sameportcount' ]
```

```
df5 = pd.DataFrame()
for col in non_bin_col:
    column, lambda_v = boxcox(df4[col] + 1)
    df5[col] = column
```

```
bin_col = df.columns[~df.columns.isin(non_bin_col)]
bin_col
```

```
Index(['protocoltype', 'service', 'flag', 'land', 'wrongfragment', 'urgent',
       'numfailedlogins', 'loggedin', 'rootshell', 'suattempted', 'numshells',
       'ishostlogin', 'isguestlogin', 'serrorrate', 'srvserrorrate',
       'rerrorrate', 'srvrerrorrate', 'samesrvrate', 'diffsrvrate',
       'srvdiffhostrate', 'dsthostsamesrvrate', 'dsthostdiffsrvrate',
       'dsthostsamesrcportrate', 'dsthostsrvdiffhostrate', 'dsthostserrorrate',
       'dsthostsrvserrorrate', 'dsthostrerrorrate', 'dsthostsrvrerrorrate',
       'attack', 'attack?', 'diffhostcount_'],
      dtype='object')
```

```
df5 = pd.concat([df5, df[bin_col]], axis =1)
```

```
df5.drop_duplicates(inplace = True)
```

```
df5
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	...	dsthostdiffsrvrate	dsthostsamesrcportrate	dsthos
0	0.0	5.216758	0.000000	0.0	0.0	0.0	0.0	0.0	1.065146	0.913617	...	0.03	0.17	
1	0.0	4.340009	0.000000	0.0	0.0	0.0	0.0	0.0	2.451380	0.616156	...	0.60	0.88	
2	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.218551	1.415276	...	0.05	0.00	
3	0.0	4.681479	5.189610	0.0	0.0	0.0	0.0	0.0	1.703883	1.334621	...	0.00	0.03	
4	0.0	4.569257	4.118550	0.0	0.0	0.0	0.0	0.0	3.120817	2.007436	...	0.00	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
125967	0.0	4.995837	4.068516	0.0	0.0	0.0	0.0	0.0	1.333292	1.100699	...	0.00	0.33	
125968	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.519658	1.952287	...	0.06	0.00	
125970	0.0	6.232019	4.079050	0.0	0.0	0.0	0.0	0.0	0.679724	0.616156	...	0.06	0.00	
125971	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.337114	1.537869	...	0.05	0.00	
125972	0.0	4.365110	0.000000	0.0	0.0	0.0	0.0	0.0	0.679724	0.616156	...	0.03	0.30	

119904 rows × 48 columns

```
df5.describe()
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	...	dsthostsamesrvrate
count	119904.000000	119904.000000	119904.000000	119904.000000	119904.000000	119904.000000	119904.000000	119904.000000	119904.000000	119904.000000	...	119904.000000
mean	0.008033	2.821430	2.073710	0.000856	0.000122	0.000051	0.000007	0.000007	2.717955	1.341009	...	0.527284
std	0.028525	2.464774	2.213537	0.005670	0.001168	0.000693	0.000151	0.000132	1.595189	0.570815	...	0.447034
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.065146	0.616156	...	0.050000
50%	0.000000	3.437573	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.565947	1.415276	...	0.550000
75%	0.000000	4.833069	4.303786	0.000000	0.000000	0.000000	0.000000	0.000000	4.326631	1.781393	...	1.000000
max	0.109366	12.223208	6.922971	0.038427	0.011342	0.009442	0.003084	0.002378	5.255751	2.552794	...	1.000000

8 rows × 44 columns

```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 119904 entries, 0 to 125972
Data columns (total 48 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   duration                119904 non-null  float64
 1   srcbytes                119904 non-null  float64
 2   dstbytes                119904 non-null  float64
```

```
 3   hot                     119904 non-null  float64
 4   numcompromised          119904 non-null  float64
 5   numroot                 119904 non-null  float64
 6   numfilecreations        119904 non-null  float64
 7   numaccessfiles          119904 non-null  float64
 8   count                   119904 non-null  float64
 9   srvcount                119904 non-null  float64
 10  dsthostcount            119904 non-null  float64
 11  dsthostsrvcount         119904 non-null  float64
 12  lastflag                119904 non-null  float64
 13  diffsrvcount_           119904 non-null  float64
 14  dsthostcount_           119904 non-null  float64
 15  diffportcount           119904 non-null  float64
 16  sameportcount           119904 non-null  float64
 17  protocoltype            119904 non-null  object
 18  service                 119904 non-null  object
 19  flag                    119904 non-null  object
 20  land                    119904 non-null  int64
 21  wrongfragment           119904 non-null  int64
 22  urgent                  119904 non-null  int64
 23  numfailedlogins         119904 non-null  int64
 24  loggedin                119904 non-null  int64
 25  rootshell               119904 non-null  int64
 26  suattempted             119904 non-null  int64
 27  numshells               119904 non-null  int64
 28  ishostlogin             119904 non-null  int64
 29  isguestlogin            119904 non-null  int64
 30  serrorrate              119904 non-null  float64
 31  srvserrorrate           119904 non-null  float64
 32  rerrorrate              119904 non-null  float64
 33  srvrerrorrate           119904 non-null  float64
 34  samesrvrate             119904 non-null  float64
 35  diffsrvrate             119904 non-null  float64
 36  srvdiffhostrate         119904 non-null  float64
 37  dsthostsamesrvrate      119904 non-null  float64
 38  dsthostdiffsrvrate      119904 non-null  float64
 39  dsthostsamesrcportrate  119904 non-null  float64
 40  dsthostsrvdiffhostrate  119904 non-null  float64
 41  dsthostserrorrate       119904 non-null  float64
 42  dsthostsrvserrorrate    119904 non-null  float64
 43  dsthostrerrorrate       119904 non-null  float64
 44  dsthostsrvrerrorrate    119904 non-null  float64
 45  attack                  119904 non-null  object
 46  attack?                 119904 non-null  int64
 47  diffhostcount_          119904 non-null  float64
dtypes: float64(33), int64(11), object(4)
memory usage: 44.8+ MB
```

```
df5.drop(columns= ['flag', 'attack'], inplace = True)
```

We have two categoric columns - 'protocoltype' and 'service' - both of which are important columns to detect anamoly as we have proved that in hypothesis testing. So we will use target encoding to treat them.

```
## We will concatenate both the columns to create a single unique identifier.
df5['proto_service'] = df5[['protocoltype', 'service']].apply(lambda x: x[0] + x[1], axis = 1)
```

```
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_29102/1138965760.py:2: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ve
  df5['proto_service'] = df5[['protocoltype', 'service']].apply(lambda x: x[0] + x[1], axis = 1)
```

```
df5['proto_service'].value_counts()
```

```
proto_service
tcphttp       40331
tcpothers     26589
tcpprivate    18132
udpdomain_u    9004
tcpsmtp        7313
tcpftp_data    6859
icmpothers     6042
udpprivate     3277
udpothers      2357
Name: count, dtype: int64
```

```
df5.columns
```

```
Index(['duration', 'srcbytes', 'dstbytes', 'hot', 'numcompromised', 'numroot',
       'numfilecreations', 'numaccessfiles', 'count', 'srvcount',
       'dsthostcount', 'dsthostsrvcount', 'lastflag', 'diffsrvcount_',
       'dsthostcount_', 'diffportcount', 'sameportcount', 'protocoltype',
       'service', 'land', 'wrongfragment', 'urgent', 'numfailedlogins',
       'loggedin', 'rootshell', 'suattempted', 'numshells', 'ishostlogin',
       'isguestlogin', 'serrorrate', 'srvserrorrate', 'rerrorrate',
       'srvrerrorrate', 'samesrvrate', 'diffsrvrate', 'srvdiffhostrate',
       'dsthostsamesrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdiffhostrate', 'dsthostserrorrate', 'dsthostsrvserrorrate',
       'dsthostrerrorrate', 'dsthostsrvrerrorrate', 'attack?',
       'diffhostcount_', 'proto_service'],
      dtype='object')
```

Let's split the data and then do target encoding to prevent data leakage

```
x_train, x_test, y_train, y_test = train_test_split(df5.drop(columns = 'attack?'), df5['attack?'], train_size = 0.8)
```

```
ab = pd.concat([x_train['proto_service'], y_train], axis = 1)
ab = ab.groupby('proto_service')['attack?'].mean().reset_index()
x_train = x_train.merge(ab, how = 'left', left_on = 'proto_service', right_on = 'proto_service').drop(columns = 'proto_service').rename(columns = {'attack?': 'proto_service'})
```

```
x_train
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	...	dsthostsamesrvrate	dsthostdiffsrvrate	dsthostsame
0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.679724	0.616156	...	1.00	0.00	
1	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.750545	1.100699	...	0.01	0.08	
2	0.0	5.892295	4.008413	0.0	0.0	0.0	0.0	0.0	0.679724	0.616156	...	0.76	0.03	
3	0.0	4.655892	4.665098	0.0	0.0	0.0	0.0	0.0	1.538295	1.233493	...	1.00	0.00	
4	0.0	5.725758	0.000000	0.0	0.0	0.0	0.0	0.0	4.852442	2.481006	...	1.00	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
95918	0.0	5.609052	4.011165	0.0	0.0	0.0	0.0	0.0	0.679724	0.616156	...	0.70	0.22	
95919	0.0	4.519559	4.441672	0.0	0.0	0.0	0.0	0.0	2.755477	1.863515	...	1.00	0.00	
95920	0.0	4.241521	3.871789	0.0	0.0	0.0	0.0	0.0	0.679724	0.616156	...	0.10	0.68	
95921	0.0	4.668804	4.436005	0.0	0.0	0.0	0.0	0.0	1.961710	1.481742	...	1.00	0.00	
95922	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	4.631014	1.481742	...	0.09	0.11	

95923 rows × 46 columns

```
x_test = x_test.merge(ab, how = 'left', left_on = 'proto_service', right_on = 'proto_service').drop(columns = 'proto_service').rename(columns = {'attack?': 'proto_service'})
```

```
x_test.drop(columns = ['protocoltype', 'service'], inplace = True)
x_train.drop(columns = ['protocoltype', 'service'], inplace = True)
```

Normalizing using Standard Scaler

```
sc = StandardScaler()
x_train = pd.DataFrame(sc.fit_transform(x_train), columns = x_train.columns)
x_test = pd.DataFrame(sc.fit_transform(x_test), columns = x_test.columns)
```

Logistic regression

```
log = LogisticRegression(solver='saga', max_iter=1000)
log.fit(x_train, y_train)
log.coef_
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
array([[-0.08703018, -0.89485218, -1.82535715,  1.22971573,  0.43769488,
        -0.18797965, -0.11215032, -0.04058675,  0.80381424,  1.0493165 ,
         0.26019038, -0.64970714, -2.06902272,  0.93763635,  1.45384587,
         0.53140858, -0.05615895, -0.10236996,  1.21822459,  0.04542203,
         0.00715617,  0.23365181, -0.06067659, -0.13198858,  0.00234145,
```

```
       -0.04101684, -0.18893475, -0.21946405,  0.98335932,  0.03287016,
        0.19807985,  0.15527979, -0.08162754,  0.39042286,  0.87639379,
       -0.1440164 ,  1.06070248,  0.11761507,  0.35917536,  1.48310768,
        0.87876486, -0.04903877, -0.19826502,  1.96230972]])
```

We see that logistic regression model doesn't converge while training - the reason being that there are too many feature - around 46 columns. So we will do dimensionality reduction using PCA.

Trial 1: keeping 95% variance

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(x_train)


log = LogisticRegression(solver='saga', max_iter=1000)
log.fit(X_pca, y_train)
log.coef_
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
array([[ 2.35903109,  0.34933753,  0.62873558, -1.19276347,  0.02205653,
         1.95076054, -1.66258561,  0.77639476,  0.00367257,  0.21753442,
         0.19528831,  0.1262824 , -0.62767168, -0.44845364,  0.04486449,
         0.29807879,  0.16050605,  0.24697323, -0.37057346,  0.73470023,
        -0.35509366, -0.07727516, -0.25977769,  0.48538217, -0.40506234]])
```

Still there is no convergance

Trial 2: keeping 90% variance

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.90)
X_pca = pca.fit_transform(x_train)


log = LogisticRegression(solver='saga', max_iter=1000)
log.fit(X_pca, y_train)
log.coef_
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
array([[ 2.3239272 ,  0.35785834,  0.73965971, -1.07893093,  0.31976039,
         2.16451934, -1.49067104,  0.82884659, -0.09180538, -0.047379  ,
         0.18929323,  0.27353152, -0.61853949, -0.96464668,  0.2168513 ,
         0.2207898 ,  0.32933889,  0.25425151, -0.19057005,  0.59268995]])
```

Still there is no convergance

Trial 3: keeping 85% variance

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.85)
X_pca = pca.fit_transform(x_train)
```

```
log = LogisticRegression(solver='saga', max_iter=1000)
log.fit(X_pca, y_train)
log.coef_
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
array([[ 2.29322123,  0.35684741,  0.612915  , -0.98726712,  0.25457742,
         2.16828692, -1.56117206,  0.84596634, -0.01356864,  0.1287486 ,
         0.14018764,  0.13721959, -0.54265689, -0.61853989,  0.27523331,
         0.13461764,  0.20270448,  0.27078601]])
```

Even with 85% variance we still could not find the convergance. So we will just proceed with the 85% variance model

```
pca
```

```
▾ PCA
PCA(n_components=0.85)
```

```
X_pca_test = pca.transform(x_test)
```

```
y_train_pred = log.predict(X_pca)
y_test_pred = log.predict(X_pca_test)
```

```
print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
-----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
-----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
-----------------------------------------------------\n \
```

```
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 0.9794731190642495
test accuracy: 0.9798173554063634
----------------------------------------------------
train precision: 0.9826689364921738
test precision: 0.9814710308502633
----------------------------------------------------
train f1_score: 0.9768600674571929
test f1_score: 0.9773344572445443
----------------------------------------------------
train recall: 0.9711194709909573
test recall: 0.9732326058571162
```

Result Logistic regression: f1_score and accuracy of around 97% on test data.

Support Vector Machines

```
from sklearn import svm
```

```
svm_c = svm.SVC(kernel = 'linear')
```

```
svm_c.fit(X_pca, y_train)
```

```
▾ SVC
SVC(kernel='linear')
```

```
y_train_pred = svm_c.predict(X_pca)
y_test_pred = svm_c.predict(X_pca_test)
```

```
print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 0.9807449725300501
test accuracy: 0.9801926525165756
----------------------------------------------------
train precision: 0.9833459500378501
test precision: 0.9818847381265252
```

```
    ----------------------------------------------------
    train f1_score: 0.9782662414835909
    test f1_score: 0.9778006262560172
    ----------------------------------------------------
    train recall: 0.9732387441174405
    test recall: 0.9737503490645071
```

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test_pred, y_test)
```

```
array([[13045,   193],
       [  282, 10461]])
```

SVM Result: f1_score of 97%, accuracy of 98%

Decision tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
DT = DecisionTreeClassifier(random_state = 45, criterion='gini', max_features = 'sqrt')
```

```
x_train, x_test, y_train, y_test = train_test_split(df5.drop(columns = 'attack?'), df5['attack?'], train_size = 0.8)
```

```
ab = pd.concat([x_train['proto_service'], y_train], axis = 1)
ab = ab.groupby('proto_service')['attack?'].mean().reset_index()
x_train = x_train.merge(ab, how = 'left', left_on = 'proto_service', right_on = 'proto_service').drop(columns = 'proto_service').rename(columns = {'attack?': 'proto_service'})
```

```
x_test = x_test.merge(ab, how = 'left', left_on = 'proto_service', right_on = 'proto_service').drop(columns = 'proto_service').rename(columns = {'attack?': 'proto_service'})
```

```
x_test.drop(columns = ['protocoltype', 'service'], inplace = True)
x_train.drop(columns = ['protocoltype', 'service'], inplace = True)
```

```
DT.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_features='sqrt', random_state=45)
```

```
y_train_pred = DT.predict(x_train)
y_test_pred = DT.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
```

```
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 1.0
test accuracy: 0.9981235144489388
----------------------------------------------------
train precision: 1.0
test precision: 0.9987801445059585
----------------------------------------------------
train f1_score: 1.0
test f1_score: 0.9978905920405006
----------------------------------------------------
train recall: 1.0
test recall: 0.997002622705133
```

Since we have a train accuracy of 100%, we are overfitting the model with just 1 decision tree, Let's try grid search CV to pick the best hyperparameters.

```
grid_params = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(estimator = DT, cv = 5, param_grid = grid_params, scoring = 'accuracy')
```

```
gs.fit(x_train, y_train)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[167], line 2
      1 gs.fit(x_train, y_train)
----> 2 gs.best_param_

AttributeError: 'GridSearchCV' object has no attribute 'best_param_'
```

```
gs.best_params_
```

```
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5}
```

```
gs.best_score_
```

```
0.9976335227884066
```

```
DT2 = DecisionTreeClassifier(random_state = 65, criterion = 'gini', max_features = 'sqrt', max_depth = None, min_samples_leaf = 1, min_samples_split = 5)
DT2.fit(x_train, y_train)
```

```
▾                      DecisionTreeClassifier
DecisionTreeClassifier(max_features='sqrt', min_samples_split=5,
                       random_state=65)
```

```
y_train_pred = DT2.predict(x_train)
y_test_pred = DT2.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
 train accuracy: 0.999489173607998
 test accuracy: 0.9972478211917768
 ----------------------------------------------------
 train precision: 0.9992666020014668
 test precision: 0.9963404335178756
 ----------------------------------------------------
 train f1_score: 0.9994202898550725
 test f1_score: 0.9969016993709511
 ----------------------------------------------------
 train recall: 0.9995740249905339
 test recall: 0.997463597933302
```

Result Decision Tree Classifier: 99% Accuracy and f1_score

Not the best model as it is overfitting

```
feature_importance_df = pd.DataFrame({
    'Feature': x_train.columns,
    'Importance': DT2.feature_importances_
}).sort_values(by='Importance', ascending=False)
feature_importance_df
```

	Feature	Importance
2	dstbytes	0.641629
15	diffportcount	0.084283
12	lastflag	0.057527
13	diffsrvcount_	0.049945
8	count	0.025015
1	srcbytes	0.022385
11	dsthostsrvcount	0.020256
40	dsthostrerrorrate	0.016788
28	srvserrorrate	0.012620
36	dsthostsamesrcportrate	0.012489
16	sameportcount	0.011585
29	rerrorrate	0.006561
43	proto_service	0.005068
18	wrongfragment	0.005059
39	dsthostsrvserrorrate	0.004441
37	dsthostsrvdiffhostrate	0.003365
34	dsthostsamesrvrate	0.003093
10	dsthostcount	0.002721
9	srvcount	0.002715
35	dsthostdiffsrvrate	0.001926
32	diffsrvrate	0.001760
3	hot	0.001429
14	dsthostcount_	0.001228
26	isguestlogin	0.001184
38	dsthostserrorrate	0.001104
0	duration	0.000685
5	numroot	0.000648
31	samesrvrate	0.000500
21	loggedin	0.000491
41	dsthostsrvrerrorrate	0.000466
30	srvrerrorrate	0.000383
27	serrorrate	0.000373

33	srvdiffhostrate	0.000129
42	diffhostcount_	0.000080
24	numshells	0.000040
17	land	0.000033
6	numfilecreations	0.000000
4	numcompromised	0.000000
7	numaccessfiles	0.000000
25	ishostlogin	0.000000
19	urgent	0.000000
20	numfailedlogins	0.000000
22	rootshell	0.000000
23	suattempted	0.000000

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier


rf = RandomForestClassifier(criterion = 'gini', max_features = 'sqrt', random_state = 23)


param_grid = {
    'n_estimators': [50,100,150],
    'min_samples_split': [3,7,11],
    'min_samples_leaf': [1, 3, 7, 9]
}


## Using Grid Search CV to pick the best model
gs = GridSearchCV(estimator = rf, cv = 5, n_jobs = -1, scoring = 'accuracy', param_grid = param_grid)


gs.fit(x_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
gs.best_params_
```

```
{'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 150}
```

```
gs.best_score_
```

```
0.9994996003749975
```

This model is most likely overfitting because of just 1 min_samples_leaf and just 3 min_split_samples. Apart from that it is giving 99.9% accuracy which is a signal that it is overfitting.

```
param_grid = {
    'max_depth': [5,10,20],
    'min_samples_split': [3,7,11],
    'min_samples_leaf': [3, 7, 9]
}
```

```
gs = GridSearchCV(estimator = rf, cv = 5, n_jobs = -1, scoring = 'accuracy', param_grid = param_grid)
```

```
gs.fit(x_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

```
gs.best_params_
```

```
{'max_depth': 20, 'min_samples_leaf': 3, 'min_samples_split': 7}
```

```
gs.best_score_
```

```
0.9993432260355961
```

```
rf = RandomForestClassifier(criterion = 'gini', max_features = 'sqrt', random_state = 23, n_estimators = 150,
                           min_samples_split = 7, min_samples_leaf = 3, max_depth = 20)
```

```
rf.fit(x_train, y_train)
y_train_pred = rf.predict(x_train)
y_test_pred = rf.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
```

```
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 0.9998331995454688
test accuracy: 0.999207706100663
----------------------------------------------------
train precision: 0.9998580520002839
test precision: 0.9996246598479872
----------------------------------------------------
train f1_score: 0.9998107449551702
test f1_score: 0.9991090269636577
----------------------------------------------------
train recall: 0.9997634423863933
test recall: 0.9985939257592801
```

Result Random Forest: 99.9% Accuracy and f1_score

Very much unlikely to overfit as we have changed the hyperparameters.

Overall a good model.

```
feature_importance_df = pd.DataFrame({
    'Feature': x_train.columns,
    'Importance': rf.feature_importances_
}).sort_values(by='Importance', ascending=False)
```

```
feature_importance_df
```

	Feature	Importance
1	srcbytes	0.164305
43	proto_service	0.136203
2	dstbytes	0.083289
13	diffsrvcount_	0.080454
8	count	0.058244
12	lastflag	0.056124
31	samesrvrate	0.055139
14	dsthostcount_	0.055130
32	diffsrvrate	0.053162
34	dsthostsamesrvrate	0.029169
11	dsthostsrvcount	0.026893
28	srvserrorrate	0.021370
39	dsthostsrvserrorrate	0.018268
21	loggedin	0.017698
35	dsthostdiffsrvrate	0.016780
38	dsthostserrorrate	0.016611
36	dsthostsamesrcportrate	0.015053
9	srvcount	0.014682
27	serrorrate	0.010684
16	sameportcount	0.009375
37	dsthostsrvdiffhostrate	0.009355
40	dsthostrerrorrate	0.007827
10	dsthostcount	0.006844
3	hot	0.005932
15	diffportcount	0.005321
41	dsthostsrvrerrorrate	0.004852
4	numcompromised	0.004580
33	srvdiffhostrate	0.003400
29	rerrorrate	0.002919
0	duration	0.002905
30	srvrerrorrate	0.002633
18	wrongfragment	0.002175

42	diffhostcount_	0.001730
26	isguestlogin	0.000514
5	numroot	0.000162
20	numfailedlogins	0.000082
6	numfilecreations	0.000061
22	rootshell	0.000025
17	land	0.000018
24	numshells	0.000017
7	numaccessfiles	0.000015
23	suattempted	0.000004
25	ishostlogin	0.000000
19	urgent	0.000000

XGBoost

```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-2.1.2-py3-none-macosx_10_15_x86_64.macosx_11_0_x86_64.macosx_12_0_x86_64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from xgboost) (1.26.1)
Requirement already satisfied: scipy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from xgboost) (1.11.3)
Downloading xgboost-2.1.2-py3-none-macosx_10_15_x86_64.macosx_11_0_x86_64.macosx_12_0_x86_64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 4.0 MB/s eta 0:00:0000:01:00:01
Installing collected packages: xgboost
Successfully installed xgboost-2.1.2

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

```
from xgboost import XGBClassifier
```

```
# Initialize the XGBoost Classifier
xgb_classifier = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)

# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'min_child_weight': [1,3,5,10,15],
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7],
    'subsample': [0.5, 0.7],
    'colsample_bytree': [0.5, 0.7]
```

```
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42),
                           param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)

# Fit GridSearchCV
grid_search.fit(x_train, y_train)

# Output best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)


xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42, n_estimators = 200,
                           eta = 0.3, min_samples_leaf = 3, max_depth = 7, min_child_weight = 1, subsample = 0.5,
                  colsample_bytree = 0.7)


xgb.fit(x_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/xgboost/core.py:158: UserWarning: [11:23:27] WARNING: /Users/runner/work/xgboost/xgboost/src
Parameters: { "min_samples_leaf", "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.7, device=None, early_stopping_rounds=None,
              enable_categorical=False, eta=0.3, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.3, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=7,
              max_leaves=None, min_child_weight=1, min_samples_leaf=3,
              missing=nan, monotone_constraints=None, multi_strategy=None,
              n_estimators=200, n_jobs=None, ...)
```

```
y_train_pred = xgb.predict(x_train)
y_test_pred = xgb.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 1.0
test accuracy: 0.9995830032108752
----------------------------------------------------
train precision: 1.0
test precision: 0.9998123299239936
----------------------------------------------------
train f1_score: 1.0
test f1_score: 0.99953095684803
----------------------------------------------------
train recall: 1.0
test recall: 0.9992497420988464
```

With train accuracy of 100% it suggests that it is overfitting, let's change the hyperparameters

```
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42, n_estimators = 150,
                           eta = 0.1, min_samples_leaf = 3, max_depth = 7, min_child_weight = 5, subsample = 0.5,
                  colsample_bytree = 0.7)
```

```
xgb.fit(x_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/xgboost/core.py:158: UserWarning: [11:26:24] WARNING: /Users/runner/work/xgboost/xgboost/src
Parameters: { "min_samples_leaf", "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
▾                          XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.7, device=None, early_stopping_rounds=None,
              enable_categorical=False, eta=0.1, eval_metric='mlogloss',
              feature_types=None, gamma=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=7,
              max_leaves=None, min_child_weight=5, min_samples_leaf=3,
              missing=nan, monotone_constraints=None, multi_strategy=None,
              n_estimators=150, n_jobs=None, ...)
```

```
y_train_pred = xgb.predict(x_train)
y_test_pred = xgb.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
```

```
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 0.9997497993182032
test accuracy: 0.999374504816313
----------------------------------------------------
train precision: 0.9997870780004259
test precision: 0.9997184948859904
----------------------------------------------------
train f1_score: 0.9997161241483725
test f1_score: 0.9992965342587816
----------------------------------------------------
train recall: 0.999645180366647
test recall: 0.9988749296831052
```

Result XGBoost: 99.9% accuracy and f1_score

unlikely to overfit as we have given higher value to min_samples_split and min_samples_leaf. And lower max_depth and eta values.

```
feature_importance_df = pd.DataFrame({
    'Feature': x_train.columns,
    'Importance': xgb.feature_importances_
}).sort_values(by='Importance', ascending=False)
feature_importance_df
```

	Feature	Importance
43	proto_service	0.209783
13	diffsrvcount_	0.169984
1	srcbytes	0.110850
2	dstbytes	0.088170
4	numcompromised	0.065443
8	count	0.052293
9	srvcount	0.031995
12	lastflag	0.028999
38	dsthostserrorrate	0.027059
21	loggedin	0.026680
14	dsthostcount_	0.022064
28	srvserrorrate	0.019147
32	diffsrvrate	0.014257
39	dsthostsrvserrorrate	0.013054
18	wrongfragment	0.011778
26	isguestlogin	0.011431
3	hot	0.011056
41	dsthostsrvrerrorrate	0.008546
27	serrorrate	0.007836
31	samesrvrate	0.007785
16	sameportcount	0.006965
37	dsthostsrvdiffhostrate	0.005542
0	duration	0.005524
10	dsthostcount	0.005249
34	dsthostsamesrvrate	0.004871
5	numroot	0.004736
36	dsthostsamesrcportrate	0.004535
40	dsthostrerrorrate	0.004040
11	dsthostsrvcount	0.003784
35	dsthostdiffsrvrate	0.003716
15	diffportcount	0.003682
30	srvrerrorrate	0.002833

29	rerrorrate	0.002648
42	diffhostcount_	0.002562
33	srvdiffhostrate	0.000652
6	numfilecreations	0.000447
7	numaccessfiles	0.000000
24	numshells	0.000000
22	rootshell	0.000000
25	ishostlogin	0.000000
20	numfailedlogins	0.000000
19	urgent	0.000000
17	land	0.000000
23	suattempted	0.000000

LightGBM

```
pip install lightgbm
```

```
Collecting lightgbm
  Downloading lightgbm-4.5.0-py3-none-macosx_10_15_x86_64.whl.metadata (17 kB)
Requirement already satisfied: numpy>=1.17.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from lightgbm) (1.26.1)
Requirement already satisfied: scipy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from lightgbm) (1.11.3)
Downloading lightgbm-4.5.0-py3-none-macosx_10_15_x86_64.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 4.3 MB/s eta 0:00:00a 0:00:01m
Installing collected packages: lightgbm
Successfully installed lightgbm-4.5.0

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

```
import lightgbm as lgb
lgbm = lgb.LGBMClassifier(random_state=42)
param_grid = {
    'num_leaves': [31, 50, 70],
    'max_depth': [-1, 10, 20],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [50, 100, 200]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=lgbm,
                           param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)
grid_search.fit(x_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

```
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 200, 'num_leaves': 31}
Best Cross-Validation Accuracy: 0.9996976755888355
```

```
lgbm = lgb.LGBMClassifier(random_state=42, n_jobs=-1, learning_rate = 0.1, max_depth = 10, n_estimators = 200, num_leaves = 31)
lgbm.fit(x_train, y_train)
```

```
[LightGBM] [Info] Number of positive: 42269, number of negative: 53654
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.023979 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 4052
[LightGBM] [Info] Number of data points in the train set: 95923, number of used features: 42
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.440656 -> initscore=-0.238502
[LightGBM] [Info] Start training from score -0.238502
```

```
▾ LGBMClassifier
LGBMClassifier(max_depth=10, n_estimators=200, n_jobs=-1, random_state=42)
```

```
y_train_pred = lgbm.predict(x_train)
y_test_pred = lgbm.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
 train accuracy: 1.0
 test accuracy: 0.9996664025687002
```

```
---------------------------------------------------
train precision: 1.0
test precision: 0.9997184948859904
---------------------------------------------------
train f1_score: 1.0
test f1_score: 0.9996246950647402
---------------------------------------------------
train recall: 1.0
test recall: 0.9995309128436063
```

Train accuracy of 100%, suggesting overfitting, so lets change the hyperparameters

```
lgbm = lgb.LGBMClassifier(random_state=42, n_jobs=-1, learning_rate = 0.1, max_depth = 7, n_estimators = 150, num_leaves = 31)
lgbm.fit(x_train, y_train)
```

```
[LightGBM] [Info] Number of positive: 42269, number of negative: 53654
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.016955 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4052
[LightGBM] [Info] Number of data points in the train set: 95923, number of used features: 42
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.440656 -> initscore=-0.238502
[LightGBM] [Info] Start training from score -0.238502
LGBMClassifier
LGBMClassifier(max_depth=7, n_estimators=150, n_jobs=-1, random_state=42)
```

```
y_train_pred = lgbm.predict(x_train)
y_test_pred = lgbm.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
---------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
---------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
---------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
train accuracy: 1.0
test accuracy: 0.9995830032108752
---------------------------------------------------
train precision: 1.0
test precision: 0.9997184948859904
---------------------------------------------------
train f1_score: 1.0
test f1_score: 0.9995309128436063
---------------------------------------------------
train recall: 1.0
test recall: 0.9993434011818779
```

Result LightGBM: 99.95% test accuracy and f1_score

100% results on train data suggest overfit even after tuning the hyperparameters with lower max_depth and lower n_estimators

```
confusion_matrix(y_test_pred, y_test)
```

```
array([[13317,     3],
       [    7, 10654]])
```

We successfully bought down the false negatives and false positives to single digits!

```
feature_importance_df = pd.DataFrame({
    'Feature': x_train.columns,
    'Importance': lgbm.feature_importances_
}).sort_values(by='Importance', ascending=False)
feature_importance_df
```

	Feature	Importance
1	srcbytes	703
12	lastflag	612
43	proto_service	406
38	dsthostserrorrate	183
14	dsthostcount_	179
39	dsthostsrvserrorrate	177
11	dsthostsrvcount	168
10	dsthostcount	159
8	count	145
2	dstbytes	139
16	sameportcount	129
40	dsthostrerrorrate	123
34	dsthostsamesrvrate	111
15	diffportcount	110
9	srvcount	106
13	diffsrvcount_	101
36	dsthostsamesrcportrate	95
37	dsthostsrvdiffhostrate	85
35	dsthostdiffsrvrate	84
21	loggedin	79
18	wrongfragment	70
41	dsthostsrvrerrorrate	69
0	duration	60
31	samesrvrate	59
3	hot	57
27	serrorrate	55
28	srvserrorrate	52
29	rerrorrate	33
4	numcompromised	28
33	srvdiffhostrate	27
5	numroot	23
32	diffsrvrate	19

26	isguestlogin	17
30	srvrerrorrate	10
6	numfilecreations	9
20	numfailedlogins	7
17	land	6
42	diffhostcount_	4
22	rootshell	1
7	numaccessfiles	0
24	numshells	0
25	ishostlogin	0
23	suattempted	0
19	urgent	0

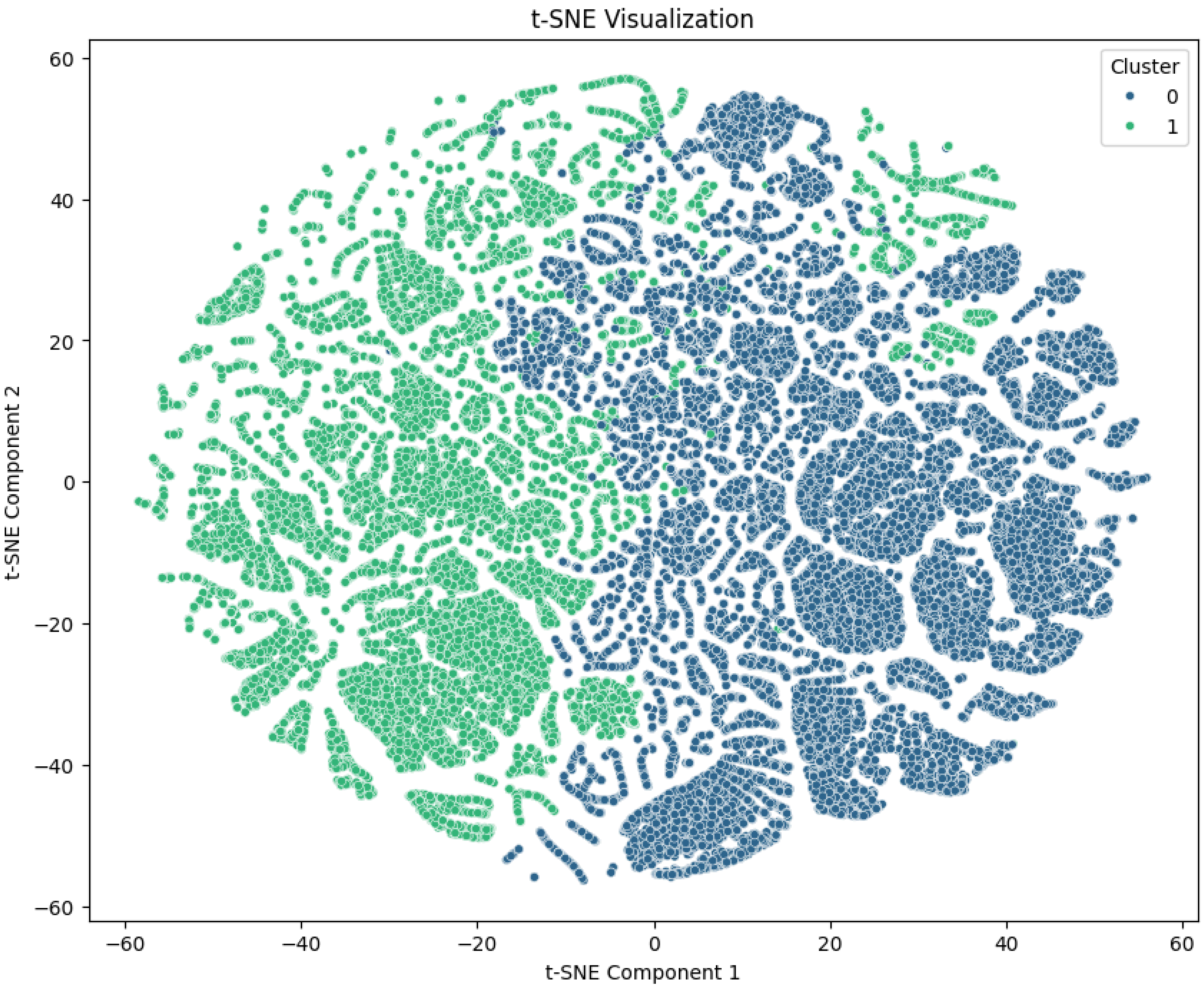
Unsupervised Learning - T-SNE

```
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(pd.concat([x_train, x_test], axis = 0)), columns = x_train.columns)
X
```

	duration	srcbytes	dstbytes	hot	numcompromised	numroot	numfilecreations	numaccessfiles	count	srvcount	...	dsthostsamesrvrate	dsthostdiffsrvrate	dsthc
0	-0.281624	-1.144706	-0.936835	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	1.266899	-0.188356	...	-1.134784	-0.154986	
1	-0.281624	0.765893	1.351715	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	-0.868029	-0.420996	...	1.057454	-0.431552	
2	-0.281624	-1.144706	-0.936835	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	1.001790	0.429495	...	-1.090044	-0.044359	
3	-0.281624	0.749513	1.149976	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	-0.062863	0.811697	...	1.057454	-0.431552	
4	-0.281624	-1.144706	-0.936835	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	0.856355	-1.269861	...	-1.000565	-0.044359	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
119899	-0.281624	-1.144706	-0.936835	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	0.984668	0.848828	...	-1.045305	-0.044359	
119900	3.552382	-0.269541	0.344227	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	-1.277742	-1.269861	...	-1.157153	-0.320925	
119901	-0.281624	-1.144706	-0.936835	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	1.348884	0.771503	...	-1.045305	0.010954	
119902	3.552382	0.754656	1.038561	6.626301	-0.104123	-0.073771	-0.048983	-0.055711	-1.277742	-1.269861	...	-0.821607	-0.210299	
119903	-0.281624	-1.144706	-0.936835	-0.150914	-0.104123	-0.073771	-0.048983	-0.055711	1.031050	-0.011191	...	-1.134784	-0.099672	

119904 rows × 44 columns

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200, random_state=42)
X_tsne = tsne.fit_transform(X)


y_tsne = pd.concat([y_train, y_test], axis = 0)


# Plot t-SNE results
plt.figure(figsize=(10, 8))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=y_tsne, palette='viridis', s=20)
plt.title("t-SNE Visualization")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.legend(title="Cluster")
plt.show()
```

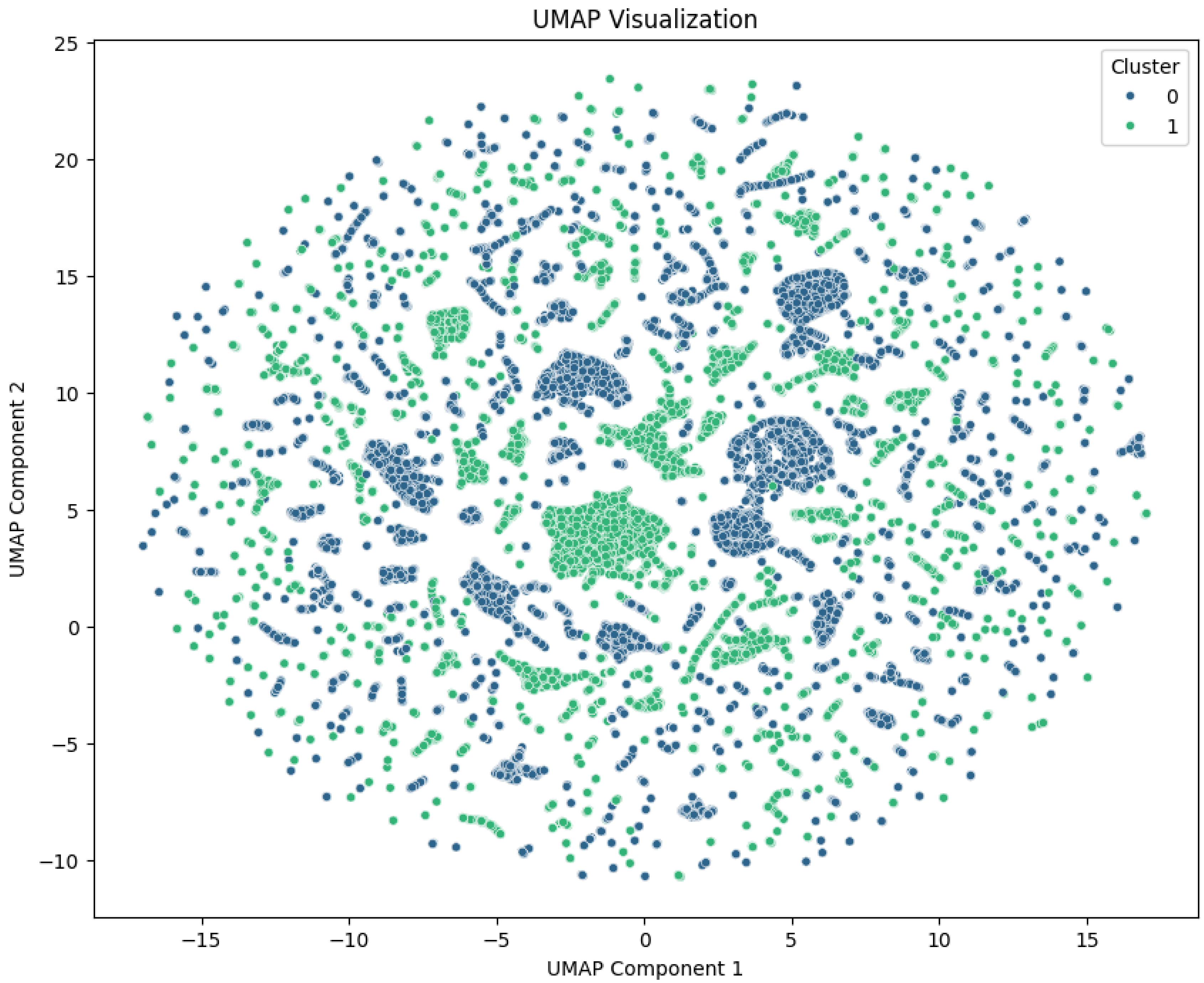


UMAP

```
import umap

# Initialize UMAP with parameters
reducer = umap.UMAP(n_components=2, n_neighbors=20, min_dist=0.1)
X_umap = reducer.fit_transform(X)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not fully connected, spec
  warnings.warn(
```

```
# Plot t-SNE results
plt.figure(figsize=(10, 8))
sns.scatterplot(x=X_umap[:, 0], y=X_umap[:, 1], hue=y_tsne, palette='viridis', s=20)
plt.title("UMAP Visualization")
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.legend(title="Cluster")
plt.show()
```



Saving the pickle file

```
def transform_(test):
    test.drop(columns = ['numoutboundcmds', 'flag'], inplace = True)
    test['srvcount'] = np.where(test['srvcount']> test['count'], test['count'], test['srvcount'])
    test['diffsrvcount_'] = test['diffsrvrate']*test['count']
    test['diffhostcount_'] = test['srvdiffhostrate']*test['srvcount']
    test['dsthostcount_'] = test['dsthostdiffsrvrate']*test['dsthostcount']
    test['sameportcount'] = test['dsthostsamesrcportrate']*test['dsthostsrvcount']
    test['diffportcount'] = test['dsthostsrvdiffhostrate']*test['dsthostsrvcount']

    test['proto_service'] = test[['protocoltype', 'service']].apply(lambda x: x[0] + x[1], axis = 1)

    test = test.merge(ab, how = 'left', left_on = 'proto_service', right_on = 'proto_service').drop(columns = 'proto_service').rename(columns = {'attack?': 'proto_service'})
```

```
    test.drop(columns = ['protocoltype', 'service'], inplace = True)
    return test


df = pd.read_csv('Network_anomaly_data.csv')
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42, n_estimators = 100,
                           eta = 0.3, min_samples_leaf = 3, max_depth = 7, min_child_weight = 1, subsample = 0.5,
                  colsample_bytree = 0.7)
df['attack?'] = np.where(df['attack']=="normal",0,1)
X = transform_(df.drop(columns = ['attack', 'attack?']))
Y = df['attack?']
x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size = 0.8)
xgb.fit(x_train, y_train)


y_train_pred = xgb.predict(x_train)
y_test_pred = xgb.predict(x_test)

print(f" train accuracy: {accuracy_score(y_train_pred, y_train)} \n \
test accuracy: {accuracy_score(y_test_pred, y_test)} \n \
----------------------------------------------------\n \
train precision: {precision_score(y_train_pred, y_train)}\n \
test precision: {precision_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train f1_score: {f1_score(y_train_pred, y_train)}\n \
test f1_score: {f1_score(y_test_pred, y_test)}\n \
----------------------------------------------------\n \
train recall: {recall_score(y_train_pred, y_train)}\n \
test recall: {recall_score(y_test_pred, y_test)}")
```

```
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_29102/1934187345.py:10: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future v
  test['proto_service'] = test[['protocoltype', 'service']].apply(lambda x: x[0] + x[1], axis = 1)
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/xgboost/core.py:158: UserWarning: [21:11:41] WARNING: /Users/runner/work/xgboost/xgboost/src
Parameters: { "min_samples_leaf", "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
 train accuracy: 0.9999801543987775
 test accuracy: 0.9994840246080572
 ----------------------------------------------------
 train precision: 0.9999786589269708
 test precision: 0.9995752633367312
 ----------------------------------------------------
 train f1_score: 0.9999786589269708
 test f1_score: 0.9994479126852677
 ----------------------------------------------------
 train recall: 0.9999786589269708
 test recall: 0.9993205944798301
```

```
import pickle

# Save the model to a pickle file
with open("xgboost_model.pkl", "wb") as file:
    pickle.dump(xgb, file)

with open("target_enc.pkl", "wb") as file:
```

```
        pickle.dump(ab, file)


    # Load the model from the pickle file
    with open("xgboost_model.pkl", "rb") as file:
        loaded_model = pickle.load(file)
```