NETWORK INTRUSION DETECTION

Aim :

- To improve cyber security, machine learning algorithms can be implemented to detect cyber attacks.
- The approach involves analyzing network data to identify potential attacks by identifying correlations between various variables.
- By leveraging machine learning algorithms, the accuracy and efficiency of cyber attack detection can be improved. It will enhance the security of digital networks and systems.

Cyber attack data :

- The data is collected by the University of New South Wales (Australia). That includes records of different types of cyber attacks. The dataset contains network packets captured in the Cyber Range Lab of UNSW Canberra. The data is provided in two sets of training and testing data.
- The dataset includes nine types of attacks, including:

1. Fuzzers: Attack that involves sending random data to a system to test its resilience and identify any vulnerabilities.
2. Analysis: A type of attack that involves analyzing the system to identify its weaknesses and potential targets for exploitation.
3. Backdoors: Attack that involves creating a hidden entry point into a system for later use by the attacker.
4. DoS (Denial of Service): Attack that aims to disrupt the normal functioning of a system, making it unavailable to its users.
5. Exploits: Attack that leverages a vulnerability in a system to gain unauthorized access or control.
6. Generic: A catch-all category that includes a variety of different attack types that do not fit into the other categories.
7. Reconnaissance: Attack that involves gathering information about a target system, such as its vulnerabilities and potential entry points, in preparation for a future attack.
8. Shellcode: Attack that involves executing malicious code, typically in the form of shell scripts, on a target system.
9. Worms: A type of malware that spreads itself automatically to other systems, often causing harm in the process.

- These nine categories cover a wide range of attack types that can be used to exploit a system, and it is important to be aware of them to protect against potential security threats.

About Dataset

> These features are described in UNSW-NB15_features.csv file.

> A partition from this dataset is configured as a training set and testing set, namely, UNSW_NB15_training-set.csv and UNSW_NB15_testing-set.csv respectively.

> The number of records in the training set is 175,341 records and the testing set is 82,332 records from the different types, attack and normal.

The details of the UNSW-NB15 dataset are published in following the papers:

> Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015. Moustafa, Nour, and Jill Slay. "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 dataset and the comparison with the KDD99 dataset." Information Security Journal: A Global Perspective (2016): 1-14. Moustafa, Nour, et al. . "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks." IEEE Transactions on Big Data (2017). Moustafa, Nour, et al. "Big data analytics for intrusion detection system: statistical decision-making using finite dirichlet mixture models." Data Analytics and Decision Support for Cybersecurity. Springer, Cham, 2017. 127-156.

In this notebook, the operations conducted include:

- Preprocessing the data to prepare for training ML models.
- Training ML models based on cross-validation.
- Evaluating ML models based on testing data.
- Saving the model as pickle file and Evaluating the model

Context

```
#Import necessary packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# sklearn: data preprocessing
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder

# sklearn: train model
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_validate, StratifiedKFold
from sklearn.metrics import precision_recall_curve, precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix, classification_report

# sklearn classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

#save as pickle
import pickle

import warnings
warnings.filterwarnings('ignore')
```

```
train_df = pd.read_csv("/content/drive/MyDrive/main_project/UNSW_NB15_testing-set.csv")
test_df = pd.read_csv("/content/drive/MyDrive/main_project/UNSW_NB15_training-set.csv")
```

```
train_df.shape, test_df.shape
```

```
((175341, 45), (82332, 45))
```

```
all(test_df.columns == train_df.columns)
```

```
True
```

```
con_df = pd.concat([test_df,train_df]).drop('id',axis=1)
con_df = con_df.reset_index(drop=True)
```

```
con_df.head()
```

	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	..
0	0.000011	udp	-	INT	2	0	496	0	90909.0902	254	
1	0.000008	udp	-	INT	2	0	1762	0	125000.0003	254	
2	0.000005	udp	-	INT	2	0	1068	0	200000.0051	254	
3	0.000006	udp	-	INT	2	0	900	0	166666.6608	254	
4	0.000010	udp	-	INT	2	0	2126	0	100000.0025	254	

5 rows × 44 columns

Data Preprocessing and Analysis

```
con_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 257673 entries, 0 to 257672
Data columns (total 44 columns):
 #   Column             Non-Null Count   Dtype
```

```
---  ------             --------------   -----
 0   dur                257673 non-null  float64
 1   proto              257673 non-null  object
 2   service            257673 non-null  object
 3   state              257673 non-null  object
 4   spkts              257673 non-null  int64
 5   dpkts              257673 non-null  int64
 6   sbytes             257673 non-null  int64
 7   dbytes             257673 non-null  int64
 8   rate               257673 non-null  float64
 9   sttl               257673 non-null  int64
 10  dttl               257673 non-null  int64
 11  sload              257673 non-null  float64
 12  dload              257673 non-null  float64
 13  sloss              257673 non-null  int64
 14  dloss              257673 non-null  int64
 15  sinpkt             257673 non-null  float64
 16  dinpkt             257673 non-null  float64
 17  sjit               257673 non-null  float64
 18  djit               257673 non-null  float64
 19  swin               257673 non-null  int64
 20  stcpb              257673 non-null  int64
 21  dtcpb              257673 non-null  int64
 22  dwin               257673 non-null  int64
 23  tcprtt             257673 non-null  float64
 24  synack             257673 non-null  float64
 25  ackdat             257673 non-null  float64
 26  smean              257673 non-null  int64
 27  dmean              257673 non-null  int64
 28  trans_depth        257673 non-null  int64
 29  response_body_len  257673 non-null  int64
 30  ct_srv_src         257673 non-null  int64
 31  ct_state_ttl       257673 non-null  int64
 32  ct_dst_ltm         257673 non-null  int64
 33  ct_src_dport_ltm   257673 non-null  int64
 34  ct_dst_sport_ltm   257673 non-null  int64
 35  ct_dst_src_ltm     257673 non-null  int64
 36  is_ftp_login       257673 non-null  int64
 37  ct_ftp_cmd         257673 non-null  int64
 38  ct_flw_http_mthd   257673 non-null  int64
 39  ct_src_ltm         257673 non-null  int64
 40  ct_srv_dst         257673 non-null  int64
 41  is_sm_ips_ports    257673 non-null  int64
 42  attack_cat         257673 non-null  object
 43  label              257673 non-null  int64
dtypes: float64(11), int64(29), object(4)
memory usage: 86.5+ MB
```

```
# seperate the different dtypes and name their columns

numeric_cols = con_df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_cols = con_df.select_dtypes(include=['object']).columns.tolist()
date_cols = con_df.select_dtypes(include=['datetime64[ns]']).columns.tolist()

# con_df.columns = ['_'.join(col.split(' ')) for col in con_df.columns]
```

```
print("length :", len(numeric_cols))
print(numeric_cols)
print("length:" , len(categorical_cols)  , )
print(categorical_cols)
```

```
length : 40
['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit
length: 4
['proto', 'service', 'state', 'attack_cat']
```

No null values found in the dataset

```
con_df.isnull().sum()
```

```
dur                  0
proto                0
service              0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sttl                 0
dttl                 0
sload                0
```

```
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
djit                 0
swin                 0
stcpb                0
dtcpb                0
dwin                 0
tcprtt               0
synack               0
ackdat               0
smean                0
dmean                0
trans_depth          0
response_body_len    0
ct_srv_src           0
ct_state_ttl         0
ct_dst_ltm           0
ct_src_dport_ltm     0
ct_dst_sport_ltm     0
ct_dst_src_ltm       0
is_ftp_login         0
ct_ftp_cmd           0
ct_flw_http_mthd     0
ct_src_ltm           0
ct_srv_dst           0
is_sm_ips_ports      0
attack_cat           0
label                0
dtype: int64
```

Checking the Number of Unique values present in each features

```
con_df.nunique()
```

```
dur                  109945
proto                   133
service                  13
state                    11
spkts                   646
dpkts                   627
sbytes                 9382
dbytes                 8653
rate                 115763
sttl                     13
dttl                      9
sload                121356
dload                116380
sloss                   490
dloss                   476
sinpkt               114318
dinpkt               110270
sjit                 117101
djit                 114861
swin                     22
stcpb                114473
dtcpb                114187
dwin                     19
tcprtt                63878
synack                57366
ackdat                53248
smean                  1377
dmean                  1362
trans_depth              14
response_body_len      2819
ct_srv_src               57
ct_state_ttl              7
ct_dst_ltm               52
ct_src_dport_ltm         52
ct_dst_sport_ltm         35
ct_dst_src_ltm           58
is_ftp_login              4
ct_ftp_cmd                4
ct_flw_http_mthd         11
ct_src_ltm               52
ct_srv_dst               57
is_sm_ips_ports           2
attack_cat               10
label                     2
dtype: int64
```

```
# unique values of the Categorical data
for cols in categorical_cols :
  print(f"{cols }: ",con_df[cols].unique())
```

```
proto:  ['udp' 'arp' 'tcp' 'igmp' 'ospf' 'sctp' 'gre' 'ggp' 'ip' 'ipnip' 'st2'
 'argus' 'chaos' 'egp' 'emcon' 'nvp' 'pup' 'xnet' 'mux' 'dcn' 'hmp' 'prm'
 'trunk-1' 'trunk-2' 'xns-idp' 'leaf-1' 'leaf-2' 'irtp' 'rdp' 'netblt'
 'mfe-nsp' 'merit-inp' '3pc' 'idpr' 'ddp' 'idpr-cmtp' 'tp++' 'ipv6' 'sdrp'
 'ipv6-frag' 'ipv6-route' 'idrp' 'mhrp' 'i-nlsp' 'rvd' 'mobile' 'narp'
 'skip' 'tlsp' 'ipv6-no' 'any' 'ipv6-opts' 'cftp' 'sat-expak' 'ippc'
 'kryptolan' 'sat-mon' 'cpnx' 'wsn' 'pvp' 'br-sat-mon' 'sun-nd' 'wb-mon'
 'vmtp' 'ttp' 'vines' 'nsfnet-igp' 'dgp' 'eigrp' 'tcf' 'sprite-rpc' 'larp'
 'mtp' 'ax.25' 'ipip' 'aes-sp3-d' 'micp' 'encap' 'pri-enc' 'gmtp' 'ifmp'
 'pnni' 'qnx' 'scps' 'cbt' 'bbn-rcc' 'igp' 'bna' 'swipe' 'visa' 'ipcv'
 'cphb' 'iso-tp4' 'wb-expak' 'sep' 'secure-vmtp' 'xtp' 'il' 'rsvp' 'unas'
 'fc' 'iso-ip' 'etherip' 'pim' 'aris' 'a/n' 'ipcomp' 'snp' 'compaq-peer'
 'ipx-n-ip' 'pgm' 'vrrp' 'l2tp' 'zero' 'ddx' 'iatp' 'stp' 'srp' 'uti' 'sm'
 'smp' 'isis' 'ptp' 'fire' 'crtp' 'crudp' 'sccopmce' 'iplt' 'pipe' 'sps'
 'ib' 'icmp' 'rtp']
service:  ['-' 'http' 'ftp' 'ftp-data' 'smtp' 'pop3' 'dns' 'snmp' 'ssl' 'dhcp' 'irc'
 'radius' 'ssh']
state:  ['INT' 'FIN' 'REQ' 'ACC' 'CON' 'RST' 'CLO' 'ECO' 'PAR' 'URN' 'no']
attack_cat:  ['Normal' 'Reconnaissance' 'Backdoor' 'DoS' 'Exploits' 'Analysis'
 'Fuzzers' 'Worms' 'Shellcode' 'Generic']
```

```
#service feature contains a " - " value replaced by None
con_df['service'] = con_df['service'].replace("-", "None")
print(con_df["service"].unique())
```

```
['None' 'http' 'ftp' 'ftp-data' 'smtp' 'pop3' 'dns' 'snmp' 'ssl' 'dhcp'
 'irc' 'radius' 'ssh']
```

Numerical Types

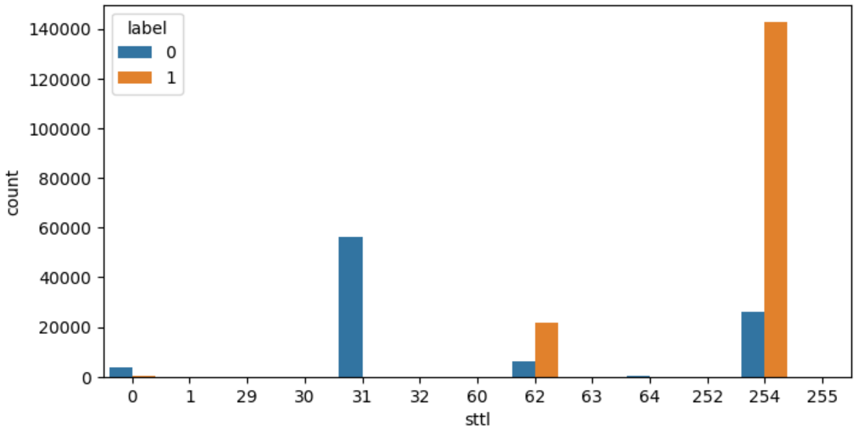
- CONTINUOUS
- DISCREATE

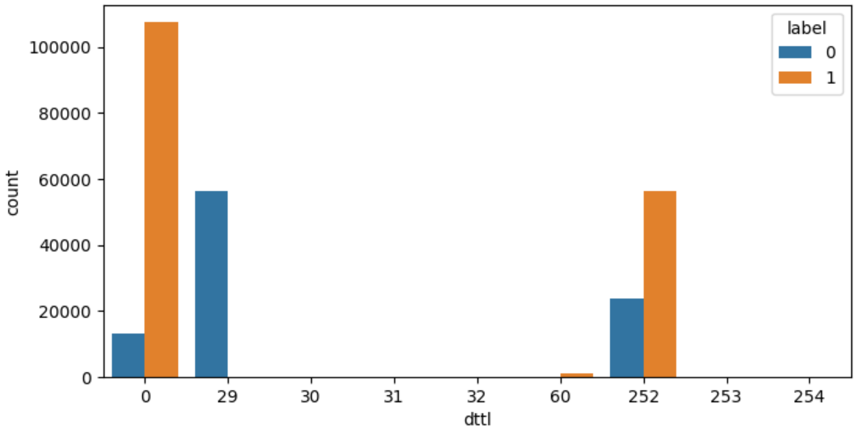
```
#DISCREATE
dis_val = [col for  col in numeric_cols if len(con_df[col].unique())<25]
print("discreate :",dis_val)

#CONTINUOUS
conti_val = [col for col in numeric_cols if col not in dis_val]
print("Continuous :", conti_val)
```

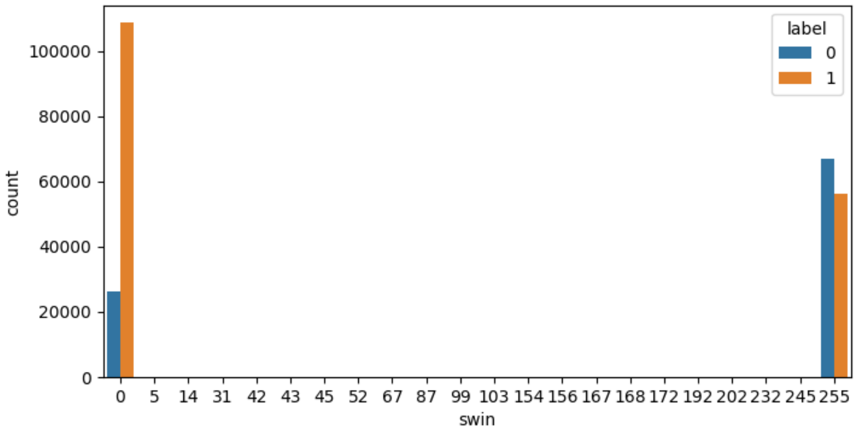
```
discreate : ['sttl', 'dttl', 'swin', 'dwin', 'trans_depth', 'ct_state_ttl', 'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'is_sm
Continuous : ['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit',
```

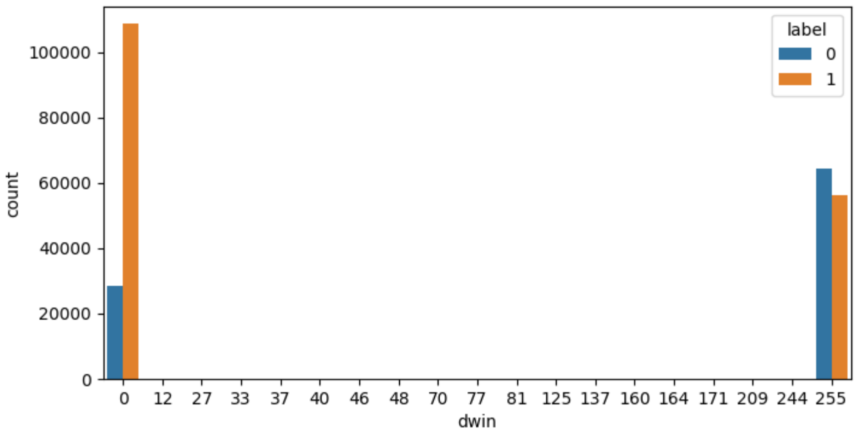
```
# prompt: create a for loop code to check the outlayers using box plot for all numerical datatype columns

import matplotlib.pyplot as plt

for col in numeric_cols:
    plt.figure(figsize=(5, 3))
    plt.boxplot(con_df[col])
    plt.title(col)
    plt.show()
```

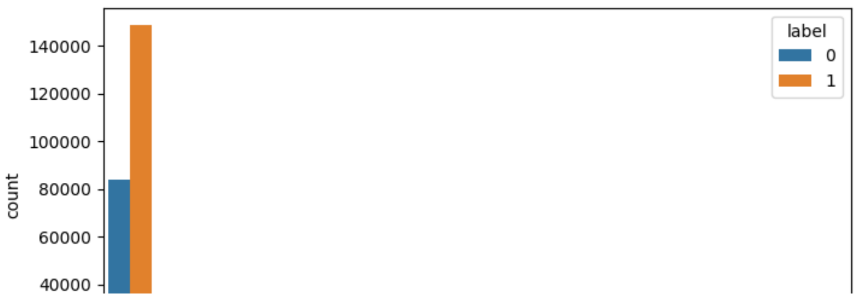
```
#Discreate varibales

for col in dis_val:
    plt.figure(figsize=(8,4))
    sns.countplot(con_df,x=con_df[col],hue='label')
    plt.xlabel(col)
    plt.ylabel('count')
    plt.show()
```

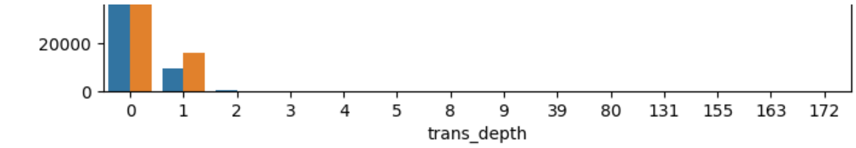

label
0
1
count
140000
120000
100000
80000
60000
40000
20000
0
0 1 29 30 31 32 60 62 63 64 252 254 255
sttl

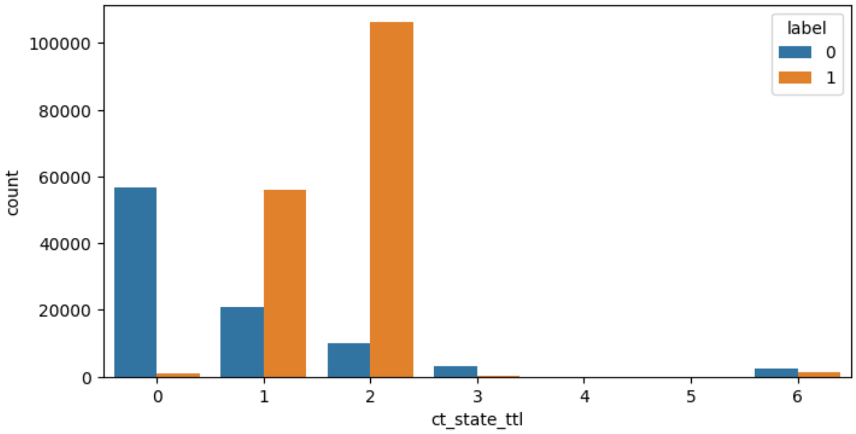

label
0
1
count
100000
80000
60000
40000
20000
0
0 29 30 31 32 60 252 253 254
dttl

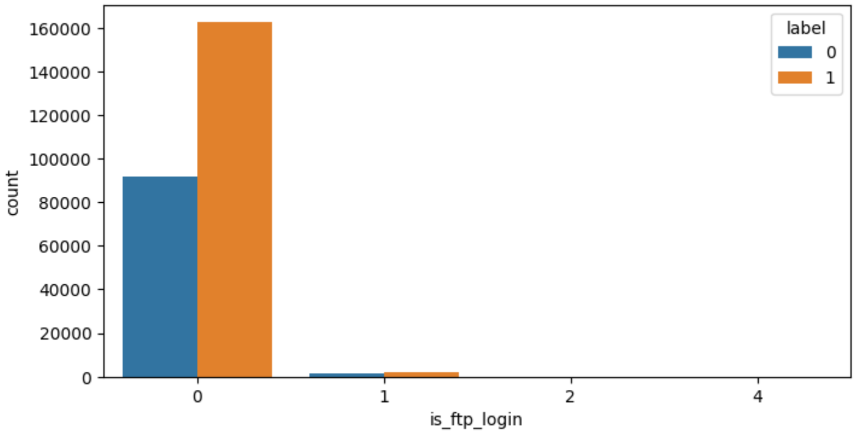

label
0
1
count
100000
80000
60000
40000
20000
0
0 5 14 31 42 43 45 52 67 87 99 103 154 156 167 168 172 192 202 232 245 255
swin

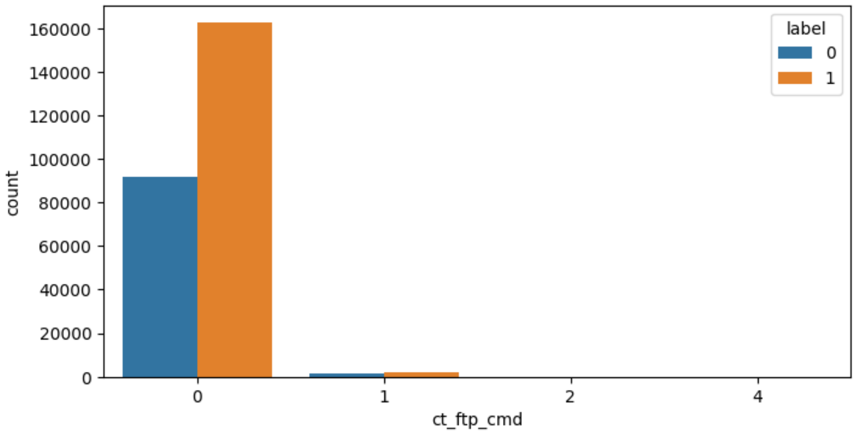

label
0
1
count
100000
80000
60000
40000
20000
0
0 12 27 33 37 40 46 48 70 77 81 125 137 160 164 171 209 244 255
dwin

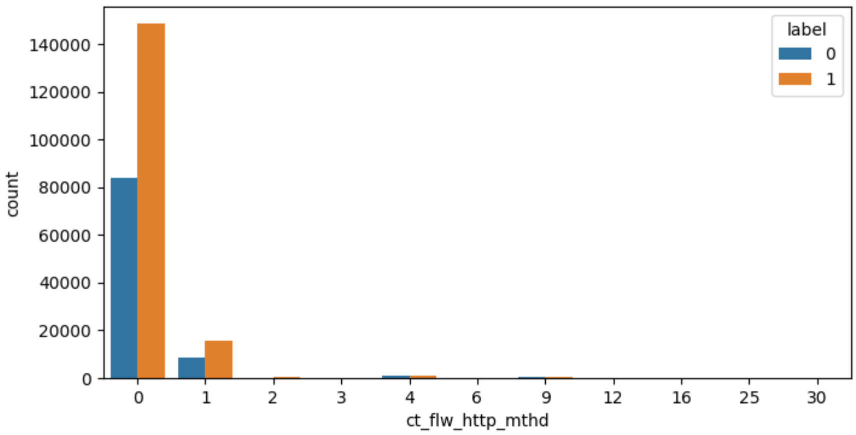

label
0
1
count
140000
120000
100000
80000
60000
40000


20000
0
0
1
2
3
4
5
8
9
39
80
131
155
163
172
trans_depth

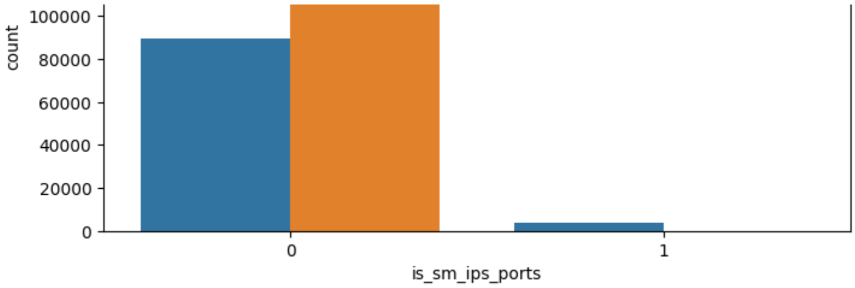

label
0
1
count
100000
80000
60000
40000
20000
0
0
1
2
3
4
5
6
ct_state_ttl

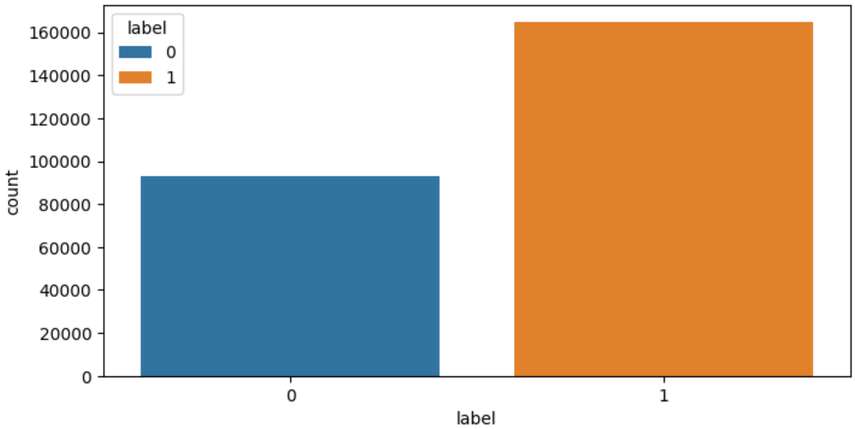

label
0
1
count
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
4
is_ftp_login


label
0
1
count
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
4
ct_ftp_cmd

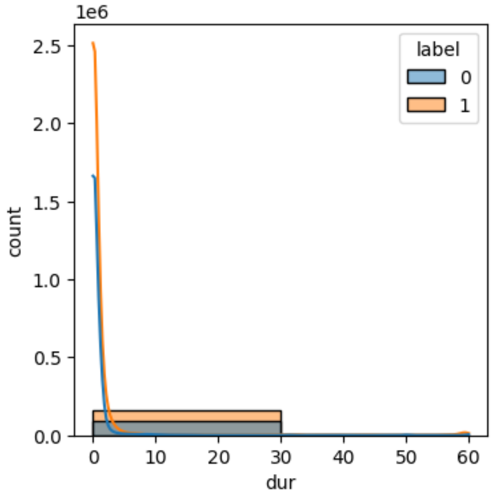

label
0
1
count
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
3
4
6
9
12
16
25
30
ct_flw_http_mthd

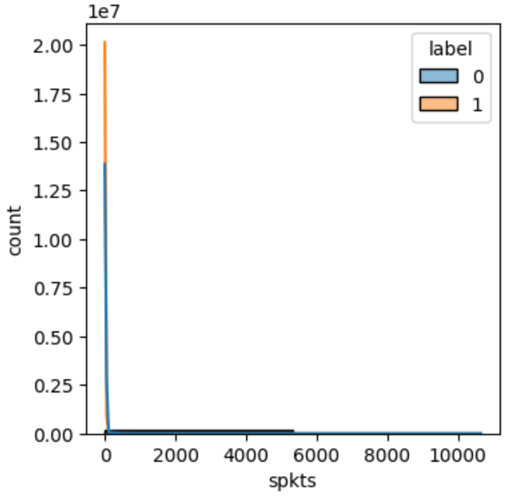

label
0
1
160000
140000
120000

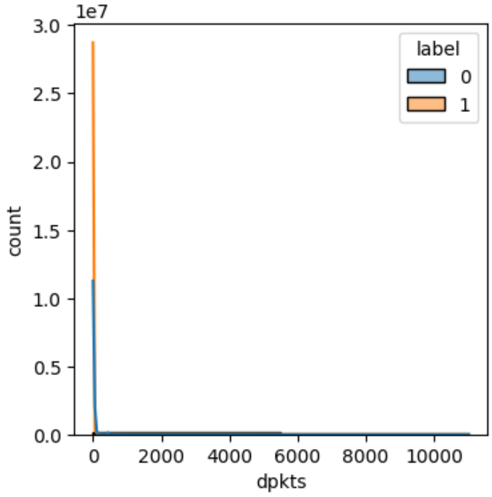


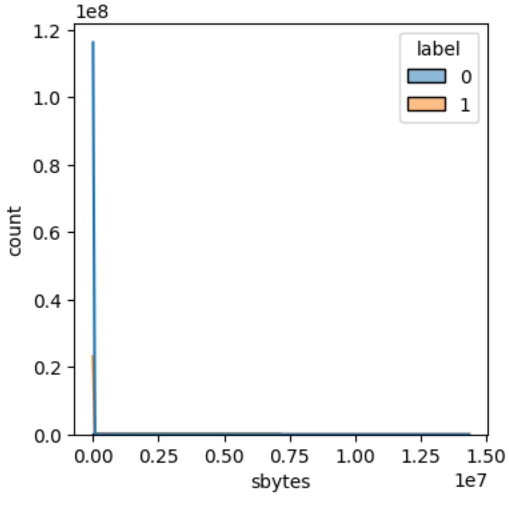



```
# Continuous Variables

for col in conti_val:
  plt.figure(figsize=(4,4))
  sns.histplot(con_df,x=con_df[col],kde=True,hue='label',bins=2)
  plt.xlabel(col)
  plt.ylabel('count')
  plt.show()
```

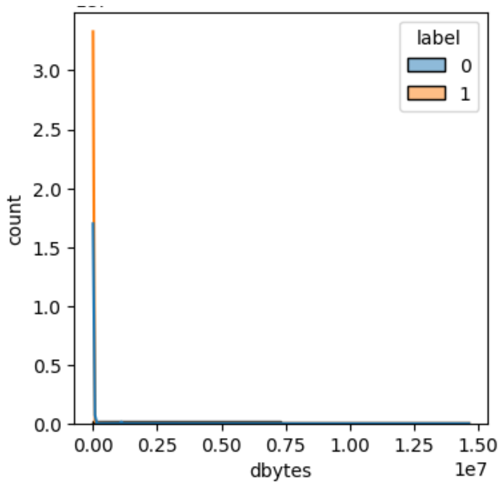

1e6
2.5
2.0
1.5
1.0
0.5
0.0
count
label
0
1
0 10 20 30 40 50 60
dur

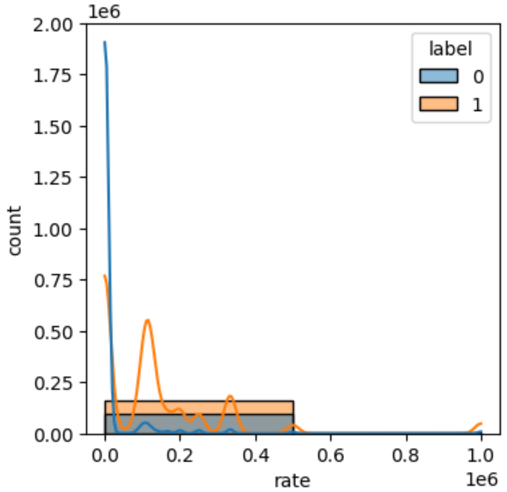

1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
count
label
0
1
0 2000 4000 6000 8000 10000
spkts

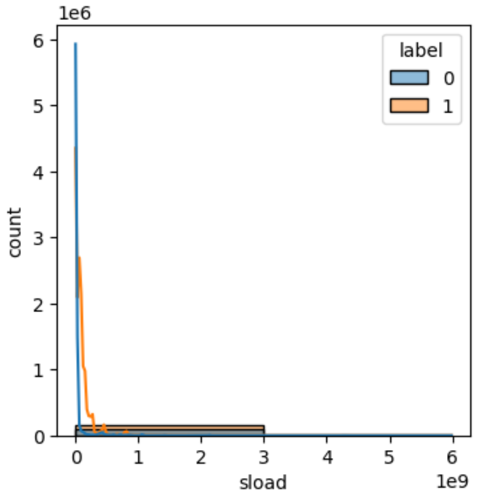

1e7
3.0
2.5
2.0
1.5
1.0
0.5
0.0
count
label
0
1
0 2000 4000 6000 8000 10000
dpkts

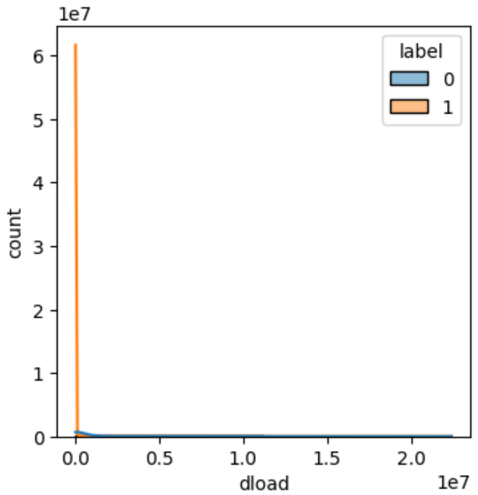

1e8
1.2
1.0
0.8
0.6
0.4
0.2
0.0
count
label
0
1
0.00 0.25 0.50 0.75 1.00 1.25 1.50
sbytes
1e7

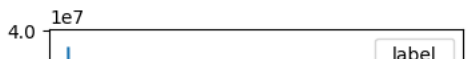
1e7

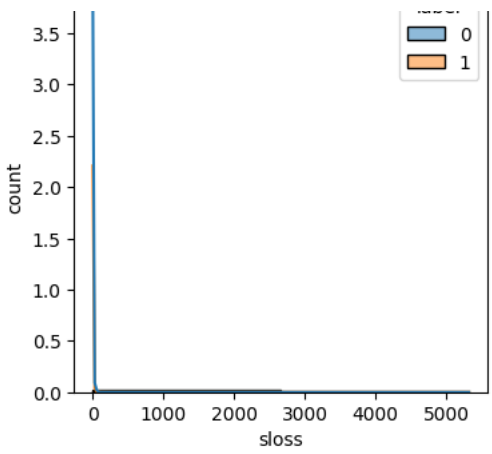

label
0
1
count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
dbytes
1e7

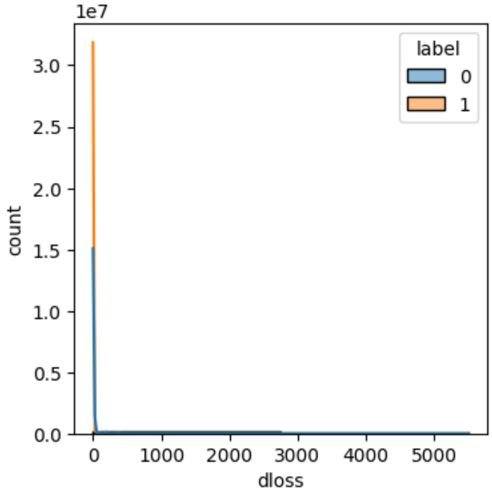

1e6
label
0
1
count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.0
0.2
0.4
0.6
0.8
1.0
rate
1e6

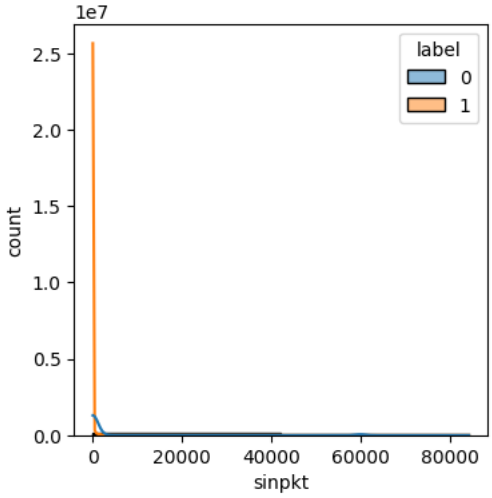

1e6
label
0
1
count
0
1
2
3
4
5
6
sload
1e9

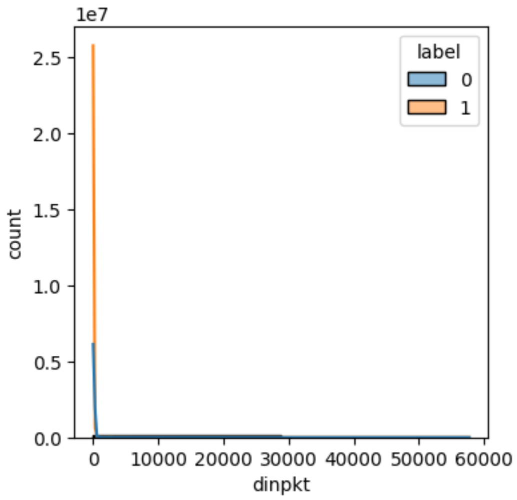

1e7
label
0
1
count
0
1
2
3
4
5
6
0.0
0.5
1.0
1.5
2.0
dload
1e7

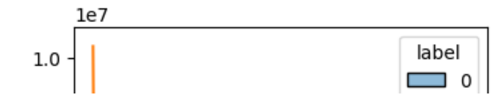

1e7
4.0
label

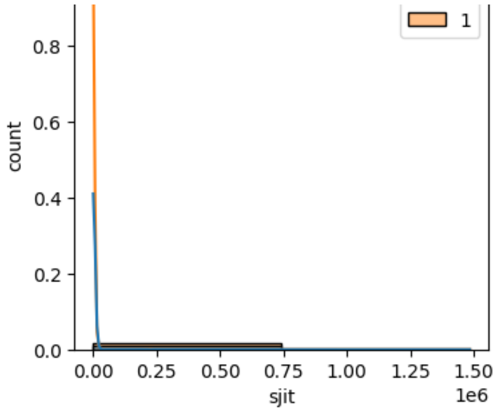

label
0
1
count
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0 1000 2000 3000 4000 5000
sloss

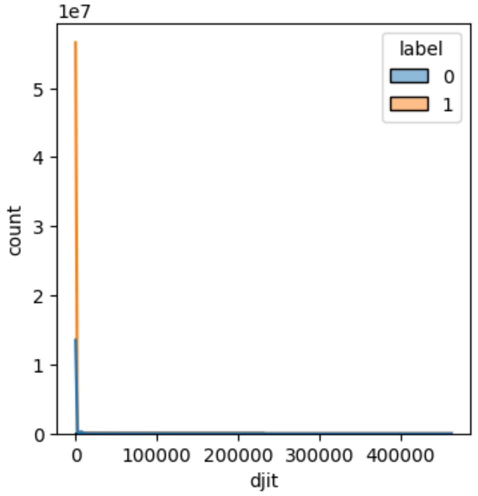

1e7
label
0
1
count
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0 1000 2000 3000 4000 5000
dloss

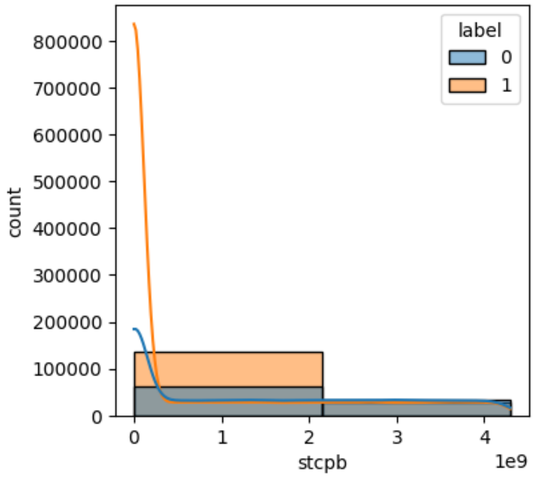

1e7
label
0
1
count
2.5
2.0
1.5
1.0
0.5
0.0
0 20000 40000 60000 80000
sinpkt

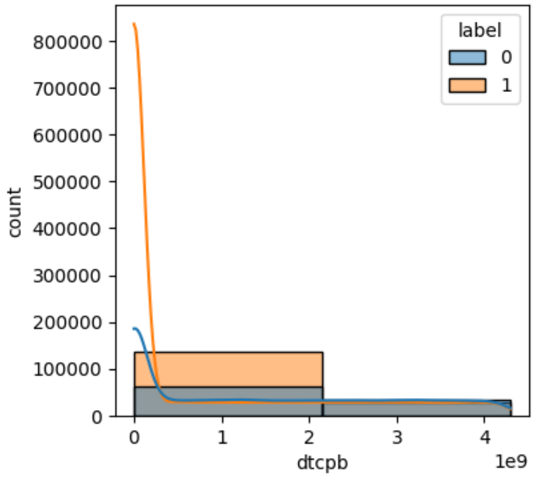

1e7
label
0
1
count
2.5
2.0
1.5
1.0
0.5
0.0
0 10000 20000 30000 40000 50000 60000
dinpkt


1e7
1.0
label
0

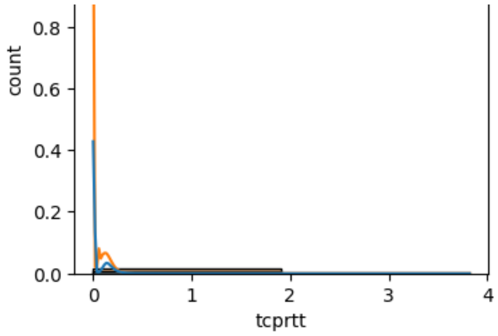

1
count
sjit
1e6

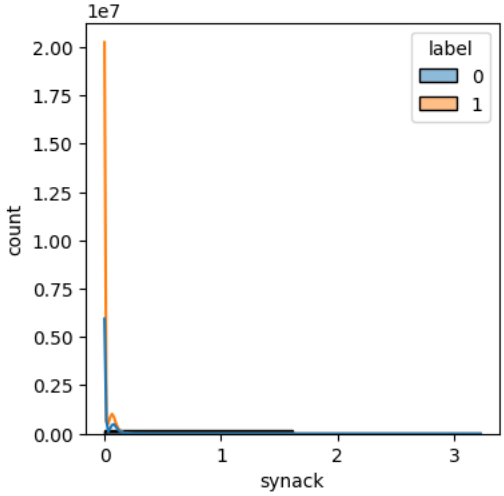

1e7
label
0
1
count
djit

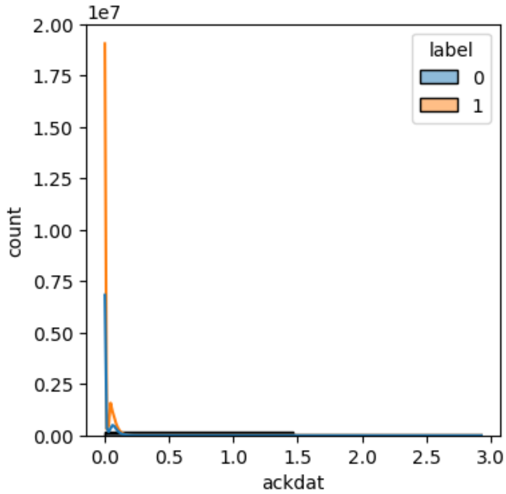

label
0
1
count
stcpb
1e9

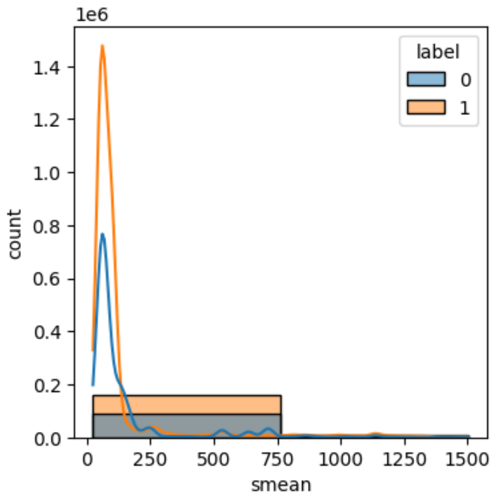

label
0
1
count
dtcpb
1e9


1e7
label
0
1

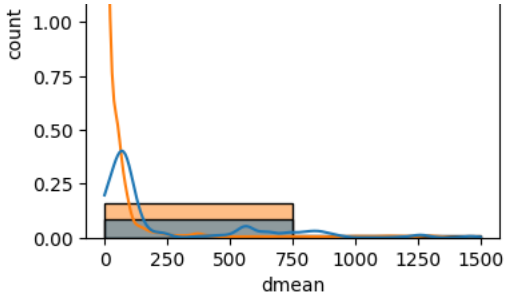

count
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
tcprtt

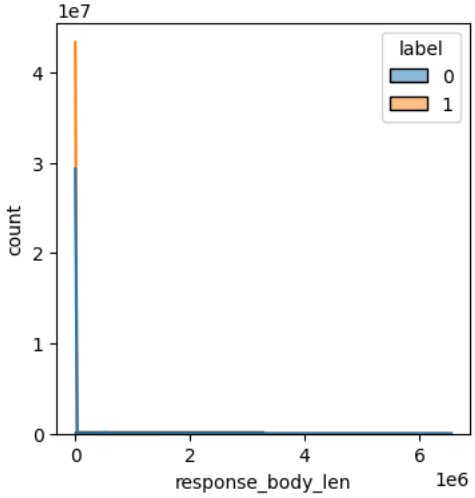

1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
count
label
0
1
0
1
2
3
synack

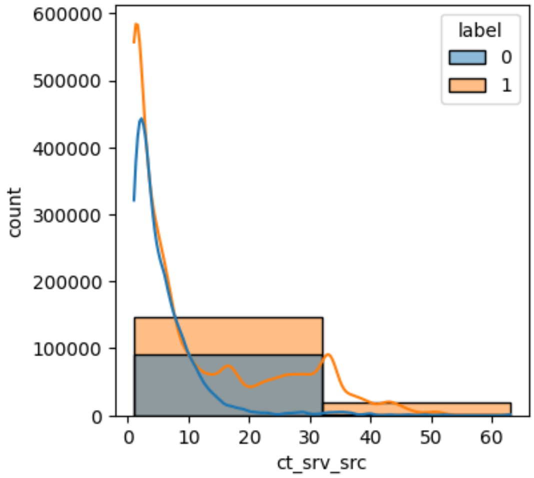

1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
count
label
0
1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
ackdat

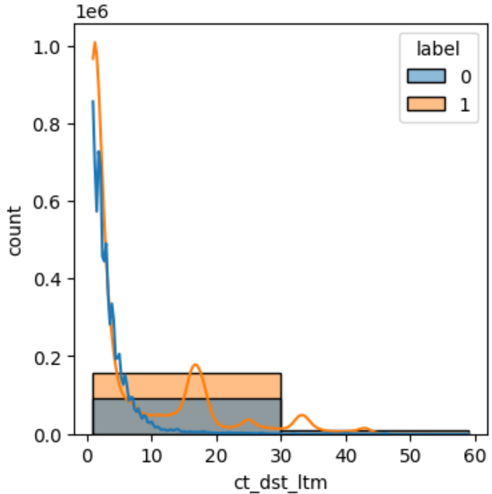

1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
count
label
0
1
0
250
500
750
1000
1250
1500
smean


1e6
1.75
1.50
1.25
label
0
1

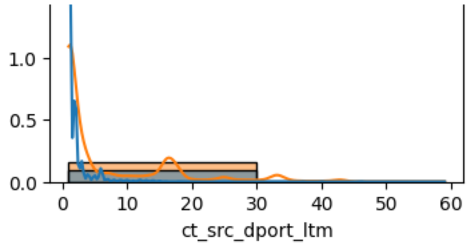

count
1.00
0.75
0.50
0.25
0.00
0 250 500 750 1000 1250 1500
dmean

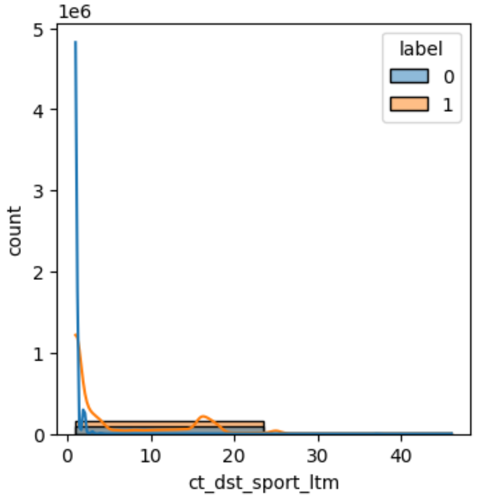

1e7
label
0
1
count
4
3
2
1
0
0 2 4 6
response_body_len
1e6

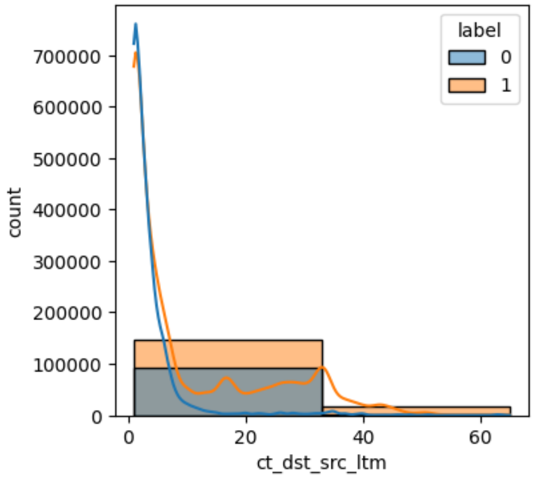

label
0
1
count
600000
500000
400000
300000
200000
100000
0
0 10 20 30 40 50 60
ct_srv_src

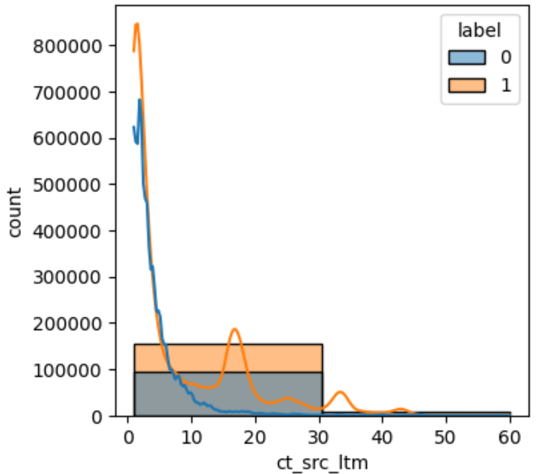

1e6
label
0
1
count
1.0
0.8
0.6
0.4
0.2
0.0
0 10 20 30 40 50 60
ct_dst_ltm

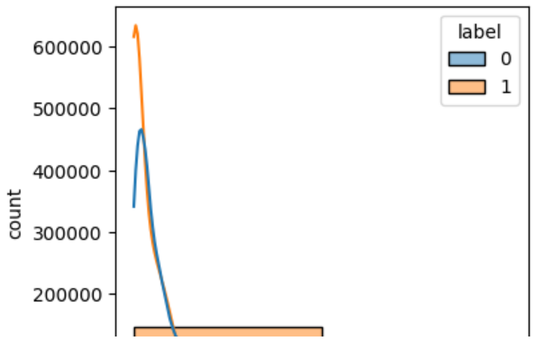

1e6
label
0
1
count
3.0
2.5
2.0
1.5

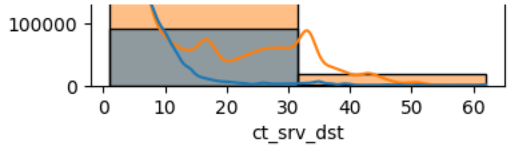

1.0
0.5
0.0
0
10
20
30
40
50
60
ct_src_dport_ltm


1e6
count
label
0
1
ct_dst_sport_ltm


count
label
0
1
ct_dst_src_ltm


count
label
0
1
ct_src_ltm


count
label
0
1



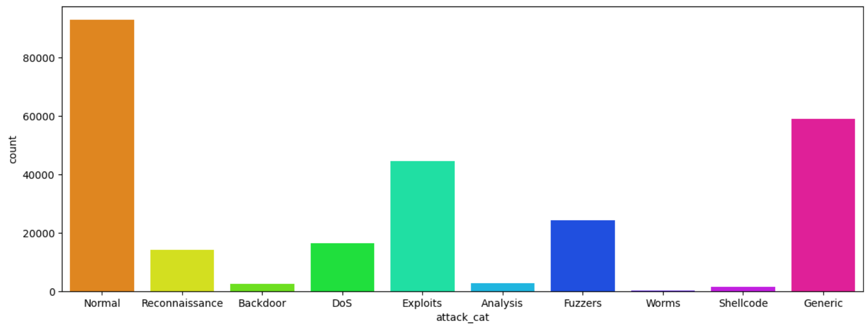
Visualizing the Categorical Variables

Types of Attacks

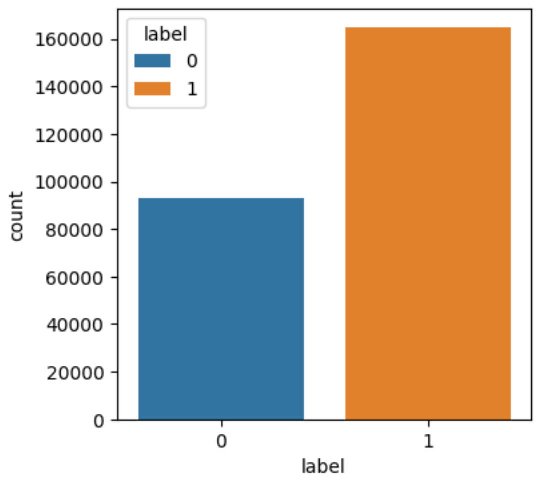
attack_cat: This dataset has nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms.

Label: 0 for normal and 1 for attack records

```
plt.figure(figsize=(14,5))
sns.countplot(con_df,x=con_df["attack_cat"], palette='hsv')
plt.xlabel("attack_cat")
plt.ylabel('count')
plt.show()
```




```
plt.figure(figsize=(4,4))
sns.countplot(con_df,x=con_df["label"],hue="label")
plt.xlabel("label")
plt.ylabel('count')
plt.show()
```



Splitting Test and Train sets

```
#convert the object datatype tp int - Categorical features
for col in ['proto', 'service', 'state','attack_cat']:
    con_df[col] = con_df[col].astype('category').cat.codes.astype(int)
```

```
#Mapping for the labels coverted into Intergers

for col in ['proto', 'service', 'state','attack_cat']:
    con_df[col] = con_df[col].astype('category')
    print(f"{col} mapping: ")
    print(dict(enumerate(con_df[col].cat.categories)))
    con_df[col] = con_df[col].cat.codes.astype(int)
```

```
proto mapping:
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: 11, 12: 12, 13: 13, 14: 14, 15: 15, 16: 16, 17: 17, 18: 18,
service mapping:
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: 11, 12: 12}
state mapping:
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10}
attack_cat mapping:
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}
```

```
con_df.head()
```

	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	..
0	0.000011	119	0	5	2	0	496	0	90909.0902	254	
1	0.000008	119	0	5	2	0	1762	0	125000.0003	254	
2	0.000005	119	0	5	2	0	1068	0	200000.0051	254	
3	0.000006	119	0	5	2	0	900	0	166666.6608	254	
4	0.000010	119	0	5	2	0	2126	0	100000.0025	254	

5 rows × 44 columns

```
con_df['attack_cat'].unique()
```

```
array([6, 7, 1, 2, 3, 0, 4, 9, 8, 5])
```

```
# Splitting X and Y Values

X= con_df.drop(columns = ['attack_cat', 'label'])

y2 = con_df['label'].values  # Classes With Attack and Normal Labels
y1 = con_df['attack_cat'].values # Classes with Types of  Attacks

X_train, X_test, y1_train, y1_test = train_test_split(X, y1, test_size=0.3, random_state=11)
X_train, X_test, y2_train, y2_test = train_test_split(X, y2, test_size=0.3, random_state=11)


#from splitted data

X_test = test_df.drop(columns = ['attack_cat', 'label'])
y2_test = test_df['label'].values
y1_test = test_df['attack_cat'].values



# X_train = train_df.drop(columns = ['attack_cat', 'label'])
# y1_train = train_df['attack_cat'].values
# y2_train = train_df["label"].values


feature_names = list(X.columns)

print("X_train shape: ", X_train.shape)
print("y_train shape: ", y1_train.shape,y2_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y1_test.shape,y2_test.shape)
```

```
X_train shape:  (180371, 42)
y_train shape:  (180371,) (180371,)
X_test shape:  (82332, 43)
y_test shape:  (82332,) (82332,)
```

```
X_test.head()
```

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...
0	1	0.000011	udp	-	INT	2	0	496	0	90909.0902	...
1	2	0.000008	udp	-	INT	2	0	1762	0	125000.0003	...
2	3	0.000005	udp	-	INT	2	0	1068	0	200000.0051	...
3	4	0.000006	udp	-	INT	2	0	900	0	166666.6608	...
4	5	0.000010	udp	-	INT	2	0	2126	0	100000.0025	...

5 rows × 43 columns

```
# determine categorical and numerical columns
numerical_cols = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = X_train.select_dtypes(include=['object', 'bool']).columns
```

```
len(numerical_cols), len(categorical_cols)
```

```
(42, 0)
```

```
# define the transformation methods for the columns
t = [('ohe', OneHotEncoder(drop='first'), categorical_cols),
    ('scale', StandardScaler(), numerical_cols)]

col_trans = ColumnTransformer(transformers=t)

# fit the transformation on training data
col_trans.fit(X_train)
```

```
ColumnTransformer
ohe                scale
OneHotEncoder      StandardScaler
```

```
X_train_transform = col_trans.transform(X_train)
```

```
X_test_transform = col_trans.transform(X_test)
```

```
# look at the transformed training data
X_train_transform.shape , X_test_transform.shape
```

```
((180371, 42), (77302, 42))
```

```
X_test_transform.shape
```

```
(77302, 42)
```

```
# Note that the distinct values/labels in `y2` target are 1 and 2.
pd.unique(y1),pd.unique(y2)
```

```
(array([6, 7, 1, 2, 3, 0, 4, 9, 8, 5]), array([0, 1]))
```

```
y1_train , y2_train
```

```
(array([5, 6, 6, ..., 5, 6, 3]), array([1, 0, 0, ..., 1, 0, 1]))
```

```
# Define a LabelEncoder() transformation method and fit on y1_train
target_trans = LabelEncoder()
target_trans.fit(y1_train)
```

```
# apply transformation method on y1_train and y1_test
y1_train_transform = target_trans.transform(y1_train)
y1_test_transform = target_trans.transform(y1_test)
```

```
# view the transformed y1_train
y1_train_transform
```

```
array([5, 6, 6, ..., 5, 6, 3])
```

```
# Define a LabelEncoder() transformation method and fit on y2_train
target_tran = LabelEncoder()
target_tran.fit(y2_train)
y2_train_transform = target_tran.transform(y2_train)
y2_test_transform = target_tran.transform(y2_test)
```

```
# view the transformed y2_train
y2_test_transform
```

```
array([1, 1, 0, ..., 1, 1, 0])
```

Training the ML Models

```
# ===== Step 1: cross-validation ========
# define a Logistic Regression classifier
clf = LogisticRegression(solver='lbfgs', random_state=123, max_iter = 4000)

# define  Stratified 5-fold cross-validator
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=123)

# define metrics for evaluating
scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

# perform the 5-fold CV and get the metrics results
cv_results = cross_validate(estimator=clf,X=X_train_transform,y=y2_train_transform,scoring=scoring,
                            cv=cv,return_train_score=False)
```

```
cv_results
```

```
{'fit_time': array([7.03564334, 5.33885598, 6.05562663, 4.49421859, 6.11079121]),
 'score_time': array([0.06029081, 0.05586624, 0.05689836, 0.06166244, 0.06509757]),
 'test_accuracy': array([0.89593902, 0.89859733, 0.89723901, 0.89704496, 0.89942895]),
 'test_precision': array([0.8832227 , 0.88688361, 0.88286325, 0.88264882, 0.88545101]),
 'test_recall': array([0.96457872, 0.96418805, 0.96744368, 0.96740027, 0.96770413]),
 'test_f1': array([0.92210972, 0.92392163, 0.92322031, 0.9230833 , 0.92475215]),
 'test_roc_auc': array([0.96516718, 0.96602007, 0.96578601, 0.96488457, 0.96610454])}
```

```
cv_results['test_accuracy'].mean()
```

```
0.8976498534178858
```

```
# ======== Step 2: Evaluate the model using testing data =======

# fit the Logistic Regression model
clf.fit(X=X_train_transform, y=y2_train_transform)

# predition on testing data
y_pred_class = clf.predict(X=X_test_transform)
y_pred_score = clf.predict_proba(X=X_test_transform)[:, 1]

# AUC of ROC
auc_ontest = roc_auc_score(y_true=y2_test_transform, y_score=y_pred_score)
# confusion matrix
cm_ontest = confusion_matrix(y_true=y2_test_transform, y_pred=y_pred_class)
# precision score
precision_ontest = precision_score(y_true=y2_test_transform, y_pred=y_pred_class)
# recall score
recall_ontest = recall_score(y_true=y2_test_transform, y_pred=y_pred_class)
# classifition report
cls_report_ontest = classification_report(y_true=y2_test_transform, y_pred=y_pred_class)

# print the above results
print('The model scores {:1.5f} ROC AUC on the test set.'.format(auc_ontest))
print('The precision score on the test set: {:1.5f}'.format(precision_ontest))
print('The recall score on the test set: {:1.5f}'.format(recall_ontest))
print('Confusion Matrix:\n', cm_ontest)
# Print classification report:
print('Classification Report:\n', cls_report_ontest)
```

```
The model scores 0.96686 ROC AUC on the test set.
The precision score on the test set: 0.88762
The recall score on the test set: 0.96672
Confusion Matrix:
 [[21757  6057]
 [ 1647 47841]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.78      0.85     27814
           1       0.89      0.97      0.93     49488

    accuracy                           0.90     77302
   macro avg       0.91      0.87      0.89     77302
weighted avg       0.90      0.90      0.90     77302
```

ML - Models

We will implement several ML models

1. LogisticRegression()
2. DecisionTreeClassifier()
3. RandomForestClassifier()
4. MLPClassifier()

Note : In MLPClassifier() we set the solver as lbfgs which has better performance for small size of data. Also, we set the maximum iterations as 5000 to ensure convergence. random_state is used to ensure reproducible results.

Binary Classification Models Using Y1

```
# Define four models
models = [('LogisticRegression', LogisticRegression(random_state=123, max_iter=5000)),
          ('DecisionTree', DecisionTreeClassifier(random_state=123)),
          ('RandomForest', RandomForestClassifier(random_state=123)),
          ('MultiLayerPerceptron', MLPClassifier(random_state=123, solver='adam', max_iter=8000))
         ]
```

```
#We could check the hyperparameters values in these models:

for model_name, clf in models:
    print(clf)
```

```
LogisticRegression(max_iter=5000, random_state=123)
DecisionTreeClassifier(random_state=123)
RandomForestClassifier(random_state=123)
MLPClassifier(max_iter=8000, random_state=123)
```

Code to perform the above four ML models and store their cross-validation results and evaluation results on testing data.

```
# define several lists and dataframe to store the CV results and evaluation results on testing data
model_names_list = []
cv_fit_time_mean_list = []
cv_accuracy_mean_list = []
cv_precision_mean_list = []
cv_recall_mean_list = []
cv_f1_mean_list = []
cv_roc_auc_mean_list = []

test_accuracy_list = []
test_precision_list = []
test_recall_list = []
test_f1_list = []
test_roc_auc_list = []

test_roc_curve_df = pd.DataFrame()


for model_name, clf in models:

    # ==== Step 1: Cross-validation =====

    # define  Stratified 5-fold cross-validator
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=123)
    # define metrics for evaluating
    scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
    # perform the 5-fold CV and get the metrics results
    cv_results = cross_validate(estimator=clf,
                                X=X_train_transform,
                                y=y2_train_transform,
                                scoring=scoring,
                                cv=cv,
                                return_train_score=False)  # prevent to show the train scores on cv splits.

    # calculate the mean values of those scores
    cv_fit_time_mean = cv_results['fit_time'].mean()
    cv_accuracy_mean = cv_results['test_accuracy'].mean()
    cv_precision_mean = cv_results['test_precision'].mean()
    cv_recall_mean = cv_results['test_recall'].mean()
    cv_f1_mean = cv_results['test_f1'].mean()
    cv_roc_auc_mean = cv_results['test_roc_auc'].mean()

    # store CV results into those lists
    model_names_list.append(model_name)
    cv_fit_time_mean_list.append(cv_fit_time_mean)
    cv_accuracy_mean_list.append(cv_accuracy_mean)
    cv_precision_mean_list.append(cv_precision_mean)
    cv_recall_mean_list.append(cv_recall_mean)
    cv_f1_mean_list.append(cv_f1_mean)
    cv_roc_auc_mean_list.append(cv_roc_auc_mean)

    # ==== Step 2: Evaluation on Testing data =====

    # fit model
    clf.fit(X=X_train_transform, y=y2_train_transform)

    # predition on testing data

    # predicted label or class
    y_pred_class = clf.predict(X=X_test_transform)

    # predicted probability of the label 1
    y_pred_score = clf.predict_proba(X=X_test_transform)[:, 1]

    #save as pickle file
    filename = f"{model_name}.pkl"
    with open(filename, "wb") as f:
      pickle.dump(clf, f)

    # accuracy
    accuracy_ontest = accuracy_score(y_true=y2_test_transform, y_pred=y_pred_class)

    # auc of ROC
    auc_ontest = roc_auc_score(y_true=y2_test_transform, y_score=y_pred_score)

    # precision score
    precision_ontest = precision_score(y_true=y2_test_transform, y_pred=y_pred_class)

    # recall score
    recall_ontest = recall_score(y_true=y2_test_transform, y_pred=y_pred_class)

    # F1 score
```

```
    f1_ontest = f1_score(y_true=y2_test_transform, y_pred=y_pred_class)

    # roc curve dataframe
    fpr, tpr, threshold_roc = roc_curve(y_true=y2_test_transform, y_score=y_pred_score)

    roc_df = pd.DataFrame(list(zip(fpr, tpr, threshold_roc)),
                          columns=['False Positive Rate', 'True Positive Rate', 'Threshold'])

    roc_df['Model'] = '{} (AUC = {:.3f})'.format(model_name, auc_ontest)


    # store the above values
    test_accuracy_list.append(accuracy_ontest)
    test_roc_auc_list.append(auc_ontest)
    test_precision_list.append(precision_ontest)
    test_recall_list.append(recall_ontest)
    test_f1_list.append(f1_ontest)

    test_roc_curve_df = pd.concat([test_roc_curve_df, roc_df],
                                  ignore_index=True)
```

Model Comparison

```
results_dict = {'Model Name': model_names_list,
                'CV Fit Time': cv_fit_time_mean_list,
                'CV Accuracy mean': cv_accuracy_mean_list,
                'CV Precision mean': cv_precision_mean_list,
                'CV Recall mean': cv_recall_mean_list,
                'CV F1 mean': cv_f1_mean_list,
                'CV AUC mean': cv_roc_auc_mean_list,
                'Test Accuracy': test_accuracy_list,
                'Test Precision': test_precision_list,
                'Test Recall': test_recall_list,
                'Test F1': test_f1_list,
                'Test AUC': test_roc_auc_list
                }

results_df = pd.DataFrame(results_dict)

# sort the results according to F1 score on testing data
results_df.sort_values(by='Test F1', ascending=False)
```

	Model Name	CV Fit Time	CV Accuracy mean	CV Precision mean	CV Recall mean	CV F1 mean	CV AUC mean	Acc
2	RandomForest	22.644684	0.948689	0.961804	0.957685	0.959740	0.991346	0.9
3	MultiLayerPerceptron	88.148951	0.935832	0.951307	0.948162	0.949680	0.987619	0.9
1	DecisionTree	2.073154	0.934707	0.949142	0.948587	0.948863	0.930897	0.9

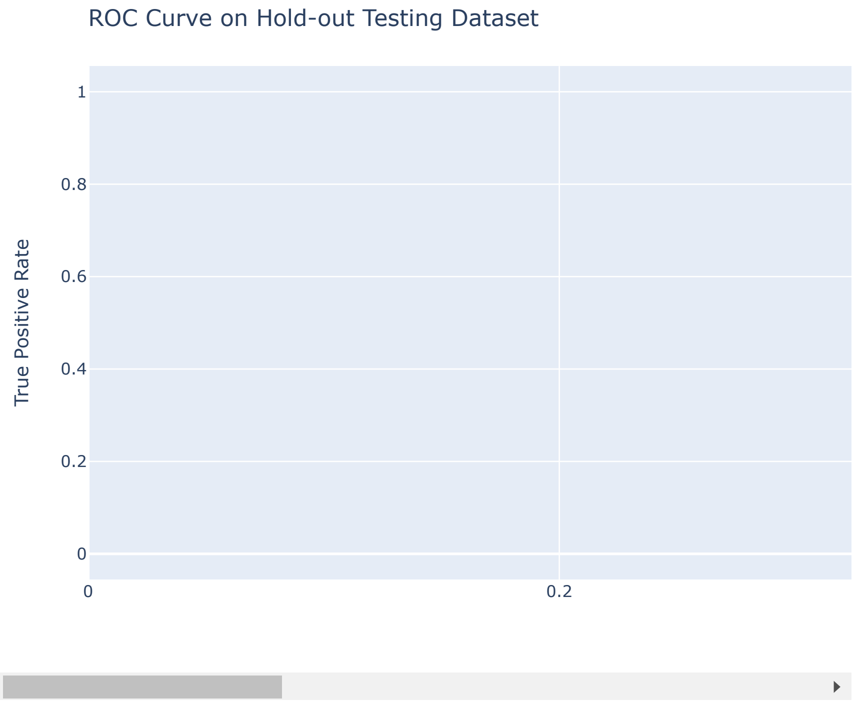
```
test_roc_curve_df.head()
```

	False Positive Rate	True Positive Rate	Threshold	Model
0	0.000000	0.000000	2.0	LogisticRegression (AUC = 0.967)
1	0.000072	0.000707	1.0	LogisticRegression (AUC = 0.967)
2	0.000072	0.000768	1.0	LogisticRegression (AUC = 0.967)

```
# !pip install plotly
# !pip install cufflinks
```

```
# plotly imports
import plotly.express as px
import plotly.graph_objects as go
```

```
ROC_fig = px.line(test_roc_curve_df, x='False Positive Rate',y='True Positive Rate',color='Model',hover_data=['Threshold'])

ROC_fig.update_layout(
    legend=go.layout.Legend(x=0.5,y=0.1,traceorder="normal",font=dict(# family="sans-serif",size=9,color="black"),
        bgcolor="LightSteelBlue",
        bordercolor="Black",
        borderwidth=2),
    title=go.layout.Title(text="ROC Curve on Hold-out Testing Dataset",xref="paper",x=0),
    xaxis=go.layout.XAxis(
        title=go.layout.xaxis.Title(
            text="False Positive Rate")),
    yaxis=go.layout.YAxis(
        title=go.layout.yaxis.Title(
            text="True Positive Rate")))
ROC_fig.show()
```



Using the saved Pickle File (Models) Validating the Prediction of the model's Accuracy - Y1

- Y2 model
- Top

```
# Load the saved RandomForest model
with open("/content/drive/MyDrive/main_project/saved_pickle_models/RandomForest.pkl", "rb") as f:
    loaded_model = pickle.load(f)

# # Take a random row from the test set
# import random
# random_row_index = random.randint(0, len(X_train_transform) - 1)
# random_row = X_train_transform[random_row_index]

#random_row=[-0.1487111723733928, 0.13476510922023754, -0.6923040283191014, -0.384387903872219 77, -0.074408064173589, -0.11286019724893

# Predict the class and probability for the random row
predicted_class = loaded_model.predict([random_row])
predicted_probability = loaded_model.predict_proba([random_row])[0][1]

# Print the results
print(f"Predicted class: {predicted_class[0]}")
print(f"Predicted probability of being class 1: {predicted_probability:.2f}")
```

```
Predicted class: 0
Predicted probability of being class 1: 0.00
```

Taking Random Samples for evaluation the pickle model

```
#Taking Samples on each Category for test the model

# Get the unique attack categories
unique_attack_cats = con_df['attack_cat'].unique()

# Initialize empty lists for each attack category
attack_cat_data = {}
for cat in unique_attack_cats:
  attack_cat_data[cat] = []

# Loop through the rows in X_test_transform
for i in range(len(X_test_transform)):
  # Get the attack category for the current row
  attack_cat = y1_test[i]

  # If the attack category is in the unique categories list
  if attack_cat in unique_attack_cats:
    # If the list for that category has less than 4 elements
    if len(attack_cat_data[attack_cat]) < 4:
      # Add the row to the list for that category
      attack_cat_data[attack_cat].append(X_test_transform[i])

# Print the data for each attack category
for cat, data in attack_cat_data.items():
  print(f"Attack Category: {cat}")
  for row in data:
    print(list(row))
  print()
```

Multiclass Clasification Using Y2

```
# Define four models
models_mc = [('LogisticRegression', LogisticRegression(multi_class='ovr', solver='lbfgs', random_state=123, max_iter=5000)),
          ('DecisionTree', DecisionTreeClassifier(random_state=123)),
          ('RandomForest', RandomForestClassifier(random_state=123)),
          ('MultiLayerPerceptron', MLPClassifier(random_state=123, solver='adam', max_iter=8000))
         ]
```

```
for model_name, clf in models_mc:
    print(clf)
```

```
LogisticRegression(max_iter=5000, multi_class='multinomial', random_state=123)
DecisionTreeClassifier(random_state=123)
RandomForestClassifier(random_state=123)
MLPClassifier(max_iter=8000, random_state=123)
```

```
# define several lists and dataframe to store the CV results and evaluation results on testing data
model_names_list = []
cv_fit_time_mean_list = []
cv_accuracy_mean_list = []
cv_precision_mean_list = []
cv_recall_mean_list = []
cv_f1_mean_list = []
cv_roc_auc_mean_list = []

test_accuracy_list = []
test_precision_list = []
test_recall_list = []
test_f1_list = []
test_roc_auc_list = []

test_roc_curve_df = pd.DataFrame()


for model_name, clf in models:

    # ==== Step 1: Cross-validation =====

    # define  Stratified 5-fold cross-validator
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=123)
    # define metrics for evaluating
    scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro', 'roc_auc_ovr_weighted']
    # perform the 5-fold CV and get the metrics results
    cv_results = cross_validate(estimator=clf,
                                X=X_train_transform,
                                y=y1_train_transform,
                                scoring=scoring,
                                cv=cv,
                                return_train_score=False)  # prevent to show the train scores on cv splits.

    # calculate the mean values of those scores
    cv_fit_time_mean = cv_results['fit_time'].mean()
    cv_accuracy_mean = cv_results['test_accuracy'].mean()
    cv_precision_mean = cv_results['test_precision_macro'].mean()
    cv_recall_mean = cv_results['test_recall_macro'].mean()
    cv_f1_mean = cv_results['test_f1_macro'].mean()
    cv_roc_auc_mean = cv_results['test_roc_auc_ovr_weighted'].mean()

    # store CV results into those lists
    model_names_list.append(model_name)
    cv_fit_time_mean_list.append(cv_fit_time_mean)
    cv_accuracy_mean_list.append(cv_accuracy_mean)
    cv_precision_mean_list.append(cv_precision_mean)
    cv_recall_mean_list.append(cv_recall_mean)
    cv_f1_mean_list.append(cv_f1_mean)
    cv_roc_auc_mean_list.append(cv_roc_auc_mean)

    # ==== Step 2: Evaluation on Testing data =====

    # fit model
    clf.fit(X=X_train_transform, y=y1_train_transform)

    # predition on testing data

    # predicted label or class
    y_pred_class = clf.predict(X=X_test_transform)

    # predicted probability of the label 1
    y_pred_score = clf.predict_proba(X=X_test_transform)

    #save as pickle file
    filename = f"{model_name}_mc.pkl"
    with open(filename, "wb") as f:
      pickle.dump(clf, f)

    # accuracy
    accuracy_ontest = accuracy_score(y_true=y1_test_transform, y_pred=y_pred_class)

    # auc of ROC
    auc_ontest = roc_auc_score(y_true=y1_test_transform, y_score=y_pred_score, multi_class='ovr')

    # precision score
    precision_ontest = precision_score(y_true=y1_test_transform, y_pred=y_pred_class, average='macro')

    # recall score
    recall_ontest = recall_score(y_true=y1_test_transform, y_pred=y_pred_class, average='macro')
```

```
    # F1 score
    f1_ontest = f1_score(y_true=y1_test_transform, y_pred=y_pred_class, average='macro')

    # # roc curve dataframe
    # fpr, tpr, threshold_roc = roc_curve(y_true=y1_test_transform, y_score=y_pred_score)

    # roc_df = pd.DataFrame(list(zip(fpr, tpr, threshold_roc)),
    #                       columns=['False Positive Rate', 'True Positive Rate', 'Threshold'])

    # roc_df['Model'] = '{} (AUC = {:.3f})'.format(model_name, auc_ontest)


    # store the above values
    test_accuracy_list.append(accuracy_ontest)
    test_roc_auc_list.append(auc_ontest)
    test_precision_list.append(precision_ontest)
    test_recall_list.append(recall_ontest)
    test_f1_list.append(f1_ontest)

    # test_roc_curve_df = pd.concat([test_roc_curve_df, roc_df],
    #                               ignore_index=True)


results_dict = {'Model Name': model_names_list,
                'CV Fit Time': cv_fit_time_mean_list,
                'CV Accuracy mean': cv_accuracy_mean_list,
                'CV Precision mean': cv_precision_mean_list,
                'CV Recall mean': cv_recall_mean_list,
                'CV F1 mean': cv_f1_mean_list,
                'CV AUC mean': cv_roc_auc_mean_list,
                'Test Accuracy': test_accuracy_list,
                'Test Precision': test_precision_list,
                'Test Recall': test_recall_list,
                'Test F1': test_f1_list,
                'Test AUC': test_roc_auc_list
                }

results_df = pd.DataFrame(results_dict)

# sort the results according to F1 score on testing data
results_df.sort_values(by='Test F1', ascending=False)
```

	Model Name	CV Fit Time	CV Accuracy mean	CV Precision mean	CV Recall mean	CV F1 mean	CV AUC mean	Ac
1	DecisionTree	2.735960	0.802047	0.595574	0.556170	0.565568	0.912380	0
2	RandomForest	29.585187	0.825365	0.683970	0.538028	0.565101	0.967137	0
3	MultiLayerPerceptron	184.823595	0.815236	0.691499	0.493369	0.507000	0.976701	0

Using the saved Pickle File (Models) Validating the Prediction of the model's Accuracy - Y2

- Y1 Model
- Top

```
# Load the saved RandomForest model
with open("/content/drive/MyDrive/main_project/saved_pickle_models/RandomForest_mc.pkl", "rb") as f:
    loaded_model = pickle.load(f)

# Take a random row from the test set
import random
random_row_index = random.randint(0, len(X_train_transform) - 1)
random_row = X_train_transform[random_row_index]
#random_row= [9.49069773792972, -1.5020363456963437, -0.6923040283191014, 2.9964724684825184, 0.3130548581711686, -0.16719265743958084,

# Predict the class and probability for the random row
predicted_class = loaded_model.predict([random_row])
predicted_probabilities = loaded_model.predict_proba([random_row])[0]

# Print the results
print(f"Predicted class: {predicted_class[0]}")
for i, prob in enumerate(predicted_probabilities):
  if prob >0:
    print(f"Probability of being class {i}: {prob:.2f}")
print(f"Input row: {random_row}")
```

```
Predicted class: 4
Probability of being class 2: 0.01
Probability of being class 3: 0.03
Probability of being class 4: 0.81
Probability of being class 6: 0.12
Probability of being class 7: 0.03
Input row: [-0.11698842  0.13476511  1.53564372 -0.3843879  -0.07440806 -0.1128602
 -0.04758834 -0.097594   -0.56835216  0.72305349  1.48327957 -0.37490679
 -0.27122846 -0.04505519 -0.10815118 -0.12340462 -0.0039824  -0.04931663
 -0.11809069  1.04622027  1.93138788  0.78427399  1.06684561  1.32823115
  1.1258416   1.36920277 -0.41969619 -0.30251606  1.18832318 -0.04023785
 -0.6808824  -0.32617494 -0.616987   -0.51832708 -0.51896889 -0.56644837
 -0.11098207 -0.11097388  1.26824158 -0.69005268 -0.65373502 -0.12046336]
```

The Below mentioned Work is for the future work which are :

- Create a web application which take input as this mentioned ranged values as Excel and gives the output as Attack or Normal
- To evaluate whether the model is Overfitted or Not
- For enhancing the Precision and Recall score of the Y2 model we can do Hyperparameter Tuning :

```
import pandas as pd

# Read the CSV file
df = con_df.copy()

# Initialize an empty dictionary
column_min_max = {}

# Loop through each column in the dataframe
for col in df.columns:
    # Get the minimum and maximum of the column
    min_val = df[col].min()
    max_val = df[col].max()

    # Add the column name and its minimum and maximum to the dictionary
    column_min_max[col] = [min_val, max_val]

# Print the dictionary

for key, value in column_min_max.items():
    print(f"{key}: {value}")
```

```
dur: [0.0, 59.999989]
proto: [0, 132]
service: [0, 12]
state: [0, 10]
spkts: [1, 10646]
dpkts: [0, 11018]
sbytes: [24, 14355774]
dbytes: [0, 14657531]
rate: [0.0, 1000000.003]
sttl: [0, 255]
dttl: [0, 254]
sload: [0.0, 5988000256.0]
dload: [0.0, 22422730.0]
```

```
    sloss: [0, 5319]
    dloss: [0, 5507]
    sinpkt: [0.0, 84371.496]
    dinpkt: [0.0, 57739.24]
    sjit: [0.0, 1483830.917]
    djit: [0.0, 463199.2401]
    swin: [0, 255]
    stcpb: [0, 4294958913]
    dtcpb: [0, 4294881924]
    dwin: [0, 255]
    tcprtt: [0.0, 3.821465]
    synack: [0.0, 3.226788]
    ackdat: [0.0, 2.928778]
    smean: [24, 1504]
    dmean: [0, 1500]
    trans_depth: [0, 172]
    response_body_len: [0, 6558056]
    ct_srv_src: [1, 63]
    ct_state_ttl: [0, 6]
    ct_dst_ltm: [1, 59]
    ct_src_dport_ltm: [1, 59]
    ct_dst_sport_ltm: [1, 46]
    ct_dst_src_ltm: [1, 65]
    is_ftp_login: [0, 4]
    ct_ftp_cmd: [0, 4]
    ct_flw_http_mthd: [0, 30]
    ct_src_ltm: [1, 60]
    ct_srv_dst: [1, 62]
    is_sm_ips_ports: [0, 1]
    attack_cat: [0, 9]
    label: [0, 1]
```

Start coding or generate with AI.

code for eval Overfit

```
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np

def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")

    return plt

# for model_name, clf in models:
#     print(clf)
#     clf = GridSearchCV(clf, grid_params[model_name], cv=5)
#     print(f"Performing hyperparameter tuning on {model_name}")
```

```
#     plot_learning_curve(clf, model_name, X_train_transform, y2_train_transform, cv=5)
#     plt.show()

for idx, (model_name, clf) in enumerate(models):
    print(clf)
    clf = GridSearchCV(clf, grid_params[idx], cv=5)
    print(f"Performing hyperparameter tuning on {model_name}")
    plot_learning_curve(clf, model_name, X_train_transform, y2_train_transform, cv=5)
    plt.show()

    # Rest of your code...
```

Hyperparameter tunning Using Grid Search CV

```
from sklearn.model_selection import GridSearchCV

# Define four models
models = [('LogisticRegression', LogisticRegression(random_state=123, max_iter=5000)),
          ('DecisionTree', DecisionTreeClassifier(random_state=123)),
          ('RandomForest', RandomForestClassifier(random_state=123)),
          ('MultiLayerPerceptron', MLPClassifier(random_state=123, solver='adam', max_iter=8000))
         ]

# Define the hyperparameters to be tuned
logistic_params = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
}

decision_tree_params = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10]
}

random_forest_params = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10]
}

mlp_params = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}

grid_params = [('LogisticRegression', logistic_params),
               ('DecisionTree', decision_tree_params),
               ('RandomForest', random_forest_params),
               ('MultiLayerPerceptron', mlp_params)]

# define several lists and dataframe to store the CV results and evaluation results on testing data
model_names_list = []
cv_fit_time_mean_list = []
cv_accuracy_mean_list = []
cv_precision_mean_list = []
cv_recall_mean_list = []
cv_f1_mean_list = []
cv_roc_auc_mean_list = []
```